Submodularity in Optimization and Property Testing

Xingyou Song

Abstract

In this report we discuss recent topics and findings related to submodular optimization from the standpoint of both property testing and random sampling. We show that there are many variants of querying, ranging from user-defined to sampling from an adversarial distribution, and how this may affect approximation, submodular greedy algorithm, and curvature.

Contents

1 Introduction and Background

Submodular functions are ubiquitous in many fields, such as machine learning, optimization, and computational economics. A more detailed overview of the techniques can be found in [AK12], but we present the basics for now.

Definition 1. *A set function $f : 2^{[X]} \to \mathbb{R}$ is submodular if it satisfies the property*

$$f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$$

for all $A, B \subseteq [X]$.

Equivalent definitions are - $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B) \;\; \forall A \subseteq B$. There are many types of submodular functions; take for example: [1]

- Coverage Function - There exists a ground set $U = \{e_1, ..., e_m\}$, weights w_i corresponding to each e_i, and subsets of $U : A_1, ..., A_X$, such that

  $$f(T) = w\left(\bigcup_{i \in T} A_i\right)$$

  where $w(S) = \sum_{i \in S} w_i$.
- Or -XOR - $f(S) = \max_{(S_1,...,S_\ell) \text{ partitions of } S} \sum_{i=1}^{\ell} f_i(S_i)$, $\forall i \;\; f_i(S) = \max_{j \in S} w_{ij}$
- Unit Demand - $f(S) = \max_{j \in S} w_j$
- Linear - $f(S) = \sum_{i \in S} w_i$
- Rank - $f(S) = \text{Rank}(\{\bigcup_{i \in S} r_i\})$ for some vectors $r_1, ..., r_X$ in some linear space.

[1] To have a general sense of why the above are submodular, they all follow the fact that they give diminishing returns.

While the submodular function need not necessarily be monotonically increasing, for the purposes of this report, we will assume it is always monotone.

Coverage functions may be interpreted as the amount of coverage for wireless networks on a map, etc., in which one object may only be covered once. For Or-XOR and Unit Demand, these are motivated by combinatorial auctions, in which each an auctioneer wishes to maximize the value he sees, or find the maximal partitioning of objects that will give optimal social welfare. Rank functions of matroids are generally used in many machine learning applications, such as reducing or increasing the dimensionality of a set of feature vectors. [2]

In general, while f has 2^X inputs, we can see that for some functions, the "complexity" of them are not so high; for coverage functions, if $|U|$ is polynomial in size, then f may be computed efficiently, whereas an exponential $|U|$ may cause problems. In various literature, it is not assumed that an $O(1)$ oracle access to f is allowed for various reasons.

One of which is exactly the complexity issue, but other reasons are due to the lack of a proper representation of f, especially if f is the expected value of a stochastic process, especially in utility maximization and social networks.

2 Optimization

In general, submodular functions are important mainly due to their properties in optimization. There are various problems associated with submodular functions, such as the following:

- Submodular Constrained Maximization (NP-Hard) - $\max_{|S|=k} f(S)$ [3]
- Submodular Argument Minimization (NP-Hard) - Find S such that $|S|$ is minimized and $f(S) \geq c$ for some constant c.
- Submodular Minimization (Pseudo-Polynomial Time) - $\min_{|S|=k} f(S)$ [4]

Non-monotone submodular optimization encompass a variety of problems as well, mostly within the setting of graph cutting algorithms. However, they in general do not follow the same principle of greedy search, and obtain a general (3/4)-hard approximation ratio. [**?**]

In this report, we will only focus on monotone submodular (NP-Hard) maximization. The following is a well known theorem that has a strict $(1 - 1/e)$ approximation ratio: $f(S_{poly}) = (1-1/e) \cdot f(S_{opt})$ where $|S_{poly}|, |S_{opt}| = k$ and S_{poly} is the result of a polynomial time approximation algorithm (greedy), and S_{opt} is the true maximizer:

Algorithm 1 Greedy Algorithm

1: **for do** $1 \leq i \leq k$
2: Find $e \in [X] - S_i$ such that $f(S_i \cup e)$ is maximized.
3: $S_{i+1} = S_i \cup e$
4: Output S_k

The $(1 - 1/e)$ result comes directly from a proof of Fiege on the hardness of approximation of set cover [Fei98], using Fiege's k-prover system, with a reduction from 3SAT-5.

