Smoothed Analysis of Algorithms and the Fujishige-Wolfe Algorithm to Minimize Submodular Functions

Pravesh Kothari, Rashish Tandon
(kothari, rashish@cs.utexas.edu)

December 5, 2011

Abstract

We study the method of smoothed analysis for understanding the complexity of algorithms. We first discuss the need for defining an alternative to worst case complexity and discuss some alternatives considered in analysis of algorithms. We then discuss smoothed analysis, introduced by Spielman and Teng [22] and describe how one could do a smoothed analysis of an algorithm. There are a few design questions here which we attempt to answer. As an instance we study the time complexity of the Simplex method with Two-Phase Shadow Vertex Pivot rule which was shown to have a polynomial smoothed complexity by Spielman and Teng [22].

We intend to apply smoothed analysis to analyze the time complexity of of a well known algorithm in combinatorial optimization for submodular function minimization, the Fujishige-Wolfe min-norm algorithm [9] . The time complexity has been unknown for a long time, although it has been shown to perform extremely well in practice. We study the Fujishige-Wolfe method to minimize a submodular function and prove properties of the algorithm that make the details clear. We then consider an approach to do a smoothed complexity analysis of this algorithm.

1 Introduction

The standard approach to proving performance guarantees on algorithms is to do a worst case analysis. Although this approach looks pessimistic, it has proven to be highly successful in quantifying the performance of several algorithms. However, there are several natural algorithms in scientific computing, combinatorial optimization etc. which have been used and perfected over the years and perform very well in practice. However, the worst case bound on their performance is known to be pessimistically large (even exponential in several cases). Such examples could be explained by saying that the input for these algorithms is obtained in a way that skips the bad instances. However, this doesn't give a quantitatively satisfying explanation of this good performance.
To make things a little more formal: for an algorithm $\mathcal{A}$ let $\mathcal{X}$ be the domain of the input and $\mathcal{T}_{\mathcal{A}}$ be the measure of time complexity of $\mathcal{A}$ on input $x \in \mathcal{X}$. Partition $\mathcal{X}$ into $\cup_{n\geq 1}\mathcal{X}_n$ where $\mathcal{X}_n$ is the subset of inputs of size n.

Definition 1.1 (Worst Case Complexity)**.** *For inputs drawn from $\mathcal{X} = \cup_{n>0}\mathcal{X}_n$, the worst case complexity of algorithm $\mathcal{A}$, $C_{worst}(n)$ given by*

$$\mathcal{C}_{worst}(n) = max_{x\in\mathcal{X}_n}\mathcal{T}_{\mathcal{A}}(x)$$

An example of such an algorithm is the simplex algorithm for linear programming which has been used in practice from early 1940s. Consider a linear programming instance in d dimensions with m constraints. The simplex algorithm is, in fact, a meta heuristic and becomes a method only with the introduction of a *pivot rule*. For almost every pivot rule, however, researchers were able to obtain a contrived family of input instances where the algorithm ran in exponential time ([14], [11]). The best present analysis time for a randomized pivot tule takes an expected time $m^{O(\sqrt{d})}$ which is clearly not polynomial.

To bridge this gap in the understanding of complexity and the practical performance of algorithms, researchers introduced the notion of average case analysis. Average Case analysis introduces a probability distribution on the inputs and analyzes the expected time complexity of an algorithm when the input is drawn from such a distribution.
Thus,

Definition 1.2 (Average Case Complexity). *For a family of distributions parametrized by the size, μ_n: a probability distribution on $\mathcal{X}_n$. the average case time complexity of an algorithm $\mathcal{A}$ is given by :*

$$\mathcal{C}_{avg}(n) = \mathbb{E}_{x\sim\mu_n}\mathcal{T}_{\mathcal{A}}(x).$$

Ideally, one would empirically estimate a distribution on the inputs arising in practice and then try to analyze the expected performance of an algorithm on inputs drawn from this distribution. However analysis of these algorithms on arbitrary distributions has turned out to be extremely hard and the known results are mostly on "natural" distributions (Normal, Uniform etc.). Even though a polynomial expected time complexity on such distribution gives an indication of the good performance in practice, it hardly explains it. This is because the inputs in practice could be far from the distribution that analysis has been performed on.
Average Case analysis of simplex yielded some satisfying results. The simplex algorithm was analyzed in a variety of settings and its average case complexity was shown to be polynomial. ([20])
Spielman and Teng in their groundbreaking paper [22] introduced the idea of smoothed analysis of algorithms which is in a sense a compromise between worst case and average case analysis. It inherits the merits of both these methods.

Definition 1.3 (Smoothed Complexity). *The smoothed complexity of algorithm $\mathcal{A}$ on input space $\mathcal{X}$ is $C_{smooth}(n,\sigma)$ if*

$$C_{smooth}(n,\sigma) = max_{x\in\mathcal{X}_n}\mathbb{E}_{r\sim\mathcal{N}(0,1)}\mathcal{T}_{\mathcal{A}}(x+\sigma||x||r)$$

where $||x||$ is an appropriately chosen norm.

