Sparsification of Decomposable Submodular Functions

Akbar Rafiey,[1] **Yuichi Yoshida**[2]

[1] Simon Fraser University
[2] National Institute of Informatics
arafiey@sfu.ca, yyoshida@nii.ac.jp

Abstract

Submodular functions are at the core of many machine learning and data mining tasks. The underlying submodular functions for many of these tasks are decomposable, i.e., they are sum of several simple submodular functions. In many data intensive applications, however, the number of underlying submodular functions in the original function is so large that we need prohibitively large amount of time to process it and/or it does not even fit in the main memory. To overcome this issue, we introduce the notion of sparsification for decomposable submodular functions whose objective is to obtain an accurate approximation of the original function that is a (weighted) sum of only a few submodular functions. Our main result is a polynomial-time randomized sparsification algorithm such that the expected number of functions used in the output is independent of the number of underlying submodular functions in the original function. We also study the effectiveness of our algorithm under various constraints such as matroid and cardinality constraints. We complement our theoretical analysis with an empirical study of the performance of our algorithm.

Introduction

Submodularity of a set function is an intuitive diminishing returns property, stating that adding an element to a smaller set helps gaining more return than adding it to a larger set. This fundamental structure has emerged as a very beneficial property in many combinatorial optimization problems arising in machine learning, graph theory, economics, game theory, to name a few. Formally, a set function $f\colon 2^E \to \mathbb{R}$ is *submodular* if for any $S \subseteq T \subseteq E$ and $e \in E \setminus T$ it holds that

$$f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T).$$

Submodularity allows one to efficiently find provably (near-)optimal solutions. In particular, a wide range of efficient approximation algorithms have been developed for maximizing or minimizing submodular functions subject to different constraints. Unfortunately, these algorithms require number of function evaluations which in many data intensive applications are infeasible or extremely inefficient. Fortunately, several submodular optimization problems arising in machine learning have structure that allows solving them more efficiently. A novel class of submodular functions are *decomposable* submodular functions. These are functions that can be written as sums of several "simple" submodular functions, i.e.,

$$F(S) = \sum_{i=1}^{N} f_i(S) \qquad \forall S \subseteq E,$$

where each $f_i\colon 2^E \to \mathbb{R}$ is a submodular function on the ground set E with $|E| = n$.

Decomposable submodular functions encompass many of the examples of submodular functions studied in the context of machine learning as well as economics. For example, they are extensively used in economics in the problem of welfare maximization in combinatorial auctions (Dobzinski and Schapira 2006; Feige 2006; Feige and Vondrák 2006; Papadimitriou, Schapira, and Singer 2008; Vondrák 2008).

Example 1 (Welfare maximization)**.** Let E be a set of n resources and $a_1, \ldots, a_N$ be N agents. Each agent has an interest over subsets of resources which is expressed as a submodular function $f_i : 2^E \to \mathbb{R}$. The objective is to select a small subset of resources that maximizes the happiness across all the agents, the "social welfare". More formally, the goal is to find a subset $S \subseteq E$ of size at most k that maximizes $F(S) = \sum_{i=1}^{N} f_i(S)$, where k is a positive integer.

Decomposable submodular functions appear in various machine learning tasks such as data summarization, where we seek a representative subset of elements of small size. This has numerous applications, including exemplar-based clustering (Dueck and Frey 2007; Gomes and Krause 2010), image summarization (Tschiatschek et al. 2014), recommender systems (Parambath, Usunier, and Grandvalet 2016), and document and corpus summarization (Lin and Bilmes 2011). The problem of maximizing decomposable submodular functions has been studied under different constraints such as cardinality and matroid constraints in various data summarization settings (Mirzasoleiman, Badanidiyuru, and Karbasi 2016; Mirzasoleiman et al. 2016; Mirzasoleiman, Zadimoghaddam, and Karbasi 2016), and differential privacy settings (Chaturvedi, Nguyen, and Zakynthinou 2021; Mitrovic et al. 2017; Rafiey and Yoshida 2020).

In many of these applications, the number of underlying submodular functions are too large (i.e., N is too large) to even fit in the main memory, and building a compressed representation that preserves relevant properties of the submodular function is appealing. This motivates us to find a *sparse* representation for a decomposable submodular function F. Sparsification is an algorithmic paradigm where a dense object is replaced by a sparse one with similar "features", which often leads to significant improvements in efficiency of algorithms, including running time, space complexity, and communication. In this work, we propose a simple and very effective algorithm that yields a sparse and accurate representation of a decomposable submodular function. To the best of our knowledge this work is the first to study sparsification of decomposable submodular functions.

Our Contributions

General setting. Given a decomposable submodular function $F = \sum_{i=1}^{N} f_i$, we present a randomized algorithm that yields a sparse representation that approximates F. Our algorithm chooses each submodular function f_i with probability proportional to its "importance" in the sum $\sum_{i=1}^{N} f_i$ to be in the sparsifier. Moreover, each selected submodular function will be assigned a weight which also is proportional to its "importance". We prove this simple algorithm yields a sparsifier of small size (***independent of*** N) with a very good approximation of F. Let $|\mathcal{B}(f_i)|$ denote the number of extreme points in the base polytope of f_i, and $B = \max_{i\in[N]} |\mathcal{B}(f_i)|$.

Theorem 2. *Let $F = \sum_{i=1}^{N} f_i$ be a decomposable submodular function. For any $\epsilon > 0$, there exists a vector $\boldsymbol{w} \in \mathbb{R}^N$ with at most $O(\frac{B\cdot n^2}{\epsilon^2})$ non-zero entries such that for the submodular function $F' = \sum_{i=1}^{N} \boldsymbol{w}_i f_i$ we have*

$$(1-\epsilon)F'(S) \le F(S) \le (1+\epsilon)F'(S) \quad \forall S \subseteq E.$$

Moreover, if all f_i's are monotone, then there exists a polynomial-time randomized algorithm that outputs a vector $\boldsymbol{w} \in \mathbb{R}^N$ with at most $O(\frac{B\cdot n^{2.5}\log n}{\epsilon^2})$ non-zero entries in expectation such that for the submodular function $F' = \sum_{i=1}^{N} \boldsymbol{w}_i f_i$, with high probability, we have

$$(1-\epsilon)F'(S) \le F(S) \le (1+\epsilon)F'(S) \quad \forall S \subseteq E.$$

Remark 3 (Tightness)**.** The existential result is almost tight because in the special case of directed graphs, we have $\max_i |\mathcal{B}(f_i)| = 2$ and it is known that we need $\Omega(n^2)$ edges to construct a sparsifier (Cohen et al. 2017).

Sparsifying under constraints. We consider the setting where we only are interested in evaluation of F on particular sets. For instance, the objective is to optimize F on subsets of size at most k, or it is to optimize F over subsets that form a matroid. Optimizing submodular functions under these constraints has been extensively studied and has an extremely rich theoretical landscape. Our algorithm can be tailored to these types of constraints and constructs a sparsifier of even smaller size.

