2016 IEEE 57th Annual Symposium on Foundations of Computer Science

A New Framework for Distributed Submodular Maximization

Rafael da Ponte Barbosa
Department of Computer Science
University of Warwick
Coventry, UK
rafael@dcs.warwick.ac.uk

Alina Ene
Department of Computer Science
Boston University
Boston, USA
aene@bu.edu

Huy L. Nguyễn
College of Computer and Information Science
Northeastern University
Boston, USA
hlnguyen@cs.princeton.edu

Justin Ward
School of Computer and Communication Sciences
EPFL
Lausanne, Switzerland
justin.ward@epfl.ch

***Abstract*—A wide variety of problems in machine learning, including exemplar clustering, document summarization, and sensor placement, can be cast as constrained submodular maximization problems. A lot of recent effort has been devoted to developing distributed algorithms for these problems. However, these results suffer from high number of rounds, suboptimal approximation ratios, or both. We develop a framework for bringing existing algorithms in the sequential setting to the distributed setting, achieving near optimal approximation ratios for many settings in only a constant number of MapReduce rounds. Our techniques also give a fast sequential algorithm for non-monotone maximization subject to a matroid constraint.**

I. Introduction

The general problem of maximizing a submodular function appears in a variety of contexts, both in theory and practice. From a theoretical perspective, the class of submodular functions is extremely rich, including examples as varied as cut functions of graphs and digraphs, the Shannon entropy function, weighted coverage functions, and log-determinants. Recently, there has been a great deal of interest in practical applications of submodular optimization, as well. Variants of facility location, sampling, sensor selection, clustering, influence maximization in social networks, and welfare maximization problems are all instances of submodular maximization. In practice, many of these applications involve processing enormous datasets requiring *efficient, distributed algorithms*.

In contrast, most successful approaches for submodular maximization have been based on *sequential* greedy algorithms, including the standard greedy algorithm [1], [2], the continuous greedy algorithm [3], [4], and the double greedy algorithm [5]. Indeed, such approaches attain the best-possible, tight approximation guarantees in a variety of settings [6], [7], [8], but unfortunately they all share a common limitation, inherited from the standard greedy algorithm: they are inherently sequential. This presents a seemingly fundamental barrier to obtaining efficient, highly parallel variants of these algorithms.

A. Our Contributions

As demonstrated by the extensive prior works on submodular maximization, the community has a good understanding of the problem under remarkably general types of constraints, which are handled by a small collection of general algorithms. In contrast, the existing works in the distributed setting are either tailored to special cases, giving approximation factors far from optimal or requiring a large number of distributed rounds. One cannot help but wonder if, instead of retracing the individual advances made in the sequential setting over the last few decades, it may be possible to obtain a generic technique to carry over the algorithms in the sequential setting to the parallel world.

In this work, we present a significant step toward resolving the above question. Our main contribution is a *generic parallel algorithm* that allows us to parallelize a broad class of sequential algorithm with almost no loss in performance. The crux of our approach is a *common abstraction* that allows us to capture and parallelize both the standard and continuous greedy algorithms, and it provides a novel unifying perspective for these algorithmic paradigms. Our framework leads to the first distributed algorithms that nearly match the state of the art approximation guarantees for the sequential setting in only a constant number of rounds. In the following, we summarize our main contributions.

A parallel greedy algorithm. We obtain the following general result by parallelizing the standard greedy algorithm:

Theorem V.2. *Let $f : 2^V \to \mathbb{R}_+$ be a submodular function, and $\mathcal{I} \subseteq 2^V$ be a hereditary set system*[1]*. For any $\epsilon > 0$ there is a randomized distributed $O(1/\epsilon)$-round algorithm*

[1]A set system is hereditary if for any $S \in \mathcal{I}$, all subsets of S are also in $\mathcal{I}$.

0272-5428/16 $31.00 © 2016 IEEE
DOI 10.1109/FOCS.2016.74

Table I
NEW RESULTS FOR DISTRIBUTED SUBMODULAR MAXIMIZATION.[3]

Monotone functions			
Constraint	Rounds	Approx.	Citation
cardinality	$O(\frac{\log \Delta}{\epsilon})$	$1-\frac{1}{e}-\epsilon$	[9]
	2	0.545	[10]
	$O(\frac{1}{\epsilon})$	$1-\frac{1}{e}-\epsilon$	Theorem V.2
matroid	$O(\frac{\log \Delta}{\epsilon})$	$\frac{1}{2}-\epsilon$	[9]
	2	$\frac{1}{4}$	[11]
	$O(\frac{1}{\epsilon})$	$1-\frac{1}{e}-\epsilon$	Theorem VI.3
p-system	$O(\frac{\log \Delta}{\epsilon})$	$\frac{1}{p+1}-\epsilon$	[9]
	2	$\frac{1}{2(p+1)}$	[11]
	$O(\frac{1}{\epsilon})$	$\frac{1}{p+1}-\epsilon$	Theorem V.2

Non-monotone functions			
Constraint	Rounds	Approx.	Citation
cardinality	2	$\frac{1-\frac{1}{m}}{2+e}$	[10]
	2	$(1-\frac{1}{m})\frac{1}{e}(1-\frac{1}{e})$	Full version [12]
matroid	2	$\frac{1}{10}$	[11]
	2	$\frac{1-\frac{1}{m}}{2+e}$	Theorem VII.1
	$O(\frac{1}{\epsilon})$	$\frac{1}{e}-\epsilon$	Theorem VI.3
p-system	2	$\frac{1}{2+4(p+1)}$	[11]
	2	$\frac{3(1-\frac{1}{m})}{5p+7+\frac{2}{p}}$	Theorem VII.1

that can be implemented in the MapReduce framework[2]*. The algorithm is an $(\alpha - O(\epsilon))$-approximation with constant probability for the problem $\max_{S\in\mathcal{I}} f(S)$, where α is the approximation ratio of the standard, sequential greedy algorithm for the same problem.*

Our constant number of rounds is a significant improvement over the sample and prune technique of [9], which requires a number of rounds depending logarithmically on the *value* of the single best element. Remarkably, even for the especially simple case of a cardinality constraint, no previous work could get close to the approximation ratio of the simple sequential greedy algorithm in a constant number of rounds. Our framework nearly matches the approximation ratio of greedy in all situations in a constant number of rounds and immediately resolves this problem.

A parallel continuous greedy algorithm. We obtain new distributed approximation results for maximization over matroids, by using a heavily discretized variant of the measured continuous greedy algorithm, obtaining approximation guarantees nearly matching those attained by the continuous greedy in the sequential setting.

[2]We define the MapReduce model in Section II.

[3]Here $\Delta = \max_{i\in V} f(\{i\})$ and m is the number of machines. In the results of [9], in the number of rounds, Δ can be replaced by the maximum size of a solution. All algorithms in previous works and ours are randomized and the approximation guarantees stated hold in expectation, and they can be strengthened to hold with high probability by repeating the algorithms in parallel.

Theorem VI.3. *Let $f : 2^V \to \mathbb{R}_+$ be a submodular function, and $\mathcal{I} \subseteq 2^V$ be a matroid. For any $\epsilon > 0$ there is a randomized distributed $O(1/\epsilon)$-round algorithm that can be implemented in the MapReduce framework. The algorithm is an $(\alpha - O(\epsilon))$-approximation with constant probability for the problem $\max_{S\in\mathcal{I}} f(S)$, where α is $(1-1/e)$ for monotone f and $1/e$ for general f.*

Improved two-round algorithms and fast sequential algorithms. We also give improved two-round approximations for non-monotone submodular maximization under hereditary constraints. We make use of the same "strong greedy property" utilized in [11] but attain approximation guarantees strictly better than were given there. Our algorithm is based on a combination of the standard greedy algorithm Greedy and an additional, arbitrary algorithm Alg. Again, we suppose that f is a (not necessarily monotone) submodular function and $\mathcal{I}$ is any hereditary constraint. In the following theorems and throughout the paper, $n := |V|$ is the size of the ground set, $k := \max_{S\in\mathcal{I}} |S|$ is the maximum size of a solution, and m is the number of machines employed by the distributed algorithm.