2 Smoothed Analysis

In smoothed analysis of an algorithm, we assume that the inputs are perturbed by a slight random noise. We measure the complexity of the algorithm as an expectation of the time taken by the algorithm over an input where the expectation is over the random perturbation. The time complexity is now given in terms of the input size and the variance of the perturbation used. Note that, a very large perturbation will essentially reduce to average case analysis while a perturbation of zero corresponds to the worst case analysis. Thus in a very concrete sense, smoothed analysis is a middle-ground between the two standard approaches to measure the performance of algorithms.
If an algorithm has low smoothed complexity , then one expects it to work well in practice. This is because, the data that are input to our algorithms are inherently subjected to noise from the measurements. Consider the example of linear programming algorithms applied to machine learning problems. A yet another way to understand low smoothed complexity is to note that one must be unlucky to choose the input on which the algorithm performs badly.
Note that this idea is very different from average case analysis. In average case analysis, the inputs that make the algorithm take high runtimes could have small probability mass and yet be dense in a region. Thus, if you take a worst case input for the algorithm and perturb it slightly, there is no guarantee that the new input thus produced wouldn't be just as bad for the algorithm. Smoothed complexity on the other hand guarantees such a behavior. Thus, in a very strong *input-distribution independent* sense, a low smoothed complexity guarantees low runtime for the algorithm.
It should also be noted that smoothed complexity is not equivalent to analyzing higher moments in average case analysis. Thus in a region of very small probability mass under the distribution, the algorithm could have an exponential behavior making every bounded moment small. However, the smoothed complexity of such an algorithm will be exponential.

We now formally define what we mean by *low smoothed complexity*. We say that an algorithm $\mathcal{A}$ has a low smoothed complexity, if $C_{smooth}(\mathcal{A})(n)$ is polynomial in n and $\frac{1}{\sigma}$.

3 Perturbation Models

The major design choice in smoothed analysis is the random perturbation. When describing the simplex algorithm for constraint vectors in $\mathbb{R}^n$, Spielman and Teng used gaussian perturbation which we define below. In problem with input vectors in $\mathbb{R}^n$ (continuous sets) such perturbations make sense. However, this is not always the case. Consider for example algorithms whose inputs are drawn from $\{0,1\}^n$, a discrete set. Here discuss some of the perturbation models used in recently proved results. We start with gaussian perturbations in the analysis of the simplex algorithm. This list is based on the survey by Spielman and Teng ([21]).

Definition 3.1 (Gaussian Perturbations)**.** *For $\mathbf{x} \in \mathbb{R}^n$, a gaussian perturbation of $\mathbf{x}$ is defined as the random variable $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{g}$ where $\mathbf{g}$ is a gaussian random variable.*

Definition 3.2 (Boolean Perturbations)**.** *Let $\mathbf{x} = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ or $\{-1,1\}^n$. A σ-boolean perturbation of $\mathbf{x}$ is a random string $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$ where $\tilde{x}_i = x_i$ with probability $(1-\sigma)$.*

In problems such as scheduling, packing and sorting, the inputs are integers of certain size. Banderier, Beier and Mehlhorn [2] propose the following partial bit randomization model.

Definition 3.3 (Partial Bit Randomization)**.** *Let z be an integer and k be a positive integer indicating the size of the perturbation. A k-partial bit randomization of z is an integer $\tilde{z}$ obtained from z by replacing its k least significant bits by a random number in $[0 : 2^{k-1}]$ according to some distribution over $[0 : 2^{k-1}]$.*

However, in comparison based sorting and online problems each input consists of a sequence of elements. Bander, Beier and Mehlhorn [2] introduce the following the *partial permutation model*.

Definition 3.4 (Partial Permutation Perturbations)**.** *Let s be a sequence of n elements. Let $0 \leq \sigma \leq 1$ be the magnitude of the perturbations. A σ- partial permutation of s isa random sequence $\tilde{s}$ obtained from s by first building a subset S by independently selecting each index number from $[n]$ with probability σ and then randomly permuting elements of s in position S while rataining the positions of all other elements.*

For analyzing computational geometry algorithms, it might make more sense to analyze the expected time complexity of the algorithm under uniform perturbations in an appropriate ball.

Definition 3.5 (Uniform Ball Perturbation)**.** *Let $x \in \mathbb{R}^n$. A uniform ball perturbation of radius σ of x is a random vector $\tilde{x}$ chosen uniformly from the ball of radius σ centered at x.*

In analyzing scientific computing algorithms, that depend on the sparsity of the problem instance, clearly the perturbation above would destroy the sparse structure. For these problems, the following might serve as a better models of perturbation.

