Submodular Optimization and Machine Learning: Theoretical Results, Unifying and Scalable Algorithms, and Applications

Rishabh Iyer

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2015

Reading Committee:

Jeff Bilmes, Chair

Andreas Krause

Carlos Guestrin

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Submodular Optimization and Machine Learning: Theoretical Results, Unifying and Scalable Algorithms, and Applications

Rishabh Iyer

Chair of the Supervisory Committee:
Professor Jeff Bilmes
Department of Electrical Engineering

In this dissertation, we explore a class of unifying and scalable algorithms for a number of submodular optimization problems, and connect them to several machine learning applications. These optimization problems include,

1. Constrained and Unconstrained Submodular Minimization,

2. Constrained and Unconstrained Submodular Maximization,

3. Difference of Submodular Optimization

4. Submodular Optimization subject to Submodular Constraints

The main focus of this thesis, is to study these problems theoretically, in the light of the machine learning problems where they naturally occur. We provide scalable, practical and unifying algorithms for all the above optimization problems, which retain good theoretical guarantees, and investigate the underlying hardness of these optimization problems. We also study natural subclasses of submodular functions, along with theoretical constructs, which help connect theory to practice by providing tighter worst case guarantees. While the focus of this thesis is mainly theoretical, we also empirically demonstrate the applicability of these techniques on several synthetic and real world problems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would firstly like to thank my advisor Jeff Bilmes for introducing me to several interesting problems in the intersection of discrete optimization, machine learning and submodularity, and for always helping me strike the right balance between theory and practice, ultimately culminating in this thesis. I particularly like his insight in very theoretical and mathematical problems, and at the same time, maintaining an amazing intuition into what works in practical real world applications. I would also like to thank him for his support and understanding of several personal issues, and accomodating requests which ordinarily would be hard to accomodate. On a whole, Jeff, you have been an amazingly helpful friend, guide and advisor. I would also like to thank my thesis committee members, Andreas Krause, Carlos Guestrin, Rekha Thomas, Anna Karlin and Maryam Fazel for their comments, which helped me refine several aspects of this thesis. I thank Rekha and Maryam for two amazing courses on Discrete optimization and Convex optimization, and Jeff for his course on Submodular Optimization (for which, I also acted as a Teaching Assistant subsequently), all of which helped me significantly in my research.

I would like to thank my mentors at Microsoft Research, Redmond – Matthai Phillipose, Chris Meek, Patrice Simard and Max Chickering, during my two internships there. Their collaboration and guidance has had a deep impact in trying to solve challenging real world problems. I would also like to thank my many collaborators – Stefanie Jegelka, Yoshinobu Kawahara, Sebastian Tschiatschek, Ganesh Ramakrishnan, Hui Lin, Chris Meek, Max Chickering, Patrice Simard, Matthai Phillipose, Kai Wei, Yuzong Liu, Bethany Herwaldt, Subhasis Chaudhuri, Sunita Sarawagi, and Ramakrishna Bairi, for giving me a very diverse research experience in the projects we worked together. Special thanks to my undergraduate

advisor Subhasis Chaudhuri for first introducing me to research, and my colleagues Ronak Shah, Rushikesh Borse, Soham Mehta and Jaideep Joshi for giving me the taste of initial research. I would like to also thank my past and present Melodi lab mates John Halloran, Andrew Guillory, Hui Lin, Stefanie Jegelka, Kai Wei, Shenjie Wang, Bethany Herwaldt, Chandrashekhar Lavania, Rahul Kidambi, Galen Andrew, Eric Xie, and Jennifer Gillenwater for our numerous discussions and for making my office life fun. I also thank Karthik Narayanan, Dennis Meng, Amin Jalali, Dvijottham Krishnamurthy and Scott Wisdom for several fruitful discussions and support. I would also like to acknowledge the sources of my funding which include an NSF grant, an Intel Science Fellowship, a Microsoft Research Ph.D Fellowship and a Yang outstanding student award.

I would like to thank my spiritual teacher Shri Radhanath Swamiji, whose words of wisdom, books and teachings have sustained my personal life. Special gratitude to my mentor Shri Govinda Das for always being my best friend, philoshpher, guide, and councellour and for being personally there to listen to, and provide the perfect advice for every problem, without any expectations. I would also like to thank the huge list of my friends at the Sri Sri Radha-Gopinath community in Chowpatty, Mumbai whose affection and kind words have led me this far. Also my sincere appreciation extends to the Vedic Cultural Centre, Sammamish and Shri Harivilas Prabhu, for the amazing experiences, which has helped me maintain my personal life in Seattle, including the free foods, weekend volunteering and amazing Indian festivals in the Seattle area. I thank Rasavati, Srivas, Rakesh, Gupta and others for their close friendship, and I am indebted to my roommates and flatmates, Rakesh, Ashwin, Sravan and Akash for their care and support through the years. I also thank several friends Aniruddh, Rahul, Baldev, Jaishank, Priyav and many others for their friendship, support and the fun we had together growing up.

I would like to express my immense gratitude to my parents Krishnan and Uma, for their unconditional love, encouragement and support, and for being there with me physically,

emotionally and mentally, every single day of my life. I sincerely thank them for careful nurturing and upbringing they provided to mould me into what I am, and for being there with me whenever I needed them, to the extent to making several trips to the US to visit me. Ideally, I would need to write another dissertation to express my gratitude to both of you, but I summarize my deep love and affection for you with these few words, and by dedicating this thesis to both of you. Last but not the least, I thank my best friend and fiancèe Wedashree for coming into my life at the right time – in fact, exactly when I was beginning to write this thesis. I thank her for all the love, emotional and mental support, and most importantly, for her companionship, which made this phase of writing my thesis so enjoyable. Thank you for everything, Wedah!

DEDICATION

To My Parents Krishnan and Uma, My Spiritual Mentors Radhanath Swamiji, Govinda Dasji and my best friend Wedashree

Chapter 1

INTRODUCTION

1.1 Submodular Functions

Having been known as a key structural property for problems in combinatorial optimization, economics, operations research, and game theory, submodularity is gaining popularity in a large number of areas. Along with its natural connection to many application domains, it also admits a number of interesting theoretical characterizations. Submodular functions are a special class of discrete set functions, defined on the Boolean lattice $[0,1]^n$. Formally given a ground set $V = \{1, 2, \cdots, n\}$, a function $f : 2^V \rightarrow \mathbb{R}$ is submodular if,

$$f(S) + f(T) \geq f(S \cap T) + f(S \cup T), \forall S, T \subseteq V. \tag{1.1}$$

One can define the *gain* of adding an item j to a set S, as,

$$f(j|S) \triangleq f(S \cup j) - f(S) \tag{1.2}$$

An equivalent definition of submodularity, is that f is submodular if and only if

$$f(j|S) \geq f(j|T) \text{ for all } S \subseteq T \text{ and } j \notin T. \tag{1.3}$$

The function f is monotone iff $f(j|S) \geq 0, \forall j \notin S, S \subseteq V$. Moreover, if the inequalities in Eqns (1.1) and (1.3) occur as equalities, the function f is *modular*. In other words, a modular function satisfies $f(X) = f(\emptyset) + \sum_{i \in X} f(i)$. Furthermore, a set function f is *supermodular*, if the set function $g(X) = -f(X)$ is submodular. Through the rest of this thesis, we shall also assume without loss of generality that $f(\emptyset) = 0$.

Submodular Functions are a rich class of set functions, for which many otherwise intractable discrete optimization problems become easy to solve or approximate. Akin to convexity,

what makes this class interesting is the existence of optimal and near optimal algorithms for submodular function minimization and maximization. Besides having nice theoretical properties, submodular functions also naturally arise in many real world applications.

1.2 Submodularity in Machine Learning

Though combinatorial problems abound in machine learning and a number of application domains, the role of submodular functions in machine learning is much less well understood than that of convex functions. In particular, while convex optimization has found enormous applicability in machine learning, submodularity is still only beginning to show broad applicability in machine learning and its applications. Moreover, work on submodular optimization in the optimization and operations research literature has been largely unaware of unique problems arising in machine learning and its applications. Therefore, existing standard algorithms do not exploit certain variants of the submodular problems arising in machine learning. In this dissertation, we shall investigate theoretical constructs, and provide unifying and scalable algorithms for several natural submodular optimization problems occurring in machine learning applications.

The general problem of machine learning, is to construct fitting models for making predictions based on some observed data. To be precise, denote $\mathbf{x}$ as the input (which could be defined via a set of features) and $y \in \mathcal{C}$ be a structured output representation (it could denote, for example, sets, matchings, cuts or paths etc). The task is to find a model $f(\mathbf{x}; y; \Lambda)$, where Λ denotes a set of parameters. Hence, the prediction problem, which is also called inference, involves optimizing (either minimizing or maximizing, depending on whether f is a score function or a loss function) the function $f(\mathbf{x}; y; \Lambda)$ over the set $\mathcal{C}$. The function f could either be a discriminant function (non probabilistic), or could be a probabilistic of the form, $f(\mathbf{x}; y; \Lambda) = p(y|\mathbf{x}; \Lambda)$. In this dissertation, we assume that the output space y is binary – i.e can be represented as a set $Y \subseteq V = \{1, 2, \cdots, n\}$, and is structured. y, equivalently Y, could represent for example, a set, a matching in a graph, an s-t cut or a s-t path etc. Correspondingly, $\mathcal{C}$ could represent a set of sets, a set of matchings, cuts, paths etc. Since $|\mathcal{C}|$

is exponential, a naïve search would clearly be infeasible.

Many machine learning problems occur as subset selection problems, and thereby fit the problem description above. These include, for example, news, document or image summarization, where one wants to find a subset representing a *summary* of the data. Similarly, in the problem of sensor placement, one wants to find a subset of locations to place sensors. Image segmentation, involves finding a subset of foreground pixels, while the problems of training data subset selection and active learning, involve finding subsets of training data to train classifiers. Many these problems can naturally be modeled as submodular optimization problems.

Submodular functions offer a rich class of *expressive models* for many of these machine learning problems. They have a unique property of enabling complex interactions between the objects, while simultaneously retaining guarantees for inference. They occur in applications, either as probabilistic point processes, like the pairwise Markov Random Fields (also called Ising models) [Geman and Geman, 1984] and Determinantal Point Processes [Kulesza and Taskar, 2012, Macchi, 1975], or as discriminant functions [Lin and Bilmes, 2011c, 2010, Tschiatschek et al., 2014, Simon et al., 2007, He et al., 2012, Jegelka and Bilmes, 2011d] – i.e directly modeled as loss or score functions in applications. Submodular functions naturally model two interesting phenomena: they capture notions of cooperation between items [Jegelka and Bilmes, 2011d, Boykov and Kolmogorov, 2004], and simultaneously model aspects of coverage and diversity [Lin and Bilmes, 2011c, 2010, Kulesza and Taskar, 2012]. The former property is due to the diminishing returns property and subadditivity, since the cumulative cost of a sum of items, is less than the sum of the individual costs of items (i.e a submodular function f satisfies $f(X) \leq \sum_{j \in X} f(j)$. This is often very realistic in modeling costs between items, since more the number of items one buys, larger discount in the price. This property is also called economies of scale. The flip side of submodularity, is in relation to modeling diversity and coverage. This follows, because natural models for coverage like set-cover function, and diversity in the form of similarity penalizing functions or determinantal point processes, are either submodular or closely related to submodularity [Lin and Bilmes, 2011c,

Kulesza and Taskar, 2012]. Hence submodular functions provide a rich class of models for these very important and challenging machine learning applications.

1.3 Optimization Problems related to Submodular Functions

In this dissertation, we investigate five classes of optimization problems related to submodular functions. The first two problems are minimization and maximization problems of the following form:

$$\text{Problem 1: } \operatorname*{argmin}_{X \in \mathcal{C}} f(X), \qquad \text{Problem 2: } \operatorname*{argmax}_{X \in \mathcal{C}} f(X) \tag{1.4}$$

where $f : 2^V \to \mathbb{R}$ is a submodular set function, and $\mathcal{C} \subseteq 2^V$ is a family of feasible solution sets. The set $\mathcal{C}$ could express, for example, that solutions must be an independent set in a matroid, a limited budget knapsack, or a cut (or spanning tree, path, or matching) in a graph. Both these problems occur naturally in applications. Submodular minimization is natural in applications when one wants to model notions of cooperation, attraction and algorithmic complexity. Examples of this include image segmentation and denoising in computer vision. Similarly applications where submodular maximization is natural, is when one wants to capture diversity, coverage and value of information. Examples of these problems include data summarization, feature selection and sensor placement.

Sometimes, one might want to simultaneously model diversity or coverage and cooperation. In this case, it is natural to consider formulations for simultaneously minimizing one submodular function, while maximizing another. The last three optimization problems address this. The third optimization problem involves minimizing the difference between submodular functions. Given two submodular functions f and g, and a set function $v(X) \triangleq f(X) - g(X)$, consider the optimization problem:

$$\text{Problem 3: } \operatorname*{argmin}_{X \subseteq V} [v(X)] \equiv \operatorname*{argmin}_{X \subseteq V} [f(X) - g(X)]. \tag{1.5}$$

Observe that minimizing the difference of submodular functions, is equivalent to to maximizing the difference of submodular functions. In particular, note that minimizing $v(X)$ (Problem 3)

is equivalent to maximizing $-v(X)$, which is again a difference of submodular functions:

$$\text{Problem } 3': \ \operatorname*{argmax}_{X \subseteq V}[-v(X)] \equiv \operatorname*{argmax}_{X \subseteq V}[g(X) - f(X)] \tag{1.6}$$

This formulation is natural in applications where one wants to simultaneously maximize one submodular function (say, diversity, coverage or information) while minimizing another (say, cooperation or algorithmic complexity). Furthermore, a number of combinatorial optimization problems can naturally be expressed as a difference of submodular functions. In fact, as we shall show, every set function optimization problem is expressible as a difference of submodular functions.

Finally, we consider two new optimization problems:

$$\text{Problem 4: } \min\{f(X) \,|\, g(X) \geq c\}, \quad \text{and} \quad \text{Problem 5: } \max\{g(X) \,|\, f(X) \leq b\}, \tag{1.7}$$

where f and g are submodular functions, and where b and c refer to budget and cover parameters respectively. The corresponding constraints are called the submodular cover [Wolsey, 1982] and submodular knapsack [Atamtürk and Narayanan, 2009] respectively and hence we refer to Problem 4 as *Submodular Cost Submodular Cover* (henceforth SCSC) and Problem 5 as *Submodular Cost Submodular Knapsack* (henceforth SCSK). Note that, Problems 4 and 5 are a special case of Problems 1 and 2, where the constraints $\mathcal{C}$ are themselves defined via submodular functions. Moreover, both these problems are closely related to Problem 3, which can be seen as a Lagrangian form of both problems. Similar to Problem 3, the motivation for Problems 4 and 5 stems from problems that require minimizing a certain submodular function f while simultaneously maximizing another submodular function g. In many of these applications, one of the submodular functions naturally occurs as a constraint, and hence Problems 4 and 5 are fitting models.

This thesis aims to investigate unifying algorithms and theoretical results for each of the above optimization problems. We also motivate all of them by several machine learning applications, and empirically demonstrate the applicability of our techniques.

1.4 Important Properties of Submodular Functions

In this section, we define some of the properties which we shall repeatedly use through this thesis. Several of these ideas depends on the intimate connection between submodularity, convexity and concavity, which we investigate in detail in Chapter 2.

Similar to convexity, submodular functions have a number of desirable properties, which ensures that several transformations are submodular. For a more elaborate description of the properties of submodular functions, refer to [Fujishige, 2005].

Proposition 1.1. *If f is submodular then the following are also submodular:*

- *Complement:* $g(X) \triangleq f(V \setminus X)$
- *Truncations:* $g(X) \triangleq \min(f(X), c)$ *where c is a constant, f is monotone submodular.*
- *Min of two Submodular functions:* $h(X) \triangleq \min(f(X), g(X))$*, if* $m(X) = f(X) - g(X)$ *is monotone submodular.*
- *Scalar Multiplication:* $g(X) \triangleq cf(X)$ *where c is a non-negative constant.*
- *Subset Minimum:* $g(X) \triangleq \min_{Y \subseteq X} f(Y)$
- *Superset Minimum:* $g(X) \triangleq \min_{Y \supseteq X} f(Y)$
- *Fixed Set Intersection:* $g(X) \triangleq f(X \cap Y)$ *where Y is some fixed set.*
- *Fixed Set Union:* $g(X) \triangleq f(X \cup Y)$ *where Y is some fixed set.*
- *Mixtures of submodular functions: If* $f_1, f_2, \cdots, f_m$ *are m submodular functions, the mixture function,* $g(X) = \sum_{i=1}^{m} \lambda_i f_i(X)$*, where* $\lambda_i \geq 0$*, is submodular.*

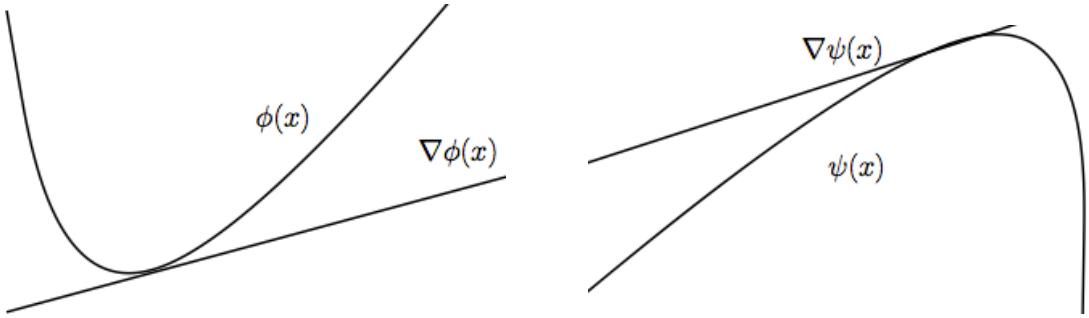

Figure 1.1: Convex and Concave functions with sub and super gradients respectively

Submodular functions have been strongly associated with convex functions, to the extent that submodularity is often regarded as the discrete analog of convexity [Fujishige, 2005]. This relationship is evident by the fact that submodular function minimization is easy, in that there exist strongly polynomial time algorithms which achieve it. This is akin to convex minimization which is also easy. In addition, submodular functions also admit a natural convex extension, known as the Lovász extension, which is easy to evaluate [Lovász, 1983] and optimize. We investigate these connections in detail throughout this thesis, and particularly in Chapter 2.

Submodular functions also have some properties, which are unlike convexity, but perhaps more akin to concavity. Submodular function maximization is known to be NP hard. However, there exist a number of constant factor approximation algorithms, based on simple greedy or local search heuristics [Feige et al., 2011b, Lee et al., 2009a, Nemhauser et al., 1978], and some recent continuous approximation methods [Chekuri et al., 2011, Feldman et al., 2011]. This is unlike convexity where maximization can be hopelessly poor. Furthermore, submodular functions have a diminishing returns property which is akin to concavity, and concave over modular functions, sometimes called decomposable functions [Stobbe and Krause, 2010], are known to be submodular.

1.4.1 Discrete Semi-gradients:

Convex functions naturally have subgradients (linear lower bounds), while concave functions have supergradients (linear upper bounds) – shown in Figure 1.3. Submodular functions have both sub and super gradients. While the subgradients of submodular functions were known and studied [Fujishige, 2005], we show that they also form linear upper bounds and supergradients. We investigate the properties of the semigradients and semidifferentials in Chapter 2.

Subdifferential: The subdifferential $\partial_f(Y)$ of a submodular set function $f : 2^V \rightarrow \mathbb{R}$ for a set $Y \subseteq V$ is defined [Fujishige, 2005], analogously to the subdifferential of a continuous convex function:

$$\partial_f(Y) = \{y \in \mathbb{R}^n : f(X) - y(X) \geq f(Y) - y(Y) \text{ for all } X \subseteq V\} \tag{1.8}$$

For a vector $x \in \mathbb{R}^V$ and $X \subseteq V$, we write $x(X) = \sum_{j \in X} x(j)$ — in such case, we say that x is a normalized *modular* function.

We shall denote a subgradient at Y by $h_Y \in \partial_f(Y)$. The extreme points of $\partial_f(Y)$ may be computed via a greedy algorithm: Let σ be a permutation of V that assigns the elements in Y to the first $|Y|$ positions ($\sigma(i) \in Y$ if and only if $i \leq |Y|$).

Each such permutation defines a chain with elements $S^\sigma_0 = \emptyset$, $S^\sigma_i = \{\sigma(1), \sigma(2), \ldots, \sigma(i)\}$ and $S^\sigma_{|Y|} = Y$. This chain defines an extreme point h^σ_Y of $\partial_f(Y)$ with entries (see figure 1.2 for an illustration)

$$h^\sigma_Y(\sigma(i)) = f(S^\sigma_i) - f(S^\sigma_{i-1}). \tag{1.9}$$

Superdifferentials: We can also define superdifferentials $\partial^f(Y)$ of a submodular function [Jegelka and Bilmes, 2011d, Iyer and Bilmes, 2012a, Iyer et al., 2013b] at Y:

$$\partial^f(Y) = \{y \in \mathbb{R}^n : f(X) - y(X) \leq f(Y) - y(Y); \text{for all } X \subseteq V\} \tag{1.10}$$

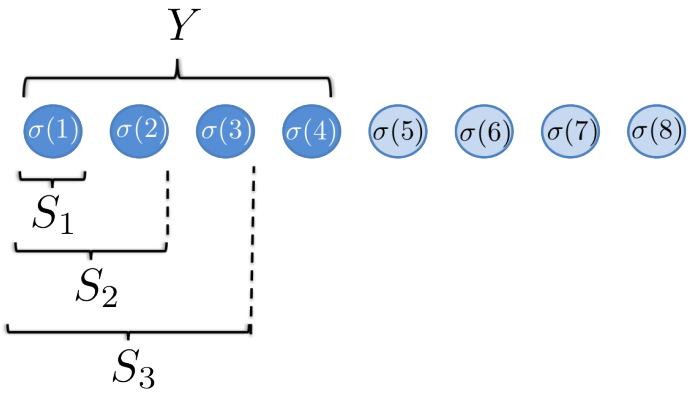

Figure 1.2: Illustration of the chain of sets and permutation σ

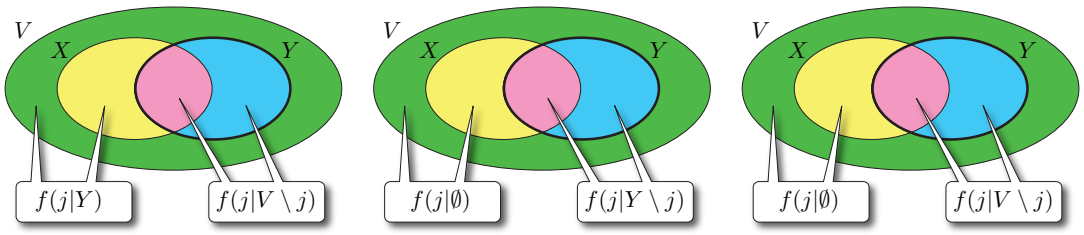

Figure 1.3: Illustrating the supergradients $\hat{g}, \check{g}$ and $\bar{g}$.

We denote a generic supergradient at Y by g_Y. It is easy to show that the polyhedron ∂^f is non-empty. We define three special supergradients $\hat{g}_Y$, $\check{g}_Y$ and $\bar{g}_Y$ as follows:

$$\begin{aligned} \hat{g}_Y(j) &= f(j \mid V \setminus \{j\}) & \hat{g}_Y(j) &= f(j \mid Y) \\ \check{g}_Y(j) &= f(j \mid Y \setminus \{j\}) & \check{g}_Y(j) &= f(j \mid \emptyset) \\ \underbrace{\bar{g}_Y(j) = f(j \mid V \setminus \{j\})} & & \underbrace{\bar{g}_Y(j) = f(j \mid \emptyset)} & \\ \text{for } j \in Y & & \text{for } j \notin Y. & \end{aligned} \tag{1.11}$$

We use these gradients to provide scalable optimization algorithms for all the optimization problems considered in this thesis.

1.4.2 Continuous extensions of submodular functions:

We investigate the Lovász extension [Lovász, 1983] and the multi-linear extension [Calinescu et al., 2011] as surrogates for many submodular optimization problems.

Convex relaxations: The Lovász extension [Lovász, 1983] reveals a connection between submodularity and convexity. Any vector $y \in [0,1]^n$, defines a permutation σ_y induced by ordering its elements in non-increasing order, and correspondingly a chain of sets $S_0^{\sigma_y} \subseteq \ldots \subseteq S_n^{\sigma_y}$, with $S_j^{\sigma_y} = \{\sigma_y(1), \cdots, \sigma_y(j)\}$ for $j \in \{1, 2, \ldots, n\}$. The Lovász extension $\breve{f}$ of f is defined [Lovász, 1983] as:

$$\breve{f}(y) = \sum_{j=1}^{n} y[\sigma_y(j)](f(S_j^{\sigma_y}) - f(S_{j-1}^{\sigma_y})) \tag{1.12}$$

The Lovász extension is convex if and only if f is submodular. Even though σ_y might not be unique (there might be many orderings if y has duplicate entries) the Lovász extension is unique. Since it agrees with f on the vertices of the hypercube, $f(X) = \breve{f}(1_X), \forall X \subseteq V$[1], $\breve{f}$ is a natural convex extension of a submodular function. The Lovász extension is a non-smooth (piece-wise linear) convex function for which a subgradient $h^f_{\sigma_y}$ can be computed efficiently for each $y \in [0,1]^n$ via Edmonds's greedy algorithm [Edmonds, 1970]:

$$h^f_{\sigma_y}(\sigma_y(j)) = f(S_j^{\sigma_y}) - f(S_{j-1}^{\sigma_y}), \forall j \in \{1, 2, \cdots, n\} \tag{1.13}$$

It is not hard to see then that, $\breve{f}(w) = \langle h^f_{\sigma_w}, w \rangle$ is a piece-wise linear function.

Multilinear relaxations: For maximization problems, a commonly used continuous relaxation is the multilinear extension, which can be defined for any set function f as:

$$\tilde{f}(x) = \sum_{X \subseteq V} f(X) \prod_{i \in X} x_i \prod_{i \notin X} (1 - x_i). \tag{1.14}$$

The multilinear extension has particularly nice properties when the set function f is submodular. In particular, $\frac{\partial \tilde{f}}{\partial x_i} \geq 0$ iff f is monotone and $\frac{\partial^2 \tilde{f}}{\partial x_i \partial x_j} \leq 0$ iff f is submodular. This implies

[1] where 1_X is the characteristic vector of X, i.e., $1_X(j) = I(j \in X)$

that for a non-decreasing set function, $\tilde{f}$ is increasing along any positive direction, and for a submodular function, $\tilde{f}$ is concave along any non-negative direction. The gradient of the multilinear extension can be obtained as:

$$\nabla_j \tilde{f}(x) = \partial \tilde{f}/\partial x_j = \tilde{f}(x \vee e_j) - \tilde{f}(x \vee e_j - e_j). \tag{1.15}$$

where e_j satisfies $e_j(j) = 1, e_j(i) = 0, i \neq j$, and $\{x \vee y\}(i) = \max(x(i), y(i))$. This gradient can be computed in $O(n)$ evaluations of $\tilde{f}$.

Concave relaxations: Submodular functions also admit a concave relaxation, which can be defined in the following manner. Instead of taking a particular distribution (as in the case of the multilinear extension), define a continuous extension as the supremum over all valid distributions,

$$\hat{f}(x) = \max \Big\{ \sum_{X \subseteq V} p(X) f(X),\ p \in \Delta_x \Big\} \tag{1.16}$$

where $\Delta_x = \{p \in [0,1]^{2^V} : \sum_X p_X = 1, \forall j \in V, \sum_{X: j \in X} p_X = x_j\}$. Note that the multilinear extension is defined via a specific choice of $p_X = \prod_{i \in X} x_i \prod_{j \notin X} (1 - x_j)$. The resulting function $\hat{f}(x)$ is concave and a valid continuous extension, and hence a concave extension of f. Unfortunately, this extension is NP-hard to evaluate [Vondrák, 2007].

1.5 Structural Properties of Submodular Function

Since many of the worst case guarantees are often much more pessimistic compared to the ones we observe in practice, it means that these lower bounds are specific to rather contrived classes of functions, whereas much better results can be achieved in many practically relevant cases. Given the increasing importance of submodular functions in machine learning, these observations beg the question of qualifying and quantifying properties that make sub-classes of submodular functions more amenable to learning and optimization. In this thesis, we take additional steps towards solving these problems, by investigating several theoretical characterizations and parameters, which provide better connections between theory and practice - the curvature, monotonicity ratio and the submodularity ratio.

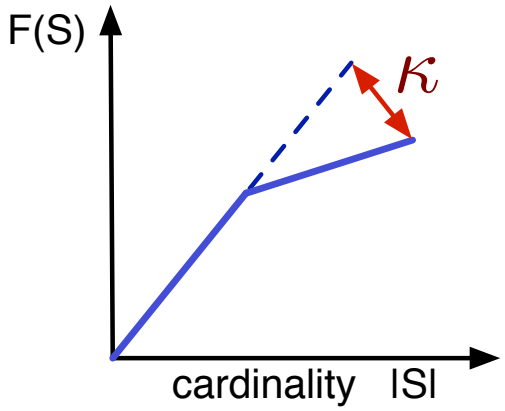

Figure 1.4: Illustrating Curvature

1.5.1 Curvature

This construct concerns monotone submodular functions. While this class of functions often admits better guarantees compared to the general class of submodular functions, the worst case bounds for many problems are still quite bad (for example, polynomial factors of $O(\sqrt{n})$ in certain cases). We refine these bounds via a generic notion of *curvature* [Conforti and Cornuejols, 1984, Vondrák, 2010, Iyer et al., 2013a] – the deviation from modularity – of a monotone submodular function. This quantity provides improved bounds for both minimization and maximization problems. Though the concept of curvature has been used to tighten bounds for submodular maximization problems [Conforti and Cornuejols, 1984, Vondrák, 2010], its effect on minimization problems has been open. Our results complete a unifying picture of the effect of curvature on submodular problems. Moreover, curvature is still a fairly generic concept, as it only depends on the marginal gains of the submodular function, while simultaneously allowing for a smooth transition between the 'easy' functions and the 'really hard' subclasses of monotone submodular functions.

Without loss of generality, assume that $f(j) > 0$ for all $j \in V$. This follows since, if there exists an element $j \in V$ such that $f(j) = 0$, we can safely remove element j from the ground

set, since for every set X, $f(j|X) = 0$ (from submodularity), and including or excluding j does not make any difference to the cost function. We define a number of variants of the curvature.

$$\kappa_f = 1 - \min_{j \in V} \frac{f(j \mid V \setminus i)}{f(j)}, \;\; \kappa_f(S) = 1 - \min_{j \in S} \frac{f(j|S \backslash j)}{f(j)}, \tag{1.17}$$

The first notion of curvature κ_f is the worst case curvature of a set function, and correspondingly is the weakest notion. This notion of curvature has been used in several papers on submodular optimization [Conforti and Cornuejols, 1984, Vondrák, 2010, Iyer et al., 2013a]. The curvature $\kappa_f(S)$ is slightly stronger, and is defined with respect to the set S. We also define two additional notions of curvature at a set S.

$$\hat{\kappa_f}(S) = 1 - \frac{\sum_{j \in S} f(j|S \backslash j)}{\sum_{j \in S} f(j)} \;\; \tilde{\kappa}_f(S) = 1 - \min_{T \subseteq V} \frac{f(T|S) + \sum_{j \in S \cap T} f(j|S \cup T \backslash j)}{f(T)} \tag{1.18}$$

Rather than the minimum used in $\kappa_f(S)$, if we take the average, we obtain the average curvature $\hat{\kappa_f}(S)$. Finally, $\tilde{\kappa}_f(S)$ is a version of curvature defined in [Vondrák, 2010]. These different forms of curvature are closely related.

Proposition 1.2. *For any monotone submodular function and set $S \subseteq V$,*

$$0 \leq \hat{\kappa_f}(S) \leq \kappa_f(S) \leq \tilde{\kappa}_f(S) \leq \kappa_f(V) = \kappa_f \leq 1 \tag{1.19}$$

Note that $\hat{\kappa_f}(S)$ is the tightest notion of curvature, followed by $\kappa_f(S)$, and then $\tilde{\kappa}_f(S)$, and finally κ_f which is the worst case, and weakest notion. These different notions of curvature appear in a number of bounds related to various submodular optimization problems. A modular function has curvature $\kappa_f = 0$, and a matroid rank function has maximal curvature $\kappa_f = 1$.