Theorem VII.1. *Suppose that* Greedy *satisfies the strong greedy property with constant γ and that* Alg *is a β-approximation for the problem $\max_{S\in\mathcal{I}} f(S)$. Then there is a randomized, two-round distributed algorithm that achieves a $(1-\frac{1}{m})\frac{\beta\gamma}{\beta+\gamma}$ approximation in expectation for $\max_{S\in\mathcal{I}} f(S)$.*

We show that by simulating the machines in this last distributed algorithm, we also obtain a fast, sequential algorithm for maximizing a non-monotone submodular function subject to a matroid constraint. Our algorithm shows that one can preprocess the instance in $O(\frac{n}{\epsilon}\log n)$ time and obtain a set X of size $O(k/\epsilon)$ so that *it suffices to solve the problem on X*. By using a variant of the continuous greedy algorithm on the resulting set X, we obtain the following result.

Theorem VII.2. *There is a sequential, randomized $(\frac{1}{2+e}-\epsilon)$-approximation algorithm for the problem $\max_{S\in\mathcal{I}} f(S)$, where $\mathcal{I}$ is any matroid constraint, running in time $O(\frac{n}{\epsilon}\log n) + \text{poly}(\frac{k}{\epsilon})$.*

B. Techniques

In contrast with the previous framework by [9] which is based on repeatedly eliminating bad elements, our framework is more in line with the greedy approach of identifying good elements. The algorithm maintains a pool of good elements that is grown over several rounds. In each round, the elements are partitioned randomly into groups. Each group selects the best among its elements and the good pool using the sequential algorithm. Finally, the best elements from all groups are added to the good pool. The best solution among the ones found in the execution of the algorithm

is returned at the end. The previous works based on 2 rounds of MapReduce such as [11] can be viewed as a single phase of our algorithm. The first phase can already identify a constant fraction of the weight of the solution, thus obtaining a constant factor approximation. However, it is not clear how to obtain the best approximation factor from such an approach. Our main insight is that, with a right measure of progress, we can grow the solution iteratively and obtain solutions that are arbitrarily close to those of sequential algorithms. We show that after only $O(\frac{1}{\epsilon})$ rounds, the pool of good elements already contains a good solution with constant probability.

C. Related Work

There has been a recent push toward obtaining fast, practical algorithms for submodular maximization problems arising in a variety of applied settings. Research in this direction has yielded a variety of techniques for speeding up the continuous greedy algorithm for monotone maximization [13], [14], as well as new approaches for non-monotone maximization based on insights from both the continuous greedy and double greedy algorithms [15], [16]. Of particular relevance to our results is the case of maximization under a matroid constraint. Here, for monotone functions the fastest current sequential algorithm gives a $1 - 1/e - \epsilon$ approximation using $O(\frac{\sqrt{k}n}{\epsilon^5} \ln^2(\frac{n}{\epsilon}) + \frac{k^2}{\epsilon})$ value queries. For non-monotone functions, Buchbinder *et al.* [16] give an $\frac{1+e^{-2}}{4} > 0.283$-approximation in time $O(kn \log n + Mk)$, where M is the time required to compute a perfect matching on bipartite graph with k vertices per side. They also give a simple, combinatorial $1/4$-approximation in time $O(kn \log n)$. In comparison, the sequential algorithm we present here is faster by a factor of $\Omega(k)$, at the cost of a slightly-weaker $\frac{1}{2+e} > 0.211$-approximation.

Work on parallel and distributed algorithms for submodular maximization has been comparatively limited. Early results considered the special case of maximum k-coverage, and attained an $O(1 - 1/e - \epsilon)$-approximation [17], [18]. Later, Kumar *et al.* [9] considered the more general problem of maximizing an arbitrary monotone submodular function subject to a matroid, knapsack, or p-system constraint. Their approach attains a $\frac{1}{2+\epsilon}$ approximation for matroids, and requires $O(\frac{1}{\epsilon} \log \Delta)$ MapReduce rounds, where Δ is the value of the best single element. More generally, they obtain a $\frac{1}{p+1+\epsilon}$ approximation for p-systems in $O(\frac{1}{\epsilon} \log \Delta)$ rounds. The factor of $\log \Delta$ in the number of rounds is inherent in their approach: they adapt the threshold greedy algorithm, which sequentially picks elements in $\log \Delta$ different thresholds. In another line of work, Mirzasoleiman *et al.* [19] introduced a simple, two-round distributed greedy algorithm for submodular maximization. While their algorithm is only an $O(\frac{1}{m})$-approximation in the worst case, it performs very well in practice, and attains provable constant-factor guarantees for submodular functions exhibiting certain additional structure. Barbosa *et al.* [11] recently gave a more sophisticated analysis of this approach and showed that, if the initial distribution of elements is performed randomly, the algorithm indeed gives an expected, constant-factor guarantee for a variety of problems. Finally, Mirrokni and Zadimoghaddam [10] gave the currently-best 0.545-approximation for the cardinality constraint case using only 2 rounds of MapReduce.

II. The model

We adopt the most stringent MapReduce-style model among [20], [21], [22], [23], the Massively Parallel Communication (MPC) model from [22] as specified by [23]. Let N be the size of the input. In this model, there are M machines each with space S. The total memory of the system is $M \cdot S = O(N)$, which is at most a constant factor more than the input size. Computation proceeds in synchronous rounds. In each round, each machine can perform local computation and at the end, it can send at most a total of $O(S)$ words to other machines. These $O(S)$ words could form a single message of size S, S messages of size 1, or any other combination whose sum is at most $O(S)$. Following [20], we restrict both $M, S < N^{1-\Omega(1)}$. The typical main complexity measure is the number of rounds.

Note that not all previous works on MapReduce-style algorithms for submodular maximization satisfy the strict requirements of the MPC model. For instance, as stated, the previous work by Kumar et al. [9] uses $\Theta(N \log N)$ total memory and thus it does not fit in this model (though it might be possible to modify their algorithms to satisfy this).

We assume that the size of the solution is at most N^{1-2c} for some constant $0 < c < 1/2$. Thus, an entire solution can be stored on a single machine in the model. This assumption is also used in previous work such as [10].

III. Preliminaries

Due to space constraints, we defer some of the proofs to the full version of the paper [12].

A function $f : 2^V \to \mathbb{R}_+$ is submodular if and only if $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ for all $A \subseteq B$ and $e \notin B$. If $f(A \cup \{e\}) - f(A) \geq 0$ for all A and $e \notin A$ we say that f is *monotone*. Here we consider the general problem $\max\{f(S) : S \subseteq V, S \in \mathcal{I}\}$, where $\mathcal{I}$ is any hereditary constraint (i.e., a downward-closed family of subsets of V).

Throughout the paper, $n \coloneqq |V|$ is the size of the ground set, $k \coloneqq \max_{S \in \mathcal{I}} |S|$ is the maximum size of a solution, and m is the number of machines employed by the distributed algorithm.

We shall consider both monotone and non-monotone submodular functions. However, the following simple observation shows that even non-monotone submodular functions are monotone when restricted to the optimal solution of a problem of the sort we consider.