Theorem 4. *Let $F = \sum_{i=1}^{N} f_i$ be a decomposable submodular function. For any $\epsilon > 0$ and a matroid $\mathcal{M}$ of rank r, there exists a vector $\boldsymbol{w} \in \mathbb{R}^N$ with at most $O(\frac{B\cdot r\cdot n}{\epsilon^2})$ non-zero entries such that for the submodular function $F' = \sum_{i=1}^{N} \boldsymbol{w}_i f_i$ we have*

$$(1-\epsilon)F'(S) \le F(S) \le (1+\epsilon)F'(S) \quad \forall S \subseteq \mathcal{M}.$$

Moreover, if all f_i's are monotone, then there exists a polynomial-time randomized algorithm that outputs a vector $\boldsymbol{w} \in \mathbb{R}^N$ with at most $O(\frac{B\cdot r\cdot n^{1.5}\log n}{\epsilon^2})$ non-zero entries in expectation such that for the submodular function $F' = \sum_{i=1}^{N} \boldsymbol{w}_i f_i$, with high probability, we have

$$(1-\epsilon)F'(S) \le F(S) \le (1+\epsilon)F'(S) \quad \forall S \subseteq \mathcal{M}.$$

Applications, speeding up maximization/minimization. Our sparsifying algorithm can be used as a preprocessing step in many settings in order to speed up algorithms. To elaborate on this, we consider the classical greedy algorithm of Nemhauser, Wolsey, and Fisher for maximizing monotone submodular functions under cardinality constraints. We observe that sparsifying the instance reduces the number of function evaluations from $O(knN)$ to $O(\frac{Bk^2n^2}{\epsilon^2})$, which is a significant speed up when $N \gg n$. Regarding minimization, we prove our algorithm gives an approximation on the *Lovász extension*, thus it can be used as a preprocessing step for algorithms working on Lovász extensions such as the ones in (Axiotis et al. 2021; Ene, Nguyen, and Végh 2017). One particular regime that has been considered in many results is where each submodular function f_i acts on $O(1)$ elements of the ground set which implies $B = \max_i |\mathcal{B}(f_i)|$ is $O(1)$. Using our sparsifier algorithm as a preprocessing step is quite beneficial here. For instance, it improves the running time of Axiotis et al. (2021) from $\widetilde{O}(T_{\mathrm{maxflow}}(n, n+N)\log\frac{1}{\epsilon})$ to $\widetilde{O}(T_{\mathrm{maxflow}}(n, n+\frac{n^2}{\epsilon^2})\log\frac{1}{\epsilon})$. Here, $T_{\mathrm{maxflow}}(n, m)$ denotes the time required to compute the maximum flow in a directed graph of n vertices and m arcs with polynomially bounded integral capacities.

Well-known examples. In practice, the bounds on the size of sparsifiers are often better than the ones presented in Theorems 2 and 4 e.g. B is a constant. We consider several examples of decomposable submodular functions that appear in many applications, namely, MAXIMUM COVERAGE, FACILITY LOCATION, and SUBMODULAR HYPERGRAPH MIN CUT problems. For the first two examples, sparsifiers of size $O(\frac{n^2}{\epsilon^2})$ can be constructed in time linear in N. For SUBMODULAR HYPERGRAPH MIN CUT when each hyperedge is of constant size sparsifiers of size $O(\frac{n^2}{\epsilon^2})$ exist, and in several specific cases with various applications efficient algorithms are employed to construct them.

Empirical results. Finally, we empirically examine our algorithm and demonstrate that it constructs a concise sparsifier on which we can efficiently perform algorithms.

Related Work

To the best of our knowledge there is no prior work on sparsification algorithms for decomposable submodular functions. However, special cases of this problem have attracted much attention, most notably *cut sparsifiers* for graphs. The cut function of a graph $G = (V, E)$ can be seen as a decomposable submodular function $F(S) = \sum_{e \in E} f_e$, where $f_e(S) = 1$ if and only if $e \cap S \neq \emptyset$ and $e \cap (V \setminus S) \neq \emptyset$. The problem of sparsifying a graph while approximately preserving its cut structure has been extensively studied, (See Ahn, Guha, and McGregor (2012, 2013); Bansal, Svensson, and Trevisan (2019); Benczúr and Karger (2015) and references therein.) The pioneering work of Benczúr and Karger (1996) showed for any graph G with n vertices one can construct a weighted subgraph G' in nearly linear time with $O(n \log n/\epsilon^2)$ edges such that the weight of every cut in G is preserved within a multiplicative $(1 \pm \epsilon)$-factor in G'. Note that a graph on n vertices can have $N = \Omega(n^2)$ edges. The bound on the number of edges was later improved to $O(n/\epsilon^2)$ (Batson, Spielman, and Srivastava 2012) which is tight (Andoni et al. 2016).

A more general concept for graphs called *spectral sparsifier* was introduced by Spielman and Teng (2011). This notion captures the spectral similarity between a graph and its sparsifiers. A spectral sparsifier approximates the *quadratic form of the Laplacian* of a graph. Note that a spectral sparsifier is also a cut sparsifier. This notion has numerous applications in linear algebra (Mahoney 2011; Li, Miller, and Peng 2013; Cohen et al. 2015; Lee and Sidford 2014), and it has been used to design efficient approximation algorithms related to cuts and flows (Benczúr and Karger 2015; Karger and Levine 2002; Madry 2010). Spielman and Teng's sparsifier has $O(n \log^c n)$ edges for a large constant $c > 0$ which was improved to $O(n/\epsilon^2)$ (Lee and Sun 2018).

In pursuing a more general setting, the notions of cut sparsifier and spectral sparsifier have been studied for *hypergraphs*. Observe that a hypergraph on n vertices can have exponentially many hyperedges i.e., $N = \Omega(2^n)$. For hypergraphs, Kogan and Krauthgamer (2015) provided a polynomial-time algorithm that constructs an ϵ-cut sparsifier with $O(n(r + \log n)/\epsilon^2)$ hyperedges where r denotes the maximum size of a hyperedge. The current best result is due to Chen, Khanna, and Nagda (2020) where their ϵ-cut sparsifier uses $O(n \log n/\epsilon^2)$ hyperedges and can be constructed in time $O(Nn^2 + n^{10}/\epsilon^2)$ where N is the number of hyperedges. Recently, Soma and Yoshida (2019) initiated the study of spectral sparsifiers for hypergraphs and showed that every hypergraph admits an ϵ-spectral sparsifier with $O(n^3 \log n/\epsilon^2)$ hyperedges. For the case where the maximum size of a hyperedge is r, Bansal, Svensson, and Trevisan (2019) showed that every hypergraph has an ϵ-spectral sparsifier of size $O(nr^3 \log n/\epsilon^2)$. Recently, this bound has been improved to $O(nr(\log n/\epsilon)^{O(1)})$ and then to $O(n(\log n/\epsilon)^{O(1)})$ (Kapralov et al. 2021a,b). This leads to the study of sparsification of submodular functions which is the focus of this paper and provides a unifying framework for these previous works.

