Submodular Optimization in the MapReduce Model

Paul Liu and Jan Vondrak
Stanford University, USA
{paulliu, jvondrak}@stanford.edu

Abstract

Submodular optimization has received significant attention in both practice and theory, as a wide array of problems in machine learning, auction theory, and combinatorial optimization have submodular structure. In practice, these problems often involve large amounts of data, and must be solved in a distributed way. One popular framework for running such distributed algorithms is MapReduce. In this paper, we present two simple algorithms for cardinality constrained submodular optimization in the MapReduce model: the first is a $(1/2 - o(1))$-approximation in 2 MapReduce rounds, and the second is a $(1 - 1/e - \epsilon)$-approximation in $\frac{1+o(1)}{\epsilon}$ MapReduce rounds.

2012 ACM Subject Classification Theory of computation → MapReduce algorithms; Distributed computing models; Algorithm design techniques; Submodular optimization and polymatroids

Keywords and phrases mapreduce, submodular, optimization, approximation algorithms

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.18

1 Introduction

Let $f : 2^V \to \mathbb{R}^+$ be a function satisfying $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ for all $A \subseteq B$ and $e \notin B$. Such a function is called *submodular*. When f satisfies the additional property $f(A \cup \{e\}) - f(A) \geq 0$ for all A and $e \notin A$, we say f is *monotone*.

Many combinatorial optimization problems can be cast as submodular optimization problems. Such problems include classics such as max cut, min cut, maximum coverage, and minimum spanning tree [6]. Although submodular optimization encompasses several NP-Hard problems, well-known greedy approximation algorithms are known [11]. We focus on the special case of monotone submodular maximization under a cardinality constraint k, i.e.

$$OPT := \max_{S \subseteq V, |S| \leq k} f(S), \ f \text{ is monotone.}$$

In particular, it is known that one can approximate a cardinality constrained monotone submodular maximization problem to a factor of $1 - 1/e$ of optimal.

Due to rapidly growing datasets, recent focus has been on submodular optimization in distributed models [1, 3, 4, 5, 8, 10]. In this work, we focus on the MapReduce model, where complexity is measured as the number of synchronous communication rounds between the machines involved. The current state of the art for cardinality constrained submodular maximization is the algorithm of Barbosa et al. [5], which achieves a $1/2 - \epsilon$ approximation in 2 rounds and was the first to achieve a $1 - 1/e - \epsilon$ approximation in $O\left(\frac{1}{\epsilon}\right)$ rounds. Both algorithms actually require significant duplication of the ground set (each element being sent to $\Omega(\frac{1}{\epsilon})$ machines). Since this might be an issue in practice, [5] mentions that without duplication, the two algorithms could be implemented in $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ and $O(\frac{1}{\epsilon^2})$ rounds,

respectively. Earlier, Mirrokni and Zadimoghaddam [10] gave a 0.27-approximation in 2 rounds without duplication and a 0.545-approximation with $\Theta(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ duplication.

Our contribution. We focus on the most practical regime of MapReduce algorithms for cardinality constrained submodular maximization, which is a small constant number of rounds and no duplication of the dataset. To our knowledge, the 0.27-approximation of [10] has been the best result in this regime so far.

We describe a simple thresholding algorithm which achieves the following: In 2 rounds of MapReduce, with one random partitioning of the dataset (no duplication), we obtain a $(1/2-\epsilon)$-approximation. In 4 rounds, we obtain a 5/9-approximation. More generally, in $2t$ rounds, we obtain a $(1-(1-\frac{1}{t+1})^t-\epsilon)$-approximation, which we show to be optimal for this type of algorithm. Crucially, the parameter ϵ does not affect the number of rounds, and only mildly affects the memory (in that ϵ can be taken to $\tilde{O}(\sqrt{k/n})$ without asymptotically increasing the memory).

Our algorithm is inspired by the work of Kumar et al. [8] and McGregor-Vu [9] in the streaming setting. It is also similar to a recent algorithm of Assadi-Khanna [2], who study the communication complexity of the maximum coverage problem. As such, our algorithm is not particularly novel, but we believe that our analysis of its performance in the MapReduce model is, thus simplifying and improving the previous work of [5] and [10].