Definition 3.6 (Relative Gaussian Perturbations)**.** *Let $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$. A relative gaussian perturbation of $\mathbf{x}$ is a random vector $\tilde{\mathbf{x}} = (x_1, x_2, ..., x_n)$ where $\tilde{x}_i = x_i(1+g_i)$ where g_i is a gaussian random variable with standard deviation σ.*

Definition 3.7 (Zero-Preserving Gaussian Perturbations)**.** *Let $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$. A zero-preserving σ-gaussian perturbation of $\mathbf{x}$ is a vector $\tilde{\mathbf{x}} = (\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$ where $\tilde{x}_i = \tilde{x}_i + (1 - IsZero(x_i))g_i$ where g_i ia gaussian random variable with standard deviation σ and $IsZero(x) = 1$ if $x = 0$ and is 0 otherwise.*

4 The Shadow-Vertex-Pivot Simplex Algorithm

Roughly, the simplex method proceeds by doing a walk from one vertex of the feasible polyhedron of the LP to the other. At each step, it walks to a vertex that improves the objective value of the LP. There are several version of the simplex algorithm based on the method to choose the next vertex in the walk. These methods are known as the pivot rules. Spielman and Teng use the shadow vertex pivot rule in the simplex method and show that the smoothed complexity of the simplex algorithm with the shadow vertex pivot rule is polynomial.

We describe the shadow vertex simplex method for linear programming here. This description is based on [22]. The shadow vertex pivot rule was introduced by Gaas and Saaty [10]. The algorithm begins at a vertex $\mathbf{x}$ and chooses an objective function $\mathbf{t}$ that is optimized at $\mathbf{x}$. The algorithm then gradually interpolates between $\mathbf{t}$ and $\mathbf{z}$ (the objective of the linear program), and finds an optimal solution for each such interpolant. The vertices that the algorithm walks through are contained in the set of vectors that optimize the functions in the span of $\mathbf{t}$ and $\mathbf{z}$.

We now describe this algorithm in a little more detail. First we note the following facts which are standard about a solution to a linear program. Consider a linear program:

$$\begin{aligned} &\text{maximize } \mathbf{z}^T\mathbf{x} \\ &\text{subject to:} \langle \mathbf{a_i}|\mathbf{x}\rangle \leq \mathbf{y_i} for 1 \leq i \leq n \end{aligned}$$

Claim 4.1 (Optima are extreme points.)**.** *A point* $\mathbf{x}$ *is a solution to the linear program* (**??**) *if and only if*

1. $\mathbf{x}$ *is feasible:* $\langle \mathbf{a_i}|\mathbf{x}\rangle \leq \mathbf{y_i}$ *for* $1 \leq i \leq n$
2. $\mathbf{x}$ *is extremal: there exists a non-empty set* B *of inequalities such that* $\langle \mathbf{a_i}|\mathbf{x}\rangle = \mathbf{y_i} \forall i \in B$.
3. $\mathbf{x}$ *is optimal:* $\mathbf{z}$ *can be expressed as* $\mathbf{z} = \sum i \in B \alpha_i \mathbf{a_i}$ *for* $\alpha_i \geq 0$.

Definition 4.1 (**optVert**)**.** *Let* $\mathbf{z}, \mathbf{a_1}, \mathbf{a_2}, \cdots, \mathbf{a_n} \in \mathbb{R}^d$ *and* $\mathbf{y} \in \mathbb{R}^n$*. Then* $\mathbf{optVert}_z(\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_n}; \mathbf{y})$ *is the set of* $\mathbf{x}$ *solving:*

$$\begin{aligned} &\textit{maximize } \mathbf{z}^T\mathbf{x} \\ &\textit{subject to:} \langle \mathbf{a_i}|\mathbf{x}\rangle \leq \mathbf{y_i} for 1 \leq i \leq n \end{aligned}$$

When there is no $\mathbf{x}$ *solving the above linear program, we set* **optVert** *to be . We will write* $\mathbf{optVert}_z$ *when the other parameters are understood by context.*

To understand this method, let's first consider the case when we know of vectors $\mathbf{t}$ and $\mathbf{x}_0$ such that $\mathbf{optVert_t} = \mathbf{x}_0$. We will then modify this algorithm to work without this assumption. For $\mathbf{t}$ and $\mathbf{z}$ we define the interpolated objective function

$$\mathbf{q}_\lambda = (1 - \lambda)\mathbf{t} + \lambda\mathbf{z}$$

The shadow vertex pivot rule based simplex proceeds by varying λ from 0 to 1 and tracks $\mathbf{optVert}_{q_\lambda}$. Let $\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k$ be the vertices encountered by the algorithm. Set λ_i such that $\mathbf{x}_i \in [\lambda_i, \lambda_{i+1}]$.

The shadow vertex method is motivated by the observation that the simplex algorithm is very simple in two dimensions. The set of feasible points in two dimensions form a polygon and the simplex method walks along the boundary of the polygon. The shadow vertex method builds along this intuition. Let $\mathbf{z}$ be the objective of a linear program and let $\mathbf{t}$ be an objective function optimized by $\mathbf{x}$, a vertex of the polytope of feasible points for the linear program. The shadow vertex ethod considers the *shadow* of the polytope- the projection of the polytope onto the plane spanned by $\mathbf{z}$ and $\mathbf{t}$.