Preliminaries

For a positive integer n, let $[n] = \{1, 2, \ldots, n\}$. Let E be a set of elements of size n which we call the *ground set*. For a set $S \subseteq E$, $\mathbf{1}_S \in \mathbb{R}^E$ denotes the characteristic vector of S. For a vector $\boldsymbol{x} \in \mathbb{R}^E$ and a set $S \subseteq E$, $\boldsymbol{x}(S) = \sum_{e \in S} \boldsymbol{x}(e)$.

Submodular functions. Let $f\colon 2^E \to \mathbb{R}_+$ be a set function. We say that f is *monotone* if $f(S) \leq f(T)$ holds for every $S \subseteq T \subseteq E$. We say that f is *submodular* if $f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T)$ holds for any $S \subseteq T \subseteq E$ and $e \in E \setminus T$. The *base polytope* of a submodular function f is defined as

$$\mathcal{B}(f) = \{\boldsymbol{y} \in \mathbb{R}^E \mid \boldsymbol{y}(S) \leq f(S)\ \forall S \subseteq E, \boldsymbol{y}(E) = f(E)\},$$

and $|\mathcal{B}(f)|$ denotes the number of *extreme points* in the base polytope $\mathcal{B}(f)$.

Definition 5 (ϵ-sparsifier)**.** *Let f_i ($i \in D$) be a set of N submodular functions, and $F(S) = \sum_{i \in D} f_i(S)$ be a decomposable submodular function. A vector $\boldsymbol{w} \in \mathbb{R}^N$ is called an ϵ-sparsifier of F if, for the submodular function $F' := \sum_{i \in D} \boldsymbol{w}_i f_i$, the following holds for every $S \subseteq E$*

$$(1 - \epsilon)F'(S) \leq F(S) \leq (1 + \epsilon)F'(S). \tag{1}$$

The size *of an ϵ-sparsifier $\boldsymbol{w}$,* $\mathrm{size}(\boldsymbol{w})$*, is the number of indices i's with $\boldsymbol{w}_i \neq 0$.*

Matroids and matroid polytopes. A pair $\mathcal{M} = (E, \mathcal{I})$ of a set E and $\mathcal{I} \subseteq 2^E$ is called a *matroid* if (1) $\emptyset \in \mathcal{I}$, (2) $A \in \mathcal{I}$ for any $A \subseteq B \in \mathcal{I}$, and (3) for any $A, B \in \mathcal{I}$ with $|A| < |B|$, there exists $e \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{I}$. We call a set in $\mathcal{I}$ an *independent set*. The *rank function* $r_{\mathcal{M}}\colon 2^E \to \mathbb{Z}_+$ of $\mathcal{M}$ is $r_{\mathcal{M}}(S) = \max\{|I| : I \subseteq S, I \in \mathcal{I}\}$. An independent set $S \in \mathcal{I}$ is called a *base* if $r_{\mathcal{M}}(S) = r_{\mathcal{M}}(E)$. We denote the rank of $\mathcal{M}$ by $r(\mathcal{M})$. The *matroid polytope* $\mathcal{P}(\mathcal{M}) \subseteq \mathbb{R}^E$ of $\mathcal{M}$ is $\mathcal{P}(\mathcal{M}) = \mathrm{conv}\{\mathbf{1}_I : I \in \mathcal{I}\}$, where conv denotes the convex hull. Or equivalently (Edmonds 2001), $\mathcal{P}(\mathcal{M}) = \{\boldsymbol{x} \geq \mathbf{0} : \boldsymbol{x}(S) \leq r_{\mathcal{M}}(S)\ \forall S \subseteq E\}$.

Constructing a Sparsifier

In this section, we propose a probabilistic argument that proves the existence of an accurate sparsifier and turn this argument into an (polynomial-time) algorithm that finds a sparsifier with high probability.

For each submodular function f_i, let

$$p_i = \max_{A \subseteq E} \frac{f_i(A)}{F(A)}. \tag{2}$$

The values p_i's are our guide on how much weight should be allocated to a submodular function f_i and with what probability it might happen. To construct an ϵ-sparsifier of F, for each submodular function f_i, we assign weight $1/(\kappa \cdot p_i)$ to $\boldsymbol{w}_i$ with probability $\kappa \cdot p_i$ and do nothing for the complement probability $1 - \kappa \cdot p_i$ (see Algorithm 1). Here κ depends on n, ϵ and δ where δ is the failure probability of our algorithm. Observe that, for each f_i, the expected weight $\boldsymbol{w}_i$ is exactly one. We show that the expected number of entries of $\boldsymbol{w}$ with $\boldsymbol{w}_i > 0$ is $n^2 \cdot \max_{i \in D} |\mathcal{B}(f_i)|$. Let $B = \max_{i \in D} |\mathcal{B}(f_i)|$ in the rest of the paper.

Lemma 6. *Algorithm 1 returns $\boldsymbol{w}$ which is an ϵ-sparsifier of F with probability at least $1 - \delta$.*

Lemma 7. *Algorithm 1 outputs an ϵ-sparsifier with the expected size $O(\frac{B \cdot n^2}{\epsilon^2})$.*

Algorithm 1

```
Require: Submodular functions $f_i$ in dataset $D$ where each $f_i : \{0,1\}^E \to \mathbb{R}$, $\epsilon, \delta \in (0,1)$
1: $\boldsymbol{w} \leftarrow \mathbf{0}$
2: $\kappa \leftarrow 3\log(2^{n+1}/\delta)/\epsilon^2$
3: for $f_i$ in $D$ do
4:     $p_i \leftarrow \max_{A\subseteq E} f_i(A)/F(A)$
5:     $\kappa_i \leftarrow \min\{1, \kappa \cdot p_i\}$
6:     $\boldsymbol{w}_i \leftarrow 1/\kappa_i$ with probability $\kappa_i$    ▷ do nothing with probability $1-\kappa_i$
7: return $\boldsymbol{w} \in \mathbb{R}^D$.
```

Proof. In Algorithm 1, each $\boldsymbol{w}_i$ is greater than zero with probability κ_i and it is zero with probability $1-\kappa_i$. Hence,

$$\mathbb{E}[\text{size}(\boldsymbol{w})] = \sum_{i\in D} \kappa_i \le \kappa \sum_{i\in D} p_i \le O\left(n/\epsilon^2\right) \sum_{i\in D} p_i \qquad (3)$$

It suffices to show an upper bound for $\sum_{i\in D} p_i$.

Claim 8. $\sum_{i\in D} p_i \le n \cdot \max_{i\in D} |\mathcal{B}(f_i)| = n \cdot B$.

Claim 8 and inequality (3) yield the desired bound. □

Lemmas 6 and 7 proves the existence part of Theorem 2. That is, for every $\epsilon, \delta \in (0,1)$, there exists an ϵ-sparsifier of size at most $O(\frac{B\cdot n^2}{\epsilon^2})$ with probability at least $1-\delta$.