Open question. The most intriguing remaining question in our opinion (for the cardinality constrained submodular problem) is whether $\Theta(1/\epsilon)$ rounds are necessary to achieve a $(1-1/e-\epsilon)$-approximation. So far there is no evidence that a $(1-1/e)$-approximation in a constant number of rounds is impossible.

1.1 The MapReduce Model

There are many variants of MapReduce models, and algorithms between the different models are largely transferable. We use a variant of the $\mathcal{MRC}$ model of Karloff et al. [7]. In this model, an input of size N is distributed across $O(N^\delta)$ machines, each with $O(N^{1-\delta})$ memory. We relax the model slightly, and allow one central machine to have memory slightly expanded to $\tilde{O}(N^{1-\delta})$.

Computation then proceeds in a sequence of synchronous communication rounds. In each round, each machine receives an input of size $O(N^{1-\delta})$. Each machine then performs computations on that input, and produces output messages which are delivered to other machines (specified in the message) as input at the start of the next round. The total size of these output messages must also be $O(N^{1-\delta})$ per machine. We refer the reader to the work of Karloff et al. [7] for additional details.

In our applications, we assume the input is a set of elements V and a cardinality parameter k. Each machine has an oracle that allows it to evaluate f. Under these constraints, we assume that each machine has memory $O(\sqrt{nk})$ (except for a single 'central' machine with $O(\sqrt{nk}\log k)$ memory) and that there are $\sqrt{n/k}$ machines in total.

2 A thresholding algorithm for submodular maximization

In the following algorithms, let $f: 2^V \to \mathbb{R}^+$ be a monotone submodular function, $n = |V|$, and $f_S(e) = f(S\cup\{e\}) - f(S)$. We refer to $f_S(e)$ as the *marginal* of e with respect to S. Let k be the maximum cardinality of the solution, and $m = \sqrt{n/k}$ be the number of machines.

Algorithm 1: ThresholdGreedy(S, G, τ)

Input: An input set S, a partial greedy solution G with $|G| \leq k$, and a threshold τ.
Output: A set $G' \supseteq G$ such that $f_{G'}(e) < \tau$ for all $e \in S$ if $|G| < k$ or $f(G) \geq \tau k$.
$G' \leftarrow G$
for $e \in S$ **do**
  if $f_{G'}(e) \geq \tau$ **and** $|G'| < k$ **then** $G' \leftarrow G' \cup \{e\}$
return G'

Algorithm 2: ThresholdFilter(S, G, τ)

Input: An input set S, a partial greedy solution G, and a threshold τ.
Output: A set $S' \subseteq S$ such that $f_G(e) \geq \tau$ for all $e \in S'$.
$S' \leftarrow S$
for $e \in S$ **do**
  if $f_G(e) < \tau$ **then** $S' \leftarrow S' \setminus \{e\}$
return S'

2.1 A $1/2 - o(1)$ approximation in 2 rounds

First, we present a simple 1/2-approximation in 2 rounds, assuming we know the exact value of OPT. We will relax this assumption later. The algorithm requires two helper functions ThresholdGreedy and ThresholdFilter, which forms the basis of all of our algorithms. Roughly speaking, ThresholdGreedy greedily adds to a set of elements while there exists an element of high marginal in the input set. ThresholdFilter filters elements of low marginal out of the input set.

We define an additional function PartitionAndSample which simply initializes all of our algorithms by partitioning the input set randomly and drawing a random sample from it.

Algorithm 3: PartitionAndSample(V)

$S \leftarrow$ sample each $e \in V$ with probability $p = 4\sqrt{k/n}$
partition V randomly into sets $V_1, V_2, \ldots V_m$ to the m machines (one set per machine)
send S to each machine and a central machine C

Using these three helper algorithms, our approximation algorithm is quite easy to implement, and can be found in Algorithm 4.

