Maximizing Unconstrained Non-Monotone Submodular Functions in a MapReduce Setting

Sikander Randhawa Ben Chugg Angad Kalra

Abstract

Submodular functions arise in a variety of applications. Likewise, in the age of big data many organizations have adopted the MapReduce framework as a model for parallel computation. In this project, we study the problem of maximizing an unconstrained, non-monotone, submodular set function $f : 2^{\mathcal{U}} \to \mathbb{R}$, where the ground set $\mathcal{U}$ may be too large to fit on a constant number of machines. We examine the potential of three well known algorithms to be parallelized in the MapReduce model: Deterministic and smooth Local Search given in [4] and Double Greedy given in [2].

1 Introduction

The problem of maximizing submodular functions has been a subject of much interest in combinatorial optimization, algorithmic game theory and machine learning. For example, the cut function on a graph is submodular and finding a max cut gives rise to a submodular maximization problem. Further applications include combinatorial auctions and the social welfare problem. The general problem of maximizing submodular functions is NP-Hard [4], so we settle for approximations.

Submodular maximization has been studied extensively in various settings and under various constraints. For instance, submodular maximization subject to matroid or cardinality constraints is a well-understood subject (e.g., [1],[9]). A tight 1/2-approximation for the unconstrained case has also recently been given [2]. However, most practical applications of submodular maximization are in a setting where the data is too large to be processed on a single machine. It is important, therefore, to understand how to parallelize these approximation algorithms.

1.1 Previous Work and Motivation

Recently, efforts have been made involving parallelizing approximation algorithms for non-monotone submodular maximization. This began with Mirzasoleiman et al. [10] proposing a two-stage protocol tailored towards the Map-Reduce model. This algorithm arbitrarily partitions the data amongst the machines and then runs a black box approximation algorithm for non-monotone submodular maximization on each machine. The sub-solutions are then aggregated and in the second stage the black box algorithm is run again with respect to the aggregate solutions. The approximation obtained from this protocol is inversely proportional to the number of machines. In fact, they obtain a $(\frac{\tau}{p})$-approximation, where τ is the approximation factor of the black-box algorithm and p is the number of machines. Therefore, as the size of our data grows, this algorithm performs rather poorly since the number of necessary machines will grow.

Barbosa et al. [3] modify this two-stage idea by introducing random partitioning and slight modifications to each round. Complete details can be found in section 4 of [3], but the essence of the algorithm parallels the approach taken by Mirzasoleiman. Introducing randomization eliminates the dependence on the number of machines, however they achieve an approximation factor which does not compete well with the state-of-the-art centralized approximations. They achieve a $\frac{\beta\gamma}{\beta+2\gamma}$ approximation, where β is the approximation factor which the standard Greedy algorithm obtains

with respect to non-monotone submodular maximization subject to matroid constraints, and γ is the approximation factor of any arbitrary black-box algorithm for the same problem.

Barbosa et al. and Mirzasoleiman et al. consider only non-monotone submodular maximization subject to hereditary constraints, whereas we are interested in unconstrained submodular maximization. Both approaches use a two-stage protocol which builds a set of solutions in the first round, and in the second round uses these solutions as a subuniverse on which to apply some approximation algorithm.

We are motivated by trying to close the gap between the centralized approximation guarantees and the distributed approximation guarantees. We will try to analyze well known algorithms for unconstrained submodular maximization, and explore simulating these algorithms in a parallelized manner in hopes of achieving the same approximation factor. In doing so, we require a shift from the current two stage algorithms, and instead allow our procedures to run until completion. The Local Search procedure given by Feige et al. [4] shows good promise in terms of its ability to be simulated in a distributed setting. However, as will be seen in this paper, the main challenge comes from bounding the number of rounds until completion.

1.2 Our Contribution

Our main contribution is to propose a parallelization of the Deterministic Local Search algorithm (LS) of Feige et al. [4] in the MapReduce setting. We will demonstrate that correctness follows immediately from the centralized version, and make some exploratory attempts to bound the number of rounds. To this end, we provide the results of several numerical experiments which suggest that the number of rounds is indeed polylogarithmic. As a negative result, we demonstrate that the expected number of rounds is $\Omega(\log n)$, for an input of size n. Additionally, we study potential parallelizations of Smooth Local Search (SLS) and Double Greedy (DG).

The rest of the paper will be laid out as follows. We end this section with the relevant definitions and background. Section 2 will concern the parallelization of deterministic Local Search. Here we will present the algorithm and the relevant experimental results. We explore several ideas pertaining to bounding the number of rounds required by the algorithm. We end this section with a brief discussion on extensions to Smooth Local Search. Finally, section 3 contains a discussion of a parallelization attempt of the Double Greedy Algorithm.

1.3 Preliminaries and Background

Unconstrained Submodular Maximization (USM). We are concerned with the problem of maximizing a *non-monotone* submodular function, that is, $\max_{S \subset \mathcal{U}} f(S)$, where f is submodular in the following sense.

