Bicriteria Distributed Submodular Maximization in a Few Rounds

Alessandro Epasto
Google, New York, NY, USA
aepasto@google.com

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

Morteza Zadimoghaddam
Google, New York, NY, USA
zadim@google.com

ABSTRACT

We study the problem of efficiently optimizing submodular functions under cardinality constraints in distributed setting. Recently, several distributed algorithms for this problem have been introduced which either achieve a sub-optimal solution or they run in super-constant number of rounds of computation. Unlike previous work, we aim to design distributed algorithms in multiple rounds with almost optimal approximation guarantees at the cost of outputting a larger number of elements. Toward this goal, we present a distributed algorithm that, for any $\epsilon > 0$ and any constant r, outputs a set S of $O(rk/\epsilon^{\frac{1}{r}})$ items in r rounds, and achieves a $(1-\epsilon)$-approximation of the value of the optimum set with k items. This is the first distributed algorithm that achieves an approximation factor of $(1-\epsilon)$ running in less than $\log \frac{1}{\epsilon}$ number of rounds. We also prove a hardness result showing that the output of any $1-\epsilon$ approximation distributed algorithm limited to one distributed round should have at least $\Omega(k/\epsilon)$ items. In light of this hardness result, our distributed algorithm in one round, $r = 1$, is asymptotically tight in terms of the output size. We support the theoretical guarantees with an extensive empirical study of our algorithm showing that achieving almost optimum solutions is indeed possible in a few rounds for large-scale real datasets.

1 INTRODUCTION

As a prominent problem in machine learning and data mining applications, submodular maximization have attracted a great amount of research in the past decade. A set function $f : 2^{\mathbb{N}} \rightarrow R$ on a ground set $\mathbb{N}$ is *submodular* if for any two sets A and B, $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$, or equivalently, it satisfies the following diminishing return property, for any two sets $A \subseteq B$ and an element x, $f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$. Several machine learning and data mining applications can be formalized as a submodular maximization problem. In the majority of such applications, the goal is to select a subset of representatives in a universe of elements and optimize some objective function. Some of those machine learning applications include exemplar based clustering [13], coverage problems [2], document summarization [20], and active set selection for non-parametric learning [15], and feature selection for training complex models [20]. More recently, motivated by several large-scale applications, various techniques have been developed for solving this problem in a distributed manner [5, 6, 21, 23]. These distributed algorithms, however, either achieve a sub-optimal solution for the submodular maximization problem, or they run in super-constant[1] number of rounds of computation (which make them less appealing in distributed frameworks like MapReduce). In this paper, we aim to present distributed algorithms addressing the above issues achieving asymptotically optimal approximation guarantees in a constant number of rounds of computation by allowing to output more items.

More specifically, we focus on submodular maximization problem subject to a cardinality constraint: given a cardinality constraint k, and a submodular function f defined on subsets of $\mathbb{N}$ ($|\mathbb{N}| = n$), the goal is to find a set S of at most k items with maximum value $f(S)$. Let OPT be the set of k items that achieves the maximum value, i.e. OPT $= \arg\max_{S \subseteq \mathbb{N} \& |S| \leq k} f(S)$. In the rest of the paper, we focus on non-negative monotone submodular functions since for non-monotone submodular functions it is impossible to get better than 1/2-approximation factor using sub-exponential number of evaluations of function f even without any cardinality constraint [12]. For monotone submodular functions, it is computationally hard to approximate this problem within a factor better than $1-1/e \approx 63\%$ [11]. The best approximation guarantees for maximizing a general monotone submodular function in a scalable manner in two rounds is a 54%-approximation proposed by [21]; they also show that even if machines have unbounded computational power achieving an approximation factor better than $1-1/e$ is impossible in distributed settings (an information-theoretic hardness result). Other approximation algorithms developed for this problem achieve $1-1/e$-approximation, but they run in super-constant number of rounds of computation, e.g. logarithmic or $1/\epsilon$ rounds to achieve $1-1/e-\epsilon$. In practice, however, it is desirable to achieve an almost optimum solution, i.e., $(1-\epsilon)$-approximation. While such an approximation factor is not achievable even on a single machine when limiting the output size to k elements, a naive approach of getting $(1-\epsilon)$-approximation in a centralized way is to repeatedly run the greedy algorithm on the data, and output $O(k \ln(1/\epsilon))$ elements. In order to achieve such an approximation factor in a distributed manner, a naive idea is to run distributed constant-factor approximation algorithms for submodular maximization [5, 21, 23], and get to $(1-\epsilon)$-approximation in $O(\ln(1/\epsilon))$ rounds. In the distributed setting, even if some $(1-\epsilon)$-approximation algorithms are used by the distributed and central machines to select items, the overall distributed approximation factor will not be more than 1/2 based on Theorem 5 of [5]. Therefore still $\log(1/\epsilon)$ rounds are needed to achieve a $(1-\epsilon)$ approximation factor. We show these naive greedy approaches in Table 1. The main issue with these

[1] The number of rounds of other methods is some function of $1/\epsilon$ which increases as we aim for better approximation guarantees, i.e. smaller ϵ. For a constant ϵ, the other methods also give constant (but possibly large) number of rounds.

SPAA '17, July 24-26, 2017, Washington DC, USA

ACM ISBN 978-1-4503-4593-4/17/07.
https://doi.org/10.1145/3087556.3087574

Algorithm	Rounds	Size of output $\lvert S\rvert$	Approximation
GreedyScaling [18]	$O(\log(\Delta)/\epsilon)$	k	$1-1/e-\epsilon$
GreeDi [23]	1	k	$1/\min\{m,k\} \geq 1/n^{1/3}$
PseudoGreedy [21]	1	k	0.54
RandGreedi [5]	1	k	0.316
ParallelAlg [6]	$O(1/\epsilon)$	k	$1-1/e-\epsilon$
Naive Distributed Greedy	$O(\log(1/\epsilon))$	$k\log(1/\epsilon)$	$1-\epsilon$
BicriteriaGreedy*,	r	$O(rk\ln^2(1/\epsilon^{1/r})/\epsilon^{2/r})$	$1-\epsilon$
BicriteriaGreedy with multiplicity*	r	$O(rk\ln(1/\epsilon^{1/r})/\epsilon^{1/r})$	$1-\epsilon$
HybridAlg*	r	$O(rk/\epsilon^{1/r})$	$1-\epsilon$

Table 1: Summary of our results and the state of the art. The number of rounds shows the number of times the central algorithm interacts with the distributed machines. The results with * are the new results of this paper. The main advantage of our methods are achieving almost optimal solutions for any number of rounds $r > 0$.

approaches is the number of rounds which is a major bottleneck in making these algorithms scalable. This leaves open the problem of defining greedy-based algorithms in distributed settings with limited number of rounds. We propose the distributed Algorithm BicriteriaGreedy with pseudo-code as Algorithm 1 that outputs a set S of more than k items, and achieves the $1-\epsilon$ times the value of the optimum set for a given $\epsilon > 0$. The number of selected items, $|S|$, depends on k and ϵ as expected, and the dependence on ϵ can be reduced exponentially by increasing the number of rounds. So one can choose a small number of rounds (two or three), and still have a good guarantee on the number of selected items without sacrificing the scalability of the algorithm. We also prove that the polynomial dependence of the output size on $1/\epsilon$ is necessary by providing a hardness result that also shows our results are asymptotically tight when the algorithm has to perform in one distributed round[2]. In addition to the mathematical analysis of our algorithm, we support the theoretical guarantees with an extensive empirical study of our algorithm, and show that achieving almost optimum solutions are indeed possible in a few rounds. We highlight that our main contribution is not to introduce a new algorithmic technique for submodular maximization (we build on the well-known greedy algorithm). We explore theoretically and experimentally the trade-offs between outputting more items and number of rounds when we aim for a $1-\epsilon$ approximation guarantee.