▶ **Lemma 1.** *The approximation ratio of Algorithm 4 is at least* 1/2.

Proof. The following lemma is folklore, but we present it for completeness.

First, we note that G_0 is the same on each machine so long as the loop iterating through S is done in a fixed order. We assume that this is the case. From this, it is clear that Algorithm 4 returns a set G for which $f_G(e) < \frac{OPT}{2k}$ for any $e \in V$.

Let G be the set returned at the end of the algorithm. Either $|G| = k$, or there is no $e \in V$ for which the marginal with respect to G is greater than $OPT/2$. In the former case,

[1] Note that S and the V_i are not stored on one machine by PartitionAndSample. We simply use the assignment to denote that the variables have been initialized and sent to their respective machines.

Algorithm 4: A simple 2-round 1/2 approximation, assuming OPT is known.

round 1:
$S, V_1, \ldots, V_m \leftarrow$ PartitionAndSample(V)[1]
on *each machine* M_i *(in parallel)* **do**
  $\tau \leftarrow \frac{OPT}{2k}$
  $G_0 \leftarrow$ ThresholdGreedy $(S, \emptyset, \tau)$
  if $|G_0| < k$ **then** $R_i \leftarrow$ ThresholdFilter (V_i, G_0, τ)
  else $R_i \leftarrow \emptyset$
  send R_i to a central machine C
round 2 (only on C**):**
compute G_0 from S as in first round
$G \leftarrow$ ThresholdGreedy $(\cup_i R_i, G_0, \tau)$
return G

we have k elements of value at least $\frac{OPT}{2k}$ so we are done. In the latter case, let O be the optimal solution. By monotonicity and submodularity,

$$OPT = f(O) \leq f(O \cup G) \leq f(G) + \sum_{e \in O \setminus G} f_G(e) \leq f(G) + k \cdot \frac{OPT}{2k}. \quad \blacktriangleleft$$

Lemma 1 shows that the algorithm is correct. Each machine in round 1 clearly uses $O(\sqrt{nk})$ memory. It remains to bound the memory of the central machine in round 2.

▶ **Lemma 2.** *With probability* $1 - e^{-\Omega(k)}$*, the number of elements sent to the central machine* C *has cardinality at most* $\sqrt{nk}$*.*

Proof. The expected number of elements in S is $4\sqrt{nk}$. By a Chernoff bound (Theorem 9) the probability that $|S| < 3\sqrt{nk}$ is at most $e^{-\Omega(\sqrt{nk})} \leq e^{-\Omega(k)}$. So we can assume that $|S| \geq 3\sqrt{nk}$. Let N_S denote the number of elements of marginal at least $OPT/(2k)$ with respect to G_0. The number of elements sent to C in round two is exactly $N_S + |S|$.

Consider breaking the sample set S into $3k$ blocks of size $\sqrt{n/k}$ and processing each block sequentially. If before each block, there are at least $\sqrt{nk}$ remaining elements of marginal value at least $OPT/(2k)$, we have probability at least $1 - \left(1 - \sqrt{\frac{k}{n}}\right)^{\sqrt{\frac{n}{k}}} > 1/2$ of adding an additional element to G_0. This happens conditioned on any prior history of the algorithm, since we can imagine that the blocks are sampled independently one at a time. Therefore, we can use a martingale argument to bound the number of elements selected in S. If X_i is the indicator random variable for the event that at least one element is selected from the i-th block, then we have $E[X_i \mid X_1, \ldots, X_{i-1}] \geq 1/2$. Hence we can define $Y_i = \sum_{j=1}^{i}(X_i - 1/2)$ and the sequence $Y_1, Y_2, \ldots$ is a submartingale, which means $E[Y_i \mid Y_1, \ldots, Y_{i-1}] \geq Y_{i-1}$. Moreover, $|Y_i - Y_{i-1}| \leq 1$. By Azuma's inequality (Theorem 10), $\Pr[Y_{3k} < -\frac{1}{2}k] < e^{-\Omega(k)}$. This means that with probability $1 - e^{-\Omega(k)}$, $\sum_{j=1}^{3k} X_j = Y_{3k} + \frac{3}{2}k \geq k$, and we include at least k elements overall. In that case, we are done and do not send anything to the central machine. Otherwise, the number of remaining elements of marginal value at least $OPT/(2k)$ drops below $\sqrt{nk}$. ◀