Definition 1. *A function $f : 2^{\mathcal{U}} \to \mathbb{R}$ is submodular over the ground-set $\mathcal{U}$, iff for all $S, T \subset \mathcal{U}$, $f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$. We refer to the ground set $\mathcal{U}$ as the universe.*

In what follows, we will always assume that $|\mathcal{U}| = n < \infty$. We also assume an oracle access model: Given a current set A, we can evaluate $f(A)$ in $\Theta(1)$-time. In the seminal work of Feige, Mirrokni and Vondrak [4], they demonstrate that simply choosing a random set gives an $(1/4 - o(1))$-approximation, and give another non-adaptive algorithm which provides a $(1/3 - o(1))$-approximation. Both of these algorithms are naturally parallelizable, as they simply involve choosing random sets with certain probabilities. As such, any MapReduce algorithm for USM should strive to beat these two approximation ratios. The authors also provide two adaptive algorithms mentioned in the previous section, LS and SLS which achieve approximation ratios of $1/3 - o(1)$,

and $2/5-o(1)$ respectively. These two algorithms are our main focus for parallelization. We focused mainly on LS, as SLS is a natural extension of LS, and so it seemed reasonable that a parallelization of SLS would follow naturally from a parallelization of LS.

More recently, Buchbinder et al. provided a $1/2$ approximation to USM, which is known to be tight [2]. They do this via a double greedy algorithm, which we will only briefly explore as a possibility for parallelization.

The MapReduce Model and the $\mathcal{MRC}$ class. We adopt the MapReduce framework as presented by Karloff, Suri and Vassilvitskii in [5]. We give a brief overview of the framework here. A unit of data in this model is a key-value pair $\langle k, v\rangle$. Mappers are functions which send a single key-value pair to a multiset of key-value pairs. A reducer takes as input all values with the same key and emits a multiset of key-value pairs. A single round of a procedure consists of the following three phases:

1. (Mapping Phase). Each key-value pair is acted on by a mapper.
2. (Shuffle Phase). Here, we gather all the values which share the same key.
3. (Reduce Phase). For each key, apply the reducer to the set of values which share this key.

We remark that a single procedure may require multiple rounds. Karloff et al. also a define a class of problems, $\mathcal{MRC}$, which we define next.

Definition 2. *For any $j \in \mathbb{N}$, we define the class $\mathcal{MRC}^j$ as the class of all problems for which there exists an algorithm $\mathscr{A}$ obeying the following restrictions: 1) $\mathscr{A}$ is implemented in MapReduce, 2) $\mathscr{A}$ uses a sublinear number of machines, 3) each machine uses a sublinear amount of space, 4) $\mathscr{A}$ uses $O(\log^j n)$ rounds.*

We will strive to have our proposed algorithms meet the requirements of the class $\mathcal{MRC}^j$ for some j. In defining $\mathcal{MRC}$, Karloff et al. only require that the algorithm return the correct answer with probability greater than $3/4$. We do not admit the same relaxation.

2 Distributed Local Search

In this section, we propose a parallelization of Local Search. First, let us recall deterministic Local Search (LS) presented in [4]. Throughout, we will assume that all submodular functions are positive, i.e., $f : 2^{\mathcal{U}} \to \mathbb{R}_{\geq 0}$, as this assumption is made by the authors of LS. This does not lead to any loss of generality, as we may simply shift the values of a non-positive submodular function.

Algorithm: Local Search (LS)
Input: A submodular function $f : 2^{\mathcal{U}} \to \mathbb{R}$; A seed solution, S; A universe, $\mathcal{U}$; An accuracy parameter ϵ.
Procedure:

If there exists $e \in \mathcal{U} \setminus S$ such that $f(S \cup \{e\}) > (1 + \frac{\epsilon}{n^2})f(S)$, then let $S := S \cup \{e\}$.

If there exists $e \in S$ such that $f(S \setminus \{e\}) > (1 + \frac{\epsilon}{n^2})f(S)$, then let $S := S \setminus \{e\}$.

Repeat the above two steps until no such e exists.

Return $\text{argmax}\{f(S), f(\mathcal{U} \setminus S)\}$.

The authors in [4] prove that LS achieves a $1/3 - o(1)$ approximation by observing that upon termination, the solution S is a $(1 + \frac{\epsilon}{n^2})$-approximate local optimum, where an approximate local optimum is defined as follows.

Definition 3. *Let $f : 2^{\mathcal{U}} \to \mathbb{R}$ be a set function. A set $S \subseteq \mathcal{U}$ is a $(1+\alpha)$-approximate local optimum for $\alpha > 0$ if $f(S \cup \{u\}) \leq (1+\alpha)f(S)$ and $f(S \setminus \{s\}) \leq (1+\alpha)f(S)$ for all $u \in \mathcal{U} \setminus S$ and all $s \in S$.*

In order to parallelize LS, we take advantage of the fact that the value of the solution is constantly increasing. We would like to maintain the fact that upon termination of the algorithm the solution is a $(1 + \epsilon/n^2)$-approximate local optimum.

Algorithm: Distributed Local Search (DLS)
Input: A submodular function $f : 2^{\mathcal{U}} \to \mathbb{R}$.
Procedure:

$S \leftarrow \{u\}$, where $u \in \text{argmax}_{u \in \mathcal{U}} f(\{u\})$.

While any machine improves the value of the solution,

Randomly Partition $\mathcal{U} \setminus S$ into p, $\{U_1, \ldots, U_p\}$, where each element e goes to machine i with probability $(1/p)$.

Run $\mathsf{LS}(f, S, U_i, \epsilon)$ on each machine i, to obtain solution S_i.

Let $S = \text{argmax}_i\{S_i\}$

Return $\text{argmax}\{f(S),\ f(\mathcal{U} \setminus S)\}$

Observe that given a set S, we can compute $f(\mathcal{U} \setminus S)$ without explicit knowledge of $\mathcal{U} \setminus S$ via oracle access with just the elements S available to a machine. This is because the elements of S immediately expose the elements which are in $\mathcal{U} \setminus S$. So, given our assumption that our intermediate solution S will always be able to fit on a machine, we really are able to compute $\text{argmax}\{f(S),\ f(\mathcal{U} \setminus S)\}$.