Our Contributions. We provide the first distributed algorithm that achieves an approximation factor of $(1-\epsilon)$ running in less than $\log\frac{1}{\epsilon}$ number of rounds. We use the same distributed framework of [21, 23]. A central machine partitions the ground set randomly among a set of distributed machines (workers). Each machine runs the greedy algorithm to select a subset of its items, and return them to the central machine. Among all returned items, a final output set will be selected. We generalize the proof techniques of [21] and present algorithm BicriteriaGreedy that outputs a solution with value arbitrarily close to optimum. In particular for any $\alpha > 1$, we show that an approximation factor of $1-1/\alpha$ is achieved if each distributed machine greedily returns $O(\alpha k)$ items to the central machine, and then $\tilde{O}(\alpha^2 k)$ of these returned items are greedily selected as the final solution. We note that [21] provides a 0.54-approximation factor by outputting k items, and here we analyse the effect of growing the output size beyond k on the approximation factor and how fast we converge to an almost optimum solution. Using the simple trick of sending each item to $\alpha\ln(\alpha)$ random machines instead of a random partitioning of ground set, we can show that outputting $\tilde{O}(\alpha k)$ items as the final solution suffices to achieve an approximation factor of $1-1/\alpha$. We call this $\alpha\ln(\alpha)$ term the multiplicity factor as it is the number of machines we send each item to. By selecting a larger α, one can achieve better approximation guarantees while outputting more items. We further improve our results by presenting algorithm HybridAlg that outputs $O(\alpha k)$ items while having the $1-1/\alpha$ approximation (this gets rid of the extra log factor in the size upper bound).

So far, we have described our methods using one distributed round of computation. A major contribution of our work is to get the same approximation factors with outputting much fewer items when more number of rounds is allowed. We start with $S=\emptyset$, and we want to reduce the gap $f(\text{OPT})-f(S)$ to at most $\epsilon f(\text{OPT})$ (equivalent of getting a $1-\epsilon$ approximation). At the beginning this gap is equal to $f(\text{OPT})$, and therefore in r rounds we want to reduce it by a total multiplicative factor of ϵ. This can be achieved by reducing the gap by a factor of $\epsilon^{1/r}$ in each round. Our proof techniques are of independent interest as they resemble some of the ideas of the *egg dropping puzzle* [14]. By setting $\alpha = O(1/\epsilon^{1/r})$, we can get a $1-\epsilon^{1/r}$ approximation in each round which is equivalent of reducing the gap by $\epsilon^{1/r}$. Therefore after r rounds, we have a $(1-\epsilon)$ approximation factor using $O(r\alpha k)$ items in HybridAlg, $\tilde{O}(r\alpha k)$ items in BicriteriaGreedy with multiplicity $\alpha\ln(\alpha)$, and $\tilde{O}(r\alpha^2 k)$ items in BicriteriaGreedy with just a random partitioning (multiplicity one).

Furthermore, we show that for a distributed algorithm that achieves a $1-\epsilon$ approximation guarantee in one distributed round, the algorithm needs to have an output size of at least $\Omega(k/\epsilon)$. This hardness result proves the tightness of our algorithmic result for $r=1$, and also provides an insight on why the output size should have a polynomial dependence on $1/\epsilon$ in distributed setting versus the logarithmic dependence in centralized single machine setting.

[2]Having only one distributed round is an important case specially in time-sensitive applications in which the algorithm should process the data very fast and provide a solution instantly.

1.1 Related Work.

Submodular maximization in a distributed manner have attracted a significant amount of research over the last few years [4–6, 8–10, 18, 19, 21, 23]. From a theoretical point of view, for the coverage maximization problem, [9] present a $(1-1/e)$-approximation algorithm in polylogarithmic number of MapReduce rounds, and [8] improved this result and achieved $\log^2 n$ number of rounds. Recently, [18] present a $(1-1/e)$-approximation algorithm using a logarithmic number of rounds of MapReduce. They also derive $(1/2-\epsilon)$-approximation algorithm that runs in $O(\frac{1}{\delta})$ number of rounds of MapReduce (for a constant δ), but this algorithm needs a $\log n$ blowup in the communication complexity, and number of rounds could become large for small δ. [6] present a distributed $1-1/e-\epsilon$ approximation algorithm that runs in $O(1/\epsilon)$ rounds, and their space requirement also grows linearly with $1/\epsilon$. As observed in various empirical studies [17], the communication complexity and the number of MapReduce rounds are important factors in determining the performance of a MapReduce-based algorithm and a $\log n$ blowup in the communication complexity can play a crucial role in applicability of the algorithm in practice. Our algorithm on the other hand runs only in a constant number of rounds. Recently, distributed approximation algorithms have been developed for this problem that run in two rounds [5, 21, 23], however, they do not achieve optimal approximation factor of $1-1/e$ for this problem, or they do not achieve a general result for all submodular functions. [23] shows the effectiveness of applying algorithm Greedy over a random partitioning empirically for several machine learning applications. The authors also prove theoretical guarantees for algorithm Greedy for special classes of submodular functions satisfying a certain Lipschitz condition [23].

2 ALGORITHM BICRITERIAGREEDY

We present Algorithm BicriteriaGreedy depicted as Algorithm 1 which uses the Greedy algorithm described as Algorithm 2 as a subroutine and achieves approximation guarantee of $1-\epsilon$. Algorithm BicriteriaGreedy receives in the input the ground set $\mathbb{N}$, as the number of rounds r, approximation error ϵ, the number of machines m, and cardinality constraint k. The algorithm works for any $r > 0$, and the upper bound on the number of selected items improves as r grows. However one should be careful with setting r to a large number as the number of rounds directly influences the scalability of the algorithm. The algorithm also works for any number of machines $m > 0$ but for the sake of analysis we need $m \geq \alpha \ln(\alpha)$ for $\alpha = 3/\epsilon^{1/r}$. Increasing m reduces the workload on each of the distributed m machines, however it increases the number of items the central machine should process[3].

BicriteriaGreedy constructs output set S by starting with $S = \emptyset$, and adding items to S as follows. In each of the r rounds (lines 6–11), it partitions the items randomly between m machines, giving set T_i to machine i. This random partitioning in each round might seem an overhead, but since the running times of each machine is super-linear in the number of items it receives, repartitioning does not become a bottleneck both in terms of the asymptotic complexity, and also the running times of our experiments in Section 4. Every machine i runs Algorithm 2 (Greedy) to select a subset $S_i \subset T_i$. In both Algorithms 1 and 2, we use notation $\Delta(x, A)$ to denote the marginal value of adding item x to A, i.e. $\Delta(x, A) = f(\{x\} \cup A) - f(A)$. The selected items ($\cup_{i=1}^{m} S_i$) are sent to a central machine which does another filtering and selects a subset $A \subseteq \cup_{i=1}^{m} S_i$, and adds A to set S. This extra filtering corresponds to lines $9-11$ in Algorithm 1. Parameter ϵ determines how much suboptimal we are allowed to be compared to the optimum solution.