However, the $(1 - 1/e)$ bound may be improved to $\frac{1}{\mathcal{K}_f}(1 - e^{-\mathcal{K}_f})$ if we know the curvature $\mathcal{K}_f = 1 - \min_{j \in V} \frac{f(j|V_J)}{f(j)}$ [RI13], which is useful for different submodular functions, e.g. linear vs

[2] It is well known that the rank function also admits a hardness of approximation for $(1 - 1/e)$.

[3] In general cases, S comes from a matroid constraint

[4] This is called pseudo-polynomial because it relies on polytope cutting methods when looking at the representation of f in a polytope setting, which gives minimums, but is not always guarenteed to work.

covering. The curvature in some sense denotes the property of the function at various points, which is related to the fact that most points are concentrated around the center at $|S| = X/2$.

In the greedy algorithm, since we cannot assume that evaluating $f(1)$ runs in $O(1)$, thus the running time of this algorithm, if $k << X$, will be $O(kX \cdot R(f))$ where $R(f)$ is the time to evaluate f. In fact, an example of such a function is the optimization of social networks - a graph with X vertices and Y edges will take $O(XY)$ to merely simulate the expectation of influence propagation [5]. Another example is evaluating a covering function with an exponentially sized universe U; storing the information of U as well as performing set union tasks can be computationally expensive.

2.1 Faster Optimization

2.1.1 Social Network Case

In the influence maximization problem, we are given a directed graph $G = (V, E)$, in which the edges (i, j) are weighted with probability p_{ij}. An "active" node i attempts to convince his inactive node j with probability p_{ij}. If he fails, there may be other attempts to influence j, but i will be considered "dead" after this turn. Given that we are allowed to active k nodes initially, we want to maximize expected number of influenced nodes overall in this propagation model, $f(S) = \mathbb{E}[\sigma(S)]$, where $\sigma(S)$ is the number of nodes influenced after one simulation, given S as seed. This can be thought of as an "extended" coverage function; we can let $V = X$ and the first neighbors of X as the "ground set", second-degree neighbors as a "second ground set", etc. However, this function f is very difficult to calculate, as it involves multiple Monte-Carlo simulations; it is one of the most complex but naturally occuring functions.

In [Bor14], Borgs used a reverse polling technique for an approximation of the submodular function associated to the expected influence on a graph. Similar in spirit to approximating a sparse graph with a spanning tree, it calculated a low-complexity coverage function $g(S)$ in order to approximate $f(S)$, by randomly sampling and generating covers as needed.
Their algorithm streamed a cover e_ℓ (with weight $w_\ell = 1$ always) one at a time; by doing so, maximization for the coverage function could be done efficiently by merely having counters c_j for each $j \in [X]$ which counted the number of covers that contained j. The Greedy Algorithm would then only take the element $\arg\max_{a \in [X]-S_i} c_a$ and then update the surrounding c_j's after removing all covers on j. The algorithm ran in $\tilde{O}(k(X + Y))$ with the same results of the greedy algorithm, a significant speedup in practicality for large graphs.
This algorithm, which streams essentially "data" from a complex set and then performs faster optimization is well motivated by the sublinear community as well as the PAC-learning machine learning community. Although this algorithm explicitly uses properties of the function itself to perform approximation (e.g. it is a social network), there are both results when f is treated as a value oracle, as well as when a full description of f is given. For the value-oracle case, this follows PMAC-learning.

2.1.2 Probably Mostly Approximately Correct

In PAC (Probably Approximately Correct) learning, the goal of an algorithm is to determine the best (or correct) parameters that represent a data distribution (which maps features to $\{0, 1\}$ booleans) in machine learning, through sampling. [6] This can be generalized to optimization, in which we want to correctly perform an approximation algorithm that is based on samples. Balcan and Harvey [MB11] extended this to the PMAC (Probably Mostly Approximately Correct) framework, from boolean to real valued functions. Formally, given a polynomially number of

[5] more details can be seen in [Bor14]

[6] The complexity of boolean functions here is deemed Rademacher Complexity, a machine learning term which approximates how well a hypothesis class of learner functions can predict given random data from a fixed distribution. This is not too far from random sampling in an adversarial case.

samples $(S_i, f^*(S_i)$ from an unknown distribution D on 2^X, from target function $f^* : 2^X \rightarrow \mathbb{R}_+$, the goal of the algorithm is to generate f that approximates f^*, i.e.

$$\Pr_{S_1,S_2,...\sim D}[\Pr_{S\sim D}[f(S) \leq f^*(S) \leq \alpha f(S)] \geq 1-\epsilon] \geq 1-\delta$$

for some ϵ, δ, α. Note that this is slightly different from property testing, in the sense that there is an unknown distribution for the samples involved; the sampling is adversarial in nature. This is a worst case bound for normal property testing, because the user needs to discover properties of the function for any given adversarial distribution over samples.