Remaining issues. Since we do not know the exact value of OPT, we will need to guess the value within a factor of ϵ without increasing the number of rounds. This will increase memory usage on the central machine by a factor of $\frac{1}{\epsilon}\log k$. To do this, we classify the inputs

into two classes: when the input contains more than $\sqrt{nk}$ elements of value at least $\frac{OPT}{2k}$, and when there are less than $\sqrt{nk}$ such elements. We call the former class of inputs "dense" and the latter class "sparse". For each input class, we design a 1/2-approximation in 2 rounds. Given the input, we can run both in parallel and return the better of the two solutions: each machine simply runs both algorithms at the same time, keeping the number of machines the same. The full analysis is given in the Appendix, but we outline the algorithms below.

A 2-round algorithm for "dense" inputs

Let v be the maximum value of a single element of the random sample S in Algorithm 4. When the input is dense, v is likely to be at least $\frac{OPT}{2k}$ and at most OPT. A straightforward analysis shows that $\tau_j := v(1+\epsilon)^j$ is within a $(1+\epsilon)$ multiplicative factor of $OPT/2$ for some $j \in \{1, \ldots, \frac{1}{\epsilon}\log k\}$. Running Algorithm 4 with τ_j instead of $OPT/2$ produces an approximation of value at least $\frac{OPT}{2(1+\epsilon)} > \frac{OPT}{2}(1-\epsilon)$. Thus if each machine runs $\frac{1}{\epsilon}\log k$ copies of Algorithm 4, the best solution must have value at least $\frac{OPT}{2}(1-\epsilon)$.

A 2-round algorithm for "sparse" inputs

Call an element e "large" if $f(e) \geq \frac{OPT}{2k}$. The algorithm simply sends all the large elements of the input onto one machine and then runs a sequential algorithm in the second round. To get all the large elements onto one machine, we randomly partition the input set onto the m machines, and then send the $O(k)$ largest elements on each machine to the central machine. On the central machine, we can run the same thresholding procedure as in the "dense" case to find a threshold close to $OPT/(2k)$. We then run a sequential version of Algorithm 4.

In both the algorithms, ϵ can be taken to $\tilde{O}(\sqrt{k/n})$ without asymptotically increasing the memory, so we have a $(1/2 - o(1))$-approximation.

2.2 A $1 - \left(1 - \frac{1}{t+1}\right)^t$ approximation in $2t$ rounds

Here we show how our algorithm extends to t thresholds. The number of MapReduce rounds becomes $2t+2$. This can be reduced to $2t$ using tricks similar to Section 2.1, but we omit this here. The approximation factor with t thresholds is $1-(1-\frac{1}{t+1})^t$, which converges to $1-1/e$. We note that we need $\Theta(1/\epsilon)$ rounds to obtain a $(1-1/e-\epsilon)$-approximation, similar to Barbosa et al. [5], but in contrast we do not need any duplication of the ground set. Barbosa et al. does not specify the constant factor in $\Theta(1/\epsilon)$ but it seems that our dependence is better; a calculation yields that we need $(1+o(1))/\epsilon$ rounds to get a $(1-1/e-\epsilon)$-approximation.

For now, we assume (as in Algorithm 4) that we know the exact value of OPT. We deal with this assumption later. In a nutshell our algorithm works just like Algorithm 1 but with multiple thresholds used in a sequence. We set the threshold values as follows:

$$\alpha_\ell = \left(1 - \frac{1}{t+1}\right)^\ell \frac{OPT}{k}$$

for $1 \leq \ell \leq t$. (Note that for $t = 1$, we get $\alpha_1 = \frac{OPT}{2k}$ as in Algorithm 1.) For each threshold, we first select elements above the threshold from a random sample set, and then use this partial solution to prune the remaining elements. Finally, the solution at this threshold is completed on a central machine, and we proceed to the next threshold. The full description of the algorithm is presented in Algorithm 5. The analysis is as follows.