Lemma III.1. *Let f be a submodular function and* $\mathrm{OPT} = \arg\max_{S \in \mathcal{I}} f(S)$ *for some hereditary constraint $\mathcal{I}$. Then, $f(A \cap \mathrm{OPT}) \leq f(B \cap \mathrm{OPT})$ for all $A \subseteq B$.*

In this paper, we work with two standard continuous extensions of submodular functions, the multilinear extension and the Lovász extension. The *multilinear extension* of f is the function $F : [0,1]^V \to \mathbb{R}_+$ such that $F(\mathbf{x}) = \mathbf{E}[f(R(\mathbf{x}))]$, where $R(\mathbf{x})$ is a random subset of V in which each element e appears independently with probability x_e.

The *Lovász extension* of f is the function $f^- : [0,1]^V \to \mathbb{R}_+$ such that $f^-(\mathbf{x}) = \mathbf{E}_{\theta \in \mathcal{U}(0,1)}[f(\{e : x_e \geq \theta\})]$, where $\mathcal{U}(0,1)$ is the uniform distribution on [0,1]. For any submodular function f, the Lovász extension f^- satisfies: $f^-(\mathbf{1}_S) = f(S)$ for all $S \subseteq V$; f^- is convex; and the restricted scale invariance property $f^-(c \cdot \mathbf{x}) \geq c \cdot f^-(\mathbf{x})$ for any $c \in [0,1]$. We shall make use of the following lemmas.

Lemma III.2 ([11], Lemma 1)**.** *Let S be a random set with $\mathbf{E}[\mathbf{1}_S] = c \cdot \mathbf{p}$ (for $c \in [0,1]$). Then, $\mathbf{E}[f(S)] \geq c \cdot f^-(\mathbf{p})$.*

Lemma III.3. *Let $f : 2^V \to \mathbb{R}_+$ be a submodular function that is monotone when restricted to $X \subseteq V$. Further, let $T, S \subseteq X$, and let R be a random subset of T in which every element occurs with probability at least p. Then, $\mathbf{E}[f(R \cup S)] \geq p \cdot f(T \cup S) + (1-p)f(S)$.*

Proof: Recall that f^- is the Lovász extension of f. Since f^- is convex,

$$\begin{aligned}\mathbf{E}[f(R \cup S)] &= \mathbf{E}[f^-(\mathbf{1}_{R \cup S})] \\ &\geq f^-(\mathbf{E}[\mathbf{1}_{R \cup S}]) = f^-(\mathbf{E}[\mathbf{1}_{R \setminus S}] + \mathbf{1}_S).\end{aligned}$$

Since every element of T occurs in R with probability at least p, we have $\mathbf{E}[\mathbf{1}_{R \setminus S}] \geq p \cdot \mathbf{1}_{T \setminus S}$. Then, since f is monotone with respect to $X \supseteq S \cup T$, we must have:

$$f^-(\mathbf{E}[\mathbf{1}_{R \setminus S}] + \mathbf{1}_S) \geq f^-(p \cdot \mathbf{1}_{T \setminus S} + \mathbf{1}_S).$$

Finally, from the definition of f^-, we have

$$f^-(p \cdot \mathbf{1}_{T \setminus S} + \mathbf{1}_S) = p \cdot f(T \cup S) + (1-p)f(S).$$

■

IV. Generic Parallel Algorithm for Submodular Maximization

In this section, we give a generic approach for parallelizing *any sequential algorithm* Alg for the problem $\max_{S \subseteq V \colon S \in \mathcal{I}} f(S)$, where $f : 2^V \to \mathbb{R}_+$ is a submodular function and $\mathcal{I} \subseteq 2^V$ is a hereditary constraint.

As a starting point, we need a common abstract description of existing sequential algorithms. Towards that end, we turn to the standard Greedy and Continuous Greedy algorithms for inspiration. The Greedy algorithm directly constructs a solution, whereas the Continuous Greedy algorithm first constructs a fractional solution $\mathbf{x}$ which is then rounded to get an integral solution. In the common abstraction, we will need both the integral solution *and* the support of the fractional solution $\mathbf{x}$. To account for this, we will have the algorithm Alg return a pair of sets, $(\mathsf{AlgSol}(V), \mathsf{AlgRel}(V))$, where $\mathsf{AlgSol}(V) \in I$ is a feasible solution for the problem and $\mathsf{AlgRel}(V)$ is a set providing additional information. When using the standard Greedy algorithm for Alg, $\mathsf{AlgSol}(V)$ and $\mathsf{AlgRel}(V)$ will both be equal to the Greedy solution. When using the Continuous Greedy algorithm for Alg, $\mathsf{AlgSol}(V)$ will be the integral solution and $\mathsf{AlgRel}(V)$ will be the support of the fractional solution constructed by the Continuous Greedy algorithm.

More importantly, we will need an abstraction that captures the greedy behavior of these algorithms. We encapsulate the crucial properties of greedy-like algorithms in the following definition. We believe that this framework is one of the most valuable and insightful contributions of this work, and it provides a general abstraction for a broader class of algorithms.

We assume that the algorithm Alg satisfies the following properties.

1) (α-Approximation) For every input $N \subseteq V$, $\mathsf{AlgSol}(N)$ is an α-approximate solution to $\max_{S \subseteq N \colon S \in \mathcal{I}} f(S)$.
2) (Consistency) Let A and B be two disjoint subsets of V. Suppose that, for each element $e \in B$, we have $\mathsf{AlgRel}(A \cup \{e\}) = \mathsf{AlgRel}(A)$. Then $\mathsf{AlgSol}(A \cup B) = \mathsf{AlgSol}(A)$.

Armed with this definition, we can now describe our approach for parallelizing an abstract sequential algorithm Alg with almost no loss in the approximation guarantee.

Parallel algorithm ParallelAlg **based on** Alg**.** As before, let α be the approximation guarantee of the sequential algorithm Alg. Let $s \coloneqq \max_{N \subseteq V} |\mathsf{AlgSol}(N) \cup \mathsf{AlgRel}(N)|$ be the maximum size of the sets returned by Alg. Let $\epsilon > 0$ be the desired accuracy, i.e., we will aim that ParallelAlg achieves an $(\alpha - \epsilon)$ approximation.

The algorithm uses $g \coloneqq \Theta(1/(\alpha\epsilon))$ groups of machines with m machines in each group (and thus the total number of machines is gm). The number m of machines can be chosen arbitrarily and it will determine the amount of space needed on each machine, since the dataset is divided roughly equally among each of the m machines in each group. An optimal setting is $gm \coloneqq O(\sqrt{n/s})$.

The algorithms performs $\Theta(1/\epsilon)$ *runs*. Throughout the process, we maintain two quantities: an *incumbent* solution S_{best}, which is the best solution produced on any single machine so far in the process, and a *pool* of elements $C \subseteq V$ (we assume that the incumbent solution is stored on one designated machine).

Each run of the algorithm proceeds as follows. Amongst each group of m machines, we partition V uniformly at random; each element e chooses an index $i \in [m]$ uniformly and independently at random and is assigned to the ith machine in the group. We do this separately for each group

of machines, i.e., each element appears on exactly one machine in each group. For an individual machine $i \in [gm]$, let $X_{i,r}$ denote the set of elements that are assigned to i in run r by this procedure. Additionally, we place on each machine the same pool of elements C_{r-1}, constructed at the end of run $r-1$.

Once the elements have been distributed as described above, on each machine i, we run the algorithm Alg on the input $X_{i,r} \cup C_{r-1}$ on the machine to obtain $(\mathsf{AlgSol}(X_{i,r} \cup C_{r-1}), \mathsf{AlgRel}(X_{i,r} \cup C_{r-1}))$. We update the incumbent solution S_{best} to be the better of the current solution S_{best} and the solutions $\mathsf{AlgSol}(X_{i,r} \cup C_{r-1})$ constructed on each of the machines; this is achieved by having each machine send $\mathsf{AlgSol}(X_{i,r} \cup C_{r-1})$ to some designated machine maintaining S_{best}, and this machine will update S_{best} in the next round. We update the pool by setting $C_r \coloneqq C_{r-1} \bigcup_i \mathsf{AlgRel}(X_{i,r} \cup C_{r-1})$; this is achieved by having each machine send $\mathsf{AlgRel}(X_{i,r} \cup C_{r-1})$ to every other machine, and thus ensuring that the pool C_r is available on each machine during the next round.

