Distributed Submodular Maximization: Identifying Representative Elements in Massive Data

Baharan Mirzasoleiman
ETH Zurich

Amin Karbasi
ETH Zurich

Rik Sarkar
University of Edinburgh

Andreas Krause
ETH Zurich

Abstract

Many large-scale machine learning problems (such as clustering, non-parametric learning, kernel machines, etc.) require selecting, out of a massive data set, a manageable yet representative subset. Such problems can often be reduced to maximizing a submodular set function subject to cardinality constraints. Classical approaches require centralized access to the full data set; but for truly large-scale problems, rendering the data centrally is often impractical. In this paper, we consider the problem of submodular function maximization in a distributed fashion. We develop a simple, two-stage protocol GREEDI, that is easily implemented using MapReduce style computations. We theoretically analyze our approach, and show, that under certain natural conditions, performance close to the (impractical) centralized approach can be achieved. In our extensive experiments, we demonstrate the effectiveness of our approach on several applications, including sparse Gaussian process inference and exemplar-based clustering, on tens of millions of data points using Hadoop.

1 Introduction

Numerous machine learning algorithms require selecting representative subsets of manageable size out of large data sets. Applications range from exemplar-based clustering [1], to active set selection for large-scale kernel machines [2], to corpus subset selection for the purpose of training complex prediction models [3]. Many such problems can be reduced to the problem of maximizing a submodular set function subject to cardinality constraints [4, 5].

Submodularity is a property of set functions with deep theoretical and practical consequences. Submodular maximization generalizes many well-known problems, e.g., maximum weighted matching, max coverage, and finds numerous applications in machine learning and social networks, such as influence maximization [6], information gathering [7], document summarization [3] and active learning [8, 9]. A seminal result of Nemhauser et al. [10] states that a simple greedy algorithm produces solutions competitive with the optimal (intractable) solution. In fact, if assuming nothing but submodularity, no efficient algorithm produces better solutions in general [11, 12].

Data volumes are increasing faster than the ability of individual computers to process them. Distributed and parallel processing is therefore necessary to keep up with modern massive datasets. The greedy algorithms that work well for centralized submodular optimization, however, are unfortunately sequential in nature; therefore they are poorly suited for parallel architectures. This mismatch makes it inefficient to apply classical algorithms directly to distributed setups.

In this paper, we develop a simple, parallel protocol called GREEDI for distributed submodular maximization. It requires minimal communication, and can be easily implemented in MapReduce style parallel computation models [13]. We theoretically characterize its performance, and show that under some natural conditions, for large data sets the quality of the obtained solution is competitive with the best centralized solution. Our experimental results demonstrate the effectiveness of our approach on a variety of submodular maximization problems. We show that for problems such as exemplar-based clustering and active set selection, our approach leads to parallel solutions that are very competitive with those obtained via centralized methods (98% in exemplar based clustering and 97% in active set selection). We implement our approach in Hadoop, and show how it enables sparse Gaussian process inference and exemplar-based clustering on data sets containing tens of millions of points.

2 Background and Related Work

Due to the rapid increase in data set sizes, and the relatively slow advances in sequential processing capabilities of modern CPUs, parallel computing paradigms have received much interest. Inhabiting a sweet spot of resiliency, expressivity and programming ease, the MapReduce style computing model [13] has emerged as prominent foundation for large scale machine learning and data mining algorithms [14, 15]. MapReduce works by distributing the data to independent machines, where it is processed in parallel by *map tasks* that produce key-value pairs. The output is shuffled, and combined by *reduce* tasks. Hereby, each *reduce* task processes inputs that share the same key. Their output either comprises the ultimate result, or forms the input to another MapReduce computation.

The problem of centralized maximization of submodular functions has received much interest, starting with the seminal work of [10]. Recent work has focused on providing approximation guarantees for more complex constraints. See [5] for a recent survey. The work in [16] considers an algorithm for online distributed submodular maximization with an application to sensor selection. However, their approach requires k stages of communication, which is unrealistic for large k in a MapReduce style model. The authors in [4] consider the problem of submodular maximization in a streaming model; however, their approach is not applicable to the general distributed setting. There has also been new improvements in the running time of the greedy solution for solving SET-COVER when the data is large and disk resident [17]. However, this approach is not parallelizable by nature.

Recently, specific instances of distributed submodular maximization have been studied. Such scenarios often occur in large-scale graph mining problems where the data itself is too large to be stored on one machine. Chierichetti et al. [18] address the MAX-COVER problem and provide a $(1-1/e-\epsilon)$ approximation to the centralized algorithm, however at the cost of passing over the data set many times. Their result is further improved by Blelloch et al. [19]. Lattanzi et al. [20] address more general graph problems by introducing the idea of filtering, namely, reducing the size of the input in a distributed fashion so that the resulting, much smaller, problem instance can be solved on a single machine. This idea is, in spirit, similar to our distributed method GREEDI. In contrast, we provide a more general framework, and analyze in which settings performance competitive with the centralized setting can be obtained.

3 The Distributed Submodular Maximization Problem

We consider the problem of selecting subsets out of a large data set, indexed by V (called ground set). Our goal is to maximize a non-negative set function $f : 2^V \rightarrow \mathbb{R}_+$, where, for $S \subseteq V$, $f(S)$ quantifies the utility of set S, capturing, e.g., how well S represents V according to some objective. We will discuss concrete instances of functions f in Section 3.1. A set function f is naturally associated with a *discrete derivative*

$$\triangle_f(e|S) \doteq f(S \cup \{e\}) - f(S), \tag{1}$$

where $S \subseteq V$ and $e \in V$, which quantifies the increase in utility obtained when adding e to set S. f is called *monotone* iff for all e and S it holds that $\triangle_f(e|S) \geq 0$. Further, f is *submodular* iff for all $A \subseteq B \subseteq V$ and $e \in V \setminus B$ the following diminishing returns condition holds:

$$\triangle_f(e|A) \geq \triangle_f(e|B). \tag{2}$$

Throughout this paper, we focus on such monotone submodular functions. For now, we adopt the common assumption that f is given in terms of a value oracle (a black box) that computes $f(S)$ for any $S \subseteq V$. In Section 4.5, we will discuss the setting where $f(S)$ itself depends on the entire data set V, and not just the selected subset S. Submodular functions contain a large class of functions that naturally arise in machine learning applications (c.f., [5, 4]). The simplest example of such functions are *modular* functions for which the inequality (2) holds with equality.

The focus of this paper is on maximizing a monotone submodular function (subject to some constraint) in a distributed manner. Arguably, the simplest form of constraints are cardinality constraints. More precisely, we are interested in the following optimization problem:

$$\max_{S \subseteq V} f(S) \quad \text{s.t.} \quad |S| \leq k. \tag{3}$$

We will denote by $A^{\mathrm{c}}[k]$ the subset of size at most k that achieves the above maximization, i.e., the best centralized solution. Unfortunately, problem (3) is NP-hard, for many classes of submodular functions [12]. However, a seminal result by Nemhauser et al. [10] shows that a simple greedy algorithm provides a $(1 - 1/e)$ approximation to (3). This greedy algorithm starts with the empty set S_0, and at each iteration i, it chooses an element $e \in V$ that maximizes (1), i.e., $S_i = S_{i-1} \cup \{\arg\max_{e \in V} \triangle_f(e|S_{i-1})\}$. Let $A^{\mathrm{gc}}[k]$ denote this greedy-centralized solution of size at most k. For several classes of monotone submodular functions, it is known that $(1 - 1/e)$ is the best approximation guarantee that one can hope for [11, 12, 21]. Moreover, the greedy algorithm can be accelerated using lazy evaluations [22].