▶ **Lemma 3.** *The approximation ratio of Algorithm 5 is at least* $1 - \left(1 - \frac{1}{t+1}\right)^t$.

Proof. By induction, we prove the following statement: The value of the first $\frac{\ell}{t}k$ elements selected by the algorithm is at least $(1-(1-\frac{1}{t+1})^{\ell})OPT$. (If $\frac{\ell}{t}k$ is not an integer, we count the marginal value of the $\lceil\frac{\ell}{k}k\rceil$-th selected element weighted by its respective fraction.)

Clearly this is true for $\ell = 0$. Assume that the claim is true for $\ell - 1$. We consider two cases.

Either all the elements among the first $\lceil\frac{\ell}{t}k\rceil$ are selected above the α_ℓ threshold. This means that since the value of the first $\frac{\ell-1}{t}k$ elements was at least $(1-(1-\frac{1}{t+1})^{\ell-1})OPT$, and the marginal value of each additional element is at least α_ℓ, the total value of the first $\frac{\ell}{k}OPT$ (with fractional elements counted appropriately) is at least

$$\left(1-\left(1-\frac{1}{t+1}\right)^{\ell-1}\right)OPT + \frac{1}{t}\cdot\left(1-\frac{1}{t+1}\right)^{\ell}OPT = \left(1-\left(1-\frac{1}{t+1}\right)^{\ell}\right)OPT.$$

The other case is that not all these elements are selected above the α_ℓ threshold, which means that if we denote by S_ℓ the set of the first $\lfloor\frac{\ell}{t}k\rfloor$ selected elements, then there are no elements with marginal value more than α_ℓ with respect to S_ℓ. But then for the optimal solution O, we get

$$OPT - f(S_\ell) \leq f_{S_\ell}(O) \leq k\alpha_\ell = \left(1-\frac{1}{t+1}\right)^{\ell}OPT$$

which means that $f(S_\ell) \geq (1-(1-\frac{1}{t+1})^{\ell})OPT$.

For $\ell = t$, we obtain the statement of the lemma. ◀

The probabilistic analysis of the number of pruned elements that need to be sent to the central machine is exactly the same as in Section 2.1. The requirement of knowing OPT can be also handled in the same way — we can use an extra initial round to determine the maximum-value element on the input, which gives us an estimate of the optimum within a factor of k. Then we can try $O(\frac{1}{\epsilon}\log k)$ different estimates of OPT to ensure that one of them is within a relative error of $1+\epsilon$ of the correct value. Finally, we use an extra final round to choose the best of the solutions that we found for different estimates of OPT. Alternatively, we can use additional tricks as in Section 2.1 to eliminate these 2 extra rounds, but we omit the details here.

Algorithm 5: A $2t$-round $1-\left(\frac{t}{t+1}\right)^{t}$ approximation, assuming OPT is known.

```
G ← ∅
for ℓ = 1, ... t do
    round 2ℓ − 1:
    S, V_1, ..., V_m ← PartitionAndSample(V)
    on each machine M_i (in parallel) do
        G_0 ← ThresholdGreedy(S, G, α_ℓ)
        if |G_0| < k then R_i ← ThresholdFilter(V_i, G_0, α_ℓ)
        else R_i ← ∅
        send R_i to a central machine C
    round 2ℓ (only on C):
    compute G_0 from S as in first round
    G ← ThresholdGreedy(∪_i R_i, G_0, α_ℓ)
return G
```

3 Optimality of our choice of thresholds

Here we present a proof that there is no way to modify the thresholding algorithm and achieve a better approximation factor with a different choice of thresholds.