However, the description of the algorithm that we gave works under the assumption that an objective function $\mathbf{t}$ and an extreme point $\mathbf{x}_0$ can be found. The Two Phase shadow vertex method is a simplex algorithm which modifies the primal shadow vertex method described above in order incorporate this step.

Algorithm 1 Primal Shadow-Vertex Method

Require: Input: $\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_n}, \mathbf{y}, \mathbf{z}$, and $\mathbf{x_0}$ ad $\mathbf{t}$ satisfying $\{x_0\} = optVert_t(\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_n}; \mathbf{y})$.

1: Set $\lambda_0 = 0$ and $i = 0$.
2: Set λ_1 to be maximal such that $\{x_0\} = optVert_{\mathbf{q}_\lambda}$ for $\lambda \in [\lambda_0, \lambda_1]$.
3: **while** $\lambda_{i+1} < 1$ **do**
4: Set $i = i + 1$
5: Find an x_i for which there exists a $\lambda_{i+1} > \lambda_i$ such that $x_i \in optVert_{\mathbf{q}_\lambda}$ for $\lambda \in [\lambda_i, \lambda_{i+1}]$. If no such x_i exists then return *unbounded*.
6: Let λ_{i+1} be maximal such that $x_i \in optVert_{\mathbf{q}_\lambda}$ for $\lambda \in [\lambda_i, \lambda_{i+1}]$.
7: **end while**
8: **return** $\mathbf{x}_i$.

4.1 Analysis of Two-Phase Shadow-Vertex Method

The proof of polynomial smoothed complexity of the shadow vertex method is quite involved to be presented in brief here. So we discuss a high level approach to the proof. The basic idea is to show that the shadow size, that is the size number of vertices in the projection of the feasible polytope over the two dimension plane spanned by **any** two fixed vectors $\mathbf{z}$ and $\mathbf{t}$ is small after a random gaussian perturbation. This is the major idea in the paper.

Theorem 4.1 (Shadow Size)**.** *Let $d > 3$ and $n > d$. Let c and t be independent vectors in $\mathbb{R}^d$ and let $a_1, a_2, ..., a_n$ be gaussian random vectors in $\mathbb{R}^d$ of variance $\sigma^2 \leq \frac{1}{9d \log n}$ centered at points each of norm at most 1. Then, the expected number of vertices of the shadow polygon formed by the projection of $\{x : x^t a_i \leq 1\}$ onto $Span(t, c)$ is at most $O(\frac{nd^3}{\sigma^6})$.*

The exponent of σ in this bound was later improved to 2 by Deshpande and Spielman [3].
However, the proof is not done yet. Note that in the simplex method, the vectors $\mathbf{z}$ and $\mathbf{t}$ are not independent of the vectors defining the LP instance. Thus the shadow size bound above doesn't hold for the case of shadow vertex method. However, this bound is only a bit worsened when analyzing the method using the Shadow size bound as a blackbox:

Theorem 4.2 (Time Complexity of Simplex Method)**.** *Let $d > 3$ and $n > d$. Let $c \in \mathbb{R}^d$ and $b \in \{-1, 1\}^n$. Let $a_1, a_2, ..., a_n$ be gaussian vectors in $\mathbb{R}^d$ of variance $\sigma^2 \leq \frac{1}{9d \log n}$ centered at points each of norm at most 1. Then the expected number of simplex steps taken by the two-phase shadow vertex simplex algorithm to solve the program specified by b, c, and $a_1, a_2, ..., a_n$ is at most*

$$(\frac{nd}{\sigma})^{O(1)}$$

5 Submodular Function Minimization

We now discuss a widely used algorithm for submodular function minimization. First we start by discussing the submodular functions. A function $f : \{0, 1\}^n \to \mathbb{R}$ is *submodular* if

$$\forall S, T \subseteq [n] : f(S \cup T) + f(S \cap T) \leq f(S) + f(T). \tag{1}$$

Submodular functions have been extensively studied in the context of combinatorial optimization [6, 18, 8, 15] where the functions under consideration (such as the cut function of a graph) are submodular. An equivalent formulation of submodularity is that of decreasing marginal returns,

$$\forall S \subseteq T \subseteq [n], i \in [n] \setminus T : f(T \cup \{i\}) - f(T) \leq f(S \cup \{i\}) - f(S), \tag{2}$$

and thus submodular functions are also a topic of study in economics and the algorithmic game theory community [4, 17]. In most contexts, the submodular functions considered are non-negative [4, 7, 17, 24, 19, 1, 12].