Proof. It is easy to see that all these notions of curvature are between zero and one. To show

that $\kappa_f(S) \leq \tilde{\kappa}_f(S)$, note that,

$$\begin{aligned}
1 - \tilde{\kappa}_f(S) &= \min_{T \subseteq V} \frac{f(T|S) + \sum_{j \in S \cap T} f(j|S \cup T \backslash j)}{f(T)} \\
&\leq \min_{T \subseteq V : |T| = 1} \frac{f(T|S) + \sum_{j \in S \cap T} f(j|S \cup T \backslash j)}{f(T)} \\
&\leq \min\{\min_{j \in S} \frac{f(j|S \backslash j)}{f(j)}, \min_{j \notin S} \frac{f(j|S)}{f(j)}\} \\
&\leq \min_{j \in S} \frac{f(j|S \backslash j)}{f(j)} \\
&\geq 1 - \kappa_f(S)
\end{aligned}$$

We then prove that $\hat{\kappa_f}(S) \leq \kappa_f(S)$. Note that,

$$\begin{aligned}
1 - \kappa_f(S) &= \min_{j \in S} \frac{f(j|S \backslash j)}{f(j)} \\
&\leq \frac{f(j|S \backslash j)}{f(j)}, \forall j \in S
\end{aligned}$$

Also,

$$\begin{aligned}
1 - \hat{\kappa_f}(S) &= \frac{\sum_{j \in S} f(j|S \backslash j)}{\sum_{j \in S} f(j)} \\
&\geq \frac{\sum_{j \in S} (1 - \kappa_f(S)) f(j)}{\sum_{j \in S} f(j)} \\
&\geq 1 - \kappa_f(S)
\end{aligned}$$

Hence, $\hat{\kappa_f}(S) \leq \kappa_f(S)$.

Finally, we show that $\tilde{\kappa}_f(S) \leq \kappa_f$. In order to show this, note that,

$$1 - \hat{\kappa_f}(S) = 1 - \min_{T \subseteq V} \frac{f(T|S) + \sum_{j \in S \cap T} f(j|S \cup T \backslash j)}{f(T)} \tag{1.20}$$

Let $\tilde{T}$ be the set which obtains the minimum above. Hence,

$$1 - \hat{\kappa_f}(S) = 1 - \frac{f(\tilde{T}|S) + \sum_{j \in S \cap \tilde{T}} f(j|S \cup \tilde{T} \backslash j)}{f(\tilde{T})} \tag{1.21}$$

$$\geq \frac{\sum_{j \in \tilde{T} \backslash S} f(j|S \cup \tilde{T} \backslash j) + \sum_{j \in S \cap \tilde{T}} f(j|S \cup \tilde{T} \backslash j)}{f(\tilde{T})} \tag{1.22}$$

$$\geq \frac{\sum_{j \in \tilde{T}} f(j|S \cup \tilde{T} \backslash j)}{f(\tilde{T})} \tag{1.23}$$

$$\geq \frac{\sum_{j \in \tilde{T}} f(j|V \backslash j)}{\sum_{j \in \tilde{T}} f(j))} \tag{1.24}$$

$$\geq \min_{j \in V} \frac{f(j|V \backslash j)}{f(j)} \tag{1.25}$$

$$\geq 1 - \kappa_f \tag{1.26}$$

Hence proved. □

Many practically interesting functions have smaller curvature and (as we show) admit better worst case guarantees, a fact that also has been observed in practice [Iyer et al., 2013b, Lin and Bilmes, 2011c]. An example for such functions with $\kappa_f < 1$, is the class of concave over modular functions, used e.g., in applications for speech [Lin and Bilmes, 2011c], and computer vision [Jegelka and Bilmes, 2011d]. This class comprises, for instance, functions of the form $f(X) = \sum_{i=1}^{k} (w_i(X))^a$, for some $a \in [0, 1]$ and a non-negative weight vectors w_i. Sometimes these functions are defined over clusters, in which case, the weights w_i are restricted to a cluster $C_i \subseteq V$ [Lin and Bilmes, 2011c, Jegelka and Bilmes, 2011d, Iyer and Bilmes, 2012b]. We shall see that this class of functions, as well as many others, can provably be better approximated and minimized than general submodular functions, thanks to the characterization of the *curvature* of a submodular function.

1.5.2 Monotonicity Ratio

We next investigate the class of general (non-monotone) submodular functions. These problems are often harder than the subclass of monotone submodular functions. We define

a notion of approximate monotonicity called the *monotonicity ratio*. This captures the degree of monotonicity of a general submodular function. We show improved bounds both in the context of minimization and maximization, and this notion helps extend many of the algorithms for monotone submodular functions to the non-monotone setting. We can define this as,

$$\nu_f^{k,l} = \max_{S \subseteq T \subseteq V : |S|=k, k \leq |T| \leq l} \frac{f(S)}{f(T)}, \text{for } k \leq l \tag{1.27}$$

The monotonicity ratio is defined with respect to two parameters k and l. Note that for the definition of $\nu_f^{k,l}$ to make sense, $k \leq l$. Moreover, it is easy to see that $\nu_f^{k,k} = 1$, $\nu_f^{k,l} \geq 1$ for general set functions. For a monotone submodular function, $\nu_f^{k,l} = 1$. Interestingly, the definition of the monotonicity ratio does not depend on the submodularity of f, and could potentially extend to non-submodular functions as well. However, many of the guarantees we provide would depend on submodularity of f. Similar to curvature, we shall see that many non-monotone submodular functions have monotonicity ratio $\nu_f^{k,l} \approx 1$, thereby ensuring improved guarantees for many algorithms [Lin and Bilmes, 2010, Krause and Guestrin, 2005a].

1.5.3 Submodularity Ratio

We consider the notion of approximate submodularity called the *submodularity ratio* (defined by [Das and Kempe, 2011], to help extend some of the algorithms known to hold for submodular functions, to more general class of non-submodular and *approximately* submodular functions. The *submodularity ratio* is defined as:

$$\gamma_f^{U,k} = \min_{L \subseteq U, S : |S| \leq k, S \cap L = \emptyset} \frac{\sum_{x \in S} f(x|L)}{f(S|L)} \tag{1.28}$$

The submodularity ratio is defined with respect to a set $U \subseteq V$ and a parameter $k \geq 1$. Note that f is submodular if and only if $\gamma_f^{U,k} = 1$ for all sets U and $k \geq 1$. Moreover, in general $\gamma_f^{U,k} \leq 1$. This parameter shows the decay of approximation bounds when an algorithm for submodular maximization is applied to non-submodular functions. The submodularity ratio measures how "close" f is to submodularity, and helps characterize theoretical bounds for functions which are approximately submodular [Das and Kempe, 2011].

1.6 Characterizing Subclasses of Submodular Functions

In this dissertation, we shall consider several important subclasses of submodular functions in applications. From a theoretical perspective, such a characterization is important, a large number of optimization problems are hard when restricted to the general family of submodular functions. Moreover, these are the classes of functions which appear naturally in real world applications. Below, we define several subclasses of submodular functions, which naturally occur in practice.

1.6.1 Weighted Matroid Rank functions:

A common class of submodular functions are sums of weighted matroid rank functions, defined as,

$$f(X) = \sum_i \max\{w_i(A) | A \subseteq X, A \in \mathcal{I}_i\}, \tag{1.29}$$

for linear weights $w_i(j)$. These functions form a rich class of coverage functions for summarization tasks [Lin, 2012]. One special case of this function is the *facility location* function [Nemhauser et al., 1978, Simon et al., 2007, Tschiatschek et al., 2014]:

$$f(X) = \sum_{i \in V} \max_{j \in X} s_{ij}, \tag{1.30}$$

for pairwise similarities s_{ij}, with a rank 1 uniform matroid. We can generalize it to a rank k uniform matroid, as well as other matroidal structures like the graphic matroid, partition matroid etc.

1.6.2 Set Covers and Related Functions

Set Covers: Given a set of sets $\{\mathcal{S}_1, \cdots, S_n\}$ and the universe $\mathcal{U} = \cup_i \mathcal{S}_i$, let,

$$f(X) = w(\cup_{i \in X} \mathcal{S}_i), \tag{1.31}$$

where w_j denotes the weight of item $j \in \mathcal{U}$. This setup can alternatively be expressed via a neighborhood function $\Gamma : 2^V \rightarrow 2^{\mathcal{U}}$ such that $\Gamma(X) = \cup_{i \in X} \mathcal{S}_i$. Then,

$$f(X) = w(\Gamma(X)). \tag{1.32}$$

This function also occurs in many applications in submodular minimization and maximization [Bach, 2013, Lin, 2012].

Bipartite Neighborhoods: This is a class of functions closely related to set covers. Given a bipartite graph, the neighborhood functions are most generally defined as,

$$f(X) = \psi(w(\Gamma(X))), \tag{1.33}$$

where $\Gamma(.)$ refers to the neighbors of the set X in the graph, $\psi(.)$ is a concave function, and w is a weight vector. When $\psi(x) = x$, we get back the set cover function. These functions have been used in many data subset selection problems, and in particular, the problem of finding a limited vocabulary speech corpus [Lin and Bilmes, 2011a].

Probabilistic Coverage Functions: This is a generalization of the set cover function, which has been used in a number of models for summarization problems [El-Arini et al., 2009]. This provides a probabilistic notion to the set cover function, and is defined as,

$$f(X) = \sum_{i \in \mathcal{U}} w_i [1 - \prod_{j \in X} (1 - p_{ij})] \tag{1.34}$$

where $\mathcal{U}$ is the universe (as in the set cover case). We get back the set cover function, if $p_{ij} = I(i \in S_j)$

1.6.3 Graph Based Submodular Functions

Graph Cut related functions Graph Cuts are widely used, and may be written as,

$$f(X) = \sum_{i \in X, j \notin X} s_{ij}. \tag{1.35}$$

This general class of functions can be written as [Lin and Bilmes, 2011c, Lin, 2012],

$$f(X) = \sum_{i \in V, j \in X} s_{ij} - \lambda \sum_{i,j \in X} s_{ij} \tag{1.36}$$

With $\lambda = 1$, we get back the standard graph cut function. For $\lambda < 0.5$, $f(X)$ is monotone non-decreasing submodular function.

Truncations: These functions are special cases of concave over modular functions, and are defined as,

$$f(X) = \min\{w(X), \alpha\} \tag{1.37}$$

A special case of this is the truncated graph-cut function. This function is defined as,

$$f(X) = \sum_{i \in V} \min\{\sum_{j \in X} s_{ij}, \alpha_i\} \tag{1.38}$$

These functions have found a lot of applicability in modeling coverage and diversity [Lin and Bilmes, 2011c]. Moreover, a number of models in probabilistic inference for classification and segmentation problems can be modeled via truncations [Stobbe and Krause, 2010].

1.6.4 Sparse Pseudo-Boolean functions.

For graphical models, in particular in computer vision, set functions are often written as polynomials [Ishikawa, 2009]. Any set function can be written as a polynomial,

$$p_f(x) = \sum_{T \subseteq V} \alpha_T \prod_{i \in T} x_i, \tag{1.39}$$

where $x \in \{0,1\}^n$ is the indicator vector of a set. In other words, $f(S) = \sum_{T \subseteq S} \alpha_T$. Submodular functions are a subclass of these polynomials. We are particularly interested in functions which are sparse, i.e have a few non-zero values of α_T. This class of functions also subsumes graph cut like functions above and the functions considered in [Stobbe and Krause, 2012, Ishikawa, 2009].

1.6.5 Spectral functions:

Diversity can also be encouraged via spectral regularizers [Das et al., 2012]. Given a positive definite matrix $S \in \mathbb{R}^{n \times n}$, define S_X to be the $|X| \times |X|$ sub-matrix of the rows and columns indexed by X. Any scalar function ψ whose derivative is operator-antitone defines a submodular function, by applying it to the eigenvalues of S_X [Friedland and Gaubert, 2011]

$$f(X) = \sum_{i=1}^{|X|} \psi(\lambda_i(S_X)). \tag{1.40}$$

The resulting class of submodular functions includes the log determinants occurring in DPP inference [Gillenwater et al., 2012], and, more generally, a smoothed log-determinant function Kelmans and Kimelfeld [1983],

$$f(X) = \log\det(S_X + \delta I_X) = \sum_{i=1}^{|X|} \log(\lambda_i(S_X) + \delta). \tag{1.41}$$

The smoothed function above is monotone for $\delta \geq 1$. Other special cases of this class of functions are, $f(X) = -\sum_{i=1}^{|X|}(\lambda_i(S_X) - 1)^2$ and $f(X) = \sum_{i=1}^{|X|}[\lambda_i(S_X)]^a]$, for $a \in (0, 1]$ [Das et al., 2012].

1.6.6 Concave over modular functions:

Another useful class of functions is the class of concave over modular functions:

$$f(X) = \sum_i \psi_i(m_i(X)). \tag{1.42}$$

where the functions ψ_is are concave functions. The saturated coverage function discussed above is also a special case of this class of functions. A particular interesting subclass of these are

$$f(X) = \sum_i [w_i(X)]^a, \tag{1.43}$$

where w_i are modular functions and $a \in (0, 1]$. These type of functions have extensively been used in numerous applications [Jegelka and Bilmes, 2011b, Iyer and Bilmes, 2012b, Lin and

Bilmes, 2011c, Lin, 2012]. One can also define sums of concave over modular functions. In particular, the class of clustered sums of concave over modular functions are,

$$f(X) = \sum_{j=1}^{M} \psi(w_j(X \cap C_M)), \tag{1.44}$$

where $C_1, C_2, \cdots, C_M$ are clusters of similar items in the ground set V. This function simultaneously captures diversity in maximization problems [Lin and Bilmes, 2011c], and notions of cooperation in minimization problems [Jegelka and Bilmes, 2011d, Iyer and Bilmes, 2012b]. Another instantiation of sums of concave over modular functions are feature based functions, defined as

$$f(X) = \sum_{e \in \mathcal{F}} \psi(w_e(X)) \tag{1.45}$$

where $|\mathcal{F}| = M$, and $w_e(j)$ captures how much item j covers feature $e \in \mathcal{F}$. These functions have been used in data subset selection applications [Wei et al., 2014b, Kirchhoff and Bilmes, 2014].

1.6.7 Entropy and Mutual Information:

The natural information covering function ,entropy is submodular, when seen as a set function,

$$f(A) = H(X_A), \tag{1.46}$$

where X_A stands for the subset of random variables indexed by the set A. Similarly mutual information,

$$f(A) = I(X_A; X_{V \setminus A}) \tag{1.47}$$

is symmetric submodular. Moreover, under a conditional independence model, the mutual information,

$$f(A) = I(X_A; C) \tag{1.48}$$

is also submodular. These occur naturally in many applications involving sensor placement and feature selection [Krause and Guestrin, 2005b, Krause et al., 2008b, Iyer and Bilmes, 2012b].

1.7 Previous Work on Submodular Optimization

The search for optimal algorithms for submodular optimization has seen substantial progress in recent years, but is still an ongoing endeavor. Below is a summary on the recent progress on these problems.

Submodular Minimization (Problem 1): Unconstrained submodular minimization, (when $\mathcal{C} = 2^V$), is polynomial time, and the first poly-time algorithm used the ellipsoid method [Grötschel et al., 1981, Grotschel et al., 1984], and several combinatorial algorithms followed [Iwata et al., 2001, Iwata, 2002, Fleischer and Iwata, 2003, Iwata, 2003, Iwata and Orlin, 2009, Orlin, 2009]. For a detailed summary, see [McCormick, 2005]. The best complexity to date is $O(n^5\gamma + n^6)$ [Orlin, 2009] (γ is the cost of evaluating f). An alternative strategy is to consider continuous relaxations strategies for submodular minimization. Examples of these include the Minimum Norm Point (MNP) algorithm [Fujishige and Isotani, 2011, Frank and Wolfe, 1956], which still has a worst case running time of $\tilde{O}(n^5\gamma + n^7)$ [Chakrabarty et al., 2014]. Another class of continuous algorithms uses the convex relaxation of the submodular function [Bach, 2013], but in practice, have been observed to work slower than MNP [Bach, 2013]. This has motivated studies on faster, possibly special case or approximate methods [Stobbe and Krause, 2010, Jegelka et al., 2011, Bach, 2013]. Unlike unconstrained submodular minimization, constrained forms of submodular minimization are very hard – even with simple constraints such as a cardinality lower bound, the problems are NP-hard, and not approximable better than a polynomial factor of $\Omega(\sqrt{n})$ [Svitkina and Fleischer, 2008]. Similarly the problems of minimizing a submodular function under covering constraints [Iwata and Nagano, 2009], spanning trees, perfect matchings, paths [Goel et al., 2009] and cuts [Jegelka and Bilmes, 2011c] have very similar hardness factors. In all of these cases, matching upper bounds (i.e approximation algorithms) have been provided [Svitkina and Fleischer, 2008, Iwata and Nagano, 2009, Goel et al., 2009, Jegelka and Bilmes, 2011c]. Algorithms for these problems in literature are based on two basic paradigms: they are either

combinatorial algorithms using surrogate submodular functions, or they rely relaxing the discrete problem to a continuous version followed by rounding.

Submodular Maximization (Problem 2): Submodular maximization has been extensively studied, mainly in the past couple of years. The first set of results for submodular maximization were shown in [Nemhauser et al., 1978, Nemhauser and Wolsey, 1978], where they provide a $1 - 1/e$ approximation algorithm (in the form of a simple greedy heuristic) for maximizing a monotone submodular function under a cardinality constraint. Variants of the greedy algorithm were further extended to handle matroid and knapsack constraints [Fisher et al., 1978, Sviridenko, 2004, Krause and Guestrin, 2005c, Lin and Bilmes, 2010]. The factor $1 - 1/e$ was also shown to be optimal under the value oracle model [Feige, 1998, Nemhauser and Wolsey, 1978]. The first systematic study for non-monotone submodular function maximization was performed by Fiege *et al* [Feige et al., 2011b], where they obtain a $1/3$ and a randomized $2/5$ approximation for unconstrained submodular maximization. They also show an absolute hardness of $1/2$ for this problem. They raise an open question, whether there exists a tight $1/2$ approximation algorithm for this problem. This question was resolved in [Buchbinder et al., 2012], where they show that a simple randomized linear time algorithm achieves an approximation factor of $1/2$. Subsequently, many of these results were extended to matroid and knapsack constraints in [Lee et al., 2009a,b]. Similarly [Buchbinder et al., 2014] propose a near optimal algorithm for non-monotone submodular maximization subject to cardinality constraint. Similar to submodular minimization, the algorithms for submodular maximization are either combinatorial, or based on continuous relaxations. The combinatorial algorithms include greedy algorithms, in the form of forward greedy, randomized greedy, reverse greedy as well as bidirectional greedy variants, and local search techniques. The continuous relaxation algorithms use the multilinear extension, followed by variants of the continuous greedy algorithm [Calinescu et al., 2011, Chekuri et al., 2011, Vondrák, 2007, Buchbinder et al., 2014].

Difference of Submodular (DS) Optimization (Problem 3): This problem was first introduced in [Narasimhan and Bilmes, 2005], where they propose the submodular-supermodular procedure, as a general purpose approximate algorithm for minimizing the difference of submodular functions. [Byrnes, 2009, Kawahara and Washio, 2011] later propose exact algorithms for this problem, which in the worst case is exponential. Subsequently, a number of papers have investigated special cases of DS optimization [Kolmogorov and Rother, 2007, Borodin et al., 2012].

Submodular Optimization subject to Submodular Constraints – SCSC/SCSK (Problems 4 and 5): This is a new class of problems we investigate in this thesis. A number of special cases of this problem have been studied in the past, including cardinality constrained submodular minimization, submodular maximization subject to knapsack constraints, and the submodular set cover [Krause and Guestrin, 2005b, Svitkina and Fleischer, 2008, Sviridenko, 2004, Wolsey, 1982].

1.8 Our Contributions

Though there has been phenomenal work from the algorithms and theory communities in addressing the optimization problems above, there are still some issues in large scale application of these techniques in machine learning. The following are our main contributions:

- **Scalability:** A potential stumbling block is that machine learning problems are often large (e.g., "big data") and are getting larger. For general unconstrained submodular minimization, the computational complexity often scales as a high-order polynomial. These algorithms are designed to solve the most general case and the worst-case instances are often contrived and unrealistic. Typical-case instances are much more benign, so simpler algorithms (e.g., graph-cut) might suffice. In the constrained case, however, the problems often become NP-complete, thus requiring (sometimes complicated) approximation algorithms. Hence, there is an urgent need for efficient, practical, and

scalable algorithms for the aforementioned problems if submodularity is to have a lasting impact on the field of machine learning. In this thesis, we address this issue, by proposing a scalable class of algorithms, which work on large scale machine learning problems.

- **Unifying Algorithms:** Given the large class of potential algorithms for different classes of optimization problems, a natural question is which algorithm to use when? Moreover, several algorithms, are specific to the given constraint at hand. Similarly algorithms for submodular maximization are very different in nature from their submodular minimization cohorts. On a whole, there is no unifying algorithmic framework for these problems. In this dissertation, we provide a unifying class of algorithmic techniques which work for Submodular Minimization, Submodular Maximization, Difference of Submodular Optimization, and Submodular Optimization subject to Submodular Constraints. In several cases, we show how many of the algorithms studied in the past, can be viewed within these frameworks. We also provide new algorithms for several of these optimization problems.

- **Theoretical Guarantees and Hardness:** An important component of algorithms, and approximation algorithms is to characterize the goodness of the algorithms. Consider a general minimization and maximization problem of a set function s,

$$\text{Minimization: } X^* = \operatorname*{argmin}_{X \in \mathcal{C}} s(X), \quad \text{Maximization: } X^* = \operatorname*{argmax}_{X \in \mathcal{C}} s(X) \tag{1.49}$$

All the problems we consider in this thesis (Problems 1-5) can be represented by either one of the above two formulations. Either an algorithm is exact, i.e it can find the optimal solution, or it will find an approximate solution. We measure the performance of an approximate solution by the *worst case approximation factor* [Vazirani, 2004]: For minimization problems, we say that an algorithm admits an approximation factor $\alpha \geq 1$, if it can find a set $\hat{X}$ such that,

$$s(X^*) \leq s(\hat{X}) \leq \alpha s(X^*) \tag{1.50}$$

Similarly for maximization problems, an algorithm admits an approximation factor $\beta \leq 1$, if it can find a set $\hat{X}$ such that,

$$\beta s(X^*) \leq s(\hat{X}) \leq s(X^*) \tag{1.51}$$

An optimization algorithm is polynomial time, if its running time can be upper bounded by a polynomial function of the input (in this case, n). In combinatorial optimization problems, the running time also often depends on the complexity of function evaluations, which is assumed to be given via an oracle [Fujishige, 2005]. Another important aspect of optimization problems is the underlying hardness of the optimization problem. One such way to quantify the hardness is *NP-hardness*, which informally means that it is very unlikely that there exists a polynomial time algorithm for the given problem, unless P=NP. A stricter notion of hardness is hardness of approximation. An optimization problem has a hardness factor of α (equivalently β), if no polynomial time algorithm can achieve an approximation factor better than α (equivalently β). In this dissertation, we address the worst case approximation factors of our algorithms, along with the underlying hardness of the problems.

- **Improving the worst case results – Connecting theory and practice:** Unfortunately, several optimization problems considered above, are not only NP hard, but as we shall see, do not even admit constant or logarithmic approximation factors in polynomial time [Goel et al., 2010, Goemans et al., 2009, Iwata and Nagano, 2009, Svitkina and Fleischer, 2008, Jegelka and Bilmes, 2011c, Iyer and Bilmes, 2013]. In fact, most of the bounds are polynomial factors of $\Omega(\sqrt{n})$ and $\Omega(n)$. These rather pessimistic results however stand in sharp contrast to empirical observations seen in applications, thus begging the need for better theoretical characterizations and connections between theory and practice. In this dissertation, we explore subclasses of submodular functions, as well as structural properties of submodular functions (like curvature, monotonicity ratio, submodularity ratio etc.), with the aim of providing improved theoretical worst case guarantees, and also improving the complexity of the optimization algorithms.

1.9 Main Algorithmic Ideas in this dissertation

Following are the main algorithmic ideas we use throughout this dissertation. We explore two classes of algorithms, which we use for the optimization problems above:

- Combinatorial Algorithms
- Continuous Relaxation based Algorithms.

In the paragraphs below, we argue the significance of these classes of algorithms in the light of the issues pointed out above.

1.9.1 Combinatorial Algorithms

We investigate two classes of combinatorial algorithms.

Majorization-Minimization based Semigradient Algorithms We provide a unifying majorization-minimization and minorization-maximization framework for all the submodular optimization problems described above (viz. Problems 1 - 5). The resulting algorithms, which repeatedly compute and then efficiently optimize submodular semigradients, offer new and generalize many old methods for submodular optimization. Our approach, takes steps towards providing a unifying paradigm applicable to both submodular minimization and maximization (i.e Problems 1 and 2), which historically have been treated quite distinctly. Moreover, we show how these ideas can further be extended to Problems 3-5, thus providing a unifying and general algorithmic framework. Moreover, our algorithms are extremely scalable, and easy to implement, and the practicality of our algorithms is important since interest in submodularity, owing to its natural and wide applicability, has recently been in ascendance within machine learning. We analyze theoretical properties of our algorithms for these optimization problems, and also argue about their scalability. Majorization-minimization and Minorization-Maximization algorithms are known to be useful in machine learning [Hunter and Lange, 2004]. Notable examples include the EM algorithm [McLachlan and Krishnan,

1997] and the convex-concave procedure [Yuille and Rangarajan, 2002]. This thesis provides a new class of MM algorithms for many general submodular optimization problems.

Algorithms based on Approximations and Surrogate Functions The second class of algorithms are based on using approximations or surrogate submodular function. Given a submodular function f, let $\hat{f}$ be a surrogate submodular function such that,

$$\hat{f}(X) \leq f(X) \leq \alpha \hat{f}(X), \forall X \subseteq V \tag{1.52}$$

The idea is then to use $\hat{f}$ as a surrogate function instead of f, in the optimization problems we study in this thesis. We show that we can transfer this multiplicative factor into the approximation guarantee for the corresponding problem. In this thesis, we consider several approximations, and show how we can obtain different approximation factors depending on the tightness of the approximation.

1.9.2 Continuous Relaxation based Algorithms

This class of algorithms uses continuous relaxations of a submodular function. Denote the continuous relaxation of f as $\bar{f}$. The continuous relaxation $\bar{f}$ of f is a continuous function $\bar{f} : [0,1]^n \Rightarrow \mathbf{R}$ such that $\bar{f}(1_X) = f(X)$, where 1_X is the indicator function of set X (i.e $1_X \in \mathbf{R}^n : 1_X(j) = 1, j \in X$ and $1_X(j) = 0, j \notin X$. Relaxation approaches for submodular optimization follow a two-stage procedure:

1. Find the optimal (or approximate) solution $\hat{x}$ to the relaxed version of the corresponding optimization problem.

2. Round the continuous solution $\hat{x}$ to obtain the discrete indicator vector of set $\hat{X}$.

We show that with the right choice of the continuous relaxation and rounding scheme, we can obtain bounded approximation factors, for many of the optimization problems we study in this thesis.

1.10 Road-Map of this Thesis

The road-map of this thesis is as follows. Chapters 3, 4, 5, 6 and 7 provide algorithms for the optimization problems in this thesis. Chapters 3, 4, 5, 6 deal with combinatorial algorithms for Problems 1-5, while Chapter 7 studies the continuous relaxation based algorithms for these problems. Following is the chapter-wise summary.

- In Chapter 2, we investigate important properties of submodular functions connected to convexity and concavity. In particular, we investigate polyhedra connected with submodular functions, and also study the sub and super-differentials of a submodular function in detail. We see how the polyhedral properties of submodular functions are connected with the continuous extensions of a submodular function, namely, the convex, concave and the multilinear extension. Several of these properties shall play an important role in the algorithms we study in this thesis.

- Chapters 3 and 4 investigate combinatorial algorithms for submodular minimization and maximization (Problems 1 and 2). This includes the majorization-minimization based semidifferential based algorithms, and algorithms based on using surrogate submodular functions. Majorization-Minimization and Minorization-Maximization based algorithms for submodular minimization and maximization respectively, provide a unifying framework for these optimization problems. In the case of minimization (Chapter 3), we provide a new class of algorithms which are extremely efficient and practical. We also investigate algorithms based on choosing surrogate (or proxy submodular functions), and show how by choosing tight approximations, we can obtain theoretically the tightest approximation guarantees. We perform several experiments on real and synthetic data, to validate our algorithms. In the case of maximization (Chapter 4), we demonstrate that many algorithms for submodular maximization, including several greedy and local search schemes, may be viewed as special cases of a generic minorize-maximize framework that relies on discrete semidifferentials.

- In Chapter 5, we investigate combinatorial algorithms for difference of submodular (DS) optimization (Problem 3). In particular, we propose Majorization-Minimization (MM) based algorithms, which repeatedly use the submodular semigradients (as modular upper and lower bounds). This framework subsumes the Submodular-Supermodular procedure Narasimhan and Bilmes [2005], an algorithm inspired by the convex-concave procedure [Yuille and Rangarajan, 2002]. Along with the submodular-supermodular procedure, we propose two new algorithms, the supermodular-submodular and the modular-modular procedures. An important aspect of our algorithms (particularly, the latter two algorithms) is that they scale very well to large scale problems. We investigate the hardness of this problem, and show that it is not only NP hard, but also inapproximable – thus justifying the need for heuristic methods to solve this problem. We empirically show the validity of our methods on the problem of feature subset selection.

- In Chapter 6, we propose combinatorial algorithms for submodular optimization subject to submodular constraints – SCSC and SCSK (Problems 4 and 5). We show that Problems 4 and 5 are intimately connected, in that any approximation algorithm for either problem can be used to provide guarantees for the other problem as well. We then provide a Majorization-Minimization (MM) algorithmic framework, which is not only scalable, but unifies several algorithms considered in the past for special cases of SCSC and SCSK [Sviridenko, 2004, Wolsey, 1982]. We investigate the hardness of these problems, and provide a tight approximation algorithm, which matches the hardness this up to log factors. The tight algorithms use surrogate submodular functions. Finally, we empirically validate our algorithms on two real world applications, involving data subset selection, and sensor placement.

- Finally, in Chapter 7 we develop a continuous relaxation methodology for Problems 1-3. For submodular minimization (Problem 1), we use the Lovász extension, while for submodular maximization (Problem 2), we use the multilinear extension and the concave

extension. Similar to majorization-minimization scheme, we show that this provides a unifying algorithmic framework for both these problems. The relaxation viewpoint, moreover, complements the approximation guarantees obtained by the combinatorial algorithms. We also show that this technique can also be extended to provide heuristics for Problem 3 (i.e DS optimization).

Chapter 2 is being prepared for submission. Chapter 3 and 4 was originally presented in two conference papers:

- R. Iyer, S. Jegelka, and J. Bilmes. Fast Semidifferential based Submodular function optimization. In *ICML*, 2013b

- R. Iyer, S. Jegelka, and J. Bilmes. Curvature and Optimal Algorithms for Learning and Minimizing Submodular Functions . In *NIPS*, 2013a

The former was also published as a workshop paper:

- R. Iyer, S. Jegelka, and J. Bilmes. Mirror descent like algorithms for submodular optimization. *NIPS Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2012

Chapter 5 was originally published in a conference paper:

- R. Iyer and J. Bilmes. Algorithms for approximate minimization of the difference between submodular functions, with applications. *In UAI*, 2012b

Chapter 6 was originally published in the following conference paper:

- R. Iyer and J. Bilmes. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *NIPS*, 2013

Chapter 7 was published in a conference paper:

- R. Iyer, S. Jegelka, and J. Bilmes. Monotone Closure of Relaxed Constraints in Submodular Optimization: Connections Between Minimization and Maximization. In *UAI*, 2014

Chapter 2

SUBMODULARITY, CONVEXITY AND CONCAVITY

2.1 Introduction

Submodular functions have been strongly associated with convex functions, to the extent that submodularity is sometimes regarded as a discrete analogue of convexity [Fujishige, 2005]. This relationship is evident by the fact that submodular function minimization is easy, in that there exist strongly polynomial time algorithms which achieve it. This is akin to convex minimization which is also easy. A number of recent results, however, make this relationship much more formal. For example, similar to convex functions, submodular functions have tight modular lower bounds and admit a sub-differential characterization [Fujishige, 1984a]. Moreover, it is possible [Fujishige, 1984c] to provide optimality conditions, in a manner analogous to the Karush-Kuhn-Tucker (KKT) conditions from convex programming, for submodular function minimization. Furthermore, the Fenchel duality theorem and the discrete separation theorem, both of which are known to hold for convex functions have been shown to go through for submodular functions [Fujishige, 1984c, Frank, 1982]. In addition, submodular functions also admit a natural convex extension, known as the Lovász extension, which is easy to evaluate [Lovász, 1983] and optimize. The Lovász extension, moreover, also has no integrality gap and minimizing a submodular function is equivalent to minimizing its Lovász extension. All these results show that submodularity is indeed extremely closely related to convexity, and seem to verify the claim that submodularity is the discrete analog of convexity.