▶ **Theorem 4.** *The thresholding algorithm with t thresholds cannot achieve a factor better than $1 - \left(1 - \frac{1}{t+1}\right)^t$.*

Proof. Assume that the optimum O consists of k elements of total value kv^*. Since we are proving a hardness result, we can assume that the algorithm has this information and we can even let it choose v^*; in the following, we denote this choice $v^* = \alpha_0$. In addition, the algorithm chooses thresholds $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_t$. It might be the case that $\alpha_0 < \alpha_1$, but then we can ignore all the thresholds above α_0 and design our hard instance based on the thresholds below α_0, which would reduce to a case with fewer thresholds. Thus we can assume $\alpha_0 \geq \alpha_1 \geq \ldots \geq \alpha_t$.

We design an adversarial instance as follows. In addition to the k elements of value v^*, we have a set S of other elements where element i has value v_i, such that $\sum_{i \in S} v_i \leq kv^*$. The objective function is defined as follows: for $O' \subseteq O$ and $S' \subseteq S$,

$$f(S' \cup O') = \sum_{i \in S'} v_i + \left(1 - \frac{\sum_{i \in S'} v_i}{kv^*}\right) |O'| v^*.$$

It is easy to verify that this is a monotone submodular function. (It can be realized as a coverage function, which we leave as an exercise.)

Now we specify more precisely the values of elements in S. We will have n_ℓ elements of value α_ℓ, for each $1 \leq \ell \leq t$. The idea is that the algorithm will pick these n_ℓ elements at threshold value α_ℓ, at which point the marginal value of the optimal elements drops below α_ℓ, so we have to move on to the next threshold. A computation yields that we should have $n_\ell = (\frac{\alpha_{\ell-1}}{\alpha_\ell} - 1)k$.[2] The total value of these elements is $\sum_{i \in S} v_i = \sum_{\ell=1}^t n_\ell \alpha_\ell = \sum_{\ell=1}^t (\alpha_\ell - \alpha_{\ell-1})k = (\alpha_0 - \alpha_t)k \leq v^* k$ as required above.

Then, assuming that the marginal value of the optimum after processing $\ell - 1$ thresholds was $\alpha_{\ell-1}k$, the marginal value after processing the ℓ-th threshold will be $\alpha_{\ell-1}k - n_\ell \alpha_\ell = \alpha_\ell k$. By induction, the algorithm selects exactly n_ℓ elements of value α_ℓ, unless the constraint of k selected elements is reached. Let us denote by n'_ℓ the actual number of elements selected by the algorithm at threshold level α_ℓ. We have $n'_\ell \leq n_\ell$, and $\sum_{\ell=1}^t n'_\ell \leq k$, as discussed above.

The total value collected by the algorithm is $\sum_{\ell=1}^t n'_\ell \alpha_\ell$. Since we have $n'_\ell \leq n_\ell = (\frac{\alpha_{\ell-1}}{\alpha_\ell} - 1)k$, and $\alpha_\ell \leq \alpha_{\ell-1}$, we can define inductively $\alpha'_0 = \alpha_0$ and $\alpha'_\ell \geq \alpha_\ell$ such that $n'_\ell = (\frac{\alpha'_{\ell-1}}{\alpha'_\ell} - 1)k$. Then the total value collected by the algorithm is

$$\sum_{\ell=1}^t n'_\ell \alpha_\ell \leq \sum_{\ell=1}^t n'_\ell \alpha'_\ell = \sum_{\ell=1}^t (\alpha'_{\ell-1} - \alpha'_\ell)k = (\alpha'_0 - \alpha'_t)k.$$

Let us denote $\beta'_\ell = \frac{\alpha'_{\ell-1}}{\alpha'_\ell}$. We have $\sum_{\ell=1}^t (\beta'_\ell - 1) = \frac{1}{k} \sum_{\ell=1}^t n'_\ell \leq 1$, hence $\sum_{\ell=1}^t \beta'_\ell \leq t + 1$. Recall that the value achieved by the algorithm is $\sum_{\ell=1}^t n'_\ell \alpha_\ell \leq (\alpha'_0 - \alpha'_t)k = (1 - 1/\prod_{\ell=1}^t \beta'_\ell) v^* k$. By the AMGM inequality, $\prod_{\ell=1}^t \beta'_\ell$ is maximized subject to $\sum_{\ell=1}^t \beta'_\ell \leq t + 1$ when all the β'_ℓ are equal, $\beta'_\ell = \frac{t+1}{t}$. Then, the value achieved by the algorithm is $(1 - 1/\prod_{\ell=1}^t \beta'_\ell) v^* k = (1 - (\frac{t}{t+1})^t) OPT$. ◀

[2] We ignore the issue that n_ℓ might not be an integer. For large k, it is easy to see that the rounding errors are negligible.