Various approximations of submodular functions have been studied using this model, but we will show the main recent ones, that all give various insights to the properties and optimization of submodularity.

3 Property Testing and Sampling

3.1 Coverage Functions and Succinctness

One important aspect of coverage functions is their representability, and the complexity of their approximation. In [Bad11], Badanidiyuru et al. show that coverage functions with high universe cardinality can be approximated by coverage functions with much lower universe cardinality (admit good "sketches"). More formally, they show that one can obtain a $(1+\epsilon)$ "sketch" (i.e. $f(S)/(1+\epsilon) \leq f'(S) \leq f(S)\ \forall S$) coverage function f' of (coverage function) f with a universe $|U'| \leq O(X^2/\epsilon^2)$ space, with high probability.

Their assumption states that however, the user must have complete access to the original universe $|U|$ (which might be exponential size), and is allowed to efficiently generate a probability distribution on the elements of $|U|$, for the purposes of sampling. While this is targeted for the purposes of pre-processing, in general, this is inefficient when $|U|$ is exponentially sized, as well as the fact that there does not exist a MCMC-type efficient sampling algorithm. The gist of their solution is to define $q(u) = \max\{\frac{w(u)}{f(\{i\})}, i \in X, u \in A_i\}$ and then sample proportionally, i.e. pick u from $|U|$ with probability proportional to $q(u)$. After t samples $x_1, ..., x_t$, they let $U' = \{x_1, ..., x_t\}$ and weight the distinct elements by the number of times they appear. From a use of the Chernoff bound for probability concentration, $t = O(X^2)$. This algorithm intuitively is choosing the ground elements based on their highest marginal ratio.

The unfortunate part of their algorithm is that it does not admit an efficient approximation for a coverage function with an exponential size universe, unless sampling can be done efficiently. However, they do establish the existence of low-complexity approximate coverage functions. A key open question will be the establishment of an approximate coverage function that is more efficient.

A question proposed however, is the optimizability of coverage functions based on value sampling.

3.1.1 Sublinear Time for Known Covers

In this section, there are fully sublinear algorithms that are independent of the size of the universe, given that the universe had been stored originally. Investigated recently in [Fu16] If the sets $A_1, ..., A_X$ are provided as oracles, in which a runtime of $O(1)$ is given for querying size of $|A_i|$, random sampling $u \in A_i$, and membership (is $u \in A_i$?) then [Fu] formally shows that there is an $(1-1/e)$ approximation which runs in $O(poly(k)X \cdot \log X)$ time, or more specifically, an algorithm that outputs a $|S| = k$ such that $f(S) \geq (1-1/e)f(S_{opt})$ with probability of success greater than 3/4, in runtime $O(k^6 \log(3m/k)m)$. Noting that this runtime essentially ignores the size of the universe. In his work, the algorithm applies the heuristic of approximating the cardinality of $|B-A|$, given two sets A,B. [7]

[7] $B-A$ is defined as all elements in B that are not in A

Using this approximation, after greedily selecting $x_1, ..., x_i \in [X]$ which correspond to the sets $A_{x_1}, ..., A_{x_i}$, he approximates the marginal benefit of adding a new element x_{i+1} by approximating $|A_{x_{i+1}} - (A_{x_1} \cup ... \cup A_{x_i})|$. The sampling technique follows from this idea, in which a constant number of elements are sampled from $A_{x_{i+1}}$, and checked for membership within any of $A_{x_1}, ..., A_{x_i}$. The concentration bounds for constant number of samples imply the independence from the size of the universe. Naturally, this follows from certain data structures that support the $O(1)$ membership queries, usually in modern databases that use the B-tree structure once the universe elements are memoized.