In many machine learning applications where the ground set $|V|$ is large (e.g., cannot be stored on a single computer), running a standard greedy algorithm or its variants (e.g., lazy evaluation) in a centralized manner is infeasible. Hence, in those applications we seek a distributed solution, e.g., one that can be implemented using MapReduce-style computations (see Section 5). From the algorithmic point of view, however, the above greedy method is in general difficult to parallelize, since at each step, only the object with the highest marginal gain is chosen and every subsequent step depends on the preceding ones. More precisely, the problem we are facing in this paper is the following. Let the ground set V be partitioned into $V_1, V_2, \ldots, V_m$, i.e., $V = V_1 \cup V_2, \cdots \cup V_m$ and $V_i \cap V_j = \emptyset$ for $i \neq j$. We can think of V_i as a subset of elements (e.g., images) on machine i. The questions we are trying to answer in this paper are: how to distribute V among m machines, which algorithm should run on each machine, and how to merge the resulting solutions.

3.1 Example Applications Suitable for Distributed Submodular Maximization

In this part, we discuss two concrete problem instances, with their corresponding submodular objective functions f, where the size of the datasets often requires a distributed solution for the underlying submodular maximization.

Active Set Selection in Sparse Gaussian Processes (GPs): Formally a GP is a joint probability distribution over a (possibly infinite) set of random variables $\mathbf{X}_V$, indexed by our ground set V, such that every (finite) subset $\mathbf{X}_S$ for $S = \{e_1, \ldots, e_s\}$ is distributed according to a multivariate normal distribution, i.e., $P(\mathbf{X}_S = \mathbf{x}_S) = \mathcal{N}(\mathbf{x}_S; \mu_S, \Sigma_{S,S})$, where $\mu_S = (\mu_{e_1}, \ldots, \mu_{e_s})$ and $\Sigma_{S,S} = [\mathcal{K}_{e_i,e_j}](1 \leq i, j \leq k)$ are the prior mean vector and prior covariance matrix, respectively. The covariance matrix is parametrized via a (positive definite kernel) function $\mathcal{K}$. For example, a commonly used kernel function in practice where elements of the ground set V are embedded in a Euclidean space is the squared exponential kernel $\mathcal{K}_{e_i,e_j} = \exp(-|e_i - e_j|_2^2/h^2)$. In GP regression, each data point $e \in V$ is considered a random variable. Upon observations $\mathbf{y}_A = \mathbf{x}_A + \mathbf{n}_A$ (where $\mathbf{n}_A$ is a vector of independent Gaussian noise with variance σ^2), the predictive distribution of a new data point $e \in V$ is a normal distribution $P(\mathbf{X}_e \mid \mathbf{y}_A) = \mathcal{N}(\mu_{e|A}, \Sigma^2_{e|A})$, where

$$\mu_{e|A} = \mu_e + \Sigma_{e,A}(\Sigma_{A,A} + \sigma^2\mathbf{I})^{-1}(\mathbf{x}_A - \mu_A), \quad \sigma^2_{e|A} = \sigma^2_e - \Sigma_{e,A}(\Sigma_{A,A} + \sigma^2\mathbf{I})^{-1}\Sigma_{A,e}. \tag{4}$$

Note that evaluating (4) is computationally expensive as it requires a matrix inversion. Instead, most efficient approaches for making predictions in GPs rely on choosing a small – so called *active* – set of data points. For instance, in the Informative Vector Machine (IVM) one seeks a set S such that the information gain, $f(S) = I(\mathbf{Y}_S; \mathbf{X}_V) = H(\mathbf{X}_V) - H(\mathbf{X}_V|\mathbf{Y}_S) = \frac{1}{2}\log\det(\mathbf{I} + \sigma^{-2}\Sigma_{S,S})$ is maximized. It can be shown that this choice of f is monotone submodular [21]. For medium-scale problems, the standard greedy algorithms provide good solutions. In Section 5, we will show how GREEDI can choose near-optimal subsets out of a data set of 45 million vectors.

Exemplar Based Clustering: Suppose we wish to select a set of exemplars, that best represent a massive data set. One approach for finding such exemplars is solving the k-medoid problem [23], which aims to minimize the sum of pairwise dissimilarities between exemplars and elements of the dataset. More precisely, let us assume that for the data set V we are given a distance function $d : V \times V \to \mathbb{R}$ (not necessarily assumed symmetric, nor obeying the triangle inequality) such that $d(\cdot, \cdot)$ encodes dissimilarity between elements of the underlying set V. Then, the loss function for k-medoid can be defined as follows: $L(S) = \frac{1}{|V|} \sum_{e \in V} \min_{v \in S} d(e, v)$. By introducing an auxiliary element e_0 (e.g., $= 0$) we can turn L into a monotone submodular function: $f(S) = L(\{e_0\}) - L(S \cup \{e_0\})$. In words, f measures the decrease in the loss associated with the set S versus the loss associated with just the auxiliary element. It is easy to see that for suitable choice of e_0, maximizing f is equivalent to minimizing L. Hence, the standard greedy algorithm provides a very good solution. But again, the problem becomes computationally challenging when we have a large data set and we wish to extract a small set of exemplars. Our distributed solution GREEDI addresses this challenge.

3.2 Naive Approaches Towards Distributed Submodular Maximization

One way of implementing the greedy algorithm in parallel would be the following. We proceed in rounds. In each round, all machines – in parallel – compute the marginal gains of all elements in their sets V_i. They then communicate their candidate to a central processor, who identifies the globally best element, which is in turn communicated to the m machines. This element is then taken into account when selecting the next element and so on. Unfortunately, this approach requires synchronization after each of the k rounds. In many applications, k is quite large (e.g., tens of thousands or more), rendering this approach impractical for MapReduce style computations.

An alternative approach for large k would be to – on each machine – greedily select k/m elements independently (without synchronization), and then merge them to obtain a solution of size k. This approach is much more communication efficient, and can be easily implemented, e.g., using a single MapReduce stage. Unfortunately, many machines may select redundant elements, and the merged solution may suffer from diminishing returns.

In Section 4, we introduce an alternative protocol GREEDI, which requires little communication, while at the same time yielding a solution competitive with the centralized one, under certain natural additional assumptions.

4 The GREEDI Approach for Distributed Submodular Maximization

In this section we present our main results. We first provide our distributed solution GREEDI for maximizing submodular functions under cardinality constraints. We then show how we can make use of the geometry of data inherent in many practical settings in order to obtain strong data-dependent bounds on the performance of our distributed algorithm.