The for loop in lines $7-8$ of Algorithm 1 is parallelized, and machine i runs the subroutine $Greedy(\alpha k, S, T_i)$ described as Algorithm 2 for each $1 \leq i \leq m$. In lines $9-11$ of Algorithm 1, the central machine gathers all selected items $\cup_{i=1}^{m} S_i$, and in each of the $(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$ iterations, chooses the item with maximum marginal gain among the selected items to add to S. To summarize, Algorithm 1 selects $(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$ items in each round.

1 **Input:** $\mathbb{N}$, r, ϵ, m, and k.
2 **Output:** Set $S \subset \mathbb{N}$ with $f(S) \geq (1-\epsilon) f(\text{OPT})$.
3 $\alpha \leftarrow 3/(\epsilon^{1/r})$;
4 $S \leftarrow \emptyset$;
5 **forall the** $1 \leq \ell \leq r$ **do**
6 Send each item in $\mathbb{N}$ independently into one of $\{T_i\}_{i=1}^{m}$ uniformly at random;
7 **forall the** $1 \leq i \leq m$ **do**
8 $S_i \leftarrow Greedy(\alpha k, S, T_i)$;
9 **forall the** $1 \leq j \leq (\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$ **do**
10 $x^* \leftarrow max_{x \in \cup_{i=1}^{m} S_i} \Delta(x, S)$;
11 $S \leftarrow S \cup \{x^*\}$;
12 Return S;

Algorithm 1: Algorithm BicriteriaGreedy

1 **Input:** k', S, and T_i.
2 **Output:** Set $S_i \subset T_i$ with $|S_i| \leq k'$.
3 $S_i \leftarrow \emptyset$;
4 **forall the** $1 \leq i \leq k'$ **do**
5 $x^* \leftarrow max_{x \in T_i} \Delta(x, S_i \cup S)$;
6 $S_i \leftarrow S_i \cup \{x^*\}$;
7 Return S_i;

Algorithm 2: Algorithm Greedy

2.1 Analysis

We start by proving that Algorithm BicriteriaGreedy returns a solution with $\mathbb{E}[f(S)] \geq (1-\epsilon) f(\text{OPT})$ and $|S| \leq r(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$. In Subsection 2.2, we show that it is possible to reduce the number of selected items ($|S|$) by some slight changes while maintaining the $1-\epsilon$ approximation guarantee. In Algorithm 1, we iteratively run the for loop in lines $6-11$ for r consecutive times. We call each of these r executions a round. The high level proof plan is to show that at each round $f(S)$ is increased by at least $(1-\epsilon^{1/r})(f(\text{OPT}) - f(S))$. In other words, the gap $f(\text{OPT}) - f(S)$

[3] Setting $m = \sqrt{n/k'}$ makes sure that the number of items each distributed machine processes, and the central machine processes are the same where k' is the number of items each distributed machine returns in line 8 of the algorithm.

is reduced by a multiplicative factor of $\epsilon^{1/r}$ in each round, and therefore after r rounds this gap is at most $\epsilon f(\text{OPT})$ which implies that $f(S) \geq (1-\epsilon)f(\text{OPT})$. We will formalize and elaborate this argument in the proofs. To avoid confusion, we define A_ℓ to be the set of selected items (set S) in the first ℓ rounds for any $0 \leq \ell \leq r$. At the beginning set S is equal to $A_0 = \emptyset$, and the final output is A_r. We start by showing that at each round, we get at least $1-\epsilon^{1/r}$ closer to OPT. We provide most of the proofs in the supplemental material, and include a high level intuition.

Lemma 2.1. *The expected value of $\mathbb{E}[f(A_\ell)] - f(A_{\ell-1})$ (increase in value of S at round ℓ) is at least $(1-\epsilon^{1/r})(f(OPT) - f(A_{\ell-1}))$ for any $1 \leq \ell \leq r$, and set $A_{\ell-1}$.*

We first prove that Lemma 2.1 is sufficient to achieve the $1-\epsilon$ approximation guarantee.

Theorem 2.2. *Algorithm BicriteriaGreedy returns a set S with expected value at least $(1-\epsilon)f(OPT)$ and at most $r(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$ items where α is $\frac{3}{\epsilon^{1/r}}$.*

Proof. Algorithm 1 has r rounds and in each round, $(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$ items are added to S. Therefore size of S can not be more than $r(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$. To lower bound the value of final set $S = A_r$, we define a_ℓ to be $\mathbb{E}[A_\ell]$. Using Lemma 2.1, we know that $a_\ell - a_{\ell-1}$ is at least $(1-\epsilon^{1/r})(f(\text{OPT}) - a_{\ell-1})$. In other words, $f(\text{OPT}) - a_\ell$ is at most $\epsilon^{1/r}(f(\text{OPT}) - a_{\ell-1})$. By combining all these lower bounds for different values of $1 \leq \ell \leq r$, we have $f(\text{OPT}) - a_r \leq (\epsilon^{1/r})^r (f(\text{OPT}) - f(A_0)) = \epsilon f(\text{OPT})$, and therefore $\mathbb{E}[f(S)] = \mathbb{E}[f(A_r)] = a_r \geq (1-\epsilon)f(\text{OPT})$. □

To complete the analysis, we provide the main ideas of the proof of Lemma 2.1, and include the formal proof after that. We need to borrow some notations from [21], and use some of the techniques developed there for our analysis. Let OPT^S be the part of OPT that is selected by the machines, i.e. $\text{OPT}^S = \text{OPT} \cap (\cup_{i=1}^m S_i)$. Let OPT^{NS} be $\text{OPT} \setminus \text{OPT}^S$.

We will also borrow the following notations which are going to be used in the proof of Lemma 2.1. Therefore we can refer to set $A_{\ell-1}$ which is mentioned in the statement of Lemma 2.1 without any confusion. For every machine i, we define a partition of optimum set OPT into two sets OPT_i^S and OPT_i^{NS}. Set OPT_i^S consists of optimum items that if they were sent to machine i, they would be selected. Formally, OPT_i^S is defined as $\{x | x \in \text{OPT AND } x \in \mathit{Greedy}(k', A_{\ell-1}, T_i \cup \{x\})\}$ where $\mathit{Greedy}(k', S, T)$ is the output of Algorithm Greedy (depicted as Algorithm 2) with inputs k', S, and T. In other words, if we send an item $x \in \text{OPT}$ along with set T_i to machine i in round ℓ and machine i selects item x as part of its output, we will put x in set OPT_i^S. Any other optimum item is put in set $\text{OPT}_i^{NS} = \text{OPT} \setminus \text{OPT}_i^S$. Formally, OPT_i^{NS} is defined as $\{x | x \in \text{OPT AND } x \notin \mathit{Greedy}(k', A_{\ell-1}, T_i \cup \{x\})\}$. We also fix an arbitrary permutation of items in OPT, and for every $x \in \text{OPT}$, we define OPT^x to be the items in OPT that appear before x in the fixed permutation.