At the end of the $\Theta(1/\epsilon)$ runs, the algorithm returns the incumbent solution S_{best}. This completes the description of our algorithm.

Avoiding duplicating the dataset. The algorithm above partitions the dataset over $\Theta(1/\epsilon)$ groups of machines and thus it duplicates the dataset $\Theta(1/\epsilon)$ times (this problem also applies to previous work [10]). This is done in order to achieve the best theoretical guarantee on the number of runs, but in practice it is undesirable to duplicate the data. Instead, we can use a single group of m machines and perform the computation of a single run sequentially over $\Theta(1/\epsilon)$ sub-run, where each sub-run performs the computation of one of the group of machines. This will lead to an algorithm that performs $\Theta(1/\epsilon^2)$ runs using m machines and it does not duplicate the dataset.

The analysis. We devote the rest of this section to the analysis of the algorithm ParallelAlg. We start by noting that, if we choose g and m so that $gm = O(\sqrt{n/s})$, the algorithm uses the following resources and thus it satisfies the requirements of the model in Section II.

Lemma IV.1. ParallelAlg *can be implemented in the parallel model in Section II using the following resources.*

- *The number of rounds is* $O(1/\epsilon)$.
- *The number of machines is* $O(\sqrt{n/s})$.
- *The amount of space used on each machine is* $O(\sqrt{ns}/(\epsilon\alpha))$ *with high probability.*
- *In each round, the total amount of communication from a machine to all other machines is* $O(\sqrt{ns}/(\epsilon\alpha))$ *with high probability. The total amount of communication over all machines in a given round is* $O(n/(\epsilon\alpha))$.

Proof: We will choose $gm \coloneqq \sqrt{n/s}$ as our number of machines. Using this choice, we can provide the guarantees stated in the lemma.

Note that we can combine the update step of the incumbent solution and the pool of a given run with the next run's distribution of elements into a single round of communication. Specifically, each machine computes a new random assignment for each element of its sample $X_{i,r}$, assigns all of its new pool elements to *all* machines, and sends its solution to the designated machine. Thus each run corresponds to a round of communication. In each round, a machine communicates its sample $X_{i,r}$, which has size $O(n/m) = O(\sqrt{ns}/(\epsilon\alpha))$ with high probability, and the sets $\mathsf{AlgSol}(X_{i,r} \cup C_{r-1})$ and $\mathsf{AlgRel}(X_{i,r} \cup C_{r-1})$ that have size $O(s)$ to all other machines. Thus the total amount that a machine communicates is $O(\sqrt{ns}/(\epsilon\alpha) + s \cdot gm) = O(\sqrt{ns}/(\epsilon\alpha))$ with high probability, and the total amount that all machines communicate is $O(n + n/m \cdot gm) = O(n/(\epsilon\alpha))$.

In every round, the space used on a given machine is the size of its sample $X_{i,r}$, which is $O(n/m) = O(\sqrt{ns}/(\epsilon\alpha))$ with high probability; the size of the incumbent solution, which is $O(s)$; and the size of the pool, which is $O(gm \cdot s/\epsilon) = O(\sqrt{ns}/\epsilon)$. Therefore the total amount of space used on each machine is $O(\sqrt{ns}/(\epsilon\alpha))$ with high probability. ■

Thus it remains to analyze the quality of the solution constructed by the algorithm. In the remainder of this section, we show that, if Alg satisfies the α-approximation and consistency properties defined above, the parallel algorithm ParallelAlg achieves an $(\alpha - O(\epsilon))$ approximation. For simplicity, in this section we assume that Alg is deterministic; the extended analysis, in which Alg is randomized, is deferred to the full version of the paper [12].

We start by introducing some notation. Let $\mathcal{V}(1/m)$ denote the distribution over random subsets of V where each element is included independently with probability $1/m$. Let OPT be an optimal solution. Recall that $X_{i,r} \sim \mathcal{V}(1/m)$ is the random sample placed on machine i at the beginning of run r and C_{r-1} is the pool of elements at the beginning of run r. The following theorem is the crux of our analysis.

Theorem IV.2. *Consider a run* $r \geq 1$ *of the algorithm. Let* $\widehat{C}_{r-1} \subseteq V$. *Then one of the following must hold:*

(1) $\mathbf{E}_{X_{1,r}}[f(\mathsf{AlgSol}(C_{r-1} \cup X_{1,r})) \mid C_{r-1} = \widehat{C}_{r-1}] \geq (1-\epsilon)^2\alpha \cdot f(\mathrm{OPT})$, *or*

(2) $\mathbf{E}[f(C_r \cap \mathrm{OPT}) \mid C_{r-1} = \widehat{C}_{r-1}] - f(\widehat{C}_{r-1} \cap \mathrm{OPT}) \geq \frac{\epsilon}{2} \cdot f(\mathrm{OPT})$.

Intuitively, Theorem IV.2 shows that, in expectation, if we have not found a good solution on some machine after $O(1/\epsilon)$ runs, then the current pool C, available to every machine, must satisfy $f(C \cap \mathrm{OPT}) = f(\mathrm{OPT})$, and so each machine in the next run will in fact return a solution of quality at least $\alpha f(\mathrm{OPT})$. The following theorem, whose proof we give in the appendix, makes this formal.

Theorem IV.3. ParallelAlg *achieves an* $(1-\epsilon)^3\alpha$ *approximation with constant probability.*

We devote the rest of this section to the proof of Theorem IV.2. Consider a run r of the algorithm. Let $\widehat{C}_{r-1} \subseteq V$. In the following, we condition on the event that $C_{r-1} = \widehat{C}_{r-1}$.

For each element $e \in V$, let $p_r(e) = \Pr_{X\sim\mathcal{V}(1/m)}[e \in \mathsf{AlgRel}(\widehat{C}_{r-1}\cup X\cup\{e\})]$ if $e \in \mathrm{OPT}\setminus\widehat{C}_{r-1}$, and 0 otherwise. As shown in the following lemma, the probability $p_r(e)$ gives us a handle on the probability that e is in the union of the relevant sets.

Lemma IV.4. *For each element* $e \in \mathrm{OPT}\setminus\widehat{C}_{r-1}$,

$$\Pr[e \in \cup_{1\le i\le gm}\mathsf{AlgRel}(\widehat{C}_{r-1}\cup X_{i,r})] = 1-(1-p_r(e))^g,$$

where g *is the number of groups into which the machines are partitioned.*

Proof sketch: For each group G_j, we can show that e is *not* in the union of the relevant sets for that group with probability $1-p_r(e)$. Since different groups have independent partitions, e is not in the union of the relevant sets for all machines with probability $(1-p_r(e))^g$, and the lemma follows. ∎

Returning to the proof of Theorem IV.2, we define a partition (P_r, Q_r) of $\mathrm{OPT}\setminus\widehat{C}_{r-1}$ as follows: $P_r = \{e \in \mathrm{OPT}\setminus\widehat{C}_{r-1}\colon p_r(e) < \epsilon\}$ and $Q_r = \{e \in \mathrm{OPT}\setminus\widehat{C}_{r-1}\colon p_r(e) \ge \epsilon\}$.