Next, observe that correctness of DLS follows easily from the centralized version.

Lemma 1. *Let $f : 2^{\mathcal{U}} \to \mathbb{R}_{\geq 0}$ be a submodular function.* DLS *provides a $(1/3 - o(1))$-approximation to $\max_{S \subset \mathcal{U}} f(S)$.*

Proof. Upon termination of DLS, no machine has made any improvements to the previous solution S. Therefore, S is a $(1 + \frac{\epsilon}{n^2})$-approximate local optimum, and correctness follows from **Theorem 3.4** in [4]. ◀

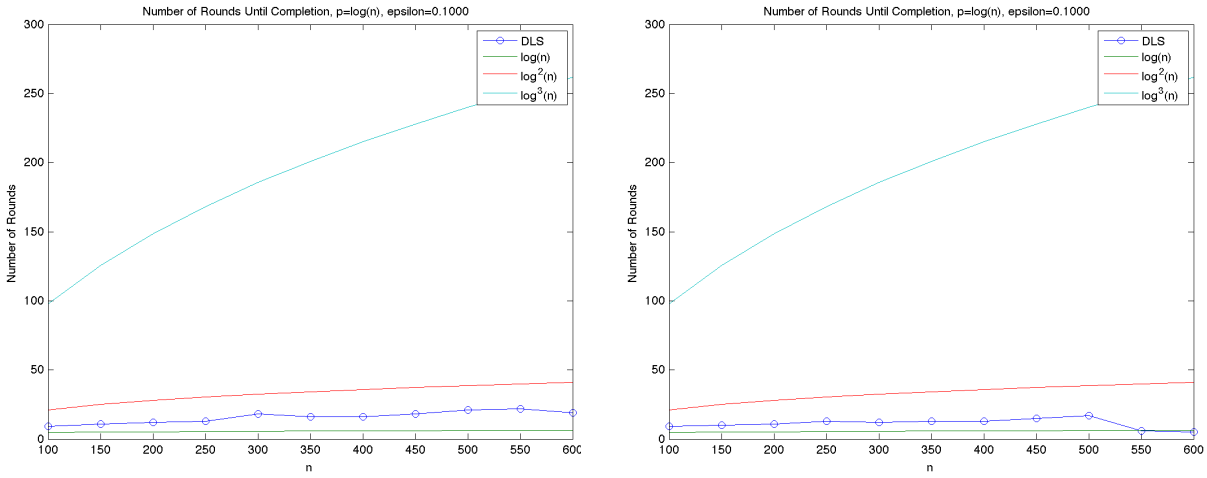
Of course, to ensure that USM $\in \mathcal{MRC}^j$ for some j, we must bound the number of rounds of DLS to be polylogarithmic. We consider one round to be one iteration of the while loop, namely partitioning the universe, running LS on each machine, and gathering the best solution. It is worth remarking at this point that the number of rounds of DLS, in the above sense, is not the number of MapReduce rounds required for DLS. However, each round of DLS can be implemented in a constant number of MapReduce rounds—therefore to bound the number of DLS rounds is to asymptotically bound the rounds of DLS implemented in MapReduce by the same amount.

2.1 Experiments

We present the results of numerical experiments, which seem to suggest that the number of rounds may indeed by bounded above by some polylog. The following experiments were performed in MATLAB. DLS was not truly implemented in parallel; it was imitated by simply running LS on each set of the random partition one after the other and recording the best solution. The code can be found at `https://github.com/bchugg/usm-mr`. We ran a variety of tests on two submodular functions: the directed cut function and the function measuring the mutual information of random variables. It is well known that the cut function of a graph is submodular. The function measuring symmetric mutual information is discussed in [7]. Formally these functions are defined as follows.

Definition 4. *Let $G = (V, E)$ be a weighted, directed graph. For a set $C \subseteq V$, let $\delta_C = \{(u, v) : (u, v) \in E,\ u \in C,\ v \in V \setminus C\}$, define $f : 2^V \to \mathbb{Z}_{\geq 0}$ by $f(C) = |\delta_C|$. We say f is the cut-function on the directed graph G.*

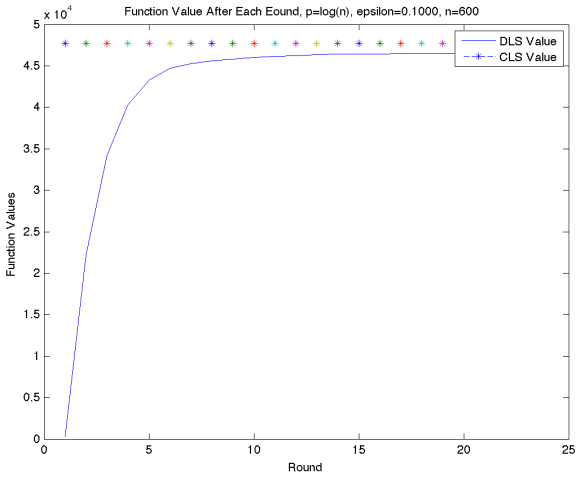
Definition 5. *Let $\Omega = \{X_1, \ldots, X_n\}$ be a set of random variables. For $A \subseteq \Omega$, if $I(A; \Omega \setminus A)$ is the mutual information, define $g(A) = I(A; \Omega \setminus A)$. Then g is a non-monotone submodular function.*