3.1.2 Optimization and Recoverability of Coverage Functions

In Aviad et al. [EB16a], they show that it is hard to merely optimize based on uniform sampling given a value oracle for a coverage function f. Formally speaking, given polynomially many samples $\{(S_i, f(S_i)\}$ where S_i is drawn uniformly over 2^X, it is impossible to then generate a set $|S| = k$ such that $f(S) \geq \alpha \cdot \max_{|T|=k} f(T)$, where $\alpha = O(n^{-1/4})$ as well as $2^{-\Omega(\sqrt{\log X})}$. They show that there exist a class of coverage functions $\mathcal{F}$, each function $f_j \in \mathcal{F}$ parametrized by a set $T_j \subseteq [X]$, that have a common two coverage functions g, b (representing "good" and "bad"). Formally, they define

$$f_i(S) = g(S \cap T_j) + \sum_{i=1, i \neq j}^{m} b(S \cap T_j)$$

These g, b in which for small sets ($|S| \leq \ell$), they agree completely ($g(S) = b(S)$), but g is much larger than b on higher order S points, with a fixed curvature. Since the uniform sampling is in general concentrated on S such that $|S| \approx X/2$, it follows that the sampling most of the time will only see the values of the "good" function g, and thus the f_j's are indistinguishable among others in $\mathcal{F}$. Note that the idea of curvature is introduced here, and this effectively shows that many coverage functions can have a high concentration for similar values around the middle of $X/2$, although completely different in other places, making optimization difficult. The variant of this is when the samples are user-chosen instead, and one seeks to understand the structure of the function without adversarial samples. This is reinforced by the fact in the following paper, on coverage testing.

3.1.3 Testing Coverage

Because coverage functions are a strict subset of submodular functions, it is harder to test for coverage. In fact, the following work shows this is the case.

In [DC15]. Chakrabarty et al. shows that for coverage functions, if f is a coverage function with ground set U, given as a value oracle, it requires $O(m|U|)$ queries to the oracle, which will reconstruct the function. However, if f is not a coverage function, he proves that it requires at least $Q = 2^{X-1}$ queries to determine this, or else there exists a coverage function g that will agree with f on $Q < 2^{X-1}$ queries.

Their proof relies on the fact that coverage functions have a particular decomposition similar to the Fourier Decomposition. They define a transformation $D : f \to d$ in which

$$d(S) = \sum_{T: S \cup T = [m]} (-1)^{|S \cap T|+1} f(T)$$

For the inverse, they show that

$$f(T) = \sum_{S \subseteq [m]: S \cap T \neq 0} d(S)$$

The coefficients $\{d(S) : S \subseteq [X]\}$ define a unique function, and are non-negative iff f is coverage. They then define a measure of distance, similar to Hamming distance, called the D-distance, which is the fraction of D-coefficients that are negative. Their algorithm for determining the weights and

ground set of the coverage function is based on an inductive procedure that calculates the relations between each of the elements in $[X]$, in order to determine their support. Note that in general, $f(T)$ is close to the summation of the coefficients $d(S)$ with large S - a probabilistic measure will imply higher concentration larger $d(S)$.

However, their results only consider the Hamming distance, rather than the ℓ_2 distance, which is more natural for optimization. It is open to construct an approximation for f using mainly the coefficients $d(S)$ where S has high cardinality; in general, this seems unlikely, because $d(S)$ is determined by many values of $f(T)$ because $S \cup T = [m]$ means T is likely to be large.

3.1.4 Fourier and Spectral Analysis of Submodular Functions

In this section, we note some worthy results that are motivated by the Fourier tools in property testing. It is well known that in general, approximations of a function by low degree polynomials are useful for their efficient querying.

In [VF15], Feldman and Vondrak noted that there is an efficient low-degree polynomial (i.e. linear combination of low-degree boolean Fourier Characters) that approximates various functions, including coverage and submodular functions. Formally, if f is submodular, then they show that $\deg_\epsilon^{\ell_2}(f) = O(\log(1/\epsilon)/\epsilon^{4/5})$, where $deg_\epsilon^{\ell_2}$ is the lowest possible degree number (largest $|T|$) for a Fourier function $g(S) = \sum_{T \subseteq [X]} \hat{g}(T) \cdot \chi_T(S)$, such that $\sqrt{\mathbb{E}_{\mathcal{U}}[(f(S) - g(S))^2]} = \ell_2(f, g) \leq \epsilon$ where $\mathcal{U}$ is the uniform distribution over all subsets of $[X]$. They construct this approximate boolean polynomial g using $2^{\tilde{O}(1/\epsilon^{4/5})} \cdot \log X$ queries. In fact, they show that coverage functions give a degree lower bound $\Omega(\log(1/\epsilon))$ and an upper bound $O(\log(1/\epsilon))$ while also giving a degree lower bound $\Omega(1/\epsilon^{4/5})$ and upper bound $O(1/\epsilon^{4/5} \cdot \log(1/\epsilon))$. It is interesting to note however, these are independent of the size of the set X, which implies that we may be able to run faster algorithms that ignore the size of X.

