Near-optimal Approximate Discrete and Continuous Submodular Function Minimization

Brian Axelrod
Stanford University
baxelrod@cs.stanford.edu [*]

Yang P. Liu
Stanford University
yangpatil@gmail.com [†]

Aaron Sidford
Stanford University
sidford@stanford.edu [‡]

September 4, 2019

In this paper we provide improved running times and oracle complexities for approximately minimizing a submodular function. Our main result is a randomized algorithm, which given any submodular function defined on n-elements with range $[-1, 1]$, computes an ε-additive approximate minimizer in $\tilde{O}(n/\varepsilon^2)$ oracle evaluations with high probability. This improves over the $\tilde{O}(n^{5/3}/\varepsilon^2)$ oracle evaluation algorithm of Chakrabarty et $al.$ (STOC 2017) and the $\tilde{O}(n^{3/2}/\varepsilon^2)$ oracle evaluation algorithm of Hamoudi et $al.$.

Further, we leverage a generalization of this result to obtain efficient algorithms for minimizing a broad class of nonconvex functions. For any function f with domain $[0, 1]^n$ that satisfies $\frac{\partial^2 f}{\partial x_i \partial x_j} \leq 0$ for all $i \neq j$ and is L-Lipschitz with respect to the L^∞-norm we give an algorithm that computes an ε-additive approximate minimizer with $\tilde{O}(n \cdot \text{poly}(L/\varepsilon))$ function evaluation with high probability.

1 Introduction

A function f which assigns real values to subsets of a finite universe U is *submodular* if it satisfies the *decreasing marginal returns* property, i.e. $f(T \cup \{i\}) - f(T) \leq f(S \cup \{i\}) - f(S)$ for all subsets $S \subseteq T \subseteq U$ and elements $i \notin T$. Such functions are natural, arise in many applications, and have been studied extensively since the 1950s [Cho54, Edm70, McC05, Fuj05, Lov83]. For example, the sizes of cuts in directed graphs or hypergraphs, the rank function of a matroid, and the entropy of subsets of random variables are all submodular. Further the utility functions of agents purchasing a subset of items is often assumed to be submodular. Given their prevalence, the optimization of submodular functions is fundamental to combinatorial optimization and both submodular function maximization [FMV11, CVZ14] and minimization have been studied extensively.

In this work, we focus on submodular function minimization (SFM), i.e. finding a subset $S \subseteq U$ minimizing $f(S)$. As submodular functions need not be monotone and SFM generalizes multiple fundamental combinatorial optimization problems, including computing s-t minimum cuts in directed graphs and hypergraphs, SFM is nontrivial. More recently, SFM has been applied to many problem domains, such as image segmentation [BVZ99, KKT08, KT10], speech analysis [LB10, LB11a, LB11b], and machine learning [Bac13, KG11].

In this paper we consider the standard and well-studied model for SFM where f can be accessed only through an *evaluation oracle* which when queried with $S \subseteq U$ returns $f(S)$. For simplicity, we measure the complexity of our algorithms by the number of queries, i.e. *oracle calls*, or *function calls*, that we make to the evaluation oracle; the additional runtime of all new algorithms in this paper can be nearly linear in the number of oracle calls. Throughout the introduction, we refer to the time needed for an oracle call as EO. An amazing result is that SFM can be solved with a number of queries polynomial in n, the number of elements in the universe J. This was demonstrated initially via the ellipsoid algorithm [GLS84] spawning a long line of work faster algorithms.

Previous research on algorithms for SFM has focused on three main regimes: strongly polynomial, weakly polynomial, and pseudopolynomial time [Wol76, Fuj80, CJK14, LJ15]. Letting M be the maximum absolute value of the integer-valued submodular function f on an n-element universe, strongly polynomial, weakly polynomial, and pseudopolynomial refer to algorithms whose runtimes are all polynomial in n and independent of M, logarithmic in M, and polynomial in M respectively. For these regimes, the best known dependence on n in terms of the number of oracle calls needed has a clear picture: nearly cubic in the strongly polynomial regime [LSW15], quadratic in the weakly polynomial regime [LSW15], and linear in the pseudopolynomial regime [CLSW17].

We can also view these results in a slightly different way. Instead, let f be a *real-valued* submodular function with range $[-1, 1]$, and consider the goal of finding an ε-additive approximate minimizer. Here, it is natural to study approximate SFM algorithms whose runtimes are independent of ε, depends logarithmically on ε, and depends polynomially on ε. These correspond to the strongly polynomial, weakly polynomial, and approximate regimes respectively [CLSW17]. The best known runtime in the strongly polynomial and weakly polynomial regimes continue to be cubic and quadratic respectively in this view. However, despite the well-studied nature of SFM and clear picture in terms of M dependencies, the runtime of approximate SFM is less understood. The state-of-the-art such runtime is $\widetilde{O}(n^{3/2}/\varepsilon^2)$[1] which was achieved by the contemporary work of [HRRS19] and improved upon the $\widetilde{O}(n^{5/3}/\varepsilon^2)$ runtime algorithm of [CLSW17]. In this paper, we close this gap and give a nearly linear time, $\widetilde{O}(n/\epsilon^2)$ time algorithm for ε-approximate SFM. .

---

[1] Throughout, we use $\widetilde{O}$ to hide $\text{poly}(\log n, \log(1/\varepsilon), \log M)$ factors.

Nonconvex optimization: Another key motivation for the results of this paper is obtaining provably faster algorithms for obtaining global minimizers of broad classes of non-convex functions. Consider the problem $\min_{x \in \mathcal{X}} f(x)$ for a function $f : \mathbb{R}^n \to \mathbb{R}$. For *convex optimization*, when f and $\mathcal{X}$ are both convex, there are numerous methods for solving this problem: gradient descent, cubic regularized newton, cutting plane, etc. On the other hand, the situation for nonconvex optimization, i.e. finding the global minimum of a nonconvex function f in general, is computationally intractable: finding an ε-approximate minimizer for a k-times continuously differentiable $f : \mathbb{R}^n \to \mathbb{R}$ requires $\Omega((1/\varepsilon)^{n/k})$ evaluations of the function and its first k derivatives, ignoring problem dependent parameters such as the Lipschitz smoothness of f, etc. [NY83].

Nevertheless, as many practical problems, e.g. training neural networks and matrix completion, are nonconvex, it still important to understand what guarantees we can achieve for nonconvex optimization. Because computing an ε-approximate global minimizer of a general convex function, as discussed, intractable in general, some work has focused on finding ε-approximate stationary points or approximate local minima [NP06, Nes12, CDHS16, BGM$^+$17]. In addition, there are specific problems such as matrix completion where all local minima are in fact global minima [GLM16]. Given these results, it would be tantalizing to find large classes of nonconvex functions for which we can find a global minimizer; however, this has been a challenging task achieved in only a few situations [GG17, NGGD18, HSS19].

In recent work, Bach [Bac19] considered a class of nonconvex functions $f : [0,1]^n \to \mathbb{R}$ satisfying $\frac{\partial^2 f}{\partial x_i \partial x_j} \leq 0$ for all $i \neq j$, which are a continuous generalization of submodular functions. Several interesting functions satisfy this property, e.g. $f(x) = x^T Q x$ where Q is symmetric with negative off diagonal entries and $f(x) = g(\sum_i c_i x_i)$ for some concave function $g : \mathbb{R} \to \mathbb{R}$ and positive weights c_i. The former function is neither convex nor concave, and the latter is concave.

Despite the fact that these functions can be nonconvex, [Bac19] provided an algorithm to find ε-approximate global minimizers in time polynomial in n, ε, and problem dependent parameters. In our work, we improve upon Bach's cubic dependence on n and show that these functions can in fact be minimized *almost as efficient as convex functions* in terms of the best known methods: nearly linear in the dimension n, and polynomial in ε and the L^∞ Lipschitz constant.

1.1 Our results

In this paper we address a key open problem in the work of [CLSW17] and [HRRS19] regarding whether we can achieve a nearly linear runtime for approximate SFM. We resolve this problem in this paper, giving an $\widetilde{O}(n\varepsilon^{-2} \cdot \text{EO})$ time algorithm for approximate SFM. This also directly improves the previous pseudopolynomial time algorithms in terms of their dependence on M, the range of an integer submodular function. Further, due to the subgradient oracle lower bound given in [CLSW17] and the fact that a subgradient oracle yields more information than an evaluation oracle, this bound is known to be optimal up to the dependence on ε.

Theorem 1 (Nearly linear time submodular function minimization)**.** *Given a submodular function* $f : \{0,1\}^n \to [-1,1]$ *and an* $\varepsilon > 0$, *we can compute a random set* S *with*

$$\mathbb{E}[f(S)] \leq \min_{T \subseteq [n]} f(T) + \varepsilon$$

in $\widetilde{O}(n/\varepsilon^2)$ *calls to an evaluation oracle for* f.

We can convert the guarantee of Theorem 1 to a w.h.p.[2] guarantee as follows. Note that the probability that $f(S) > \min_{T \subseteq [n]} f(T) + 2\varepsilon$ is at most $1/2$ by Markov's inequality. Consequently, we can amplify to the success probability to $1 - \frac{1}{\text{poly}(n)}$ by running the algorithm $O(\log n)$ times to half the expected error and outputing the smallest value.

We also achieve *sublinear* time algorithms for pseudopolynomial SFM in settings where we know that the minimizer of f is s-sparse, i.e. only has s nonzero entries.

Theorem 2 (Pseudopolynomial submodular function minimization). *Consider an integer valued submodular function $f : \{0,1\}^n \to [-M, M]$ with s-sparse minimizer, i.e. there is a set $S^{\text{opt}} \in \text{argmin}_{S \subseteq \{0,1\}^n} f(S)$ satisfying $|S^{\text{opt}}| \le s$. Then we can compute an exact minimizer of f in $\widetilde{O}(sM^2)$ calls to an oracle for f w.h.p.*

Note that for small M (say $M = \widetilde{O}(1)$) and $s = O(n^{1-\delta})$ for some $\delta > 0$, this algorithm uses a number of oracle calls to f which is *sublinear* in n. This is the first sublinear time algorithm for SFM. The previous bottleneck for obtaining such sublinear results was that it seemed necessary to compute a full subgradient of the Lovasz extension, a well-known continuous extension of submodular functions, which naively requires $\Omega(n)$ oracle calls. We overcome this by designing an algorithm that computes all $O(M)$ nonzero entries of the subgradient at 0 with $\widetilde{O}(M^2)$ oracle calls.

These results makes progress towards completing the picture for SFM algorithms: strongly polynomial algorithms use a cubic number of queries, weakly polynomial algorithms use a quadratic number of queries, and pseudopolynomial/approximate algorithms make a linear number of queries.

Following the work of Bach [Bac19], our results extend to a more general class of submodular functions not necessarily defined on $\{0,1\}^n$, such as those defined on $[k]^n$ for a positive integer k. Leveraging this result, we obtain a nearly linear time algorithm for computing approximate minimizers of a class of nonconvex functions studied by Bach [Bac19], improving upon the cubic running time given in that paper.

Theorem 3 (Nearly linear time continuous submodular function minimization). *Let $f : [0,1]^n \to \mathbb{R}$ be a twice differentiable function with $\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \le 0$ for all $i \ne j$. There is an algorithm that computes an ε-additive approximate minimizer of f in $\widetilde{O}(nL^6/\varepsilon^6)$ function evaluation calls w.h.p., where L is the L^∞-Lipschitz constant of f.*

1.2 Previous Work

The first polynomial time algorithm for SFM was via the ellipsoid algorithm [GLS84]. This spawned a line of work on faster algorithms (see Table 1 for the state-of-the-art bounds) and combinatorial algorithms [Cun85,IFF00,Sch00,IO09], which were achieved later. Building on a long line of work on SFM, Lee *et al.* [LSW15] gave the current state-of-the-art running times for weekly polynomial SFM, $O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$, and strongly polynomial SFM, $O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$. See [LSW15] for more comprehensive coverage of previous improvements.