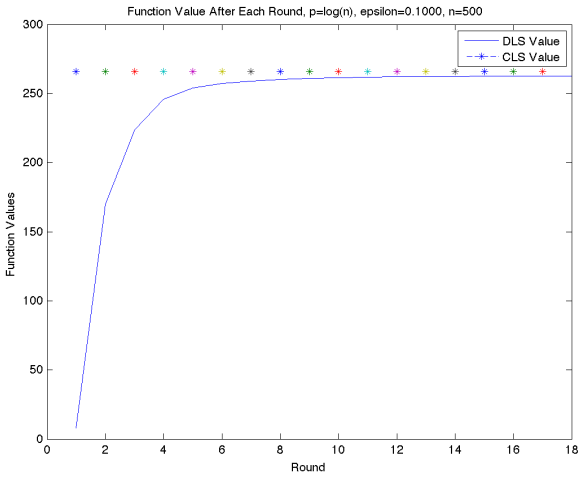
(a) Cut Function

(b) Mutual Information

Figure 1: The number of rounds until completion of two submodular functions. The lines $\log^3(n)$, $\log^2(n)$, $\log(n)$ were plotted for comparison.

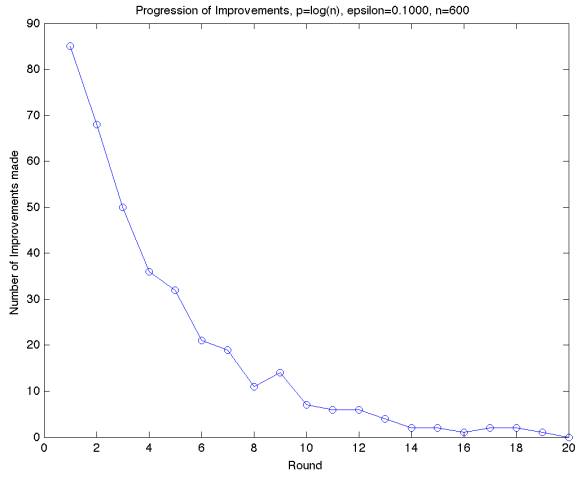


(a) Cut Function

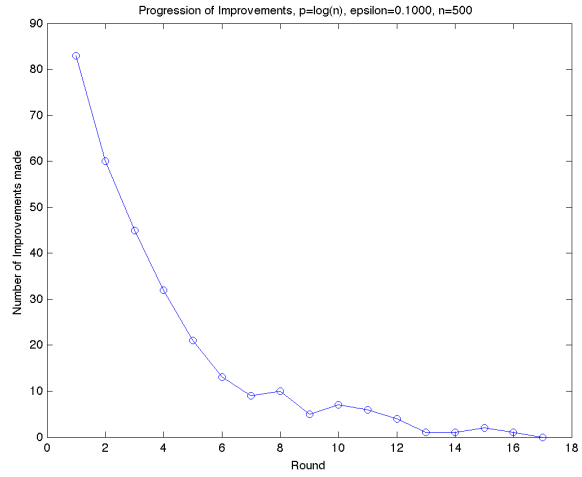


(b) Mutual Information

Figure 2: The progression of the value of the solution of DLS compared to the value of LS run on the entire universe. Here, CLS denotes Centralized Local Search.



(a) Cut Function



(b) Mutual Information

Figure 3: An examination of the progress made at during each round of DLS, for $p = \log(n)$.

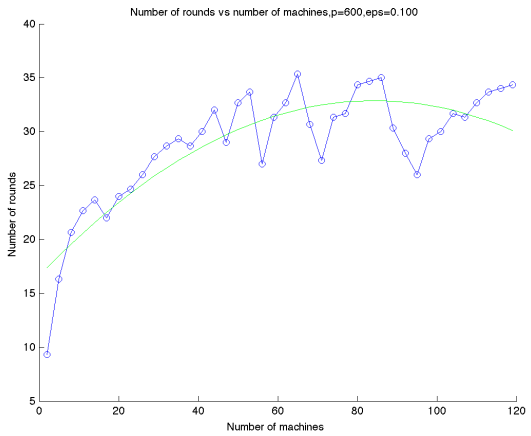


Figure 4: Plot of the rounds required until completion under a various number of machines. Throughout, $n = 600$ was used. We remark that the title of the plot is misleading, it should read $n = 600$, not $p = 600$.

The software used to generate and evaluate the submodular functions was the Submodular Function Optimization Toolbox written by Andreas Krause [6].

With the caveat that all the experiments were run for relatively small values of n, there does seem to be some behaviour indicative of an upper bound on the number of rounds of DLS. Moreover, instead of DLS linearly approaching the solution of LS, there seems to be great gains made at the beginning followed by a period of relatively slow growth. It is not clear, however, how to use this observation in any analysis to bound the number of rounds—this will be further discussed in Section 2.3.

2.2 A Lower Bound on the Number of rounds of DLS

In this section we demonstrate that DLS requires at least $\Omega(p)$ rounds in expectation, where p is the number of machines. Consider the following example.