Submodular functions also have some properties, which are unlike convexity, but perhaps more akin to concavity. Submodular function maximization is known to be NP hard. However, there exist a number of constant factor approximation algorithms based on simple greedy or local search heuristics [Feige et al., 2011b, Lee et al., 2009a, Nemhauser et al., 1978] and some

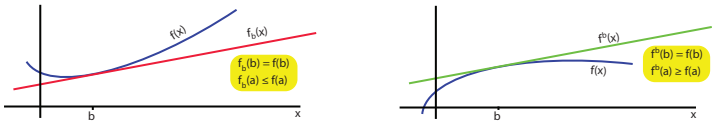

Figure 2.1: Convex and Concave functions with sub and super gradients respectively

recent continuous approximation methods [Chekuri et al., 2011, Feldman et al., 2011]. This is unlike convexity where maximization can be hopelessly difficult [Sahni, 1974]. Furthermore, submodular functions have a diminishing returns property which is akin to concavity, and concave over modular functions are known to be submodular. In addition submodular function has been shown to have tight modular upper bounds [Iyer et al., 2013b, Iyer and Bilmes, 2012a,b, Jegelka and Bilmes, 2011b, Jegelka et al., 2011], and as we show, form superdifferentials and supergradients like concave functions. The multi-linear extension of a submodular function, which has become useful recently [Chekuri et al., 2011] in the context of submodular maximization, is known to be concave when restricted to a particular direction. All these seem to indicate that submodular functions are related to both convexity and concavity, and have some strange properties enabling them to get the best of both classes of functions.

2.1.1 Motivation and Past Work

Since almost four decades, researchers have been investigating several theoretical and algorithmic aspects relating to submodular functions. Most of the bulk of this work [Fujishige, 2005, Edmonds, 1970, Lovász, 1983, Fujishige, 1984a, Frank, 1982, Fujishige, 1984b], has been in relating submodular functions to convexity from a polyhedral perspective, thereby enabling efficient algorithms for submodular minimization. From a polyhedral perspective, Fujishige, Edmonds and others [Fujishige, 2005, Edmonds, 1970, Fujishige, 1984a], provided a characterization of the submodular polyhedron, the base polytope, and sub-differentials of a submodular functions. Thanks to these results, Lovász [Lovász, 1983] provided an efficient characterization of the convex extension of a submodular function, which is called

the Lovász extension. Finally, the connection between submodularity and convexity was made more precise when it was shown [Frank, 1982, Fujishige, 1984b], that the Discrete Separation Theorem, Fenchel duality Theorem, and the Minkowski Sum theorems hold for submodular functions, when seen in analogy to convexity. From an algorithmic perspective, these results have provided several algorithms for submodular function minimization. In particular, [Fujishige, 2005, Bach, 2013] use the submodular polyhedron and the convex extension to provide an exact algorithm for submodular minimization. Similarly, [Iwata et al., 2001, Iwata, 2003] and others have used many of these ideas to provide exact algorithms for submodular minimization.

While submodular functions have been seen to be related to concavity (as discussed above), the polyhedral aspects of submodular functions from a maximization perspective are much lesser understood. Most work here has been in relation to approximation algorithms for submodular maximization [Nemhauser et al., 1978, Nemhauser and Wolsey, 1978, Fisher et al., 1978, Sviridenko, 2004, Krause and Guestrin, 2005c, Lin and Bilmes, 2010, Feige et al., 2011b, Buchbinder et al., 2012]. Polyhedral aspects of submodular maximization and its relation to concavity, has been studied only in a limited context in previous work [Feige et al., 2011b, Calinescu et al., 2007, Vondrák, 2007, Dughmi, 2009, Nemhauser et al., 1978, Iyer and Bilmes, 2012a,b, Jegelka and Bilmes, 2011b, Jegelka et al., 2011]. For example, a recent chain of papers by Jan Vondrák and others [Calinescu et al., 2007, Vondrák, 2007, Dughmi, 2009] investigated concave extensions of a submodular function, which were shown to be NP hard to evaluate [Vondrák, 2007]. Similarly the subgradients and supergradients of a submodular function have inspired a unifying Majorization-Minimization framework for submodular optimization [Narasimhan and Bilmes, 2005, Iyer and Bilmes, 2012a,b, Jegelka and Bilmes, 2011b, Jegelka et al., 2011, Iyer and Bilmes, 2013, Iyer et al., 2013a]. Furthermore, the sub and super differentials have also been used in the context of approximate inference in a class of probability distributions defined via submodular functions (Iyer and Bilmes [2014b], Djolonga and Krause [2014]). However, as far as we know, no formal and unifying characterization has been provided from a polyhedral perspective to relate submodularity

and concavity.

In this work, we attempt to provide a first unifying characterization of the concave aspects of submodular functions from a polyhedral perspective, thereby extending many of the observations made in [Lovász, 1983]. In this effort, we discover a number of interesting connections between these different aspects of submodular functions connecting concavity, and contrast them to known results of submodularity and convexity.

2.1.2 Road-Map of this chapter

In Sections 2.2, 2.3, 2.4 and 2.5, we review the connections between submodularity and convexity. Most of the results in these sections are from [Lovász, 1983, Fujishige, 2005], and in some cases we provide some generalizations. In Section 2.2, we review polyhedral aspects of submodularity and convexity, and investigate the submodular polyhedron, submodular subdifferentials etc. In Section 2.3, we study the convex extensions of a submodular function, while in Section 2.4 we review the optimality conditions of submodular function minimization from a polyhedral perspective. Finally, in Section 2.5, we review the Discrete Separation Theorem, Fenchel Duality Theorem, and Minkowski Sum Theorem, all from the perspective of the convex analogy of submodular functions. In Section 2.6 we investigate the polyhedral aspects of submodularity and concavity, by defining the submodular upper polyhedron, and the submodular superdifferentials. In Section 2.7, we provide a characterization of the concave extension of a submodular function. In Section 2.8, we study the optimality conditions of submodular function maximization from a polyhedral perspective. Finally, in Section 2.9, we provide versions of the Discrete Separation Theorem, Fenchel Duality Theorem and the Minkowski Sum Theorem, from the perspective of concavity of a submodular function.

2.2 Polyhedral aspects of Submodularity and Convexity

Most of the results in this section are covered in [Lovász, 1983, Fujishige, 2005] and the references contained therein, so for more details please refer to these texts. We use this section to review existing work on the polyhedral connections between submodularity and convexity,

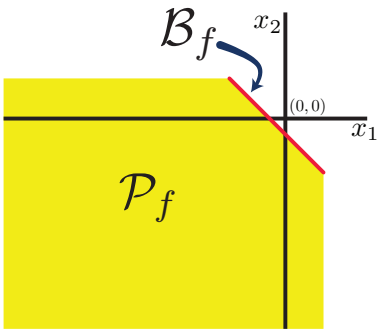

Figure 2.2: The Submodular Polyhedron $\mathcal{P}_f$ and the Base Polytope $\mathcal{B}_f$ in two dimensions

and to help contrast these with the corresponding results on the polyhedral connections between submodularity and concavity starting in Section 2.6.

2.2.1 Submodular (Lower) Polyhedron

For a submodular function f, the (lower) submodular polyhedron[1] and the base polytope of a submodular function [Fujishige, 2005] are defined, respectively, as:

$$\mathcal{P}_f = \{x : x(S) \leq f(S), \forall S \subseteq V\}, \quad \mathcal{B}_f = \mathcal{P}_f \cap \{x : x(V) = f(V)\}. \tag{2.1}$$

where $x(S) = \sum_{i \in S} x_i$. The submodular polyhedron has a number of interesting properties. An important property of the polyhedron is that the extreme points and facets can easily be characterized even though the polyhedron itself is described by a exponential number of inequalities. In fact, surprisingly, every extreme point of the submodular polyhedron is an extreme point of the base polytope. These extreme points admit an interesting characterization in that they can be computed via a simple greedy algorithm — let σ be a permutation of $V = \{1, 2, \cdots, n\}$. Each such permutation defines a chain with elements $S_0^\sigma = \emptyset$, $S_i^\sigma = \{\sigma(1), \sigma(2), \ldots, \sigma(i)\}$ such that $S_0^\sigma \subseteq S_1^\sigma \subseteq \cdots \subseteq S_n^\sigma$. This chain defines an

[1]Since the submodular polyhedron consists of modular lower bounds of a submodular function, we shall also call it the *lower* submodular polyhedron to contrast with the submodular *upper* polyhedron we introduce in Section 2.6.1.

extreme point h^σ of $\mathcal{P}_f$ with entries

$$h^\sigma(\sigma(i)) = f(S_i^\sigma) - f(S_{i-1}^\sigma). \tag{2.2}$$

Each permutation of V characterizes an extreme point of $\mathcal{P}_f$ and all possible extreme points of $\mathcal{P}_f$ can be characterized in this manner [Fujishige, 2005]. Furthermore, the problem $\max_{y \in \mathcal{P}_f} y^\top x$, which is a linear program over a submodular polyhedron, can be very efficiently computed through the greedy algorithm [Edmonds, 1970]. The following lemma gives the greedy algorithm for finding this.

Lemma 2.1. *[Edmonds, 1970, Lovász, 1983] Given a vector $w \in \mathbb{R}_+^n$, consider a permutation σ_w, such that $w[\sigma_w(1)] \geq w[\sigma_w(2)] \geq \cdots \geq w[\sigma_w(n)]$. Define $s^*(\sigma_w(i)) = f(S_i^{\sigma_w}) - f(S_{i-1}^{\sigma_w})$ for $i \in \{1, 2, \cdots, n\}$. Then $\operatorname{argmax}_{s \in P_f} w^\top s = \operatorname{argmax}_{s \in B_f} w^\top s = s^*$. Further $\max_{s \in P_f} w^\top s = \sum_{i=1}^n w(\sigma_w(i))[f(S_i^{\sigma_w}) - f(S_{i-1}^{\sigma_w}]$*

It is easy to see that the optimizers s^* above form extreme points of the submodular polyhedron. Given a submodular function f such that $f(\emptyset) = 0$, the condition that $x \in \mathcal{P}_f$ can be checked in polynomial time for every x. This follows directly from the fact that submodular function minimization is polynomial time.

Proposition 2.1. *Given a submodular function f, checking if $x \in \mathcal{P}_f$ is equivalent to the condition $\min_{X \subseteq V}[f(X) - x(X)] \geq 0$, which can be checked in poly-time.*

2.2.2 The Submodular Subdifferential

Another aspect of the connection between submodular functions and convexity is the submodular subdifferentials [Fujishige, 1984a]. The subdifferential $\partial_f(X)$ of a submodular set function $f : 2^V \to \mathbb{R}$ for a set $Y \subseteq V$ is defined [Fujishige, 1984a, 2005] analogously to the subdifferential of a continuous convex function:

$$\partial_f(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \geq f(X) - x(X) \text{ for all } Y \subseteq V\} \tag{2.3}$$

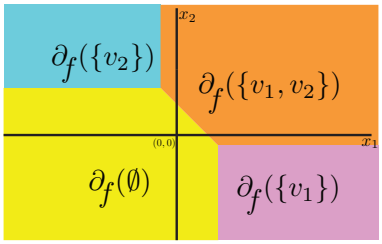

Figure 2.3: The Subdifferentials $\partial_f(Y)$ of a submodular function for different sets Y in two dimensions. Notice that the subdifferentials partition the space $\mathbb{R}^2$. In this case, $V = \{v_1, v_2\}$.

The polyhedra above can be defined for any (not necessarily submodular) set function. When the function is submodular however, it can be characterized efficiently. Firstly, note that for normalized submodular functions, for any $h_X \in \partial_f(X)$, we have $f(X) - h_X(X) \leq 0$ which follows by the constraint at $Y = \emptyset$. Akin to the submodular polyhedron, the extreme points of the submodular subdifferential also admits interesting characterizations. We shall denote a subgradient at X by $h_X \in \partial_f(X)$. Similar to the submodular polyhedron, the extreme points of $\partial_f(Y)$ may be computed via a greedy algorithm: Let σ be a permutation of V that assigns the elements in X to the first $|X|$ positions ($i \leq |X|$ if and only if $\sigma(i) \in X$) and $S^\sigma_{|X|} = X$. An illustration of this is shown in Figure 2.4.

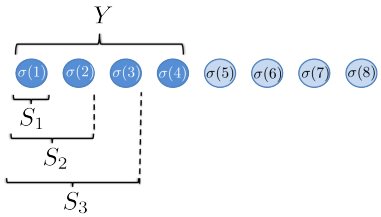

Figure 2.4: A visualization of the chain of sets and permutations σ.

This chain defines an extreme point h^σ_X of $\partial_f(X)$ with entries

$$h^\sigma_X(\sigma(i)) = f(S^\sigma_i) - f(S^\sigma_{i-1}). \tag{2.4}$$

Note that for every subgradient $h_X \in \partial_f(X)$, we can define a modular lower bound $m_X(Y) = f(X) + h_X(Y) - h_X(X), \forall Y \subseteq V$, which satisfies $m_X(Y) \leq f(Y), \forall Y \subseteq V$. Moreover, we have that $m_X(X) = f(X)$, and hence the subdifferential exactly correspond to the set of tight modular lower bounds of a submodular function, at a given set X. If we choose h_X to be an extreme subgradient, the modular lower bound becomes $m_X(Y) = h_X(Y)$, resulting in a normalized modular function[2].

The subdifferential defined in Eqn. (2.3) is defined via an exponential number of inequalities. A key observation however is that many of these inequalities are redundant. Define three polyhedra:

$$\partial_f^1(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \geq f(X) - x(X), \forall Y \subseteq X\} \tag{2.5}$$

$$\partial_f^2(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \geq f(X) - x(X), \forall Y \supseteq X\} \tag{2.6}$$

$$\partial_f^3(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \geq f(X) - x(X), \forall Y : Y \not\subseteq X, Y \not\supseteq X\} \tag{2.7}$$

We immediately have that $\partial_f(X) = \partial_f^1(X) \cap \partial_f^2(X) \cap \partial_f^3(X)$. The following Lemma shows that the inequalities in $\partial_f^3(X)$ are redundant in characterizing $\partial_f(X)$ when given $\partial_f^1(X)$ and $\partial_f^2(X)$.

Lemma 2.2. *([Fujishige, 2005, Lemma 6.4]) Given a submodular function f, $\partial_f(X) = \partial_f^1(X) \cap \partial_f^2(X)$. Hence,*

$$\partial_f(X) = \{\{x \in \mathbb{R}^n : f(Y) - x(Y) \geq f(X) - x(X) \text{ for all } Y \in [\emptyset, X] \cup [X, V]\} \tag{2.8}$$

In the above, $[A, B] = \{X \subseteq V : A \subseteq X \subseteq B\}$ whenever $A \subseteq B$. We see that for $X \neq \{\emptyset, V\}$, many of the inequalities defining $\partial_f(X)$ are in fact redundant.

The subdifferential at the emptyset has a special relationship since $\partial_f(\emptyset) = \mathcal{P}_f$. Similarly $\partial_f(V) = \mathcal{P}_{f^\#}$, where $f^\#(X) = f(V) - f(V \backslash X)$ is the submodular dual of f. Furthermore, since $f^\#$ is a supermodular function, it holds that $\partial_f(V)$ is a *supermodular polyhedron* (for a supermodular function g, the supermodular polyhedron is defined as $\mathcal{P}_g = \{x : x(X) \geq g(X), \forall X \subseteq V\}$).

[2]A set function h is said to be normalized if $h(\emptyset) = 0$).

The following lemma shows another instructive fact about the subdifferentials:

Lemma 2.3. *([Fujishige, 2005, Lemma 6.5]) For any submodular function f, $\partial_f(X) = \partial_{f^X}(X) \times \partial_{f_X}(\emptyset)$, where $f^X(Y) = f(Y), \forall Y \subseteq X$, and $f_X(Y) = f(Y \cup X) - f(X), \forall Y \subseteq V \backslash X$, and $\times$ denotes the direct product.*

Finally define:

$$\partial_f^{\Delta(1,1)}(X) = \{x \in \mathbb{R}^V : \forall j \in X, f(j|X\backslash j) \leq x(j) \text{ and } \forall j \notin X, f(j|X) \geq x(j)\}. \tag{2.9}$$

Notice that $\partial_f^{\Delta(1,1)}(X) \supseteq \partial_f(X)$ since we are reducing the constraints of the subdifferential. In particular $\partial_f^{\Delta(1,1)}(X)$ just considers n inequalities, by choosing the sets Y in Eqn. (2.8) such that $|Y\backslash X| = 1$ or $|X\backslash Y| = 1$. This polyhedron will be useful in characterizing local minimizers of a submodular function (see Section 2.4) and motivating analogous constructs for local maxima (see, for example, Proposition 2.4).

2.2.3 Generalized Submodular Lower Polyhedron

In this section, we define a generalization of the submodular polyhedron, which we call the *generalized submodular lower polyhedron*. While this construct has not been defined explicitly before, we investigate it primarily with the aim of contrasting this with the results on the concave polyhedral aspects of a submodular function (Section 2.6).

Define the generalized submodular lower polyhedron as:

$$\mathcal{P}_f^{\text{gen}} = \{(x, c), x \in \mathbb{R}^n, c \in \mathbb{R} : [x(X) + c] \leq f(X), \forall X \subseteq V\} \tag{2.10}$$

This generalized polyhedron $\mathcal{P}_f^{\text{gen}} \subseteq \mathbb{R}^{n+1}$ intuitively captures the affine (or unnormalized) modular lower bounds of f. The definition above holds for any arbitrary set function, not necessarily submodular, in which case we call it the *generalized lower polyhedron*. In the case of submodular functions, this generalized lower polyhedron has interesting connections to the submodular polyhedron. In particular, note that $\mathcal{P}_f^{\text{gen}} \cap \{(x, c) : c = 0\} = \{(x, c) : x \in \mathcal{P}_f, c = 0\}$. In other words, the slice $c = 0$ of the *generalized submodular polyhedron* is the

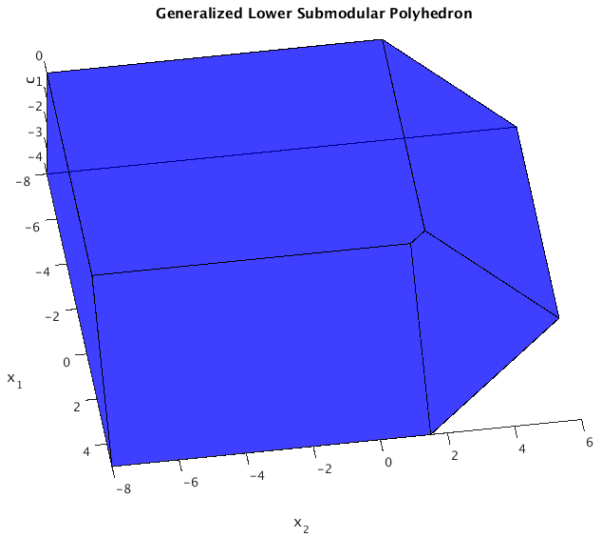

Figure 2.5: The generalized submodular lower function for a two dimensional submodular function $f : 2^{\{1,2\}} \to \mathbb{R}$, satisfying $f(\emptyset) = 0$, $f(\{1\}) = 1, f(\{2\}) = 2, f(\{1,2\}) = 2.5$.

submodular polyhedron of f. Also notice that for a normalized submodular function f, the constraint at $X = \emptyset$, requires that $c \leq 0$.

The generalized polyhedron has interesting connections with the subdifferential – the following is a characterization of its facial structure.

Lemma 2.4. *Given a set function f, $(x, c) \in \mathcal{P}_f^{gen}$ lies on a face of the polyhedron if and only if there exists a set X such that $x \in \partial_f(X)$ and $c = f(X) - x(X)$.*

Proof. Notice that (x, c) lies on a face of $\mathcal{P}_f^{\text{gen}}$ if and only if there exists a set X such that $x(X)+c = f(X)$ and for all $Y \subseteq V, x(Y)+c \leq f(Y)$. Since then $x(Y)-x(X) \leq f(Y)-f(X)$, we have that $x \in \partial_f(X)$ and $c = f(X) - x(X)$ that, as mentioned above, has $c \leq 0$ when f is submodular. □

The extreme points of $\mathcal{P}_f^{\text{gen}}$ also are easy to characterize when f is submodular. Surprisingly, all the extreme points lie exactly on the hyperplane $c = 0$ with x being the extreme points of $\mathcal{P}_f$.

Lemma 2.5. *Given a submodular function f, (x, c) is an extreme point of $\mathcal{P}_f^{gen}$ if and only if x is an extreme point of $\mathcal{P}_f$ and $c = 0$. Furthermore, for any $y \in \mathbb{R}^n$,*

$$\underbrace{\max_{(x,c)\in\mathcal{P}_f^{gen}}[\langle x, y\rangle + c]}_{(i)} = \underbrace{\max\{\max_{x\in\partial_f(X)}[\langle x, y\rangle + f(X) - x(X)] \mid X \subseteq V\}}_{(ii)} = \underbrace{\max_{x\in\mathcal{P}_f}\langle x, y\rangle}_{(iii)} \quad (2.11)$$

Proof. First we show that (i) = (ii). Notice that a maximum in (i) (denoted by (x^*, c^*)) occurs at a face of $\mathcal{P}_f^{\text{gen}}$ which, by Lemma 2.4, says there exists an X such that $x^* \in \partial_f(X)$ with $c^* = f(X) - x(X)$. This subdifferential is considered in (ii)'s outer max implying (ii) $\geq$ (i). Moreover, it is also easy to see that the point $(x, f(X) - x(X)) \in \mathcal{P}_f^{\text{gen}}$, for any $x \in \partial_f(X)$, from the definitions of the generalized submodular lower polyhedron and the subdifferential. Hence (ii) $\geq$ (i), since it is a max over a much larger set.

We then show that (i) = (iii). It is immediate that (iii) $\leq$ (i) since (iii) is a more constrained case of (i) under $c = 0$. Next, we show (iii) $\geq$ (i), which states that for a submodular function, the linear program over the generalized submodular polyhedron is equivalent to a linear program over the submodular polyhedron. This result follows as a corollary from Lemma 2.1. Specifically, for any $(x, c) \in \mathcal{P}_f^{\text{gen}}$, we have that

$$\max_{s\in\mathcal{P}_f} w^\top s = \sum_i \lambda_i f(S_i^{\sigma_w}) \geq \sum_i \lambda_i[\langle x, 1_{S_i^{\sigma_w}}\rangle + c] \geq \langle x, w\rangle + c, \quad (2.12)$$

where the last inequality follows from the facts that $\sum_i \lambda_i 1_{S_i^{\sigma_w}} = w$ and $\sum_i \lambda_i = 1$. In particular, this also means that in the optimization problem in ii, the maximum over $X \subseteq V$ occurs at $X = \emptyset$, when $\partial_f(X) = \partial_f$.

Lastly, note that since every linear program over the generalized submodular polyhedron can be cast as a linear program over the submodular polyhedron, the extreme points of both polyhedra must also be the same. □

Another way to look at this, which also shows the second part of the Lemma, is that (x, c) is an extreme point if x is an extreme point of a subdifferential $\partial_f(X)$ for some set X. Since the extreme points of the subdifferentials are exactly the extreme points of the submodular polyhedron, the result follows.

Finally it is worth mentioning that similar to the submodular polyhedron, the *generalized submodular polyhedron membership problem* (i.e does $(x, c) \in \mathcal{P}_f^{\text{gen}}$) is polynomial time, and can be solved via submodular minimization. This is again parallel to the submodular polyhedron.

Proposition 2.2. *Given a submodular function f, $(x, c) \in \mathcal{P}_f^{gen}$ if and only if $c \leq \min_{X \subseteq V}[f(X) - x(X)]$. Since submodular minimization is polynomial time, the generalized submodular polyhedral membership problem is also polynomial time.*

A visualization of the generalized submodular lower polyhedron for a submodular function on $V = \{v_1, v_2\}$ is shown in Figure 2.5.

2.3 Convex extensions of a Submodular Function

We now introduce the convex extension of a submodular functions. We shall see a number of equivalent ways to characterize this extension and observe how they can be computed very efficiently as what is known as the Lovász extension [Lovász, 1983]. The results of this section are mainly from [Lovász, 1983, Dughmi, 2009]. Following [Vondrák, 2007, Dughmi, 2009], we consider two main characterizations of the convex extensions, as what we call *polyhedral characterization* and *distributional characterization*. Again, the main purpose of this section, is to review the existing work, and contrast the results with the ones we shall see in Section 2.7, on the concave extensions of submodular functions.

2.3.1 Polyhedral characterization of the convex extensions

The convex extension of any set function (not necessarily submodular) can be seen as the point-wise supremum of convex functions which lower bound the set function [Dughmi, 2009]. More clearly, let

$$\Phi_f = \{\phi : \phi \text{ is convex in } [0,1]^V \text{ and } \phi(1_X) \leq f(X), \forall X \subseteq V\}. \tag{2.13}$$

Then define:

$$\breve{f}(w) = \max_{\phi \in \Phi_f} \phi(w), \text{ for } w \in [0,1]^n \tag{2.14}$$

It is not hard to show that $\breve{f}$ is convex and satisfies the relation $\breve{f}(1_X) = f(X)$. The above expression can in fact be simplified for any set function, and it suffices to consider affine lower instead of convex lower bounds. In particular Eqn. (2.14) can be expressed as a linear program over the generalized polyhedron.

Lemma 2.6. *Given a set function f, the convex extension of f in Eqn.* (2.14) *can be expressed as:*

$$\breve{f}(w) = \max_{(x,c) \in \mathcal{P}_f^{gen}} [\langle x, w \rangle + c], \forall w \in [0,1]^n \tag{2.15}$$

Proof. The proof of the equivalence follows from a simple observation. For a given w, let $\hat{\phi}$ be an argmax in Eqn. (2.14). Then since $\hat{\phi}$ is a convex function in $[0,1]^V$, there exists a subgradient $x \in \mathbb{R}^n$ at w and value d, such that $\langle x, y \rangle + d \leq \hat{\phi}(y), \forall y$ and $\langle x, w \rangle + d = \hat{\phi}(w)$. In other words, $\langle x, y \rangle + d$ is a linear lower bound of $\hat{\phi}(y)$, tight at w. Hence, at value w, $\breve{f}(w)$ takes value $\langle x, w \rangle + d$. Finally notice that $(x, d) \in \mathcal{P}_f^{\text{gen}}$ since $x(X) + d \leq \hat{\phi}(1_X) \leq f(X), \forall X \subseteq V$. □

So far, in the above, we have not invoked the submodularity of f. If f is submodular however, the above polyhedral characterization can be replaced by an linear program over the submodular polyhedron. In other words,

Lemma 2.7. *For a submodular function f, the expressions in Eqn.* (2.14), (2.15) *can be rewritten as:*

$$\breve{f}(w) = \max_{s \in \mathcal{P}_f} w^\top s, \forall w \in [0,1]^n. \tag{2.16}$$

Proof. This follows directly from Lemma 2.5. □

The above result is not surprising given that the extreme points of $\mathcal{P}_f^{\text{gen}}$ are identical to the extreme points of $\mathcal{P}_f$, when f is submodular.

2.3.2 Distributional characterization of the convex extension

Another way to characterize the continuous extension of a set function f is as follows. Denote Λ_w as the set:

$$\Lambda_w = \Big\{\{\lambda_S, S \subseteq V\} : \sum_{S\subseteq V} \lambda_S 1_S = w, \sum_{S\subseteq V} \lambda_S = 1, \text{ and } \forall S, \lambda_S \geq 0\Big\}, \text{ for } w \in [0,1]^n \quad (2.17)$$

Then the convex extension $\breve{f}$ can be equivalently written as:

$$\breve{f}(w) = \min_{\lambda\in\Lambda_w} \sum_{S\subseteq V} \lambda_S f(S) \quad (2.18)$$

The reason this representation is called distributional is that the convex extension here is computed by minimizing over particular distributions over sets. Again, it is not hard to see that this characterization is a convex extension.

For a submodular function, the distribution characterization takes on a nice form, which is known classically as the Lovász extension. This result can be found, for example, in [Dughmi, 2009]:

Lemma 2.8. *[Dughmi, 2009]*

Given a submodular function f,

$$\breve{f}(w) = \sum_{i=1}^{n} w(\sigma_w(i))(f(S_i^{\sigma_w}) - f(S_{i-1}^{\sigma_w}) = w(\sigma_w(n)))f(S_n^{\sigma_w}) + \sum_{i=1}^{n-1}(w(\sigma_w(i)) - w(\sigma_w(i+1)))f(S_i^{\sigma_w}), \quad (2.19)$$

where σ_w is a permutation satisfying $w(\sigma_w(1)) \geq w(\sigma_w(2) \geq \cdots \geq w(\sigma_w(n))$.

It is clear from above that the minimizing distribution λ is a form of a chain distribution, where the chain here is the sequence of sets $S_0^{\sigma_w}, S_1^{\sigma_w}, \cdots, S_n^{\sigma_w}$ defined in Lemma 2.1. We also see the relationship between the two characterizations in the case of submodular functions, since Eqn. (2.19) is exactly the solution of the linear program over the submodular polyhedron (see Lemma 2.1). Hence the two forms of convex extensions, i.e the distributional characterization from Lemma 2.8 and polyhedral characterization from Lemma 2.7, are identical for a submodular function. The resulting convex function $\breve{f}$ is the Lovász extension.

The equivalence between the two characterizations holds for general set functions, not necessarily submodular. In other words, Eqn. (2.18) and Eqn. (2.14), (2.15) are identical for any set function. This follows directly from the arguments in [Dughmi, 2009]. The only catch, however, is that Lemmas 2.7 and 2.8 do not hold for general set functions and $\breve{f}$ can be NP hard to evaluate in general [Dughmi, 2009, Vondrák, 2007].

2.3.3 Convex Extensions and Submodular Minimization

The Lovász extension plays an important role in submodular minimization. In particular, minimizing the Lovász extension is equivalent to minimizing a submodular function:

Lemma 2.9. *[Lovász, 1983]) Given a submodular function f,*

$$\min_{X \subseteq V} f(X) = \min_{x \in [0,1]^n} \breve{f}(x) \tag{2.20}$$

Furthermore, given the minimizer x^* of the RHS above, we can obtain a set X^* such that $f(X^*) = \breve{f}(x^*)$.

This implies that unconstrained submodular minimization has an integrality gap of one and the two problems are equivalent.

2.4 Optimality conditions for submodular minimization

Fujishige [Fujishige, 1984c] provides some interesting characterizations to the optimality conditions for unconstrained submodular minimization. The following theorem can be thought of as a discrete analog to the KKT conditions:

Lemma 2.10. *([Fujishige, 2005, Lemma 7.1]) A set $A \subseteq V$ is a minimizer of $f : 2^V \to \mathbb{R}$ if and only if:*

$$\boldsymbol{0} \in \partial_f(A) \tag{2.21}$$

This immediately provides necessary and sufficient conditions for optimality of f:

Lemma 2.11. *([Fujishige, 2005, Theorem 7.2]) A set A minimizes a submodular function f if and only if $f(A) \leq f(B)$ for all sets B such that $B \subseteq A$ or $A \subseteq B$.*

In other words, it is sufficient to check only the subsets and supersets of A to ensure that A is a global optimizer of f. The above Lemma follows from Eqn. 2.8 and Lemma 2.10. Analogous characterizations have also been provided for constrained forms of submodular minimization, and interested readers may look at [Fujishige, 1984c]. Finally, we can provide a simple characterization on the local minimizers of a submodular function.

