DASH: A Distributed and Parallelizable Algorithm for Size-Constrained Submodular Maximization

Tonmoy Dey[1], **Yixin Chen**[2], **Alan Kuhnle**[2]

[1] Department of Computer Science, Florida State University
[2] Department of Computer Science & Engineering, Texas A&M University
tdey@fsu.edu, chen777@tamu.edu, kuhnle@tamu.com

Abstract

MapReduce (MR) algorithms for maximizing monotone, submodular functions subject to a cardinality constraint (SMCC) are currently restricted to the use of the linear-adaptive (non-parallelizable) algorithm GREEDY. Low-adaptive algorithms do not satisfy the requirements of these distributed MR frameworks, thereby limiting their performance. We study the SMCC problem in a distributed setting and propose the first MR algorithms with sublinear adaptive complexity. Our algorithms, R-DASH, T-DASH and G-DASH provide $0.316-\varepsilon$, $3/8-\varepsilon$, and $1-1/e-\varepsilon$ approximation ratios, respectively, with nearly optimal adaptive complexity and nearly linear time complexity. Additionally, we provide a framework to increase, under some mild assumptions, the maximum permissible cardinality constraint from $O(n/\ell^2)$ of prior MR algorithms to $O(n/\ell)$, where n is the data size and ℓ is the number of machines; under a stronger condition on the objective function, we increase the maximum constraint value to n. Finally, we provide empirical evidence to demonstrate that our sublinear-adaptive, distributed algorithms provide orders of magnitude faster runtime compared to current state-of-the-art distributed algorithms.

1 Introduction

Submodular maximization has become an important problem in data mining and machine learning with real world applications ranging from influence and revenue maximization in social networks to more complex tasks such as image and video summarization. A submodular function captures the diminishing gain property of adding an element to a set that decreases with increase in size of the set. Formally, a non-negative set function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ is submodular iff for all sets $A \subseteq B \subseteq \mathcal{N}$ and $x \in \mathcal{N}\backslash B$, $f(A \cup x) - f(A) \geq f(B \cup x) - f(B)$. The submodular function is monotone if $f(A) \leq f(B)$ for all $A \subseteq B$. In this paper, we study the following optimization problem for submodular optimization under cardinality constraint (SMCC):

$$O \leftarrow \arg\max_{S \subseteq \mathcal{N}} f(S); \text{ subject to } |S| \leq k \qquad (1)$$

Classical results of Nemhauser, Wolsey, and Fisher (1978); Nemhauser and Wolsey (1978) show that a standard greedy algorithm achieves optimal approximation ratio of $1 - 1/e$.

Challenges from Big Data. Data from different sources and modalities are growing at an unprecedented rate. Such massive data sets are often too large to fit on a single machine and call for techniques that can work on data that is distributed over many machines. The MapReduce (MR) model is a formalization of distributed computation into MapReduce rounds of communication between machines. A dataset of size n is distributed on ℓ machines, each with memory to hold at most Ψ elements of the ground set. The total memory of the machines is constrained to be $\Psi \cdot \ell = O(n)$. After each round of computation, a machine may send $O(\Psi)$ amount of data to other machines. We assume $\ell \leq n^{1-c}$ for some constant $c \geq 1/2$.

MapReduce Algorithms for SMCC. Motivated by the growing data size, extensive effort has been made to develop distributed algorithms for SMCC using the MapReduce (MR) framework, *e.g.* Mirzasoleiman et al. (2013); Mirrokni and Zadimoghaddam (2015); Barbosa et al. (2016); Kazemi et al. (2021) (see Table 1). The proposed distributed algorithms of Barbosa et al. (2015) and Barbosa et al. (2016) provide a way to utilize a given centralized, approximation algorithm ALG in a distributed setting, where ALG is executed on the distributed data over a constant number of MR rounds to yield a constant approximation. In order for their theoretical guarantees to hold, ALG must satisfy a consistency property (formally defined in Property 1). The standard greedy GREEDY algorithm (Nemhauser, Wolsey, and Fisher 1978) and the continuous greedy algorithm (Calinescu et al. 2007) are the only known algorithms to satisfy this property and achieve constant approximation in a distributed setting.

Parallelizable (Adaptive) Algorithms for SMCC. Recently, parallelizable algorithms for SMCC in the *adaptive complexity* model have been extensively studied (Balkanski, Rubinstein, and Singer 2019b; Ene and Nguyen 2019; Fahrbach, Mirrokni, and Zadimoghaddam 2019; Chekuri and Quanrud 2019; Balkanski and Singer 2018; Breuer, Balkanski, and Singer 2019; Balkanski, Rubinstein, and Singer 2019a; Chen, Dey, and Kuhnle 2021; Kuhnle 2021b,a; Kazemi et al. 2019). In this model, queries to the objective function f are organized into as few *adaptive rounds* as possible; within each adaptive round, the function queries are independent and thus may be arbitrarily parallelized. Several algorithms have been proposed that are