Additionally, there has been work towards understanding *pseudopolyomial* algorithms for SFM. Specifically, the Fujishige-Wolfe [Wol76, Fuj80] algorithm which is often used in practice can be shown to run in pseudopolynomial time $O(n^2M^2 \cdot \text{EO} + n^3M^2)$ [CJK14, LJ15]. More recently, Chakrabarty *et al.* [CLSW17] gave a nearly linear pseudopolynomial algorithms for SFM with run-time $\widetilde{O}(nM^3 \cdot \text{EO})$. Additionally, they studied the problem of approximate SFM, that is minimizing

---

a real-valued submodular function f with range $[-1, 1]$ to additive ε error. They achieved a sub-quadratic $\widetilde{O}(n^{5/3}\varepsilon^{-2} \cdot \text{EO})$ time algorithm for this problem. They also studied SFM in the case where f is known to have a s-sparse minimizer, i.e. the minimizer of f has only s nonzero entries, achieving an $\widetilde{O}\left((n + sn^{2/3})\text{EO}\epsilon^{-2}\right)$ algorithm.

Simultaneously with this work, Hamoudi *et al.* [HRRS19] improved the runtime of approximate SFM to $\widetilde{O}(n^{3/2}\varepsilon^{-2} \cdot \text{EO})$. Additionally, they also achieved a $\widetilde{O}(n^{5/4}\varepsilon^{-5/2} \cdot \text{EO})$ quantum algorithm for approximate SFM through a new method for sampling with high probability T independent elements from any discrete probability distribution of support size n in time $O(\sqrt{Tn})$.

Regime	Previous Best Running Time	Our Result
Strongly Polynomial	$O(n^3 \log^2 n \cdot \text{EO} + n^4 \log^{O(1)} n)$ [LSW15]	
Weakly Polynomial	$O(n^2 \log nM \cdot \text{EO} + n^3 \log^{O(1)} nM)$ [LSW15]	
Pseudopolynomial	$\widetilde{O}(nM^3 \cdot \text{EO})$ [CLSW17]	$\widetilde{O}(nM^2 \cdot \text{EO})$
ε-Approximate	$\widetilde{O}(n^{3/2} \cdot \text{EO}/\varepsilon^2)$ [HRRS19]	$\widetilde{O}(n \cdot \text{EO}/\varepsilon^2)$
Sparse Pseudopolynomial	$\widetilde{O}((n + sM^3) \cdot \text{EO})$ [CLSW17]	$\widetilde{O}(sM^2 \cdot \text{EO})$

Table 1: Running times for minimizing a submodular function f on subsets of an n element set. EO denotes the time needed to make an oracle call to f. In all but the approximate SFM regime, f is integer valued with maximum absolute value M. In the approximate SFM regime, f is real valued with range $[-1, 1]$. s is the sparsity of the minimizer of f. Table adapted from [CLSW17].

Additionally, there has been work towards extending the notion of submodularity beyond functions defined on subsets of a universe U. Bach [Bac19] has shown that the notion of submodularity extends naturally to functions defined on $[k]^n$ (instead of $\{0, 1\}^n$) and even to functions defined on continuous domains such as $[0, 1]^n$. This work shows how to extend the classical polynomial time algorithms for submodular optimization to this setting, and gives polynomial time algorithms for optimizing a large class of nonconvex functions.

1.3 Organization

For the remainder of the introduction, we give an overview for our techniques in Section 1.4. In Section 2, we state the necessary preliminaries for our algorithms. In Section 3 we give our main algorithm for nearly linear time SFM and prove Theorem 2. In Section 4 we give our sublinear time algorithms for pseudopolynomial SFM when the minimizer is sparse and prove Theorem 2. Finally, in Section 5 we extend our earlier results to submodular functions on the domain $[k]^n$ and $[0, 1]^n$, and prove Theorem 3 in Section 5.3.

1.4 Overview

Here we give a less technical overview of the ideas behind our algorithm. For simplicity, we only describe our algorithm in the situation when f is a standard submodular function on $\{0, 1\}^n$. For more technical discussion and discussion about the sparse regime and submodular functions over $[k]^n$, see Section 3, Section 4, and Section 5.

Our algorithms, like those of [CLSW17], are based on projected stochastic subgradient descent on the Lovasz extension of f, which is a well-known continuous convex extension of f (see Definition 2.1). The algorithms of [CLSW17] exploited submodularity to build a data structure and

get gradient updates with fewer evaluations than the $O(n)$ required by the naïve method. To obtain our result we leverage the techniques built by [CLSW17] but show that a more efficient binary tree based data structure can be built to support gradient estimates with little preprocessing.

More precisely, the algorithm of [CLSW17] computes $x_0, x_1, \cdots, x_T$, a (stochastic) sequence of points, where each x_{i+1} is computed by taking a stochastic subgradient step from x_i. To do this, the algorithm writes the gradient at x_{i+1} (we'll denote it as $g(x_{i+1})$) as the sum of smaller terms of the form $g(x_a) - g(x_b)$ and $g(x_0)$, estimates each, and sums them. As the number of terms summed increases, the variance of the estimate grows and the convergence rate of subgradient descent decreases. To achieve there fastest algorithm [CLSW17] thereby trades off leveraging such stochastic estimates and recomputing the initial estimator.

In this paper we improve this datastructure by, as we step through the trajectory $x_1, \ldots, x_T$, choosing to evaluate $g(x_a) - g(x_b)$ at carefully chosen intervals along the trajectory. This allows us to amortize the maintenance of the data structure while simultaneously maintaining a low variance of the resulting stochastic gradient. This leads to our nearly linear time algorithm. We hope that this general framework of using data structures to maintain the ability to do point updates and sample gradients can find uses in other optimization methods where we desire sublinear gradient calls, such as coordinate descent.

In order to extend our results to the domain $[k]^n$, we use the continuous extension of a submodular function f developed by Bach [Bac19], which is the analogue of the Lovasz extension. We show that our algorithms extend to this setting.

Finally, we explain our key ingredient to obtaining sublinear time algorithms in the regime where f is integer valued with maximum absolute value M and has a sparse minimizer. The main idea behind the algorithm is to compute an initial subgradient at 0 that doesn't require computing all n coordinates of the subgradient, which naïvely requires n function calls. To do this, we use that the origin has many subgradients, and develop an algorithm that can find one such subgradient for which we can compute all its nonzero entries in $\widetilde{O}(M^2)$ function calls. After this, we can simply plug this initial subgradient into our earlier algorithms and get the desired result.

2 Preliminaries

Here we provide notation and basic facts about classic submodular functions. Preliminaries for submodular functions on $[k]^n$ and continuous submodular functions are deferred to Section 5.

Miscellaneous notation. We let $[n] \stackrel{\text{def}}{=} \{1, 2, \ldots, n\}$. For $a, b \in \mathbb{R}$ we let $[a, b] \stackrel{\text{def}}{=} \{x : a \leq x \leq b\}$. For permutation $P = \{P_1, P_2, \cdots, P_n\}$ of $[n]$, we let $P[j] \stackrel{\text{def}}{=} \{P_1, P_2, \cdots, P_j\}$ be the set containing the first j elements of P. For a point $x \in \mathbb{R}^n$ we call a permutation P of $[n]$ *consistent* with x if $x_{P_1} \geq x_{P_2} \geq \cdots \geq x_{P_n}$. We let $e_1, e_2, \cdots, e_n$ denote the standard basis vectors for $\mathbb{R}^n$, so that e_i is the vector with a 1 in the i-th coordinate and 0 in all other coordinates. We call a vector s-sparse if it has at most s nonzero entries.

Submodular functions. Let $\{0, 1\}^n \subseteq \mathbb{R}^n$ denote the set of n-tuples, where each coordinate is either 0 or 1. There is a natural bijection between $x \in \{0, 1\}^n$ and subsets $S \subseteq [n]$ where x_i being 1 corresponds to element i being in the set. We use these interchangeably. Throughout, we let $f : \{0, 1\}^U \to \mathbb{R}$ be the submodular function we are trying to optimize, where U is a ground set. Without loss of generality, we assume $U = [n]$. Additionally, we assume that $f(\emptyset) = 0$, which we

can enforce by subtracting a constant from all values of f while preserving submodularity. We say that a function $f : \{0,1\}^n \to \mathbb{R}$ is submodular if it satisfies the property of decreasing marginal returns, specifically for all sets $S \subseteq T \subseteq [n]$ and element $i \notin T$, we have

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

An alternate (but equivalent) definition is that for all $S, T \subseteq [n]$ we have that

$$f(S) + f(T) \geq f(S \cap T) + f(S \cup T).$$

In this work, we measure the complexity of our algorithms through the number of calls we make to an evaluation oracle for f, as the additional runtime of all new algorithms in this paper can be nearly linear in the number of oracle calls.

Lovasz extension. The Lovasz extension is a well-known continuous, convex extension of a submodular function $f : \{0,1\}^n \to \mathbb{R}$ to a function $\hat{f} : [0,1]^n \to \mathbb{R}$. We now state its definition.

Definition 2.1 (Lovasz extension). *Given a submodular function $f : \{0,1\}^n \to \mathbb{R}$, the* Lovasz *extension of f, denoted as $\hat{f} : [0,1]^n \to \mathbb{R}$, is defined for any $x \in [0,1]^n$ as*

$$\hat{f}(x) = \sum_{j=1}^{n} (f(P[j]) - f(P[j-1])) x_{P_j}, \tag{1}$$

where $P = \{P_1, P_2, \cdots, P_n\}$ is a permutation which is consistent with x.

We leverage the following well known properties of the Lovasz extension [Lov83, Fuj05].

Theorem 4. *Let $f : \{0,1\}^n \to \mathbb{R}$ be a submodular function, and let $\hat{f}$ be its Lovasz extension. We have that: $\hat{f}$ is convex; for all $x \in \{0,1\}^n$, $\hat{f}(x) = f(x)$; and $\min_{x \in [0,1]^n} \hat{f}(x) = \min_{S \subseteq [n]} f(S)$. Additionally, the vector $g(x) \in \mathbb{R}^n$ defined by $g(x)_{P_j} \stackrel{\text{def}}{=} f(P[j]) - f(P[j-1])$ for $1 \leq j \leq n$ is a subgradient of $\hat{f}$ at x, where $P = (P_1, P_2, \cdots, P_n)$ is any permutation consistent with x.*

Note that the vector $g(x)$ as defined in Theorem 4, despite being a subgradient of $\hat{f}$ of x, only depends on P. Thus, sometimes we define the *gradient (at zero) associated with permutation P*, denoted g^P, as $g^P_{P_j} \stackrel{\text{def}}{=} f(P[j]) - f(P[j-1])$ for $1 \leq j \leq n$.

We now explain (and this is standard) that given a point $x \in [0,1]^n$, we can find a set $S \subseteq [n]$ with $f(S) \leq \hat{f}(x)$ in $O(n)$ oracle calls to f. In other words, we only need to pay an extra $\widetilde{O}(n)$ oracle calls to convert an approximate minimizer of the Lovasz extension $\hat{f}$ of f to an approximate minimizer of f itself. For completeness, we state this as a lemma and prove it below.