Polynomial time algorithm. Observe that computing p_i's (2) may not be a polynomial-time task in general. However, to guarantee that Algorithm 1 outputs an ϵ-sparsifier with high probability it is sufficient to instantiate it with an upper bound for each p_i (see proof of Lemma 6). Fortunately, the result of Bai et al. (2016) provides an algorithm to approximate the ratio of two monotone submodular functions.

Theorem 9 (Bai et al. (2016))**.** *Let f and g be two monotone submodular functions. Then there exists a polynomial-time algorithm that approximates $\max_{S\subseteq E} \frac{f(S)}{g(S)}$ within $O(\sqrt{n}\log n)$ factor.*

Hence, when all f_i's are monotone we can compute $\hat{p}_i$'s with $p_i \le \hat{p}_i \le O(\sqrt{n}\log n)p_i$ in polynomial time which leads to a polynomial-time randomized algorithm that constructs an ϵ-sparsifier of the expected size at most $O(\frac{B\cdot n^{2.5}\log n}{\epsilon^2})$. This proves the second part of Theorem 2.

As we will see, in various applications, the expected size of the sparsifier is often much better than the ones presented in this section. Also, we emphasize that once a sparsifier is constructed it can be reused many times (possibly for maximization/minimization under several different constraints). Hence computing or approximating p_i's should be regarded as a preprocessing step. Finally, it is straightforward to adapt our algorithm to sparsify decomposable submodular functions of the form $\sum_{i\in D} \alpha_i f_i$, known as *mixtures of submodular functions* (Bairi et al. 2015; Tschiatschek et al. 2014).

Algorithm 2

```
Require: Submodular functions $f_i : \{0,1\}^E \to \mathbb{R}$ in dataset $D$, matroid $\mathcal{M} = (E,\mathcal{I})$, and $\epsilon, \delta \in (0,1)$
1: $\boldsymbol{w} \leftarrow \mathbf{0}$
2: $\kappa \leftarrow 3\log(2n^{r+1}/\delta)/\epsilon^2$, where $r$ is the rank of $\mathcal{M}$.
3: for $f_i$ in $D$ do
4:     $p_i \leftarrow \max_{A\in\mathcal{I}} f_i(A)/F(A)$
5:     $\kappa_i \leftarrow \min\{1, \kappa \cdot p_i\}$
6:     $\boldsymbol{w}_i \leftarrow 1/\kappa_i$ with probability $\kappa_i$    ▷ do nothing with probability $1-\kappa_i$
7: return $\boldsymbol{w} \in \mathbb{R}^D$.
```

Constructing a Sparsifier under Constraints

Here we are interested in constructing a sparsifier for a decomposable submodular function F while the goal is to optimize F subject to constraints. One of the most commonly used and general constraints are matroid constraints. That is, for a matroid $\mathcal{M} = (E,\mathcal{I})$, the objective is finding $S^* = \text{argmax}_{S\subseteq E, S\in\mathcal{I}} F(S)$.

In this setting it is sufficient to construct a sparsifier that approximates F only on independent sets. It turns out that we can construct a *smaller* sparsifier than the one constructed to approximate F everywhere. For each submodular function f_i, let

$$p_i = \max_{A\in\mathcal{I}} \frac{f_i(A)}{F(A)}. \qquad (4)$$

Other than different definition for p_i's and different κ, Algorithm 2 is the same as Algorithm 1.

Theorem 10. *Algorithm 2 returns a vector $\boldsymbol{w}$ with expected size at most $O(\frac{B\cdot r\cdot n}{\epsilon^2})$ such that, with probability at least $1-\delta$, for $F' = \sum_{i\in D} \boldsymbol{w}_i f_i$ we have*

$$(1-\epsilon)F'(S) \le F(S) \le (1+\epsilon)F'(S) \quad \forall S \subseteq \mathcal{M}.$$

Theorem 10 proves the existence part of Theorem 4. Algorithm 2 can be turned into a polynomial-time algorithm if one can approximate p_i's (4). By modifying the proof of Theorem 9 we prove the following.

Theorem 11. *Let f and g be two monotone submodular functions and $\mathcal{M} = (E,\mathcal{I})$ be a matroid. Then there exists a polynomial-time algorithm that approximates $\max_{S\subseteq E, S\in\mathcal{I}} \frac{f(S)}{g(S)}$ within $O(\sqrt{n}\log n)$ factor.*

By this theorem, when all f_is are monotone we can compute $\hat{p}_i$'s with $p_i \le \hat{p}_i \le O(\sqrt{n}\log n)p_i$ in polynomial time which leads to a polynomial-time randomized algorithm that constructs an ϵ-sparsifier of the expected size at most $O(\frac{B\cdot r\cdot n^{1.5}\log n}{\epsilon^2})$. This proves the second part of Theorem 4.

Applications

Submodular Function Maximization with Cardinality Constraint

Our sparsification algorithm can be used as a preprocessing step and once a sparsifier is constructed it can be reused

Algorithm 3

```
Require: Submodular function $F = \sum_{i \in D} f_i$ with each $f_i : \{0,1\}^E \to \mathbb{R}$, constant $k$, and $\epsilon, \delta \in (0,1)$
1: Compute $F' = \sum_{i \in D} \boldsymbol{w}_i f_i$, an $\epsilon$-sparsifier for $F$.
2: $A \leftarrow \emptyset$.
3: while $|A| \leq k$ do
4:     $a_i \leftarrow \mathrm{argmax}_{a \in E \setminus A}(F'(A \cup \{a\}) - F'(A))$.
5:     $A \leftarrow A \cup \{a_i\}$.
6: return $A$.
```

many times (possibly for maximization/minimization under several different constraints). To elaborate on this, we consider the problem of maximizing a submodular function subject to a cardinality constraint. That is finding $S^* = \mathrm{argmax}_{S \subseteq E, |S| \leq k} F(S)$. Cardinality constraint is a special case of matroid constraint where the independent sets are all subsets of size at most k and the rank of the matroid is k. A celebrated result of Nemhauser, Wolsey, and Fisher (1978) states that for non-negative monotone submodular functions a simple greedy algorithm provides a solution with $(1-1/\mathrm{e})$ approximation guarantee to the optimal (intractable) solution. For a ground set E of size n and a monotone submodular function $F = \sum_{i \in D} f_i$, this greedy algorithm needs $O(knN)$ function evaluations to find S of size k such that $F(S) \geq (1 - 1/\mathrm{e})F(S^*)$. We refer to this algorithm as `GreedyAlg`. In many applications where $N \gg n$, having a sparsifier is beneficial. Applying `GreedyAlg` on an ϵ-sparsifier of size $O(Bkn/\epsilon^2)$ improves the number of function evaluations to $O(Bk^2n^2/\epsilon^2)$ and yields S of size k such that $F(S) \geq (1 - 1/\mathrm{e} - \epsilon)F(S^*)$ with high probability (see Algorithm 3).