Let f be the Directed-Cut function from definition 4. We consider the problem of finding the maximum sized Directed-Cut of a graph G, that is,

$$\max_{C \subseteq V(G)} f(C).$$

Now, let $G = (V, E)$ be a bipartite graph and let p be the number of machines in the DLS algorithm and assume $p \in \mathcal{O}(n^{1-\epsilon})$. Suppose that G contains a perfect matching, M, of size p and the remaining vertices of the graph have no edges between them (so that there are only p edges in the graph). We assume that all edges go from black vertices to white vertices in the bipartite graph. It is clear that the optimal solution would be to take the p black vertices involved in the perfect matching, M. Let us call these vertices M_B.

Local search will only add vertices from M_B, and will never remove these vertices. Given a current intermediate solution $S \subseteq M_B$, local search will add another vertex $v \in M_B \setminus S$ if and only if $f(S \cup \{v\}) \geq (1 + \epsilon/n^2) f(S)$. This is equivalent to

$$f_S(v) \geq \frac{\epsilon}{n^2} f(S).$$

Observe that $f_S(v) = 1$ for any S and $v \in M_B$. Assuming that $\epsilon \leq 1$ we see that $\frac{\epsilon}{n^2} f(S) \leq 1$ since $f(S) \leq |S| = \mathcal{O}(n^{1-\epsilon})$, since $S \subseteq M_B$ in the case of this graph. So, local search will add all vertices in M_B and halt once $S = M_B$.

To show that we require at least $\Omega(p)$ rounds, it is sufficient to prove the following claim:

Claim 1. *After round $r \leq p$, $\mathbb{E}[f(S)] \leq r + 1$.*

Proof. We prove the above claim via induction. We use S_i to denote the intermediate solution at round i. Before the first round we assign $S_0 \leftarrow \{v\}$ where $v = \text{argmax} f(\{v\})$. Therefore the claim holds trivially before the first round.

Suppose that the claim holds after round $r-1$. Let X be a random variable which denotes the number of vertices we add to S_r in round r. We can write

$$\mathbb{E}[f(S_r)] = \mathbb{E}[f(S_{r-1}) + X] = \mathbb{E}[f(S_{r-1})] + \mathbb{E}[X] \leq r + \mathbb{E}[X].$$

It remains to calculate $\mathbb{E}[X]$. Let $X_{i,j}$ be a 0-1 random variable which denotes whether or not vertex $j \in M_B \setminus S_{r-1}$ was mapped to machine i. The number of elements from $M_B \setminus S_{i-1}$ on machine i is $X_i = \sum_{j \in M_B \setminus S_{i-1}} X_{i,j}$. Observe that the number of elements we add to S_{r-1} in round r is $\max_{i=1}^{p} \{X_i\}$. Furthermore, we send each element $j \in M_B \setminus S_{r-1}$ to machine i with probability $(1/p)$, and $|M_B \setminus S_{r-1}| \leq p$, so

$$\mathbb{E}[X_i] = \sum_{j \in M_B \setminus S_{r-1}} \mathbb{E}[X_{i,j}] \leq 1.$$

Therefore, at round r we expect to add at most 1 element to our current solution, and the result follows. ◀

Since DLS will add all p elements in M_B, and for every round r we expect $f(S) \leq r + 1$, we see that we require $\Omega(p)$ rounds when we have p machines. Therefore, for there to be any hope of DLS to fit into $\mathcal{MRC}^k$ for some k, we will require that $p \in \mathcal{O}(\log^k n)$.

2.3 Upper Bounding the Number of Rounds

Now we provide several attempts to provide an upper bound on the number of rounds of DLS. We begin with some notation:

- Let $\mathsf{LS}(V, S)$ denote running the Deterministic Local Search Procedure on the subuniverse $V \subseteq U$ and beginning from the initial seed solution S.
- If $\{U_1, U_2, \ldots, U_p\}$ is a uniform random partition of U and S is some initial seed, then let $X_i \equiv \mathsf{LS}(U_i, S)$ be the resultant set of running local search starting from S with respect to the sub-universe U_i.
- Let $\delta_S(x)$ be the indicator function which tests whether x is a member of S. If $x \in S$ then $\delta_S(x) = 1$, and otherwise is 0. Let $\mathbf{1}_S$ be the characteristic vector of a set S.
- As a result of the example in section 2.2, we will assume from this point forward that $p \in \mathcal{O}(\log^k n)$ for some $k \in \mathbb{N}$.

Approach 1. In order to bound the number of rounds of DLS, it is tempting to examine the sequence of additions and removals made by running LS on the entire universe (that is, in a centralized setting) and attempt to argue that at least one machine makes some non-trivial fraction of these improvements. That is, examine $\mathsf{LS}(\mathcal{U}, S_{r-1})$ and $X_i = \mathsf{LS}(U_i, S_{r-1})$ for each round r. However, it is hard to proceed this way for the following reason. If $\mathsf{LS}(\mathcal{U}, S_{r-1})$ begins by making a additions, it is clear that at least one machine can make a/p additions by submodularity. This is true because there must exist a machine with at least a/p of these added items. So, for any element, x, of these a/p items belonging to one machine, we can argue that the set we are adding x to on this machine is a subset of the set x was added to in the centralized setting. By submodularity, we can add x to the solution on this machine.

If $\mathsf{LS}(\mathcal{U}, S_{r-1})$ follows the a initial additions by making r removals, then nothing can be said about the action of any machine on any sub-universe. This is because at this point each machine's solution is a subset of the centralized solution, by the argument in the above paragraph. But by submodularity, removing elements is marginally better with respect to larger sets. Therefore it is not immediate that any machine makes even a fraction of the r removals made in the centralized setting. At this point, there is no subset-superset relation between the partial solutions of $\mathsf{LS}(\mathcal{U}, S_{r-1})$ and $\mathsf{LS}(U_i, S_{r-1})$, making any further arguments along this track inherently difficult.