Reference	Ratio	MR Rounds	Adaptivity (Θ)	Queries (Ξ)	k_{max}	Parallel Runtime (Π)
RG [4]	$\frac{1}{2}(1-\frac{1}{e})$	$\mathbf{1}$	$O(k)$	$O(nk+k^2\ell)$	$O(\frac{n}{\ell^2})$	$O(\frac{nk}{\ell}+k^2\ell)$
PALG [5]	$\mathbf{1-\frac{1}{e}-\varepsilon}$	$O(\frac{1}{\varepsilon^2})$	$O(\frac{k}{\varepsilon^2})$	$O(\frac{nk}{\varepsilon^2}+\frac{k^2\ell^2}{\varepsilon^4})$	$O(\frac{n\varepsilon^2}{\ell^2})$	$O(\frac{nk}{\ell\varepsilon}+\frac{k^2\ell}{\varepsilon^3})$
DDIST [13]	$\mathbf{1-\frac{1}{e}-\varepsilon}$	$O(\frac{1}{\varepsilon})$	$O(\frac{k}{\varepsilon})$	$O(\frac{nk}{\varepsilon}+\frac{k^2\ell^2}{\varepsilon^2})$	$O(\frac{n\varepsilon}{\ell^2})$	$O(\frac{nk}{\ell\varepsilon}+\frac{k^2\ell}{\varepsilon^2})$
BCG [11]	$1-\varepsilon$	r	$O\left(\frac{rk}{\varepsilon^{2/r}}\log^2(\frac{1}{\varepsilon^{1/r}})\right)$	$O(\frac{nkr}{\varepsilon^{1/r}}+\frac{k^2\ell r m^2(\varepsilon^{1/r})}{\varepsilon^{3/r}})$	$O(\frac{n\varepsilon^{1/r}}{\ell^2})$	$O(\frac{nkr}{\ell\varepsilon^{1/r}}+\frac{k^2\ell r m^2(\varepsilon^{1/r})}{\varepsilon^{3/r}})$
PG [17]	$0.545-\varepsilon$	$\mathbf{1}$	$O(k^2)$	$O(nk+k^3)$	$O(\frac{n}{\ell^2})$	$O(\frac{nk}{\ell}+k^3)$
R-DASH	$\frac{1}{2}(1-\frac{1}{e}-\varepsilon)$	$\mathbf{1}$	$O(\frac{\log(k)}{\varepsilon^2}(\log(\frac{n}{\ell})+\log(k\ell)))$	$\boldsymbol{O(\frac{\log(k)}{\varepsilon^2}(n+k\ell))}$	$O(\frac{n}{\ell^2})$	$\boldsymbol{O(\frac{\log(k)}{\varepsilon^2}(\frac{n}{\ell}+k\ell))}$
G-DASH	$\mathbf{1-\frac{1}{e}-\varepsilon}$	$O(\frac{1}{\varepsilon})$	$O\left(\frac{\log(k)}{\varepsilon^3}\log(\frac{n}{\ell}+\frac{k\ell}{\varepsilon})\right)$	$O(\frac{\log(k)}{\varepsilon^3}(n+\frac{k\ell^2}{\varepsilon}))$	$O(\frac{n\varepsilon}{\ell^2})$	$O(\frac{\log(k)}{\varepsilon^3}(\frac{n}{\ell}+\frac{k\ell}{\varepsilon}))$
T-DASH	$\frac{3}{8}-\varepsilon$	$\mathbf{1}$	$\boldsymbol{O(\frac{1}{\varepsilon}(\log(\frac{n}{\ell})+\log(k\ell)))}$	$\boldsymbol{O(\frac{\log(k)}{\varepsilon^2}(n+k\ell))}$	$O(\frac{n}{\ell^2\log(k)})$	$\boldsymbol{O(\frac{\log(k)}{\varepsilon^2}(\frac{n}{\ell}+k\ell))}$
MED	$1-e^{-\gamma}$	m	$m\cdot\Theta\left(\text{ALG}(n,\frac{k}{m})\right)$	$m\cdot\Xi\left(\text{ALG}(n,\frac{k}{m})\right)$	$\boldsymbol{n}, \boldsymbol{\frac{n}{\ell-1}}$	$O(\frac{m}{\ell}\Pi(\text{ALG}(n,k)))$

Table 1: Theoretical comparison of our algorithms to the following MapReduce model algorithms: RANDGREEDI (RG) of Barbosa et al. (2015), PARALLELALG (PALG) of Barbosa et al. (2016), BICRITERIAGREEDY (BCG) of Epasto, Mirrokni, and Zadimoghaddam (2017), DISTRIBUTEDDISTORTED (DDIST) of (Kazemi et al. 2021) and PSEUDOGREEDY of Mirrokni and Zadimoghaddam (2015). Theoretical guarantees provided for MED are when running a γ-approximate subroutine ALG over m iterations. Θ, Ξ and Π represents the adaptivity, query complexity, and the parallel runtime of the algorithm. The column k_{max} provides the maximum cardinality constraint that the algorithm can compute. For MED, the listed k_{max} values are achievable under certain conditions.

nearly optimal in terms of ratio, adaptivity, and total query calls with $O(\log(n))$ adaptivity and $O(n\text{polylog}(n))$ query complexity (Balkanski, Rubinstein, and Singer 2019b; Ene and Nguyen 2019; Fahrbach, Mirrokni, and Zadimoghaddam 2019; Chekuri and Quanrud 2019; Breuer, Balkanski, and Singer 2019; Chen, Dey, and Kuhnle 2021). LS+PGB of Chen, Dey, and Kuhnle (2021) is the current state-of-the-art adaptive algorithm that obtains nearly the optimal $1-1/e-\varepsilon$ approximation in $O(\log(n))$ adaptive rounds and $O(n)$ query complexity in expectation. To the best of our knowledge, all algorithms with sublinear adaptivity require randomized access to the entire ground set in each processor and hence are ill-suited to a distributed scenario. Moreover, no low-adaptive algorithm is known to satisfy the consistency properties and hence cannot be used in the existing MapReduce frameworks.

Combining Parallelizability and Distributed Settings. Next, we motivate why one would want to combine the MapReduce and adaptive complexity models. In the existing MapReduce algorithms, the number of processors ℓ can be made large, which may lead one to suggest setting ℓ to the number of processors available to ensure the usage of all available resources. However, there are a number of disadvantages to this approach: 1) A large number of machines severely restricts the size of the cardinality constraint k – since, in all of the MapReduce algorithms, a set of size $k\ell$ must be stored on a single machine; therefore, we must have $k \leq O(\Psi/\ell) = O(n/\ell^2)$. For example, with $n = 10^6$, $\ell = 256$ implies that $k \leq 15 \cdot C$, for some C. However, if each machine has 8 processor cores, the number of machines ℓ can be set to 32 (and then parallelized on each machine), which enables the use of 256 processors with $k \leq 976 \cdot C$. As the number of cores and processors per physical machine continues to grow, large practical benefits can be obtained by a parallelized and distributed algorithm. 2) The total time complexity (that is, the sum of time taken by each machine) of existing MR algorithms is at least $\Omega(nk)$ in the worst case, which does not scale well for large k values. 3) Finally, in a practical sense, modern CPU architectures have many cores that all share a common memory, and it is inefficient to view communication between these cores to be as expensive as communication between separate machines in a cluster.