We point out that sampling techniques such as (Mitrovic et al. 2018; Mirzasoleiman et al. 2015) sample elements from the ground set E rather than sampling from functions $f_1, \ldots, f_N$. Hence their running time depend on N, which could be slow when N is large — the regime we care about. Besides, our algorithm can be used as a preprocessing step for these algorithms. For instance, the lazier than lazy greedy algorithm (Mirzasoleiman et al. 2015) requires $O(nN \log \frac{1}{\epsilon})$ function evaluations. However, when N is much larger than n it is absolutely beneficial to use our sparsification algorithm and reduce the number of submodular functions that one should consider.

Two Well-known Examples

Maximum Coverage Problem. Let $[N]$ be a universe and $E = \{S_1, \ldots, S_n\}$ with each $S_i \subseteq N$ be a family of sets. Given a positive integer k, in the MAX COVERAGE problem the objective is to select at most k of sets from E such that the maximum number of elements are covered, i.e., the union of the selected sets has maximal size. One can formulate this problem as follows. For every $i \in [N]$ and $A \subseteq [n]$ define $f_i(A)$ as

$$f_i(A) = \begin{cases} 1 & \text{if there exists } a \in A \text{ such that } i \in S_a, \\ 0 & \text{otherwise.} \end{cases}$$

Note that f_i's are monotone and submodular. Furthermore, define $F : 2^n \to \mathbb{R}_+$ to be $F(A) = \sum_{i \in [N]} f_i(A)$ which is monotone and submodular as well. Now the MAX COVERAGE problem is equivalent to $\max_{A \subseteq [n], |A| \leq k} F(A)$. For each submodular function f_i, the corresponding p_i is

$$p_i = \max_{A \subseteq [n], |A| \leq k} \frac{f_i(A)}{F(A)} = \max_{S_a \in E, i \in S_a} \frac{f_i(\{a\})}{F(\{a\})}$$
$$= \max_{S_a \in E, i \in S_a} \frac{1}{F(\{a\})} = \max_{S_a \in E, i \in S_a} \frac{1}{|S_a|}.$$

We can compute all the p_i's in $O(\sum |S_i|)$ time, which is the input size. Then we can construct a sparsifier in $O(N)$ time. In total, the time required for sparsification is $O(\sum |S_i| + N)$. On the other hand, for this case we have

$$\sum_{i=1}^{N} p_i = \sum_{i=1}^{N} \max_{S_a \in \mathcal{S}, i \in S_a} \frac{1}{|S_a|} \leq \sum_{i=1}^{n} \frac{|S_i|}{|S_i|} = n.$$

By Lemma 7, this upper bound provides that our algorithm constructs an ϵ-sparsifier of size at most $O(kn/\epsilon^2)$. Algorithm 3 improves the running time of the `GreedyAlg` from $O(knN)$ to $O(k^2n^2/\epsilon^2)$. Furthermore, Algorithm 3 returns a set A of size at most k such that $(1-1/\mathrm{e}-\epsilon)\mathtt{OPT} \leq F(A)$. ($\mathtt{OPT}$ denotes $F(S^*)$ where $S^* = \mathrm{argmax}_{S \subseteq E, |S| \leq k} F(S)$.)

Facility Location Problem. Let I be a set of N clients and E be a set of facilities with $|E| = n$. Let $c : I \times E \to \mathbb{R}$ be the cost of assigning a given client to a given facility. For each client i and each subset of facilities $A \subseteq E$, define $f_i(A) = \max_{j \in A} c(i,j)$. For any non-empty subset $A \subseteq E$, the value of A is given by

$$F(A) = \sum_{i \in I} f_i(A) = \sum_{i \in I} \max_{j \in A} c(i,j).$$

For completeness, we define $F(\emptyset) = 0$. An instance of the MAX FACILITY LOCATION problem is specified by a tuple (I, E, c). The objective is to choose a subset $A \subseteq E$ of size at most k maximizing $F(A)$. For each submodular function f_i, the corresponding p_i is

$$p_i = \max_{A \subseteq E, |A| \leq k} \frac{\max_{j \in A} c(i,j)}{F(A)} = \max_{j \in E} \frac{c(i,j)}{F(\{j\})}$$

It is clear that p_i's can be computed in $O(|I| \cdot |E|)$ time, which is the input size. In this case, we have

$$\sum_{i \in I} p_i = \sum_{i \in I} \max_{j \in E} \frac{c(i,j)}{F(\{j\})} \leq \sum_{j=1}^{|E|} \frac{\sum_{i \in I} c(i,j)}{F(\{j\})} = \sum_{j=1}^{|E|} \frac{F(\{j\})}{F(\{j\})}$$
$$= |E| = n.$$

Hence, by Lemma 7, our algorithm construct a sparsifier of size $O(kn/\epsilon^2)$. Algorithm 3 improves the running time of the `GreedyAlg` from $O(knN)$ to in $O(k^2n^2/\epsilon^2)$. Furthermore, Algorithm 3 returns a set A of size at most k such that $(1 - 1/\mathrm{e} - \epsilon)\mathtt{OPT} \leq F(A)$.

Remark 12. Lindgren, Wu, and Dimakis (2016) sparsify an instance of the FACILITY LOCATION problem by zeroing out entries in the cost matrix — this is not applicable to the general setting. The runtime of the `GreedyAlg` applied on their sparsified instance is $O(nN/\epsilon)$. This runtime is huge when N is large — the regime we care about. Moreover, we can first construct our sparsifier and apply the algorithm of Lindgren, Wu, and Dimakis on it.

Submodular Function Minimization

Besides the applications regarding submodular maximization, our sparsification algorithm can be used as a preprocessing step for submodular minimization as well. In many applications of the submodular minimization problem such as image segmentation (Shanu, Arora, and Singla 2016), Markov random field inference (Fix et al. 2013; Kohli, Ladicky, and Torr 2009; Vicente, Kolmogorov, and Rother 2009), hypergraph cuts (Veldt, Benson, and Kleinberg 2020b), covering functions (Stobbe and Krause 2010), the submodular function at hand is a decomposable submodular function. Many of recent advances on decomposable submodular minimization such as (Ene, Nguyen, and Végh 2017; Axiotis et al. 2021) have leveraged a mix of ideas coming from both discrete and continuous optimization. Here we discuss that our sparsifying algorithm approximates the so called *Lovász extension*, a natural extension of a submodular function to the continuous domain $[0,1]^n$.

Lovász extension. Let $\boldsymbol{x} \in [0,1]^n$ be the vector $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$. Let $\pi : [n] \to [n]$ be a sorting permutation of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, which means if $\pi(i) = j$, then $\boldsymbol{x}_j$ is the i-th largest element in the vector $\boldsymbol{x}$. Hence, $1 \geq \boldsymbol{x}_{\pi(1)} \geq \cdots \geq \boldsymbol{x}_{\pi(n)} \geq 0$. Let $\boldsymbol{x}_{\pi(0)} = 1$ and $\boldsymbol{x}_{\pi(n+1)} = 0$. Define sets $S_0^\pi = \emptyset$ and $S_i^\pi = \{\pi(1), \ldots, \pi(i)\}$. The *Lovász extension* of f is defined as follows $f^L(\boldsymbol{x}) = \sum_{i=0}^n (\boldsymbol{x}_{\pi(i)} - \boldsymbol{x}_{\pi(i+1)}) f(S_i^\pi)$. It is well-known that $f^L(\boldsymbol{x}) = \max_{\boldsymbol{y} \in \mathcal{B}(f)} \langle \boldsymbol{y}, \boldsymbol{x} \rangle$.