The following subsets of P_r and Q_r are key to our analysis (recall that $X_{i,r}$ is the random sample placed on machine i at the beginning of the run r): $P'_r = \{e \in P_r\colon e \notin \mathsf{AlgRel}(\widehat{C}_{r-1}\cup X_{1,r}\cup\{e\})\}$ and $Q'_r = Q_r \cap (\cup_{i=1}^{gm}\mathsf{AlgRel}(\widehat{C}_{r-1}\cup X_{i,r}))$.

Note that each element $e \in P_r$ is in P'_r with probability $1-p_r(e) \ge 1-\epsilon$. Further, by Lemma IV.4, each element $e \in Q_r$ is in Q'_r with probability $1-(1-p_r(e))^g \ge 1-\frac{1}{e} \ge \frac{1}{2}$.

It follows from the definition of P'_r and the consistency property of Alg that

$$\mathsf{AlgSol}(\widehat{C}_{r-1}\cup X_{1,r}) = \mathsf{AlgSol}(\widehat{C}_{r-1}\cup X_{1,r}\cup P'_r).$$

Let $\mathrm{OPT}_{r-1} = \widehat{C}_{r-1}\cap\mathrm{OPT}$ be the part of OPT in this iteration's pool. Then, since Alg is an α approximation and $P'_r\cup\mathrm{OPT}_{r-1} \subseteq \mathrm{OPT}$ is a feasible solution, we have

$$f(\mathsf{AlgSol}(\widehat{C}_{r-1}\cup X_{1,r})) \ge \alpha\cdot f(P'_r\cup\mathrm{OPT}_{r-1}).$$

Taking expectations on both sides, we have:

$$\begin{aligned}&\mathbf{E}[f(\mathsf{AlgSol}(\widehat{C}_{r-1}\cup X_{1,r}))]\\ &\quad\ge \alpha\cdot\mathbf{E}[f(P'_r\cup\mathrm{OPT}_{r-1})]\\ &\quad\ge (1-\epsilon)\alpha\cdot f(P_r\cup\mathrm{OPT}_{r-1}),\end{aligned}\tag{1}$$

where the final inequality follows from Lemma III.3, since f is monotone when restricted to $\mathrm{OPT}\supseteq P_r\cup\mathrm{OPT}_{r-1}$, and P'_r contains every element of P_r with probability at least $(1-\epsilon)$.

Note that $Q'_r \subseteq (\mathrm{OPT}\cap C_r)\setminus\mathrm{OPT}_{r-1}$. As before, f is monotone when restricted to OPT. Additionally, Q'_r contains every element of Q_r with probability at least $1/2$. Thus,

$$\begin{aligned}&\mathbf{E}[f(C_r\cap\mathrm{OPT})\mid C_{r-1}=\widehat{C}_{r-1}]\\ &\quad\ge\mathbf{E}[f(Q'_r\cup\mathrm{OPT}_{r-1})]\\ &\quad\ge\frac{1}{2}\cdot f(Q_r\cup\mathrm{OPT}_{r-1})+\frac{1}{2}\cdot f(\mathrm{OPT}_{r-1}),\end{aligned}$$

where the final inequality follows from Lemma III.3. Rearranging this inequality using the condition $C_{r-1}=\widehat{C}_{r-1}$ and the definition $\mathrm{OPT}_{r-1}=\widehat{C}_{r-1}\cap\mathrm{OPT}$ we obtain:

$$\begin{aligned}&\mathbf{E}[f(C_r\cap\mathrm{OPT})-f(C_{r-1}\cap\mathrm{OPT})\mid C_{r-1}=\widehat{C}_{r-1}]\\ &\ge\frac{1}{2}\left(f(Q_r\cup\mathrm{OPT}_{r-1})-f(\mathrm{OPT}_{r-1})\right)\\ &\ge\frac{1}{2}\left(f(P_r\cup Q_r\cup\mathrm{OPT}_{r-1})-f(P_r\cup\mathrm{OPT}_{r-1})\right)\\ &=\frac{1}{2}\left(f(\mathrm{OPT})-f(P_r\cup\mathrm{OPT}_{r-1})\right),\end{aligned}\tag{2}$$

where the second inequality follows from submodularity.

Now, if $f(P_r\cup(\widehat{C}_{r-1}\cap\mathrm{OPT}))\ge(1-\epsilon)\cdot f(\mathrm{OPT})$ then this fact together with (1) imply that the first property in the statement of Theorem IV.2 must hold. Otherwise, $f(\mathrm{OPT})-f(P_r\cup(\widehat{C}_{r-1}\cap\mathrm{OPT}))\ge\epsilon\cdot f(\mathrm{OPT})$; this fact together with (2) implies that the second property must hold.

This completes the description of our generic approach. In the following sections, we instantiate the algorithm Alg with the standard Greedy algorithm and a heavily discretized Continuous Greedy algorithm, and obtain our main results stated in the introduction.

V. A Parallel Greedy Algorithm

In this section, we combine the generic approach from Section IV with the standard greedy algorithm, and give our results for monotone maximization stated in Theorem V.2.

We let Alg be the standard Greedy algorithm. We let $\mathsf{AlgRel}(N) = \mathsf{AlgSol}(N) = \mathsf{Greedy}(N)$. It was shown in previous work that the Greedy algorithm satisfies the consistency property.

Lemma V.1 ([11], Lemma 2)**.** *Let* $A\subseteq V$ *and* $B\subseteq V$ *be two disjoint subsets of* V*. Suppose that, for each element* $e\in B$*, we have* $\mathsf{Greedy}(A\cup\{e\}) = \mathsf{Greedy}(A)$*. Then* $\mathsf{Greedy}(A\cup B) = \mathsf{Greedy}(A)$.

Informally, this simply means that if Greedy rejects some element e when presented with input $A\cup\{e\}$, then adding other similarly rejected elements to $A\cup\{e\}$ cannot cause e to be accepted. This allows us to immediately apply the result from Section IV and obtain the following result.

DCGreedy: Discretized continuous greedy
Input: $N \subseteq V$

1: $\mathbf{x}(0) \leftarrow \mathbf{0}$
2: **if** $t \leftarrow 1$ to $1/\epsilon$ **then**
3: $\quad \mathbf{y}(t) \leftarrow \mathsf{GreedyStep}(N, \mathbf{x}(t))$
4: $\quad \mathbf{x}(t) \leftarrow \mathbf{x}(t-1) + \mathbf{y}(t)$
5: $S \leftarrow \mathsf{SwapRounding}(\mathbf{x}(1/\epsilon), \mathcal{I})$
6: Let T be the support of $\mathbf{x}(1/\epsilon)$
7: **return** (S, T)

Figure 1. Discretized continuous greedy algorithm.

GreedyStep: Greedy Update Step
Input: $N \subseteq V$, $\mathbf{x} \in [0,1]^N$

1: $W \leftarrow \emptyset$, $\mathbf{y} \leftarrow \mathbf{0}$
2: **repeat**
3: $\quad D \leftarrow \{e \in N \setminus W : W \cup \{e\} \in \mathcal{I}\}$
4: $\quad$ **for all** $e \in D$ **do**
5: $\quad\quad w_e \leftarrow \mathbf{E}[f(R(\mathbf{x}+\mathbf{y}) \cup \{e\}) - f(R(\mathbf{x}+\mathbf{y}))]$
6: $\quad$ Let $e^* = \arg\max_{e \in D} w_e$
7: $\quad$ **if** $D = \emptyset$ or $w_{e^*} < 0$ **then**
8: $\quad\quad$ **return** $\mathbf{y}$
9: $\quad$ **else** $y_{e^*} \leftarrow y_{e^*} + \epsilon(1 - x_{e^*})$
10: $\quad\quad W \leftarrow W \cup \{e^*\}$
11: **until** $D = \emptyset$ or $w_{e^*} < 0$

Figure 2. Greedy Update Step. On line 5, for a vector $\mathbf{z} \in [0,1]^N$, we use $R(\mathbf{z})$ to denote a random subset of N that contains each element e independently with probability z_e. The weights on line 5 cannot be computed exactly in polynomial time, but they can be efficiently approximated using random samples.