Thus, the following questions are posed: *Q1: Is it possible to design constant-factor approximation distributed algorithms with 1) constant number of MapReduce rounds; 2) sublinear adaptive complexity (highly-parallelizable); and 3) nearly linear total runtime? Q2: Can we design practical distributed algorithms, preferably with 1) one MR round; 2) sublinear adaptivity; 3) nearly linear total runtime; and 4) parallel runtime that scales linearly with the number of machines?*

Contributions. Our first contribution is an analysis of variants of two, recently introduced, low-adaptive algorithms of Chen, Dey, and Kuhnle (2021) (LAT and LAG, Section 2). We show these algorithms satisfy the *randomized consistency property*:

Property 1 (Randomized Consistency Property of Barbosa et al. (2016))**.** *Let* $\mathbf{q}$ *be a fixed sequence of random bits; and let* ALG *be a randomized algorithm with randomness determined by* $\mathbf{q}$*. Suppose* A *and* B *are disjoint subsets of* $\mathcal{N}$*, and that for each* $b \in B$*,* $b \notin \text{ALG}(A \cup \{b\}, \mathbf{q})$*. Also, suppose that* $\text{ALG}(A, \mathbf{q})$ *terminates successfully. Then* $\text{ALG}(A \cup B, \mathbf{q})$ *terminates successfully and* $\text{ALG}(A \cup B, \mathbf{q}) = \text{ALG}(A, \mathbf{q})$*.*

This property enables the use of LAT and LAG in the design of low-adaptive, distributed algorithms. First, we show in Section 2 that LAG works in the DISTRIBUTEDDISTORTED

framework of Kazemi et al. (2019), which yields the algorithm G-DASH that achieves nearly optimal ratio, runtime, and adaptive complexity in $O(1/\varepsilon)$ MR rounds. More generally, we show in the Appendix that any ALG satisfying the randomized consistency property can be used in the DISTRIBUTEDDISTORTED framework to obtain an $O(1/\varepsilon)$ MR round algorithm without loss of approximation.

Next, we use LAG and LAT to develop R-DASH and T-DASH, which are nearly linear-time, single-round MR algorithms with $O(\log(k)\log(n))$ and $O(\log n)$ adaptivity, respectively, with approximation ratios of $\frac{1}{2}(1 - 1/e - \varepsilon)$ ($\simeq 0.316$) and $3/8 - \varepsilon$. R-DASH is our most practical algorithm and may be regarded as a parallelized version of RANDGREEDI, the first MapReduce algorithm for SMCC and the state-of-the-art MR algorithm in terms of empirical performance. T-DASH is a novel algorithm that improves the theoretical properties of R-DASH at the cost of empirical performance (see Section 5). Notably, the adaptivity of T-DASH is close to the best known adaptivity for a constant factor algorithm of Chen, Dey, and Kuhnle (2021), which has adaptivity of $O(\log(n/k))$.

Our next contribution is MED, a general plug-in framework for distributed algorithms, which increases the size of the maximum cardinality constraint at the cost of more MR rounds. As discussed above, the maximum k value of any prior MR algorithm for SMCC is $O(n/\ell^2)$, where ℓ is the number of machines; MED increases this to $O(n/\ell)$. We also show that under certain assumptions on the objective function (which are satisfied by all of the empirical applications evaluated in Section 5), MED can be run with $k_{max} = n$, which removes any cardinality restrictions. When used in conjunction with a γ-approximation algorithm, MED provides an $(1 - e^{-\gamma})$-approximate solution.

Finally, an extensive empirical evaluation across four applications of our algorithms and the current state-of-the-art shows that R-DASH is orders of magnitude faster than state-of-the-art MR algorithms and demonstrates an exponential improvement in scaling with larger k. Moreover, we show that MR algorithms slow down with increasing number of machines past a certain point, even if enough memory is available, which further motivates distributing a parallelizable algorithm over smaller number of machines.

Notation. We use $\Delta(x \mid S)$ to denote the marginal game of element x to set S: $\Delta(x \mid S) = f(S \cup \{x\}) - f(S)$.

2 Consistent Low-Adaptive Procedures

Motivation. *Is sublinear adaptivity possible for an MR algorithm with constant MR rounds?* In this section, we analyze two low-adaptive procedures, LAT (Alg. 1) and LAG (Alg. 2), variants of low-adaptive procedures proposed in Chen, Dey, and Kuhnle (2021). This analysis enables their use in the distributed, MapReduce setting. For convenience, we regard the randomness of the algorithms to be determined by a sequence of random bits $\mathbf{q}$, which is an input to each algorithm.

Observe that the randomness of both of LAT and LAG only comes from the random permutations of V_j on Line 8 of LAT, since LAG employs LAT as a subroutine. Consider an equivalent version of LAT in which the entire ground set $\mathcal{N}$ is permuted randomly, from which the permutation of V_j is extracted. That is, if σ is the permutation of $\mathcal{N}$, the permutation of V is given by $v < w$ iff $\sigma(v) < \sigma(w)$, for $v, w \in V$. Then, the random vector $\mathbf{q}$ specifies a sequence of permutations of $\mathcal{N}$: $(\sigma_1, \sigma_2, \ldots)$, which completely determines the behavior of both algorithms.

Algorithm 1: Low-Adaptive Threshold Algorithm (LAT)

1: **procedure** LAT($f, X, k, \delta, \varepsilon, \tau, \mathbf{q}$)
2: **Input:** evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, subset $X \subseteq \mathcal{N}$, constraint k, confidence δ, error ε, threshold τ, fixed sequence of random bits $\mathbf{q} \leftarrow (\sigma_1, \ldots, \sigma_{M+1})$
3: Initialize $S_0 \leftarrow \emptyset$, $V_0 \leftarrow X$, $M \leftarrow \lceil 4(1 + 2/\varepsilon)\log(n/\delta)\rceil$
4: **for** $j \leftarrow 1$ to $M + 1$ **do**
5: Update $V_j \leftarrow \{x \in V_{j-1} : \Delta(x \mid S_{j-1}) \geq \tau\}$ and filter out the rest
6: **if** $|V_j| = 0$ or $|S_{j-1}| = k$ **then**
7: **return** S_{j-1}, success
8: $V_j \leftarrow$ **random-permutation**(V_j, σ_j).
9: $s \leftarrow \min\{k - |S_{j-1}|, |V_j|\}$
10: $\Lambda \leftarrow \{\lfloor (1+\varepsilon)^u \rfloor : 1 \leq \lfloor (1+\varepsilon)^u \rfloor \leq s, u \in \mathbb{N}\} \cup \{s\}$
11: $B \leftarrow \emptyset$
12: **for** $\lambda \in \Lambda$ in parallel **do**
13: $T_\lambda \leftarrow \{v_1, v_2, \ldots, v_\lambda\}$
14: **if** $\Delta(T_\lambda \mid S)/|T_\lambda| \geq (1-\varepsilon)\tau$ **then**
15: $B \leftarrow B \cup \{\lambda\}$
16: $\lambda_j^* \leftarrow \min\{\lambda \in \Lambda : \lambda > b, \forall b \in B\}$
17: $S_j \leftarrow S_{j-1} \cup T_{\lambda^*}$
18: **return** S_{M+1}, failure

2.1 Low-Adaptive Threshold (LAT) Algorithm

This section presents the analysis of the low-adaptive threshold algorithm, LAT (Alg. 1; a variant of THRESHOLDSEQ from Chen, Dey, and Kuhnle (2021), that incorporates the randomness $\mathbf{q}$ to provide consistency in a distributed setting.