Approach 2. The next approach might be to examine $\mathbb{E}[f(S_r)]$ after each round r, or $\mathbb{E}[f(S_r) - f(S_{r-1})]$ to try and gauge the expected amount of progress made. We present what might be the base case in an inductive argument of this sort. It is unclear, however, how to extend the analysis past this point.

Definition 6. *The convex extension (equivalently Lovasz extension), $f^- : [0,1]^{\mathcal{U}} \to \mathbb{R}_{\geq 0}$, of a submodular function, f, is given by: $f^-(\mathbf{x}) = \min_{\mathcal{D}} \mathbb{E}_{R \sim \mathcal{D}}[f(R)]$, where the minimum is taken over all distributions such that $\mathbb{E}[\mathbf{1}_R] = \mathbf{x}$.*

We now outline some basic properties of the convex extension which will be useful for us:

Lemma 2 ([3])**.** *The convex extension, f^- satisfies the following properties:*

1. *$f^-(\mathbf{1}_S) = f(S)$ for every $S \subseteq \mathcal{U}$.*
2. *f^- is convex.*
3. *$f^-(c \cdot \mathbf{x}) \geq c \cdot f^-(\mathbf{x})$ for any $c \in [0,1]$.*

Below is an interesting property of any submodular function:

Lemma 3 ([3])**.** *Given a submodular function, f, (not necessarily monotone), the function g, given by $g(S) = f(S \cap OPT)$ is monotone, where $OPT = \text{argmax}_{T \subseteq \mathcal{U}}\{f(T)\}$.*

Therefore, we can obtain a *monotone* submodular function from any non-monotone submodular function.

Lemma 4. *After round 1, $\mathbb{E}[f(X_i)] \geq (\frac{1}{3p} - \frac{\epsilon}{n}) f(OPT)$, where $OPT = \text{argmax}_{S \subseteq \mathcal{U}}\{f(S)\}$ and the expectation is taken over the choice of random partition of $\mathcal{U}$.*

Proof. Before the first round we randomly partition $\mathcal{U} \setminus \{u\}$ into $\{\mathcal{U}_i\}_{i=1}^p$. Recall that in the first round, X_i is the solution of running Local Search starting from $S = \{u\}$ on the sub-universe $\mathcal{U}_i$. Therefore, after Local Search on this subuniverse terminates, S is a $(1 + \frac{\epsilon}{n^2})$-approximate local optimum with respect to $\mathcal{U}_i \cup S$. Using this fact, we can apply the analysis in the proof of **Theorem**

3.4 in [4] to see that $f(X_i) \geq (\frac{1}{3} - \frac{\epsilon}{(n/p)})f(OPT_{\mathcal{U}_i \cup S})$, where $OPT_{\mathcal{U}_i \cup S} = \text{argmax}_{T \subseteq \mathcal{U}_i \cup S}\{f(T)\}$ and (n/p) is the expected size of $\mathcal{U}_i$.

$$
\begin{aligned}
\mathbb{E}[f(X_i)] &\geq (\frac{1}{3} - \frac{\epsilon}{(n/p)})\mathbb{E}[f(OPT_{\mathcal{U}_i \cup S})] && X_i \text{ is a } (\frac{1}{3} - \frac{\epsilon}{(n/p)})\text{-approx. local optimum,} \\
&\geq (\frac{1}{3} - \frac{\epsilon}{(n/p)})\mathbb{E}[f((\mathcal{U}_i \cup S) \cap OPT)] && OPT_{\mathcal{U}_i} \text{ is optimal over } \mathcal{U}_i, \\
&\geq (\frac{1}{3} - \frac{\epsilon}{(n/p)})\mathbb{E}[f(\mathcal{U}_i \cap OPT)] && \text{by Lemma 3,} \\
&= (\frac{1}{3} - \frac{\epsilon}{(n/p)})\mathbb{E}[f^-(\mathbf{1}_{OPT \cap \mathcal{U}_i})] && \text{property of the convex extension,} \\
&\geq (\frac{1}{3} - \frac{\epsilon}{(n/p)})f^-(\mathbb{E}[\mathbf{1}_{OPT \cap \mathcal{U}_i}]) && \text{Jensen's inequality,} \\
&= (\frac{1}{3} - \frac{\epsilon}{(n/p)})f^-(\mathbb{E}[\sum_{e \in OPT} \delta_{\mathcal{U}_i}(e) \cdot \mathbf{1}_{\{e\}}]) \\
&= (\frac{1}{3} - \frac{\epsilon}{(n/p)})f^-(\sum_{e \in OPT} \mathbf{1}_{\{e\}} \cdot \mathbb{E}[\delta_{\mathcal{U}_i}(e)]) \\
&= (\frac{1}{3} - \frac{\epsilon}{(n/p)})f^-(\frac{1}{p} \cdot \mathbf{1}_{OPT}) \\
&\geq (\frac{1}{3} - \frac{\epsilon}{(n/p)})(\frac{1}{p}) \cdot f^-(\mathbf{1}_{OPT}) && \text{by Lemma 2,} \\
&= (\frac{1}{3p} - \frac{\epsilon}{n})f(OPT) && \text{by Lemma 2.}
\end{aligned}
$$

◀

Remark 1. This lemma uses no details of the algorithm being used. Presumably to extend this lemma to an inductive claim regarding $\mathbb{E}[f(S_r)]$ for any round r, some facts regarding how the algorithm operates would be used. The initial hope for this type of analysis was to show that after $R \in \mathcal{O}(\log^j n)$ rounds, we would have $\mathbb{E}[f(S_R)] \geq (\frac{1}{3} - \frac{\epsilon}{n})f(OPT)$ and that **Lemma 4** would be a base case in an inductive argument.