We focus on the claim of Lemma 2.1 for the first round and a similar argument works for the rest. We show there exists a small set B^* of selected items $\{S_i\}_{i=1}^m$ with value almost as large as $f(\text{OPT})$. Define $B^* = \text{OPT}^S \cup S_1 \cup S_2 \cdots S_C$ where $C = \alpha \ln(\alpha)$. By submodularity of f, we have $f(\text{OPT}) - f(B^*) \leq \sum_{x \in \text{OPT}} \Delta(x, B^*)$, and we show that the marginal values $\Delta(x, B^*)$ are all small in expectation. Optimum items that are selected, OPT^S, are already in B^* and therefore have zero marginal value to B^*. If some item x would not be selected by some machine $1 \leq i \leq C$ if x were sent to i, we can say that machine i would have preferred other αk items in S_i, and therefore $\Delta(x, S_i)$ is less than the average marginal values of selected items $f(S_i)/(\alpha k)$ which suffices to show $\Delta(x, B^*)$ is small. For any other optimum item x, we know all these C machines would have picked x, if it was sent to any of them, but apparently x was sent to some other machine that did not pick it. Similar to Lemma 3.2 of [21], we can prove that this happens with only a small probability of $1/\alpha$ with our choice of C. We conclude that expected marginal value $\Delta(x, B^*)$ is small for every $x \in \text{OPT}$, and therefore $f(B^*)$ should be almost as large as $f(\text{OPT})$. To see the rest of the proof, we note that all items of B^* are available to be chosen by the central machine in lines 9 − 11 of Algorithm 1. Using the classic analysis of algorithm Greedy, choosing $|B^*| \ln(1/\epsilon)$ items at this step suffices to have $f(S) \geq (1-\epsilon)f(B^*)$ which completes the proof. We are ready to formalize all these main ideas as the proof of Lemma 2.1.

Proof of Lemma 2.1 We first define set function $g(B)$ to be $f(B \cup A_{\ell-1}) - f(A_{\ell-1})$ for any subset of items B. Submodularity of f implies that g is also submodular. It suffices to show that $\mathbb{E}[g(A_\ell)]$ is at least $(1-\epsilon^{1/r})\mathbb{E}[g(\text{OPT})]$. We also note that Greedy returns the same solution when it maximizes g instead of f. In other words, proving the claim for every round ℓ is equivalent of proving it for the first round in which f and g are the same. First of all we show that among the selected items of all machines $\cup_{i=1}^m S_i$, there exists a set B^* with expected g value at least $(1-\frac{2}{\alpha})g(\text{OPT})$, and size at most $(1+\alpha^2 \ln(\alpha))k$. Then we can show that running greedy on the set of all selected items, and choosing $|B^*| \ln(\alpha)$ items yields a final solution S with $g(S) \geq (1-\frac{1}{\alpha})g(B^*)$ which completes the proof.

We claim that for set $B^* = \text{OPT}^S \cup (\cup_{i=1}^{\alpha \ln(\alpha)} S_i)$, we have that $\mathbb{E}[g(B^*)] \geq (1-\frac{2}{\alpha})g(\text{OPT})$. It is important to note that we are lower bounding the expected value of $g(B^*)$. By definition, we have $\sum_{x \in \text{OPT}} \Delta(x, B^* \cup \text{OPT}^x)$ is equal to $g(B^* \cup \text{OPT}) - g(B^*)$ which is at least $g(\text{OPT}) - g(B^*)$. At this point, we only need to show that the expected value of $\sum_{x \in \text{OPT}} \Delta(x, B^* \cup \text{OPT}^x)$ is at most $\frac{2}{\alpha} g(\text{OPT})$. Since the expected value of a sum is equal to the sum of the expected value of summands, we can focus on upper bounding expected value of $\Delta(x, B^* \cup \text{OPT}^x)$ for each $x \in \text{OPT}$. Each $x \in \text{OPT}$ belongs to one of the following three categories:

- $x \in \text{OPT}^S$: In this case, x is also in B^*, and therefore $\Delta(x, B^* \cup \text{OPT}^x)$ is zero.
- $x \in \text{OPT}_i^{NS}$ for some $1 \leq i \leq \alpha \ln(\alpha)$: Item x was not chosen as one of the αk items in S_i. Therefore the marginal value of each item added to S_i was higher than the marginal value of adding x at that moment. Let δ_j be how much the value S_i increased when its j^{th} item was added to it. So we have $g(S_i) = \sum_{j=1}^{\alpha k} \delta_j$. Since f is submodular, the marginal values to set S_i decrease as we add more items to S_i. So $\Delta(x, S_i)$ (at the end when S_i has all its αk items) is less than δ_j for each $1 \leq j \leq \alpha k$. We conclude

that $\Delta(x,S_i) \leq g(S_i)/(\alpha k)$. If $g(S_i)$ is at least $g(\text{OPT})$, the claim is proved because we know $S_i \subset B^*$, and therefore $g(B^*) \geq g(S_i) \geq g(\text{OPT})$ in this case. Otherwise, $\Delta(x,S_i)$ is less than $\frac{g(\text{OPT})}{\alpha k}$. Therefore for every $x \in \cup_{i=1}^{\alpha \ln(\alpha)} \text{OPT}_i^{NS}$, we have $\Delta(x, B^* \cup \text{OPT}^x) \leq \Delta(x,S_i) < \frac{g(\text{OPT})}{\alpha k}$ where the first inequality holds by definition of submodularity, and i is chosen such that $x \in \text{OPT}_i^{NS}$.

- The last case is when x is not in any of the sets OPT^S, and $\{\text{OPT}_i^{NS}\}_{i=1}^{\alpha \ln(\alpha)}$ in which we upper bound $\mathbb{E}[\Delta(x, B^* \cup \text{OPT}^x)]$ as follows.

We show that for any item $x \in \text{OPT}$, the probability of x being outside all these sets is at most $1/\alpha$. We also know by submodularity that $\Delta(x, B^* \cup \text{OPT}^x) \leq \Delta(x, \text{OPT}^x)$. Therefore the expected value of $\Delta(x, B^* \cup \text{OPT}^x)$ cannot be more than $\Delta(x, \text{OPT}^x)/\alpha$ in this case.

Intuitively, the probability of selecting an item x ($Pr[x \in \cup_{i=1}^{m} S_i]$) is the same as $Pr[x \in \text{OPT}_i^S]$ for any arbitrary machine i. The event that item $x \in \text{OPT}$ is not selected ($\notin \text{OPT}^S$), and it also is not part of any of the $\alpha \ln(\alpha)$ sets $\{\text{OPT}_i^{NS}\}_{i=1}^{\alpha \ln(\alpha)}$ is equivalent of saying that among $\alpha \ln(\alpha)$ randomly chosen machines $(1, 2, \cdots, \alpha \ln(\alpha))$, they all select x if it is sent to them. But x is in fact sent to some other machine that did not select x. This is a very unlikely event as follows. If the probability x being selected (event $x \in \text{OPT}^S$) is higher than $1 - \frac{1}{\alpha}$, the above event has probability less than $\frac{1}{\alpha}$. Otherwise, assuming the sets $\{\text{OPT}_i^{NS}\}_{i=1}^{\alpha \ln(\alpha)}$ are independent, the probability that x is in none of them is at most $(1 - \frac{1}{\alpha})^{\alpha \ln(\alpha)} \leq \frac{1}{\alpha}$ which concludes the proof. This part of the proof and how to deal with the dependencies are formalized in Lemma 3.2 of [21].