4.1 An Intractable, yet Communication Efficient Approach

Before we introduce GREEDI, we first consider an intractable, but communication–efficient parallel protocol to illustrate the ideas. This approach, shown in Alg. 1, first distributes the ground set V to m machines. Each machine then finds the *optimal* solution, i.e., a set of cardinality at most k, that maximizes the value of f in each partition. These solutions are then merged, and the optimal subset of cardinality k is found in the combined set. We call this solution $f(A^{\mathrm{d}}[m, k])$.

As the optimum centralized solution $A^{\mathrm{c}}[k]$ achieves the maximum value of the submodular function, it is clear that $f(A^{\mathrm{c}}[k]) \geq f(A^{\mathrm{d}}[m, k])$. Further, for the special case of selecting a single element $k = 1$, we have $A^{\mathrm{c}}[1] = A^{\mathrm{d}}[m, 1]$. In general, however, there is a gap between the distributed and the centralized solution. Nonetheless, as the following theorem shows, this gap cannot be more than $1/\min(m, k)$. Furthermore, this is the best result one can hope for under our two-round model.

Theorem 4.1. *Let f be a monotone submodular function and let $k > 0$. Then, $f(A^{d}[m, k])) \geq \frac{1}{\min(m,k)} f(A^{\mathrm{c}}[k])$. In contrast, for any value of m, and k, there is a data partition and a monotone submodular function f such that $f(A^{\mathrm{c}}[k]) = \min(m, k) \cdot f(A^{d}[m, k])$.*

Algorithm 1 Exact Distrib. Submodular Max.

Input: Set V, #of partitions m, constraints k.
Output: Set $A^{\mathrm{d}}[m,k]$.

1: Partition V into m sets $V_1, V_2, \ldots, V_m$.
2: In each partition V_i find the optimum set $A_i^c[k]$ of cardinality k.
3: Merge the resulting sets: $B = \cup_{i=1}^m A_i^c[k]$.
4: Find the optimum set of cardinality k in B. Output this solution $A^{\mathrm{d}}[m,k]$.

Algorithm 2 Greedy Dist. Subm. Max. (GREEDI)

Input: Set V, #of partitions m, constraints l, κ.
Output: Set $A^{\mathrm{gd}}[m,\kappa,l]$.

1: Partition V into m sets $V_1, V_2, \ldots, V_m$.
2: Run the standard greedy algorithm on each set V_i. Find a solution $A_i^{gc}[\kappa]$.
3: Merge the resulting sets: $B = \cup_{i=1}^m A_i^{gc}[\kappa]$.
4: Run the standard greedy algorithm on B until l elements are selected. Return $A^{\mathrm{gd}}[m,\kappa,l]$.

The proof of all the theorems can be found in the supplement. The above theorem fully characterizes the performance of two-round distributed algorithms in terms of the best centralized solution. A similar result in fact also holds for non-negative (not necessarily monotone) functions. Due to space limitation, the result is reported in the appendix. In practice, we cannot run Alg. 1. In particular, there is no efficient way to identify the optimum subset $A_i^c[k]$ in set V_i, unless P=NP. In the following, we introduce our efficient approximation GREEDI.

4.2 Our GREEDI Approximation

Our main efficient distributed method GREEDI is shown in Algorithm 2. It parallels the intractable Algorithm 1, but replaces the selection of optimal subsets by a greedy algorithm. Due to the approximate nature of the greedy algorithm, we allow the algorithms to pick sets slightly larger than k. In particular, GREEDI is a two-round algorithm that takes the ground set V, the number of partitions m, and the cardinality constraints l (final solution) and κ (intermediate outputs). It first distributes the ground set over m machines. Then each machine separately runs the standard greedy algorithm, namely, it sequentially finds an element $e \in V_i$ that maximizes the discrete derivative shown in (1). Each machine i – in parallel – continues adding elements to the set $A_i^{gc}[\cdot]$ until it reaches κ elements. Then the solutions are merged: $B = \cup_{i=1}^m A_i^{gc}[\kappa]$, and another round of greedy selection is performed over B, which this time selects l elements. We denote this solution by $A^{\mathrm{gd}}[m,\kappa,l]$: the greedy solution for parameters m, κ and l. The following result parallels Theorem 4.1.

Theorem 4.2. *Let f be a monotone submodular function and let $l, \kappa, k > 0$. Then*

$$f(A^{\mathrm{gd}}[m,\kappa,l])) \geq \frac{(1-e^{-\kappa/k})(1-e^{-l/\kappa})}{\min(m,k)} f(A^{\mathrm{c}}[k]).$$

For the special case of $\kappa = l = k$ the result of 4.2 simplifies to $f(A^{\mathrm{gd}}[m,\kappa,k]) \geq \frac{(1-1/e)^2}{\min(m,k)} f(A^{\mathrm{c}}[k])$. From Theorem 4.1, it is clear that in general one cannot hope to eliminate the dependency of the distributed solution on $\min(k,m)$. However, as we show below, in many practical settings, the ground set V and f exhibit rich geometrical structure that can be used to prove stronger results.

4.3 Performance on Datasets with Geometric Structure

In practice, we can hope to do much better than the worst case bounds shown above by exploiting underlying structures often present in real data and important set functions. In this part, we assume that a metric d exists on the data elements, and analyze performance of the algorithm on functions that change gracefully with change in the input. We refer to these as *Lipschitz functions*. More formally, a function $f : 2^V \to \mathbb{R}$ is λ-*Lipschitz*, if for equal sized sets $S = \{e_1, e_2, \ldots, e_k\}$ and $S' = \{e'_1, e'_2, \ldots, e'_k\}$ and for any matching of elements: $M = \{(e_1, e'_1), (e_2, e'_2) \ldots, (e_k, e'_k)\}$, the difference between $f(S)$ and $f(S')$ is bounded by the total of distances between respective elements: $|f(S) - f(S')| \leq \lambda \sum_i d(e_i, e'_i)$. It is easy to see that the objective functions from both examples in Section 3.1 are λ-Lipschitz for suitable kernels/distance functions. Two sets S and S' are ε-*close* with respect to f, if $|f(S) - f(S')| \leq \varepsilon$. Sets that are close with respect to f can be thought of as good candidates to approximate the value of f over each-other; thus one such set is a good representative of the other. Our goal is to find sets that are suitably close to $A^{\mathrm{c}}[k]$. At an element $v \in V$, let us define its α-*neighborhood* to be the set of elements within a distance α from

v (i.e., α-close to v): $N_\alpha(v) = \{w : d(v, w) \leq \alpha\}$. We can in general consider α-neighborhoods of points of the metric space.

Our algorithm GREEDI partitions V into sets $V_1, V_2, \ldots V_m$ for parallel processing. In this subsection, *we assume that* GREEDI *performs the partition by assigning elements uniformly randomly to the machines.* The following theorem says that if the α-neighborhoods are sufficiently dense and f is a λ-lipschitz function, then this method can produce a solution close to the centralized solution:

Theorem 4.3. *If for each* $e_i \in A^c[k], |N_\alpha(e_i)| \geq km \log(k/\delta^{1/m})$, *and algorithm* GREEDI *assigns elements uniformly randomly to* m *processors , then with probability at least* $(1 - \delta)$,

$$f(A^{gd}[m, \kappa, l]) \geq (1 - e^{-\kappa/k})(1 - e^{-l/\kappa})(f(A^c[k]) - \lambda\alpha k).$$

4.4 Performance Guarantees for Very Large Data Sets

Suppose that our data set is a finite sample drawn from an underlying *infinite* set, according to some unknown probability distribution. Let $A^c[k]$ be an optimal solution in the infinite set such that around each $e_i \in A^c[k]$, there is a neighborhood of radius at least α^*, where the probability density is at least β at all points, for some constants α^* and β. This implies that the solution consists of elements coming from reasonably dense and therefore representative regions of the data set.