Lemma 2.2 (Going from $\hat{f}$ to f). *For a point $x \in [0,1]^n$, we can in $O(n)$ oracle calls compute a set S such that $f(S) \leq \hat{f}(x)$. In particular, we can go from an ε-additive approximate minimizer of $\hat{f}$ to an ε-additive approximate minimizer of f in $O(n)$ oracle calls.*

Proof. We can rewrite Eq. (1) as

$$\hat{f}(x) = f(P[n]) x_{P_n} + \sum_{j=1}^{n-1} f(P[j])(x_{P_j} - x_{P_{j+1}}),$$

thus $\hat{f}(x)$ is a non-negative linear combination of $f(\emptyset), f(P[1]), f(P[2]), \cdots, f(P[n])$. Therefore, either $f(\emptyset) \leq \hat{f}(x)$ or there is an $1 \leq i \leq n$ with $f(P[i]) \leq \hat{f}(x)$, as desired. $\square$

Subgradient descent. Our algorithms are primarily based on (projected stochastic) subgradient descent. For a convex function f on a convex compact set $S \subseteq \mathbb{R}^n$, we say that a vector g is a *subgradient* of f at $x \in S$ if for all $y \in S$ we have that

$$f(y) - f(x) \geq g^T(y - x).$$

We let $\partial f(x)$ be the set of all subgradients of f at x. A *subgradient oracle* for f is an algorithm which at a point $x \in S$ returns a vector g with $g \in \partial f(x)$. A *stochastic subgradient oracle* for f is an algorithm which at a point $x \in S$ returns a stochastic vector $\mathbf{g}(x)$ with $\mathbb{E}[\mathbf{g}(x)] \in \partial f(x)$.[3] Finally, intermediate points computed during projected subgradient descent may lie outside S. We define the *projection* of a point y onto S to be

$$\mathrm{proj}(y, S) = \mathrm{argmin}_{x \in S} \|x - y\|_2^2.$$

We now state a theorem which contains the guarantees of projected stochastic subgradient descent which we use. The version we state is adapted from [Bub15] and suffices for our purposes.

Theorem 5 (Projected stochastic subgradient descent [Bub15])**.** *Let f be a convex function on a compact convex set $S \subseteq \mathbb{R}^n$ and $\mathbf{g}$ be a stochastic subgradient oracle for f. Define parameters $R^2 \stackrel{\text{def}}{=} \max_{x \in S} \frac{1}{2} \|x\|_2^2$, B such that $\mathbb{E}[\|\mathbf{g}(x)\|_2^2] \leq B^2$ for all $x \in S$ and consider the following iterative algorithm*

$$x_1 \stackrel{\text{def}}{=} \mathrm{argmin}_{x \in S} \|x\|_2^2$$

and

$$x_{i+1} = \mathrm{proj}(x_i - \eta \mathbf{g}(x_i), S) \text{ for } i \in [T-1]$$

Then for $\eta = \frac{R}{B}\sqrt{\frac{2}{T}}$, we have that

$$\mathbb{E}\left[f\left(\frac{1}{T} \sum_{i=1}^{T} x_i \right) \right] \leq \min_{x \in S} f(x) + RB\sqrt{\frac{2}{T}}.$$

Note that for $T = 2R^2 B^2 / \varepsilon^2$ in Theorem 5 we achieve additive error ε off the minimum function value in expectation.

3 Submodular Function Minimization over $\{0,1\}^n$

In this section we present our improved algorithm for SFM over $\{0,1\}^n$. For a submodular function $f : \{0,1\}^n \to \mathbb{R}$, our algorithms perform projected stochastic subgradient descent on the Lovasz extension $\hat{f}$ of f.

Looking at the guarantees of Theorem 5, we want to design an algorithm that can compute stochastic subgradients with low expected ℓ_2 norm without having to make many oracle calls to f. Specifically, in the case of Theorem 1, we show how to construct an algorithm that

- Computes a sequence of points $x_1, x_2, \cdots, x_T \in [0,1]^n$ and stochastic subgradients $\mathbf{g}^i$ of f at x_i, where $x_1 = 0$ and $x_{i+1} = \mathrm{proj}(x_i - \eta \mathbf{g}^i, [0,1]^n)$.

- Makes $\widetilde{O}(T)$ oracle calls to f.

---

[3]Throughout, we use boldface (e.g. $\mathbf{g}$) for stochastic variables and normal text (e.g. g) for not stochastic variables

- Each stochastic subgradient $\mathbf{g}^i$ is 1-sparse.

- Each stochastic subgradient $\mathbf{g}^i$ has $\mathbb{E}[\|\mathbf{g}^i\|_2^2] = \widetilde{O}(1)$.

By the guarantees of Theorem 5, choosing $T = \widetilde{O}(n/\varepsilon^2)$ suffices to prove Theorem 1, as $R^2 = n$ and $B^2 = \widetilde{O}(1)$.

3.1 Subgradients of the Lovasz extension

Here we state important results on the struture of the subgradients of the Lovasz extension. We use this structure in order to sample stochastic subgradients of $\hat{f}$ in sublinear time. Lemma 3.1 is due to Jegelka and Blimes [JB11] (also Hazan and Kale [HK12]). All of Lemma 3.1, Lemma 3.2, and Lemma 3.3 were proven in [CLSW17].

The first lemma is a bound on the L^1 norm of the subgradients.

Lemma 3.1. *For a submodular function $f : \{0,1\}^n \to [-M, M]$, all subgradients g of the Lovasz extension satisfy $\|g(x)\|_1 \leq 3M$.*

The second lemma allows us to relate the gradients of two points $x, y \in \{0,1\}^n$ whose difference $x - y$ is a strictly positive (or negative) vector.

Lemma 3.2. *Let $x \in [0,1]^n$ and let $d \in \mathbb{R}_{\geq 0}^n$ be such that $y = x + d$ (resp. $y = x - d$). Let S denote the non-zero coordinates of d. Then for all $i \notin S$ we have $g(x)_i \geq g(y)_i$ (resp. $g(x)_i \leq g(y)_i$).*

The final lemma allows us to efficiently compute the sum of multiple contiguous coordinates of a subgradient.

Lemma 3.3. *Let $x \in [0,1]^n$ and let P be the permutation consistent with x. Then we have for any integers $1 \leq a \leq b \leq n$ that*

$$\sum_{i=a}^{b} g(x)_{P_i} = f(P[b]) - f(P[a-1]).$$

3.2 Nearly linear time approximate submodular function minimization

In this section we provide a nearly linear time algorithm for minimizing a submodular function $f : \{0,1\}^n \to [-1,1]$ to additive error ε. We give a randomized algorithm that uses at most $\widetilde{O}(n/\varepsilon^2)$ oracle calls to f that computes a point $x \in [0,1]^n$ with $\hat{f}(x) \leq \min_T f(T) + \varepsilon$, where $\hat{f}$ is the Lovasz extension of f.

Our algorithm follows the broad framework of [CLSW17] which minimizes the Lovasz extension using projected stochastic subgradient descent. By Theorem 5, this algorithm yields an ε-additive approximate minimizer in $\widetilde{O}(n/\varepsilon^2)$ provided each subgradient has expected $\widetilde{O}(1)$ ℓ_2 norm. Because naïvely computing a full subradient gradient g of $\hat{f}$ at x requires $\Omega(n)$ oracle calls, to achieve our runtime improvements we must do something more sophisticated to compute stochastic subgradients. To overcome this issue, as in [CLSW17], we leverage Lemma 3.1. This lemma implies that there is stochastic gradient oracle which outputs subgradients which are both sparse and have low ℓ_2 norm. Indeed, because $\|g\|_1 \leq 3$ (by Lemma 3.1) for all subgradients g of the Lovasz extension, we can compute a 1-sparse stochastic subgradient $\mathbf{g}$ with $\mathbb{E}[\|\mathbf{g}\|_2^2] \leq \|g\|_1^2 \leq 9$: sample $\mathbf{g} = \mathrm{sign}(g_i)\|g\|_1 e_i$ with probability $|g_i|/\|g\|_1$.

While Lemma 3.1 does give a sparse sparse stochastic subgradient oracle with low ℓ_2 norm, a naïve implementation would require knowing all of g and therefore naively, $\Omega(n)$ oracle calls. To get around this issue, a key insight of [CLSW17] was that if g was guaranteed to have all positive coordinates, we could use a binary search to sample a stochastic subgradient with $O(1)$ variance in $\widetilde{O}(1)$ oracle calls by applying Lemma 3.3 to sample recursively. This procedure simply samples an interval with probability proportional to the sum of its coordinates, computing the sum using Lemma 3.3 (further details are given in the proof of Lemma 3.4 in Appendix B). Unfortunately, this only works in the case where all coordinates of g are positive, as then there is no cancellation when we compute the sum of coordinates in an interval.

However, imagine that we have already computed the gradient g^0 at x_0 (the starting point of our method) and $x_1 = x_0 - \eta \mathbf{g}^0$ where $\mathbf{g}^0$ is a 1-sparse stochastic subgradient at x_0. To sample a stochastic subgradient $\mathbf{g}^1$ at x_1 we write $g^1 = g^0 + (g^1 - g^0)$. In order to sample $\mathbf{g}^1$, we sample an estimate of g^0, an estimate of $g^1 - g^0$, and sum them. Call this estimate $\mathbf{d}$. If we could efficiently sample a 1-sparse $O(1)$ variance estimate of $g^1 - g^0$, then the resulting estimate $\mathbf{d}$ would be 2-sparse with $O(1)$ variance. To get $\mathbf{g}^1$ simply sample twice a random nonzero coordinate of $\mathbf{d}$. Thus $\mathbf{g}^1$ would be 1-sparse with $O(1)$ variance.

To efficiently sample an estimate of $g^1 - g^0$, [CLSW17] noted that if the difference $x_1 - x_0$ is 1-sparse, then by submodularity one can show that $g^1 - g^0$ can be split into $O(1)$ intervals, each of which is either all positive or all negative. We can then sample this efficiently by the same algorithm for sampling all positive gradients above. This intuition is formalized and generalized in the following lemma which is a slight modification of Lemma 12 proven in [CLSW17]. It gives us the ability to efficiently sample a sparse, low variance estimate of $g(x) - g(y)$ where $x - y$ is sparse. For example, in the paragraph above where $x_1 - x_0$ is 1-sparse, we could efficiently sample a 1-sparse estimate of $g^1 - g^0$.
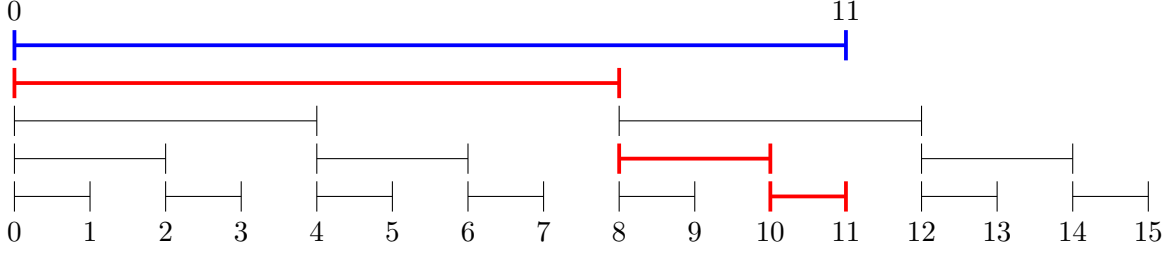
Lemma 3.4. *Let $f : \{0,1\}^n \to [-1,1]$ be a submodular function with Lovasz extension $\hat{f}$. Let g denote the subgradients of $\hat{f}$. Let $x, y \in [0,1]^n$ be vectors such that $y - x$ is k-sparse. There is a data structure which after $O(k)$ calls to f of preprocessing, supports the following: sample a 1-sparse random variable $\mathbf{z}$ with $\mathbb{E}[\mathbf{z}] = g(y) - g(x)$ and $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(1)$ in $\widetilde{O}(1)$ calls to f. Preprocessing is called through $\mathrm{PROCESS}(x, y, f)$, and the sampling is called through $\mathrm{SAMPLE}(x, y, f)$.*

In other words, Lemma 3.4 implies for fixed x, y, we build a data structure with $O(k)$ oracle calls that supports sampling estimates of $g(y) - g(x)$ using $\widetilde{O}(1)$ oracle calls per sample We give the proof in Appendix B for completeness. Careful application of this lemma and the idea of sampling from gradient differences yields the runtimes in [CLSW17] and, with modification to the datastructure, [HRRS19].