Lemma 4 has an immediate corollary.

Corollary 1. *If $\mathcal{A}$ is any algorithm which provides an α approximation to unconstrained, non-monotone, submodular maximization, then running one round of* DLS *by replacing* LS *with $\mathcal{A}$, we may achieve an $\frac{\alpha}{p}$-approximation.*

Approach 3. In this section, we take a look at the result of our experiments and conjecture a statement regarding the amount of gain our solution makes in terms of function value at each round. We begin by observing that interestingly, in each of our experiments, DLS approaches the value that centralized local search attained exponentially fast. Furthermore, the value of DLS seems to approach the final value of centralized local search, but remains below it. Let us denote the value which centralized local search attains on the universe $\mathcal{U}$ as CLS.

It is interesting to analyze the "gap" between our current solution value and our goal:

$$\mathsf{CLS} - f(S_r).$$

We conjecture that with high probability, we can make up some constant fraction of the gap remaining from our current solution to CLS at each round. Formally:

Conjecture 1. *There exists a constant c such that for any machine i and previous solution S_r, $\Pr\big[f(X_i) < \frac{1}{c}(\mathsf{CLS} - f(S_{r-1})) + f(S_{r-1})\big] < d$ for some $d < 1$. Recall that $X_i = \mathsf{LS}(\mathcal{U}_i, S_{r-1})$.*

Supposing the above conjecture were true, then we would be able to explain our experiments. Recall that $S_r = \text{argmax}_{i=1}^{p}\{f(X_i)\}$. Then:

$$\begin{aligned}\Pr\left[f(S_r) < \frac{1}{c}(\mathsf{CLS} - f(S_{r-1})) + f(S_{r-1})\right] &= \prod_{i=1}^{p} \Pr\left[f(X_i) < \frac{1}{c}(\mathsf{CLS} - f(S_{r-1})) + f(S_{r-1})\right] \\ &< d^{\mathcal{O}(\log^k n)} \\ &\leq \frac{1}{n}.\end{aligned}$$

Therefore, we can gain a constant factor of the gap between CLS and our current solution at each round. This implies that with high probability we need only a logarithmic number of rounds to converge to CLS.

2.4 Distributed Smooth Local Search

In this section we make a few remarks regarding the parallelization of Smooth Local Search (SLS). Instead of deterministically finding an approximate local optimum, smooth local search finds such a set via random biased sampling of the universe. Recall that SLS achieves a (2/5)-approximation to USM, and is a natural extension to deterministic local search. The hope was to easily parallelize deterministic local search, and then naturally extend the parallelization to smooth local search.

Definition 7 ([4])**.** *A set A is sampled with bias δ if elements in A are sampled with probability $(1+\delta)/2$, and elements outside of A are sampled with probability $(1-\delta)/2$. We denote this random set by $\mathcal{R}(A, \delta)$.*

Formally, SLS maximizes the function $\mathbb{E}[f(\mathcal{R}(S, \delta))]$ for the set S. This can also be seen as a maximization over the multilinear extension of f; see [4] for more details.

Algorithm: Smooth Local Search (SLS)
Input: Submodular function $f : 2^{\mathcal{U}} \to \mathbb{R}_{\geq 0}$, $\delta, \delta' \in [-1, 1]$

1. $S \leftarrow \emptyset$.
2. Set $\omega_{S,\delta}(x) \equiv \mathbb{E}[f(\mathcal{R}(S, \delta) \cup \{x\}) - f(\mathcal{R}(S, \delta) \setminus \{x\})]$ for each $x \in \mathcal{U}$. Obtain $\hat{\omega}_{S,\delta}(x) \in [\omega_{S,\delta}(x) - \frac{1}{n^2}OPT, \omega_{S,\delta}(x) + \frac{1}{n^2}OPT]$ by repeated sampling.
3. If there exists $x \in \mathcal{U} \setminus S$ with $\hat{\omega}_{S,\delta}(x) > \frac{2}{n^2}OPT$, add x to S. Return to step 2.
4. If there exists $s \in S$ with $\hat{\omega}_{S,\delta}(x) < -\frac{2}{n^2}OPT$, remove s from S. Return to step 2.
5. Return $\mathcal{R}(S, \delta')$.

We remark that an efficient (polynomial time) approximation of $\omega_{S,\delta}(x)$ is guaranteed by **Lemma 2.5** in [4].

It is tempting to try and proceed along the same lines to parallelize SLS as was done with DLS. However, the estimation of $\omega_{S,\delta}(x)$ becomes a problem. The authors of SLS show in the proof of **Theorem 3.6** [4] that the algorithm may update the solution up to n^2/δ times. After each update

we must recompute $w_{S,\delta}$, which is done by repeated sampling across the universe $\mathcal{U}$. However, each time we do so, we would require a round of MapReduce. Therefore, this is an infeasible way of parallelizing SLS. From here, it is perhaps natural to want to estimate $\omega_{S,\delta}(x)$ with respect to the elements on each machine independently. This would allow each machine to make many improvements per round and would make this parallelization more similar to DLS. However, it is unclear how to relate these local estimates to a global estimate over $\mathcal{U}$. This poses an obvious problem for the termination of any distributed implementation, as it is not clear that if no machine can make any progress that the same is true for the centralized algorithm. We leave this exploration for future work, once we are able to understand how to bound the number of rounds of DLS.