Lemma 1. LAT *satisfies the randomized consistency property (Property 1).*

Proof. Consider that the algorithm runs for all $M + 1$ iterations of the outer **for** loop; if the algorithm would have returned at iteration j, the values S_i and V_i, for $i > j$ keep their values from when the algorithm would have returned. The proof relies upon the fact that every call to LAT$(\cdot, \mathbf{q})$ uses the same sequence of permutations of $\mathcal{N}$: $(\sigma_1, \sigma_2, \ldots, \sigma_{M+1})$. We refer to iterations of the outer **for** loop on Line 4 of Alg. 1 simply as iterations.

We consider the runs of (1) LAT$(A, \mathbf{q})$, (2) LAT$(A \cup \{b\}, \mathbf{q})$, and (3) LAT$(A \cup B, \mathbf{q})$ together. Variables of (1) are given the notation defined in the pseudocode; variables

Algorithm 2: Low-Adaptive Greedy (LAG)

Input: Evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, subset C, constraint k, accuracy parameter ε
$\Gamma \leftarrow \max_{x \in \mathcal{N}} f(x)$
Initialize $\delta \leftarrow 1/(\log_{1-\varepsilon}(1/(3k)) + 1)$, $S \leftarrow \emptyset$, fixed sequence of random bits $(\mathbf{q}_1, \mathbf{q}_2, \ldots)$, $i \leftarrow 0$
for $i \leftarrow 0$ to $\log_{1-\varepsilon}(3k)$ **do**
    $\tau \leftarrow \tau(1-\varepsilon)^i$, $g(\cdot) \leftarrow f(S \cup \cdot)$
    $T \leftarrow \mathrm{LAT}(g, C, k - |S|, \delta, \varepsilon/3, \tau, \mathbf{q}_i)$
    $S \leftarrow S \cup T$
    if $|S| = k$ **then**
        return S
return S

of (2) are given the superscript b; and variables of (3) are given the superscript $'$.

Let $P(i)$ be the statement that

(i) $S_i = S'_i$, and

(ii) for all $b \in B$, $S_i = S_i^b$, and

(iii) $V_i = V'_i \setminus B$, and

(iv) for all $b \in B$, $V_i = V_i^b \setminus \{b\}$.

We prove in the Appendix B that $P(i)$ is true for every $i \leq M+1$. Then, we consider the successful termination of $\mathrm{LAT}(A, \mathbf{q})$ in two cases. First, suppose $\mathrm{LAT}(A, \mathbf{q})$ terminates on iteration $j \leq M+1$ with $|S_{j-1}| = k$. Then by $P(j)$, this termination condition also holds in each of the other runs, so $\mathrm{LAT}(A \cup B, \mathbf{q})$ succeeds and $\mathrm{LAT}(A \cup B, \mathbf{q}) = \mathrm{LAT}(A, \mathbf{q})$. Secondly, suppose $\mathrm{LAT}(A, \mathbf{q})$ terminates on iteration $j \leq M+1$ with $|V_j| = 0$. Then on iteration j of $\mathrm{LAT}(A \cup B, \mathbf{q})$, $V'_j = \emptyset$ as well, and also terminates successfully. Moreover, it returns S'_{j-1}, which is equal to S_{j-1} by $P(j-1)$. □

2.2 Low-Adaptive Greedy (LAG) Algorithm

Another building block for our distributed algorithms is a simple, low-adaptive greedy algorithm LAG (Alg. 2). This algorithm is an instantiation of the PARALLELGREEDYBOOST framework of Chen, Dey, and Kuhnle (2021), and it relies heavily on the low-adaptive procedure LAT (Alg. 1). The analysis for this algorithm can be found in Appendix C.

Guarantees. By Theorem 3 of Chen, Dey, and Kuhnle (2021), setting $\Gamma = \max_{x \in \mathcal{N}} f(\{x\})$ and $\alpha = 1/k$, LAG achieves ratio $1 - 1/e - \varepsilon$ in $O(\log k \log n)$ adaptive rounds and $O(n \log k)$ total queries with probability at least $1 - 1/n$.

Lemma 2. LAG *satisfies the randomized consistency property (Property 1).*

2.3 GREEDY-DASH and RANDOMIZED-DASH

Once we have the randomized consistency property of LAG, we can almost immediately use it to obtain parallelizable MapReduce algorithms.

Algorithm 3: RANDOMIZED-DASH (R-DASH)

Input: Evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}$, constraint k, error ε, available machines $M \leftarrow \{1, 2, ..., \ell\}$
for $e \in \mathcal{N}$ **do**
    Assign e to machine chosen uniformly at random
for $i \in M$ **do**
    ▷ On machine i
    Let $\mathcal{N}_i$ be the elements assigned to machine i
    $S_i \leftarrow \mathrm{LAG}(f, \mathcal{N}_i, k, \varepsilon)$
    Send S_i to primary machine
▷ On primary machine
Gather $S \leftarrow \bigcup_{i=1}^{\ell} S_i$
$T \leftarrow \mathrm{LAG}(f, S, k, \varepsilon)$ on machine m_1
return $V \leftarrow \arg\max \{f(T), f(S_1)\}$

Algorithm 4: GREEDY-DASH(G-DASH)

Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}$, constraint k, error ε, available machines $M \leftarrow \{1, 2, ..., \ell\}$
$S \leftarrow \emptyset, C_0 \leftarrow \emptyset$
for $r \leftarrow 1$ to $\lceil \frac{1}{\varepsilon} \rceil$ do **do**
    $X_{r,i} \leftarrow$ Elements assigned to machine i chosen uniformly at random in round r
    $\mathcal{N}_{r,i} \leftarrow X_{r,i} \cup C_{r-1}$
    for $i \in M$ in parallel **do**
        $S_{r,i} \leftarrow \mathrm{LAG}(f, \mathcal{N}_{r,i}, k, \varepsilon)$
    $S \leftarrow \arg\max \{f(S), f(S_{r,1}), \cdots, f(S_{r,\ell})\}$
    $C_r \leftarrow \bigcup_{i=1}^{\ell} S_{r,i} \cup C_{r-1}$
return S

RANDOMIZED-DASH (R-DASH, Alg. 3) is a single MR round algorithm obtained by plugging LAG into the RANDGREEDI algorithm of Mirzasoleiman et al. (2013); Barbosa et al. (2015). R-DASH runs in one MapReduce round, $O(\log(k) \log(n))$ adaptive rounds, and guarantees the ratio $\frac{1}{2}(1 - 1/e - \varepsilon)$ ($\simeq 0.316$).

Description. The ground set is initially distributed at random by R-DASH across all machines M. In its sole MR round, R-DASH runs LAG on every machine to obtain S_i in $O(\log(k) \log(|\mathcal{N}_i|))$ adaptive rounds. The solution from every machine is then returned to the primary machine, where LAG selects the output solution that guarantees $\frac{1}{2}(1 - 1/e - \varepsilon)$ approximation in $O(\log(k) \log(|S|))$ adaptive rounds as stated in Theorem 4 (Appendix C). The proof is a minor modification of the proof of Barbosa et al. (2015) to incorporate randomized consistency; for completeness, it is provided in Appendix D.

G-DASH. Next, we obtain the nearly the optimal ratio by applying LAG and randomized consistency to DISTRIBUTEDDISTORTED proposed by Kazemi et al. (2021). DISTRIBUTEDDISTORTED is a distributed algorithm for regularized submodular maximization that relies upon (a distorted version of) the standard greedy algorithm. For SMCC, the distorted greedy algorithm reduces to the stan-

Algorithm 5: THRESHOLD-DASH Knowing OPT

Input: Evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}$, constraint k, error ε, available machines $M \leftarrow \{1, 2, ..., \ell\}$, and OPT
Initialize $\delta \leftarrow 1/(\ell+1)$, $\mathbf{q} \leftarrow$ a fixed sequence of random bits.
Set $\alpha \leftarrow \frac{3}{8}$, $(\mathbf{q}_1, \ldots, \mathbf{q}_{\ell+1}) \leftarrow \mathbf{q}$
for $e \in \mathcal{N}$ do **do**
    Assign e to each machine independently with probability $1/\ell$
for $i \in M$ **do**
    ▷ On machine i
    Let $\mathcal{N}_i$ be the elements assigned to machine i
    $S_i \leftarrow \text{LAT}(f, \mathcal{N}_i, k, \delta, \varepsilon, \alpha\text{OPT}/k, \mathbf{q}_i)$
    if $|S_i| = k$ **then** return $T' \leftarrow S_i$
    Send S_i to primary machine
▷ On primary machine
$S \leftarrow \bigcup_{i=1}^{\ell} S_i$
Let $g(\cdot) \leftarrow f(S_1 \cup \cdot) - f(S_1)$
$T \leftarrow \text{LAT}(g, S, k - |S_1|, \delta, \varepsilon, \alpha\text{OPT}/k, \mathbf{q}_{\ell+1})$
$T' \leftarrow S_1 \cup T$
return T'

dard greedy algorithm GREEDY. In Appendix E, we show that LAG can be used in place of GREEDY to yield an algorithm (G-DASH, Alg. 4) that achieves the near optimal $(1 - 1/e - \varepsilon)$ expected ratio in $\lceil \frac{1}{\varepsilon} \rceil$ MapReduce rounds, $O(\log(n)\log(k))$ adaptive rounds, and $O(n \log(k))$ total queries. We also generalize this to any algorithm that satisfies randomized consistency.

3 THRESHOLD-DASH: Single MR-Round Algorithm with Improved Theoretical Properties

In this section, we present a novel $(0.375 - \varepsilon)$-approximate, single MR-round algorithm THRESHOLD-DASH (T-DASH) that achieves nearly optimal $O\log(n)$ adaptivity in $O(n)$ total time complexity.

Description. The T-DASH algorithm (Alg. 5) is a single MR-round algorithm that runs LAT concurrently on every machine with a specified threshold value of $\alpha\text{OPT}/k$. The primary machine then builds up its solution S_1 by adding elements with LAT from the pool of solutions returned by the other machines. Notice that there is a small amount of data duplication as elements of the ground set are not randomly partitioned in the same way as in the other algorithms. This version of the algorithm requires to know the OPT value; in Appendix F we show how to remove this assumption.

Theorem 1. *Let (f, k) be an instance of* SM *where $k < \frac{\psi}{\ell}$.* T-DASH *knowing* OPT *returns set T' with a single MR round, $O\left(\frac{1}{\varepsilon}\log(n)\right)$ adaptive rounds, $O\left(\frac{n}{\varepsilon}\right)$ total queries, $O(\frac{n}{\ell\varepsilon} + \frac{\ell k}{\varepsilon})$ parallel runtime, $O(n + k\ell)$ communication complexity, and probability at least $1 - n^{-c}$ such that*

$$\mathbb{E}\left[f(T')\right] \geq \left(\frac{3}{8} - \varepsilon\right) \text{OPT}.$$

Proof. In Algorithm 5, there are $\ell + 1$ independent calls of LAT. With $|\mathcal{N}_i| \geq n^c$, the success probability of each call of LAT is larger than $1 - \frac{1}{n^c(\ell+1)}$. Thus, Algorithm 5 succeeds with probability larger than $1 - n^{-c}$. For the remainder of the analysis, we condition on the event that all calls to LAT succeed.