For a decomposable submodular function $F = \sum_{i \in D} f_i$, its Lovász extension is

$$F^L(\boldsymbol{x}) = \sum_{j=0}^{n} \sum_{i \in D} (\boldsymbol{x}_{\pi(j)} - \boldsymbol{x}_{\pi(j+1)}) f_i(S_j^\pi).$$

Recall the definition of p_i's (2), they can be expressed in an equivalent way in terms of permutations as follow

$$p_i = \max_{A \subseteq E} \frac{f_i(A)}{F(A)} = \max_{\pi} \max_{j \in [n]} \frac{f_i(S_j^\pi)}{F(S_j^\pi)}. \quad (5)$$

Furthermore, note that $F^L(\boldsymbol{x})$ is a linear combination of $F(S)$, $S \subseteq E$. Given these, we prove Algorithm 1 outputs a sparsifier that not only approximates the function itself but also approximates its Lovász extension.

Theorem 13. *Algorithm 1 returns a vector $\boldsymbol{w}$ with expected size at most $O(\frac{B \cdot n^2}{\epsilon^2})$ such that, with probability at least $1 - \delta$, for $F' = \sum_{i \in D} \boldsymbol{w}_i f_i$ it holds that*

$$(1-\epsilon) F'^L(\boldsymbol{x}) \leq F^L(\boldsymbol{x}) \leq (1+\epsilon) F'^L(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in [0,1]^n.$$

Remark 14 (Relation to spectral sparsification of graphs)**.** The cut function of a graph $G = (V, E)$ can be seen as a decomposable submodular function $F(S) = \sum_{e \in E} f_e$, where $f_e(S) = 1$ if and only if $e \cap S \neq \emptyset$ and $e \cap (V \setminus S) \neq \emptyset$. The goal of spectral sparsification of graphs (Spielman and Teng 2011) is to preserve the quadratic form of the Laplacian of G, which can be rephrased as $\sum_{e \in E} f_e^L(\boldsymbol{x})^2$. In contrast, our sparsification preserves $F^L(\boldsymbol{x}) = \sum_{e \in E} f_e^L(\boldsymbol{x})$. Although we can construct a sparsifier that preserves $\sum_{e \in E} f_e^L(\boldsymbol{x})^2$ in the general submodular setting, we adopted the one used here because, in many applications where submodular functions are involved, we are more interested in the value of $\sum_{e \in E} f_e^L(\boldsymbol{x})$ than $\sum_{e \in E} f_e^L(\boldsymbol{x})^2$, and the algorithm for preserving the former is simpler than that for preserving the latter.

Because our algorithm gives an approximation on the Lovász extension, it can be used as a preprocessing step for algorithms working on Lovász extensions such as the ones in (Axiotis et al. 2021; Ene, Nguyen, and Végh 2017). For instance, it improves the running time of Axiotis et al. (2021) from $\widetilde{O}(T_{\mathrm{maxflow}}(n, n+N) \log \frac{1}{\epsilon})$ to $\widetilde{O}(T_{\mathrm{maxflow}}(n, n + \frac{n^2}{\epsilon^2}) \log \frac{1}{\epsilon})$ in cases where each submodular function $f_i \in D$ acts on $O(1)$ elements of the ground set which implies $B = \max_i |\mathcal{B}(f_i)|$ is $O(1)$. An example of such cases is hypergraph cut functions with $O(1)$ sized hyperedges.

Next we discuss several examples for which computing p_i's is a computationally efficient task, thus achieving a polynomial-time algorithm to construct sparsifiers. Recall that the cut function of a graph $G = (V, E)$ can be seen as a decomposable submodular function. In this case, computing each p_e for an edge $e = st \in E$ is equivalent to finding the minimum s-t cut in the graph, which is a polynomial time task. A more general framework is the *submodular hypergraph minimum s-t cut* problem discussed in what follows.

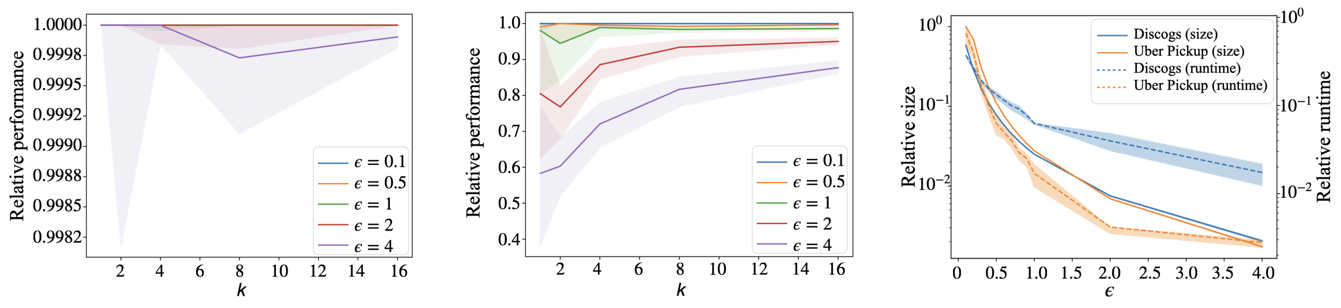
Submodular hypergraphs (Li and Milenkovic 2018; Yoshida 2019). Let $\mathcal{H}$ be a hypergraph with vertex set V and set of hyperedges E where each hyperedge is a subset of vertices V. A submodular function f_e is associated to each hyperedge $e \in E$. In the submodular hypergraph minimum s-t cut problem the objective is

$$\mathrm{minimize}_{S \subset V} \sum_{e \in E} f_e(e \cap S) \quad (6)$$

subject to $s \in S, t \in V \backslash S$. This problem has been studied by Veldt, Benson, and Kleinberg (2020a) and its special cases where submodular functions f_e take particular forms have been studied with applications in semi-supervised learning, clustering, and rank learning (see Li and Milenkovic (2018); Veldt, Benson, and Kleinberg (2020a) for more details). Examples of such special cases include:

- Linear penalty: $f_e(S) = \min\{|S|, |e \setminus S|\}$
- Quadratic Penalty: $f_e(S) = |S| \cdot |e \setminus S|$

We refer to Table 1 in Veldt, Benson, and Kleinberg (2020a) for more examples. These examples are cardinality-based,



(a) Uber pickup (b) Discogs (c)

Figure 1: Figures (a) and (b) show the relative performance of the greedy method on sparsifiers on Uber pickup and Discogs datasets, respectively. Figure (c) shows relative size of sparsifiers and relative runtime of the greedy method on sparsifiers.

that is, the value of the submodular function depends on the cardinality of the input set (see Definition 3.2 of Veldt, Benson, and Kleinberg (2020a)). It is known that if all the submodular functions are cardinality-based, then computing the s-t minimum cut in the submodular hypergraph can be reduced to that in an auxiliary (ordinary) graph (Theorem 4.6 of Veldt, Benson, and Kleinberg (2020a)), which allows us to compute p_e's in polynomial time.