Lemma 2.12. *A set $A \subseteq V$ is a local minimizer[3] of a submodular function if and only if $\mathbf{0} \in \partial_f^{\Delta(1,1)}(A)$.*

As was shown in [Iyer et al., 2013b], a local minimizer of a submodular function, in the unconstrained setting, can be found efficiently in $O(n^2)$ complexity.

While unconstrained submodular minimization is easy, most forms of constrained submodular minimization become NP hard. For example, a simple cardinality lower bound constraint makes the problem of submodular minimization (even with monotone submodular functions) NP hard without even constant factor approximation guarantees [Svitkina and Fleischer, 2008]. These results, however, can be extended when the constraints are lattice constraints [Fujishige, 2005] in which case many of the results above still hold.

2.5 Convex Characterizations: Discrete Separation Theorem and Fenchel Duality Theorem

Finally we review some interesting theorems which characterize convex functions, but also go through for submodular functions.

2.5.1 The Discrete Separation Theorem (DST)

The discrete separation theorem known in context of convexity, states that given a convex function ϕ and a concave function ψ such that $\forall x, \phi(x) \geq \psi(x)$, these exists a affine function

[3] A set A is a local minimizer of a submodular function if $f(X) \geq f(A), \forall X : |X \backslash A| \leq 1,$ and $|A \backslash X| = 1$, that is all sets X no more than hamming-distance one away from A.

$\langle h, x\rangle + c$ such that $\forall x, \psi(x) \geq \langle h, x\rangle + c \geq \psi(x)$. A similar relation holds for submodular functions.

The lemma below was shown by Frank [Frank, 1982] in the context of submodular functions:

Lemma 2.13. *[Frank, 1982], [Fujishige, 2005, Theorem 4.12] Given a submodular function f and a supermodular function g such that $f(X) \geq g(X), \forall X$ (and which satisfy $f(\emptyset) = g(\emptyset) = 0$), there exists a modular function h such that $f(X) \geq h(X) \geq g(X)$. Furthermore, if f and g are integral so may be h.*

This Lemma can also be shown through the Lovász extension. In particular, given a submodular function f and a supermodular function g such that $f(X) \geq g(X), \forall X$, we can construct the convex and concave extensions $\breve{f}$ and $\hat{g}$ of f and g (the concave extension $\hat{g}$ can be constructed via the Lovász extension of $-g$). From the expressions of $\breve{f}$ and $\hat{g}$, it is not hard to see that $\breve{f}(x) \geq \hat{g}(x), \forall x$. Hence using the Discrete Separation Theorem from convex analysis, we can find a linear function h, which when restricted to $0/1$ vectors, gives the modular function h.

2.5.2 Fenchel Duality Theorem (FDT)

The Fenchel duality theorem in the context of convexity [Rockafellar, 1997] provides a relation between the minimizers of the function and it is dual. Given a convex function ϕ and a concave function ψ, the Fenchel dual ϕ^* of ϕ, and ψ^* of ψ is given as follows:

$$\phi^*(y) = \max_{x \in \text{dom}(\phi)} [\langle x, y\rangle - \phi(x)], \quad \psi^*(y) = \min_{x \in \text{dom}(\psi)} [\langle x, y\rangle - \psi(y)]. \tag{2.22}$$

The dual functions ϕ^* and ψ^* are convex and concave respectively. The Fenchel duality theorem then states that:

$$\min_x [\phi(x) - \psi(x)] = \max_y [\psi^*(y) - \phi^*(y)] \tag{2.23}$$

Analogous characterizations also hold for submodular functions [Fujishige, 1984c]. Given a submodular function f (or equivalently supermodular function g), the Fenchel dual f^* (or equivalently g^*) is:

$$f^*(x) = \max_{X \subseteq V}[x(X) - f(X)], \quad g^*(x) = \min_{X \subseteq V}[x(X) - g(X)]. \tag{2.24}$$

The Fenchel duals f^* and g^* are convex and concave functions respectively. Then the following Lemma holds:

Lemma 2.14. *([Fujishige, 2005, Theorem 6.3]) Given a submodular function f and a supermodular function g,*

$$\min_{X \subseteq V}[f(X) - g(X)] = \max_{x}[g^*(x) - f^*(x)]. \tag{2.25}$$

Further if f and g are integral, the maximum on the right hand side is attained by an integral vector x.

2.5.3 The Minkowski-Sum theorem

Submodular polyhedra and also the subdifferentials have an interesting characterization related to Minkowski sums.

Lemma 2.15. *([Fujishige, 2005, Theorem 6.8]) Given two submodular functions f_1 and f_2, it holds that that (the addition of the polyhedra below corresponds to a point-wise addition):*

$$\mathcal{P}_{f_1+f_2} = \mathcal{P}_{f_1} + \mathcal{P}_{f_2}, \;\; \partial_{f_1+f_2}(X) = \partial_{f_1}(X) + \partial_{f_2}(X) \tag{2.26}$$

Similarly it holds that, $\mathcal{P}^{gen}_{f_1+f_2} = \mathcal{P}^{gen}_{f_1} + \mathcal{P}^{gen}_{f_2}$.

The Minkowski Sum Theorem for the generalized submodular polyhedron, follows directly from the definition.

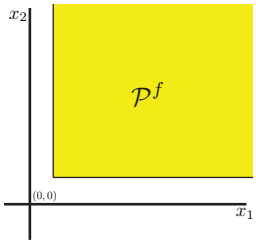

Figure 2.6: The submodular upper Polyhedron $\mathcal{P}^f$ in two dimensions.

2.6 Concave Polyhedral Aspects of Submodular Functions

In this section, we investigate several polyhedral aspects of submodular functions relating them to concavity, thus complementing the results from Section 2.2. This provides a complete picture on the relationship between submodularity, convexity and concavity. We investigate the submodular upper polyhedron, submodular superdifferential and the generalized submodular upper polyhedron.

2.6.1 The submodular upper polyhedron

A first step in characterizing the concave aspects of a submodular function is the submodular upper polyhedron. Intuitively this is the set of tight modular upper bounds of the function, and we define it as follows:

$$\mathcal{P}^f = \{x \in \mathbb{R}^n : x(S) \geq f(S), \forall S \subseteq V\} \tag{2.27}$$

The above polyhedron can in fact be defined for any set function. In particular, when f is supermodular, we get what is known as the *supermodular polyhedron* [Fujishige, 2005]. Presently, we are interested in the case when f is submodular and hence we call this the *submodular upper polyhedron*. Interestingly this has a very simple characterization.

Lemma 2.16. *Given a submodular function f,*

$$\mathcal{P}^f = \{x \in \mathbb{R}^n : x(j) \geq f(j)\} \tag{2.28}$$

Proof. Given $x \in \mathcal{P}^f$ and a set S, we have $x(S) = \sum_{i \in S} x(i) \geq \sum_{i \in S} f(i)$, since $\forall i, x(i) \geq f(i)$ by Eqn. (2.27). Hence $x(S) \geq \sum_{i \in S} f(i) \geq f(S)$. Thus, the irredundant inequalities are the singletons. □

Thus, the submodular upper polyhedron has a particularly simple characterization due to the submodularity of f. In other words, this polyhedron is not *polyhedrally tight* in that many of the defining inequalities are redundant. This polyhedron alone is not particularly interesting to define a concave extension. We shall define a generalization of this in Section 2.6.3 that shall prove useful in characterizing the concave extension.

We end this section by investigating the submodular upper polyhedron membership problem. Owing to its simplicity, this problem is particularly simple which might seem surprising at first glance since $x \in \mathcal{P}^f$ is equivalent to, Eqn. (2.27), checking if $\max_{X \subseteq V} f(X) - x(X) \leq 0$. This involves maximization of a submodular function which is NP hard. The following lemma shows that this problem is actually easy.

Corollary 2.1. *Given a submodular function f and vector x, let X be a set such that $f(X) - x(X) > 0$. Then there exists an $i \in X : f(i) - x(i) > 0$.*

Proof. Observe that $f(X) - x(X) \leq \sum_{i \in X} f(i) - x(i)$. Since the l.h.s. is greater than 0, it implies that $\sum_{i \in X} f(i) - x(i) > 0$. Hence there should exist an $i \in X$ such that $f(i) - x(i) > 0$. □

Thus, it is sufficient to check the singleton values, i.e $f(i) - x(i)$, and if all these are less than or equal to zero, then $x \in \mathcal{P}^f$. This also follows immediately from Lemma 2.16.

An interesting corollary of the above, is that it is in fact easy to check if the maximizer of a submodular function is greater than equal to zero. Given a submodular function f, the problem is whether $\max_{X \subseteq V} f(X) \geq 0$. This can easily be checked without resorting to submodular function maximization.

Corollary 2.2. *Given a submodular function f with $f(\emptyset) = 0$, $\max_{X \subseteq V} f(X) > 0$ if and only if there exists an $i \in V$ such that $f(i) > 0$.*

Proof. If for any j, $f(j) > 0$ it implies that $\max_{X \subseteq V} f(X) \geq f(j) > 0$. On the other hand, if $\forall j, f(j) \leq 0$, we have that $\forall X \subseteq V, f(X) \leq \sum_{i \in X} f(i) \leq 0$. Hence $\max_{X \subseteq V} f(X) = 0$. □

This fact is true only for a submodular function. For general set functions, even when $f(\emptyset) = 0$, it could potentially require an exponential search to determine if $\max_{X \subseteq V} f(X) > 0$.

2.6.2 The Submodular Superdifferentials

Given a submodular function f we can characterize its superdifferential as follows. We denote for a set X, the superdifferential with respect to X as $\partial^f(X)$. Observe then that,

$$\partial^f(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \leq f(X) - x(X), \forall Y \subseteq V\} \tag{2.29}$$

This characterization is analogous to the subdifferential of a submodular function defined in Eqn. (2.3). This is also akin to the superdifferential corresponding to a continuous concave function.

Each supergradient $g_X \in \partial^f(X)$, defines a modular upper bound of a submodular function. In particular, define $m^X(Y) = g_X(Y) + f(X) - g_X(X)$. Then $m^X(Y)$ is a modular function which satisfies $m^X(Y) \leq f(Y), \forall Y \subseteq X$ and $m^X(X) = f(X)$.

Note that the superdifferential is defined by an exponential (i.e., $2^{|V|}$) number of inequalities. However owing to the submodularity of f and akin to the subdifferential of f, we can reduce the number of inequalities. Define three polyhedrons as:

$$\partial_1^f(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \leq f(X) - x(X), \forall Y \subseteq X\} \tag{2.30}$$

$$\partial_2^f(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \leq f(X) - x(X), \forall Y \supseteq X\} \tag{2.31}$$

$$\partial_3^f(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \leq f(X) - x(X), \forall Y : Y \not\subseteq X, Y \not\supseteq X\} \tag{2.32}$$

A trivial observation is that: $\partial^f(X) = \partial_1^f(X) \cap \partial_2^f(X) \cap \partial_3^f(X)$. As we show below for a submodular function f, $\partial_1^f(X)$ and $\partial_2^f(X)$ are actually very simple polyhedra.

Lemma 2.17. *For a submodular function f,*

$$\partial_1^f(X) = \{x \in \mathbb{R}^n : f(j|X\backslash j) \geq x(j), \forall j \in X\} \tag{2.33}$$

$$\partial_2^f(X) = \{x \in \mathbb{R}^n : f(j|X) \leq x(j), \forall j \notin X\}. \tag{2.34}$$

Proof. Consider $\partial_1^f(X)$. Notice that the inequalities defining the polyhedron can be rewritten as $\partial_1^f(X) = \{x \in \mathbb{R}^n : x(X\backslash Y) \leq f(X) - f(Y), \forall Y \subseteq X\}$. We then have that $x(X\backslash Y) = \sum_{j \in X\backslash Y} x(j) \leq \sum_{j \in X\backslash Y} f(j|X\backslash j)$, since $\forall j \in X, x(j) \leq f(j|X\backslash j)$ (this follows by considering only the subset of inequalities of $\partial_1^f(X)$ with sets Y such that $|X\backslash Y| = 1$). Hence $x(X\backslash Y) \leq \sum_{j \in X\backslash Y} f(j|X\backslash j) \leq f(X) - f(Y)$. Hence an irredundant set of inequalities include those defined only through the singletons.

In order to show the characterization for $\partial_2^f(X)$, we have that $\partial_2^f(X) = \{x \in \mathbb{R}^n : x(Y\backslash X) \geq f(Y) - f(X), \forall Y \supseteq X\}$. It then follows that, $x(Y\backslash X) = \sum_{j \in Y\backslash X} x(j) \geq \sum_{j \in Y\backslash X} f(j|X)$, since $\forall j \notin X, x(j) \geq f(j|X)$. Hence $x(X\backslash Y) \geq \sum_{j \in Y\backslash X} f(j|X) \geq f(Y) - f(X)$, and again, an irredundant set of inequalities include those defined only through the singletons. □

The above characterization significantly reduces the inequalities governing $\partial^f(X)$, and in fact, the polytopes $\partial_1^f(X)$ and $\partial_2^f(X)$ are very simple polyhedra. Recall that this is analogous to the submodular subdifferential, where again owing to submodularity the number of inequalities are reduced significantly. In that case, we just need to consider the sets Y which are subsets and supersets of X. It is interesting to note the contrast between the redundancy of inequalities in the subdifferentials and the superdifferentials. In particular, here, the inequalities corresponding to sets Y being the subsets and supersets of X are mostly redundant, while the non-redundant ones are the rest of the inequalities. In other words, in the case of the subdifferential, $\partial_f^1(X)$ and $\partial_f^2(X)$ were non-redundant, while $\partial_f^3(X)$ was entirely redundant given the first two. In the case of the superdifferentials, $\partial_1^f(X)$ and $\partial_3^f(X)$ are mostly internally redundant (they can be represented using only by n inequalities), while $\partial_3^f(X)$ has no redundancy in general.

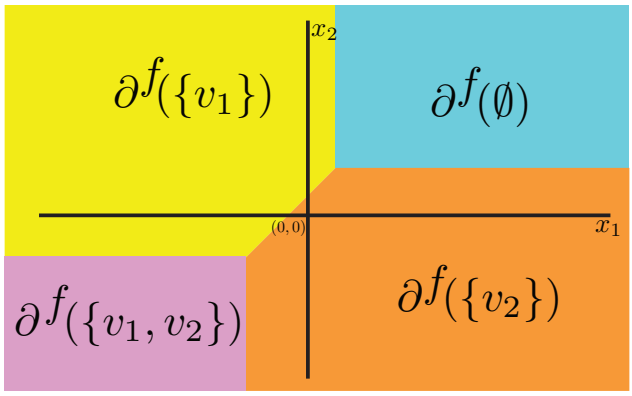

Figure 2.7: A visualization of the four submodular superdifferentials $\partial^f(Y)$ for different sets Y in two dimensions $V = \{v_1, v_2\}$, as described in Example 2.1.

In order to gain more intuition for the superdifferentials, we consider some examples in both two and three dimensions.

Example 2.1. *We consider here superdifferentials when* $V = \{1, 1\}$. *Then from the lemma above,* $\partial^f(\emptyset) = \{x \in \mathbb{R}^2 : f(j|\emptyset) \leq x(j), \forall j \in \{1, 1\}\}$. *Similarly* $\partial^f(\{1, 2\}) = \{x \in \mathbb{R}^2 : f(j|V\backslash j) \geq x(j), \forall j \in \{1, 2\}\}$. *Now consider* $\partial^f(1)$. *Then the governing inequalities for this are:*

$$\partial^f(1) = \{x \in \mathbb{R}^2 : x_1 \leq f(\{1\}), \tag{2.35}$$

$$x_2 \geq f(\{2\}|\{1\}), \tag{2.36}$$

$$x_1 - x_2 \leq f(\{1\}) - f(\{2\})\} \tag{2.37}$$

The extreme points of this polyhedron are the vectors $\{f(\{1\}), f(\{2\})\}$ *and* $\{f(\{1\}|\{2\}), f(\{2\}|\{1\})\}$. *The way we obtain the extreme points is as follows. Setting the inequalities* (2.35) *and* (2.37) *as equalities, we get the extreme point* $\{f(\{1\}), f(\{2\})\}$. *The inequality* (2.36) *then is* $x_2 = f(\{2\}) \geq f(\{2\}|\{1\})$, *which holds. We then set inequalities* (2.36) *and* (2.37) *as equalities, which gives* $x_2 = f(\{2\}|\{1\})$ *and* $x_1 = f(\{1\}|\{2\})$, *thus giving the second extreme point* $\{f(\{1\}|\{2\}), f(\{2\}|\{1\})\}$. *It is easy to see that inequality* (2.35) *is satisfied. Finally,*

if we set inequalities (2.35) *and* (2.36) *as equalities, we get* $x_1 = f(\{1\}), x_2 = f(\{2\}|\{1\})$. *Then we have that* $x_1 - x_2 = f(\{1\}) - f(\{2\}|\{1\}) = 2f(\{1\}) - f(\{1\},\{2\})$. *Inequality* (2.37), *then requires,* $2f(\{1\}) - f(\{1,2\}) \geq f(\{1\}) - f(\{2\}) \Rightarrow f(\{1\}) + f(\{2\}) \leq f(\{1,2\})$, *which does not hold. Hence the only extreme points are the two vectors above. One can similarly investigate* $\partial^f(\{2\})$, *which has the same extreme points.*

As it is clear from the above example the superdifferentials in 2-dimensional case are easy to find and characterize. However this is not the case in three dimensions where the shape of the superdifferentials depends strongly on the particulars of the submodular functions.

Example 2.2. *Let* $V = \{1,2,3\}$. *Recall that* $f(\emptyset) = 0$. *Then consider* $\partial^f(\{1\}) = \{x \in \mathbb{R}^3 : f(Y) - x(Y) \leq f(\{1\}) - x(\{1\}), \forall Y \subseteq \{1,2,3\}\}$ *This polyhedron can be represented via the following irredundant inequalities:*

$$\partial^f(\{1\}) = \{x \in \mathbb{R}^3 : x_1 \leq f(\{1\}), \qquad \text{for } Y = \{\emptyset\} \tag{2.38}$$

$$x_2 \geq f(\{2\}|\{1\}), \qquad \text{for } Y = \{1,2\} \tag{2.39}$$

$$x_2 - x_1 \geq f(\{2\}) - f(\{1\}), \qquad \text{for } Y = \{2\} \tag{2.40}$$

$$x_3 \geq f(\{3\}|\{1\}), \qquad \text{for } Y = \{1,3\} \tag{2.41}$$

$$x_3 - x_1 \geq f(\{3\}) - f(\{1\}), \qquad \text{for } Y = \{3\} \tag{2.42}$$

$$x_2 + x_3 - x_1 \geq f(\{2,3\}) - f(\{1\}) \qquad \text{for } Y = \{2,3\}\} \tag{2.43}$$

The other two inequalities (for $Y = \{1,2,3\}$ *and* $Y = \{1\}$*) are redundant. We now consider the extreme points of this polyhedron. We consider Eqns.* (2.38), (2.40), (2.42) *with equality, and we obtain an extreme point* $\{f(\{1\}), f(\{2\}), f(\{3\})\}$. *It is easy to see that all other inequalities are satisfied. Consider next Eqns.* (2.40), (2.42), *and* (2.43) *with equality and we get a potential extreme point* $\{f(\{1\}) + f(\{2,3\}) - f(\{2\}) - f(\{3\}), f(\{2\}|\{3\}), f(\{3\}|\{2\})\}$. *Observe that* $x_1 = f(\{1\}) + f(\{2,3\}) - f(\{2\}) - f(\{3\}) \leq f(\{1\})$, *and hence Eqn.* (2.38) *is satisfied. However* $x_2 = f(\{2\}|\{3\})$ *may be bigger or smaller than* $f(\{2\}|\{1\})$ *and Eqn.* (2.39) *may or may not be violated. Similarly,* $x_3 = f(\{3\}|\{2\})$ *is not comparable to* $f(\{3\}|\{1\})$, *and hence Eqn.* (2.41) *might or might not be violated. Consequently we cannot determine if by*

combining together Eqns. (2.40), (2.42), *and* (2.43), *we obtain an extreme point, unless we have more information about the current submodular function being used. Hence we see from this example that we cannot hope to find the extreme points analytically.*

The above example shows that a particular expression (obtained via a combination of inequalities) might or might not be extreme, depending on the particular submodular function and its valuation. This is unlike the subdifferential, where an analytical expression is always extreme for all submodular functions. Thus, unlike the subdifferentials, we cannot expect a closed form expression for the extreme points of $\partial^f(Y)$. Moreover, they also seem to be hard to characterize algorithmically. For example, the superdifferential membership problem is NP hard.

Lemma 2.18. *Given a submodular function f and a set $Y : \emptyset \subset Y \subset V$, the membership problem $y \in \partial^f(Y)$ is NP hard.*

Proof. Notice that the membership problem $y \in \partial^f(Y)$ is equivalent to asking $\max_{X \subseteq V} f(X) - y(X) \leq f(Y) - y(Y)$. In other words, this is equivalent to asking if Y is a maximizer of $f(X) - y(X)$ for a given vector y. This is the decision version of the submodular maximization problem and correspondingly is NP hard when $\emptyset \subset Y \subset V$. □

Given that the membership problem is NP hard, it is also NP hard to solve a linear program over this polyhedron [Grotschel et al., 1984, Schrijver, 2003]. The superdifferential of the empty and ground set, however, can be characterized easily:

Lemma 2.19. *For any submodular function f such that $f(\emptyset) = 0$, $\partial^f(\emptyset) = \{x \in \mathbb{R}^n : f(j) \leq x(j), \forall j \in V\}$. Similarly $\partial^f(V) = \{x \in \mathbb{R}^n : f(j|V \backslash j) \geq x(j), \forall j \in V\}$. Furthermore, $\partial^f(\emptyset) = \mathcal{P}^f$ and $\partial^f(V) = \mathcal{P}^{f^\#}$.*

Proof. This lemma follows directly from the definitions. □

As we saw above, it is hard to characterize the superdifferential of a submodular function. It is however possible to give inner and outer bounds as shown in the following subsections. Using these, we can also find certain specific useful supergradients.

Outer Bounds on the Superdifferential

It is possible to provide a number of useful and practical outer bounds on the superdifferential. Recall that $\partial_1^f(Y)$ and $\partial_2^f(Y)$ are already simple polyhedra. We can then provide outer bounds on $\partial_3^f(Y)$ that, together with $\partial_1^f(Y)$ and $\partial_2^f(Y)$, provide simple bounds on $\partial^f(Y)$. Define for $1 \leq k, l \leq n$:

$$\partial^f_{3,\Delta(k,l)}(X) = \{x \in \mathbb{R}^n : f(Y) - x(Y) \leq f(X) - x(X), \\ \forall Y : Y \not\subseteq X, Y \not\supseteq X, |Y \backslash X| \leq k-1, |X \backslash Y| \leq l-1\} \tag{2.44}$$

We can then define the outer bound:

$$\partial^f_{\Delta(k,l)}(X) = \partial_1^f(X) \cap \partial_2^f(X) \cap \partial^f_{3,\Delta(k,l)}(X). \tag{2.45}$$

Observe that $\partial^f_{\Delta(k,l)}(X)$ is expressed in terms of $O(n^{k+l})$ inequalities, and hence for a given k, l we can obtain the representation of $\partial^f_{\Delta(k,l)}(X)$ in polynomial time. We will see that this provides us with a hierarchy of outer bounds on the superdifferential:

Theorem 2.1. *For a submodular function f:*

1. $\partial^f_{\Delta(1,1)}(X) = \partial_1^f(X) \cap \partial_2^f(X)$
2. $\forall 1 \leq k' \leq k, 1 \leq l' \leq l, \partial^f(X) \subseteq \partial^f_{\Delta(k,l)}(X) \subseteq \partial^f_{\Delta(k',l')}(X) \subseteq \partial^f_{\Delta(1,1)}(X)$
3. $\partial^f_{\Delta(n,n)}(X) = \partial^f(X)$.

Proof. The proofs of items 1 and 3 follow directly from definitions. To see item 2, notice that the polyhedra $\partial^f_{\Delta(k,l)}$ become tighter as k and l increase finally approaching the superdifferential. □

Similar to the submodular subdifferential, we shall call $\partial^f_{\Delta(1,1)}(Y)$ as the local approximation of the superdifferential. In particular,

$$\partial^f_{\Delta(1,1)}(X) = \{x \in \mathbb{R}^n : x(j) \leq f(j|X \backslash j) \forall j \in X, x(j) \geq f(j|X) \forall j \notin X\} \tag{2.46}$$

We shall see in Section 2.8 that these outer bounds have interesting connections with approximation algorithms for submodular maximization.

Inner Bounds on the Superdifferential

While it is hard to characterize the extreme points of the superdifferential, we can provide some specific supergradients. Define three vectors as follows:

$$\hat{g}_X(j) = \begin{cases} f(j|X-j) & \text{if } j \in X \\ f(j) & \text{if } j \notin X \end{cases} \tag{2.47}$$

$$\check{g}_X(j) = \begin{cases} f(j|V-j) & \text{if } j \in X \\ f(j|X) & \text{if } j \notin X \end{cases} \tag{2.48}$$

$$\bar{g}_X(j) = \begin{cases} f(j|V-j) & \text{if } j \in X \\ f(j) & \text{if } j \notin X \end{cases} \tag{2.49}$$

Then we have the following theorem:

Theorem 2.2. *For a submodular function f, $\hat{g}_X, \check{g}_X, \bar{g}_X \in \partial^f(X)$. Hence for every submodular function f and set X, $\partial^f(X)$ is non-empty.*

Proof. For submodular f, the following bounds are known to hold [Nemhauser et al., 1978]:

$$f(Y) \leq f(X) - \sum_{j \in X \setminus Y} f(j|X \setminus j) + \sum_{j \in Y \setminus X} f(j|X \cap Y), \tag{2.50}$$

$$f(Y) \leq f(X) - \sum_{j \in X \setminus Y} f(j|X \cup Y \setminus j) + \sum_{j \in Y \setminus X} f(j|X) \tag{2.51}$$

Using submodularity, we can loosen these bounds further to provide tight modular upper bounds [Ahmed and Atamtürk, 2009, Jegelka and Bilmes, 2009, 2011b, Iyer and Bilmes, 2012a,b]:

$$f(Y) \leq f(X) - \sum_{j \in X \setminus Y} f(j|X - \{j\}) + \sum_{j \in Y \setminus X} f(j|\emptyset) \tag{2.52}$$

$$f(Y) \leq f(X) - \sum_{j \in X \setminus Y} f(j|V - \{j\}) + \sum_{j \in Y \setminus X} f(j|X) \tag{2.53}$$

$$f(Y) \leq f(X) - \sum_{j \in X \setminus Y} f(j|V - \{j\}) + \sum_{j \in Y \setminus X} f(j|\emptyset). \tag{2.54}$$

From the three bounds above, and substituting the expressions of the supergradients, we may immediately verify that these are supergradients, namely that $\hat{g}_X, \check{g}_X, \bar{g}_X \in \partial^f(X)$. For example, starting with Eqn. (2.52), we have:

$$f(Y) \leq f(X) - \sum_{j \in X \setminus Y} f(j|X - \{j\}) + \sum_{j \in Y \setminus X} f(j|\emptyset) \tag{2.55}$$

$$= f(X) - \sum_{j \in X} f(j|X \setminus \{j\}) + \sum_{j \in X \cap Y} f(j|X \setminus \{j\}) + \sum_{j \in Y \setminus X} f(j|\emptyset) \tag{2.56}$$

$$= f(X) - \hat{g}_X(X) + \hat{g}_X(Y) \tag{2.57}$$

□

These supergradients characterize inner bounds for the superdifferential. Define two polyhedra:

$$\partial^f_\emptyset(X) = \{x \in \mathbb{R}^n : f(j) \leq x(j), \forall j \notin X\}, \tag{2.58}$$

$$\partial^f_V(X) = \{x \in \mathbb{R}^n : f(j|V \setminus j) \geq x(j), \forall j \in X\}. \tag{2.59}$$

Then define:

$$\partial^f_{i,1}(X) = \partial^f_1(X) \cap \partial^f_V(X) = \{x \in \mathbb{R}^n : f(j|X \setminus j) \geq x(j), \forall j \in X \text{ and } f(j) \leq x(j), \forall j \notin X\} \tag{2.60}$$

$$\partial^f_{i,2}(Y) = \partial^f_2(Y) \cap \partial^f_\emptyset(Y) = \{x \in \mathbb{R}^n : f(j|V \setminus j) \geq x(j), \forall j \in X \text{ and } f(j|X) \leq x(j), \forall j \notin X\} \tag{2.61}$$

$$\partial^f_{i,3}(Y) = \partial^f_V(Y) \cap \partial^f_\emptyset(Y) = \{x \in \mathbb{R}^n : f(j|V \setminus j) \geq x(j), \forall j \in X \text{ and } f(j) \leq x(j), \forall j \notin X\}. \tag{2.62}$$

Then note that $\partial^f_{i,1}(Y)$ is a polyhedron with $\hat{g}_Y$ as an extreme point. Similarly $\partial^f_{i,2}(Y)$ has $\check{g}_Y$, while $\partial^f_{i,3}(Y)$ has $\bar{g}_Y$ as its extreme points. All these are simple polyhedra, with a single extreme point. Also define,

$$\partial^f_{i,(1,2)}(Y) = \text{conv}(\partial^f_{i,1}(Y), \partial^f_{i,2}(Y)) \tag{2.63}$$

where conv(.,.) represents the convex combination of two polyhedra[4]. Then $\partial^f_{i,(1,2)}(Y)$ is a

[4]Given two polyhedra $\mathcal{P}_1, \mathcal{P}_2$, $\mathcal{P} = \text{conv}(\mathcal{P}_1, \mathcal{P}_2) = \{\lambda x_1 + (1-\lambda)x_2, \lambda \in [0,1], x_1 \in \mathcal{P}_1, x_2 \in \mathcal{P}_2\}$

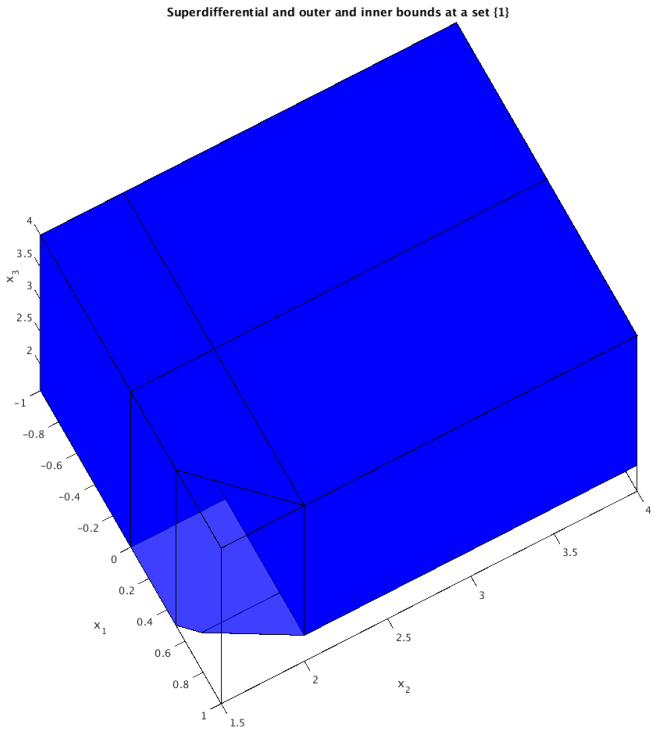

Figure 2.8: A visualization of the inner and outer bounds of the superdifferential. The submodular function here is $f : 2^{\{1,2,3\}} \to \mathbb{R}$, defined as $f(\emptyset) = 0, f(\{1\}) = 1, f(\{2\}) = 2, f(\{3\}) = 2, f(\{1,2\}) = 2.5, f(\{2,3\}) = 3, f(\{1,3\}) = 2.6, f(\{1,2,3\}) = 3$, and the superdifferential $\partial^f(X)$ is at $X = \{1\}$. The inner bounds $\partial^f_{i,1}(X)$ and $\partial^f_{i,2}(X)$ are shown in dark blue, and the superdifferential itself is shown in light blue. The outer white region is the outer bound $\partial^f_{\Delta(1,1)}(X)$.

polyhedron which has $\hat{g}_Y$ and $\check{g}_Y$ as its extreme points. The following lemma characterizes the inner bounds of the superdifferential:

Lemma 2.20. *Given a submodular function f,*

$$\partial^f_{i,3}(Y) \subseteq \partial^f_{i,2}(Y) \subseteq \partial^f_{i,(1,2)}(Y) \subseteq \partial^f(Y), \tag{2.64}$$

$$\partial^f_{i,3}(Y) \subseteq \partial^f_{i,1}(Y) \subseteq \partial^f_{i,(1,2)}(Y) \subseteq \partial^f(Y) \tag{2.65}$$

Proof. The proof of this lemma follows directly from the definitions of the supergradients, corresponding polyhedra and submodularity. □

Connections between the subdifferential and superdifferential at X

Finally we point out interesting connections between $\partial_f(X)$ and $\partial^f(X)$. Firstly, it is clear from the definitions that $\partial_f(X) \subseteq \partial_f^{\Delta(1,1)}(X)$ and $\partial^f(X) \subseteq \partial^f_{\Delta(1,1)}(X)$. Notice also that both $\partial_f^{\Delta(1,1)}(X)$ and $\partial^f_{\Delta(1,1)}(X)$ (from eqns (2.9) and (2.46) respectively) are simple polyhedra with a single extreme point,

$$\tilde{g}_X(j) = \begin{cases} f(j|X-j) & \text{if } j \in X \\ f(j|X) & \text{if } j \notin X \end{cases} \tag{2.66}$$

The point $\tilde{g}_X$, is in general, neither a subgradient nor a supergradient at X. However both the semidifferentials are contained within (different) polyhedra defined via $\tilde{g}_X$. An illustration of this is in Figure 2.9. The subdifferential $\partial_f(X)$ is the red polyhedron, while the superdifferential $\partial^f(X)$ is the blue polyhedron. Moreover, the light red and the light blue polyhedra are $\partial_f^{\Delta(1,1)}(X)$ and $\partial^f_{\Delta(1,1)}(X)$ respectively, defined on $X = \{1\}$.