3 Double Greedy Exploration

In this section, we explore trying to use the Sample and Prune method [8] to parallelize the Double Greedy algorithm [2]. We show that the trivial idea of plugging the Double Greedy algorithm into the Sample and Prune method does not work by providing an example for which it does not satisfy a requisite property of Sample and Prune.

Kumar et al. [8] present a framework for parallelizing Greedy algorithms. This is done by making use of the Sample and Prune algorithm:

Algorithm: Sample and Prune
Input: A set $\mathcal{U}$, Greedy algorithm G, parameter ℓ.

$X \leftarrow$ random subset of $\mathcal{U}$ where each point sampled with probability $\min\{1, \frac{\ell}{|\mathcal{U}|}\}$.

$S \leftarrow G(X)$.

$M_S \leftarrow \{u \in \mathcal{U} \setminus S \ : \ u \in G(S \cup \{u\})\}$.

Return (S, M_S).

To parallelize, at each round we use G to obtain a seed solution on a random sample of the universe. We then remove all elements of the universe which are incompatible with our current solution (i.e. elements of $\mathcal{U}$ which we would never choose moving forward given our current solution S). In order to bound the number of rounds in the above algorithm, Kumar et. al relies on the fact that G satisfies $G(A) \subseteq G(B)$ whenever $A \subseteq B$. The authors massage the $\epsilon - Greedy$ algorithm for Submodular Maximization subject to a matroid constraint into the Sample and Prune framework in order to obtain an algorithm which fits in the $\mathcal{MRC}$ framework.

The Double-Greedy [2] algorithm is a procedure which provides a (1/3)-approximation to non-monotone, unconstrained submodular maximization (and a (1/2)-approximation with a randomized variant). A variant of the Greedy algorithm is to iteratively pass through the input universe and add an element if it's marginal value is positive with respect to the current solution. This variant of Greedy applied to non-monotone unconstrained submodular maximization performs rather poorly. Similarly, one can define a procedure, Reverse Greedy, which begins with the seed solution being the universe $\mathcal{U}$, and removes elements if their removal results in a positive gain. Again, this procedure performs poorly as well. The idea of the Double-Greedy algorithm is to combine these ideas. With one iterative pass through the algorithm, we maintain solutions for the Greedy algorithm and the Reverse greedy algorithm, where we add elements if the marginal of adding an element in the Greedy algorithm at a step exceeds the marginal of removing that same element in the Reverse

greedy algorithm at the same step. Formally:

Algorithm: Double-Greedy
Input: A submodular function f, a universe $\mathcal{U}$.

$X_0 \leftarrow \emptyset$, $Y_0 \leftarrow \mathcal{U}$.

For $i = 1$ to n, do:

$a_i \leftarrow f_{X_{i-1}}(u_i)$, $b_i \leftarrow -f_{Y_{i-1}\setminus\{u_i\}}(u_i)$.

If $a_i \geq b_i$:

$X_i \leftarrow X_{i-1} \cup \{u_i\}$, $Y_i \leftarrow Y_{i-1}$.

Else:

$X_i \leftarrow X_{i-1}$, $Y_i \leftarrow Y_{i-1} \setminus \{u_i\}$.

Return X_n.

Since the Double-Greedy [2] algorithm provides a (1/3)-approximation to non-monotone, unconstrained submodular maximization (and a (1/2)-approximation with a randomized variant), one would be tempted to simply plug the Double-Greedy algorithm into the Sample and Prune algorithm as G. We provide an example below to show that this is not possible because $DG(A) \not\subseteq DG(B)$ in some cases when $A \subseteq B$. This implies that one must be more clever in order to fit DoubleGreedy into $\mathcal{MRC}$.

Indeed, taking $\mathcal{U} = \{u_1, u_2\}$ and $f(\emptyset) = 0,\ f(u_1) = 1,\ f(u_2) = 1,\ f(u_1, u_2) = -5$ and $A = \{u_1\},\ B = \{u_1, u_2\}$ we see that $u_1 \in DG(A)$ but $u_1 \notin DG(B)$.

Acknowledgements. We thank Dr. Harvey for taking the time to help us talk through ideas and Chris Liaw for a helpful discussion.

References

[1] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1433–1452, 2014.

[2] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. A tight linear time 1/2-approximation for unconstrained submodular maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2014.

[3] Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. *CoRR*, abs/1502.02606, 2015.

[4] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrak. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40.4:1133–1153, 2011.

[5] Karloff Howard, Siddharth Suri ad, and Sergei Vassilvitskii. A model of computation for mapreduce. *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948, 2010.

[6] Andreas Krause. Sfo: A toolbox for submodular function optimization. *Journal of Machine Learning Research*, 2010.

[7] Andreas Krause and Daniel Golovin. Survey: Submodular function maximization. `https://las.inf.ethz.ch/files/krause12survey.pdf`.

[8] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.*, 2(3):14:1–14:22, September 2015.

[9] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discrete Math*, 23(4):2053–2078, 2010.

[10] Baharan Mirzasoleimann, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization. *Journal of Machine Learning Research*, pages 1–41, 2015.