Where we depart from [CLSW17] (and improve upon it) is how we use the data structure of Lemma 3.4. Recall that at iteration t, we want to sample an estimate of $g(x_t)$. Instead of using $g(x_t) - g(x_0)$ as above, we will carefully choose a short sequence, $x_{i_0}, x_{i_1}, \ldots, x_{i_m}$, where $i_0 = 0$ and $i_m = t$ for $m = \widetilde{O}(1)$. Now, we sample an estimate of $g(x_t)$ using the identity $g(x_t) = g(x_0) + \sum_{j=0}^{m-1} \left[g(x_{i_{j+1}}) - g(x_{i_j}) \right]$. Specifically, we sample an estimate of $g(x_0)$ and each of the remaining terms $g(x_{i_{j+1}}) - g(x_{i_j})$ using Lemma 3.4, and sum the estimates. Note that the sum is $\widetilde{O}(1)$-sparse and has $\widetilde{O}(1)$-variance by Lemma 3.4. Now, we can sample a 1-sparse estimate of this sum with $\widetilde{O}(1)$ variance. The key to our algorithm is then choosing the sequence so that the amortized cost of preprocessing all segments $(x_{i_j}, x_{i_{j+1}})$ is $\widetilde{O}(1)$ per iteration.

It suffices to choose the segments using the binary representation of the step counter t. For

Figure 1: Intervals processed in line 12 and a decomposition of $[0, 11]$

example, for 11 (1011 in binary) we would choose $0, 8, 10, 11$ ($0, 1000, 1010, 1011$ in binary). See Fig. 1 for the corresponding segments.

At this point, we are ready to state our algorithm.

Algorithm 1 SFM(f, ε). Takes a submodular function $f : \{0, 1\}^n \to [-1, 1]$ and returns a random point $x \in [0, 1]^n$ with $\mathbb{E}[\hat{f}(x)] \le \min_T f(T) + \varepsilon$.

1: $T \leftarrow \widetilde{O}(n/\varepsilon^2)$.
2: $x_0 \leftarrow 0 \in \mathbb{R}^n$.
3: $g^0 \leftarrow g(x_0)$.
4: **for** $i = 1$ to T **do**
5:      $b \leftarrow$ the number of 1 bits in the binary representation of $i - 1$.
6:      $k_0 \leftarrow i - 1$, $k_{j+1} \leftarrow k_j - 2^{\nu_2(k_j)}$ for $0 \le j \le b - 1$.   ▷ $\nu_2(y)$ is the maximum integer t such that 2^t divides y
7:      $\mathbf{d} \leftarrow \|g^0\|_1 \cdot \text{sign}(g_k^0)e_k$ with probability $|g_k^0|/\|g^0\|$.   ▷ Estimate of g^0
8:      $\mathbf{g}^{\star i} \leftarrow \mathbf{d} + \sum_{j=0}^{b-1} \text{SAMPLE}(x_{k_{j+1}}, x_{k_j}, f)$.
9:      Let $c_1, c_2, \cdots, c_s$ be the nonzero coordinates of $\mathbf{g}^{\star i}$.   ▷ $s \le b + 1$
10:     $\mathbf{g}^i \leftarrow s \cdot \mathbf{g}_{c_k}^{\star i} \cdot e_{c_k}$ with probability $\frac{1}{s}$.   ▷ $\mathbf{g}^i$ is 1-sparse
11:     $x_i \leftarrow \text{proj}(x_{i-1} - \eta \mathbf{g}^i, [0, 1]^n)$.
12:     PROCESS$(x_{i-2^{\nu_2(i)}}, x_i, f)$.
13: **return** $\frac{1}{T+1} \sum_{i=0}^{T} x_i$.

Description of Algorithm 1. Lines 1, 2, 3 initialize the starting point, number of iterations, and gradient g^0 at the the initial point. Line 4 corresponds to the projected stochastic gradient descent loop. Lines 5, 6 compute the segments using the binary representation of the iteration counter i. Lines 7, 8 use the precomputed data structures to sample a stochastic subgradient in $\widetilde{O}(1)$ time. Lines 9, 10 turn this stochastic subgradient into a $1-$sparse stochastic subgradient which is used in the descent step in line 11. Line 12 updates the data structures for segments that will be used in future iterations. We prove later that this has an amortized $\widetilde{O}(1)$ cost.

Analysis of Algorithm 1. In this section we show the following theorem.

Theorem 6. *For a submodular function $f : \{0,1\}^n \to [-1,1]$, algorithm $\mathrm{SFM}(f,\varepsilon)$ returns a random point $x \in [0,1]^n$ with $\mathbb{E}[\hat{f}(x)] \le \min_T f(T) + \varepsilon$ and makes $\widetilde{O}(n/\varepsilon^2)$ oracle calls to f.*

Theorem 6 approximately minimizes the Lovasz extension. We then use Lemma 2.2 to find a discrete solution, directly implying Theorem 1.

Proof of Theorem 6. We first argue that the returned (random) point $x = \frac{1}{T+1} \sum_{i=0}^{T} x_i$ satisfies $\mathbb{E}[\hat{f}(x)] \le \min_T f(T) + \varepsilon$.

It suffices to show that each stochastic subgradient $\mathbf{g}^{\star i}$ satisfies $\mathbb{E}[\|\mathbf{g}^{\star i}\|_2^2] = \widetilde{O}(1)$. Then

$$\mathbb{E}[\|\mathbf{g}^i\|_2^2] = s \cdot \mathbb{E}[\|\mathbf{g}^{\star i}\|_2^2] = \widetilde{O}(1)$$

as $\mathbf{g}^{\star i}$ is s-sparse and $s = O(\log T) = \widetilde{O}(1)$. Then setting $T = \widetilde{O}(n/\varepsilon^2)$ suffices to apply Theorem 5 with $R^2 = n$ and $B^2 = \widetilde{O}(1)$.

To bound of $\mathbb{E}\left[\|\mathbf{g}^{\star i}\|_2^2\right]$, first define $\mathbf{z}_j = \mathrm{SAMPLE}(x_{k_{j+1}}, x_{k_j}, f)$ for $0 \le j \le b-1$ where $b = O(\log T)$, and $\mathbf{d}$ to be the estimate of g^0 as defined in line 7. Now, note that by Cauchy-Schwarz that

$$\mathbb{E}\left[\|\mathbf{g}^{\star i}\|_2^2\right] = \mathbb{E}\left[\left\|\mathbf{d} + \sum_{j=0}^{b-1} \mathbf{z}_j\right\|_2^2\right] \le (b+1) \cdot \mathbb{E}\left[\|\mathbf{d}\|_2^2 + \sum_{j=0}^{b-1} \|\mathbf{z}_j\|_2^2\right] = \widetilde{O}(1)$$

by the fact that $\mathbb{E}[\|\mathbf{z}_j\|_2^2] = O(1)$ by Lemma 3.4 and $\mathbb{E}[\|\mathbf{d}\|_2^2] \le \|g^0\|_1^2 = O(1)$ by Lemma 3.1.

Now, we argue that running algorithm $\mathrm{SFM}(f,\varepsilon)$ takes $\widetilde{O}(n/\varepsilon^2)$ oracle calls. First, we need $O(n)$ initial oracle calls to get the gradient g^0 of x_0. Now, we have to bound the total number of oracle calls from the calls to SAMPLE and PROCESS. To bound the former, we use that SAMPLE is called $\widetilde{O}(1)$ times per iteration and only requires $\widetilde{O}(1)$ oracles calls (Lemma 3.4). Note that $\mathrm{PROCESS}(x_{k_{j+1}}, x_{k_j}, f)$ has already been called before we call $\mathrm{SAMPLE}(x_{k_{j+1}}, x_{k_j}, f)$ as $k_{j+1} = k_j - 2^{\nu_2(k_j)}$ (see lines 6 and 12). Therefore, the total cost of all the calls to SAMPLE is $\widetilde{O}(T) = \widetilde{O}(n/\varepsilon^2)$ as desired.

Now we bound the total cost of oracle calls to PROCESS. Note that each $\mathbf{g}^i$ is 1-sparse, hence $x_i - x_{i-1}$ is 1-sparse. Therefore, for any $0 \le j \le i$, we have that $x_i - x_{i-j}$ is j-sparse. Thus, line 12 takes $O(2^{\nu_2(i)})$ oracle calls to f by Lemma 3.4. The total number of oracle calls is thus

$$O\left(\sum_{i=1}^{T} 2^{\nu_2(i)}\right) = \widetilde{O}(T) = \widetilde{O}(n/\varepsilon^2)$$

as desired. $\qquad\square$

4   SFM for Functions with Sparse Minimizer

In this section, we extend our above algorithm to the case where the minimizer of f is s-sparse and $f : \{0,1\}^n \to [-M, M]$ is an integer valued submodular function. In this setting we are able eliminate the linear oracle call dependence on n, the dimension of the space. A naïve application of the previous algorithm runs into the following barriers: computing the initial gradient requires $O(n)$ queries and the stochastic projected gradient descent requires $O(n/\varepsilon^2)$ iterations to converge. Previous work [CLSW17] resolves the latter issue by restricting the domain to $S^s \stackrel{\text{def}}{=} \{x \in [0,1]^n :$

$\sum_i x_i \leq s\}$ and arguing that the same algorithmic framework as described in Section 3 extends to this setting. Here, we focus on the problem of efficiently sampling a gradient of the initial point.

There are two barriers to removing the n dependence above: computing the starting subgradient and reducing the number of required iterations. To efficiently compute the starting gradient, we carefully choose a subgradient that is easier to compute. To do so, we note that at $x_0 = 0$, every permutation corresponds to a valid subgradient. Thus, it suffices to find any permutation P such that we can sample the subgradient g^P with $\widetilde{O}(1)$ oracle calls per sample after $\widetilde{O}(M^2)$ preprocessing.[4]

Efficiently sampling the initial subgradient.   Recall that for any permutation P, g^P is a subgradient at 0. In this section, we give a randomized algorithm which carefully chooses a permutation P and computes all nonzero coordinates of g^P in $\widetilde{O}(M^2)$ oracle calls.

This allows us to sample future estimates of g^P with variance $\widetilde{O}(M^2)$. In Appendix A we show that we can actually *derandomize* this part of the algorithm, deterministically finding a permutation P which we can compute all nonzero coordinates of g^P in $\widetilde{O}(\mathrm{poly}(M))$ oracle calls.

Now, we give a high level description for the algorithm. Consider any initial permutation P_0. Sample a subset $S \subseteq [n]$, where each element of $[n]$ is in S with probability $\frac{1}{10M}$. Also, let j be a coordinate such that say $g_j^{P_0} > 0$ (the $g_j^{P_0} < 0$ case is similar). Note that since g^{P_0} has integral entries, it must be $3M$-sparse. Therefore, there is at least a $\frac{1}{10M}\left(1 - \frac{1}{10M}\right)^{3M} \geq \frac{1}{20M}$ probability that $j \in S$ and for all other coordinates $j' \in S$, we have $g_{j'}^{P_0} = 0$. Condition on this event. Now, label the coordinates in S as $j_1, j_2, \cdots, j_{|S|}$, ordered as they were originally in P_0. Label the coordinates not in S as $i_1, i_2, \cdots, i_{n-|S|}$, also ordered as they were originally in P_0.

Consider the permutation $P' = \{j_1, j_2, \cdots, j_{|S|}, i_1, i_2, \cdots, i_{n-|S|}\}$. Note that because $g_t^{P_0} \geq 0$ for all $t \in S$ and $g_j^{P_0} > 0$, we have that, by submodularity, $g_{P'_t}^{P'} \geq 0$ for $1 \leq t \leq |S|$, and that there is a $1 \leq t \leq |S|$ with $g_{P'_t}^{P'} > 0$. Now, we can find such a coordinate in $\widetilde{O}(1)$ oracle calls with a binary search using Lemma 3.3. Once we find a coordinate t with $g_t^{P'} > 0$, we can move that coordinate to the left of the permutation, and continue the same algorithm on the remaining elements. We can find negative elements in a similar way by moving them to the right of the permutation. Note that submodularity ensures that moving the coordinate to the left or right of the permutation never makes it zero. Repeating this process $\widetilde{O}(M^2)$ times gives us our permutation P.

After defining some additional notation we will be ready to state the algorithm.