Examples of Inner and Outer bounds for specific superdifferentials

In this section, we investigate the inner and outer bounds of specific instances. First consider the super differential at the emptiest $\partial^f(\emptyset)$. In this case, notice that all three supergradients are the same vector, i.e $\hat{g}_\emptyset = \check{g}_\emptyset = \bar{g}_\emptyset$, with individual elements being

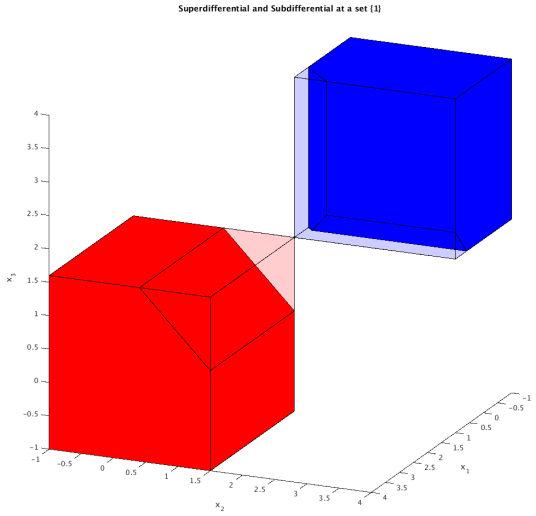

Figure 2.9: An Illustration to compare the relative positions of the sub and super differentials, defined on a submodular function $f : 2^{\{1,2,3\}} \to \mathbb{R}$, defined as $f(\emptyset) = 0, f(\{1\}) = 1, f(\{2\}) = 2, f(\{3\}) = 2, f(\{1,2\}) = 2.5, f(\{2,3\}) = 3, f(\{1,3\}) = 2.6, f(\{1,2,3\}) = 3$. The sub differentials appear in red, while the superdifferential is shown in blue, and are defined on $X = \{1\}$.

$\hat{g}_\emptyset(j) = \check{g}_\emptyset(j) = \bar{g}_\emptyset(j) = f(j), j \in V$. Hence the inner bounds are exactly the superdifferential, and $\partial^f_{i,1}(\emptyset) = \partial^f_{i,2}(\emptyset) = \partial^f_{i,3}(\emptyset) = \partial^f(\emptyset)$. Moreover, also observe that $\tilde{g}_\emptyset$ is also identical to these supergradients, and hence the outerbound $\partial^f_{\Delta(1,1)}(\emptyset) = \partial^f(\emptyset)$. In this case, all the inner and outer bounds are identical to the superdifferential. This phenomenon also occurs for the superdifferential of the grounds $\partial^f(V)$. For other sets, however, this does not hold, and the relationship between the bounds can be strict.

In the below, we analyze the inner and outer bounds on the superdifferentials for specific submodular functions.

Example 2.3. *Consider the case when $V = \{1, 2\}$. From the above, we know that the superdifferentials $\partial^f(\emptyset)$ and $\partial^f(\{1, 2\})$ are simple polyhedra and the inner and outer bounds are identical to the superdifferential itself. Consider $\partial^f(\{1\})$. Recall from Example 2.1 that the extreme points here are $\{f(\{1\}), f(\{2\})\}$ and $\{f(\{1\}|\{2\}), f(\{2\}|\{1\})\}$ respectively. Notice that $\hat{g} = \{f(\{1\}), f(\{2\})\}$ and $\check{g} = \{f(\{1\}|\{2\}), f(\{2\}|\{1\})\}$, and hence both these supergradients are extreme points of the superdifferential in two dimensions. Note that $\bar{g} = \{f(\{1\}|\{2\}), f(\{2\})\}$. Immediately, $\bar{g}$ lies in the interior of the superdifferential for a strictly submodular function*[5] *since (considering the inequalities governing $\partial^f(\{1\})$ from Example 2.1), $f(a|b) < f(\{1\}), f(\{2\}) > f(b|a)$ and $f(a|b) - f(\{2\}) = f(\{1, 2\}) - f(\{2\}) - f(\{2\}) < f(\{1\}) - f(\{2\})$ (this is true since $f(a, b) < f(\{1\}) + f(\{2\})$). Hence, in this case, $\partial^f_{i,3}(\{1\}) \subset \partial^f_{i,2}(\{1\}) \subset \partial^f_{i,(1,2)}(\{1\}) = \partial^f(\{1\})$ and $\partial^f_{i,3}(\{1\}) \subset \partial^f_{i,1}(\{1\}) \subset \partial^f_{i,(1,2)}(\{1\}) = \partial^f(\{1\})$. Similarly, observe that $\tilde{g}_a = \{f(\{1\}), f(b|\{1\}\}$ does not belong to $\partial^f(\{1\})$, when f is strictly submodular, since it violates the third inequality $f(\{1\}) - f(\{2\}|\{1\}) \leq f(\{1\}) - f(\{2\})$ (note this inequality does not hold since it would mean $f(\{2\}|\{1\}) \geq f(\{1\})$ which violates strict submodularity). Hence $\partial^f_{\Delta(1,1)}(\{1\}) \supset \partial^f(\{1\})$. The same phenomena is true for $\partial^f(\{2\})$.*

We can also consider the superdifferential in the three dimensional setting, when $V = \{1, 2, 3\}$. In this case, we consider an actual submodular function $f : 2^{\{1,2,3\}} \to \mathbb{R}$, defined

[5] A strict submodular function, is a submodular function, where all the defining inequalities act as equalities.

as $f(\emptyset) = 0, f(\{1\}) = 1, f(\{2\}) = 2, f(\{3\}) = 2, f(\{1,2\}) = 2.5, f(\{2,3\}) = 3, f(\{1,3\}) = 2.6, f(\{1,2,3\}) = 3$. Consider $\partial^f(\{1\})$. An illustration of this is in Figure 2.8. The inner bounds $\partial^f_{i,1}(X)$ and $\partial^f_{i,2}(X)$ are shown in dark blue, and the superdifferential itself is shown in light blue. The outer white region is the outer bound $\partial^f_{\Delta(1,1)}(X)$. In this case, it holds that $\partial^f_{i,3}(\{1\}) \subset \partial^f_{i,1}(\{1\}) \subset \partial^f_{i,(1,2)}(\{1\}) \subset \partial^f(\{1\}) \subset \partial^f_{\Delta(1,1)}(\{1\})$ and $\partial^f_{i,3}(\{1\}) \subset \partial^f_{i,2}(\{1\}) \subset \partial^f_{i,(1,2)}(\{1\}) \subset \partial^f(\{1\}) \subset \partial^f_{\Delta(1,1)}(\{1\})$ – i.e in other words, the subset relationship is strict in this case.

Superdifferentials of subclasses of submodular functions

While it is hard to characterize superdifferentials of general submodular functions, certain subclasses have some nice characterizations. An important such subclass if the class of $M^\natural$-concave functions [Murota, 2003]. These include a number of special cases like matroid rank functions, concave over cardinality functions etc. In some sense, these functions very closely resemble concave functions. In particular, one can maximize these functions in polynomial time [Murota, 2003]. These functions also admit simple characterizations of the superdifferential. In particular, the superdifferential of this class of functions can be represented in $O(n^2)$ inequalities. The following theorem provides a compact representation of the superdifferential of these functions.

Lemma 2.21. *Given a submodular function f which is $M^\natural$-concave on $\{0,1\}^V$, its superdifferential satisfies,*

$$\partial^f(X) = \partial^f_{\Delta(2,2)}(X) \tag{2.67}$$

In particular, it can be characterized via $O(n^2)$ inequalities.

Proof. This result follows easily from Theorem 6.61 in [Murota, 2003], where the authors show that for a $M^\natural$ convex function (which is supermodular), the subdifferential can be expressed by just considering sets Y satisfying $|X \backslash Y| \leq 1, |Y \backslash X| \leq 1$ (i.e of Hamming distance less than 2). Hence the superdifferential of a $M^\natural$ concave function (which is submodular) can

be expressed with the same number of inequalities, and the corresponding polyhedron is $\partial^f_{\Delta(2,2)}(X)$. □

2.6.3 Generalized Submodular Upper Polyhedron

In this section, we generalize the submodular upper polyhedron from Section 2.6.1. Define the *generalized submodular upper polyhedron* as the set of affine upper bounds of f:

$$\mathcal{P}^f_{\text{gen}} = \{(x, c), x \in \mathbb{R}^n, c \in \mathbb{R} : x(X) + c \geq f(X), \forall X \subseteq V\} \tag{2.68}$$

Again it is easy to see that $\mathcal{P}^f_{\text{gen}} \cap \{(x, c) : c = 0\} = \{(x, c) : x \in \mathcal{P}^f, c = 0\}$. In other words, the slice $c = 0$ of the *generalized submodular upper polyhedron* is the submodular upper polyhedron of f. Also note that the inequality at $X = \emptyset$ implies that $c \geq 0$. This polyhedron shall prove to be useful while defining the concave extensions of f. The generalized submodular upper polyhedron also has interesting connections with the superdifferentials. In particular,

Lemma 2.22. *Given a submodular function f, $(x, c) \in \mathcal{P}^f_{gen}$ lies on a face of the polyhedron if and only if there exists a set X such that $x \in \partial^f(X)$ and $c = f(X) - x(X)$.*

Proof. The proof of this Lemma is analogous to the one for the generalized submodular lower polyhedron. In particular, observe that (x, c) lies on a face of $\mathcal{P}^f_{\text{gen}}$ if and only if there exists a set X such that $x(X) + c = f(X)$ and for all $Y \subseteq V, x(Y) + c \leq f(Y)$. It directly then implies that $x \in \partial^f(X)$ and $c = f(X) - x(X)$. □

This then implies the following corollary:

Corollary 2.3. *Given a submodular function f, a point (x, c) is an extreme point of $\mathcal{P}^f_{gen}$, if and only if x is an extreme point of $\partial^f(X)$ for some set X.*

This implies an interesting characterization of a linear program over the generalized submodular upper polyhedron.

Lemma 2.23. *For submodular function f, and a $y \in \mathbb{R}^n$,*

$$\min_{(x,c)\in\mathcal{P}^f_{gen}} \langle x, y\rangle + c = \min\{\min_{x\in\partial^f(X)} \langle x, y\rangle + f(X) - x(X) \mid X \subseteq V\}. \tag{2.69}$$

Proof. To prove this result, we first show that $\min_{(x,c)\in\mathcal{P}^f_{\text{gen}}} \langle x, y\rangle + c \leq \min\{\min_{x\in\partial^f(X)} \langle x, y\rangle + f(X) - x(X) \mid X \subseteq V\}$. In order to show this, observe that for any set X, and point $x \in \partial^f(X)$, $(x, f(X) - x(X)) \in \partial^f(X)$. Hence the the second expression can be obtained by taking only a subset of the polyhedron $\mathcal{P}^f_{\text{gen}}$, and hence is a upper bound. We then show that $\min_{(x,c)\in\mathcal{P}^f_{\text{gen}}} \langle x, y\rangle + c \geq \min\{\min_{x\in\partial^f(X)} \langle x, y\rangle + f(X) - x(X) \mid X \subseteq V\}$, by invoking Corollary 2.3. The minimum on the LHS must occur at an extreme point of $\mathcal{P}^f_{\text{gen}}$, which implies that x is an extreme point of $\partial^f(X)$ for some set X, and $c = f(X) - x(X)$. Hence this implies that $\min_{(x,c)\in\mathcal{P}^f_{\text{gen}}} \langle x, y\rangle + c \geq \min\{\min_{x\in\partial^f(X)} \langle x, y\rangle + f(X) - x(X) \mid X \subseteq V\}$, since the LHS equals a particular instance of the RHS. This completes the proof. □

Unfortunately however, the generalized submodular upper polyhedron is no longer easy to characterize. This is related to the fact that the superdifferentials of a submodular function are not easy to characterize.

Lemma 2.24. *The generalized submodular upper membership problem for a submodular function f – i.e given a $x \in \mathbb{R}^n, c \in \mathbb{R}$, the problem whether $(x, c) \in \mathcal{P}^f_{gen}$ is NP hard for $c > 0$. Furthermore, for any $y \in \mathbb{R}^n$, solving a linear program over this polyhedron, i.e $\min_{(x,c)\in\mathcal{P}^f_{gen}} \langle x, y\rangle + c$ is also NP hard.*

Proof. The first part of the result follows from the fact that asking whether $(x, c) \in \mathcal{P}^f_{\text{gen}}$ is equivalent to asking whether $\max_{X\subseteq V}[f(X) - x(X) - c] \leq 0$, which can be rewritten as $\max_{X\subseteq V}[f(X) - x(X)] \leq c$. This is the decision version of submodular maximization, which is NP hard. The second part follows directly from the first since the membership problem on a polyhedron is equivalent to a linear program over this polyhedron [Grotschel et al., 1984, Schrijver, 2003]. □

We can also prove the second part, by observing that solving a linear program over the generalized submodular polyhedron is NP hard, is equivalent to computing the concave

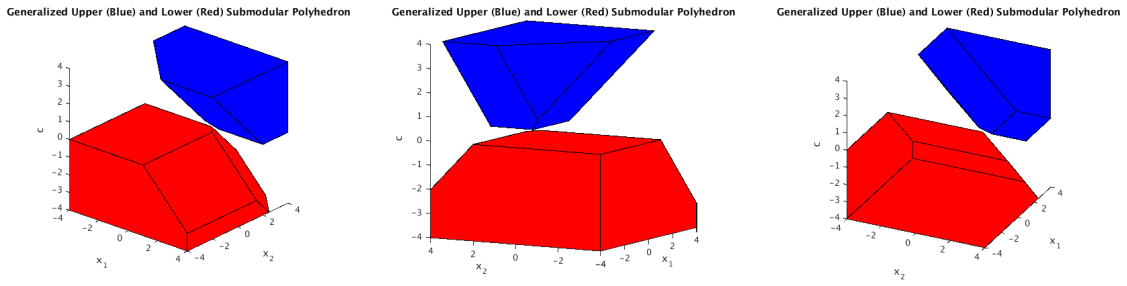

Figure 2.10: Three different views of the generalized submodular upper (shown in blue) and lower (shown in red) polyhedron, for a submodular function $f : 2^{\{1,2\}} \rightarrow \mathbf{R}$, with $f(\emptyset) = 0, f(\{1\}) = 1, f(\{2\}) = 2, f(\{1,2\}) = 2.5$. Notice that all the extreme points of the generalized submodular lower polyhedron are on the plane $c = 0$ – the two extreme points are $\{1, 1.5, 0\}$ and $\{0.5, 2, 0\}$. In the generalized submodular upper polyhedron, however, one of the extreme points is on $c = 0$ (this extreme point is $\{1, 2, 0\}$), while the other extreme point is $\{0.5, 1.5, 0.5\}$ (here $c > 0$).

extension of a submodular function (we show this in Lemma 2.82). Computing this concave extension, however, is NP hard [Dughmi, 2009, Vondrák, 2007].

Recall that in the case of the generalized submodular lower polyhedron, the extreme points of this polyhedron were identical to the extreme points of the submodular lower polyhedron (i.e all extreme points of the generalized submodular lower polyhedra occurred when $c = 0$), which meant that the linear program over the two polyhedra was the same. This is not the case in the generalized submodular upper polyhedra. To see this, consider a simple example with $V = \{1, 2\}$.

Example 2.4. *First consider the generalized submodular lower polyhedra when* $V = \{1, 2\}$.

$$\mathcal{P}^f_{gen} = \{(x, c) \in \mathbb{R}^3 : c \leq 0, \tag{2.70}$$

$$x_1 + c \leq f(\{1\}), \tag{2.71}$$

$$x_2 + c \leq f(\{2\}), \tag{2.72}$$

$$x_1 + x_2 + c \leq f(\{1, 2\})\} \tag{2.73}$$

Then, it is immediate that the only extreme points in this case are $\{f(\{1\}), f(\{2\}|\{1\}), 0\}$ *and* $\{f(\{1\}|\{2\}), f(\{2\}), 0\}$*, which are obtained by setting Eqns* (2.70)*,* (2.71)*,* (2.73) *and Eqns* (2.70)*,* (2.72)*,* (2.73) *as equalities. The extreme points in this case, are a direct product between the extreme points of* $\mathcal{P}_f$ *and* $c = 0$*. Hence all extreme points lie on the face* $c = 0$*.*

This is not the case for the generalized submodular upper polyhedron. Consider again the example with $V = \{1, 2\}$.

Example 2.5. *The generalized upper submodular polyhedron in this case is,*

$$\mathcal{P}^f_{gen} = \{(x, c) \in \mathbb{R}^3 : c \geq 0, \tag{2.74}$$

$$x_1 + c \geq f(\{1\}), \tag{2.75}$$

$$x_2 + c \geq f(\{2\}), \tag{2.76}$$

$$x_1 + x_2 + c \geq f(\{1, 2\})\} \tag{2.77}$$

Then, it is immediate that the only extreme points in this case are $\{f(\{1\}), f(\{2\}), 0\}$ *and* $\{f(\{1\}|\{2\}), f(\{2\}|\{1\}), f(\{1\}) + f(\{2\}) - f(\{1, 2\})\}$*, which are obtained by setting Eqns* (2.74)*,* (2.75)*,* (2.76) *and Eqns* (2.75)*,* (2.76)*,* (2.77) *as equalities (setting the other combination of inequalities as equalities does not give extreme points). Hence while one of the extreme points here is* $\{f(\{1\}), f(\{2\}), 0\}$*, which is the direct product between* $\mathcal{P}^f$ *and* $c = 0$*, the other extreme point occurs at* $c > 0$ *(when* f *is strictly submodular).*

Inner and outer bounds on the generalized submodular upper polyhedron

In a manner similar to the superdifferential, we can provide inner and outer bounds of the generalized submodular upper polyhedron. In particular, let $g^X \in \partial^f(X)$ be a supergradient, that is easy to obtain. Then, $m^X(Y) = f(X) + g_X(Y) - g^X(X)$ is a modular upper bound of $f(Y), \forall Y \subseteq V$. Given a set $\{g^X | X \subseteq V\}$ of such supergradients, we may define a polytope:

$$\mathcal{P}^f_{g,gen} = \text{conv-hull}\{(g_Y, f(Y) - g_Y(Y)), \forall Y \subseteq V\} \tag{2.78}$$

It is not hard to see that $\mathcal{P}^f_{g,gen} \subseteq \mathcal{P}^f_{\text{gen}}$ since it is formed using some specific supergradients. Moreover, many of these bounds can be combined together by taking the convex hull of these polyhedra. We shall be interested by the polytopes formed by the supergradients $\hat{g}_Y, \check{g}_Y$ and $\bar{g}_Y$ (which we call $\mathcal{P}^f_{1,gen}, \mathcal{P}^f_{2,gen}$ and $\mathcal{P}^f_{3,gen}$) and supergradients related to these. These bounds, as we shall see, have interesting connections to concave extensions (which we shall describe Section 2.7) and ultimately to submodular maximization. One can similarly also define outer bounds of the generalized submodular upper polyhedron, by considering only a subset of inequalities defining $\mathcal{P}^f_{\text{gen}}$. We do not pursue this here, however.

2.7 Concave extensions of a submodular function

Following the characterizations of the convex extensions of a submodular function, we can define the concave extensions also two viewpoints, one in the *distributional* setting and another in the *polyhedral* setting. These results follow in the lines of the results shown for the convex extensions.

2.7.1 Polyhedral characterization of the concave extension

Similar to the convex extension, the concave extension of any set function (not necessarily submodular) can be seen as the pointwise supremum of concave functions which lower bound the set function [Dughmi, 2009]. More clearly, let

$$\Psi_f = \{\psi : \psi \text{ is concave in } [0,1]^V \text{ and } \psi(1_X) \geq f(X), \forall X \subseteq V\}. \tag{2.79}$$

Then define:

$$\hat{f}(w) = \min_{\psi \in \Psi_f} \psi(w). \tag{2.80}$$

Following arguments similar to the convex extension, eqn. (2.80) can be expressed as a linear program over the generalized submodular upper polyhedron.

Lemma 2.25. *The concave extension in Eqn.* (2.80) *in any set function f can be expressed as:*

$$\hat{f}(w) = \min_{(y,c) \in \mathcal{P}^f_{gen}} \langle y, w \rangle + c, \forall w \in [0,1]^{|V|} \tag{2.81}$$

Proof. The proof of this Lemma follows along the lines of the proof of Lemma 2.6. For a given w, let $\hat{\psi}$ be an argmin in Eqn. (2.80). Then since $\hat{\psi}$ is a concave function in $[0,1]^V$, there exists a supergradient $x \in \mathbb{R}^n$ at w and value d, such that $\langle x, y \rangle + d \geq \hat{\psi}(y), \forall y$ and $\langle x, w \rangle + d = \hat{\psi}(w)$. In other words, $\langle x, y \rangle + d$ is a linear upper bound of $\hat{\psi}(y)$, tight at w. Hence $\hat{f}(w) = \langle x, w \rangle + d$. Finally notice that $(x, d) \in \mathcal{P}^f_{\text{gen}}$ since $x(X) + d \geq \hat{\psi}(1_X) \geq f(X), \forall X \subseteq V$. □

Unlike the case of the convex extension, however, this is not equivalent to an optimization over the submodular upper polyhedron. Moreover, this expression requires solving a linear program over the submodular upper polyhedron, it follows from Theorem 2.24 that obtaining the concave extension is NP hard. We shall revisit this result, in the next subsection, while investigating the distributional characterization.

Interestingly we can define a number of concave extensions based on relaxations of the polyhedral representation. In particular, consider the inner approximations of the generalized submodular upper polyhedron $\mathcal{P}^f_{g,gen}$, defined via a particular supergradient g. Instead of minimizing over all affine upper bounds, we can minimize only over a particular class of modular upper bounds. Then, we can define the following form of a concave extension:

$$\hat{f}_g(w) = \min_{(y,c) \in \mathcal{P}^f_{g,gen}} [\langle y, w \rangle + c] = \min_{Y \subseteq V} \ [\langle y, g_Y \rangle + f(Y) - g_Y(Y)], \quad \forall w \in [0,1]^{|V|} \tag{2.82}$$

In particular, the above turns the linear program into a discrete optimization problem. Moreover, the concave extension $\hat{f}_g$ is guaranteed to be an upper bound of $\hat{f}$. We can define

three variants of these extensions using the supergradients $\hat{g}_Y, \check{g}_Y$ and $\bar{g}_Y$, which we call $\hat{f}_1, \hat{f}_2$ and $\hat{f}_3$. It is easy to see that the concave extension $\hat{f}_3$ can in fact, be obtained in polynomial time, since it involves submodular function minimization.

The class of concave extensions from Eqn. (2.82) has some very interesting connections to a form of concave extension proposed in [Vondrák, 2007] for monotone submodular functions. In particular, where [Vondrák, 2007] defined a concave function $\hat{f}_g$:

$$\hat{f}_g(x) = \min\{[f(Y) + \sum_{j \in V} x(j) f(j|Y)] | Y \subseteq V\} \tag{2.83}$$

This extension, can be seen as a special case of Eqn. (2.82) with a particular set of supergradients g_Y for a monotone submodular function, defined as:

$$g_Y(j) = \begin{cases} 0 & \text{if } j \in Y \\ f(j|Y) & \text{if } j \notin Y \end{cases} \tag{2.84}$$

This supergradient is related to the supergradient $\check{g}_Y$ except replacing the elements for $j \in Y$ with 0. For a monotone submodular function, this continues to remain a supergradient. This form of concave extension is NP hard to evaluate (see Section 3.7 in [Vondrák, 2007]). This shall still be useful in obtaining approximate maximizers in certain special cases (we investigate this in Section 2.7.4).

2.7.2 Distributional characterization of the concave extension

An alternate and equivalent characterization can be provided through the distributional lens.

Lemma 2.26. *Denote Λ_w as the set:*

$$\Lambda_w = \{\lambda_S, S \subseteq V : \sum_{S \subseteq V} \lambda_S 1_S = w, \sum_{S \subseteq V} \lambda_S = 1 \tag{2.85}$$

The concave extension from Eqn. (2.81) *then can also be represented as:*

$$\hat{f}(w) = \max_{\lambda \in \Lambda_w} \lambda_S f(S) \tag{2.86}$$

The proof of the above follows on similar lines as the convex extension, and is shown in [Dughmi, 2009]. Unfortunately, unlike the convex extension, this extension is NP hard to evaluate and optimize over.

Proposition 2.3. *Given a submodular function f, it is NP hard to evaluate and optimize $\hat{f}$.*

This result was shown in [Vondrák, 2007].

Similar to the polyhedral characterization, we can relax the distributional characterization to consider specific distributions. In particular, we can obtain the multilinear extension, through a particular distribution, $\lambda_X = \prod_{i \in X} x_i \prod_{i \notin X} (1 - x_i)$.

$$\tilde{f}(x) = \sum_{X \subseteq V} f(X) \prod_{i \in X} x_i \prod_{i \notin X} (1 - x_i) \tag{2.87}$$

It is not hard to see that this forms a lower bound on the concave extension $\hat{f}$. This extension, unlike most extensions seen earlier, however, is not concave. Similar to the concave extension it is hard to evaluate this extension, and requires sampling [Vondrák, 2007].

2.7.3 Concave extensions of subclasses of submodular functions

While the concave extension $\hat{f}(x)$ is NP hard to compute in general, it can be done efficiently for certain subclasses of submodular functions. These include, for example, sums of weighted matroid rank functions [Vondrák, 2007], and the class of $M^\natural$-concave functions (c.f. Theorem 6.42 in [Murota, 2003]).

2.7.4 Concave extensions and submodular maximization

The concave extensions and the multilinear extension have interesting connections to submodular maximization. The following lemma from [Vondrák, 2007] connects many of these extensions:

Lemma 2.27. *[Vondrák, 2007] For every monotone submodular function f, $\hat{f}(x) \geq \tilde{f}(x) \geq (1 - \frac{1}{e})\hat{f}(y)$.*

It is also easy to relate all the three extensions of a submodular function, viz. the convex extension, the concave extension and the multilinear extension.

Lemma 2.28. *Given a submodular function, it holds that*

$$\hat{f}(x) \geq \tilde{f}(x) \geq \breve{f}(x) \tag{2.88}$$

Proof. The proof of this result follows directly from the distributional characterization of the convex and concave extensions. Note that the multilinear extension is a particular distribution, the concave extension is a pointwise maximum over all distributions, while the convex extension is a pointwise minimum over these distributions. □

The facts above were used in providing a relaxation based algorithm for maximizing a subclass of submodular functions efficiently [Vondrák, 2007]. This relaxation based algorithm, maximizes the concave extension $\hat{f}(x)$, which though NP hard to optimize in general, can be maximized in certain special cases. The particular special case which is considered in [Vondrák, 2007], is the class of weighted matroid rank functions for which the concave extension has a simple form. Furthermore, a simple pipage rounding trick ensures no integrality gap with respect to the multilinear extension, thus providing a $1 - \frac{1}{e}$ approximation algorithm for the problem of maximizing a monotone submodular function subject to a matroid constraint. Furthermore, later, a conditional gradient style algorithm, also called the continuous greedy algorithm [Vondrák, 2008] directly optimizes the multi-linear extension, thereby providing a general $1 - 1/e$ approximation algorithm for monotone submodular maximization subject to matroid constraints. This was later extended to the non-monotone case by [Chekuri et al., 2011].

2.8 Optimality Conditions for submodular maximization

Just as the subdifferential of a submodular function provides optimality conditions for submodular minimization, the superdifferential provides the optimality conditions for submodular maximization.

2.8.1 Unconstrained submodular maximization

In this section, we consider the general problem of unconstrained submodular maximization:

$$\max_{X \subseteq V} f(X) \tag{2.89}$$

Given a submodular function, we can give the KKT like conditions for submodular maximization, in the following theorem:

Lemma 2.29. *For a submodular function f, a set A is a maximizer of f, if $\boldsymbol{0} \in \partial^f(A)$.*

However as expected, finding the set A, with the property above, or even verifying if for a given set A, $0 \in \partial^f(A)$ are both NP hard problems (from Proposition 2.18). However thanks to the submodularity, we show that the outer bounds on the superdifferential provide approximate optimality conditions for submodular maximization. Moreover, unlike the superdifferential, these bounds are easy to obtain.

Proposition 2.4. *For a submodular function f, if $0 \in \partial^f_{(1,1)}(A)$ then A is a local maxima of f (that is, $\forall B \supseteq A, f(A) \geq f(B), \& \forall C \subseteq A, f(A) \geq f(C)$). Furthermore, if we define $S = \operatorname{argmax}_{X \in \{A, V \setminus A\}} f(A)$, then $f(S) \geq \frac{1}{3} OPT$ where OPT is the optimal value.*

The above result is interesting observation, since a very simple outer bound on the superdifferential, leads us to an approximate optimality condition for submodular maximization. The local optimality condition follows directly from the definition of $\partial^f_{(1,1)}(A)$ and the approximation guarantee follows from Theorem 3.4 in [Feige et al., 2011b].

We can also provide an interesting sufficient condition for the maximizers of a submodular function.

Lemma 2.30. *If for any set A, $\boldsymbol{0} \in \partial^f_{i,(1,2)}(A)$, then A is the global maxima of the submodular function. In particular, if a local maxima A is found (which is typically easy to do), it is guaranteed to be a global maxima, if $\boldsymbol{0} \in \partial^f_{i,(1,2)}(A)$.*

Proof. This proof follows from the fact that $\partial^f_{i,(1,2)}(A) \subseteq \partial^f(A) \subseteq \partial^f_{(1,1)}(A)$. Thus, if $\mathbf{0} \in \partial^f_{i,(1,2)}(A)$, it must also belong to $\partial^f(A)$, which means A is the global optimizer of f. $\square$

We can also provide similar results for constrained submodular maximization as we show in the following subsection.

2.8.2 Constrained submodular maximization

We consider here a constrained submodular maximization problem, with $\mathcal{C}$ representing a set of sets, also called *combinatorial constraints*.

$$\max_{X \in \mathcal{C}} f(X) \tag{2.90}$$

For example $\mathcal{C}$ could represent a cardinality constraint $\{X \subseteq V : |X| \leq m\}$, or a spanning tree, matching, s-t path etc. Another common type of constraints are matroid constraints. Denote $\mathcal{I}$ is the independent set of a matroid $\mathcal{M}$. Then $\mathcal{C} = \{X : X \in \mathcal{I}\}$ is a matroid constraint. Similarly $\mathcal{C} = \{X \subseteq V : c(X) \leq B\}$ represents a knapsack constraint.