Let us consider $g : \mathbb{R} \to \mathbb{R}$, the *growth function of the metric*. $g(\alpha)$ is defined to be the volume of a ball of radius α centered at a point in the metric space. This means, for $e_i \in A^c[k]$ the probability of a random element being in $N_\alpha(e_i)$ is at least $\beta g(\alpha)$ and the expected number of α neighbors of e_i is at least $E[|N_\alpha(e_i)|] = n\beta g(\alpha)$. As a concrete example, Euclidean metrics of dimension D have $g(\alpha) = O(\alpha^D)$. Note that for simplicity we are assuming the metric to be homogeneous, so that the growth function is the same at every point. For heterogeneous spaces, we require g to be a uniform lower bound on the growth function at every point.

In these circumstances, the following theorem guarantees that if the data set V is sufficiently large and f is a λ-lipschitz function, then GREEDI produces a solution close to the centralized solution.

Theorem 4.4. *For* $n \geq \dfrac{8km \log(k/\delta^{1/m})}{\beta g(\frac{\varepsilon}{\lambda k})}$, *where* $\frac{\varepsilon}{\lambda k} \leq \alpha^*$, *if the algorithm* GREEDI *assigns elements uniformly randomly to* m *processors , then with probability at least* $(1 - \delta)$,

$$f(A^{gd}[m, \kappa, l]) \geq (1 - e^{-\kappa/k})(1 - e^{-l/\kappa})(f(A^c[k]) - \varepsilon).$$

4.5 Handling Decomposable Functions

So far, we have assumed that the objective function f is given to us as a black box, which we can evaluate for any given set S *independently* of the data set V. In many settings, however, the objective f depends itself on the entire data set. In such a setting, we cannot use GREEDI as presented above, since we cannot evaluate f on the individual machines without access to the full set V. Fortunately, many such functions have a simple structure which we call *decomposable*. More precisely, we call a monotone submodular function f *decomposable* if it can be written as a sum of (non-negative) monotone submodular functions as follows: $f(S) = \frac{1}{|V|} \sum_{i \in V} f_i(S)$. In other words, there is separate monotone submodular function associated with every data point $i \in V$. We require that each f_i can be evaluated without access to the full set V. Note that the exemplar based clustering application we discussed in Section 3.1 is an instance of this framework, among many others.

Let us define the evaluation of f restricted to $D \subseteq V$ as follows: $f_D(S) = \frac{1}{|D|} \sum_{i \in D} f_i(S)$. Then, in the remaining of this section, our goal is to show that assigning each element of the data set randomly to a machine and running GREEDI will provide a solution that is with high probability close to the optimum solution. For this, let us assume the f_i's are bounded, and without loss of generality $0 \leq f_i(S) \leq 1$ for $1 \leq i \leq |V|, S \subseteq V$. Similar to Section 4.3 we assume that GREEDI performs the partition by assigning elements uniformly randomly to the machines. These machines then each greedily optimize f_{V_i}. The second stage of GREEDI optimizes f_U, where $U \subseteq V$ is chosen uniformly at random, of size $\lceil n/m \rceil$. Then, we can show the following result.

Theorem 4.5. *Let $m, k, \delta > 0$, $\epsilon < 1/4$ and let n_0 be an integer such that for $n \geq n_0$ we have $\ln(n)/n \leq \epsilon^2/(mk)$. For $n \geq \max(n_0, m\log(\delta/4m)/\epsilon^2)$, and under the assumptions of Theorem 4.4, we have, with probability at least $1-\delta$,*

$$f(A^{\text{gd}}[m, \kappa, l]) \geq (1 - e^{-\kappa/k})(1 - e^{-l/\kappa})(f(A^{\text{c}}[k]) - 2\varepsilon).$$

The above result demonstrates why GREEDI performs well on decomposable submodular functions with massive data even when they are evaluated locally on each machine. We will report our experimental results on exemplar-based clustering in the next section.

5 Experiments

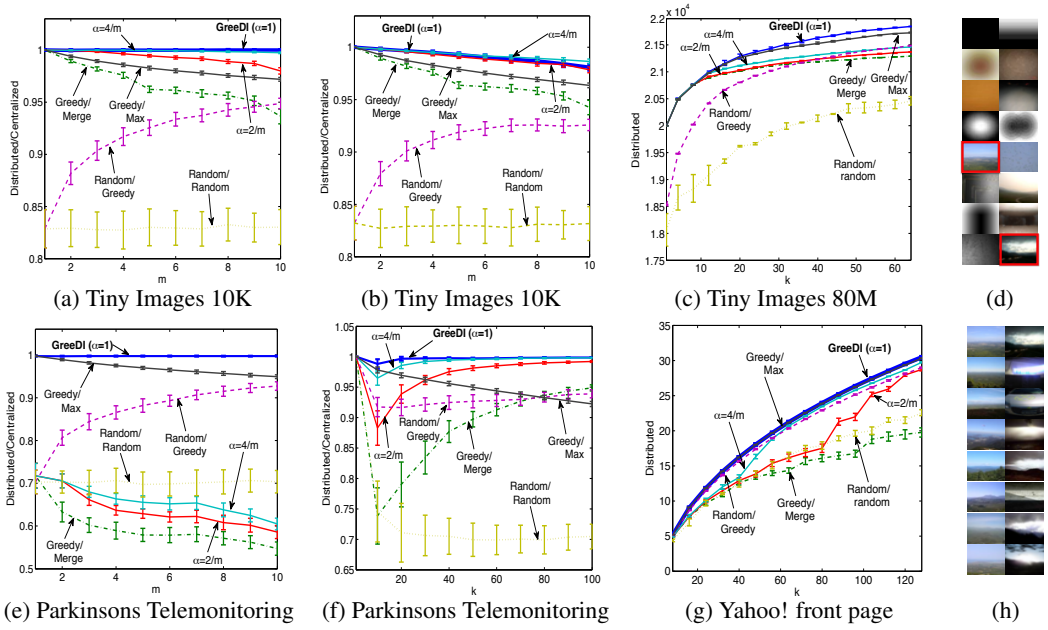
In our experimental evaluation we wish to address the following questions: 1) how well does GREEDI perform compared to a centralized solution, 2) how good is the performance of GREEDI when using decomposable objective functions (see Section 4.5), and finally 3) how well does GREEDI scale on massive data sets. To this end, we run GREEDI on two scenarios: exemplar based clustering and active set selection in GPs. Further experiments are reported in the supplement.