Problem 1 (Submodular Function Minimization)**.** *Let $f : \{0,1\}^n \to \mathbb{R}$ be a submodular function where $\{0,1\}^n$ is the set of incidence vectors of all possible subsets of the ground set $X = [n]$. Given a query access to f find $x \in \{0,1\}^n$ such that $f(x) \leq f(z)$ for every $z \in \{0,1\}^n$ in polynomial time and polynomially many queries to f.*

Submodularity is considered to be a discrete analog of convexity, and several combinatorial algorithms have been developed to minimize a submodular function that run in polynomial time.
In particular, submodular functions have a convex extension (Lovasz extension) on $[0,1]^n$. Thus the minimum of the Lovasz extension is achieved at an extreme point and minimizing the Lovasz extension is equivalent to minimizing the submodular function. One question, here, of course is the computational efficiency of evaluating the Lovasz extension. However, as we see in the definition below, computing the Lovasz extension for a submodular function, at any given value $x \in [-1,1]^n$ requires just n queries to submodular function oracle.

Definition 5.1 (Lovasz Extension)**.** *Let $f : \{0,1\}^n \to \mathbb{R}$ be a submodular function. Let $x \in [0,1]^n$. For any $0 \leq \lambda \leq 1$ define the threshold set $T_{>\lambda}(x)\{i : x_i \geq \lambda\}$. Then the Lovasz extension of f at x is given by*

$$\mathcal{L}_f(x) = \mathbb{E}_{\lambda \sim U[0,1]}[f(T_{>\lambda}(x))]$$

where $U[0,1]$ is the uniform distribution on $[0,1]$.

There have been several combinatorial algorithms for submodular function minimization. The current fastest strongly polynomial time algorithm is due to Iwata-Orlin[13] which runs in time $O(n^5.T + n^6)$. The fastest weakly polynomial time algorithm is also due to Iwata-Orlin[13], and this runs in $\widetilde{O}(n^4.T + n^5)$. Note that T here is the function querying complexity.

The Fujishige-Wolfe algorithm, which lacks any formal complexity bounds, has been seen empirically to outperform these algorithms [16]. It casts the problem of minimizing a submodular function to that of finding a minimum norm point in a convex polytope, and then uses the algorithm for this task due to Wolfe[25].

We have the following definitions for polyhedra based on submodular functions.

Definition 5.2. *Given a submodular function $f : 2^V \to \mathbb{R}$, the submodular polyhedron is defined as*

$$\mathcal{P}(f) = \{x \in \mathbb{R}^n \,|\, x(Y) \leq f(Y)\ \forall\, Y \subseteq V\},\ \text{where } x(Y) = \sum_{i \in Y} x_i$$

The base polyhedron is defined as

$$\mathcal{B}(f) = \{x \in \mathbb{R}^n \,|\, x(Y) \leq f(Y)\ \forall\, Y \subset V,\ x(V) = f(V)\}$$

Some properties of $\mathcal{B}(f)$ are as follows.

Fact 1. *Let $\mathcal{B}(f)$ be the base polyhedron of a submodular function f, then*

1. *$\mathcal{B}(f)$ is convex and compact*
2. *$\mathcal{B}(f) = conv(ext(\mathcal{B}(f)))$, where $conv(S)$ is the convex hull of a set S and $ext(S)$ is the set of extreme points of a convex set S.*
3. *$dim(\mathcal{B}(f)) \leq n - 1$*
4. *$x \in ext(\mathcal{B}(f))$ iff $\exists$ a permutation $\pi : V \to V$ s.t.*

   $$\forall\, i \in \{1, \ldots, n\},\ x_{\pi(i)} = f(B_i) - f(B_{i-1}),\ \text{where } B_i = \{\pi(j) \,|\, j \leq i\} \text{ and } B_0 = \phi$$

   Such an x is called an extreme base.

5. *By Caratheodory's theorem, any $x \in \mathcal{B}(f)$ can be expressed as a convex combination of atmost n extreme bases i.e.*

$$x = \sum_{i=1}^{n} \lambda_i x_{\pi_i},\ \lambda_i \geq 0,\ \sum_{i=1}^{n} \lambda_i = 1,$$

where x_{π_i} is the extreme base corresponding to permutation π_i.

The problem of submodular function minimization (SFM) is the following

$$\min_{A \subseteq V} f(A) \tag{3}$$

The following lemma will be useful in the subsequent theorem.

Lemma 5.1. *For any $x \in \mathcal{B}(f)$, if we have two sets X_1 and X_2 s.t. $x(X_1) = f(X_1)$ and $x(X_2) = f(X_2)$, then $x(X_1 \cup X_2) = f(X_1 \cup X_2)$ and $x(X_1 \cap X_2) = f(X_1 \cap X_2)$*

Proof. Since $x \in \mathcal{B}(f)$, we know that $x(X_1 \cap X_2) \leq f(X_1 \cap X_2)$ and $x(X_1 \cup X_2) \leq f(X_1 \cup X_2)$. Now,

$$\begin{aligned} x(X_1 \cup X_2) &= x(X_1) + x(X_2) - x(X_1 \cap X_2) \\ &\geq f(X_1) + f(X_2) - f(X_1 \cap X_2) \\ &\geq f(X_1 \cup X_2) \end{aligned}$$

Thus, $x(X_1 \cup X_2) = f(X_1 \cup X_2)$. $x(X_1 \cap X_2) = f(X_1 \cap X_2)$ follows similarly. □

Now, we have the following theorem due to Fujishige[9], which implies that SFM can be solved by finding the norm minimizer in $\mathcal{B}(f)$.