Then in this case we modify the definition of $\partial^f_{\mathcal{C}}(A)$ as follows:

$$\forall A \in \mathcal{C}, \partial^f_{\mathcal{C}}(A) = \{x \in \mathbb{R}^n : f(X) - f(A) \leq x(X) - x(A), \forall X \in \mathcal{C}\} \tag{2.91}$$

In other words we only consider the sets belonging to the constraints. Then we can trivially define a KKT like optimality condition for this problem:

Lemma 2.31. *For a submodular function f, a set A is a maximizer of the problem $\max_{X \in \mathcal{C}} f(X)$, if $0 \in \partial^f_{\mathcal{C}}(A)$.*

Clearly finding the set above is NP-hard. However, similar to the unconstrained setting, we show that, in a number of cases, approximating the superdifferential can lead to polynomial time algorithms for constrained submodular maximization with worst case approximation guarantees. Consider the following scenarios:

Constrained Monotone Submodular function Maximization

Consider here a case where f is a monotone submodular function, and $\mathcal{C}$ is the constraint that the set belongs to the intersection of the independence sets of k matroids.

Let $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_k$ represent k matroids, with independence sets $\mathcal{I}_1, \mathcal{I}_2, \cdots, \mathcal{I}_k$. Then $\mathcal{C} = \{X : X \in \cap_{i=1}^{k} \mathcal{I}_i\}$. Again, analogous to the superdifferential $\partial^f_{\mathcal{C}}$, we can also define the outer-bounds $\partial^f_{\mathcal{C},(k,l)}$ restricted to $\mathcal{C}$.

Then we have the following observation, for the problem of monotone submodular maximization subject to matroid constraints.

Observation 2.1. *Given a monotone submodular function f and a constraint $\mathcal{C} = \cap_{i=1}^{k} \mathcal{I}_i$, for any set $A \in \mathcal{C}$,*

1. *if $\mathbf{0} \in \partial^f_{\mathcal{C},(2,k+1)}(A)$, then $f(A)$ is guaranteed to be at least $\frac{1}{k+1}$ times the optimal value. In particular, for the special case of monotone submodular maximization subject to a single matroid constraint, for any set $A \in \mathcal{C}$, if $\mathbf{0} \in \partial^f_{\mathcal{C},(2,2)}(A)$, then $f(A)$ is guaranteed to be at least $\frac{1}{2}$ times the optimal value.*

2. *If $k = 1$ and $\mathcal{C}$ is a cardinality (uniform matroid constraint $\{X : |X| \leq m\}$), for any $r > 0$, a set A satisfying $\mathbf{0} \in \partial^f_{\mathcal{C},(r+1,r+1)}$ is guaranteed to have an approximation guarantee no worse than $\frac{m}{2m-r}$.*

The first part of the observation (i.e point 1) follow directly from Corollary 2.4 in [Lee et al., 2009a]. In the case of $k = 1$, the same result was shown in [Fisher et al., 1978]). Moreover, it was also shown in [Fisher et al., 1978], that for the problem of monotone submodular maximization subject to $k > 1$ matroid constraints, the approximate optimality conditions $0 \in \partial^f_{\mathcal{C},(2,2)}(A)$, can be arbitrarily bad, thus requiring 'higher order' optimality conditions. We also remark that when $r = 1$ (i.e submodular maximization subject to a single matroid constraint), this the same approximation factor can be obtained by the simple greedy algorithm [Nemhauser et al., 1978]. The second part of the above observation (which is submodular maximization subject to cardinality constraints) follows from Theorem 8 in [Filmus, 2013]. In the case when $r = 1$, the condition $\mathbf{0} \in \partial^f_{\mathcal{C},(2,2)}$ provides a guarantee of $m/(2m-1)$ which is a slight improvement of $1/2$ in the special case of cardinality constraints. An interesting observation is that with better forms of local optima (i.e the condition

$\mathbf{0} \in \partial^f_{\mathcal{C},(r+1,r+1)}$, is a local optima upto size r), imply better approximation guarantees to this problem. The main insight in these results, is that the local optima which are obtained through the local search algorithms, can all be viewed as approximate *optimality* conditions on the superdifferential of a submodular function.

The approximation factor for $k \geq 2$ matroids can actually be improved as shown in [Lee et al., 2009b].

Observation 2.2. *Given a maximization problem of a monotone submodular function f subject to $k > 1$ matroid constraints, for a set $A \in \mathcal{C}$, if $\mathbf{0} \in \partial^f_{\mathcal{C},(p+1,kp+1)}(A)$, then $f(A)$ is guaranteed to be at least $\frac{1}{k+1/p}$ times the optimal value. In particular, for a special case of monotone submodular maximization subject to 2 matroid constraints, for any set $A \in mathcalC$, if $\mathbf{0} \in \partial^f_{\mathcal{C},(p+1,2p+1)}(A)$, then $f(A)$ is guaranteed to be at least $\frac{1}{2+1/p}$ times the optimal value.*

This result is the best known result for $k > 1$ matroids and it follows from Corollary 3.1 in [Lee et al., 2009b]. Similar to the monotone setting, the observation here is that the local optimality conditions from [Lee et al., 2009b], can be viewed easily as approximate optimality conditions on the superdifferential of f.

Constrained Symmetric Submodular function Maximization

Finally we consider the case of non-monotone submodular maximization subject to k matroid constraints. Consider the case of symmetric submodular functions.

Observation 2.3. *Given a symmetric submodular function f,*

1. *If the constraint $\mathcal{C} = \cap_{i=1}^k \mathcal{I}_i$, for any set $A \in \mathcal{C}$ satisfying $\mathbf{0} \in \partial^f_{(2,k+1)}(A)$, $f(A)$ is guaranteed to be at least $\frac{1}{k+2}$ times the optimal value.*

2. *If $\mathcal{C}$ is the constraint that the set be a base of a matroid, any set A satisfying $0 \in \partial^f_{\mathcal{C},(2,2)}(A)$ is guaranteed to have a valuation at least $1/3$ of the optimal.*

All the results in this proposition, follow directly from the results in [Lee et al., 2009a]. The first part follows from Theorem 2.8, while the second part is implied by Theorem 5.1.

Constrained Non-monotone Submodular function Maximization

Finally, we provide an approximation bound for non-monotone submodular maximization.

Observation 2.4. *Given a non-monotone submodular function f, and $\mathcal{C}$ is a cardinality (uniform matroid constraint $\{X : |X| \leq m\}$), for any $r > 0$, a set A satisfying $\mathbf{0} \in \partial^f_{\mathcal{C},(r+1,r+1)}$ is guaranteed to have an approximation guarantee no worse than $\frac{r}{2m-r}$.*

This result follows directly from Theorem 8 in [Filmus, 2013]. The best bounds for non-monotone submodular maximization, require running several iterations of local search procedures. In particular, the procedure of [Lee et al., 2009a] runs $k+1$ local search procedures to obtain a $1/(k+2+1/k)$ approximation algorithm for non-monotone submodular maximization subject to a single matroid constraint. When $k = 1$, running two rounds of this local search procedure results in a $1/4$ approximation. The individual local search procedures here obtains a set A satisfying $\mathbf{0} \in \partial^f_{\mathcal{C},(2,k)}(A)$.

2.9 Concave Characterizations: Discrete Separation Theorem and Fenchel Duality Theorem

In Section 2.5, we investigated forms of the Discrete Separation Theorem, Fenchel Duality Theorem and Minkowski Sum Theorem for submodular functions and their associated polyhedra when seen from the convex perspective. We analyze forms of the Discrete Separation Theorem and Fenchel Duality Theorem for submodular functions and their associated polyhedra, now from the concave perspective.

2.9.1 Discrete Separation Theorem

We first show that a restricted version of the Discrete Separation Theorem holds. In particular, we investigate the concave variant of the Discrete Separation Theorem, and show that under

some restricted settings, given a submodular function f and a supermodular function g with $f(X) \leq g(X), \forall X$, there exists a modular function h such that $f(X) \leq h(X) \leq g(X)$. Then the following Lemma holds:

Lemma 2.32. *Given a submodular function f and a supermodular function g, such that $f(X) \leq g(X), \forall X \subseteq V$, such that $f(\emptyset) = g(\emptyset)$ or $f(V) = g(V)$, there exists a modular function h such that $f(X) \leq h(X) \leq g(X), \forall X \subseteq V$. Moreover, when f and g are integral (and satisfy the conditions above), there exists an integral h satisfying the above.*

Proof. Assume first that $f(\emptyset) = g(\emptyset)$. Then Let $h(X) = f(\emptyset) + \sum_{j \in X} f(j|\emptyset)$. Then the following chain of inequalities hold:

$$f(X) \leq h(X) = f(\emptyset) + \sum_{j \in X} f(j|\emptyset) \leq g(\emptyset) + \sum_{j \in X} g(j|\emptyset) \leq g(X) \tag{2.92}$$

Since $f(j|\emptyset) = f(j) - f(\emptyset) \leq g(j) - g(\emptyset) = g(j|\emptyset)$. The rest of the inequalities follow from submodularity (and super modularity) of f (and g). The result for when $f(V) = g(V)$ analogously follows by considering the functions $f(V \backslash X)$ and $g(V \backslash X)$ which are submodular and supermodular respectively. □

The Lemma above can in fact be generalized. In particular, it is not hard to see that the result goes through whenever $\text{argmin}_X [g(X) - f(X)]$ is either $\emptyset$ or V. In general, we can also provide another restricted form of the Discrete Separation Theorem:

Lemma 2.33. *Given a submodular function f and a supermodular function g, such that $f(X) \leq g(X), \forall X \subseteq V$, let $A = \text{argmin}_X [g(X) - f(X)]$. Then there exists a modular function h such that $f(X) \leq h(X) \leq g(X), \forall X : X \subseteq A$ or $X \supseteq A$.*

Proof. The proof of this result is analogous to the earlier one. Given $A = \text{argmin}_X \, g(X) - f(X)$, assume without loss of generality that $f(A) = g(A)$. The reason, this holds without loss of generality is that, suppose that there exists a vector h which separates f and g when $f(A) = g(A)$, the vector will continue to separate f and g when $f < g$. Then define $h(X) = f(A) + \sum_{j \in X \backslash A} f(j|A) - \sum_{j \in A \cap X} f(j|A \backslash j)$. It is easy to see that $f(X) \leq$

$h(X), \forall X \subseteq A, X \supseteq A$. Moreover, $h(X) = f(A) + \sum_{j \in X \setminus A} f(j|A) - \sum_{j \in A \cap X} f(j|A \setminus j) \leq g(A) + \sum_{j \in X \setminus A} g(j|A) - \sum_{j \in A \cap X} g(j|A \setminus j) \leq g(X), \forall X \subseteq A, X \supseteq A$. □

The Discrete Separation Theorem may not, however, hold under the most general conditions on f and g. However, the do hold for certain subclasses. For example, if f is a $M^\natural$-concave function (which is submodular), and g is a $M^\natural$-convex function (which is supermodular), the discrete separation theorem always holds (c.f Theorem 8.15 in [Murota, 2003]).

2.9.2 Fenchel Duality Theorem

Finally we show that a version of the Fenchel duality Theorem also holds in certain restricted cases. Given a submodular function f (or equivalently supermodular function g), define the concave Fenchel dual functions f_* (or equivalently g_*) as:

$$f_*(y) = \min_{X \subseteq V} y(X) - f(X), \quad g_*(y) = \max_{X \subseteq V} y(X) - g(X). \tag{2.93}$$

The Fenchel duals f_* and g_* are concave and convex functions respectively. Unlike the convex Fenchel duals, obtaining these expressions exactly is NP hard, since it corresponds to submodular maximization. These can however be approximately obtained upto constant factors. The following Lemma then gives a restricted version of Fenchel Duality Theorem:

Lemma 2.34. *Given a submodular function f and a supermodular function g such that the Discrete Separation Theorems hold,*

$$\max_{X \subseteq V} f(X) - g(X) = \min_x g_*(x) - f_*(x) \tag{2.94}$$

Further if f and g are integral (and satisfy the DST), the maximum on the right hand side is attained by an integral vector x.

Proof. The proof of this result follows directly from Theorem 4 of [Fujishige and Narayanan, 2005]. In particular, [Fujishige and Narayanan, 2005] show that given set functions f and g such that the discrete separation theorem holds, the Fenchel duality theorem will also hold for this pair of functions f and g. □

Unlike the Fenchel Duality Theorem from the convex perspective, the result above may not hold in the most general setting. Moreover, if the functions f and g are $M^\natural$-concave and $M^\natural$-convex respectively, the Fenchel duality theorem always holds (c.f Theorem 8.21 in [Murota, 2003]).

2.9.3 Minkowski Sum Theorem

Analogous to the results above, we show a certain restricted form of the Minkowski Sum Theorem.

Lemma 2.35. *Given two submodular functions f_1 and f_2, it holds that that (the addition of the polyhedra below corresponds to a point-wise addition):*

$$\mathcal{P}^{f_1+f_2} = \mathcal{P}^{f_1} + \mathcal{P}^{f_2} \tag{2.95}$$

Similarly, $\partial^{f_1+f_2}(\emptyset) = \partial^{f_1}(\emptyset) + \partial^{f_2}(\emptyset)$ and $\partial^{f_1+f_2}(V) = \partial^{f_1}(V) + \partial^{f_2}(V)$.

Proof. This result follows directly from the definitions. In particular, it is easy to see that the extreme points of these polyhedra can be explicitly characterized by a submodular function. For example, the polyhedron $\mathcal{P}^{f_1}$ has a single extreme point defined by the vector $f_1(j), j \in V$. Similarly, the extreme point of $\mathcal{P}^{f_2}$ is $f_2(j), j \in V$, and the extreme point of $\mathcal{P}^{f_1+f_2}$ is $f_1(j) + f_2(j), j \in V$, and hence the Minkowski sum theorem holds. □

Unlike the Minkowski Sum theorem on the subdifferential and submodular polyhedron, the result above may not hold for the superdifferential of any arbitrary set $\partial^f(X)$ and the generalized submodular upper polyhedron $\mathcal{P}^f_{\text{gen}}$. They do hold, however, for certain subclasses of submodular functions, like the class of $M^\natural$-concave functions. This fact easily follows from Theorem 3 in [Fujishige and Narayanan, 2005], and the fact that $M^\natural$-concave functions satisfy the Fenchel duality theorem.

2.10 Conclusions and Open Problems

In this manuscript, we investigated several connections between convex and concave aspects of submodular functions. We provided characterizations of the superdifferentials, concave extensions and separation and duality theorems related to concave aspects of a submodular function, and connected these new results to existing results on the convex aspects of submodular functions. To our knowledge, this is the first work in this direction. We also show how for specific subclasses of submodular functions, like the class of $M^\natural$-concave set functions, this characterization is exact, while for other submodular functions, this can be done approximately.

We also leave a few open problems.

- Are there are other subclasses of submodular functions (apart from the class of $M^\natural$-concave set functions, for which the concave aspects, like the superdifferentials, concave extensions and characterizations like the discrete separation theorem, Fenchel duality theorem etc. can be provided exactly. In particular, we saw that the $M^\natural$-concave set functions, satisfy the property that $\partial^f(X) = \partial^f_{\Delta(2,2)}(X)$. An interesting question is whether there are other interesting subclasses of submodular functions, which satisfy similar conditions on their superdifferential. Characterizing such functions, could have a direct consequence on maximizing these subclasses of submodular functions.

- In section 2.8, we investigated optimality conditions related to submodular maximization, and its connection to the superdifferential. An interesting open problem is if this characterization could provide insight into algorithms for submodular maximization, and conditions when submodular maximization can be done exactly. Moreover, it also is interesting that approximating the superdifferential provides different approximation algorithms for submodular maximization. It will be interesting if there is a principled relationship between these two.

- In Section 2.9, we study the Fenchel duality Theorem, Discrete Separation Theorem

and Minkowski sum theorem, and show that these results hold under restricted settings. An open question is if Edmonds intersection theorem (cf. Section 4.1 in Fujishige [2005]) also holds under certain restricted settings.

- Finally, thanks to the Minkowski sum theorem, the Lovász extension of a submodular function satisfies that $\breve{f}_1 + f_2(x) = \breve{f}_1(x) + \breve{f}_2(x)$, i.e the Lovász extension of a sum of two submodular functions is equal to the sum of the individual Lovász extensions. An open problem is whether this relation holds (under restricted settings possibly) for the concave extensions.

Chapter 3

SUBMODULAR FUNCTION MINIMIZATION

3.1 Introduction

In this chapter, we address the problem of submodular function minimization:

$$\text{Problem 1: } \min_{X \in \mathcal{C}} f(X), \tag{3.1}$$

where $f : 2^V \to \mathbb{R}$ is a discrete set function on subsets of a ground set $V = \{1, 2, \cdots, n\}$, and $\mathcal{C} \subseteq 2^V$ is a family of feasible solution sets. The set $\mathcal{C}$ could express, for example, that solutions must be an independent set in a matroid, a limited budget knapsack, or a cut (or spanning tree, path, or matching) in a graph. Without making any further assumptions about f, the above problems are trivially worst-case exponential time and moreover inapproximable. If we assume that f is submodular, however, then in many cases the above problems can be approximated and in some cases solved exactly in polynomial time.

3.1.1 Unifying Algorithmic Framework

In this chapter, we provide a class of combinatorial algorithms for Problem 1. The first class of algorithms is a Majorization-Minimization style (MM) algorithm, which is scalable and practical. This framework makes two contributions. For unconstrained minimization, we obtain new nontrivial bounds on the lattice of minimizers, thereby reducing the possible space of candidate minimizers. This method easily integrates into any other exact minimization algorithm as a preprocessing step to reduce running time. In the constrained case, we obtain practical algorithms with bounded approximation factors. We observe these algorithms to be empirically competitive to more complicated ones. While the Majorization-Minimization scheme of algorithms are not new to machine learning. most of these algorithms, however, lack

theoretical guarantees. This chapter shows, by contrast, that for submodular optimization, MM algorithms have strong theoretical properties and empirically work very well. In this thesis, we shall denote Majorization-Minimization by MMin. As a whole, this framework offers a new unifying perspective and basis for treating submodular minimization problems in both the constrained and unconstrained case.

Another class of algorithms we investigate, are based on using approximations of a submodular function. The tightest approximation is the Ellipsoidal Approximation [Goemans et al., 2009], which also correspondingly provides the tightest approximation guarantees for these problems. Unlike the MM algorithms however, these methods are not practical for real world applications.

3.1.2 Novel theoretical characterizations

Since many of the worst case guarantees are often much more pessimistic compared to the ones we observe in practice, it means that these lower bounds are specific to rather contrived classes of functions, whereas much better results can be achieved in many practically relevant cases. Given the increasing importance of submodular functions in machine learning, these observations beg the question of qualifying and quantifying properties that make sub-classes of submodular functions more amenable to learning and optimization. In this chapter, we also investigate the role of complexity parameters like the curvature, approximate monotonicity and submodularity ratio in the approximation guarantees, and show how they improve the worst case guarantees for practically relevant submodular functions, thus offering better connections between theory and practice. Interestingly, we show that the same theoretical constructs provide improved guarantees both for minimization and maximization (we shall study submodular maximization in chapter 4).

3.1.3 Worst case hardness and tight approximation algorithms

We also investigate the hardness of various instances of Problem 1, and the approximation guarantees of the algorithms, in the light of the theoretical characterizations above. We note

that, while the Ellipsoidal Approximation scheme (EA) provides theoretically the tightest guarantees, it is not scalable and practical. The Majorization-Minimization (MM) algorithms on the other hand, have slightly weaker worst case factors, but are extremely scalable and practical. Empirically however, we demonstrate that the scalable MMin algorithms perform comparable to the Ellipsoidal Approximation class of algorithms for several real world problems involving submodular minimization.

3.2 Motivating Applications

Submodularity's escalating popularity in machine learning is due to its natural applicability. Indeed, instances of Problem 1 are seen in many forms. Submodular functions very naturally capture notions of cooperation and attraction amongst items, and algorithmic complexity, when seen from the perspective of submodular minimization. Below we summarize some of the applications of Problem 1, which highlight this.

MAP inference in MRFs and Image Segmentation: Markov Random Fields with pairwise attractive potentials are important in computer vision, where MAP inference is identical to unconstrained submodular minimization solved via minimum cut [Boykov and Jolly, 2001].

Cooperative Cuts: While the MRF based models described above can be solved very efficiently via graph-cuts, they suffer from the shrinking bias problem, and images with elongated edges are not segmented properly. When modeled via a submodular function, however, the cost of a cut is not just the sum of the edge weights, but a richer function that allows cooperation between edges, and yields superior results on many challenging tasks. This was achieved in [Jegelka and Bilmes, 2011d] by partitioning the set of edges E of the grid graph into groups of similar edges, and defining concave over modular functions over these. This ensures that we offer a discount to edges of the same type. The corresponding MAP inference corresponds to Problem 1 where V is a set of edges in a graph, the function f is as

described above, and $\mathcal{C}$ is a set of cuts in this graph. This was shown to significantly improve many image segmentation results [Jegelka and Bilmes, 2011d].

Cooperative Matchings: The simplest model for matching key-points in pairs of images, which is also called the correspondence problem, can be posed as a bipartite matching. These models, however, do not capture interaction between the pixels, and does not work well in practice. One kind of desirable interaction is that similar or neighboring pixels be matched together. We can achieve this, via a minimum submodular matching [Iyer and Bilmes, 2014a], which we illustrate in the experiments section towards the end of this chapter.

Limited Vocabulary Speech Corpora: Large scale machine learning algorithms often have a complexity which is polynomial in the number of labels (for example, in speech recognition vocabulary size and in object recognition it is number of object classes). Due to the large number of labels, these systems often have large turnover times. Hence one would like to find large size corpora with limited number of labels, so as to enable accurately and rapidly prototyping novel and computationally expensive speech recognition architectures. Algorithmic complexity can naturally be modeled via submodular functions. For example, in speech recognition, the vocabulary size is an indicator of complexity and is submodular [Lin and Bilmes, 2011a]. This problem is naturally a form of size constrained submodular minimization, i.e minimizing algorithmic complexity which is submodular, given a constraint $|X| \geq k$ on the training dataset. For example in speech recognition, this corresponds to minimizing the vocabulary size given a lower bound constraint on the number of utterances.

Modeling Cooperative costs: Often submodular functions fit as natural models in modeling costs of items. This is mainly due to economies of scale, where often there is larger discounts in buying more items (buy one, get one free). These discounts often appear when buying similar items (for example when buying two sets of clothes etc.), and hence can be modeled via submodular minimization.

Minimum Power Assignment: In wireless and power networks, one seeks a connectivity structure that maintains connectivity at a minimum energy consumption. Submodular functions often naturally model the energy consumption, since there often sharing in consumption. This problem is equivalent to finding a suitable structure (e.g., a spanning tree) minimizing a submodular cost function [Wan et al., 2002].

Transportation: Costs in real-world transportation problems are often non-additive. For example, it may be cheaper to take a longer route owned by one carrier rather than a shorter route that switches carriers. Such economies of scale, or "right of usage" properties are captured in the "Categorized Bottleneck Path Problem" – a shortest path problem with submodular costs [Averbakh and Berman, 1994]. Similar costs have been considered for spanning tree and matching problems.

3.3 Combinatorial Algorithmic Framework

3.3.1 The Majorization-Minimization (MMin) framework

We define a generic Majorization-Minimization (MMin) algorithm as follows. In each iteration, the algorithm optimizes a modular approximation formed via the current solution Y. For minimization, we use an upper bound, defined via the supergradients of a submodular function (see Section 1.4.1):

$$m_Y^f(X) = f(Y) + g_Y(X) - g_Y(Y) \ \geq f(X), \tag{3.2}$$

These upper bounds are tight at the current solution, satisfying $m_Y^f(Y) = f(Y)$, and $m_Y^f(X) \geq f(X), \forall X \subseteq V$. In almost all cases, optimizing the modular approximation is much faster than optimizing the original cost function f.

Algorithm 1 shows our discrete MMin scheme for minimization for both constrained and unconstrained settings. Since we are minimizing a tight upper bound, the algorithm must make progress in each iteration.

Lemma 3.1. *Algorithm 1 monotonically decreases the objective function value at every*

Algorithm 1 Supergradient Descent Algorithm for Problem 1

Start with an arbitrary X^0.

while *until convergence* $(X^{i-1} = X^i)$ **do**

- Pick a supergradient g_{X^t} at X^t
- $X^{t+1} := \operatorname{argmin}_{X \in \mathcal{C}} m^f_{X^t}(X)$
- $t \leftarrow t + 1$

iteration for Problem 1, as long as a linear function can be exactly optimized over $\mathcal{C}$.

Proof. By definition, it holds that $f(X^{t+1}) \leq m^f_{X^t}(X^{t+1})$. Since X^{t+1} minimizes $m^f_{X^t}$, it follows that

$$f(X^{t+1}) \leq m^f_{X^t}(X^{t+1}) \leq m^f_{X^t}(X^t) = f(X^t). \tag{3.3}$$

□

Contrary to standard continuous subgradient descent schemes, Algorithm 1 produces a feasible solution at each iteration, thereby circumventing any rounding or projection steps that might be challenging under certain types of constraints. In addition, it is known that for relaxed instances of our problems, subgradient descent methods can suffer from slow convergence [Bach, 2013]. Nevertheless, the supergradient descent algorithm still relies on the choice of the supergradients defining the bounds. Therefore, we next analyze the effect of certain choices of semigradients for various optimization problems. In particular, we use MMin with the supergradients $\hat{g}_X, \check{g}_X$ and $\bar{g}_X$. In both the unconstrained and constrained settings, this yields a number of new approaches for Problem 1.

3.3.2 Ellipsoidal Approximation

In the case of constrained submodular minimization, we investigate another algorithm which provides the tightest theoretical guarantees. This algorithm is based on a generic approximation of a submodular function, called an Ellipsoidal Approximation (EA). The idea is to approximate an arbitrary submodular function, by an *easier* submodular function,

and solve the constrained optimization problem with the easier function. The approximation guarantees of this methods depend on how well the simpler surrogate function approximates the original function. The Ellipsoidal Approximation computes an approximation to a polymatroid function f in polynomial time by approximating the submodular polyhedron via an ellipsoid. This approximation approximates any submodular function, within a factor of $\alpha(n) = O(\sqrt{n}\log n)$. In other words, it provides a function $\hat{f}(X)$, which is of the form $\hat{f}(X) = \sqrt{w^f(X)}$ for a certain weight vector w^f, such that,

$$\hat{f}(X) \leq f(X) \leq \alpha(n)\hat{f}(X). \tag{3.4}$$

This then leads to an approximation algorithm obtained by minimizing $\hat{f}$ instead of the function f directly. The idea of the algorithm is very simple:

- Given a submodular function f, choose an approximation $\hat{f}$ such that $\hat{f}(X) \leq f(X) \leq \alpha(n)f(X)$.
- Choose the set $\hat{X} \in \text{argmin}_{X\in\mathcal{C}}\, \hat{f}(X)$.

We can obtain a simple approximation guarantee for this algorithm:

Lemma 3.2. *Given a submodular function f, let $\hat{f}$ be an approximation of f such that $\hat{f}(X) \leq f(X) \leq \alpha(n)f(X), \forall X \subseteq V$. Then any minimizer $\hat{X} \in \text{argmin}_{X\in\mathcal{C}}\, f(X)$ of f satisfies $f(X) \leq \alpha(n)f(X^*)$, where X^* is the optimal solution.*

Proof. Given that $\hat{f}(X) \leq f(X) \leq \alpha(n)f(X), \forall X \subseteq V$, let $\hat{X}$ be the optimal solution for minimizing $\hat{f}$ over $\mathcal{C}$. Then the following chain of inequalities holds,

$$f(\hat{X}) \leq \hat{f}(\hat{X}) \leq \alpha(n)\hat{f}(X^*) \leq \alpha(n)f(X^*) \tag{3.5}$$

□

We show that we can obtain a tighter approximation guarantee, dependent on the curvature of a submodular function. While this algorithm often provides theoretically the tightest guarantees, the algorithm is much more involved than the MM framework. Moreover, this algorithm only retains tight guarantees for constrained forms of submodular minimization.

3.4 Unconstrained Submodular Minimization

We begin with unconstrained minimization, where $\mathcal{C} = 2^V$ in Problem 1.

3.4.1 Majorization-Minimization Algorithms (MMin)

Each of the three supergradients yields a different variant of Algorithm 1, and we will call the resulting algorithms MMin-I, II and III, respectively. We make one more assumption: of the minimizing arguments in Step 4 of Algorithm 1, we always choose a set of minimum cardinality.

MMin-I is very similar to the algorithms proposed in [Jegelka et al., 2011]. Those authors, however, decompose f and explicitly represent graph-representable parts of the function f. We do not require or consider such a restriction here.

Let us define the sets $A = \{j : f(j|\emptyset) < 0\}$ and $B = \{j : f(j|V \setminus \{j\}) \leq 0\}$. Submodularity implies that $A \subseteq B$, and this allows us to define a lattice[1] $\mathcal{L} = [A, B]$ whose least element is the set A and whose greatest element is the set is B. This sublattice $\mathcal{L}$ of $[\emptyset, V]$ retains all minimizers X^* (i.e., $A \subseteq X^* \subseteq B$ for all X^*):

Lemma 3.3. *[Fujishige, 2005] Let $\mathcal{L}^*$ be the lattice of the global minimizers of a submodular function f. Then $\mathcal{L}^* \subseteq \mathcal{L}$, where we use $\subseteq$ to denote a sublattice.*

Lemma 3.3 has been used to prune down the search space of the minimum norm point algorithm from the power set of V to a smaller lattice [Bach, 2013, Fujishige and Isotani, 2011]. Indeed, A and B may be obtained by using MMin-III:

Lemma 3.4. *With $X^0 = \emptyset$ and $X^0 = V$, MMin-III returns the sets A and B, respectively. Initialized by an arbitrary X^0, MMin-III converges to $(X^0 \cap B) \cup A$.*

Proof. When using $X^0 = \emptyset$, we obtain $X^1 = \operatorname{argmin}_X f(\emptyset) + \sum_{j \in X} f(j) = A$. Since $A \subseteq B$, the algorithm will converge to $X^1 = A$. At this point, no more elements will be added,

[1] This lattice contains all sets S satisfying $A \subseteq S \subseteq B$

since for all $i \notin A$ we have $\bar{g}_{X^1}(i) = f(i \mid \emptyset) > 0$. Moreover, the algorithm will not remove any elements: for all $i \in A$, it holds that $\bar{g}_{X^1}(i) = f(i \mid V \setminus i) \leq f(i) \leq 0$. By a similar argumentation, the initialization $X^0 = V$ will lead to $X^1 = B$, where the algorithm terminates. If we start with any arbitrary X^0, MMin-III will remove the elements j with $f(j|V \setminus j) > 0$ and add the element j with $f(j|\emptyset) < 0$. Hence it will add the elements in A that are not in X^0 and remove those element from X^0 that are not in B. Let the resulting set be X^1. As before, for all $i \in A$, it holds that $\bar{g}_{X^1}(i) = f(i \mid V \setminus i) \leq f(i) \leq 0$, so these elements will not be removed in any possible subsequent iteration. The elements $i \in X^1 \setminus A$ were not removed, so $f(i \mid V \setminus i) \leq 0$. Hence, no more elements will be removed after the first iteration. Similarly, no elements will be added since for all $i \notin X^1$, it holds that $f(i \mid \emptyset) \geq f(i \mid V \setminus i) > 0$. □

Lemma 3.4 implies that MMin-III effectively provides a contraction of the initial lattice to $\mathcal{L}$, and, if X^0 is not in $\mathcal{L}$, it returns a set in $\mathcal{L}$. Henceforth, we therefore assume that we start with a set $X^0 \in \mathcal{L}$.

While the known lattice $\mathcal{L}$ has proven useful for warm-starts, MMin-I and II enable us to prune $\mathcal{L}$ even further. Let A_+ be the set obtained by starting MMin-I at $X^0 = \emptyset$, and B_+ be the set obtained by starting MMin-II at $X^0 = V$. This yields a new, smaller sublattice $\mathcal{L}_+ = [A_+, B_+]$ that retains all minimizers:

Theorem 3.1. *For any minimizer $X^* \in \mathcal{L}$, it holds that $A \subseteq A_+ \subseteq X^* \subseteq B_+ \subseteq B$. Hence $\mathcal{L}^* \subseteq \mathcal{L}_+ \subseteq \mathcal{L}$. Furthermore, when initialized with $X^0 = \emptyset$ and $X^0 = V$, respectively, both MMin-I and II converge in $O(n)$ iterations to a local minimum of f.*

By a local minimum, we mean a set X that satisfies $f(X) \leq f(Y)$ for any set Y that differs from X by a single element. We point out that Theorem 3.1 generalizes part of Lemma 3 in [Jegelka et al., 2011].