We compare the performance of our GREEDI method (using different values of $\alpha = \kappa/k$) to the following naive approaches: a) *random/random*: in the first round each machine simply outputs k randomly chosen elements from its local data points and in the second round k out of the merged mk elements, are again randomly chosen as the final output. b) *random/greedy*: each machine outputs k randomly chosen elements from its local data points, then the standard greedy algorithm is run over mk elements to find a solution of size k. c) *greedy/merge*: in the first round k/m elements are chosen greedily from each machine and in the second round they are merged to output a solution of size k. d) *greedy/max*: in the first round each machine greedily finds a solution of size k and in the second round the solution with the maximum value is reported. For data sets where we are able to find the centralized solution, we report the ratio of $f(A_{\text{dist}}[k])/f(A^{\text{gc}}[k])$, where $A_{\text{dist}}[k]$ is the distributed solution (in particular $A^{\text{gd}}[m, \alpha k, k] = A_{\text{dist}}[k]$ for GREEDI).

Exemplar based clustering. Our exemplar based clustering experiment involves GREEDI applied to the clustering utility $f(S)$ (see Sec. 3.1) with $d(x, x') = \|x - x'\|^2$. We performed our experiments on a set of 10,000 *Tiny Images* [24]. Each 32 by 32 RGB pixel image was represented by a 3,072 dimensional vector. We subtracted from each vector the mean value, normalized it to unit norm, and used the origin as the auxiliary exemplar. Fig. 1a compares the performance of our approach to the benchmarks with the number of exemplars set to $k = 50$, and varying number of partitions m. It can be seen that GREEDI significantly outperforms the benchmarks and provides a solution that is very close to the centralized one. Interestingly, even for very small $\alpha = \kappa/k < 1$, GREEDI performs very well. Since the exemplar based clustering utility function is decomposable, we repeated the experiment for the more realistic case where the function evaluation in each machine was restricted to the local elements of the dataset in that particular machine (rather than the entire dataset). Fig 1b shows similar qualitative behavior for decomposable objective functions.

Large scale experiments with Hadoop. As our first large scale experiment, we applied GREEDI to the whole dataset of 80,000,000 *Tiny Images* [24] in order to select a set of 64 exemplars. Our experimental infrastructure was a cluster of 10 quad-core machines running Hadoop with the number of reducers set to $m = 8000$. Hereby, each machine carried out a set of reduce tasks in sequence. We first partitioned the images uniformly at random to reducers. Each reducer separately performed the lazy greedy algorithm on its own set of 10,000 images (≈123MB) to extract 64 images with the highest marginal gains w.r.t. the local elements of the dataset in that particular partition. We then merged the results and performed another round of lazy greedy selection on the merged results to extract the final 64 exemplars. Function evaluation in the second stage was performed w.r.t a randomly selected subset of 10,000 images from the entire dataset. The maximum running time per reduce task was 2.5 hours. As Fig. 1c shows, GREEDI highly outperforms the other distributed benchmarks and can scale well to very large datasets. Fig. 1d shows a set of cluster exemplars discovered by GREEDI where each column in Fig. 1h shows 8 nearest images to exemplars 9 and 16 (shown with red borders) in Fig. 1d.

Active set selection. Our active set selection experiment involves GREEDI applied to the information gain $f(S)$ (see Sec. 3.1) with Gaussian kernel, $h = 0.75$ and $\sigma = 1$. We used the *Parkinsons Telemonitoring* dataset [25] consisting of 5,875 bio-medical voice measurements with 22 attributes



(a) Tiny Images 10K (b) Tiny Images 10K (c) Tiny Images 80M (d)

(e) Parkinsons Telemonitoring (f) Parkinsons Telemonitoring (g) Yahoo! front page (h)

Figure 1: Performance of GREEDI compared to the other benchmarks. a) and b) show the mean and standard deviation of the ratio of distributed vs. centralized solution for global and local objective functions with budget $k = 50$ and varying the number m of partitions, for a set of 10,000 *Tiny Images*. c) shows the distributed solution with $m = 8000$ and varying k for local objective functions on the whole dataset of 80,000,000 *Tiny Images*. e) shows the ratio of distributed vs. centralized solution with $m = 10$ and varying k for *Parkinsons Telemonitoring*. f) shows the same ratio with $k = 50$ and varying m on the same dataset, and g) shows the distributed solution for $m = 32$ with varying budget k on *Yahoo! Webscope data*. d) shows a set of cluster exemplars discovered by GREEDI, and each column in h) shows 8 images nearest to exemplars 9 and 16 in d).

from people with early-stage Parkinson's disease. We normalized the vectors to zero mean and unit norm. Fig. 1f compares the performance GREEDI to the benchmarks with fixed $k = 50$ and varying number of partitions m. Similarly, Fig 1e shows the results for fixed $m = 10$ and varying k. We find that GREEDI significantly outperforms the benchmarks.

Large scale experiments with Hadoop. Our second large scale experiment consists of 45,811,883 user visits from the Featured Tab of the Today Module on Yahoo! Front Page [26]. For each visit, both the user and each of the candidate articles are associated with a feature vector of dimension 6. Here, we used the normalized user features. Our experimental setup was a cluster of 5 quad-core machines running Hadoop with the number of reducers set to $m = 32$. Each reducer performed the lazy greedy algorithm on its own set of 1,431,621 vectors (≈34MB) in order to extract 128 elements with the highest marginal gains w.r.t the local elements of the dataset in that particular partition. We then merged the results and performed another round of lazy greedy selection on the merged results to extract the final active set of size 128. The maximum running time per reduce task was 2.5 hours. Fig. 1g shows the performance of GREEDI compared to the benchmarks. We note again that GREEDI significantly outperforms the other distributed benchmarks and can scale well to very large datasets.

6 Conclusion

We have developed an efficient distributed protocol GREEDI, for maximizing a submodular function subject to cardinality constraints. We have theoretically analyzed the performance of our method and showed under certain natural conditions it performs very close to the centralized (albeit impractical in massive data sets) greedy solution. We have also demonstrated the effectiveness of our approach through extensive large scale experiments using Hadoop. We believe our results provide an important step towards solving submodular optimization problems in very large scale, real applications.

Acknowledgments. This research was supported by SNF 200021-137971, DARPA MSEE FA8650-11-1-7156, ERC StG 307036, a Microsoft Faculty Fellowship, an ETH Fellowship, Scottish Informatics and Computer Science Alliance.

References

[1] Delbert Dueck and Brendan J. Frey. Non-metric affinity propagation for unsupervised image categorization. In *ICCV*, 2007.

[2] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. 2006.

[3] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *North American chapter of the Assoc. for Comp. Linguistics/Human Lang. Tech.*, 2011.

[4] Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *Proc. International Conference on Machine Learning (ICML)*, 2010.

[5] Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2013.

[6] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD*, 2003.

[7] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology*, 2011.

[8] Andrew Guillory and Jeff Bilmes. Active semi-supervised learning using submodular functions. In *Uncertainty in Artificial Intelligence (UAI)*, Barcelona, Spain, July 2011. AUAI.

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

[10] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 1978.

[11] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Research*, 1978.

[12] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 1998.

[13] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[14] Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, and Andrew Y. Ng. Map-reduce for machine learning on multicore. In *NIPS*, 2006.