References

1 Sepehr Assadi and Sanjeev Khanna. Randomized composable coresets for matching and vertex cover. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2017.

2 Sepehr Assadi and Sanjeev Khanna. Tight bounds on the round complexity of the distributed maximum coverage problem. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2412–2431, 2018. URL: https://doi.org/10.1137/1.9781611975031.155, doi:10.1137/1.9781611975031.155.

3 Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone submodular maximization in exponentially fewer iterations. *CoRR*, abs/1807.11462, 2018. URL: http://arxiv.org/abs/1807.11462, arXiv:1807.11462.

4 Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. *CoRR*, abs/1804.06355, 2018. URL: http://arxiv.org/abs/1804.06355, arXiv:1804.06355.

5 Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *Proceedings of the IEEE 57th Annual Symposium on Foundations of Computer Science*, 2016.

6 Satoru Fujishige. *Submodular functions and optimization*, volume 58. Elsevier, 2005.

7 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010. doi:10.1137/1.9781611973075.76.

8 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in MapReduce and streaming. *TOPC*, 2(3):14:1–14:22, 2015. doi:10.1145/2809814.

9 Andrew McGregor and Hoa T. Vu. Better streaming algorithms for the maximum coverage problem. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 22:1–22:18, 2017. URL: https://doi.org/10.4230/LIPIcs.ICDT.2017.22, doi:10.4230/LIPIcs.ICDT.2017.22.

10 Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *ACM Symposium on Theory of Computing (STOC)*, pages 153–162, 2015.

11 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. URL: https://doi.org/10.1007/BF01588971, doi:10.1007/BF01588971.

12 Martin Raab and Angelika Steger. "balls into bins" — a simple and tight analysis. In Michael Luby, José D. P. Rolim, and Maria Serna, editors, *Randomization and Approximation Techniques in Computer Science*, pages 159–170, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

Algorithm 6: A $1/2-\epsilon$ approximation in 2 rounds for "dense" inputs.

round 1:
$S, V_1, \ldots, V_m \leftarrow$ PartitionAndSample(V)
on *each machine* M_i *(in parallel)* **do**
  $v \leftarrow \max_{e \text{ on } M_i} f(\{e\})$
  for $j = 1, \ldots, \frac{1}{\epsilon}\log k$ **do**
    $\tau_j \leftarrow v(1+\epsilon)^j/k$
    $G_{0,j} \leftarrow$ ThresholdGreedy$(S, \emptyset, \tau_j)$
    if $|G_{0,j}| < k$ **then** $R_{i,j} \leftarrow$ ThresholdFilter$(V_i, G_{0,j}, \tau_j)$
    else $R_{i,j} \leftarrow \emptyset$
  send all $R_{i,j}$ to a central machine C
round 2 (only on C):
compute v, τ_j, and $G_{0,j}$ from S as in first round
for $j = 1, \ldots, \frac{1}{\epsilon}\log k$ **do**
  $G_j \leftarrow$ ThresholdGreedy$(\cup_i R_{i,j}, G_{0,j}, \tau_j)$
$G^\star = \text{argmax}_{G_j} f(G_j)$
return $G^\star$

A Appendix

In Algorithm 6, we design a 2-round $1/2-\epsilon$ approximation for "dense" inputs.