Before proving the main result, we show the following important Lemma:

Lemma 3.5. *Every iteration of MMin-I can be written as $X^{t+1} = X^t \cup \{j : f(j|X^t) < 0\}$. Similarly, every iteration of MMin-II can be expressed as $X^{t+1} = X^t \setminus \{j : f(j|X^t \setminus j) > 0\}$.*

Proof. (Lemma 3.5) Throughout this chapter, we assume that we select only the minimal minimizer of the modular function at every step. In other words, we do not choose the elements that have zero marginal cost. We observe that in iteration $t+1$ of MMin-I, we add the elements i with $\hat{g}_{X^t}(i) < 0$, i.e., $X^{t+1} = X^t \cup \{j : f(j|X^t) < 0\}$. No element will ever be removed, since $\hat{g}_{X^t}(i) = f(i \mid V \setminus i) \leq f(i \mid X^{t-1}) \leq 0$. If we start with $X^0 = \emptyset$, then after the first iteration, it holds that $X^1 = \text{argmin}_X f(\emptyset) + \sum_{j \in X} f(j)$. Hence $X^1 = A$. MMin-I terminates when reaching a set A_+, where $f(j|A_+) \geq 0$, for all $j \notin A_+$.

The analysis of MMin-II is analogous. In iteration $t+1$, we remove the elements i with $\check{g}_{X^t}(i) > 0$, i.e., $X^{t+1} = X^t \backslash \{j : f(j|X^t - j) > 0\}$. Similarly to the argumentation above, MMin-II never adds any elements. If we begin with $X^0 = V$, then $X^1 = \arg\min_X f(V) + \sum_{j \in V \setminus X} f(j|V - \{j\})$, and therefore $X^1 = B$. MMin-II terminates with a set B_+. □

Now we can prove Theorem 3.1.

Proof. (Thm. 3.1) Since, by Lemma 3.5, MMin-I only adds elements and MMin-II only removes elements, at least one in each iteration, both algorithms terminate after $O(n)$ iterations.

Let us now turn to the relation of X^* to A and B. Since $f(i) < 0$ for all $i \in A$, the set $X^1 = A$ found in the first iteration of MMin-I must be a subset of X^*. Consider any subset $X^t \subseteq X^*$. Any element j for which $f(j \mid X^t) < 0$ must be in X^* as well, because by submodularity, $f(j \mid X^*) \leq f(j \mid X^t) < 0$. This means $f(X^* \cup j) < f(X^*)$, which would otherwise contradict the optimality of X^*. The set of such j is exactly X^{t+1}, and therefore $X^{t+1} \subseteq X^*$. This induction shows that MMin-I, whose first solution is $A \subseteq X^*$, always returns a subset of X^*. Analogously, $B \supseteq X^*$, and MMin-II only removes elements $j \notin X^*$.

Finally, we argue that A_+ is a local minimum; the proof for B_+ is analogous. Algorithm MMin-I generates a chain $\emptyset = X^0 \subseteq X^1 \subseteq X^2 \cdots \subseteq A_+ = X^T$. For any $t \leq T$, consider $j \in X^t \setminus X^{t-1}$. Submodularity implies that $f(j|A_+ \setminus j) \leq f(j|X^{t-1}) < 0$. The last inequality follows from the fact that j was added in iteration t. Therefore, removing any $j \in A_+$ will increase the cost. Regarding the elements $i \notin A_+$, we observe that MMin-I has terminated,

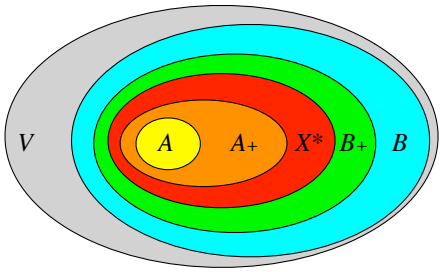

Figure 3.1: Venn diagram for the lattices obtained by MMin-I, II and III. We are searching for the optimal set $X^* \subseteq V$. The lattice $\mathcal{L}$ contains all sets S "between" A and B, i.e., $A \subseteq S \subseteq B$. The lattice $\mathcal{L}_+$ uses the sets A_+ and B_+ instead (it contains all sets T with $A_+ \subseteq T \subseteq B_+$) and therefore provides a tighter bound and smaller search space around the optimal solution X^*.

which implies that $f(i \mid A_+) \geq 0$. Hence, adding i to A_+ will not improve the solution, and A_+ is a local minimum. □

Theorem 3.1 has a number of nice implications. First, it provides a tighter bound on the lattice of minimizers of the submodular function f that, to the best of our knowledge, has not been used or mentioned before. The sets A_+ and B_+ obtained above are guaranteed to be supersets and subsets of A and B, respectively, as illustrated in Figure 3.1. This means we can start any algorithm for submodular minimization from the lattice $\mathcal{L}_+$ instead of the initial lattice 2^V or $\mathcal{L}$. When using an algorithm whose running time is a high-order polynomial of $|V|$, any reduction of the ground set V is beneficial. Second, each iteration of MMin takes linear time. Therefore, its total running time is $O(n^2)$. Third, Theorem 3.1 states that both MMin-I and II converge to a local minimum. This may be counter-intuitive if one considers that each algorithm either only adds or only removes elements. In consequence, a local minimum of a submodular function can be obtained in $O(n^2)$, a fact that is of independent interest and that does not hold for local maximizers [Feige et al., 2011b].

The following example illustrates that $\mathcal{L}_+$ can be a strict subset of $\mathcal{L}$ and therefore

provides non-trivial pruning. Let $w_1, w_2 \in \mathbb{R}^V$, $w_1 \geq 0$ be two vectors, each defining a linear (modular) function. Then the function $f(X) = \sqrt{w_1(X)} + w_2(X)$ is submodular. Specifically, let $w_1 = [3, 9, 17, 14, 14, 10, 16, 4, 13, 2]$ and $w_2 = [-9, 4, 6, -1, 10, -4, -6, -1, 2, -8]$. Then we obtain $\mathcal{L}$ defined by $A = [1, 6, 7, 10]$ and $B = [1, 4, 6, 7, 8, 10]$. The tightened sublattice contains exactly the minimizer: $A_+ = B_+ = X^* = [1, 6, 7, 8, 10]$.

As a refinement to Theorem 3.1, we can show that MMin-I and MMin-II converge to the local minima of lowest and highest cardinality, respectively.

Lemma 3.6. *The set A_+ is the smallest local minimum of f (by cardinality), and B_+ is the largest. Moreover, every local minimum Z is in $\mathcal{L}_+$: $Z \in \mathcal{L}_+$ for every local minimum Z.*

As a corollary, Lemma 3.6 implies that if a submodular function has a unique local minimum, MMin-I and II must find this minimum, which is a global one.

Proof. The proof proceeds analogously to the proof of Theorem 3.1. Let Y_s be the local minimum of smallest-cardinality, and Y_ℓ the largest one. First, we note that $X^0 = \emptyset \subseteq Y_s$. For induction, assume that $X^t \subseteq Y_s$. For contradiction, assume there is an element $j \in X^{t+1}$ that is not in Y_s. Since $j \in X^{t+1} \setminus X^t$, it holds by construction that $f(j \mid Y_s) \leq f(j \mid X^t) < 0$, implying that $f(Y_s \cup j) < f(Y_s)$. This contradicts the local optimality of Y_s, and therefore it must hold that $X^{t+1} \subseteq Y_s$. Consequently, $A_+ \subseteq Y_s$. But A_+ is itself a local minimum, and hence equality holds. The result for B_+ follows analogously.

By the same argumentation as above for Y_s and Y_ℓ, we conclude that each local minimum Z satisfies $A_+ \subseteq Z \subseteq B_+$, and therefore $Z \in \mathcal{L}_+ \subseteq \mathcal{L}$. □

3.4.2 Extensions of MMin

In the following we consider two extensions of MMin-I and II. First, we analyze an algorithm that alternates between MMin-I and MMin-II. While such an algorithm does not provide much benefit when started at $X^0 = \emptyset$ or $X^0 = V$, we see that with a random initialization $X^0 = R$, the alternation ensures convergence to a local minimum. Second, we address the question of which supergradients to select in general. In particular, we show that the supergradients $\hat{g}$

and $\check{g}$ subsume alternative supergradients and provide the tightest results with MMin. Hence, our results are the tight.

Alternating MMin-I and II and arbitrary initializations. Instead of running only one of MMin-I and II, we can run one until it stops and then switch to the other. Assume we initialize both algorithms with a random set $X^0 = R \in \mathcal{L}_+$. By Theorem 3.1, we know that MMin-I will return a subset $R^1 \supset R$ (no element will be removed because all removable elements are not in B, and $R \subset B$ by assumption). When MMin-I terminates, it holds that $\hat{g}_{R^1}(j) = f(j|R^1) \geq 0$ for all $j \notin R^1$, and therefore R^1 cannot be increased using $\hat{g}_{R_1}$. We will call such a set an *I-minimum*. Similarly, MMin-II returns a set $R_1 \subseteq R$ from which, considering that $\check{g}_{R_1}(j) = f(j|R_1 \setminus j) \leq 0$ for all $j \in R_1$, no elements can be removed. We call such a non-decreasable set a *D-minimum*. Every local minimum is both an I-minimum and a D-minimum.

We can apply MMin-II to the I-minimum R^1 returned by MMin-I. Let us call the resulting set R^2. Analogously, applying MMin-I to R_1 yields $R_2 \supseteq R_1$.

Lemma 3.7. *The sets R_2 and R^2 are local optima. Furthermore, $R_1 \subseteq R_2 \subseteq R^2 \subseteq R^1$.*

Proof. It is easy to see that $A \subseteq R_1 \subseteq B$, and $A \subseteq R^1 \subseteq B$. By Lemma 3.5, MMin-I applied to R_1 will only add elements, and MMin-II on R^1 will only remove elements. Since R^1 is an I-minimum, adding an element $j \in V \setminus R^1$ to any set $X \subset R^1$ never helps, and therefore R^1 contains all of R_1, R_2 and R^2. Similarly, R_1 is contained in R_2, R^2 and R^1. In consequence, it suffices to look at the contracted lattice $[R_1, R^1]$, and any local minimum in this sublattice is a local minimum on $[\emptyset, V]$. Theorem 3.1 applied to the sublattice $[R_1, R^1]$ (and the submodular function restricted to the sublattice) yields the inclusion $R_2 \subseteq R^2$, so $R_1 \subseteq R_2 \subseteq R^2 \subseteq R^1$, and both R_2 and R^2 are local minima. □

The following lemma provides a more general view.

Lemma 3.8. *Let $S_1 \subseteq S^1$ be such that S_1 is an I-minimum and S^1 is a D-minimum. Then there exist local minima $S_2 \subseteq S^2$ in $[S_1, S^1]$ such that initializing with any $X^0 \in [S_1, S^1]$, an*

alternation of MMin-I and II converges to a local minimum in $[S_2, S^2]$*, and*

$$\min_{X \in [S_1, S^1]} f(X) = \min_{X \in [S_2, S^2]} f(X).$$

Proof. Let S_2, S^2 be the smallest and largest local minima within $[S_1, S^1]$. By the same argumentation as for Lemma 3.7, using $X^0 \in [S_1, S^1]$ leads to a local minimum within $[S_2, S^2]$. Since by definition all local optima in $[S_1, S^1]$ are within $[S_2, S^2]$, the global minimum within $[S_1, S^1]$ will also be in $[S_2, S^2]$. □

The above lemmata have a number of implications for minimization algorithms. First, many of the properties for initializing with V or the empty set can be transferred to arbitrary initializations. In particular, the succession of MMin-I and II will terminate in $O(n^2)$ iterations, regardless of what X^0 is. Second, Lemmas 3.7 and 3.8 provide useful pruning opportunities: we can prune down the initial lattice to $[R_2, R^2]$ or $[S_2, S^2]$, respectively. In particular, if any global optimizer of f is contained in $[S_1, S^1]$, it will also be contained in $[S_2, S^2]$.

Choice of supergradients. We close this section with a remark about the choice of supergradients. The following Lemma states how $\hat{g}_X$ and $\check{g}_X$ subsume alternative choices of supergradients and MMin-I and II lead to the tightest results possible.

Lemma 3.9. *Initialized with* $X^0 = \emptyset$*, Algorithm 1 will converge to a subset of* A_+ *with* any *choice of supergradients. Initialized with* $X^0 = V$*, the algorithm will converge to a superset of* B_+ *with any choice of supergradients. Moreover, if* X^0 *is a local minimum, then the algorithm will not move with any supergradient.*

The above Lemma implies that we do not loose any generality, by considering only the three specific supergradients $\hat{g}, \check{g}$ and $\bar{g}$. Moreover, it also shows that in the context of unconstrained minimization, Algorithm 1 is bound to get stuck at local minima. Correspondingly, while it provides useful mechanisms for pruning, this framework cannot be used as an exact algorithm for unconstrained minimization. The proof of Lemma 3.9 is very similar to the proof of Theorem 3.1.

3.5 Constrained Monotone Submodular Minimization

In this section, we shall study submodular minimization under combinatorial constraints. Even simple constraints like cardinality lower bound constraints, make Problem 1 very hard: they do not even admit constant factor guarantees [Svitkina and Fleischer, 2008, Jegelka and Bilmes, 2011c, Goel et al., 2009]. This section shall primarily deal with the functions f being monotone (which are often, more realistic in applications), while in the next section (section 3.6), we shall extend these results to constrained non-monotone submodular minimization. We shall first study MMin (in section 3.5.1), followed by the Ellipsoidal Approximation (section 3.5.2). In both cases, we shall provide curvature dependent bounds, and then show how these bounds are tight (by exhibiting curvature dependent lower bounds) for several specific combinatorial constraints. The results of MMin and EA for specific constraints, along with the corresponding hardness is shown in Table 3.1.

3.5.1 Majorization-Minimization (MMin) Algorithm

MMin straightforwardly generalizes to constraints more complex than $\mathcal{C} = 2^V$, and Theorem 3.1 still holds for more general lattices or ring family constraints. Beyond lattices, however, other constrained submodular minimization problems are significantly harder than the unconstrained version. Fortunately, however, MMin still works under a large class of constraints $\mathcal{C}$. The only requirement on $\mathcal{C}$ is that there exists an efficient algorithm which minimizes a non-negative modular cost function over $\mathcal{C}$. This subroutine can even be approximate. Such algorithms are available for cardinality bounds, independent sets of a matroid and many other combinatorial constraints such as trees, paths or cuts. The main performance criteria of the algorithms shall be in the form of approximation guarantees.

The next theorem states an upper bound on the approximation factor achieved by MMin for non-negative, non-decreasing (monotone) submodular functions.

Constraint	Semigradient (MMin)	Ellipsoid approx. (EA)	Hardness
Card. LB	$\frac{k}{1+(k-1)(1-\kappa_f)}$	$O(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})$	$\tilde{\Omega}(\frac{n^{1/2})}{1+(n^{1/2}-1)(1-\kappa_f)})$
Spanning Tree	$\frac{n}{1+(n-1)(1-\kappa_f)}$	$O(\frac{\sqrt{m}\log m}{1+(\sqrt{m}\log m-1)(1-\kappa_f)})$	$\tilde{\Omega}(\frac{n}{1+(n-1)(1-\kappa_f)})$
Matchings	$\frac{n}{2+(n-2)(1-\kappa_f)}$	$O(\frac{\sqrt{m}\log m}{1+(\sqrt{m}\log m-1)(1-\kappa_f)})$	$\tilde{\Omega}(\frac{n}{1+(n-1)(1-\kappa_f)})$
Edge Cover	$\frac{n}{1+(\frac{n}{2}-1)(1-\kappa_f)}$	$O(\frac{\sqrt{m}\log m}{1+(\sqrt{m}\log m-1)(1-\kappa_f)})$	$\tilde{\Omega}(\frac{n}{1+(0.5n-1)(1-\kappa_f)}$
s-t path	$\frac{n}{1+(n-1)(1-\kappa_f)}$	$O(\frac{\sqrt{m}\log m}{1+(\sqrt{m}\log m-1)(1-\kappa_f)})$	$\tilde{\Omega}(\frac{n^{2/3}}{1+(n^{2/3}-1)(1-\kappa_f)})$
s-t cut	$\frac{m}{1+(m-1)(1-\kappa_f)}$	$O(\frac{\sqrt{m}\log m}{1+(\log m\sqrt{m}-1)(1-\kappa_f)})$	$\tilde{\Omega}(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})$

Table 3.1: Approximation bounds implied by MMin-I and the Ellipsoidal Approximation Algorithm for specific constrained submodular minimization problems. All these bounds are shown with respect to the weakest notion of curvature κ_f. See the text for more details.

Theorem 3.2. *Let $X^* \in \operatorname{argmin}_{X\in\mathcal{C}} f(X)$. The solution $\widehat{X}$ returned by MMin-I satisfies*

$$f(\widehat{X}) \leq \frac{|X^*|}{1+(|X^*|-1)(1-\hat{\kappa_f}(X^*))} f(X^*) \leq \min\{|X^*|, \frac{1}{1-\hat{\kappa_f}(X^*)}\} f(X^*)$$

If the minimization in Step 4 is done with approximation factor β, then

$$f(\widehat{X}) \leq \frac{\beta|X^*|}{1+(|X^*|-1)(1-\hat{\kappa_f}(X^*))} f(X^*) \leq \beta\min\{|X^*|, \frac{1}{1-\hat{\kappa_f}(X^*)}\} f(X^*).$$

Proof. Surprisingly, it suffices to consider just the fist iteration of MMin-I, which is:

$$X^1 = \operatorname*{argmin}_{X\in\mathcal{C}} \sum_{i\in X} f(i)$$

The simple modular upper bound above, is essentially formed via the supergradient at the emptyset $X = \emptyset$. Since further iterations only improve the objective, the approximation guarantee holds at convergence as well. The following Lemma is a key ingredient in showing this.

Lemma 3.10. *Given a monotone submodular function f,*

$$f(X) \leq \sum_{j\in X} f(j) \leq \frac{|X|}{1+(|X|-1)(1-\hat{\kappa_f}(X))} f(X)$$

The approximation guarantee immediately follows from the above Lemma, via the following sequence of inequalities:

$$f(\widehat{X}) \leq f(X^1) \leq \sum_{j \in X^1} f(j) \leq \sum_{j \in X^*} f(j) \leq \frac{|X^*|}{1 + (|X^*| - 1)(1 - \hat{\kappa_f}(X^*))} f(X^*)$$

The first inequality follows since X^1 is the solution just after the first iteration, while $\hat{X}$ is after convergence, and the second one is from the definition of the modular upper bound. The third inequality holds since X^1 is the optimizer of of the function $\hat{f}(X) = \sum_{j \in X} f(j)$ over $\mathcal{C}$, and is hence better than X^* (which is also feasible). The last inequality follows from the Lemma 3.10.

To show Lemma 3.10, we shall use the following facts, which follow from the definitions of submodularity and curvature.

$$\text{Fact 1: } (1 - \hat{\kappa_f}(X)) \sum_{j \in X} f(j) = \sum_{j \in X} f(j | X \backslash j),$$

$$\text{Fact 2: } f(X) - f(k) \geq \sum_{j \in X \backslash k} f(j | X \backslash j), \forall k \in X.$$

Sum the expressions from Fact 2, $\forall k \in X$, use Fact 1, and we obtain the following series of inequalities,

$$\begin{aligned}
|X| f(X) - \sum_{k \in X} f(k) &\geq \sum_{k \in X} \sum_{j \in X \backslash k} f(j | X \backslash j) \\
&\geq \sum_{k \in X} \sum_{j \in X} f(j | X \backslash j) - \sum_{k \in X} f(j | X \backslash k) \\
&\geq (|X| - 1) \sum_{j \in X \backslash k} f(j | X \backslash j) \\
&\geq (|X| - 1)(1 - \hat{\kappa_f}(X)) \sum_{k \in X} f(k)
\end{aligned}$$

Hence we obtain that,

$$\sum_{k \in X} f(k) \leq \frac{|X|}{1 + (|X| - 1)(1 - \hat{\kappa_f}(X))} f(X)$$

From the fact that $1 - \hat{\kappa_f}(X) \geq 1 - \kappa_f(X)$, it immediately follows that,

$$\sum_{k \in X} f(k) \leq \frac{|X|}{1 + (|X| - 1)(1 - \kappa_f(X))} f(X)$$

□

We immediately then have the following corollary:

Corollary 3.1. *The solution $\widehat{X}$ returned by MMin-I satisfies*

$$f(\widehat{X}) \leq \frac{|X^*|}{1 + (|X^*| - 1)(1 - \kappa_f)} f(X^*)$$

This result follows directly from Theorem 3.2 since κ_f is a weaker notion of curvature.

Remark 3.1. *While the results above are shown for MMin-I, they in fact hold for any algorithm which uses either of the supergradients $\hat{g}, \check{g}$ or $\bar{g}$ and starts at $X^0 = \emptyset$. Note that for showing the guarantee, it is important to start at the $X^0 = \emptyset$. Moreover, one could also alternate between these different supergradients, and choose the best solution at every iteration. In this chapter, however, we just restrict ourselves to MMIn-I.*

Note that the bound above is at most $O(\frac{n}{1+(n-1)(1-\kappa_f)})$, where $n = |V|$ is the dimension of the problem. This bound also extends to a number of combinatorial constraints including cardinality and matroid constraints, cuts, spanning trees, shortest paths and perfect matchings. For problems where $\kappa_f < 1$, Theorem 3.2 refines bounds for constrained minimization that are given in [Goel et al., 2009, Svitkina and Fleischer, 2008]. To our knowledge, this is the first curvature dependent bound for this general class of minimization problems. Table 3.1 gives a summary of the results obtained through MMin-I under different constraints. In section 3.5.7, we shall compare our results for various instances of constraints which are natural in applications.

3.5.2 Ellipsoidal Approximation (EA) Algorithm

This generic approximation scheme was introduced by [Goemans et al., 2009], where they shows how any submodular function can be approximated by a simpler submodular function

(which is explicitly representable) upto a factor of $O(\sqrt{n})$. This immediately yields a$O(\sqrt{n})$ approximation algorithm for many of the constrained optimization problems addressed in this chapter. We extend these bounds to curvature dependent ones, and show improved factors of $O(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})$ for these problems.

A crucial component of the algorithm, is the notion of *Curve-Normalized* polymatroid function. Specifically, we define the κ_f-*curve-normalized* version of f as

$$f^{\kappa}(X) = \frac{f(X) - (1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f} \tag{3.6}$$

If $\kappa_f = 0$, then we set $f^{\kappa} \equiv 0$. We call f^{κ} the curve-normalized version of f because its curvature is $\kappa_{f^{\kappa}} = 1$. The function f^{κ} allows us to decompose a submodular function f into a "difficult" polymatroid function and an "easy" modular part as $f(X) = f_{\text{difficult}}(X) + m_{\text{easy}}(X)$ where $f_{\text{difficult}}(X) = \kappa_f f^{\kappa}(X)$ and $m_{\text{easy}}(X) = (1-\kappa_f)\sum_{j\in X} f(j)$. Moreover, we may modulate the curvature of given any function g with $\kappa_g = 1$, by constructing a function $f(X) \triangleq cg(X) + (1-c)|X|$ with curvature $\kappa_f = c$ but otherwise the same polymatroidal structure as g. Our curvature-based decomposition is different from decompositions such as that into a *totally normalized* function and a modular function [Cunningham, 1983]. Indeed, the curve-normalized function has some specific properties that will be useful later on.

Lemma 3.11. *If f is monotone submodular with $\kappa_f > 0$, then*

$$f(X) \leq \sum_{j\in X} f(j), \quad f(X) \geq (1-\kappa_f)\sum_{j\in X} f(j). \tag{3.7}$$

Proof. The inequalities follow from submodularity and monotonicity of f. The first part follows from the subadditivity of f. The second inequality follows since $f(X) \geq \sum_{j\in X} f(j|V\backslash j) \geq (1-\kappa_f)\sum_{j\in X} f(j)$, since $\forall j \in X, f(j|V\backslash j) \geq (1-\kappa_f)f(j)$ by definition of κ_f. □

Lemma 3.12. *If f is monotone submodular, then $f^{\kappa}(X)$ in Eqn.* (3.6) *is a monotone non-negative submodular function. Furthermore, $f^{\kappa}(X) \leq \sum_{j\in X} f(j)$.*

Proof. Submodularity of f^{κ} is evident from the definition. To show the monotonicity, it suffices to show that $f(X) - (1-\kappa_f)\sum_{j\in X} f(j)$ is monotone non-decreasing and non-negative

submodular. To show it is non-decreasing, notice that $\forall j \notin X, f(j|V\backslash j) - (1-\kappa_f)f(j) \geq 0$, since $(1-\kappa_f)f(j) \leq f(j|V\backslash j)$ by the definition of κ_f. Non-negativity follows from monotonicity and the fact that $f^\kappa(\emptyset) = 0$. To show the second part, notice that $\frac{f(X)-(1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f} \leq \frac{\sum_{j\in X} f(j)-(1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f} = \sum_{j\in X} f(j)$. □

The curve-normalization leads to refined upper bounds for the Ellipsoidal Approximation. The key idea is to approximate only the curved part of f, and retain the modular part exactly.

Theorem 3.3. *Given a polymatroid function f with $\kappa_f < 1$, let f^κ be its* curve-normalized *version defined in Equation* (3.6)*, and let $\hat{f}^\kappa$ be a submodular function satisfying $\hat{f}^\kappa(X) \leq f^\kappa(X) \leq \alpha(n)\hat{f}^\kappa(X)$, for some $X \subseteq V$. Then the function $\hat{f}(X) \triangleq \kappa_f \hat{f}^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j)$ satisfies*

$$\hat{f}(X) \leq f(X) \leq \frac{\alpha(n)}{1+(\alpha(n)-1)(1-\kappa_f)}\hat{f}(X) \leq \frac{\hat{f}(X)}{1-\kappa_f}. \tag{3.8}$$

The above inequalities hold, even if we use an upper bound $\bar{\kappa_f}$ instead of the actual curvature κ_f.

Proof. The first inequality follows directly from definitions. To show the second inequality, note that $\hat{f}^\kappa(X) \geq \frac{f^\kappa(X)}{\alpha(n)}$, and therefore

$$\frac{f(X)}{\kappa_f \hat{f}^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j)} = \frac{\kappa_f f^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f \hat{f}^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j)} \tag{3.9}$$

$$\leq \frac{\kappa_f f^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f \frac{f^\kappa(X)}{\alpha(n)} + (1-\kappa_f)\sum_{j\in X} f(j)} \tag{3.10}$$

$$= \alpha(n)\frac{\kappa_f f^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f f^\kappa(X) + (1-\kappa_f)\alpha(n)\sum_{j\in X} f(j)} \tag{3.11}$$

$$= \frac{\alpha(n)}{1+\frac{(\alpha(n)-1)(1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f f^\kappa(X)+(1-\kappa_f)\sum_{j\in X} f(j)}} \tag{3.12}$$

$$\leq \frac{\alpha(n)}{1+(\alpha(n)-1)(1-\kappa_f)} \tag{3.13}$$

The last inequality follows since $\kappa_f f^\kappa(X) + (1-\kappa_f)\sum_{j\in X} f(j) \leq \sum_{j\in X} f(j)$. The other inequalities in Eqn. (6.3) follow directly from the definitions.

It is also easy to see that all the above inequalities will hold using an upper bound $\bar{\kappa_f} > \kappa_f$ instead of κ_f in the definition of the curve-normalized function. The bound in that case would be,

$$\hat{f}(X) \leq f(X) \leq \frac{\alpha(n)}{1 + (\alpha(n) - 1)(1 - \bar{\kappa_f})} \hat{f}(X) \leq \frac{\hat{f}(X)}{1 - \bar{\kappa_f}} \tag{3.14}$$

where, $\hat{f}(X) = \bar{\kappa_f} \hat{f\bar{\kappa}}(X) + (1 - \bar{\kappa_f}) \sum_{j \in X} f(j)$, $\hat{f\bar{\kappa}}$ is an approximation of $f^{\bar{\kappa}}$ satisfying $\hat{f\bar{\kappa}}(X) \leq f^{\kappa}(X) \leq \alpha(n) \hat{f^{\kappa}}(X)$ and,

$$f^{\bar{\kappa}} = \frac{f(X) - (1 - \bar{\kappa_f}) \sum_{j \in X} f(j)}{\bar{\kappa_f}} \tag{3.15}$$

□

The result above can directly be applied to the Ellipsoidal Approximation Algorithm. Recall, that for any polymatroid rank function f, one can compute a weight vector w^f and correspondingly an approximation (which is the Ellipsoidal Approximation) $\sqrt{w^f(X)}$ via a polynomial number of oracle queries such that $\sqrt{w^f}(X) \leq f(X) \leq O(\sqrt{n \log n}) \leq \sqrt{w^f(X)}$. The following corollary provides the worst case curvature-dependent results for the Ellipsoidal Approximation Algorithm.

Corollary 3.2. *Let f be a polymatroid function with $\kappa_f < 1$, and let $\sqrt{w^{f^{\kappa}}(X)}$ be the ellipsoidal approximation to the κ-curve-normalized version $f^{\kappa}(X)$ of f. Then the function $f^{ea}(X) = \kappa_f \sqrt{w^{f^{\kappa}}(X)} + (1 - \kappa_f) \sum_{j \in X} f(j)$ satisfies*

$$f^{ea}(X) \leq f(X) \leq O\left(\frac{\sqrt{n} \log n}{1 + (\sqrt{n} \log n - 1)(1 - \kappa_f)}\right) f^{ea}(X). \tag{3.16}$$

The Ellipsoidal Approximation Algorithm then just minimizes the surrogate function $f^{ea}(X)$. This function has a special form: a weighted sum of a concave function and a modular function. Minimizing such a function over constraints $\mathcal{C}$ is harder than minimizing a merely modular function, but with the algorithm in [Nikolova, 2010] we obtain an FPTAS. The FPTAS will yield a $\beta = (1 + \epsilon)$-approximation through an algorithm polynomial in $\frac{1}{\epsilon}$. We call this algorithm EA

Theorem 3.4. *For a submodular function with curvature $\kappa_f < 1$, algorithm EA will return a solution $\widehat{X}$ that satisfies*

$$f(\widehat{X}) \leq O\left(\frac{\sqrt{n}\log n}{(\sqrt{n}\log n - 1)(1-\kappa_f)+1)}\right) f(X^*). \tag{3.17}$$

The specific bounds of EA for many constrained optimization problems are in Table 3.1. While this generic scheme often provides the theoretically tightest guarantees, it is extremely hard to implement it in practice. MMin-I, however, provides a much more practical and scalable approach for addressing these problems. Interestingly, we see empirically that MMin-I actually performs comparably to EA for a number of practically relevant submodular functions (see Section 3.7).

3.5.3 Curvature Dependent Hardness Results

Previous information-theoretic lower bounds for constrained submodular minimization [Goel et al., 2009, Goemans et al., 2009, Iwata and Nagano, 2009, Svitkina and Fleischer, 2008] are *independent* of curvature and use functions with $\kappa_f = 1$. These curvature-independent bounds are proven by constructing two essentially indistinguishable matroid rank functions h and f^R, one of which depends on a random set $R \subseteq V$. One then argues that any algorithm would need to make a super-polynomial number of queries to the functions for being able to distinguish h and f^R with high enough probability. The lower bound will be the ratio $\max_{X\in\mathcal{C}} h(X)/f^R(X)$. We extend this proof technique to functions with a fixed given curvature. To this end, we define the functions

$$f^R_\kappa(X) = \kappa_f f^R(X) + (1-\kappa_f)|X| \quad \text{and} \quad h_\kappa(X) = \kappa_f h(X) + (1-\kappa_f)|X|. \tag{3.18}$$

Both of these functions have curvature κ_f. This construction enables us to explicitly introduce the effect of curvature into information-theoretic bounds for all monotone submodular functions. In section 3.5.7, we shall investigate specific combinatorial constraints, and we shall use this technique in providing lower bounds in each of these cases.

3.5.4 Role of curvature

A class of functions with $\kappa_f = 1$ are matroid rank functions, implying that these functions are difficult instances the MMin algorithms. But several classes of functions occurring in applications have more benign curvature. For example, sums of concave over modular functions have been used in a number of applications [Lin and Bilmes, 2011c, Jegelka and Bilmes, 2011d, Iyer and Bilmes, 2012b].