[15] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. Mapreduce for data intensive scientific analyses. In *Proc. of the 4th IEEE Inter. Conf. on eScience*.

[16] Daniel Golovin, Matthew Faulkner, and Andreas Krause. Online distributed sensor selection. In *IPSN*, 2010.

[17] Graham Cormode, Howard Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. In *Proc. of the 19th ACM intern. conf. on Inf. knowl. manag.*

[18] Flavio Chierichetti, Ravi Kumar, and Andrew Tomkins. Max-cover in map-reduce. In *Proceedings of the 19th international conference on World wide web*, 2010.

[19] Guy E. Blelloch, Richard Peng, and Kanat Tangwongsan. Linear-work greedy parallel approximate set cover and variants. In *SPAA*, 2011.

[20] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, 2011.

[21] A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proc. of Uncertainty in Artificial Intelligence (UAI)*, 2005.

[22] M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques, LNCS*, pages 234–243, 1978.

[23] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. Wiley-Interscience, 2009.

[24] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2008.

[25] Athanasios Tsanas, Max A Little, Patrick E McSharry, and Lorraine O Ramig. Enhanced classical dysphonia measures and sparse regression for telemonitoring of parkinson's disease progression. In *IEEE Int. Conf. Acoust. Speech Signal Process.*, 2010.

[26] Yahoo! academic relations. r6a, yahoo! front page today module user click log dataset, version 1.0, 2012.

[27] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social networks*, 2009.

Appendix A: Proofs

This appendix presents the complete proofs of theorems presented in the article. For a set function f, we use the notation $f(S \mid S') = f(S \cup S') - f(S')$.

Proof of Theorem 4.1

$\Rightarrow$ direction:
The proof follows from the following lemmas.

Lemma 6.1. $\max_i f(A_i^{\mathrm{c}}[k]) \geq \frac{1}{m} f(A^{\mathrm{c}}[k])$.

Proof: Let B_i be the elements in V_i that are contained in the optimal solution, $B_i = A^{\mathrm{c}}[k] \cap V_i$. Then we have:
$f(A^{\mathrm{c}}[k]) = f(B_1 \cup \ldots \cup B_m) = f(B_1) + f(B_2|B_1) + \ldots + f(B_m|B_{m-1}, \ldots, B_1)$. Using submodularity of f, for each $i \in 1 \ldots m$, we have $f(B_i|B_{i-1} \ldots B_1) \leq f(B_i)$ and thus, $f(A^{\mathrm{c}}[k]) \leq f(B_1) + \ldots + f(B_m)$. Since, $f(A_i^{\mathrm{c}}[k]) \geq f(B_i)$, we have $f(A^{\mathrm{c}}[k]) \leq f(A_1^{\mathrm{c}}[k]) + \ldots + f(A_m^{c}[k])$. Therefore, $f(A^{\mathrm{c}}[k]) \leq m \ \max_i f(A_i^{\mathrm{c}}[k])$. □

Lemma 6.2. $\max_i f(A_i^{\mathrm{c}}[k]) \geq \frac{1}{k} f(A^{\mathrm{c}}[k])$.

Proof: Let $f(A^{\mathrm{c}}[k]) = f(\{u_1, \ldots u_k\})$. Using submodularity of f, we have $f(A^{\mathrm{c}}[k]) \leq \sum_{i=1}^{k} f(u_i)$. Thus, $f(A^{\mathrm{c}}[k]) \leq k f(u^*)$ where $u^* = \arg\max_i f(u_i)$. Suppose that the element with highest marginal gain (u^*) is in V_j. Then the maximum value of f on V_j would be greater or equal to the marginal gain of u^*, i.e. $f(A_j^{\mathrm{c}}[k]) \geq f(u^*)$ and since $f(\max_i f(A_i^{\mathrm{c}}[k])) \geq f(A_j^{\mathrm{c}}[k])$, we can conclude that $f(\max_i f(A_i^{\mathrm{c}}[k])) \geq f(u^*) \geq \frac{1}{k} f(A^{\mathrm{c}}[k])$. □

Since $f(A^{\mathrm{d}}[m,k]) \geq \max_i f(A_i^{\mathrm{c}}[k])$; from Lemma **??** and **??** we have $f(A^{\mathrm{d}}[m,k]) \geq \frac{1}{min(m,k)} f(A^{\mathrm{c}}[k])$.

$\Leftarrow$ direction:
Let us consider a set of unbiased and independent Bernoulli random variables $X_{i,j}$ for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, k\}$, i.e., $\Pr(X_{i,j} = 1) = \Pr(X_{i,j} = 0) = 1/2$ and $(X_{i,j} \perp X_{i',j'})$ if $i \neq i'$ or $j \neq j'$. Let us also define $Y_i = (X_{i,1}, \ldots, X_{i,k})$ for $i \in \{1, \ldots, m\}$. Now assume that $V_i = \{X_{i,1}, \ldots, X_{i,k}, Y_i\}, V = \bigcup_{i=1}^{m} V_i$ and $f_{S \subseteq V}(S) = H(S)$, where H is the entropy of the subset S of random variables. Note that H is a monotone submodular function. It is easy to see that $A_i^{\mathrm{c}}[k] = \{X_{i,1}, \ldots, X_{i,k}\}$ or $A_i^{\mathrm{c}}[k] = Y_i$ as in both cases $H(A_i^{\mathrm{c}}[k]) = k$. If we assume $A_i^{\mathrm{c}}[k] = \{X_{i,1}, \ldots, X_{i,k}\}$, then $B = \{X_{i,j} | 1 \leq i \leq m, 1 \leq j \leq k\}$. Hence, by selecting at most k elements from B, we have $H(A^{\mathrm{d}}[m,k]) = k$. On the other hand, the set of k elements that maximizes the entropy is $\{Y_1, \ldots, Y_m\}$. Note that $H(Y_i) = k$ and $Y_i \perp Y_j$ for $i \neq j$. Hence, $H(A^{c}) = k \cdot m$ if $m \geq k$ or otherwise $H(A^{\mathrm{c}}[k]) = k^2$.

Proof of Theorem 4.2

Let us first mention a slight generalization of the performance guarantee for the standard greedy algorithm. It follows immediately from the argument in [10], see, e.g., [5].

Lemma 6.3. *Let f be a non-negative submodular function, and let $A^{\mathrm{gc}}[q]$ of cardinality q be the greedy selected set by the standard greedy algorithm selecting k elements. Then,*

$$f(A^{\mathrm{gc}}[q]) \geq \left(1 - e^{-\frac{q}{k}}\right) f(A^{\mathrm{c}}[k]).$$

By Lemma **??** we know that $f(A_i^{\mathrm{gc}}[\kappa]) \geq (1 - \exp(-\kappa/k)) f(A_i^{\mathrm{c}}[k])$. Now, let us define

$$B^{\mathrm{gc}} = \cup_{i=1}^{m} A_i^{\mathrm{gc}}[\kappa], \quad \tilde{A}[\kappa] = \arg \max_{S \subseteq B^{\mathrm{gc}} \& |S| \leq \kappa} f(S).$$

Then by using Lemma **??** again, we obtain

$$f(A^{\mathrm{gc}}[m,\kappa,l]) \geq (1-\exp(-l/\kappa))f(\tilde{A}[\kappa]) \geq \frac{(1-\exp(-l/\kappa))(1-\exp(-\kappa/k))}{\min(m,k)} f(A^{\mathrm{c}}[k]). \quad (5)$$

Proof of Theorem 4.3

First, we need the following lemma.