In the case that $|T'| = k$, by Theorem 3 in Appendix B, it holds that $f(T') \geq \frac{1-\varepsilon}{1+\varepsilon}\tau \cdot k \geq \left(\frac{3}{8} - \varepsilon\right)\text{OPT}$. Otherwise, we consider the case that $|T'| < k$ in the following. First, we give a definition as follows. For any $x \in \mathcal{N}$, let

$$p_x = \begin{cases} Pr_{X \sim \mathcal{N}(1/\ell), \mathbf{q}}\left[\text{LAT}(X \cup \{x\}, \mathbf{q}) \right. \\ \qquad\qquad \left. = \text{LAT}(X, \mathbf{q})\right] & \text{, if } x \in O \\ 0 & \text{, otherwise} \end{cases}.$$

Let $O_1 = \{o \in O : o \notin \text{LAT}(N_1 \cup \{o\}, q)\}$, $O_2 = S \cap O$. For any $o \in O_1$, o is not selected in S_1. Since, $|S_1| < k$, by Theorem 2 of Chen, Dey, and Kuhnle (2021), $\Delta\left(o \,|\, T'\right) < \Delta\left(o \,|\, S_1\right) < \tau$. Also, for any $o \in O_2 \backslash T$, o is not selected in T. Similarly, $\Delta\left(o \,|\, T'\right) < \tau$. Then, we can get,

$$\begin{aligned} &f(O_1 \cup O_2) - f(T') \leq f(O_1 \cup O_2 \cup T') - f(T') \\ &\leq \sum_{o \in O_1 \cup O_2 \backslash T'} \Delta\left(o \,|\, T'\right) \leq k \cdot \tau = 3\text{OPT}/8. \end{aligned} \tag{2}$$

Next, we provide the following lemma to complete the rest of the proof.

Lemma 3. *For any $o \in O$, it holds that $Pr\left(o \in O_1 \cup O_2\right) \geq 3/4$.*

Then, by Lemma 4 in Appendix, $\mathbb{E}\left[f(O_1 \cup O_2)\right] \geq 3/4 \cdot F(\mathbf{1}_O) = 3/4 \cdot \text{OPT}$. From this and Inequality 2, we can bound the approximation ratio for Algorithm T-DASH knowing OPT by follows, $\mathbb{E}\left[f(T')\right] \geq 3\text{OPT}/8$. □

4 Towards Larger k: A Memory-Efficient, Distributed Framework (MED)

In this section, we propose a general-purpose plug-in framework for distributed algorithms, MEMORYEFFICIENTDISTRIBUTED (MED, Alg. 6). MED increases the largest possible constraint value from $O(n/\ell^2)$ to $O(n/\ell)$ in the value oracle model. Under some additional assumptions, we remove all restrictions on the constraint value.

As discussed in Section 1, the k value is limited to a fraction of the machine memory for MapReduce algorithms: $k \leq O(n/\ell^2)$, since those algorithms need to merge a group of solutions and pass it to a single machine. MED works around this limitation as follows: MED can be thought of as a greedy algorithm that uses an approximate greedy selection through ALG. One machine manages the partial solutions $\{S_i\}$, built up over m iterations. In this way, each call to ALG is within the constraint restriction of ALG, *i.e.* $O(n/\ell^2)$, but a larger solution of up to size $O(n/\ell)$ can be constructed.

The restriction on k of MED of $O(n/\ell)$ comes from passing the data of the current solution to the next round. Intuitively, if we can send some auxiliary information about the

Algorithm 6: MED$(f, k, M, \text{ALG}, \mathbf{q})$

Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}$, constraint k, available machines $M \leftarrow \{1, 2, ..., \ell\}$, MR algorithm ALG, random bits $\mathbf{q}$
$n \leftarrow |\mathcal{N}|$, $\Psi \leftarrow$ Memory capacity (# elements) of primary machine
Choose $k' \leftarrow \max\{k' \in \mathbb{N} : k' \leq \frac{\Psi}{\ell}\}$
$m \leftarrow \lceil k/k' \rceil$, $(\mathbf{q}_1, \ldots, \mathbf{q}_m) \leftarrow \mathbf{q}$
for $i \leftarrow 1$ *to* m **do**
    Let $g(\cdot) \leftarrow f(\cdot \cup S_i) - f(S_i)$
    $A_i \leftarrow \text{ALG}(g, M, \min\{k', k - |S_i|\}, \mathbf{q}_i)$
    $S_{i+1} \leftarrow S_i \cup A_i$
return S_m

function instead of the elements selected, the k value can be unrestricted.

Condition 1. Let f be a set function with ground set $\mathcal{N}$ of size n. If for all $S \subseteq \mathcal{N}$, there exists a bitvector $\mathbf{v}_S$, such that the function $g(X) = f(S \cup X) - f(S)$ can be computed from X and $\mathbf{v}_S$, then f satisfies Condition 1.

We show in Appendix H that all four applications evaluated in Section 5 satisfy Condition 1. As an example, consider MaxCover, which can be expressed as $f(S) = \sum_{i \in \mathcal{N}} f_i(S)$, where $f_i(S) = \mathbf{1}_{\{i \text{ is covered by } S\}}$. Let $g_i(X) = f_i(S \cup X) - f_i(S)$, and $\mathbf{v}_S = (f_1(S), \ldots, f_n(S))$. Then, $f_i(S \cup X) = \mathbf{1}_{\{i \text{ is covered by } S\}} \vee \mathbf{1}_{\{i \text{ is covered by } X\}}$, where the first term is given by $\mathbf{v}_S$ and the second term is calculated by X. Therefore, since $g(X) = \sum_{i \in \mathcal{N}} g_i(X)$, $g(X)$ can be computed from X and $\mathbf{v}_S$.