- For sequences of integers A, B we use $A \oplus B$ to denote concatenating A and B. This is useful to allow us to express concatenating subsequences of permutations. For example $\{1, 3\} \oplus \{2, 4\} = \{1, 3, 2, 4\}$.

- For a sequence P and a subsequence P' of P, the notation $P \backslash P'$ means to delete the elements from P' from P, while keeping the remaining elements in the same order as originally in P.

- **Sequences P_l and P_r.** After finding coordinates j where g_j^P is positive or negative, we move them to the left or right of the permutation respectively. We denote these "fixed coordinates" as P_l and P_r.

---

[4]This can be improved to $\widetilde{O}(M)$ oracle calls of preprocessing. We provide a sketch in Remark 4.2.

- **Subsequence S.** This is the subset of coordinates of P that we sample in an attempt to find a positive or negative coordinate.

Algorithm 2 FINDPERM(f). Takes a integer-valued submodular function f. Returns a pair (P', g') of a permutation P of $[n]$ and the associated subgradient $g' = g^{P'}$, encoded by all its $O(M)$ nonzero coordinates.

1: $P \leftarrow \{1, 2, \cdots, n\}$.   ▷ Arbitrary initialization.
2: $P_l, P_r \leftarrow \emptyset$.
3: **for** $t = 1$ to $\widetilde{O}(M^2)$ **do**
4:      S is a random subset of P, where each element $j \in P$ is in S with probability $\frac{1}{10M}$.
5:      $Q \leftarrow P_l \oplus S \oplus (P \backslash S) \oplus P_r$ ▷ Elements S and $P \backslash S$ are ordered as in P
6:      **if** $\sum_{j \in S} g_j^Q > 0$ **then**   ▷ Check for positive elements, using Lemma 3.3
7:          $x \leftarrow$ FINDINDEX($f, Q, S, 1$).
8:          $P_l \leftarrow P_l \oplus \{x\}$.
9:          $P \leftarrow P \backslash \{x\}$.
10:         Go back to line 3.
11:      $Q \leftarrow P_l \oplus (P \backslash S) \oplus S \oplus P_r$.   ▷ Elements S and $P \backslash S$ are ordered as in P
12:      **if** $\sum_{j \in S} g_j^Q < 0$ **then**   ▷ Check for negative elements, using Lemma 3.3
13:          $x \leftarrow$ FINDINDEX($f, Q, S, -1$).
14:          $P_r \leftarrow \{x\} \oplus P_r$.
15:          $P \leftarrow P \backslash \{x\}$.
16:         Go back to line 3.
17: **return** Permutation $P' = P_l \oplus P \oplus P_r$, with $g^{P'}$ encoded by the nonzero coordinates in P_l and P_r.

Algorithm 3 FINDINDEX(f, P, S, b). Takes a integer-valued submodular function f, permutation P of $[n]$, contiguous subset S of P, and integer b which is ± 1. Returns an index $j \in S$ such that $\text{sign}(g_j^P) = b$.

1: If $S = \{x\}$ (i.e. S contains a single element), **return** x.
2: Split S in half into subintervals S' and S''.
3: **if** $\text{sign}(\sum_{j \in S'} g_j^P) = b$ **then**   ▷ Uses $O(1)$ oracle calls by Lemma 3.3
4:      **return** FINDINDEX(f, P, S', b).
5: **else**
6:      **return** FINDINDEX(f, P, S'', b).

Lemma 4.1. *With high probability in n, FINDPERM(f) computes a permutation P' and all the nonzero coordinates of the associated gradient $g^{P'}$. It uses $\widetilde{O}(M^2)$ oracle calls to f.*

Proof. Consider some point during the execution of FINDPERM(f), and the sequences P_l, P_r, P at that time. Define $P' = P_l \oplus P \oplus P_r$. Our main claim is that if $g_j^{P'} \neq 0$ for some $j \in P$, then within $\widetilde{O}(M)$ iterations of the loop starting at line 3, one of line 6 or 12 will be true. To show this, let $j \in P$ be such that $g_j^{P'} \neq 0$, and without loss of generality, say $g_j^{P'} > 0$. Because $g^{P'}$ has at most

13

$3M$ nonzero coordinates, with probability at least

$$\left(1 - \frac{1}{10M}\right)^{3M} \cdot \frac{1}{10M} \geq \frac{1}{20M}$$

it will be true that $j \in S$ and for all $t \in S$ with $t \neq j$, that $g_t^{P'} = 0$. Then by submodularity, it is clear that if we define $Q = P_l \oplus S \oplus (P \backslash S) \oplus P_r$ that $\sum_{j \in S} g_j^Q > 0$. As this happens with probability at least $\frac{1}{20M}$, it will happen w.h.p. within $\widetilde{O}(M)$ iterations.

Now, we must argue that $\text{FINDINDEX}(f, P, S, b)$ indeed computes an index $j \in S$ such that $\text{sign}(g_j^P) = b$. We do the case $b = 1$ as the other is analogous. This amounts to checking that if $\text{sign}(\sum_{j \in S} g_j^P) = b$ and S' and S'' are subintervals of S whose union is S, then either $\text{sign}(\sum_{j \in S'} g_j^P) = b$ or $\text{sign}(\sum_{j \in S''} g_j^P) = b$ but this is trivial as

$$\sum_{j \in S} g_j^P = \sum_{j \in S'} g_j^P + \sum_{j \in S''} g_j^P.$$

To finish the proof, note that line 6 and 12 can only be true $O(M)$ times, as for any permutation P' we know that $g^{P'}$ has only $3M$ nonzero coordinates. Therefore, iterating $t = \widetilde{O}(M^2)$ is sufficient by our main claim shown in the first paragraph. As each iteration takes $\widetilde{O}(1)$ function calls in $\text{FINDINDEX}(f, P, S, b)$ by Lemma 3.3, the total number of function calls is also $\widetilde{O}(M^2)$ as desired. $\qquad\square$

Remark 4.2. Here we sketch how to change Algorithm 2 to improve the number of oracle calls in Lemma 4.1 to $\widetilde{O}(M)$. This doesn't affect our main result Theorem 2 because the number of oracle calls needed to perform the projected gradient descent dominates.

If g^P has exactly t nonzero coordinates, then we can show that choosing S as a random subset of P, where each element $j \in P$ is in S with probability p for $\frac{1}{20t} \leq p \leq \frac{1}{10t}$ (analogous to line 4 of Algorithm 2) will isolate some nonzero coordinate of g^P with at least constant probability. This is because each nonzero coordinate of g^P has at least a $p \cdot (1-p)^t \geq \frac{1}{100t}$ chance of being isolated. Unioning over all t nonzero coordinates (which correspond to disjoint events) shows that there is at least a $\frac{1}{100}$ probability of some nonzero coordinate being isolated. Therefore, running this $\widetilde{O}(1)$ times isolates some coordinate w.h.p.

As we do not know t, we iterate over guesses for t, i.e. set $p = 2^{-i}$ for for $0 \leq i \leq O(\log M)$ and run the process described in the above paragraph for each value of p.

Projecting onto S^s. The ℓ_2 projection onto S^s can be computed as follows. This was stated in [CLSW17].

Lemma 4.3. *For $s \geq 0$ let $S^s = \{x \in [0,1]^n : \sum_i x_i \leq s\}$. For any $y \in \mathbb{R}^n$, we have that the point $z = \text{proj}(y, S^s)$ is given by $z_i = median(0, 1, y_i - \lambda)$, where λ is the smallest nonnegative real number such that $\sum_i z_i \leq s$.*

Note that Lemma 4.3 shows that the permutation P consistent with y is also consistent with $\text{proj}(y, S^s)$.

Algorithm description and analysis. After finding a permutation P where we can efficiently sample g^P, we set $x_0 = 0$ (the origin), which is consistent with every permutation, and run a variation of Algorithm 1 where we project onto S^s. We need the following variation on Lemma 3.4 to deal with the projections. Lemma 4.4 was also argued in [CLSW17]. A proof sketch is provided in Appendix B.

Lemma 4.4. *Let $f : \{0,1\}^n \to [-M, M]$ be a submodular function with Lovasz extension $\hat{f}$. Let g denote the subgradients of $\hat{f}$. Let $x, y \in [0,1]^n$ be vectors and let P_x and P_y be permutations consistent with x, y respectively. Assume that we can transform P_x into P_y by deleting k elements from P_x and inserting them back in other locations. There is a data structure which after $O(k)$ calls to f of preprocessing supports the following: sample a 1-sparse random variable $\boldsymbol{z}$ with $\mathbb{E}[\boldsymbol{z}] = g(y) - g(x)$ and $\mathbb{E}[\|\boldsymbol{z}\|_2^2] = O(1)$ in $\widetilde{O}(1)$ calls to f. Preprocessing is called through $\textsc{Process}(x, y, f)$, and the sampling is called through $\textsc{Sample}(x, y, f)$.*

In other words, we don't necessarily need for $y - x$ to be k-sparse as in Lemma 3.4; it suffices for their consistent permutations to only "differ" by k moves. This essentially follows from the fact that $g(y)$ only depends on P_y.

At this point we are ready to state our algorithm.

Algorithm 4 $\textsc{SparseSFM}(f, s, \varepsilon)$. Takes a submodular function $f : \{0,1\}^n \to [-M, M]$ with an s-sparse minimzer and returns a random point $x \in [0,1]^n$ with $\mathbb{E}[\hat{f}(x)] < \min_T f(T) + 1$.

1: $T \leftarrow \widetilde{O}(sM^2)$.
2: $x_0 \leftarrow 0 \in \mathbb{R}^n$.
3: $(P_0, g^0) \leftarrow \textsc{FindPerm}(f)$.
4: **for** $i = 1$ to T **do**
5:     $b \leftarrow$ the number of 1 bits in the binary representation of $i - 1$.
6:     $k_0 \leftarrow i - 1$, $k_{j+1} \leftarrow k_j - 2^{\nu_2(k_j)}$ for $0 \le j \le b - 1$.   ▷ $\nu_2(y)$ is the maximum integer t such that 2^t divides y
7:     $\mathbf{d} \leftarrow \|g^0\|_1 \cdot \text{sign}(g_k^0) e_k$ with probability $|g_k^0|/\|g^0\|_1$.   ▷ Estimate of g^0
8:     $\mathbf{g}^{\star i} \leftarrow \mathbf{d} + \sum_{j=0}^{b-1} \textsc{Sample}(x_{k_{j+1}}, x_{k_j}, f)$.
9:     Let $c_1, c_2, \cdots, c_w$ be the nonzero coordinates of $\mathbf{g}^{\star i}$ ▷ $w \le b + 1$
10:     $\mathbf{g}^i \leftarrow w \cdot \mathbf{g}_{c_k}^{\star i} \cdot e_{c_k}$ with probability $\frac{1}{w}$.   ▷ $\mathbf{g}^i$ is 1-sparse
11:     $x_i \leftarrow \text{proj}(x_{i-1} - \eta \mathbf{g}^i, S^s)$.
12:     $\textsc{Process}(x_{i-2^{\nu_2(i)}}, x_i, f)$.
13: **return** $\frac{1}{T+1} \sum_{i=0}^{T} x_i$.

Theorem 7. *For submodular function $f : \{0,1\}^n \to [-M, M]$, algorithm $\textsc{SparseSFM}(f, s, \varepsilon)$ returns random point $x \in [0,1]^n$ with $\mathbb{E}[\hat{f}(x)] < \min_T f(T) + 1$ using $\widetilde{O}(sM^2)$ oracle calls to f.*

Proof sketch. Copy the proof of Theorem 6, replacing Lemma 3.4 with Lemma 4.4. $T = \widetilde{O}(sM^2)$ suffices as we can set $\varepsilon = \frac{1}{2}$, and $B^2 = \widetilde{O}(M^2)$, and $R^2 = s$ in Theorem 5. $\qquad\square$

It is direct to see that Theorem 7 implies Theorem 2.