Lemma 6.4. *If for each $e_i \in A^{\mathrm{c}}[k], |N_\alpha(e_i)| \geq km\log(k/\delta^{1/m})$, and if V is partitioned into sets $V_1, V_2, \ldots V_m$, where each element is randomly assigned to one set with equal probabilities, then there is at least one partition with a subset $A_i^{\mathrm{c}}[k]$ such that $|f(A^{\mathrm{c}}[k]) - f(A_i^{\mathrm{c}}[k])| \leq \lambda\alpha k$ with probability at least $(1-\delta)$.*

Proof: By the hypothesis, the α neighborhood of each element in $A^{\mathrm{c}}[k]$ contains at least $km\log(k/\delta^{1/m})$ elements. For each $e_i \in A^{\mathrm{c}}[k]$, let us take a set of $m\log(k/\delta^{1/m})$ elements from its α-neighborhood. These sets can be constructed to be mutually disjoint, since each α-neighborhood contains $m\log(k/\delta^{1/m})$ elements. We wish to show that at least one of the m partitions of V contains elements from α-neighborhoods of each element.

Each of the $m\log(k/\delta^{1/m})$ elements goes into a particular V_j with a probability $1/m$. The probability that a particular V_j does not contain an element α-close to $e_i \in A^{\mathrm{c}}[k]$ is $\dfrac{\delta^{1/m}}{k}$. The probability that V_j does not contain elements α-close to one or more of the k elements is at most $\delta^{1/m}$ (by union bound). The probability that *each* $V_1, V_2, \ldots V_m$ does not contain elements from the α-neighborhood of one or more of the k elements is at most δ. Thus, with high probability of at least $(1-\delta)$, at least one of $V_1, V_2, \ldots V_m$ contains an $A_i^{\mathrm{c}}[k]$ that is $\lambda\alpha k$-close to $A^{\mathrm{c}}[k]$. □

By Lemma **??**, for some V_i, $|f(A^{\mathrm{c}}[k]) - f(A_i^{\mathrm{c}}[k]|) \leq \lambda\alpha k$ with the given probability. And $f(A_i^{gd}[\kappa]) \geq (1-e^{-\kappa/k})f(A_i^{\mathrm{c}}[k])$ Lemma **??**. Therefore, the result follows using arguments analogous to the proof of Theorem 4.2.

Proof of Theorem 4.4

The following lemma says that in a sample drawn from distribution over an infinite dataset, a sufficiently large sample size guarantees a dense neighborhood near each element of $A^{\mathrm{c}}[k]$ when the elements are from representative regions of the data.

Lemma 6.5. *A number of elements: $n \geq \dfrac{8km\log(k/\delta^{1/m})}{\beta g(\alpha)}$, where $\alpha \leq \alpha^*$, suffices to have at least $4km\log(k/\delta^{1/m})$ elements in the α-neighborhood of each $e_i \in A^{\mathrm{c}}[k]$ with probability at least $(1-\delta)$, for small values of δ.*

Proof: The expected number of α-neighbors of an $e_i \in A^{\mathrm{c}}[k]$, is $E[|N_\alpha(e_i)|] \geq 8km\log(k/\delta^{1/m})$. We now show that in a random set of samples, at least a half of this number of neighbors is realized with high probability near each element of $A^{\mathrm{c}}[k]$.

This follows from a Chernoff bound:

$$P[|N_\alpha(e_i)| \leq 4km\log(k/\delta^{1/m})] \leq e^{-km\log(k/\delta^{1/m})} \leq (\delta^{1/m}/k)^{km}.$$

Therefore, the probability that some $e_i \in A^{\mathrm{c}}[k]$ does not have a suitable sized neighborhood is at most $k(\delta^{1/m}/k)^{km}$. For $\delta \leq 1/k$, $k\delta^{km} \leq \delta^m$. Therefore, with probability at least $(1-\delta)$, the α-neighborhood of each element $e_i \in A^{\mathrm{c}}[k]$ contains at least $4km\log(1/\delta)$ elements. □

Lemma 6.6. *For $n \geq \dfrac{8km\log(k/\delta^{1/m})}{\beta g(\frac{\varepsilon}{\lambda k})}$, where $\frac{\varepsilon}{\lambda k} \leq \alpha^*$, if V is partitioned into sets $V_1, V_2, \ldots V_m$, where each element is randomly assigned to one set with equal probabilities, then for sufficiently small values of δ, there is at least one partition with a subset $A_i^{\mathrm{c}}[k]$ such that $|f(A^{\mathrm{c}}[k]) - f(A_i^{\mathrm{c}}[k])| \leq \varepsilon$ with probability at least $(1-\delta)$.*

Proof: Follows directly by combining lemma **??** and lemma **??**. The probability that some element does not have a sufficiently dense $\varepsilon/\lambda k$-neighborhood with $km\log(2k/\delta^{1/m})$ elements is at most $(\delta/2)$ for sufficiently small delta, and the probability that some partition does not contain elements from the one or more of the dense neighborhoods is at most $(\delta/2)$. Therefore, the result holds with probability at least $(1-\delta)$. □

By lemma **??**, there is at least one V_i such that $|f(A^{\mathrm{c}}[k]) - f(A_i^{\mathrm{c}}[k])| \leq \varepsilon$ with the given probability. And $f(A_i^{gd}[\kappa]) \geq (1-e^{-\kappa/k})f(A_i^{\mathrm{c}}[k])$ using Lemma **??**. The result follows using arguments analogous to the proof of Theorem 4.2.

Proof of Theorem 4.5

Note that each machine has on the average n/m elements. Let us define Π_i the event that $n/2m < |V_i| < 2n/m$. Then based on the Chernoff bound we know that $\Pr(\neg\Pi_i) \leq 2\exp(-n/8m)$. Let us also define $\xi_i(S)$ the event that $|f_{V_i}(S) - f(S)| < \epsilon$, for some fixed $\epsilon < 1$ and a fixed set S with $|S| \leq k$. Note that $\xi_i(S)$ denotes the event that the empirical mean is close to the true mean. Based on the Hoeffding inequality (without replacement) we have $\Pr(\neq \xi_i S| \leq 2\exp(-2n\epsilon^2/m)$. Hence,

$$\Pr(\xi_i(S) \wedge \Pi_i) \geq 1 - 2\exp(-2n\epsilon^2/m) - 2\exp(-n/8m).$$

Let ξ_i be an event that $|f_{V_i}(S) - f(S)| < \epsilon$, for any S such that $|S| \leq \kappa$. Note that there are at most n^κ sets of size at most κ. Hence,

$$\Pr(\xi_i \wedge \Pi_i) \geq 1 - 2n^\kappa(\exp(-2n\epsilon^2/m) - \exp(-n/8m)).$$