Theorem 2. *Let (f, k) be an instance of SMCC. If* MED *can run on this instance, then $\mathbb{E}[f(S_m)] \geq (1 - e^{-\gamma})$, where γ is the expected approximation ratio of* ALG. *If $\Psi \geq 2n/\ell$, then* MED *can run for all $k \leq n/(\ell - 1)$. Alternatively, suppose each machine has enough space to store a bitvector of length n in addition to its elements of the ground set, and that f satisfies Condition 1. Then,* MED *can run for all $k \leq n$.*

Proof. The proof of the ratio is in Appendix G. To analyze the memory requirements of MED, consider the following. To compute $g(\cdot) \leftarrow f(\cdot \cup S_i) - f(S_i)$, the current solution S_i need to be passed to each machine in M for the next call of ALG. Suppose $|S| = k - x$ where $1 \leq x \leq k$. The size of data stored on any non-primary machine in cluster M, as well as on the primary machine of M can be bounded as follows:

$$\begin{array}{l} k - x + (n - k + x)/\ell \leq \Psi \\ k - x + (k\ell)/m \leq \Psi \end{array} \Rightarrow \begin{array}{l} k \leq (\ell\Psi - n)/(\ell - 1) + 1 \\ k \leq \Psi - \ell + 1 \end{array}.$$

Therefore, if $\Psi \geq 2n/\ell$, MED can run since $\ell \leq n/\ell$ in the MapReduce model. Under the alternative assumption, it is clear that MED can run for all $k \leq n$. □

5 Empirical Evaluation

This section presents empirical comparison of R-DASH, T-DASH and G-DASH to the state-of-the-art distributed algorithms. Distributed algorithms include RANDGREEDI (RG) of Barbosa et al. (2015) and DISTRIBUTEDDISTORTED (DDIST) of Kazemi et al. (2021).

Datasets. Our datasets are categorized into two groups: **Centralized Data** include datasets that are smaller than memory size of individual machines and range in size from n= 10,000 to 100,000. These were included in the evaluation to enable the completion of expensive algorithms such as DDIST, T-DASH, and G-DASH. **Experiment 1** evaluates the algorithms on centralized data. **Decentralized Data** are large data files that are partitioned uniformly at random and assigned to ℓ machines before evaluation. Algorithms return rows of data files as solutions during MR rounds. Additional dataset details are provided in Appendix H. Groundsets range from n= 50,000 to 3,072,441. **Experiment 2** evaluates the algorithms on decentralized data.

Applications. We evaluated image summarization (ImageSumm), influence maximization (InfMax), revenue maximization (RevMax) and maximum coverage (MaxCover). Details are provided in Appendix H.2. **Environment.** All experiments were run on a cluster of eight machines, each with four available CPU cores. Thus, there were 32 cores available to the algorithms. In Experiment 1 (centralized data), we evaluated our algorithm with number of machines $\ell = 8$ and the prior MR algorithms with $\ell = 32$. Thus, all algorithms were configured to use all 32 cores. In Experiment 2, the algorithms were configured to use 8 cores. The parallel runtime (wall clock time) was measured using MPI for Python utility functions.

Results. The results of Experiment 1 (Fig. 1) show that all algorithms provide similar solution values (with T-DASH being a little worse than the others). However, there is a large difference in parallel runtime, with R-DASH the fastest by orders of magnitude. The availability of only 4 threads per machine severely limits the parallelization of T-DASH, resulting in longer runtime; access to $\log_{1+\varepsilon}(k)$ threads per machine should result in faster runtime than R-DASH.

The results of Experiment 2 (Fig. 2) demonstrate the scalability of R-DASH to large, decentralized datasets ($n \geq 10^6$), as compared with RG. RG completed only 1 instance of k for InfluenceMax within 24 hours $timeout$ and none for RevenueMax and MaxCover; on smaller groundsets, R-DASH is 190 times faster than RG with 98% of its solution value. R-DASH is 8000 times quicker than BCG on the only instance of ImageSumm it completes within $timeout$. Figure 1 and 2 provides ImageSumm and InfluenceMax results; results on other applications given in Appendix I.1 (Fig. 4, 5).

In Fig. 3(a), we show a linear speedup for R-DASH with the number of machines ℓ. In Fig. 3(b), we show how, even if enough memory is available, increasing ℓ can lead to worse performance, if $k > n/\ell^2$. Initially, RANDGREEDI with $\ell = 32$ is faster than RANDGREEDI with $\ell = 8$, as ex-