5 SFM over Domain $[k]^n$

Previous work [Bac19] has considered more general domains for submodular functions, instead of the standard $\{0, 1\}^n$. Such a domain that the definition of submodularity can be extended to is functions $f : [k]^n \to \mathbb{R}$. We call a function $f : [k]^n \to \mathbb{R}$ submodular if for all $x, y \in [k]^n$ we have that

$$f(x) + f(y) \geq f(\max\{x, y\}) + f(\min\{x, y\})$$

where max and min are applied entry-wise. We assume without loss of generality that $f((1, \cdots, 1)) = 0$.

This definition can be further extended to functions over continuous domains. This has also been considered in previous work. We call a function $f : [0, 1]^n \to \mathbb{R}$ submodular if for all $i, j \in [n]$ with $i \neq j$ we have that $\frac{\partial^2 f}{\partial x_i \partial x_j} \leq 0$, i.e. all mixed partials are non-positive everywhere.

In this section we show how to obtain algorithms for minimizing submodular functions in each setting that make a number of oracle calls nearly linear in n.

5.1 Preliminaries

We start by providing the necessary definitions for this section.

General notation. Define $[k] \overset{\text{def}}{=} \{1, 2, \ldots, k\}$ and $[k]^n \overset{\text{def}}{=} \{(x_1, x_2, \cdots, x_n) : x_i \in [k] \text{ for all } i\}$.

Continuous extension. Here, we define the continuous extension for submodular functions $f : [k]^n \to \mathbb{R}$. All the results below were proven by Bach [Bac19]. We first define its domain.

Definition 5.1 (Domain of continuous extension). *Let $f : [k]^n \to \mathbb{R}$ be a submodular function. Define the set $H_k \overset{\text{def}}{=} \{x \in [0, 1]^{k-1} : x_1 \geq x_2 \geq \cdots \geq x_{k-1}\}$. The set $H_k^n \overset{\text{def}}{=} \overbrace{H_k \times \cdots \times H_k}^{n \text{ times}}$ will be the domain of the continuous extension of f.*

For a point $x = (x^1, x^2, \cdots, x^n) \in H_k^n$, we define $x_{a,b} \overset{\text{def}}{=} x_b^a$.

We define a permutation consistent to a point $x \in H_k^n$. This is a generalization of the situation for submodular functions over $\{0, 1\}^n$.

Definition 5.2 (Associated permutation to a submodular function). *An associated permutation to a point $x = (x^1, x^2, \cdots, x^n) \in H_k^n$, denoted (P, Q), is a permutation of $[n] \times [k-1]$, given by $(P_1, Q_1), (P_2, Q_2), \cdots, (P_{(k-1)n}, Q_{(k-1)n})$, which satisfies $x_{Q_i}^{P_i} \geq x_{Q_{i+1}}^{P_{i+1}}$ for $(k-1)n > i \geq 1$.*

We now define the continuous extension of a submodular function.

Definition 5.3 (Continuous extension of a submodular function). *Let $f : [k]^n \to \mathbb{R}$ be a submodular function. We define the continuous extension $\hat{f} : H_k^n \to \mathbb{R}$ of f as follows. For a point $x = (x^1, x^2, \cdots, x^n) \in H_k^n$, let (P, Q) be an associated permutation to x. Define the sequence of points $S_0, S_1, \cdots, S_{(k-1)n} \in [k]^n$ as $S_0 = (1, 1, \cdots, 1)$ and $S_i = S_{i-1} + e_{P_i}$ for $(k-1)n \geq i \geq 1$. Then we define*

$$\hat{f}(x) = x_{Q_{(k-1)n}}^{P_{(k-1)n}} f(S_{(k-1)n}) + \sum_{i=1}^{(k-1)n-1} (x_{Q_i}^{P_i} - x_{Q_{i+1}}^{P_{i+1}}) f(S_i)$$

It is direct to see that Definition 5.3 essentially reduces to the Lovasz extension in the case $k = 2$.

We now give an example illustrating Definition 5.3.

Example 1. Consider the following submodular function $f : [3]^2 \to \mathbb{R}$.

$$f(1,1) = 0, f(1,2) = 1, f(1,3) = 2$$
$$f(2,1) = 1, f(2,2) = 2, f(2,3) = 2$$
$$f(3,1) = 0, f(3,2) = 1, f(3,3) = 0.$$

Consider the following point in H_3^2: $x = (x^1, x^2) = ((0.6, 0.3), (0.5, 0.1))$. The permutation (P, Q) consistent with x is $\{(1,1), (2,1), (1,2), (2,2)\}$ as $x_1^1 \geq x_1^2 \geq x_2^1 \geq x_2^2$. This lets us compute that

$$S_0 = (1,1), S_1 = (2,1), S_2 = (2,2), S_3 = (3,2), S_4 = (3,3).$$

Therefore, we have that

$$\hat{f}(x) = 0.1 \cdot f(S_4) + 0.2 \cdot f(S_3) + 0.2 \cdot f(S_2) + 0.1 \cdot f(S_1)$$
$$= 0.1 \cdot f(3,3) + 0.2 \cdot f(3,2) + 0.2 \cdot f(2,2) + 0.1 \cdot f(2,1)$$
$$= 0.1 \cdot 0 + 0.2 \cdot 1 + 0.2 \cdot 2 + 0.1 \cdot 1 = 0.7.$$

The following properties of the continuous extension are known. See Sections 3, 4, 5 in [Bac19] for proofs.

Theorem 8 (Properties of the continuous extension). *Let $f : [k]^n \to \mathbb{R}$ be a submodular function, and let $\hat{f}$ be its continuous extension. Then we have that*

- *$\hat{f}$ is convex.*

- *For $S = (s_1, s_2, \cdots, s_n) \in [k]^n$, if we define $x^i \in H_k$ as $x^i = \sum_{j=1}^{s_i - 1} e_j$, then for $x = (x^1, x^2, \cdots, x^n) \in H_k^n$ we have that $\hat{f}(x) = f(S)$.*

- *We have that $\min_{x \in H_k^n} \hat{f}(x) = \min_{S \in [k]^n} f(S)$.*

Additionally, the vector $g(x) \in \mathbb{R}^{n \times (k-1)}$ defined by $g(x)_{P_j, Q_j} \stackrel{\text{def}}{=} f(S_j) - f(S_{j-1})$ is a subgradient of the continuous extension, where the S_j are defined as in Definition 5.3.

5.2 SFM over $[k]^n$

In this section we sketch an algorithm and analysis for submodular function minimization of functions $f : [k]^n \to [-1, 1]$. Precisely we show the following result.

Theorem 9. *Given a submodular function $f : [k]^n \to [-1, 1]$ and an $\varepsilon > 0$, we can compute a random point $x \in [k]^n$ with*

$$\mathbb{E}[f(x)] \leq \min_{y \in [k]^n} f(y) + \varepsilon$$

in $\widetilde{O}(nk^4/\varepsilon^2)$ calls to an oracle for f.

As the algorithm is extremely similar to those presented in Section 3 we simply state the analogues of the lemmas we must show and how they imply the result. Precisely we need the analogues of Lemma 3.1, Lemma 3.2, and Lemma 3.3 for the continuous extension which was defined in Section 2. The proofs are analogous to those of Lemma 3.1, Lemma 3.2, and Lemma 3.3 which were given in [CLSW17].

Before stating the lemmas, we remark that the setup and notation we are using is as in Definition 5.3.

Lemma 5.4. *For a submodular function $f : [k]^n \to [-M, M]$, all subgradients g of the continuous extension satisfy $\|g(x)\|_1 \leq 4M(k-1)$.*

Lemma 5.5. *Let $x = (x^1, \cdots, x^n), y = (y^1, \cdots, y^n) \in H_k^n$, and $d = (d^1, \cdots, d^n) \in \mathbb{R}_{\geq 0}^{n \times (k-1)}$ be such that $y = x + d$ (respectively $y = x - d$). For all i such that $d^i = 0$ and $j \in [k-1]$ we have that $g(x)_{i,j} \geq g(y)_{i,j}$ (respectively $g(x)_{i,j} \leq g(y)_{i,j}$).*

Lemma 5.6. *Let $x \in H_k^n$ and let (P, Q) be the permutation consistent with x. Then we have for any integers $1 \leq a \leq b \leq n(k-1)$ that*

$$\sum_{i=a}^{b} g(x)_{P_i, Q_i} = f(S_b) - f(S_{a-1}),$$

where the S_j are defined as in Definition 5.3.

We prove these in Appendix B.

Additionally, as the algorithm we intend to use is projected subgradient descent, we must be able to project onto H_k^n. Projecting onto H_k is simply an isotonic regression, which can be done via the pool-adjacent-violators algorithm [BC90].

Finally, we need the analogue of Lemma 3.4. The proof is analogous to that of Lemma 3.4 and we provide a sketch in Appendix B.

Lemma 5.7. *Let $f : [k]^n \to [-1, 1]$ be a submodular function with continuous extension $\hat{f}$. Let g denote the subgradients of $\hat{f}$. Let $x = (x^1, \cdots, x^n), y = (y^1, \cdots, y^n) \in H_k^n$ be vectors. Let $d = (d^1, \cdots, d^n) \in \mathbb{R}^{n \times (k-1)}$ be the vector such that $d = y - x$, and say that there are ℓ indices $i \in [n]$ such that $d^i \neq 0$. There is a data structure which after $O(\ell k)$ calls to f of preprocessing, supports the following: sample a 1-sparse random variable $\mathbf{z}$ with $\mathbb{E}[\mathbf{z}] = g(y) - g(x)$ and $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(k^2)$ in $\widetilde{O}(1)$ calls to f. Preprocessing is called through $\text{PROCESS}(x, y, f)$, and the sampling is called through $\text{SAMPLE}(x, y, f)$.*

The condition $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(k^2)$ comes from the fact that our algorithm will satisfy $\mathbb{E}[\|\mathbf{z}\|_2^2] \leq O\left(\max_x \|g(x)\|_1^2\right) \leq O(k^2)$ by Lemma 5.4.

Proof of Theorem 9. We use basically the exactly same algorithm as Algorithm 1, except with the projections in line 11 replaced with projections onto H_k^n. We can compute that (in the language of Theorem 5) we have that $R^2 = nk$ and $B^2 = O(k^2)$, hence we have expected error ε in $O(R^2 B^2 / \varepsilon^2) = O(nk^3 / \varepsilon^2)$ steps. By Lemma 5.7, the total number of calls in the procedure corresponding to line 12 of Algorithm 1 will take on average $\widetilde{O}(k)$ times the iteration count. Therefore, the total number of function calls to f is $\widetilde{O}(k \cdot nk^3 / \varepsilon^2) = \widetilde{O}(nk^4 / \varepsilon^2)$ as desired. □

5.3 SFM for continuous functions

In this section we prove Theorem 3. Our setup is the following: we have a submodular function $f : [0,1]^n \to \mathbb{R}$, and we wish to approximately minimize f. Our algorithms are in terms of the L^∞-Lipschitz constant of f, which we denote as L.

Our algorithm is simple: we essentially just discretize f and use Theorem 9. Specifically, define $k = \frac{2L}{\varepsilon}$, and define the function $f' : [k]^n \to \mathbb{R}$ as $f'(x) = f(x/k)$ for $x \in [k]^n \subseteq \mathbb{R}^n$. Note that it is clear that

$$\min_{x\in[k]^n} f'(x) \leq \min_{x\in[0,1]^n} f(x) + \frac{L}{k} \leq \min_{x\in[0,1]^n} f(x) + \varepsilon/2$$

by our choice of k. We can also verify that f' is submodular. Therefore, it suffices to minimize f' within $\varepsilon/2$. Without loss of generality, we assume $f'(1,1,\cdots,1) = 0$.

We can almost directly apply Theorem 9, except that the range of f' is not $[-1,1]$, and is instead $[-L,L]$. This change multiplies the B^2 term in our application of Theorem 5 by L^2 (so that $B^2 = O(k^2L^2)$), giving a total complexity of $\widetilde{O}(nk^4L^2/\varepsilon^2) = \widetilde{O}(nL^6/\varepsilon^6)$ oracle calls to f as desired.

We now formally give the proof. It is essentially as described above.