We conclude that the total $\mathbb{E}[\sum_{x \in \text{OPT}} \Delta(x, B^* \cup \text{OPT}^x)]$ is upper bounded by $\sum_{x \in \text{OPT}} \frac{g(\text{OPT})}{\alpha k} + \frac{\Delta(x, \text{OPT}^x)}{\alpha}$ which is at most $\frac{2g(\text{OPT})}{\alpha}$ because there are at most k items in OPT, and we know that the sum $\sum_{x \in \text{OPT}} \Delta(x, \text{OPT}^x)$ is equal to $g(\text{OPT})$ by definition of Δ values and OPT^x. This means that set B^* has expected value at least $(1 - \frac{2}{\alpha})g(\text{OPT})$.

We note that items of B^* are among the selected items $\cup_{i=1}^{m} S_i$, and Algorithm1 has the option of adding them to set S in lines $10 - 11$. For $\alpha^2 \ln^2(\alpha) = |B^*| \ln(\alpha)$ times, the maximum marginal item is greedily chosen and added to S. Using the classic analysis of Greedy algorithm [24], we know that $g(S)$ should be at least $(1 - \frac{1}{|B^*|})^{|B^*| \ln(\alpha)} g(B^*) \geq (1 - \frac{1}{\alpha})g(B^*)$. We note that we showed $g(S) \geq (1 - \frac{1}{\alpha})g(B^*)$ not in expectation but in any case which is a stronger claim. This way we can combine it with the lower bound on expected value of $g(B^*)$. We conclude that $\mathbb{E}[g(S)]$ is at least $(1 - \frac{1}{\alpha})(1 - \frac{2}{\alpha})g(\text{OPT}) \geq (1 - \frac{3}{\alpha})g(\text{OPT}) = (1 - \epsilon^{1/r})g(\text{OPT})$ which completes the proof. □

2.2 Improving the Solution Size by Multiplicity

BicriteriaGreedy returns a set S with $\mathbb{E}[f(S)] \geq (1 - \epsilon)f(\text{OPT})$ and $\tilde{O}(\frac{k}{\epsilon^{2/r}})$ items in r rounds. Although one can choose the right number of rounds r to reduce the number of selected items, we propose the following simple trick that improves this upper bound to around $\tilde{O}(\frac{k}{\epsilon^{1/r}})$. In line 6, instead of just a random partitioning of items, we send each item to $C = \alpha \ln(\alpha)$ (multiplicity factor) randomly chosen machines which is similar to the multiplicity idea of [21]. With multiplicity C, we prove that we can achieve the same approximation guarantees as before while selecting much fewer items in lines $9 - 11$. In line 9, we will select only $(\alpha \ln^2(\alpha) + \ln(\alpha))k$ items instead of $(\alpha^2 \ln^2(\alpha) + \ln(\alpha))k$ in the new algorithm.

Theorem 2.3. *The new Algorithm 1 with multiplicity $C = \alpha \ln(\alpha)$ also returns a set S with expected value at least $(1 - \epsilon)f(OPT)$, and size at most $r(\alpha \ln(\alpha) + \ln(\alpha))k$ items where $\alpha = \frac{3}{\epsilon^{1/r}}$.*

Proof. The changes in the number of selected items in line 11 are reflected in the new upper bound on size of S. We should show that expected value of S is still at least $(1 - \epsilon)f(\text{OPT})$. Similar to Theorem 2.2, we use Lemma 2.1 to lower bound the marginal value gained at each round. The only part that changes in the proof of Lemma 2.1 is the definition of B^*. We no longer need to include $C = \alpha \ln(\alpha)$ sets $\{S_i\}_{i=1}^{C}$ in B^* because sending each item to C random sets incorporates that idea automatically. We define B^* to be $\text{OPT}^S \cup S_1$. The number of items we select in line 10 of the new algorithm is $(\alpha \ln^2(\alpha) + \ln(\alpha))k$ which is equal to $|B^*| \ln(\alpha)$ as expected. To lower bound the value of B^*, we still need to upper bound $\Delta(x, B^* \cup \text{OPT}^x)$. The first two cases we considered for x are proved the same way. We just need to show that for each item $x \in \text{OPT}$, the probability that x is not in OPT^S nor in OPT_1^{NS} is at most $\frac{1}{\alpha}$. Since each item is sent to C random machines, this event is equivalent of saying that a random machine (machine 1) wants to select x, but none of the the other C random machines that we actually send x to does not want to select x. We have upper bounded this probability in proof of Lemma 2.1 by $\frac{1}{\alpha}$. The rest of the proof remains the same. □

We provide Algorithm HybridAlg with slight changes to improve solution size. In addition to the multiplicity factor $C = \alpha \ln(\alpha)$, we change the second selection procedure in lines $9 - 11$ as follows. After the machines select sets $S_1, S_2, \cdots, S_m$, algorithm HybridAlg adds set S_1 to S, and then for $\ln(\alpha)k$ iterations, greedily chooses the item with maximum marginal value to S among items in $\cup_{i=2}^{m} S_i$, and adds it to S.

Theorem 2.4. *Algorithm HybridAlg returns a set S with expected value at least $(1 - \epsilon)f(OPT)$, and size at most $r(\alpha + \ln(\alpha))k$ items where $\alpha = \frac{3}{\epsilon^{1/r}}$.*

Proof. The proof is very similar to the proofs of Theorems 2.2 and 2.3, and crucially uses Lemma 2.1. In the proof of Theorem 2.3, we show that in each round $g(S_1 \cup \text{OPT}^S)$ has expected value at least $(1 - \frac{2}{\alpha})g(\text{OPT})$. By adding S_1 to S, we increase the value of S by $g(S_1)$. We need to show that the remaining items gain most of the remaining marginal value $g(S_1 \cup \text{OPT}^S) - g(S_1)$. Since there are at most k items in OPT^S, and we greedily insert $k \ln(\alpha)$ items to S. The extra marginal value we achieve is at least $(1 - \frac{1}{\alpha})(g(S_1 \cup \text{OPT}^S) - g(S_1))$. We conclude that in total the expected marginal value added in a round is at least $(1 - \frac{1}{\alpha})g(S_1 \cup \text{OPT}^S)$. The rest of the proof remains the same. □

3 HARDNESS RESULTS

A centralized greedy algorithm achieves a $1 - \epsilon$ approximation guarantee by outputting only $k \ln(1/\epsilon)$ items. Note that the dependence on $1/\epsilon$ is logarithmic when the algorithm has access to the whole dataset (all items). The number of items our distributed

algorithms return have polynomial dependence on ϵ, i.e. $1/\epsilon^{1/r}$ for r rounds. In this section, we provide some evidence that why this polynomial dependence is necessary. In particular, we prove achieving an approximation factor of $1-\epsilon$ in one round requires outputting $\Omega(k/\epsilon)$ items which matches our positive algorithmic bounds as well. Here by one round, we mean distributing the data either randomly or worst case among distributed machines, then gathering the selected items of all machines in one place and choosing a final solution among them. The result focuses on distributed algorithms that perform in one distributed round which consists of a) partitioning the items among m machines either randomly or in a worst case partitioning, b) each machine outputs a summary of the data it has received (e.g. a subset of its items) and finally c) a central machine puts together all summaries and outputs a final solution based on the union.