Remark 15. Our sparsification algorithm can also be used to construct *submodular Laplacian* based on the Lovász extension of submodular functions. Submodular Laplacian was introduced by Yoshida (2019) and has numerous applications in machine learning, including in learning ranking data, clustering based on network motifs (Li and Milenkovic 2017), network analysis (Yoshida 2016), and etc.

Experimental Results

In this section, we empirically demonstrate that our algorithm (Algorithm 2) generates a sparse representation of a decomposable submodular function $F : 2^E \to \mathbb{R}_+$ with which we can efficiently obtain a high-quality solution for maximizing F. We consider the following two settings.

Uber pickup. We used a dataset of Uber pickups in New York city in May 2014 consisting of a set R of 564,517 records[1]. Each record has a pickup position, longitude and latitude. Consider selecting k locations as waiting spots for idle Uber drivers. To formalize this problem, we selected a set L of 36 popular pickup locations in the database, and constructed a facility location function $F : 2^L \to \mathbb{R}_+$ as $F(S) = \sum_{v \in R} f_v(S)$, where $f_v(S) = \max_{u \in L} d(u,v) - \min_{u \in S} d(u,v)$ and $d(u,v)$ is the Manhattan distance between u and v. Then, the goal of the problem is to maximize $F(S)$ subject to $|S| \leq k$.

Discogs (Kunegis 2013). This dataset provides information about audio records as a bipartite graph $G = (L, R; E)$, where each edge $(u,v) \in L \times R$ indicates that a label v was involved in the production of a release of a style u. We have $|L| = 383$ and $|R| = 243,764$, and $|E| = 5,255,950$. Consider selecting k styles that cover the activity of as many labels as possible. To formalize this problem, we constructed a maximum coverage function $F : 2^L \to \mathbb{R}$ as $F(S) = \sum_{v \in R} f_v(S)$, where $f_v(S)$ is 1 if v has a neighbor in S and 0 otherwise. Then, the goal is to maximize $F(S)$ subject to $|S| \leq k$.

Figure 1 (a) and (b) show the objective value of the solution obtained by the greedy method on the sparsifier relative to that on the original input function with its 25th and 75th percentiles. Although our theoretical results do not give any guarantee when $\epsilon > 1$, we tried constructing our sparsifier with $\epsilon > 1$ to see its performance. The solution quality of our sparsifier for Uber pickup is more than 99.9% even when $\epsilon = 4$, and that for Discogs is more than 90% performance when $\epsilon \leq 1.0$. The performance for Uber pickup is higher than that for Discogs because the objective function of the former saturates easily. These results suggest that we get a reasonably good solution quality by setting $\epsilon = 1$.

Number of functions and speedups. Figure 1 (c) shows the size, that is, the number of functions with positive weights, of our sparsifier relative to that of the original function and the runtime of the greedy method on the sparsifier relative to that on the original function with their 25th and 75th percentiles when $k = 8$. The size and runtime are decreased by a factor of 30–50 when $\epsilon = 1$. To summarize, our experimental results suggest that our sparsifier highly compresses the original function without sacrificing the solution quality.

Conclusion

Decomposable submodular functions appear in many data intensive tasks in machine learning and data mining. Thus, having a sparsifying algorithm is of great interest. In this paper, we introduce the notion of sparsifier for decomposable submodular functions. We propose the first sparsifying algorithm for these types of functions which outputs accurate sparsifiers with expected size independent of the size of original function. Our algorithm can be adapted to sparsify mixtures of submodular functions. We also study the effectiveness of our algorithm under various constraints such as matroid and cardinality constraints. Our experimental results complement our theoretical results on real world data. This work does not present any foreseeable societal consequence.

[1]Available at https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city

Acknowledgments

Akbar Rafiey was supported by NSERC. Yuichi Yoshida was supported by JST PRESTO Grant Number JP-MJPR192B and JSPS KAKENHI Grant Number 20H05965.

References

Ahn, K. J.; Guha, S.; and McGregor, A. 2012. Graph sketches: sparsification, spanners, and subgraphs. In *PODS*, 5–14.

Ahn, K. J.; Guha, S.; and McGregor, A. 2013. Spectral Sparsification in Dynamic Graph Streams. In *APPROX*, 1–10.

Andoni, A.; Chen, J.; Krauthgamer, R.; Qin, B.; Woodruff, D. P.; and Zhang, Q. 2016. On Sketching Quadratic Forms. In *ITCS*, 311–319.

Axiotis, K.; Karczmarz, A.; Mukherjee, A.; Sankowski, P.; and Vladu, A. 2021. Decomposable Submodular Function Minimization via Maximum Flow. In *ICML*, 446–456.

Bai, W.; Iyer, R. K.; Wei, K.; and Bilmes, J. A. 2016. Algorithms for Optimizing the Ratio of Submodular Functions. In *ICML*, 2751–2759.

Bairi, R.; Iyer, R. K.; Ramakrishnan, G.; and Bilmes, J. A. 2015. Summarization of Multi-Document Topic Hierarchies using Submodular Mixtures. In *ACL*, 553–563.

Bansal, N.; Svensson, O.; and Trevisan, L. 2019. New Notions and Constructions of Sparsification for Graphs and Hypergraphs. In *FOCS*, 910–928.

Batson, J. D.; Spielman, D. A.; and Srivastava, N. 2012. Twice-Ramanujan Sparsifiers. *SIAM J. Comput.*, 41(6): 1704–1721.

Benczúr, A. A.; and Karger, D. R. 1996. Approximating *s-t* Minimum Cuts in $\tilde{O}(n^2)$ Time. In *STOC*, 47–55.

Benczúr, A. A.; and Karger, D. R. 2015. Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs. *SIAM J. Comput.*, 44(2): 290–319.

Chaturvedi, A.; Nguyen, H. L.; and Zakynthinou, L. 2021. Differentially Private Decomposable Submodular Maximization. In *AAAI*, 6984–6992.

Chen, Y.; Khanna, S.; and Nagda, A. 2020. Near-linear Size Hypergraph Cut Sparsifiers. In *FOCS*, 61–72.

Cohen, M. B.; Kelner, J.; Peebles, J.; Peng, R.; Rao, A. B.; Sidford, A.; and Vladu, A. 2017. Almost-linear-time algorithms for Markov chains and new spectral primitives for directed graphs. In *STOC*, 410–419.

Cohen, M. B.; Lee, Y. T.; Musco, C.; Musco, C.; Peng, R.; and Sidford, A. 2015. Uniform Sampling for Matrix Approximation. In *ITCS*, 181–190.

Dobzinski, S.; and Schapira, M. 2006. An improved approximation algorithm for combinatorial auctions with submodular bidders. In *SODA*, 1064–1073.