Theorem V.2. *Let $f : 2^V \to \mathbb{R}_+$ be a submodular function, and $\mathcal{I} \subseteq 2^V$ be a hereditary set system. For any $\epsilon > 0$ there is a randomized distributed $O(1/\epsilon)$-round algorithm that can be implemented in the model described in Section II. The algorithm is an $(\alpha - O(\epsilon))$-approximation with constant probability for the problem $\max_{S \in \mathcal{I}} f(S)$, where α is the approximation ratio of the standard, sequential greedy algorithm for the same problem.*

VI. A Parallel Continuous Greedy Algorithm

For monotone maximization subject to a matroid constraint, Theorem V.2 guarantees only a $(1/2 - \epsilon)$ approximation, due to the limitations of the standard greedy algorithm. We obtain a nearly optimal $(1 - 1/e - \epsilon)$ approximation by instantiating the framework in Section IV with the DCGreedy algorithm shown in Figure 1. The DCGreedy algorithm is a heavily discretized version of the measured continuous greedy approach of [4], and it first constructs an approximate fractional solution to the problem $\max_{\mathbf{x} \in P(\mathcal{I})} F(\mathbf{x})$ of maximizing the multilinear extension F of f subject to the constraint that $\mathbf{x}$ is in the matroid polytope $P(\mathcal{I})$, and then rounds the fractional solution without loss using pipage rounding or swap rounding [24], [25].

In this section, we combine the generic approach from Section IV with the DCGreedy algorithm. We use DCGreedy as Alg; the relevant set $\mathsf{AlgRel}(N)$ is the set of elements in the support of the fractional solution $\mathbf{x}(1/\epsilon)$, and $\mathsf{AlgSol}(N)$ is the integral solution obtained by rounding $\mathbf{x}(1/\epsilon)$.

Note that it is necessary to ensure that the fractional solution has small support so that the size of $\mathsf{AlgRel}(N)$ is small. We achieve this by heavily discretizing the continuous greedy algorithm, thereby limiting the number of support updates performed in lines 3 and 4 of DCGreedy. Unfortunately, performing this discretization naively introduces an error in the approximation that is too large. Thus, we make use of a key idea from [13], which can be applied in the case of a matroid constraint. This allows us to show the following lemma whose proof is deferred to the full version of the paper [12].

Lemma VI.1. *The* DCGreedy *algorithm achieves an $(1 - 1/e - O(\epsilon))$ approximation for monotone functions and an $(1/e - O(\epsilon))$ approximation for non-monotone functions.*

The lemma above provides us with the desired approximation guarantees for DCGreedy, and thus it remains to show the consistency property. Before doing so, we must address how the weights are computed on line 5 of the GreedyStep algorithm (see Figure 2). Computing the weights exactly requires exponential time, but they can be approximated in polynomial time using random samples. In this version of the paper, *we assume that the weights are computed exactly*, since this will keep the algorithm deterministic. In the full version of the paper, we remove this assumption and we analyze the resulting randomized algorithm using an extension of our framework.

Lemma VI.2. *Let A and B be two disjoint subsets of V. Suppose that, for each element $e \in B$, we have $\mathsf{DCGreedyRel}(A \cup \{e\}) = \mathsf{DCGreedyRel}(A)$. Then $\mathsf{DCGreedySol}(A \cup B) = \mathsf{DCGreedySol}(A)$.*

Proof: We will show that the GreedyStep algorithm picks the same set W on input $(A, \mathbf{x})$ and $(A \cup B, \mathbf{x})$, which implies the lemma. Suppose for contradiction that the algorithm makes different choices on input $(A, \mathbf{x})$ and $(A \cup B, \mathbf{x})$. Consider the first iteration where the two runs differ, and let e be the element added to W in that iteration on input $(A \cup B, \mathbf{x})$. Note that $e \notin A$ and thus we have $e \in B$. But then e will be added to W on input $(A \cup \{e\}, \mathbf{x})$. Thus $e \in \mathsf{DCGreedyRel}(A \cup \{e\})$, which contradicts the fact that $e \in B$. ∎

Thus we can apply the result from Section IV and obtain the following result.

Theorem VI.3. *Let $f : 2^V \to \mathbb{R}_+$ be a submodular*

function, and $\mathcal{I} \subseteq 2^V$ be a matroid. For any $\epsilon > 0$ there is a randomized distributed $O(1/\epsilon)$-round algorithm that can be implemented in the model described in Section II. The algorithm is an $(\alpha - O(\epsilon))$-approximation with constant probability for the problem $\max_{S \in \mathcal{I}} f(S)$, where α is $(1 - 1/e)$ for monotone f and $1/e$ for general f.

VII. Faster Algorithms

In this section, we build on the techniques from the previous sections to give a distributed algorithm for non-monotone maximization that requires only two rounds, rather than $O(1/\epsilon)$ rounds, and achieves an improved approximation guarantee over the two-round algorithm proposed in [11]. In the case of non-monotone maximization over a matroid, we show that our techniques can be used to obtain a new, fast *sequential* algorithm as well.

A. Two-Round Algorithms For Non-Monotone Maximization

We first give an improved two-round algorithm for non-monotone maximization subject to a any hereditary constraint. The algorithm is similar to that of [11] for *monotone* maximization; perhaps surprisingly, we show that this approach achieves a good approximation even for non-monotone functions. We randomly partition the elements onto the m machines, and run Greedy on the elements V_i on machine i to pick a set S_i. We place the sets S_i on a single machine and we run any algorithm Alg on $B \coloneqq \bigcup_i S_i$ to find a solution T. We return the best solution amongst $S_1, \ldots, S_m, T$.

We analyze the algorithm for any hereditary constraint $\mathcal{I}$ for which the Greedy algorithm satisfies the following property (for some γ), which we refer to as the strong greedy property:

$$\forall S \in \mathcal{I} : f(\mathsf{Greedy}(V)) \geq \gamma \cdot f(\mathsf{Greedy}(V) \cup S) \quad \text{(GP)}$$

By the standard Greedy analysis, we have $\gamma = 1/2$ for a matroid constraint and $\gamma = 1/(p+1)$ for a p-system constraint.

Theorem VII.1. *Suppose that* Greedy *satisfies the strong greedy property with constant γ and let* Alg *be any β-approximation for the problem $\max_{S \in \mathcal{I}} f(S)$. Then there is a randomized, two-round distributed algorithm that achieves a $(1 - \frac{1}{m})\frac{\beta\gamma}{\beta+\gamma}$ approximation in expectation for $\max_{S \in \mathcal{I}} f(S)$.*

Proof: For each element e, we let probability $p_e = \Pr_{X \sim \mathcal{V}(1/m)}[e \in \mathsf{Greedy}(X \cup \{e\})]$, if $e \in \mathrm{OPT}$, and 0 otherwise. Then, let $\mathbf{p} \in [0,1]^V$ denote the vector whose entries are given by the probabilities p_e.