Theorem 5.1. *For a submodular function f, let x^* be the min-norm-point in $\mathcal{B}(f)$ i.e.*

$$x^* = \operatorname*{argmin}_{x \in \mathcal{B}(f)} ||x||^2$$

Then, $A^* = \{i \,|\, x_i < 0\}$ is a minimizer of the submodular function f i.e.

$$A^* = \operatorname*{argmin}_{A \subseteq V} f(A)$$

Proof. Let x^* be a minimizer of the norm in $\mathcal{B}(f)$. Then, for any (u, v)- pair ($u \in V$ and $v \in V$) with $x^*_u < 0$ and $x^*_v > 0$, $\exists\, X_{uv} \subseteq V$ with $v \notin X_{uv}$ and $u \in X_{uv}$ s.t. $x^*(X_{uv}) = f(X_{uv})$.

If this is not the case for some (u, v)-pair, then we can prove a contradiction as follows. Choose $\varepsilon_1 = \min\{f(Y) - x^*(Y) \,|\, u \in Y, v \notin Y\}$ and $\varepsilon = \min\{\varepsilon_1, |x^*_u|, x^*_v\}$. Since, there is no set X_{uv}, we must have $\varepsilon > 0$. Now, construct a new x' as follows

$$x'_i = \begin{cases} x^*_i & i \neq u, v \\ x^*_u + \epsilon & i = u \\ x^*_u - \epsilon & i = v \end{cases}$$

We see that $x' \in \mathcal{B}(f)$. This is because :

$$\begin{aligned} &\forall\, Y \subseteq V \text{ s.t. } u \notin Y,\, v \notin Y, \quad x'(Y) = x^*(Y) \leq f(Y) \\ &\forall\, Y \subseteq V \text{ s.t. } u \in Y,\, v \in Y, \quad x'(Y) = x^*(Y) \leq f(Y) \\ &\forall\, Y \subseteq V \text{ s.t. } u \notin Y,\, v \in Y, \quad x'(Y) = x^*(Y) - \epsilon \leq f(Y) \\ &\forall\, Y \subseteq V \text{ s.t. } u \in Y,\, v \notin Y, \quad x'(Y) = x^*(Y) + \epsilon \leq x^*(Y) + f(Y) - x^*(Y) = f(Y) \end{aligned}$$

Also,

$$||x'||^2 = ||x^*||^2 + 2\varepsilon(x_u + \varepsilon) - 2\varepsilon x_v < ||x^*||^2$$

Thus, we have a contradiction.

Let X_{uv} correspond to the described set for the pair (u, v). Now, we see that,

$$A^* = \bigcup_{u:x^*_u<0} \bigcap_{v:x^*_v>0} X_{uv}$$

So, using Lemma 5.1, $x^*(A^*) = f(A^*)$.
But, we also have $\forall Y \subseteq V, x^*(A^*) \leq x^*(Y) \leq f(Y)$ since $x^*(A^*)$ would be the sum of only negative terms in x^*.
Thus, $f(A^*) = \min_{Y \subseteq V} f(Y)$. □

The norm minimizer of $\mathcal{B}(f)$ can be found using Wolfe's algorithm for computing the minimum norm point of a convex polytope. Given a convex set specified by its extreme points, Wolfe's algorithm provides a method to produce a point closest to the origin in this convex set. The algorithm proceeds iteratively and at each step adds an extreme point to the a set S. The algorithm then produces a minimum norm point in the aff S by solving a simple linear program and then projects this point down to the convex set by a series of steps. At the end of this step, the algorithm throws away certain extreme points and is left with a new set S. This step continues until a stopping condition is reached. One can show that the stopping condition indeed implies that the algorithm has terminated with the optimal point. The algorithm terminates in a finite time and correctly computes the points closest to the origin. We prove these results below.
However, the time complexity of this algorithm in general can be a super-linear function of the number of extreme points. When applied to the setting of submodular function, by the theorem (5.1), we want to find a point closest to the origin in the base polyhedron. The algorithm proceeds by iteratively adding the bases. Note that in our case, the number of bases can be $n!$ which is very large. However, finding a new base is made easy by a simple idea in equation (4).
Offhand, one would predict that Wolfe's algorithm when applied to submodular function minimization would take exponential time in the worst case. However, this algorithm is in fact the best performing algorithm for minimization of submodular functions and typically the run time scales as $O(n^3)$ ([16]) in experiments. A recent study has shown an empirical scaling of $\Omega(n^7)$ for a family of submodular functions [23].There is a strong belief in the community that this algorithm indeed terminates in polynomial time. However, no provably polynomial time analysis or a bad family of submodular functions taking super polynomial time is known. Thus the time complexity of this algorithm is a major question owing to its remarkable effectiveness. See Algorithm 2 for a description in our setting.
Using Wolfe's algorithm involves solving a linear optimization problem over $\mathcal{B}(f)$. Fortunately, it turns out that this problem can be solved efficiently using a greedy algorithm due to Edmonds [5], as stated in the following theorem.