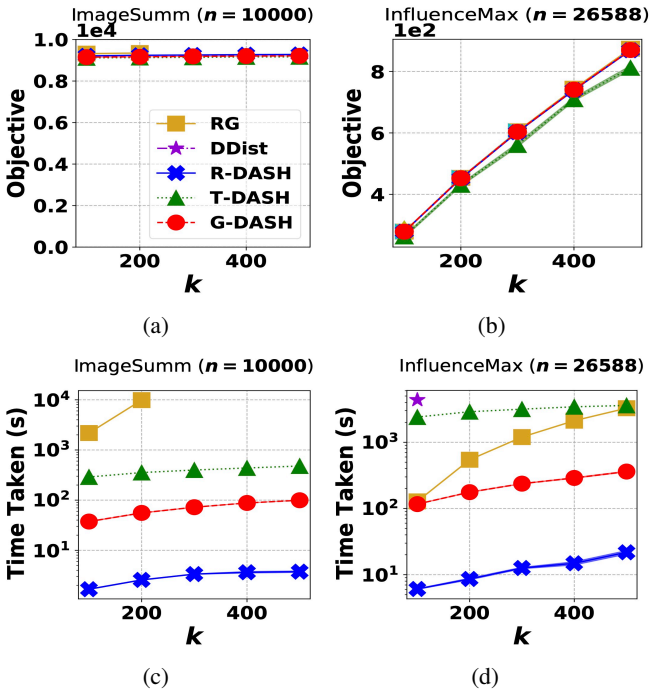

Figure 1: Centralized Datasets

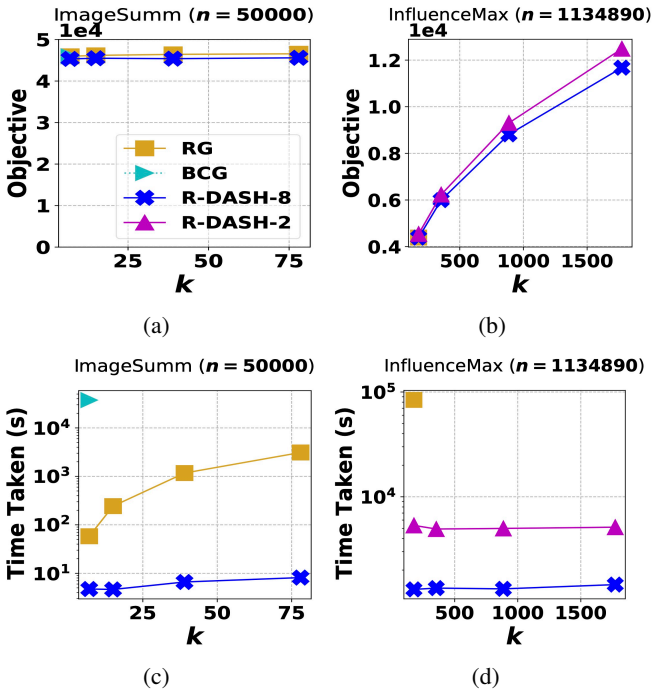

Figure 2: Distrbuted Datasets

Figure 1 and 2 presents performance comparison of distributed algorithms on centralized and decentralized data of ImageSumm and InfluenceMax; other application results shown in Appendix I.1; RANDGREEDI (RG) is run with GREEDY as the algorithm ALG (Barbosa et al. 2015) to ensure the $\frac{1}{2}(1-1/e)$ ratio. All GREEDY implementations used lazy greedy to improve the runtime. For decentralized data results, we compare two variants of R-DASH using 2 and 8 machines (ℓ) respectively. For ImageSumm results on decentralized data, R-DASH-2 could not compute the results due to the size of distributed data ($|\mathcal{N}_i| > \psi$). $Timeout$ for each instance: centralized data = 6 hours; decentralized data = 24 hours.

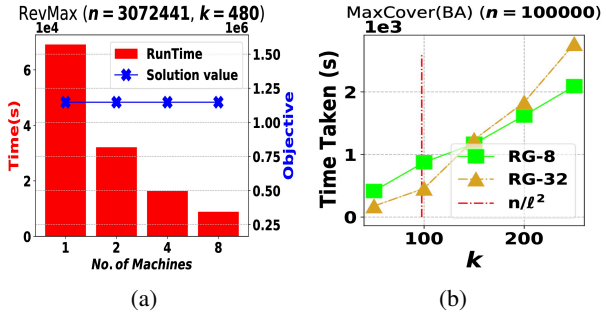

Figure 3: (a): Scalability of R-DASH vs. ℓ. (b): RANDGREEDI with $\ell = 8$ and $\ell = 32$.

pected. However, once $k > n/32^2$, we see the relative performance of RANDGREEDI with $\ell = 32$ worsen rapidly. A theoretical justification and further experiments with this behavior are provided in the Appendix. These results further justify why it is preferable to parallelize within a machine rather than treat each processer as a separate machine over which to distribute the data.

6 Conclusion

In this paper, we mitigate the practical limitations of MR algorithms by introducing R-DASH, T-DASH, and G-DASH, the first MR algorithms with sublinear adaptive complexity (highly parallelizable). We show that paralellizing within a single machine can lead to large empirical benefits. Our algorithms provide state-of-the-art performance while maintaining theoretical guarantees for a distributed setting. In addition, we propose MED, a framework that, under certain conditions, can ameliorate the cardinality constraint limitations of MR algorithms.

Acknowledgements

The work of Tonmoy Dey was partially supported by Florida State University; the work of Yixin Chen and Alan Kuhnle was partially supported by Texas A & M University. The authors have received no third-party funding in direct support of this work. The authors have no additional revenues from other sources related to this work.

References

Balkanski, E.; Rubinstein, A.; and Singer, Y. 2019a. An optimal approximation for submodular maximization under a

matroid constraint in the adaptive complexity model. In *Proceedings of the Annual ACM Symposium on Theory of Computing*.

Balkanski, E.; Rubinstein, A.; and Singer, Y. 2019b. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 283–302. SIAM.

Balkanski, E.; and Singer, Y. 2018. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th annual ACM SIGACT Symposium on Theory of Computing*.

Barbosa, R.; Ene, A.; Nguyen, H.; and Ward, J. 2015. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning*. Proceedings of Machine Learning Research.

Barbosa, R. d. P.; Ene, A.; Nguyen, H. L.; and Ward, J. 2016. A new framework for distributed submodular maximization. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE.

Breuer, A.; Balkanski, E.; and Singer, Y. 2019. The FAST Algorithm for Submodular Maximization. In *International Conference on Machine Learning (ICML)*.

Calinescu, G.; Chekuri, C.; Pál, M.; and Vondrák, J. 2007. Maximizing a Submodular Set Function subject to a Matroid Constraint. In *Integer Programming and Combinatorial Optimization (IPCO)*.

Chekuri, C.; and Quanrud, K. 2019. Submodular function maximization in parallel via the multilinear relaxation. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Chen, Y.; Dey, T.; and Kuhnle, A. 2021. Best of Both Worlds: Practical and Theoretically Optimal Submodular Maximization in Parallel. *Advances in Neural Information Processing Systems (NeurIPS)*.

Ene, A.; and Nguyen, H. L. 2019. Submodular Maximization with Nearly-optimal Approximation and Adaptivity in Nearly-linear Time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Epasto, A.; Mirrokni, V.; and Zadimoghaddam, M. 2017. Bicriteria Distributed Submodular Maximization in a Few Rounds. In *Symposium on Parallelism in Algorithms and Architectures (SPAA)*.

Fahrbach, M.; Mirrokni, V.; and Zadimoghaddam, M. 2019. Submodular Maximization with Nearly Optimal Approximation, Adaptivity, and Query Complexity. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

Kazemi, E.; Minaee, S.; Feldman, M.; and Karbasi, A. 2021. Regularized submodular maximization at scale. In *International Conference on Machine Learning*. Proceedings of Machine Learning Research.

Kazemi, E.; Mitrovic, M.; Zadimoghaddam, M.; Lattanzi, S.; and Karbasi, A. 2019. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *International Conference on Machine Learning*. Proceedings of Machine Learning Research.

Kuhnle, A. 2021a. Nearly Linear-Time, Parallelizable Algorithms for Non-Monotone Submodular Maximization. In *AAAI Conference on Artificial Intelligence*.

Kuhnle, A. 2021b. Quick Streaming Algorithms for Maximization of Monotone Submodular Functions in Linear Time. In *Artificial Intelligence and Statistics (AISTATS)*.

Mirrokni, V.; and Zadimoghaddam, M. 2015. Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the forty-seventh annual ACM Symposium on Theory of Computing*.

Mirzasoleiman, B.; Karbasi, A.; Sarkar, R.; and Krause, A. 2013. Distributed Submodular Maximization: Identifying Representative Elements in Massive Data. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Nemhauser, G. L.; and Wolsey, L. A. 1978. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Mathematics of Operations Research*.

Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*.