Dueck, D.; and Frey, B. J. 2007. Non-metric affinity propagation for unsupervised image categorization. In *ICCV*, 1–8.

Edmonds, J. 2001. Submodular Functions, Matroids, and Certain Polyhedra. In *Combinatorial Optimization - Eureka, You Shrink!*, 11–26.

Ene, A.; Nguyen, H. L.; and Végh, L. A. 2017. Decomposable Submodular Function Minimization: Discrete and Continuous. In *NeurIPS*, 2870–2880.

Feige, U. 2006. On maximizing welfare when utility functions are subadditive. In *STOC*, 41–50.

Feige, U.; and Vondrák, J. 2006. Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e. In *FOCS*, 667–676.

Fix, A.; Joachims, T.; Park, S. M.; and Zabih, R. 2013. Structured Learning of Sum-of-Submodular Higher Order Energy Functions. In *ICCV*, 3104–3111.

Gomes, R.; and Krause, A. 2010. Budgeted Nonparametric Learning from Data Streams. In *ICML*, 391–398.

Kapralov, M.; Krauthgamer, R.; Tardos, J.; and Yoshida, Y. 2021a. Spectral Hypergraph Sparsifiers of Nearly Linear Size. In *FOCS*.

Kapralov, M.; Krauthgamer, R.; Tardos, J.; and Yoshida, Y. 2021b. Towards Tight Bounds for Spectral Sparsification of Hypergraphs. In *STOC*, 598–611.

Karger, D. R.; and Levine, M. S. 2002. Random sampling in residual graphs. In *STOC*, 63–66.

Kogan, D.; and Krauthgamer, R. 2015. Sketching Cuts in Graphs and Hypergraphs. In *ITCS*, 367–376.

Kohli, P.; Ladicky, L.; and Torr, P. H. S. 2009. Robust Higher Order Potentials for Enforcing Label Consistency. *Int. J. Comput. Vis.*, 82(3): 302–324.

Kunegis, J. 2013. KONECT: the Koblenz network collection. In *WWW, Companion Volume*, 1343–1350.

Lee, Y. T.; and Sidford, A. 2014. Path Finding Methods for Linear Programming: Solving Linear Programs in Õ(vrank) Iterations and Faster Algorithms for Maximum Flow. In *FOCS*, 424–433.

Lee, Y. T.; and Sun, H. 2018. Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time. *SIAM J. Comput.*, 47(6): 2315–2336.

Li, M.; Miller, G. L.; and Peng, R. 2013. Iterative Row Sampling. In *FOCS*, 127–136.

Li, P.; and Milenkovic, O. 2017. Inhomogeneous Hypergraph Clustering with Applications. In *NeurIPS*, 2308–2318.

Li, P.; and Milenkovic, O. 2018. Submodular Hypergraphs: p-Laplacians, Cheeger Inequalities and Spectral Clustering. In *ICML*, 3020–3029.

Lin, H.; and Bilmes, J. A. 2011. A Class of Submodular Functions for Document Summarization. In *HLT*, 510–520.

Lindgren, E. M.; Wu, S.; and Dimakis, A. G. 2016. Leveraging Sparsity for Efficient Submodular Data Summarization. In *NeurIPS*, 3414–3422.

Madry, A. 2010. Fast Approximation Algorithms for Cut-Based Problems in Undirected Graphs. In *FOCS*, 245–254.

Mahoney, M. W. 2011. Randomized Algorithms for Matrices and Data. *Found. Trends Mach. Learn.*, 3(2): 123–224.

Mirzasoleiman, B.; Badanidiyuru, A.; and Karbasi, A. 2016. Fast Constrained Submodular Maximization: Personalized Data Summarization. In *ICML*, volume 48, 1358–1367.

Mirzasoleiman, B.; Badanidiyuru, A.; Karbasi, A.; Vondrák, J.; and Krause, A. 2015. Lazier Than Lazy Greedy. In *AAAI*, 1812–1818.

Mirzasoleiman, B.; Karbasi, A.; Sarkar, R.; and Krause, A. 2016. Distributed Submodular Maximization. *J. Mach. Learn. Res.*, 17: 238:1–238:44.

Mirzasoleiman, B.; Zadimoghaddam, M.; and Karbasi, A. 2016. Fast Distributed Submodular Cover: Public-Private Data Summarization. In *NeurIPS*, 3594–3602.

Mitrovic, M.; Bun, M.; Krause, A.; and Karbasi, A. 2017. Differentially Private Submodular Maximization: Data Summarization in Disguise. In *ICML*, 2478–2487.

Mitrovic, M.; Kazemi, E.; Zadimoghaddam, M.; and Karbasi, A. 2018. Data Summarization at Scale: A Two-Stage Submodular Approach. In *ICML*, volume 80, 3593–3602.

Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1): 265–294.

Papadimitriou, C. H.; Schapira, M.; and Singer, Y. 2008. On the Hardness of Being Truthful. In *FOCS*, 250–259.

Parambath, S. P.; Usunier, N.; and Grandvalet, Y. 2016. A Coverage-Based Approach to Recommendation Diversity On Similarity Graph. In *RecSys*, 15–22.

Rafiey, A.; and Yoshida, Y. 2020. Fast and Private Submodular and k-Submodular Functions Maximization with Matroid Constraints. In *ICML*, 7887–7897.

Shanu, I.; Arora, C.; and Singla, P. 2016. Min Norm Point Algorithm for Higher Order MRF-MAP Inference. In *CVPR*, 5365–5374.

Soma, T.; and Yoshida, Y. 2019. Spectral Sparsification of Hypergraphs. In *SODA*, 2570–2581.

Spielman, D. A.; and Teng, S. 2011. Spectral Sparsification of Graphs. *SIAM J. Comput.*, 40(4): 981–1025.

Stobbe, P.; and Krause, A. 2010. Efficient Minimization of Decomposable Submodular Functions. In *NeurIPS*, 2208–2216.

Tschiatschek, S.; Iyer, R. K.; Wei, H.; and Bilmes, J. A. 2014. Learning Mixtures of Submodular Functions for Image Collection Summarization. In *NeurIPS*, 1413–1421.

Veldt, N.; Benson, A. R.; and Kleinberg, J. M. 2020a. Hypergraph Cuts with General Splitting Functions. *CoRR*, abs/2001.02817.

Veldt, N.; Benson, A. R.; and Kleinberg, J. M. 2020b. Minimizing Localized Ratio Cut Objectives in Hypergraphs. In *KDD*, 1708–1718.

Vicente, S.; Kolmogorov, V.; and Rother, C. 2009. Joint optimization of segmentation and appearance models. In *ICCV*, 755–762.

Vondrák, J. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, 67–74.

Yoshida, Y. 2016. Nonlinear Laplacian for Digraphs and its Applications to Network Analysis. In *WSDM*, 483–492.

Yoshida, Y. 2019. Cheeger Inequalities for Submodular Transformations. In *SODA*, 2582–2601.