We first analyze the expected value of the Greedy solution S_1. Let

$$O = \{e \in \mathrm{OPT} \colon e \notin \mathsf{Greedy}(V_1 \cup \{e\})\}.$$

By Lemma V.1, $\mathsf{Greedy}(V_1 \cup O) = \mathsf{Greedy}(V_1) = S_1$, and by (GP), $f(S_1) \geq \gamma \cdot f(S_1 \cup O)$. Therefore

$$\begin{aligned}
\mathbf{E}[f(S_1)] &\geq \gamma \cdot \mathbf{E}[f(S_1 \cup O)] \\
&= \gamma \cdot \mathbf{E}[f^-(\mathbf{1}_{S_1 \cup O})] \\
&\geq \gamma \cdot f^-(\mathbf{E}[\mathbf{1}_{S_1 \cup O}]) \\
&= \gamma \cdot f^-(\mathbf{E}[\mathbf{1}_{S_1}] + (\mathbf{1}_{\mathrm{OPT}} - \mathbf{p})). \qquad (3)
\end{aligned}$$

On line three, we have used the fact that f^- is convex and on line four we have used the fact that $\mathbf{E}[\mathbf{1}_{S_1 \cup O}] = \mathbf{E}[\mathbf{1}_{S_1}] + (\mathbf{1}_{\mathrm{OPT}} - \mathbf{p})$.

Now consider the solution T. Since Alg is a β-approximation, we have

$$\begin{aligned}
\mathbf{E}[f(T)] &\geq \beta \cdot \mathbf{E}[f(B \cap \mathrm{OPT})] \\
&= \beta \cdot \mathbf{E}[f^-(\mathbf{1}_{B \cap \mathrm{OPT}})] \\
&\geq \beta \cdot f^-(\mathbf{E}[\mathbf{1}_{B \cap \mathrm{OPT}}]) \\
&= \beta \cdot f^-(\mathbf{p}). \qquad (4)
\end{aligned}$$

Similarly to above, we have used the convexity of f^- and the fact that $\mathbf{E}[\mathbf{1}_{B \cap \mathrm{OPT}}] = \mathbf{p}$.

By combining (3) and (4), and using convexity of f^-, we obtain

$$\begin{aligned}
&\frac{1}{\gamma}\,\mathbf{E}[f(S_1)] + \frac{1}{\beta}\,\mathbf{E}[f(T)] \\
&\quad \geq f^-(\mathbf{E}[\mathbf{1}_{S_1}] + (\mathbf{1}_{\mathrm{OPT}} - \mathbf{p})) + f^-(\mathbf{p}) \\
&\quad \geq 2 \cdot f^-\left(\frac{\mathbf{E}[\mathbf{1}_{S_1}] + \mathbf{1}_{\mathrm{OPT}}}{2}\right).
\end{aligned}$$

Since $S_1 \subseteq V_1$ and V_1 is a $1/m$ sample of V, we have $\mathbf{E}[\mathbf{1}_{S_1}] \leq \frac{1}{m} \cdot \mathbf{1}_V$. Therefore, using the definition of f^- and the non-negativity of f, we obtain $2 \cdot f^-\left(\frac{\mathbf{E}[\mathbf{1}_{S_1}] + \mathbf{1}_{\mathrm{OPT}}}{2}\right) \geq \left(1 - \frac{1}{m}\right) f(\mathrm{OPT})$. Thus $\max\{\mathbf{E}[f(S_1)], \mathbf{E}[f(T)]\} \geq \left(1 - \frac{1}{m}\right) \frac{\beta\gamma}{\beta+\gamma} \cdot f(\mathrm{OPT})$. ∎

Examples of results. We conclude this section with some examples of approximation guarantees that we can obtain using Theorem VII.1. For a matroid constraint, we have $\gamma = 1/2$ and, if we use the measured Continuous Greedy algorithm for Alg, we have $\beta = 1/e$; thus we obtain a $\left(1 - \frac{1}{m}\right)\frac{1}{2+e}$ approximation. We remark that, for a cardinality constraint, one can strengthen the proof of Theorem VII.1 slightly and obtain a $\left(1 - \frac{1}{m}\right)\frac{1}{e}\left(1 - \frac{1}{e}\right)$ approximation; we defer the details to the full version of the paper [12].

For a p-system constraint, we have $\gamma = \frac{1}{p+1}$. We can use the algorithm of Gupta *et al.* [26] for Alg that achieves an approximation $\beta = 3/\left(2p + 4 + \frac{2}{p}\right)$ when combined with the algorithm of [5] for unconstrained non-monotone maximization. Thus we obtain a $3\left(1 - \frac{1}{m}\right)/\left(5p + 7 + \frac{2}{p}\right)$ approximation.

B. A Fast Sequential Algorithm for Matroid Constraints

We now show how our approach can be used to obtain a fast *sequential* algorithm for non-monotone maximization

DThreshGreedy: Descending Thresholds Greedy
Input: $N \subseteq V$

$S \leftarrow \emptyset$, $d \leftarrow \max_{e \in N} f(\{e\})$
for $w = d$; $w \geq \frac{\epsilon}{n} d$; $w \leftarrow w(1-\epsilon)$ **do**
  for all $e \in N$ **do**
    if $S \cup \{e\} \in \mathcal{I}$ and $f(S \cup \{e\}) - f(S) \geq w$ **then**
      return $S \leftarrow S \cup \{e\}$
return S

Figure 3. Descending Thresholds Greedy algorithm [9], [13].

subject to a matroid constraint. The analysis given in Theorem VII.1 only relies on the following two properties of the Greedy algorithm: it satisfies (GP) and Lemma V.1. Thus we can replace the Greedy algorithm by any algorithm satisfying these two properties. In particular, the Descending Thresholds Greedy (shown in Figure 3) of [9], [13] satisfies these conditions with $\gamma = 1/2 - \epsilon$.

Our algorithm proceeds as follows. We randomly partition the elements into $m := 1/\epsilon$ samples $V_1, V_2, \ldots, V_m$. On each sample, we run the Descending Thresholds Greedy algorithm on V_i to obtain a solution S_i. Let $A := \operatorname{argmax}_{i \in [m]} f(S_i)$ and $B := \bigcup_i S_i$. Then, $|B| \leq k/\epsilon$, where k is the rank of the matroid. We run any β-approximation algorithm Alg on B to find a solution B', and we return the better of A and B'. We obtain the following result.

Theorem VII.2. *There is a sequential, randomized* $(\frac{1}{2+e} - \epsilon)$*-approximation algorithm for the problem* $\max_{S \in \mathcal{I}} f(S)$*, where* $\mathcal{I}$ *is any matroid constraint, running in time* $O(\frac{n}{\epsilon} \log n) + \mathrm{poly}(\frac{k}{\epsilon})$.

Proof: The running time of the Descending Thresholds Greedy algorithm on a ground set of size s is $O(\frac{s}{\epsilon} \log(\frac{s}{\epsilon}))$. Each random sample has size $O(\epsilon n)$ with high probability, and thus the total time needed to construct B is $O(\frac{n}{\epsilon} \log n)$ with high probability. It follows from the analysis in Theorem VII.1 that the best of the two solutions A and a β-approximation to $\max_{S \subseteq B : S \in \mathcal{I}} f(S)$ is a $\frac{1}{2+\frac{1}{\beta}} - \epsilon$ approximation. We can then use any $1/e$-approximation algorithm as Alg. ∎

ACKNOWLEDGMENT

This work was done in part while A.E. was with the Computer Science department at the University of Warwick and a visitor to the Toyota Technological Institute at Chicago, H.N. was with the Toyota Technological Institute at Chicago, and J.W. was with the Computer Science department at the University of Warwick. J.W. was supported by EPSRC grant EP/J021814/1 and ERC Starting Grant 335288-OptApprox.

REFERENCES

[1] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—i," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978.

[2] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions—ii," *Mathematical Programming Studies*, vol. 8, pp. 73–87, 1978.

[3] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a submodular set function subject to a matroid constraint," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1740–1766, 2011.

[4] M. Feldman, J. S. Naor, and R. Schwartz, "A unified continuous greedy algorithm for submodular maximization," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2011, pp. 570–579.