Theorem 3.1. *For any $k > 0$, there exists some n, m and an instance of submodular maximization with n items, m machines and cardinality constraint k such that any distributed algorithm with approximation guarantee $1-\epsilon$ in one distributed round should output at least $\Omega(k/\epsilon)$ items as the final solution.*

Proof. We construct a coverage maximization instance in which each item is a subset of a large ground set of L elements, and the submodular value of a collection of these subsets (items) is defined to be the number of elements that their union covers (among the L elements). In this instance, suppose $L \gg n$, and $n, m \gg k$. There three categories of items:

- A collection of $k/2$ equal size disjoint subsets (items) $\mathbb{A} = \{S_1, S_2, \cdots, S_{k/2}\}$ that cover $1-2\epsilon$ fraction of the universe altogether. In particular, each set S_i has size $\frac{1-2\epsilon}{k/2}L$ since they are all disjoint. Assume that L is chosen such that $\frac{1-2\epsilon}{k/2}L$ is an integer.
- A collection of $k/2$ equal size disjoint subsets (items) $\mathbb{B} = \{T_1, T_2, \cdots, T_{k/2}\}$ that cover the other 2ϵ fraction of the universe. So each set T_i has size $\frac{2\epsilon}{k/2}L$ since they are all disjoint. Assume that L is chosen such that $\frac{2\epsilon}{k/2}L$ is also an integer. So far all these $k/2 + k/2 = k$ sets in families $\mathbb{A}$ and $\mathbb{B}$ are disjoint and cover the whole universe. So the optimum solution consists of these k subsets and has value L.
- A collection $\mathbb{C}$ of $n-k$ subsets each with size $\frac{2\epsilon}{k/2}L$ which is equal to the sets in collection $\mathbb{B}$. Each set in $\mathbb{C}$ is a random subset of the ground set with $\frac{2\epsilon}{k/2}L$ elements, and these sets are chosen independent of each other. So unlike families $\mathbb{A}$ and $\mathbb{B}$, sets in family $\mathbb{C}$ are not disjoint and can potentially intersect with each other and all other sets.

All n sets (items) are distributed randomly between the m distributed machines. Since m is chosen to be much larger than k, with high probability, the k sets in $\mathbb{A}\cup\mathbb{B}$ end up in different machines. We focus on set $T_i \in \mathbb{B}$ that has been sent to some machine $1 \le \ell \le m$. The machine ℓ does not receive any other set in $\mathbb{A} \cup \mathbb{B}$, instead it may receive many sets from $\mathbb{C}$. In the absence of other members of $\mathbb{A} \cup \mathbb{B}$, it is information theoretically impossible to distinguish set T_i from the other sets sent to machine ℓ. Note that they all have the same size and they are random sets. So the probability that set T_i is chosen for the next round is proportional to the core-set size k', i.e. $\frac{k'}{n/m}$. Limitations on memory enforces this probability to be very low, i.e. for instance less than ϵ for some choices of $n \gg mk'$. Therefore at most ϵ fraction of sets in $\mathbb{B}$ are selected for the next round.

Even if all sets in $\mathbb{A}$ are chosen, given only $\frac{\epsilon k}{2}$ sets in $\mathbb{B}$ are chosen, one needs to output many sets in C to achieve an approximation factor of $1-\epsilon$. Formally, the selected sets of $\mathbb{A} \cup \mathbb{B}$ cover up to $1 - 2\epsilon + \epsilon \times 2\epsilon$ fraction of the ground set. To compensate for the remaining gap of $(1-\epsilon) - (1-2\epsilon+\epsilon\times 2\epsilon) = \epsilon - 2\epsilon^2 > \epsilon/2$ (for $\epsilon > 1/4$), the final output set needs to have at least $\frac{k}{10\epsilon}$ sets (items) from $\mathbb{C}$ which completes the proof. This is true because each set in $\mathbb{C}$ covers $\frac{2\epsilon}{k/2} \times 2\epsilon L$ elements that were supposed to be covered by sets of $\mathbb{B}$, and using concentration bounds this number does not exceed $\frac{5\epsilon^2}{k}L$. □

4 EMPIRICAL EVALUATION

In this section we empirically confirm the theoretical findings of our paper by evaluating the algorithms described before over several large-scale real-world and synthetic datasets. Recall that the focus of this paper is not to introduce a new algorithmic technique for submodular maximization (we use the well-known greedy algorithm) but instead to explore theoretically and experimentally the trade-offs between the number of items output, the number of rounds used and the objective value obtained by the greedy distributed algorithm for maximizing submodular functions. Notice that no previous work has explored the problem of outputting more items to improve the solution. For this reason in this section our comparison is done using the standard greedy distributed algorithm and evaluating different output sizes and number of rounds using multiple real and artificial datasets. All real datasets used are publicly available. In this section we experiment with two different instantiations of monotone submodular maximization: coverage maximization and exemplar-based clustering.

4.1 Coverage maximization

First we evaluate the greedy distributed algorithm on the coverage problem. In a coverage instance we are given a family $\mathbb{N} \subseteq 2^U$ of sets over a ground set U and we want to find k sets from $\mathbb{N}$ with maximum size of their union. We first present our real datasets for coverage maximization. We consider datasets: **DBLP co-authorship** has $\sim 300k$ sets over $\sim 300k$ elements for a total sum of sizes of all sets of $1.0m$ elements; **LiveJournal friendship** [27] has $4m$ sets over $4m$ elements for a total size of $34m$. For the previous datasets the sets represent neighborhoods of nodes. Finally we used **Gutenberg bi-grams** with $41k$ sets over $99m$ elements for a total size of more than $1b$. Here sets represent bi-grams in books. We will elaborate on these coverage datasets and the experimental setup as follows.

Experimental setup DBLP co-authorship We extracted a dataset from a DBLP snapshot [27] by creating a set for each author representing the coauthors of that author. The ground set is the set of all authors in DBLP. There are ~ 300 thousands sets over ~ 300 thousands elements for a total sum of sizes of all sets of 1.0 million.

LiveJournal friendship Here we create one set for each LiveJournal user in a snapshot of the graph [27] where each set consists of the friends of the user. The ground set is the set of all users. There

are about 4 millions sets over 4 millions elements for a total size of 34 millions.

Gutenberg bi-grams This dataset is obtained from the Gutenberg project [1, 7]. Each set represents an English text and contains all the bi-grams of the text. There are 41 thousands sets over 99 millions elements for a total size of more than 1 billion.

Synthetic instance We constructed a synthetic coverage instance which is designed to be hard for the greedy algorithm. We fix the ground set U of size n and we create an optimal solution for size K covering all n items in the following way: We create K disjoint sets by partitioning U in K equal parts of size $\frac{n}{K}$.[4] All these sets are added to the input family $\mathbb{N}$. We also add to $\mathbb{N}$ other t random sets, each consisting of $s = \lceil \frac{n}{k}(1+\epsilon_1) \rceil$ randomly picked items w/o replacement.