Theorem 5.2. *Given a vector $x \in \mathbb{R}^n$, we have a linear optimization problem*

$$\operatorname*{argmin}_{y \in \mathcal{B}(f)} x^T y$$

A solution to the problem is given as : Let π be a permutation that defines an ascending order over the elements of x i.e. $x_{\pi(1)} \leq x_{\pi(2)} \leq \ldots \leq x_{\pi(n)}$. Then, the extreme base corresponding to the permutation π, is a solution to the optimization problem.

Proof. Let, y be the required extreme base. Then, $y_{\pi(i)} = f(B_i) - f(B_{i-1})$, where $B_i = \{\pi(j) \,|\, j \leq i\}$ and $B_0 = \phi$. Also, $y(B_i) = f(B_i)$.

Algorithm 2 Computing the Min-Norm-Point in $\mathcal{B}(f)$

1: $x \leftarrow$ An extreme base corresponding to a random permutation π
2: $S \leftarrow \{x\}$
3: Compute

$$y^* = \underset{y \in \mathcal{B}(f)}{\operatorname{argmin}} \, x^T y$$

If $x^T y^* = x^T x$, then STOP. Else, $S \leftarrow S \cup \{y^*\}$.
4: Compute

$$z^* = \underset{z \in \text{aff}(S)}{\operatorname{argmin}} \, ||z||^2,$$

where aff(S) is the affine hull of set S.
If $z^* \in conv(S)$, then

- $x \leftarrow z^*$
- Let S' be the subset of S corresponding to positive coefficients in the expression of $z^* = \sum_{s \in S} \lambda_s s$. Then, $S \leftarrow S'$.
- Goto Step 3

Else, Goto Step 5
5: Let, z' be the intersection of the line joining z^* to x and $conv(S)$, which is nearest to z^*. Let, S' be the subset of S constituting the face of $conv(S)$ on which z' lies. Then,

- $x \leftarrow z'$
- $S \leftarrow S'$
- Goto Step 4

Now, for any other $y' \in \mathcal{B}(f)$, consider

$$\begin{aligned}
x^T y - x^T y' &= \sum_{i=1}^{n} x_{\pi(i)}(y_{\pi(i)} - y'_{\pi(i)}) \\
&= \sum_{i=1}^{n} x_{\pi(i)}(y(B_i) - y(B_{i-1}) - y'(B_i) + y'(B_{i-1})) \\
&= \sum_{i=1}^{n-1} (x_{\pi(i)} - x_{\pi(i+1)})(y(B_i) - y'(B_i)) + x_{\pi(n)}(y(B_n) - y'(B_n)) \\
&= \sum_{i=1}^{n-1} (x_{\pi(i)} - x_{\pi(i+1)})(f(B_i) - y'(B_i)) \\
&\leq 0, \text{ since } y' \in \mathcal{B}(f)
\end{aligned}$$

Thus, $x^T y$ is optimal. □

We now present some lemmas concerning the algorithm.

Lemma 5.2. *x is the min-norm point of $conv(P)$ iff $x^T y \geq x^T x$, $\forall\, y \in conv(P)$. Also,*

$$\min_{y \in conv(P)} x^T y = \min_{y \in P} x^T y = x^T x$$

Proof. x is a min-norm point $\Leftrightarrow$ it is the projection of $\mathbf{0}$ onto $conv(P)$

$$\begin{aligned}
&\Leftrightarrow \langle 0 - x, y - x \rangle \;\leq\; 0, \forall\, y \in conv(P) \\
&\Leftrightarrow \langle x, x \rangle \;\leq\; \langle x, y \rangle, \forall\, y \in conv(P)
\end{aligned}$$

So,

$$\min_{y \in conv(P)} x^T y = x^T x$$

Also, the minimum for $x^T y$ occurs at an extreme point of P. So,

$$\min_{y \in conv(P)} x^T y = \min_{y \in P} x^T y$$

□

Observation 5.1. *We see that*

1. *The stopping criterion is valid, since $\mathcal{B}(f)$ is convex*
2. *The hyperplane $H = \{y \mid x^T y = x^T x\}$ is a separating hyperplane for $conv(S)$ if x is a min-norm point of $conv(S)$.*

Lemma 5.3. *Just before Step 3 of the algorithm, we have an x and an S such that*

$$x = \operatorname*{argmin}_{z \in \text{aff}(S)} ||z||^2 \text{ and } x \in conv(S)$$

Thus, $x^T y = x^T x$, $\forall\, y \in \text{aff}(S)$ and x is a unique point for the given set S