Proof of Theorem 3. Define $k = \frac{2L}{\varepsilon}$, and define the function $f' : [k]^n \to \mathbb{R}$ as $f'(x) = f(x/k)$ for $x \in [k]^n \subseteq \mathbb{R}^n$. We have that

$$\min_{x\in[k]^n} f'(x) \leq \min_{x\in[0,1]^n} f(x) + \frac{L}{k} \leq \min_{x\in[0,1]^n} f(x) + \varepsilon/2.$$

Therefore, it suffices to minimize f' to additive $\varepsilon/2$.

To do this, we use the same algorithm as in the proof of Theorem 9. As in the notation of Theorem 5, we have that $R^2 = nk$, and $B^2 = O(k^2L^2)$, where the extra factor of L^2 comes from the fact that the range of f' is $O(L)$. Thus, the expected error is $\varepsilon/2$ in $O(R^2B^2/\varepsilon^2) = O(nk^3L^2/\varepsilon^2)$ iterations. By the same argument as in the proof of Theorem 9, we have that on average, each iteration requires $\widetilde{O}(k)$ function calls. The total number of calls is therefore $\widetilde{O}(nk^4L^2/\varepsilon^2) = \widetilde{O}(nL^6/\varepsilon^6)$ function calls by our choice of k. $\square$

Acknowledgements

References

[Bac13]   Francis R. Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends in Machine Learning*, 6(2-3):145–373, 2013.

[Bac19]   Francis Bach. Submodular functions: from discrete to continuous domains. *Math. Program.*, 175(1-2, Ser. A):419–459, 2019.

[BC90]   Michael J. Best and Nilotpal Chakravarti. Active set algorithms for isotonic regression; A unifying framework. *Math. Program.*, 47:425–439, 1990.

[BGM⁺17] Ernesto G. Birgin, J. L. Gardenghi, José Mario Martínez, Sandra Augusta Santos, and Philippe L. Toint. Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. *Math. Program.*, 163(1-2):359–368, 2017.

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

[BVZ99] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 377–384. IEEE, 1999.

[CDHS16] Yair Carmon, John C. Duchi, Oliver Hinder, and Aaron Sidford. Accelerated methods for non-convex optimization. *CoRR*, abs/1611.00756, 2016.

[Cho54] Gustave Choquet. Theory of capacities. *Ann. Inst. Fourier, Grenoble*, 5:131–295 (1955), 1953–1954.

[CJK14] Deeparnab Chakrabarty, Prateek Jain, and Pravesh Kothari. Provable submodular minimization using wolfe's algorithm. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 802–809, 2014.

[CLSW17] Deeparnab Chakrabarty, Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. Subquadratic submodular function minimization. In *STOC'17—Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1220–1231. ACM, New York, 2017.

[Cun85] William H. Cunningham. On submodular function minimization. *Combinatorica*, 5(3):185–192, 1985.

[CVZ14] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM Journal on Computing*, 43(6):1831–1879, 2014.

[Edm70] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)*, pages 69–87. Gordon and Breach, New York, 1970.

[FMV11] Uriel Feige, Vahab S Mirrokni, and Jan Vondrak. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[Fuj80] Satoru Fujishige. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Math. Oper. Res.*, 5(2):186–196, 1980.

[Fuj05] Satoru Fujishige. *Submodular functions and optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier B. V., Amsterdam, second edition, 2005.

[GG17] Sergey Guminov and Alexander Gasnikov. Accelerated methods for α-weakly-quasi-convex problems. *arXiv preprint arXiv:1710.00797*, 2017.

[GLM16] Rong Ge, Jason D. Lee, and Tengyu Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2973–2981, 2016.

[GLS84] M. Grötschel, L. Lovász, and A. Schrijver. Corrigendum to our paper: "The ellipsoid method and its consequences in combinatorial optimization" [Combinatorica **1** (1981), no. 2, 169–197; MR0625550 (84a:90044)]. *Combinatorica*, 4(4):291–295, 1984.

[HK12] Elad Hazan and Satyen Kale. Online submodular minimization. *Journal of Machine Learning Research*, 13:2903–2922, 2012.

[HRRS19] Yassine Hamoudi, Patrick Rebentrost, Ansis Rosmanis, and Miklos Santha. Quantum and classical algorithms for approximate submodular function minimization. *CoRR*, abs/1907.05378, 2019.

[HSS19] Oliver Hinder, Aaron Sidford, and Nimit Sharad Sohoni. Near-optimal methods for minimizing star-convex functions and beyond. *arXiv preprint arXiv:1906.11985*, 2019.

[IFF00] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 97–106. ACM, New York, 2000.

[IO09] Satoru Iwata and James B. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237. SIAM, Philadelphia, PA, 2009.

[JB11] Stefanie Jegelka and Jeff A. Bilmes. Online submodular minimization for combinatorial structures. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 345–352, 2011.

[KG11] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM TIST*, 2(4):32:1–32:20, 2011.

[KKT08] Pushmeet Kohli, M Pawan Kumar, and Philip HS Torr. P^3 & beyond: Move making algorithms for solving higher order functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1645–1656, 2008.

[KT10] Pushmeet Kohli and Philip HS Torr. Dynamic graph cuts and their applications in computer vision. In *Computer Vision*, pages 51–108. Springer, 2010.

[LB10] Hui Lin and Jeff Bilmes. An application of the submodular principal partition to training data subset selection. In *NIPS workshop on Discrete Optimization in Machine Learning*, 2010.

[LB11a] Hui Lin and Jeff Bilmes. Optimal selection of limited vocabulary speech corpora. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[LB11b] Hui Lin and Jeff A. Bilmes. A class of submodular functions for document summarization. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 510–520, 2011.

[LJ15] Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of frank-wolfe optimization variants. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 496–504, 2015.

[Lov83] L. Lovász. Submodular functions and convexity. In *Mathematical programming: the state of the art (Bonn, 1982)*, pages 235–257. Springer, Berlin, 1983.

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science—FOCS 2015*, pages 1049–1065. IEEE Computer Soc., Los Alamitos, CA, 2015.

[McC05] S Thomas McCormick. Submodular function minimization. *Handbooks in operations research and management science*, 12:321–391, 2005.

[Nes12] Yurii Nesterov. How to make the gradients small. *Optima*, 88:10–11, 2012.

[NGGD18] Yurii Nesterov, Alexander Gasnikov, Sergey Guminov, and Pavel Dvurechensky. Primal-dual accelerated gradient descent with line search for convex and nonconvex optimization problems. *arXiv preprint arXiv:1809.05895*, 2018.

[NP06]   Yurii Nesterov and Boris T. Polyak. Cubic regularization of newton method and its global performance. *Math. Program.*, 108(1):177–205, 2006.

[NY83]   Arkadii Semenovich Nemirovsky and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.

[Sch00]   Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combin. Theory Ser. B*, 80(2):346–355, 2000.

[Wol76]   Philip Wolfe. Finding the nearest point in a polytope. *Math. Programming*, 11(2):128–149, 1976.

A   Nonconstructive derandomization of Algorithm 4

In this section we explain how we can nonconstructively derandomize Algorithm 4. In other words, we sketch a deterministic algorithm such that given an integer-valued submodular function $f : \{0,1\}^n \to [-M, M]$ finds a permutation P such that we can compute all nonzero coordinates of g^P in $\widetilde{O}(\text{poly}(M))$ oracle calls.

Now we explain the main idea behind the derandomization. Let's consider the case in Algorithm 4 when $P_l = P_r = \emptyset$ at the start. Let S be the random subset generated in line 4, where each element in P is in S with probability $\frac{1}{10M}$. As explained in the proof of Lemma 4.1, this choice of S allows to make progress as long as there is exactly one index $i \in S$ such that $g_i^P \neq 0$, and for all other indices $j \in S$ with $j \neq i$ we have that $g_j^P = 0$. As long as $g^P \neq 0$, the probability of this occuring is at least

$$\frac{1}{10M} \cdot \left(1 - \frac{1}{10M}\right)^{3M} \geq \frac{1}{20M}.$$

Imagine randomly generating $T = 500M^2 \log n$ such sets $S_1, \cdots, S_T$. The probability that there is no set S_j satisfying the desired property of exactly one index $i \in S_j$ with $g_i^P \neq 0$ is at most $\left(1 - \frac{1}{20M}\right)^{500M^2 \log n} \leq n^{-20M}$. Note that because $\|g^P\|_1 \leq 3M$ by Lemma 3.1, there are at most $(2n)^{3M}$ distinct possible subgradients g^P. Because

$$(2n)^{3M} \cdot n^{-20M} < 1,$$

a union bound tells us that there exist *deterministic* sets $S_1, \cdots, S_T$ that the algorithm can precompute independent of f such that for any nonzero subgradient g^P there is a set S_j for $1 \leq j \leq T$ such that for exactly index $i \in S_j$ we have $g_i^P \neq 0$, as desired.

We now formally state this discussion as a lemma.

Lemma A.1. *There is a deterministic algorithm which give an integer-valued submodular function $f : \{0,1\}^n \to [-M, M]$ computes a permutation P and finds all nonzero entries of g^P in $\widetilde{O}(M^3)$ oracle calls.*

Proof. This proof essentially follows the above discussion. Define $T = 500M^2 \log n$. Our goal is to deterministically construct sets $S_1, S_2, \ldots, S_T \subseteq [n]$ such that for any nonzero vector $g \in \mathbb{R}^n$ with integer entries and $\|g\|_1 \leq 3M$, that for some $1 \leq j \leq T$, we have that there is exactly one element $i \in S_j$ such that $g_i \neq 0$. With this construction, we can simply copy the proof of Lemma 4.1, using that all subgradients g of the Lovasz extension have integer entries and ℓ_1 norm at most $3M$. We focus on this goal in the remainder of the proof.

Randomly generate subsets $S_1, \cdots, S_T \subseteq [n]$ as follows: each S_j is such that each $i \in P$ is independently in S_j with probaiblity $\frac{1}{10M}$. Let $g \in \mathbb{R}^n$ be a nonzero vector with integer entries and

$\|g\|_1 \leq 3M$. We now bound the probability for some S_j we have that there is exactly one element $i \in S_j$ with $g_i \neq 0$. For a fixed j, the probability that S_j satisfies this property is at least

$$\frac{1}{10M} \cdot \left(1 - \frac{1}{10M}\right)^{3M} \geq \frac{1}{20M}.$$

Therefore, the probability that no S_j satisfy the desired property is at most

$$\left(1 - \frac{1}{20M}\right)^T \leq n^{-20M}.$$

Our next goal is to count the number of possible distinct vectors $g \in \mathbb{R}^n$ with integer entries and $\|g\|_1 \leq 3M$. A direct counting argument easily shows that the number of such vectors g is at most $\sum_{k=0}^{3M} (2n)^k \leq 2(2n)^{3M}$. Therefore, by a union bound (as $2(2n)^{3M} \cdot n^{-20M} < 1$) there exists sets $S_1, \ldots, S_T \subseteq [n]$ such that for all vectors $g \in \mathbb{R}^n$ with integer entries and $\|g\|_1 \leq 3M$ that there is a j such that there is exactly one element $i \in S_j$ with $g_i \neq 0$. We can find such sets $S_1, \ldots, S_T$ just by brute forcing over all possibilities: these are independent of f, so they do not cost oracle calls to compute. $\qquad\square$

It would be interesting to give a polynomial time algorithm to deterministically construct sets $S_1, S_2, \ldots, S_T$ as described in Lemma A.1.

Remark A.2. We can improve the number of oracle calls in Lemma A.1 to $\widetilde{O}(M^2)$ and the value of T in the proof to $\widetilde{O}(M)$ using the same technique as in Remark 4.2.