Their result comes from the fact that submodular functions are Noise Stable [Che12] - If we denote the noise and stability operators as respectively [8]

$$T_\rho f(S) \sum_{T \subseteq [X]} \rho^{|T|} \hat{f}(T) \chi_T(S) \qquad \mathbb{S}_\rho \sum_{T \subseteq [X]} \rho^{|T|} \hat{f}(S)^2$$

Then they show that

$$T_\rho f(S) \geq \rho f(S) + ((1-\rho)/2)(f(0) + f([X]))$$

Informally this implies that as $\rho \to 0$, the larger sets T will have $\rho^{|T|}$ diminish faster, and hence $\hat{f}(T)$ dominates the constants within the function, for small $|T|$.

As an aside, if we are given in memory $\hat{g}(T)$ for all $|T| \leq \deg_\epsilon^{\ell_2}(f) = O(\log(1/\epsilon)/\epsilon^{4/5})$ (i.e. given the function g entirely), then suppose that we conduct the Submodular Greedy Algorithm on g. Note that this algorithm can be sped up similarly to the coverage function greedy algorithm, noticing that T act as "covers" but with varying signs. We first take the element $e_1 \in [X]$ that minimizes the sum of the available weights $\hat{g}(T)$ (we can memoize this selection by use of a counting vector $\mathbf{c} = (\mathbf{c_1}, ..., \mathbf{c_{[x]}})$ where $e_1 \in T$ because $\chi_T(\{e_1\}) = -1$ if $e_1 \in T$ and 1 otherwise. Then, flipping the signs for all constants $\hat{g}(T)$ where $e_1 \in T$, we may apply the same concept again and again in order to produce a faster greedy algorithm.

This takes $O(k \cdot \deg_\epsilon^{\ell_2}(f))$ (after memoizing over the X), with only $2^{\tilde{O}(1/\epsilon^{4/5})} \cdot \log X$ value queries for the function f. The analysis of this has not been done before, but this suggests it may lead to a more efficient algorithm that is not dependent on X, the size of the domain, which gives us a $(1 - 1/e - \delta)$ approximation, where $\delta = \delta(\epsilon)$ is a function of the ℓ_2 approximation error.

[8] Noise operators are used to check the high/low degree terms and their influence in boolean Fourier analysis

References

[AG16] S. Singla A. Gupta, V. Nagarajan. Adaptivity gaps for Stochastic Probing - Submodular and XOS Functions. 2016.

[AK12] D. Golovin A. Krause. Submodular Function Maximization. 2012.

[Bad11] A. Badanidiyuru. Sketching Valuation Functions. *STOC*, 2011.

[Bor14] C. Borgs. Maximizing social influence in nearly optimal time. *SODA*, 2014.

[Che12] M. Cheraghchi. Submodular Functions are Noise Stable. *SODA*, 2012.

[CS10] J. Vondrak C. Seshadrhi. Is sumodularity testable? 2010.

[DC15] Z. Huang D. Chakrabarty. Recognizing Coverage Functions. *SIAM J. Discrete Math*, 2015.

[EB16a] Y. Singer E. Balkanski, A. Rubinstein. The limitations of optimization from samples. *NIPS*, 2016.

[EB16b] A. Bommireddi E. Blais. Testing Submodularity and Other Properties of Valuation Functions. 2016.

[Fei98] U. Feige. A threshold of ln n for approximating set cover. *STOC*, 1998.

[Fu16] B. Fu. Partial Sublinear Time Approximation and Inapproximation for Maximum Coverage. 2016.

[MB11] N. Harvey M. Balcan. Learning Submodular Functions. *STOC*, 2011.

[MB14] Y. Liang N. Du M. Balcan, L. Song. Learning Time Varying Coverage Functions. *NIPS*, 2014.

[RI13] J. Bilmes R. Iyer, S. Jegelka. Curvature and optimal algorithms for learning and minimizing submodular functions. *NIPS*, 2013.

[UF07] J. Vondrak U. Feige, V. Mirrokni. Maximizing non-monotone Submodular Functions. *FOCS*, 2007.

[VF13a] J. Vondrak V. Feldman, P. Kothari. Representation, Approximation and Learning of Submodular Functions Using Low Rank Decision Trees. *JLMR*, 2013.

[VF13b] P. Kothari V. Feldman. Learning Coverage Functions and Private Release of Marginals. 2013.

[VF15] J. Vondrak V. Feldman. Tight Bounds on Low-Degree Spectral Concentration of Submodular and XOS Functions. *FOCS*, 2015.