[5] N. Buchbinder, M. Feldman, J. S. Naor, and R. Schwartz, "A tight linear time (1/2)-approximation for unconstrained submodular maximization." in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2012, pp. 1384–1402.

[6] U. Feige, "A threshold of ln n for approximating set cover," *Journal of the ACM (JACM)*, vol. 45, no. 4, pp. 634–652, 1998.

[7] J. Vondrák, "Symmetry and approximability of submodular maximization problems," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2009, pp. 651–670.

[8] S. Dobzinski and J. Vondrák, "From query complexity to computational complexity," in *ACM Symposium on Theory of Computing (STOC)*, 2012, pp. 1107–1116.

[9] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani, "Fast greedy algorithms in mapreduce and streaming," in *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2013, pp. 1–10.

[10] V. S. Mirrokni and M. Zadimoghaddam, "Randomized composable core-sets for distributed submodular maximization," in *ACM Symposium on Theory of Computing (STOC)*, 2015, pp. 153–162.

[11] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward, "The power of randomization: Distributed submodular maximization on massive datasets," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1236–1244.

[12] ——, "A new framework for distributed submodular maximization," *arXiv preprint at http://arxiv.org/abs/1507.03719*, 2016.

[13] A. Badanidiyuru and J. Vondrák, "Fast algorithms for maximizing submodular functions," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014, pp. 1497–1514.

[14] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause, "Lazier than lazy greedy," in *AAAI Conference on Artificial Intelligence*, 2015, pp. 1812–1818.

[15] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz, "Submodular maximization with cardinality constraints," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014, pp. 1433–1452.

[16] N. Buchbinder, M. Feldman, and R. Schwartz, "Comparing apples and oranges: Query tradeoff in submodular maximization," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015, pp. 1149–1168.

[17] F. Chierichetti, R. Kumar, and A. Tomkins, "Max-cover in map-reduce," in *International World Wide Web Conference (WWW)*, 2010, pp. 231–240.

[18] G. E. Blelloch, R. Peng, and K. Tangwongsan, "Linear-work greedy parallel approximate set cover and variants," in *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2011, pp. 23–32.

[19] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause, "Distributed submodular maximization: Identifying representative elements in massive data," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 2049–2057.

[20] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for mapreduce," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, pp. 938–948.

[21] M. T. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the mapreduce framework," in *International Symposium on Algorithms and Computation*, 2011, pp. 374–383.

[22] P. Beame, P. Koutris, and D. Suciu, "Communication steps for parallel query processing," in *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, 2013, pp. 273–284.

[23] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev, "Parallel algorithms for geometric graph problems," in *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, 2014, pp. 574–583.

[24] A. Ageev and M. Sviridenko, "Pipage rounding: A new method of constructing algorithms with proven performance guarantee," *Journal of Combinatorial Optimization*, vol. 8, no. 3, pp. 307–328, 2004.

[25] C. Chekuri, J. Vondrák, and R. Zenklusen, "Dependent randomized rounding via exchange properties of combinatorial structures," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010, pp. 575–584.

[26] A. Gupta, A. Roth, G. Schoenebeck, and K. Talwar, "Constrained non-monotone submodular maximization: Offline and secretary algorithms," in *Conference on Web and Internet Economics (WINE)*, 2010, pp. 246–257.

APPENDIX

Theorem IV.3. ParallelAlg *achieves an* $(1-\epsilon)^3\alpha$ *approximation with constant probability.*

Proof: Let $R = c/\epsilon$ be the total number of runs, and $\mathcal{C} = (C_0, C_1, \ldots, C_R)$. Let $I_r(C_{r-1}) \in \{0,1\}$ be equal to 1 if and only if

$$\mathbf{E}_{X_{1,r}}[f(\mathsf{AlgSol}(C_{r-1} \cup X_{1,r}))] \geq (1-\epsilon)^2\alpha \cdot f(\mathrm{OPT}).$$

Let

$$\Phi_r(\mathcal{C}) = I_r(C_{r-1}) + \frac{2(f(C_r \cap \mathrm{OPT}) - f(C_{r-1} \cap \mathrm{OPT}))}{\epsilon f(\mathrm{OPT})}$$

$$\begin{aligned}\Phi(\mathcal{C}) &= \sum_{r=1}^{R} \Phi_r(C_{r-1}) \leq \sum_{r=1}^{R} I_r(C_{r-1}) + \frac{2f(C_R \cap \mathrm{OPT})}{\epsilon f(\mathrm{OPT})} \\ &\leq \sum_{r=1}^{R} I_r(C_{r-1}) + \frac{2}{\epsilon}\end{aligned}$$

Taking expectation over the random choices of $\mathcal{C}$, we have

$$\mathbf{E}_{\mathcal{C}}[\Phi(\mathcal{C})] \leq \sum_{r=1}^{R} \mathbf{E}[I_r(C_{r-1})] + \frac{2}{\epsilon}$$

On the other hand, by Theorem IV.2, $\mathbf{E}[\Phi_r(C_{r-1})] \geq 1$ and therefore $\mathbf{E}[\Phi(\mathcal{C})] \geq R$. Thus

$$\frac{2}{\epsilon} + \sum_{r=1}^{R} \mathbf{E}[I_r(C_{r-1})] \geq \Phi(\mathcal{C}) \geq R.$$

Since $R > 6/\epsilon$, we have

$$\sum_{r=1}^{R} \mathbf{E}[I_r(\widehat{C}_{r-1})] \geq \frac{2R}{3}.$$

Therefore, with probability at least $2/3$, there exists a run r such that $I_r(C_{r-1}) = 1$. Fix the randomness up to the first such run, i.e., condition on a fixed $C_{r-1} = \widehat{C}_{r-1}$ such that $I_r(\widehat{C}_{r-1}) = 1$ and $C_r, \ldots, C_R$ remain random. Assume for contradiction that with probability at least $1 - \epsilon\alpha(1-\epsilon)^2$ over the choices of $X_{1,r}$,

$$f(\mathsf{AlgSol}(C_{r-1} \cup X_{1,r})) < (1-\epsilon)^3\alpha \cdot f(\mathrm{OPT}).$$

Then we have

$$\begin{aligned}&\mathbf{E}[f(\mathsf{AlgSol}(C_{r-1} \cup X_{1,r}))] \\ &\quad< (\epsilon\alpha(1-\epsilon)^2 + (1-\epsilon\alpha(1-\epsilon)^2)(1-\epsilon)^3\alpha) f(\mathrm{OPT}) \\ &\quad= \left(\epsilon + (1-\epsilon\alpha(1-\epsilon)^2)(1-\epsilon)\right)(1-\epsilon)^2\alpha f(\mathrm{OPT}) \\ &\quad< (1-\epsilon)^2\alpha f(\mathrm{OPT}),\end{aligned}$$

contradicting our assumption on C_{r-1}. Thus, with probability at least $\epsilon\alpha(1-\epsilon)^2$, we have

$$f(\mathsf{AlgSol}(C_{r-1} \cup X_{1,r})) \geq (1-\epsilon)^3\alpha \cdot f(\mathrm{OPT}).$$

Notice that the above argument applies not only to machine 1 in run r but also the first machine in each of the g groups in the same run r and their random samples $X_{i,r}$ are independent. Thus, since $g \geq c/(\epsilon\alpha)$ for a sufficiently large constant c, with probability at least $5/6$, we have $\max_i f(\mathsf{AlgSol}(C_{r-1} \cup X_{i,r})) \geq (1-\epsilon)^3\alpha \cdot f(\mathrm{OPT})$. Overall, the algorithm succeeds with probability at least $2/3 \cdot 5/6 = 5/9$. ∎