B Additional Proofs

Lemma 3.4. *Let $f : \{0,1\}^n \to [-1,1]$ be a submodular function with Lovasz extension $\hat{f}$. Let g denote the subgradients of $\hat{f}$. Let $x, y \in [0,1]^n$ be vectors such that $y - x$ is k-sparse. There is a data structure which after $O(k)$ calls to f of preprocessing, supports the following: sample a 1-sparse random variable $\mathbf{z}$ with $\mathbb{E}[\mathbf{z}] = g(y) - g(x)$ and $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(1)$ in $\widetilde{O}(1)$ calls to f. Preprocessing is called through $\text{PROCESS}(x, y, f)$, and the sampling is called through $\text{SAMPLE}(x, y, f)$.*

Proof. Let $d = y - x$. We first argue that it suffices to consider the case where d either has all nonnegative or all nonpositive coordinates. To this end, let $d^+, d^- \in \mathbb{R}^n$ be the positive and negative parts of d, precisely defined as

$$d_i^+ = \max(0, d_i) \text{ and } d_i^- = \min(0, d_i) \text{ for all } 1 \leq i \leq n.$$

Write

$$g(y) - g(x) = \big(g(x + d^+ + d^-) - g(x + d^+)\big) + \big(g(x + d^+) - g(x)\big).$$

To sample the estimate $\mathbf{z}$ for $g(y) - g(x)$, we instead sample $\mathbf{z}_1$ for $(g(x + d^+ + d^-) - g(x + d^+))$ and $\mathbf{z}_2$ for $(g(x + d^+) - g(x))$, and set $\mathbf{z}$ to be either $2\mathbf{z}_1$ or $2\mathbf{z}_2$, each with probability $\frac{1}{2}$. It is clear that if both $\mathbb{E}[\|\mathbf{z}_1\|_2^2] = O(1)$ and $\mathbb{E}[\|\mathbf{z}_2\|_2^2] = O(1)$, then $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(1)$. This shows that we can reduce to the case where either d has all nonnegative or nonpositive coordinates.

By symmetry, we only consider the case where d has all nonnegative coordinates. Let $y = x + d$. Let P_x be the permutation consistent with x, and let P_y be the permutation consistent with y. Note

that because d is k-sparse, one can transform permutation P_x into P_y deleting k elements from P_x and inserting them back. Therefore, there exist subsets $I_1, I_2, \cdots, I_{2k} \subset [n]$ that are intervals in both P_x and P_y.

The phase $\mathrm{PROCESS}(x, y, f)$ then proceeds computing $D_t \stackrel{\text{def}}{=} \sum_{j \in I_t}(g(y)_j - g(x)_j)$ for all $1 \leq t \leq 2k$. By Lemma 3.3 this requires $O(k)$ queries to f. Note that each for each $j \in I_t$, the terms $g(y)_j - g(x)_j$ are all the same sign by Lemma 3.2, hence $\sum_{t=1}^{2k}|D_t| = |g(y) - g(x)|_1$.

$\mathrm{SAMPLE}(x, y, f)$ proceeds as follows. Choose an interval I_t proportional to $|D_t|$. Let I be the interval that is chosen. Split this interval in half into two intervals I' and I''. Compute the sums $D' = \sum_{j \in I'}(g(y)_j - g(x)_j)$ and $D'' = \sum_{j \in I''}(g(y)_j - g(x)_j)$. Sample one of I' and I'' proportional to D' and D'', respectively. Now continue recursively. When the interval is size 1, say containing the element j, return the vector $\mathbf{z} = \|g(y) - g(x)\|_1 \cdot \mathrm{sign}(g(y)_j - g(x)_j) \cdot e_j$. By Lemma 3.3 this phase takes $\widetilde{O}(1)$ queries to f and returns a 1-sparse estimate $\mathbf{z}$ for $g(y) - g(x)$. We can check by the construction that $\mathbb{E}[\mathbf{z}] = g(y) - g(x)$, and

$$\mathbb{E}[\|\mathbf{z}\|_2^2] \leq \|g(y) - g(x)\|_1^2 = O(1)$$

by Lemma 3.1. $\qquad\square$

Lemma 4.4. *Let $f : \{0,1\}^n \to [-M, M]$ be a submodular function with Lovasz extension $\hat{f}$. Let g denote the subgradients of $\hat{f}$. Let $x, y \in [0,1]^n$ be vectors and let P_x and P_y be permutations consistent with x, y respectively. Assume that we can transform P_x into P_y by deleting k elements from P_x and inserting them back in other locations. There is a data structure which after $O(k)$ calls to f of preprocessing supports the following: sample a 1-sparse random variable $\mathbf{z}$ with $\mathbb{E}[\mathbf{z}] = g(y) - g(x)$ and $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(1)$ in $\widetilde{O}(1)$ calls to f. Preprocessing is called through $\mathrm{PROCESS}(x, y, f)$, and the sampling is called through $\mathrm{SAMPLE}(x, y, f)$.*

Proof sketch. It is direct to see that if we can get from P_x to P_y by deleting k elements from P_x and inserting them back in other positions, then there exist points $x', y' \in [0,1]^n$ where all coordinates of x' are distinct and all coordiantes of y' are distinct, such that P_x is consistent with x', P_y is consistent with y', and $y' - x'$ is k-sparse. Now use the proof of Lemma 3.2 above on the points x' and y'. $\qquad\square$

Lemma 5.4. *For a submodular function $f : [k]^n \to [-M, M]$, all subgradients g of the continuous extension satisfy $\|g(x)\|_1 \leq 4M(k-1)$.*

Proof. Let g be the gradient at a point x. We prove that for $1 \leq j < k$ we have that $\sum_{i=1}^n |g_{i,j}| \leq 4M$. Summing over all j then gives us that $\|g\|_1 \leq 4M(k-1)$. We first bound the sum of the positive entries of g, i.e. we show that $\sum_{i=1}^n \max(0, g(i,j)) \leq 2M$ for any $j \in [k-1]$. An analogous argument will show that $\sum_{i=1}^n \min(0, g(i,j)) \geq -2M$, which together is sufficient.

Fix $j \in [k-1]$. Let (P, Q) be the permutation corresponding to our point x. Without loss of generality, assume that $g_{1,j} \geq g_{2,j} \geq \cdots \geq g_{n,j}$. Let $a_1, a_2, \ldots, a_n$ be such that $(P_{a_y}, Q_{a_y}) = (y, j)$. Let the S_i be defined as in Definition 5.3. Note that $a_1 \leq \cdots \leq a_n$ by our assumption. Let $X = \{i \in [n] : g_{i,j} > 0\}$, let $t = |X|$ and let $i_1 \leq \cdots \leq i_t$ be the elements of X. Define $v_0 = (j-1, j-1, \cdots, j-1) \in \mathbb{R}^n$. For $1 \leq y \leq t$, define $v_y = v_{y-1} + e_{i_y}$. Note that by the definition of submodularity over $[k]^n$ that $f(v_y) - f(v_{y-1}) \geq f(S_{a_{i_y}}) - f(S_{a_{i_y}-1}) = g_{i_y,j}$. Therefore, we have that

$$2M \geq f(v_t) - f(v_0) = \sum_{y=1}^t f(v_y) - f(v_{y-1}) \geq \sum_{y=1}^t g_{i_y,j}$$

as desired. ∎

Lemma 5.5. *Let* $x = (x^1, \cdots, x^n), y = (y^1, \cdots, y^n) \in H_k^n$, *and* $d = (d^1, \cdots, d^n) \in \mathbb{R}_{\geq 0}^{n \times (k-1)}$ *be such that* $y = x + d$ *(respectively* $y = x - d$*). For all* i *such that* $d^i = 0$ *and* $j \in [k-1]$ *we have that* $g(x)_{i,j} \geq g(y)_{i,j}$ *(respectively* $g(x)_{i,j} \leq g(y)_{i,j}$*).*

Proof. We only prove the case $y = x + d$, as the $y = x - d$ case is analogous. Let (P, Q) be the permutation corresponding to x and (P', Q') is the permutation corresponding to y. Let S_i be the sets defined in Definition 5.3 for x, and let S_i' be the sets for y.

Let a, b be such that $(i, j) = (P_a, Q_a)$ and $(i, j) = (P_b', Q_b')$. Then Theorem 8 tell us that $g(x)_{i,j} = f(S_a) - f(S_{a-1})$ and $g(y)_{i,j} = f(S_b') - f(S_{b-1}')$, where $S_a = S_{a-1} + e_i$ and $S_b' = S_{b-1}' + e_i$ as defined in Definition 5.3. We will show that $(S_{a-1})_t \leq (S_{b-1}')_t$ for all indices $t \in [n]$, and $(S_{a-1})_i = (S_{b-1}')_i$. Then the inequality

$$f(S_a) - f(S_{a-1}) = f(S_{a-1} + e_i) - f(S_{a-1}) \geq f(S_{b-1}' + e_i) - f(S_{b-1}') = f(S_b') - f(S_{b-1}')$$

by the definition of submodularity over $[k]^n$.

To argue that $S_{b-1}' \geq S_{a-1}$ coordinate-wise, note that because $y = x + d$ and $d \geq 0$ that we can get from (P, Q) to (P', Q') by moving some pairs (P_t, Q_t) to the left (where the "left" has the larger elements) but without touching any (P_t, Q_t) with $P_t = i$ as $d^i = 0$. By the definition of the S_i in Definition 5.3, we can now directly check that $S_{b-1}' \geq S_{a-1}$ entry-wise, and $(S_{a-1})_i = (S_{b-1}')_i$ as desired. ∎

Lemma 5.6. *Let* $x \in H_k^n$ *and let* (P, Q) *be the permutation consistent with* x. *Then we have for any integers* $1 \leq a \leq b \leq n(k-1)$ *that*

$$\sum_{i=a}^{b} g(x)_{P_i, Q_i} = f(S_b) - f(S_{a-1}),$$

where the S_j *are defined as in Definition 5.3.*

Proof. This follows immediately from the definition of g. By Theorem 8 we have that $g(x)_{P_i, Q_i} = f(S_i) - f(S_{i-1})$. Therefore,

$$\sum_{i=a}^{b} g(x)_{P_i, Q_i} = \sum_{i=a}^{b} f(S_i) - f(S_{i-1}) = f(S_b) - f(S_{a-1}).$$

∎

Lemma 5.7. *Let* $f : [k]^n \to [-1, 1]$ *be a submodular function with continuous extension* $\hat{f}$. *Let* g *denote the subgradients of* $\hat{f}$. *Let* $x = (x^1, \cdots, x^n), y = (y^1, \cdots, y^n) \in H_k^n$ *be vectors. Let* $d = (d^1, \cdots, d^n) \in \mathbb{R}^{n \times (k-1)}$ *be the vector such that* $d = y - x$, *and say that there are* ℓ *indices* $i \in [n]$ *such that* $d^i \neq 0$. *There is a data structure which after* $O(\ell k)$ *calls to* f *of preprocessing, supports the following: sample a 1-sparse random variable* $\mathbf{z}$ *with* $\mathbb{E}[\mathbf{z}] = g(y) - g(x)$ *and* $\mathbb{E}[\|\mathbf{z}\|_2^2] = O(k^2)$ *in* $\widetilde{O}(1)$ *calls to* f. *Preprocessing is called through* PROCESS(x, y, f), *and the sampling is called through* SAMPLE(x, y, f).

Proof sketch. Using the same technique as in Lemma 3.4 we reduce to the case where d has all non-negative coordinates, so that $y = x + d$. Because there are ℓ indices $i \in [n]$ such that $d^i \neq 0$, we can transform the associated permutation (P, Q) of x to the associated permutation (P', Q') of y by deleting and reinserting $k\ell$ elements, corresponding to k coordinates per each i with $d^i \neq 0$, and there are at most ℓ such indices i.

By submodularity, we can construct $O(k\ell)$ intervals where $g(y) - g(x)$ is either all positive or all negative. We can preprocess these intervals in $O(k\ell)$ oracle calls as done in Lemma 3.4. Afterwards, we can sample 1-sparse estimates to $g(y) - g(x)$ in $\widetilde{O}(1)$ queries. As in Lemma 3.4 our estimate will satisfy $\mathbb{E}[\|\mathbf{z}\|_2^2] = \|g(y) - g(x)\|_1^2 = O(k^2)$ by Lemma 5.4. $\qquad\square$