As a result, for $\epsilon < 1/4$ we have

$$\Pr(\xi_i \wedge \Pi_i) \geq 1 - 4n^\kappa\exp(-2n\epsilon^2/m).$$

Since there are m machines, by the union bound we can conclude that

$$\Pr((\xi_i \wedge \Pi_i) \text{ on all machines}) \geq 1 - 4mn^\kappa\exp(-2n\epsilon^2/m).$$

The above calculation implies that we need to choose $\delta \geq 4mn^\kappa\exp(-2n\epsilon^2/m)$. Let n_0 be chosen in a way that for any $n \geq n_0$ we have $\ln(n)/n \leq \epsilon^2/(mk)$. Then, we need to choose n as follows:

$$n = \max\left(n_0, \frac{m\log(\delta/4m)}{\epsilon^2}\right).$$

Hence for the above choice of n, there is at least one V_i such that $|f(A^{\mathrm{c}}[k]) - f(A_i^{\mathrm{c}}[\kappa])| \leq \varepsilon$ with probability $1-\delta$. Hence the solution is ϵ away from the optimum solution with probability $1-\delta$. Now if we confine the evaluation of $f(A_i^{\mathrm{c}})$ to data points only in machine i then under the assumption of Theorem 4.4 we loose another ϵ. Formally, the result at this point simply follows by combining Theorem 4.2 and Theorem 4.4.

Appendix B: Additional Experiments

Finding maximum cuts. We also applied GREEDI to the problem of finding maximum cuts in graphs. In our setting we used a *Facebook-like social network* [27]. This dataset includes the users that have sent or received at least one message in an online student community at University of California, Irvine and consists of 1,899 users and 20,296 directed ties. Fig. **??** shows the performance of GREEDI applied to the cut function on graphs. We evaluated the objective function locally on each partition. Thus, the links between the partitions are disconnected.

This experiment violates several assumptions we made: 1) the cut function is submodular but not monotonic (and hence neither our theory holds, nor is the greedy algorithm guarantees to provide good solutions). 2) the cut function does not decompose additively over individual data points. Perhaps surprisingly, GREEDI still performs very well, and significantly outperforms the benchmarks. This suggests that our approach is quite robust, and may be more generally applicable.

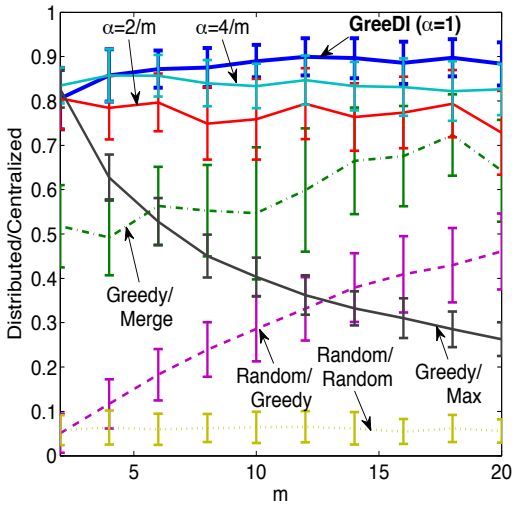


Figure 2: Facebook-like social network

Figure 3: Mean and standard deviation of the ratio of distributed to centralized solution for budget $k = 20$ with varying number of machines m on a *Facebook-like social network*.

Comparision with GREEDY SCALING**.** Kumar et al.[1] recently proposed an alternative approach – GREEDYSCALING – for parallel maximization of submodular functions. GREEDYSCALING is a randomized algorithm that carries out a number (typically less than k) of MapReduce computations. This work was not available at the time of submission of our original manuscript. In the following, we briefly discuss some of the main differences.

First, GREEDYSCALING assumes that the objective function can be evaluated on each machine for any given set. In many realistic scenarios however, the objective function may depend on the entire dataset and different machines may not have access to the full dataset. We explicitly addressed this issue in Section 4.5.

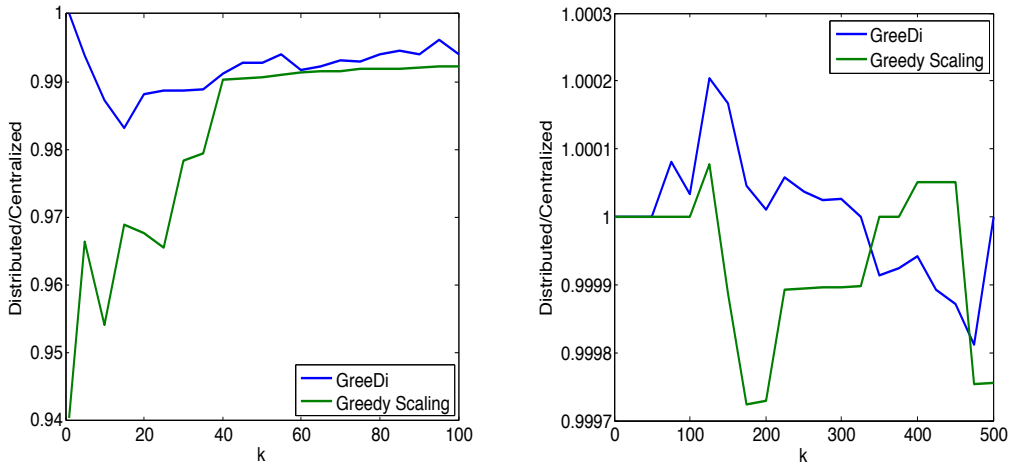
Second, Kumar et al. describe conditions under which GREEDYSCALING achieves close-to-centralized performance. These conditions do not require geometric structure, as we do in our analysis. In contrast to our results, however, it is required that the ratio between the largest and smallest marginal gains of f is bounded by δ, which restricts generality (e.g., for the entropy function, δ can be exponentially small in n).

Last, while our approach only requires two rounds of MapReduce GREEDYSCALING, the number of rounds depend on the quantity δ which may be unbounded.

We applied GREEDI to the submodular coverage problem in which given a collection S of sets, we would like to pick at most k sets from S in order to maximize the size of their union. We compared the performance of our GREEDI algorithm to the reported performance of GREEDYSCALING on the same datasets (*Accidents* and *Kosarak* [2]) used by Kumar et al. As Fig **??** shows, GREEDI significantly outperforms GREEDYSCALING on the "*Accidents*" dataset and its performance is comparable to that of GREEDYSCALING in the "*Kosarak*" dataset.

[1] Kumar, R., Moseley, B., Vassilvitskii, S., & Vattani, A. "Fast greedy algorithms in mapreduce and streaming." Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures, ACM, 2013.

[2] Datasets are available at `http://fimi.ua.ac.be/data/`



(a) Accidents (b) Kosarak

Figure 4: Performance of GREEDI compared to GREEDYSCALING. a) and b) show the ratio of distributed to centralized solution on *Accidents* and *Kosarak* datasets with 340,183 and 990,002 elements, respectively. The results are reported for varying budget k and varying number of machines $m = n/\mu$ where $\mu = O(kn^{\delta} \log n)$ and n is the size of the dataset. The results are reported for $\delta = 1/2$.