Proof. The lemma is trivially true for the initial x and S. Now, at any subsequent point, the algorithm returns to Step 3 only from Step 4, and this happens when the required conditions stated in the lemma have been satisfied.
So, since x is the minimizer over the aff(S), the inequalities stated in Lemma 5.2 hold with equality. Also, since the norm function is a strictly convex function, its minimizer over a convex set is unique. □

Lemma 5.4. *S is always affinely independent. Thus, $|S| \leq n$.*

Proof. The lemma is trivially true for the initial S. Now, suppose y gets added to S at Step 3. Since, y gets added only when $x^T y < x^T x$, we infer that $y \notin \text{aff}(S)$, using Lemma 5.3. Thus, assuming S is affinely independent just before Step 3, S remains affinely independent after Step 3. The only other step that modifies S is Step 5. However, it only replaces S by one of its subsets. Thus, if S is affinely independent just before Step 5, it is affinely independent after it. Thus, S is always affinely independent.
Now, since S lies in a space of dimension atmost n-1, we have $|S| \leq n$. □

Observation 5.2.
1. *Let us call a step 3-step 4 cycle a major cycle and a step 5-step 4 cycle a minor cycle*
2. *Total number of minor cycles in the algorithm is bounded by the total number of major cycles. This is because, every vertex removed in a minor cycle would have been added in some major cycle.*
3. *Number of major cycles of the algorithm is trivially bounded by the number of affinely independent sets. This is because every x at the beginning of a major cycle is unique and has a corresponding affinely independent set S. Now, since a major cycle guarantees decrease in the objective, this infers that no x repeats and therefore, no S repeats.*
4. *Thus, the algorithm terminates.*

Now, each major step and minor cycle require $O(n^3)$ time. Thus, the time complexity of the algorithm may be expressed as

$$O(n^3 \times \text{No. of major cycle} + n^3 \times \text{No. of minor cycles}) = O(n^3 \times \text{No. of major cycles})$$

So, if using smooth analysis, we can bound the expected number of major steps in the algorithm, subject to a suitable perturbation, we would be able to justify the empirical efficiency of the Fujishige-Wolfe algorithm.

References

[1] Maria-Florina Balcan and Nicholas J. A. Harvey. Learning submodular functions. 2011.

[2] Cyril Banderier, Cyril B, Rene Beier, and Kurt Mehlhorn. Smoothed analysis of three combinatorial problems, 2003.

[3] Amit Deshpande and Daniel A. Spielman. Improved smoothed analysis of the shadow vertex simplex method. In *FOCS*, pages 349–356, 2005.

[4] Shahar Dobzinski, Noam Nisan, and Michael Schapira. Truthful randomized mechanisms for combinatorial auctions. pages 644–652, 2006.

[5] 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.

[6] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, 1971.

[7] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. pages 461–471, 2007.

[8] Marshall L. Fischer, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions II. *Mathematical Programming Studies*, 8:73–87, 1978.

[9] Satoru Fujishige, Takumi Hayashi, and Shigueo Isotani. The minimum-norm-point algorithm applied to submodular function minimization. In *Tech. Report, Kyoto University, Kyoto Japan*, 2006.

[10] S. Gaas and T.Saaty. The computational algorithm for the parametric objective function. *Naval Research Logistics Quarterly*, pages 2:39–45, 1955.

[11] Donald Goldfarb and William Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Appl. Math.*, 1:277–285, 1979.

[12] Anupam Gupta, Moritz Hardt, Aaron Roth, and Jonathan Ullman. Privately releasing conjunctions and the statistical query barrier. 2011.

[13] 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*, SODA '09, pages 1230–1237, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[14] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities, III*, pages 159–175. Academic Press, 1972.

[15] László Lovász. Submodular functions and convexity. In A. Bachem et al., editor, *Mathematical Programming: The State of the Art*, pages 235–257. 1983.

[16] S. T. McCormick. Submodular function minimization. In *Handbook on Discrete Optimization*. Elsevier, 2006.

[17] Elchanan Mossel and Sebastien Roch. On the submodularity of influence in social networks. pages 128–134, 2007.

[18] G. L. Nemhauser, L. A. Wolsey, and M. L. Fischer. An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming*, 14(1):265–294, 1978.

[19] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. 2011.

[20] S. Smale. On the average number of steps in the simplex method of linear programming. *Mathematical Programming*, 27:241–262, 1983.

[21] Spielman and Teng. Smoothed analysis of algorithms and heuristics. In *London Mathematical Society Lecture Note Series, Cambridge University Press*, volume 331. 2006.

[22] Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *STOC*, pages 296–305, 2001.

[23] Jeffrey Bilmes Stefanie Jegelka, H. Lin. Fast approximate submodular minimization. Neural Information Processing Systems, (NIPS), 2011, 2011.

[24] Jan Vondrák. Symmetry and approximability of submodular maximization problems. pages 651–670, 2009.

[25] Philip Wolfe. Finding the nearest point in A polytope. *Mathematical Programming*, 11(1):128–149, December 1976.