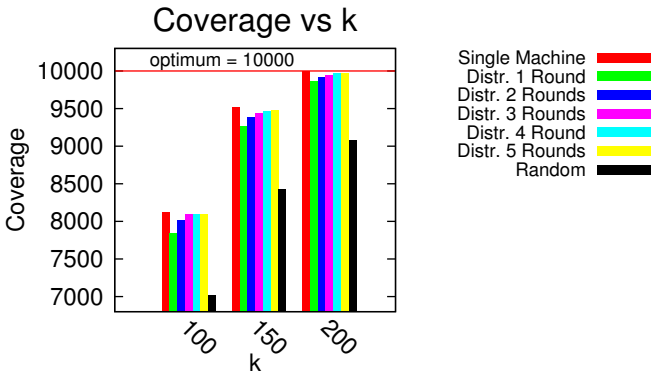
Experimental setup We first describe the implementation details of the distributed algorithm BicriteriaGreedy for coverage; the one for exemplar-based clustering is similar and will be sketched in the next section. The input of the algorithm is a dataset of n sets, a fixed size k of elements to output and number of rounds r. The algorithm outputs $k' = \lfloor \frac{k}{r} \rfloor$ sets at each round except in the last round where $k' = \lfloor \frac{k}{r} \rfloor + (k \mod r)$ sets are output (for a total of exactly k sets). Each round of the algorithm obtains k' sets in two steps. In the first step of each round the dataset is divided randomly in m blocks of data. Each set is assigned u.a.r. to a single block T_i from the m blocks $T_1, \ldots T_m$. We use multiplicity 1 as our experiments shows it is sufficient to achieve very good experiment results. We always fix m to be $\lceil \sqrt{n/k'} \rceil$. Then each block of data is analyzed independently executing the greedy max coverage algorithm to select k' sets. This is done in parallel by distributing the blocks across multiple machines. In the second step of each round, the mk' sets returned by the machines are gathered in a master machine and the same greedy algorithm is run to obtain the final k' set of the round. In this section we compare various outputting $k = K$ items and $k > K$ items for this algorithm as well as outputting uniformly at random sets.

Upperbound Since it is infeasible to compute exactly the optimum value even for small k values so we compare the algorithms with an upper-bound on the optimum solution value. One simple upperbound is given by the maximum value of the objective function (for coverage it is $|U|$). We also obtain a more sophisticated upperbound by post-processing the output of our algorithms as follows. Let $|S| = t$ be a solution obtained by our algorithm for size $t \geq k$. It can be show that $f(S)$ plus the sum of the top k marginal gains $\Delta(x, S)$ for any $x \in \mathbb{N}$ is a provable upperbound to the optimal value for any solution of size k. We report the results based on the best upperbound achieved for all (dataset, k) pairs.

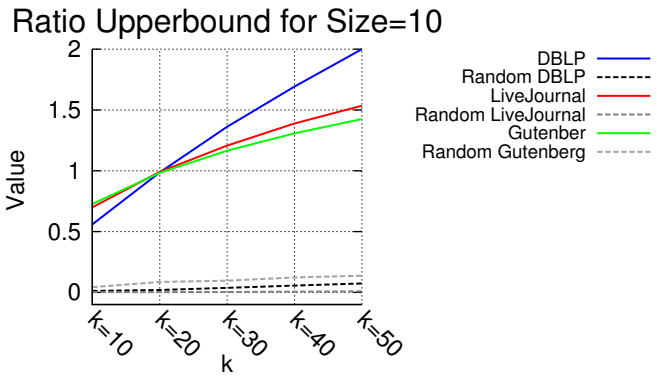
Results on synthetic instances In this experiment we set the size of universe in the synthetic instance to $|U| = 10{,}000$, the optimal solution size to $K = 100$, the number of random sets to $t = 100{,}000$ and $\epsilon_1 = 0.2$. The results are shown in Figure 1(a). It is possible to make the following observations.

First notice that consistent with our theoretical analysis, outputting $k \geq K$ items allows to converge to the optimal value for K with few additional items. In this experiment we get 95% and 99% of optimum for $K = 100$ outputting $k = 1.5K$ and $k = 2K$

[4]For simplicity we assume n multiple of K



(a) Synthetic dataset



(b) Real datasets

Figure 1: Coverage maximization

respectively. This confirm the main theoretical contribution of our paper. Second, it is possible to see that for hard instances the use of multiple rounds improves the solution w.r.t. the single round algorithm. Notice also that a small number of rounds is sufficient in practice to achieve results very close to the greedy algorithm ran on a single machine. After 5 rounds we see no significant difference (81% of upperbound with 5 rounds vs 81.2% for the single machine algorithm with $k = K = 100$). Similar results holds for other k and K values. This confirms our theoretical finding that the use of multiple rounds improves the solution in hard instances. Finally, as expected the greedy algorithm is always significantly better than a random output.

Results on real datasets We also ran the algorithms on our real datasets as shown in Figure 1(b). In this experiment, we fix a target solution size $K = 10$ and run the algorithms with different values of $k \geq K$. The figure shows the ratio of the value of the solution obtained by the distributed algorithm and the random baseline for different $k \geq K$'s sizes over the upper-bound we computed for the solution with size $K = 10$. We report the results for the distributed algorithm using a single round ($r = 1$) and $m = \sqrt{n/k}$. It is possible to notice that as expected outputting more items increases the value of the objective function. It is interesting to observe that in real instances the algorithm significantly exceeds

the worst case guarantees. With just $k = 2K$, we already obtain $> 98\%$, $> 99\%$, $> 98\%$ of our upperbound for the DBLP, LiveJournal, Gutenberg dataset respectively. We also ran the same experiments with more rounds and the results are very similar showing that for real instances algorithm already converges in one round.

4.2 Exemplar-based clustering

Exemplar-based clustering is a popular [22] way to identify k representative points from set of points with a notion of distance between them. In exemplar-based clustering we are given a set $\mathbb{N}$ of points and an arbitrary non-negative *distance* function (dist) over pairs of points.[5] For a set $S \subseteq \mathbb{N}$ we define the cost $c(S)$ of set S as $c(S) = \sum_{v \in \mathbb{N}} \min_{s \in S} \text{dist}(v, s)$. The cost $c(S)$ represent the sum of the minimum distances from every point in $\mathbb{N}$ to the nearest in S. Fix an point p_0 such that $\forall u, v \in \mathbb{N}$, $\text{dist}(u, v) \leq \text{dist}(u, p_0)$. We can now define the exemplar-based clustering as maximizing the following monotone submodular objective function $f(S) = c(\{p_0\}) - c(S \cup \{p_0\})$ for S of size k. Notice that this is equivalent to minimizing the cost of $c(S \cup \{p_0\})$.

We focus on the following datasets for exemplar-based clustering: **Wikipedia** has 3.8 millions vectors of 100 dimensions representing Wikipedia pages and **TinyImages** [26] has 80 millions vectors with 3072 dimensions representing images. We normalized all vectors in the datasets to have a unit L2 norm. We use as distance function the squared L2 distance, the maximum distance is 2 in all datasets, and we fix p_0 as a vector at distance 2 from any point in the dataset. We provide a more detailed overview on these datasets:

English Wikipedia dataset This is a dataset obtained from a snapshot of the entire English Wikipedia[6] with approximately 3.8 million articles. From each article we obtained a vector as follows: we extracted the text (discarding HTML tags, hyperlinks, removed stop words, etc) and then we ran Latent Dirichlet Allocation [16] with 100 topics using the `gensim` package [25]. The result is a probability distribution vector for each page of 100 dimensions.