▶ **Lemma 5.** *Algorithm 6 returns a* $1/2-\epsilon$ *approximation.*

Proof. Note that Algorithm 6 essentially runs Algorithm 4 with $O(\frac{1}{\epsilon}\log k)$ guesses for $OPT/2$. To get a $1/2-\epsilon$ approximation, one of these guesses needs to be within a multiplicative factor of $1+\epsilon$ from $\frac{OPT}{2k}$. By the denseness assumption on the input, we know $\frac{OPT}{2k} \le v \le OPT$ with high probability. Suppose we try $j = 1, \ldots, \ell$ for some number of thresholds ℓ. For one of the τ_j's to be within a multiplicative factor of $1+\epsilon$ from $\frac{OPT}{2k}$, we require $v(1+\epsilon)^\ell/k < \frac{OPT}{2k} < v$. The upper bound is already satisfied by the denseness assumption, and the lower bound is achieved by taking ℓ for which $\ell \ge \log\left(\frac{OPT}{2kv\log(1+\epsilon)}\right) \ge \frac{1}{\epsilon}$. ◀

▶ **Lemma 6.** *The number of elements sent to the central machine is* $O\left(\frac{1}{\epsilon}\sqrt{nk}\log k\right)$.

Proof. This follows from Lemma 2 and the fact that there are only $\frac{\log k}{\epsilon}$ thresholds. ◀

Next, we design a 2-round $1/2-\epsilon$ approximation for "sparse" inputs (Algorithm 7).

▶ **Lemma 7.** *Algorithm 7 gives a* $1/2-\epsilon$ *approximation.*

Proof. There are two things to check: that one of the τ_j's is close to $OPT/2$, and that the machine C is not missing any of the elements that it needs.

For the former, its clear that by similar reasoning to Lemma 5, one of the τ_j's will be within a $1+\epsilon$ multiplicative factor to $OPT/2$.

For the latter, note that the "sparseness" assumption implies that with high probability, the $\sqrt{nk}$ large elements will be equally distributed among the machines, and each machine will get k elements in expectation. Since we send $O(k)$ elements to the central machine, C will have all the large elements in V with high probability. This can be shown by a standard balls-and-bins analysis [12]. ◀

Algorithm 7: A $1/2 - \epsilon$ approximation in 2 rounds for "sparse" inputs.

round 1:
partition V uniformly at random to the m machines
on *each machine* M_i **do**
  send its $O(k)$ largest elements to a central machine C
round 2 (only on C**):**
$S \leftarrow$ all elements sent to C
$v \leftarrow \max_{e \in S} f(\{e\})$
for $j = 1, \ldots, \frac{1}{\epsilon} \log k$ **do**
  $\tau_j \leftarrow v(1+\epsilon)^j / k$
  $G_j \leftarrow \textsc{ThresholdGreedy}\,(S, \emptyset, \tau_j)$
$G^\star = \text{argmax}_{G_j} f(G_j)$
return $G^\star$

Finally, we note that the total memory use on C is $O(\sqrt{nk})$ since each machine sends $O(k)$ elements and there are $O(\sqrt{\frac{n}{k}})$ machines in total.

▶ **Theorem 8.** *By running Algorithms 6 and 7 in parallel, we have a 2-round* $1/2 - \epsilon$ *approximation.*

B Auxiliary Results

▶ **Theorem 9** (Chernoff bound)**.** *Let* $X_1, \ldots, X_n$ *be independent random variables such that* $X_i \in [0, 1]$ *with probability 1. Define* $X = \sum_{i=1}^{n} X_i$ *and let* $\mu = \mathbb{E}X$*. Then, for any* $\epsilon > 0$*, we have*

$$\Pr[X \geq (1+\epsilon)\mu] \leq \exp\left(-\frac{\min\{\epsilon, \epsilon^2\}\mu}{3}\right).$$

▶ **Theorem 10** (Azuma's inequality)**.** *Suppose* $X_1, \ldots, X_n$ *is a submartingale and* $|X_i - X_{i+1}| \leq c_i$*. Then, we have*

$$\Pr[X_n - X_0 \leq -t] \leq \exp\left(\frac{-t^2}{2\sum_i c_i^2}\right).$$