TinyImages dataset This dataset [26], contains about 80 millions RGB images of size 32x32 obtained by an Internet crawl. Each image is represented as a $3072 = 3 \times 32 \times 32$ dimensional vector (one dimension for each pixel-color entry). From each image we obtain a 3072 dimensional vector by subtracting to each entry the average value of the entries in the vector. In the experiments involving the TinyImages dataset, to speed up the computation, we use a standard dimensionality reduction technique (i.e., Johnson Lindenstrauss random projection as in Achilopitas [3]) to convert the 3072-dimensional vectors to 300 dimensions before processing them. Notice that all objective function values shown are always computed on the original (unmodified) vectors.

Experimental setup The implementation details of the exemplar-based clustering algorithm are similar to that of the coverage one. We now highlight only the main differences. For this section we use a lazy variation of the greedy algorithm [22]. When analyzing a block of size N' of data to obtain k' elements we iterate over the block for k' times each time selecting a new element with maximum marginal gain. Here however the iteration evaluates only a independent u.a.r. subset of size $c\frac{N'}{k'}$ of the elements in the block as

[5]The function need not to respect the distance properties.
[6]https://meta.wikimedia.org/wiki/Data_dumps

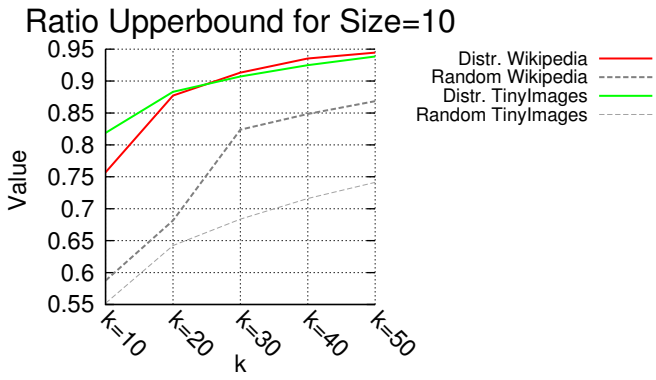


Figure 2: Exemplar-based clustering results.

in [22]. We fix $c = 3$, i.e. each element is evaluated in expectation 3 times over the iterations. Moreover, in this section, the objective function $f(S)$ in the greedy algorithm is estimated it by computing the distance of each element in S only to a u.a.r. sample V' of 500 elements from V (each machine receives an independent sample). Notice that when we report the values of the objective function for the solutions of the algorithm we do not use any estimation and we compute the exact value.

We will use similar techniques of the coverage case to compute an upperbound. The marginal gain in this case will be again estimated from a random sample of 500.

Results The results using a single round for the distributed algorithm are reported in Figure 2. Similarly to the coverage case we fix a target solution size $K = 10$ and run the algorithms with different $k \geq K$ values. The figure shows again the ratio of the value of the solution obtained for different $k \geq K$'s over the upper-bound for $K = 10$. It is possible to notice that the 1-round algorithm outputting more items increases the value objective significantly w.r.t to the outputing k items. Consistent with our theoretical analysis the algorithm approaches close to optimal value for the size $K = 10$ sets for $k \geq K$'s values (for $k = 2K$ we already obtain $> 87\%$, 88%, of our upperbound for Wikipedia and TinyImages, respectively). Similarly to the coverage case we observe strong convergence in just 1 round and great gap with a random baseline.

Speed-ups of the distributed framework We evaluated the gain obtained by the use of the distributed algorithm by comparing it to the lazy greedy algorithm ran in a single machine. Even for small datasets the centralized algorithm can take numerous hours to complete in a single machine even for small k values. Moreover, the distributed algorithm allows to analyze larger datasets as each machine need to store only a fraction $1/m$ of the dataset. Running the centralized algorithm on TinyImages would requires at least 200 GB of main memory.[7] We ran our framework fixing k and $m = \sqrt{N/k}$ and compared it with the centralized algorithm where a single machine is assigned the entire dataset. The speedup for the Wikipedia dataset using $k = 10$ and $k = 20$ was > 32 and > 37.

[7]While 200 GB memory machines are available, distributed computing with standard nodes (such as in MapReduce) is a more scalable approach widely used in practice. The algorithm could also be implemented using external memory and many passes, but this would increase even more the computational time.

Notice that we achieve substantial speedups even for such small datasets. The speedup for larger dataset are significantly larger however running the centralized algorithm on the larger dataset is simply infeasible (or expensive) for the memory requirement time required to run the experiment.

The centralized algorithm did not complete after many hours using larger k's or any other dataset. We also compared the solution obtained by the one round algorithm with the centralized one. The results shows that the distributed algorithm obtains $> 99.6\%$, $> 99.7\%$ of the value of the centralized one for $k = 10$ and $k = 20$ respectively.

5 CONCLUSIONS

We addressed the problem of submodular optimization under cardinality constraints in distributed setting. We analyzed an efficient constant-round distributed algorithm that can achieves the $1 - \epsilon$ approximation of the optimum for size k for $\epsilon > 0$ by allowing to output more than k items. We conductance an extensive empirical evaluation showing that almost optimum solutions can be obtained in few rounds in large-scale real dataset while outputting few more items.

Acknowledgements

We thank MohammadHossein Bateni for his comments and for sharing the datasets.

REFERENCES

[1] Gutenberg. search project gutenberg. https://www.gutenberg.org/ebooks/, 2016.
[2] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *KDD*, KDD '13, pages 32–40, New York, NY, USA, 2013. ACM.
[3] D. Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. of computer and System Sciences*, 2003.
[4] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, 2014.
[5] R. 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.
[6] R. Barbosa, A. Ene, H. L. Nguyen, and J. Ward. A new framework for distributed submodular maximization, 2016.
[7] M. Bateni, H. Esfandiari, and V. S. Mirrokni. Distributed coverage maximization via sketching. *CoRR*, abs/1612.02327, 2016.
[8] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan. Parallel and i/o efficient set covering algorithms. In *SPAA*, pages 82–90, 2012.
[9] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in map-reduce. In *WWW*, pages 231–240, 2010.
[10] G. Cormode, H. J. Karloff, and A. Wirth. Set cover algorithms for very large datasets. In *CIKM*, pages 479–488, 2010.
[11] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.
[12] U. Feige, V. S. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.
[13] B. J. Frey and D. Dueck. Mixture modeling by affinity propagation. In *NIPS*, pages 379–386, 2005.
[14] W. Gasarch and S. Fletcher. The egg game. http://www.cs.umd.edu/~gasarch/BLOGPAPERS/egg.pdf.
[15] A. Guillory and J. A. Bilmes. Active semi-supervised learning using submodular functions. *arXiv preprint arXiv:1202.3726*, 2012.
[16] M. Hoffman, F. R. Bach, and D. M. Blei. Online learning for latent dirichlet allocation. In *NIPS*, 2010.
[17] R. kiveris, S. Lattanzi, V. Mirrokni, V. Rastogi, and S. Vasilvitski. Connected components in mapreduce and beyond. In *ACM SOCC*, 2014.
[18] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. In *SPAA*, pages 1–10, 2013.
[19] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, pages 85–94, 2011.
[20] H. Lin and J. A. Bilmes. A class of submodular functions for document summarization. In *HLT*, pages 510–520, 2011.
[21] V. S. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *STOC*, pages 153–162, 2015.
[22] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. In *AAAI*, pages 1812–1818, 2015.
[23] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, pages 2049–2057, 2013.
[24] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsâĂŤi. *Mathematical Programming*, 14(1):265–294, 1978.
[25] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *LREC*, Valletta, Malta, 2010.
[26] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE TPAMI*, 2008.
[27] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.