Corollary 3.3. *Given weight vectors $w_1, \cdots, w_k \geq 0$, and a submodular function $f(X) = \sum_{i=1}^{k} \lambda_i [w_i(X)]^a, \lambda_i \geq 0$, for $a \in (0, 1)$, MMin-I minimized f upto a factor of $O(|X^*|^{1-a})$.*

For example, MMin approximates the function $f(X) = \sum_{i=1}^{k} \lambda_i \sqrt{w_i(X)}$ upto a factor of $\sqrt{|X^*}$.

Proof. The result above follows from a simple bound on the curvature of these functions:

Lemma 3.13. *Given weight vectors $w_1, \cdots, w_k \geq 0$, and a submodular function $f(X) = \sum_{i=1}^{k} \lambda_i [w_i(X)]^a, \lambda_i \geq 0$, for $a \in (0, 1]$, it holds that,*

$$\hat{\kappa_f}(X) \leq 1 - \frac{a}{|X|^{1-a}}$$

Plugging the bound above into Theorem 3.2 proves the result.

We now show this Lemma. Let $f(X) = [w(X)]^a$, for $w \geq 0$ and $a \in (0, 1]$. Then,

$$\begin{aligned} f(j|X\backslash j) &= f(X) - f(X\backslash j) \\ &= [w(X)]^a - [w(X) - w(j)]^a \\ &\geq \frac{aw(j)}{w(X)^{1-a}} \end{aligned}$$

The last inequality again holds due to concavity of $g(x) = x^a$. In particular, for a concave function, $g(y) - g(x) \leq g'(x)(y - x)$, where g' is the derivative of g. Hence $g(x) \geq g(y) + g'(x)(x - y)$. Substitute $y = w(X) - w(j), x = w(X)$ and $g(x) = x^a$, and we get the above expression.

Hence we have,

$$\begin{aligned} 1 - \hat{\kappa_f}(X) &= \frac{\sum_{j \in X} f(j|X \backslash j)}{\sum_{j \in X} f(j)} \\ &\geq \frac{a \sum_{j \in X} w(j) w(X)^{a-1}}{\sum_{j \in X} w(j)^a} \\ &\geq \frac{a w(X)^a}{\sum_{j \in X} w(j)^a} \\ &\geq \frac{a}{|X|^{1-a}} \end{aligned}$$

The last inequality follows from the previous Lemma. □

3.5.5 Comparing the guarantees of EA and MMin

Note that the guarantees for MMin (Theorem 3.2) holds for the tightest notion of curvature, which is basically the average over the individual gains at a set X. This is in contrast to $\kappa_f(X)$ and κ_f, which are both worst case and hence weaker – note that the curvature dependent EA has a guarantee in terms of κ_f. Moreover, $\hat{\kappa_f}(X^*)$ also depends on the curvature with respect to the optimal set, as opposed to κ_f which is the curvature at V. While many submodular functions ultimately get totally curved (in which case, the bound with respect to κ_f becomes meaningless), the optimal sets for constrained minimization problems may not be as curved. Hence, though the Ellipsoidal Approximation admits a tighter characterization with respect to n, in comparison to MMin, its dependence on curvature is weaker. In addition, the curvature provides an interesting perspective to these algorithms – as long as the submodular function f has bounded curvature, both EA and MMin admit a guarantee of $O(1/(1-\kappa_f))$ when n gets large. In our experiments, we noted that EA takes several hours to run on ground sets of size $n = 100$.

3.5.6 Running time of MMin and EA

The bounds of Theorem 3.2 hold after the first iteration. Nevertheless, empirically we often found that for problem instances that are not worst-case, subsequent iterations can improve

the solution substantially. Using Theorem 3.2, we can bound the number of iterations the algorithm will take. To do so, we assume an η-approximate version, where we proceed only if $f(X^{t+1}) \leq (1-\eta)f(X^t)$ for some $\eta > 0$. In practice, the algorithm usually terminates after 5 to 10 iterations for an arbitrarily small η.

Lemma 3.14. *MMin-I runs in* $O(\frac{1}{\eta}T \log \frac{n}{1+(n-1)(1-\kappa_f)})$ *time, where* T *is the time for minimizing a modular function subject to* $X \in \mathcal{C}$.

Proof. At the end of the first iteration, we obtain a set X^1 such that $f(X^1) \leq \frac{n}{1+(n-1)(1-\kappa_f)} f(X^*)$. The η-approximate assumption implies that $f(X^{t+1}) \leq (1-\eta)f(X^t) \leq (1-\eta)^t f(X^1)$. Using that $\log(1-\eta) \leq \eta^{-1}$ and Theorem 3.2, we see that the algorithm terminates after at most $O(\frac{1}{\eta} \log \frac{n}{1+(n-1)(1-\kappa_f)})$ iterations. □

Unlike the Majorization-Minimization Algorithm (MMin), the Ellipsoidal Approximation Algorithm is higher order polynomial [Goemans et al., 2009] and does not scale to medium and large problems.

3.5.7 Bounds for Specific Combinatorial Constraints

In this section, we apply the results above to specific optimization problems, for which we show (mostly tight) curvature-dependent upper and lower bounds. These results are summarized in Table 3.1.

Cardinality lower bounds (SLB). A simple constraint is a lower bound on the cardinality of the solution, i.e., $\mathcal{C} = \{X \subseteq V : |X| \geq k\}$. [Svitkina and Fleischer, 2008] prove that for monotone submodular functions of arbitrary curvature, it is impossible to find a polynomial-time algorithm with an approximation factor better than $\sqrt{n/\log n}$. They show an algorithm which matches this approximation factor.

Observation 3.1. *For the SLB problem, Algorithm EA and MMin are guaranteed to be no worse than factors of* $O(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})$ *and* $\frac{k}{1+(k-1)(1-\kappa_f)}$ *respectively.*

The guarantee for MMin follows directly from Theorem 3.2, by observing that $|X^*| = k$. We also show a similar asymptotic hardness result, which is quite close to the bounds in observation 3.1. These bounds are improvements over the results of [Svitkina and Fleischer, 2008] whenever $\kappa_f < 1$. Here, MMin is preferable to EA whenever k is small. The following theorem shows that the bound for EA is tight up to poly-log factors.

Theorem 3.5. *For $\kappa_f < 1$ and any $\epsilon > 0$, there exists submodular functions with curvature κ_f such that no poly-time algorithm achieves an approx. factor of $\frac{n^{1/2-\epsilon}}{1+(n^{1/2-\epsilon}-1)(1-\kappa_f)}$ for the SLB problem.*

Proof. Define two monotone submodular functions $h_\kappa(X) = \kappa_f \min\{|X|, \alpha\} + (1-\kappa_f)|X|$ and $f^R_\kappa(X) = \kappa_f \min\{\beta + |X| \cap \bar{R}|, |X|, \alpha\} + (1-\kappa_f)|X|$, where $R \subseteq V$ is a random set of cardinality α. Let α and β be an integer such that $\alpha = x\sqrt{n}/5$ and $\beta = x^2/5$ for an $x^2 = \omega(\log n)$. Also we assume that $k = \alpha$ in this case. Both h_κ and f^R_κ have curvature κ_f. Given an arbitrary $\epsilon > 0$, set $x^2 = n^{2\epsilon} = \omega(\log n)$. Then the ratio between f^R_κ and g^{κ_f} is $\frac{n^{1/2-\epsilon}}{1+(n^{1/2-\epsilon}-1)(1-\kappa_f)}$. Clearly then if any algorithm can achieve better than this bound, it can distinguish between f_R and g which is a contradiction. □

In the following problems, our ground set V consists of the set of edges in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with two distinct nodes $s, t \in V$ and $n = |\mathcal{V}|$, $m = |\mathcal{E}|$. The submodular function is $f : 2^{\mathcal{E}} \to \mathbb{R}$.

Shortest submodular s-t path (SSP). Here, we aim to find an s-t path X of minimum (submodular) length $f(X)$. [Goel et al., 2009] show a $O(n^{2/3})$-approximation with matching curvature-independent lower bound $\Omega(n^{2/3})$. By Theorem 3.2, the curvature-dependent worst-case bound for MMin is $\frac{n}{1+(n-1)(1-\kappa_f)}$ since any minimal s-t path has at most n edges. Similarly, the factor for EA is $O(\frac{\sqrt{m}\log m}{1+(\sqrt{m}\log m-1)(1-\kappa_f)})$. The bound of EA will be tighter for sparse graphs while MMin provides better results for dense ones. We can also show the following curvature-dependent lower bound:

Theorem 3.6. *Given a submodular function with a curvature $\kappa_f > 0$ and any $\epsilon > 0$, no polynomial-time algorithm achieves an approximation factor better than $\frac{n^{2/3-\epsilon}}{1+(n^{2/3-\epsilon}-1)(1-\kappa_f)}$ for the SSP problem.*

Proof. The proof of this follows in very similar lines to the earlier lower bounds using our construction and the matroid constructions in [Goel et al., 2009]. The main idea is to use their multilevel graph, but define adjusted versions of their cost functions. In particular, define $h(X) = \kappa_f \min\{|X|, \alpha\} + (1-\kappa_f)|X|$ and $f_R(X) = \kappa_f \min\{\beta + |X| \cap \bar{R}|, |X|, \alpha\} + (1-\kappa_f)|X|$. In this context R is a randomly chosen s-t path of length $n^{2/3}$ and $\alpha = n^{2/3}$. Similarly the value of $\beta = n^{\epsilon}$. The Chernoff bounds then show that the two functions above are indistinguishable (with high probability) and hence the ratio of the two functions h and f_R then provides the hardness result. □

Minimum submodular s-t cut (SSC): This problem, also known as the cooperative cut problem [Jegelka and Bilmes, 2011c,d], asks to minimize a monotone submodular function f such that the solution $X \subseteq \mathcal{E}$ is a set of edges whose removal disconnects s from t in $\mathcal{G}$. Using curvature refines the lower bound in [Jegelka and Bilmes, 2011c]:

Theorem 3.7. *No polynomial-time algorithm can achieve an approximation factor better than $\frac{n^{1/2-\epsilon}}{1+(n^{1/2-\epsilon}-1)(1-\kappa_f)}$, for any $\epsilon > 0$, for the SSC problem with a submodular function of curvature κ_f.*

Proof. This proof follows along the lines of the results shown above. It uses the construction from [Jegelka and Bilmes, 2011c]. □

Theorem 3.2 implies an approximation factor of $O(\frac{\sqrt{m}\log m}{(\sqrt{m}\log m - 1)(1-\kappa_f)+1})$ for EA and a factor of $\frac{m}{1+(m-1)(1-\kappa_f)}$ for MMin, where $m = |\mathcal{E}|$ is the number of edges in the graph. By Theorem 3.7, the factor for EA is tight for sparse graphs. Specifically for cut problems, there is yet another useful surrogate function that is exact on local neighborhoods. [Jegelka and Bilmes, 2011c] demonstrate how this approximation may be optimized via a generalized maximum flow algorithm that maximizes a *polymatroidal network flow* [Lawler and Martel,

1982]. This algorithm still applies to the combination $\hat{f} = \kappa_f \hat{f}^\kappa + (1-\kappa_f) f^m$, where we only approximate f^κ. We refer to this approximation as Polymatroidal Network Approximation (PNA).

Corollary 3.4. *Algorithm PNA achieves a worst-case approximation factor of* $\frac{n}{2+(n-2)(1-\kappa_f)}$ *for the cooperative cut problem.*

For dense graphs, this factor is theoretically tighter than that of the EA approximation.

Proof. We use the polymatroidal network flow construction from [Jegelka and Bilmes, 2011c], where the approximation $\hat{f}$ is defined via a partition of the ground set, and is separable over groups of edges. This approximation can be solved efficiently via generalized flows in polynomial time [Jegelka and Bilmes, 2011a,c]. Moreover adding a modular term (for the modulation) does not increase the complexity of the problem. This approximation satisfies $f^\kappa(X) \leq \hat{f}^\kappa(X) \leq \frac{n}{2} f^\kappa(X)$ for all cuts $X \in \mathcal{C}$.We then convert this expression in the form of Theorem 3.3 as $\frac{2\hat{f}^\kappa(X)}{n} \leq f^\kappa(X) \leq \hat{f}^\kappa(X)$. Then define $\hat{f}(X) \triangleq \kappa_f \frac{2\hat{f^\kappa}(X)}{n} + (1-\kappa_f) \sum_{j \in X} f(j)$, and using theorem 3.3, it implies that:

$$\hat{f}(X) \leq f(X) \leq \frac{n}{2+(n-2)(1-\kappa_f)} f(X) \tag{3.19}$$

Then let $\hat{X}$ be the minimizer of $\hat{f}(X)$ over $\mathcal{C}$ (using the generalized flows [Jegelka and Bilmes, 2011c]). It then follows that (let $\alpha = \frac{n}{2+(n-2)(1-\kappa_f)}$): $f(\hat{X}) \leq \alpha \hat{f}(\hat{X}) \leq \alpha \hat{f}(X^*) \leq f(X^*)$ where X^* is the optimal solution of f over $\mathcal{C}$. □

Minimum submodular spanning tree (SST). Here, $\mathcal{C}$ is the family of all spanning trees in a given graph $\mathcal{G}$. Such constraints occur for example in power assignment problems [Wan et al., 2002]. [Goel et al., 2009] show a curvature-independent optimal approximation factor of $O(n)$ for this problem.

Observation 3.2. *For the minimum submodular spanning tree problem, algorithm MMin achieves an approximation guarantee, which is no worse than* $\frac{n-r}{1+(n-r-1)(1-\kappa_f)}$*, where* r *is the number of connected components of* $\mathcal{G}$*.*

Proof. This result follows directly from Theorem 3.2 and the fact that $|X^*| = n - r$. □

In this case, Algorithm EA in fact provides slightly worse guarantees. Moreover the bound for MMin is optimal:

Theorem 3.8. *For the class of submodular functions with curvature $\kappa_f < 1$, no poly-time algorithm can achieve an approximation factor of $\frac{n^{1-3\epsilon}}{1+(n^{1-3\epsilon}-1)(1-\kappa_f)+\delta\kappa_f}$ for the SST problem for any $\epsilon, \delta > 0$.*

Proof. In this case, we use the construction of [Goel et al., 2009], and define $f_\kappa^R(X) = \kappa_f \min\{|X \cap \bar{R}| + min\{|X \cap R|, \beta\}, \alpha\} + (1-\kappa_f)|X|$, and $g^{\kappa_f}(X) = \kappa_f min\{|X|, \alpha\} + (1-\kappa_f)|X|$, where $\alpha = n^{1+\epsilon}$, $\beta = n^{3\epsilon}(1+\delta)$ and $|R| = \alpha$. For the formal graph construction, see [Goel et al., 2009]. Then with high probability R is connected in the graph [Goel et al., 2009]. Since f_R and g are indistinguishable with high probability, so are f_κ^R and g^{κ_f}. Then notice that the minimum value of f_κ^R and g^{κ_f} are $\kappa_f\beta + (1-\kappa_f)n$ and n respectively, and it is clear that the ratio between them is better than $\frac{n^{1-3\epsilon}}{1+(n^{1-3\epsilon}-1)(1-\kappa_f)+\delta\kappa_f}$. Hence if any algorithm performs better than this, it will be able to distinguish f_R and g with high probability, which is a contradiction. □

An analogous analysis applies to combinatorial constraints like Steiner trees [Goel et al., 2009].

Minimum Submodular Perfect Matching (SPM): Here, we aim to find a perfect matching in a graph that minimizes a monotone submodular function. Theorem 3.2 implies that an MMin approximation will achieve an approximation factor of at most $\frac{n}{2+(n-2)(1-\kappa_f)}$. This bound is also tight:

Theorem 3.9. *Given a submodular function with a curvature $\kappa_f > 0$ and any $\epsilon > 0$, no polynomial-time algorithm achieves an approximation factor better than $\frac{n^{1-3\epsilon}}{2+(n^{1-3\epsilon}-2)(1-\kappa_f)+2\delta\kappa_f}$ for the SPM problem.*

Proof. We use the same submodular functions as the spanning tree case, and it can be shown [Goel et al., 2009] that with high probability the set R contains a perfect matching and the two functions are indistinguishable. Taking the ratio of g^{κ_f} and $f_R^{\kappa_f}$, provides the above result. □

Minimum submodular edge cover: The minimum submodular edge cover involves finding an edge cover (subset of edges covering all vertices), with minimum submodular cost. This problem has been investigated in [Iwata and Nagano, 2009], and they show that this problem is $O(n)$ hard. Algorithm MMin provides an approximation guarantee which is no worse than $\frac{2n}{2+(n-2)(1-\kappa_f)}$. We can show a almost matching hardness lower bound for this problem.

Theorem 3.10. *Given a submodular function, with curvature coefficient κ_f and any $\epsilon, \delta > 0$, there cannot exist a polynomial-time approximation algorithm, which achieves an approximation better than $\frac{n^{1-3\epsilon}}{2+(n^{1-3\epsilon}-2)(1-\kappa_f)+2\delta\kappa_f}$ for the minimum submodular edge cover problem.*

Proof. We can use the construction of [Iwata and Nagano, 2009] to show this. However a simple observation shows that a perfect matching is also an edge cover, and hence the hardness of edge cover has to be at least as much as the hardness of perfect matchings. □

3.6 Constrained Non-Monotone Submodular Minimization

In the previous section, we assumed that the submodular function f was monotone. In this section, we show that we can extend any algorithm working with monotone functions to handle non-monotone functions, via some simple tricks, if we make some additional assumptions. In particular, we investigate two assumptions, one that the constraints are up monotone, and second, the submodular function being *approximately* monotone.

3.6.1 Up-monotone Constraints

First we assume that the the constraint $\mathcal{C}$ is up-monotone – A family $\mathcal{C}$ of feasible sets is up-monotone if $S \in \mathcal{C}$ implies that all supersets $T \supseteq S$ are also feasible ($T \in \mathcal{C}$). Examples of

up-monotone constraints include cardinality lower bound constraints, covers etc. Define a monotone extension of f as

$$f^m(X) = \min_{Y:Y\supseteq X} f(Y). \tag{3.20}$$

The following Lemma shows that minimizing f^m is *equivalent* to minimizing f for up-monotone constraints:

Lemma 3.15. *If f is submodular, then f^m is monotone submodular and computable in polynomial time. If $\mathcal{C}$ is up-monotone, then*

$$\min_{X\in\mathcal{C}} f(X) = \min_{X\in\mathcal{C}} f^m(X). \tag{3.21}$$

The solution for f can, moreover, be recovered: given an approximate minimizer $\widehat{X}$ of f^m over $\mathcal{C}$, the set $Z \in \operatorname{argmin}_{Y:Y\supseteq\hat{X}} f(X)$ satisfies the same approximation bounds for f over $\mathcal{C}$ as $\widehat{X}$ has for f^m.

Proof. It is well known that, for any submodular f, the function f^m is monotone submodular [Fujishige, 2005]. To show the equivalence (3.21), let X^{m*} be the exact minimizer of f^m over $\mathcal{C}$. The definition of f^m implies that there exists a set $X \supseteq X^{m*}$ such that $f^m(X^{m*}) = f(X)$ and moreover, since $\mathcal{C}$ is up-monotone, $X \in \mathcal{C}$. Hence $\min_{X\in\mathcal{C}} f(X) \leq f(X) = \min_{X\in\mathcal{C}} f^m(X)$. Conversely, let X^* be the minimizer of f under $\mathcal{C}$. Then $f^m(X^*) = \min_{X\supseteq X^*} f(X) \leq f(X^*)$, and therefore $\min_{Y\in\mathcal{C}} f^m(Y) \leq f^m(X^*) = \min_{X\in\mathcal{C}} f(X)$.

To show the second part, let X^m be an approximate optimizer of f^m with an approximation factor α, i.e., $f^m(X^m) \leq \alpha f^m(X^{m*})$. From the first part, it follows that both f and f^m have the same optimal value in $\mathcal{C}$. The definition of f^m implies that there exists a set $Y^m \supseteq X^m$ such that $f(Y^m) = f^m(X^m)$. Hence $Y^m \in \operatorname{argmin}_{Y\supseteq X^m} f(Y)$ satisfies that $f(Y^m) = f^m(X^m) \leq \alpha f^m(X^m) = \min_{X\in\mathcal{C}} f(X)$. □

While f^m can be obtained directly from f via submodular function minimization, it can be quite costly for general submodular functions. Fortunately however, many useful subclasses of submodular functions admit much faster algorithms. For example, those functions expressible

as generalized graph cuts [Jegelka et al., 2011] can be minimized via max flows. By using f^m instead of f, any algorithm for constrained minimization of monotone submodular functions straightforwardly generalizes to the non-monotone case.

Corollary 3.5. *Running MMin-I on f^m obtains a worst case approximation factor of $|X^*|$, for the problem $\min\{f(X)|X \in \mathcal{C}\}$, as long as $\mathcal{C}$ is up-monotone, and a linear function can be exactly optimized under it. Similarly, running EA on f^m provides a worst case guarantee of $O(\sqrt{n})$.*

For example, with cardinality lower bound constraints $\{X : |X| \geq k\}$, MMin provides a worst case guarantee of k while EA gives a factor of $O(\sqrt{n})$. The upmonotonicity of the constraints is crucial however for this to work. Hence constraints like s-t cuts, s-t paths etc. which are not up-monotone do not inherit these guarantees. Handling such constraints with non-monotone submodular functions remains open.

3.6.2 Approximately Monotone Submodular Functions

Next we assume that the function f is approximately monotone. In this case, we do not need to assume the upmonotonicity of $\mathcal{C}$. Moreover, we can provide an approximation guarantee by directly running MMin on f. We do not need to construct the proxy function f^m.

Lemma 3.16. *Given a non-monotone submodular function f, MMin provides a worst case factor of $|X^*|\nu_f^{1,|X^*|}$ for the problem $\min\{f(X)|X \in \mathcal{C}\}$, assuming that a linear function can be exactly optimized under $\mathcal{C}$.*

Proof. Note that $f(S) \leq \sum_{j \in S} f(j)$. Moreover, by the definition of $\nu^{1,|S|}$ it holds that $f(j) \leq f(S)\nu^{1,|S|}$. Hence, $f(S) \leq \sum_{j \in S} f(j) \leq |S|\nu^{1,|S|}$. Following the proof technique of Theorem 3.2, the result follows. □

Again in the case of cardinality lower bound constraints, this gives a factor of $k\nu_f^{1,k}$. Similarly, in the case of spanning trees, we have $n\nu_f^{1,n}$. While this technique provides somewhat weaker bounds, it is much more efficient than the reduction above, since we do not

need to construct f^m. Moreover, this works for any constraint, as long as a linear function can be optimized over it. On the negative side, the bounds will really make sense if $\nu_f^{1,|X^*|}$ is bounded above.

3.7 Experiments

We will next see that, apart from its theoretical properties, our algorithms (particularly, MMin) also perform well in practice. In fact, we shall see that in the constrained setting, MMin performs comparable to EA, which often has the tightest theoretical guarantees. We implement and compare algorithms using Matlab and the SFO toolbox [Krause, 2010].

3.7.1 Unconstrained minimization

We first study the results in Section 3.4 for contracting the lattice of possible minimizers. We measure the size of the new lattices relative to the ground set. Applying MMin-I and II (lattice $\mathcal{L}_+$) to Iwata's test function [Fujishige and Isotani, 2011], we observe an average reduction of 99.5% in the lattice. MMin-III (lattice $\mathcal{L}$) obtains only about 60% reduction. Averages are taken for n between 20 and 120.

In addition, we use concave over modular functions $\sqrt{w_1(X)} + \lambda w_2(V \backslash X)$ with randomly chosen vectors w_1, w_2 in $[0,1]^n$ and $n = 50$. We also consider the application of selecting limited vocabulary speech corpora. [Lin and Bilmes, 2011a, Jegelka et al., 2011] use functions of the form $\sqrt{w_1(\Gamma(X))} + w_2(V \backslash X)$, where $\Gamma(X)$ is the neighborhood function of a bipartite graph. Here, we choose $n = 100$ and random vectors w_1 and w_2. For both function classes, we vary λ such that the optimal solution X^* moves from $X^* = \emptyset$ to $X^* = V$. The results are shown in Figure 3.2. In both cases, we observe a significant reduction of the search space. When used as a preprocessing step for the minimum norm point algorithm (MN) [Fujishige and Isotani, 2011], this pruned lattice speeds up the MN algorithm accordingly, in particular for the speech data. The dotted lines represent the relative time of MN including the respective preprocessing, taken with respect to MN without preprocessing. Figure 3.2 also shows the average results over 10 random choices of weights in both cases. In order

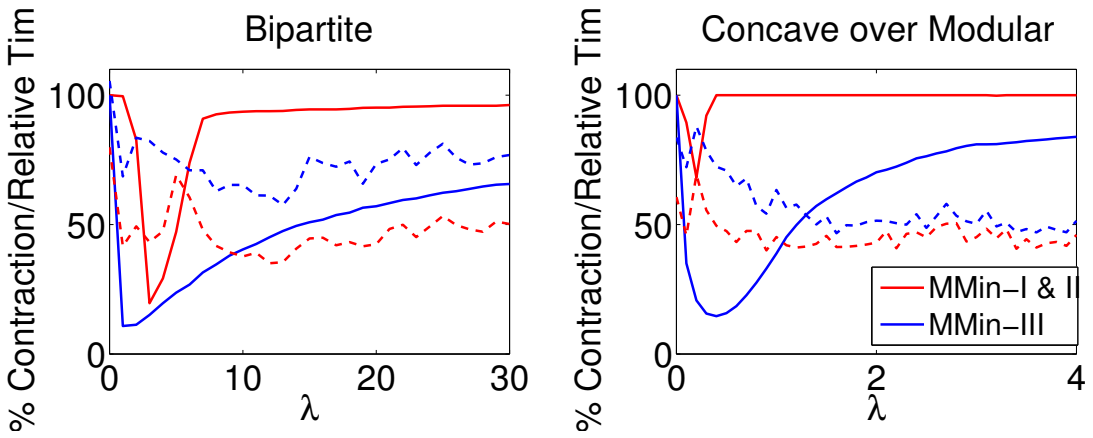

Figure 3.2: Lattice reduction (solid line), and runtime (%) of MMin+min-norm relative to unadorned min-norm (dotted).

to obtain accurate estimates of the timings, we run each experiment 5 times and take the minimum of these timing values.

3.7.2 Constrained minimization.

For constrained minimization, we compare MMin-I to two methods: a simple algorithm (MU) that minimizes the upper bound $g(X) = \sum_{i \in X} f(i)$ [Goel et al., 2009], and the more complex Ellipsoidal Approximation algorithm (EA). MU is identical to the first iteration of MMin-I, and has the same theoretical bounds of MMin (i.e Theorem 3.2). EA has a superior dependence on n, albeit a weaker dependence on curvature.

We show two experiments: the theoretical worst-case and average-case instances.

Worst case. We first demonstrate the worst case behavior of MMin and EA, and their precise dependence on curvature. We use a very hard cost function [Goemans et al., 2009]

$$f^R(X) = \min\{|X \cap \bar{R}| + \beta, |X|, \alpha\}, \tag{3.22}$$

where $\alpha = n^{1/2+\epsilon}$ and $\beta = n^{2\epsilon}$, and R is a random set such that $|R| = \alpha$. This function is the theoretical worst case. Figure 3.3 shows results for cardinality lower bound constraints; the

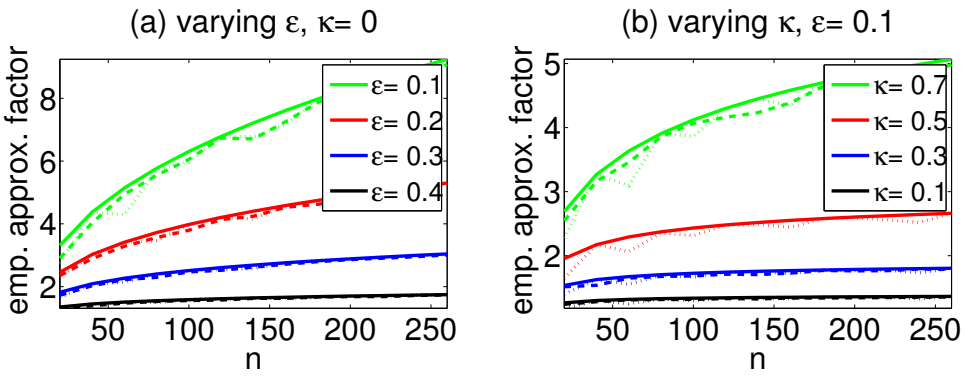

Figure 3.3: Minimization of f_κ^R (Equation 3.23) for cardinality lower bound constraints. (a) fixed $\kappa = 0$, $\alpha = n^{1/2+\epsilon}, \beta = n^{2\epsilon}$ for varying ϵ; (b) fixed $\epsilon = 0.1$, but varying κ. Dashed lines: MMin (or MU), dotted lines: EA, solid lines: worst case theoretical bound.

results for other, more complex constraints are similar. As ϵ shrinks, the problem becomes harder. This function has curvature $\kappa_f = 1$. To obtain a function with specific curvature κ, we define

$$f_\kappa^R(X) = \kappa f^R(X) + (1 - \kappa)|X|. \tag{3.23}$$

In all our experiments, we take the average over 20 random draws of R. We first set $\kappa = 1$ and vary ϵ. Figure 3.3(a) shows the empirical approximation factors obtained using EA and MMin, and the theoretical bound. The empirical factors follow the theoretical results very closely. Empirically, we also see that the problem becomes harder as ϵ decreases. Moreover, EA and MMin achieve about the same empirical approximation factors, which matches the theoretical guarantee of $n^{1/2-\epsilon}$. Furthermore, for this class of functions, MMin essentially works like MU, i.e it converges after a single iteration. Hence we show them together. Next we fix $\epsilon = 0.1$ and vary the curvature κ in f_κ^R. Figure 3.3(b) illustrates that the theoretical and empirical approximation factors improve significantly as κ decreases. Hence, much better approximations than the previous theoretical lower bounds are possible if κ is not too large.

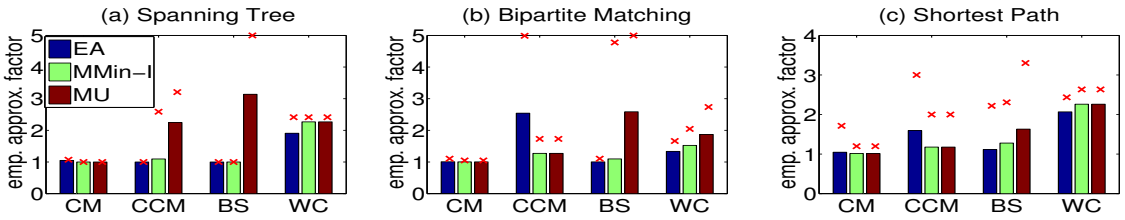

Figure 3.4: Constrained minimization for average-case instances on spanning trees, matchings and paths. The bars are average approximation factors and crosses show the worst observed results. CM - Concave over Mod., CCM - Clust. Concave Mod., BS - Best Set and WC - Worst Case

This observation can be very important in practice. Here, too, the empirical upper bounds follow the theoretical bounds very closely. In particular, we see the theoretical and empirical effect of curvature: as n grows, the bounds saturate and approximate a constant $1/(1-\kappa)$ – they do not grow polynomially in n. Overall, we see that the empirical results quite closely follow our theoretical results, and that, as the theory suggests, curvature significantly affects the approximation factors.

Average case. We next compare the algorithms on more realistic functions that occur in applications. Figure 3.4 shows the empirical approximation factors for minimum submodular-cost spanning tree, bipartite matching, and shortest path. We use four classes of randomized test functions: (1) concave (square root or log) over modular (CM), (2) clustered CM (CCM) of the form $f(X) = \sum_{i=1}^{k} \sqrt{w(X \cap C_k)}$ for clusters $C_1, \cdots, C_k$, (3) Best Set (BS) functions where the optimal feasible set R is chosen randomly ($f(X) = I(|X \cap R| \geq 1) + \sum_{j \in R \setminus X} w_j$) and (4) worst case-like functions (WC) similar to equation (3.22). Functions of type (1) and (2) have been used in speech and computer vision [Lin and Bilmes, 2011c, Jegelka and Bilmes, 2011d, Iyer and Bilmes, 2012b] and have reduced curvature ($\kappa_f < 1$). Functions of type (3) and (4) have $\kappa_f = 1$. In all four cases, we consider both sparse and dense graphs, with random weight vectors w. The plots show averages over 20 instances of these graphs. For

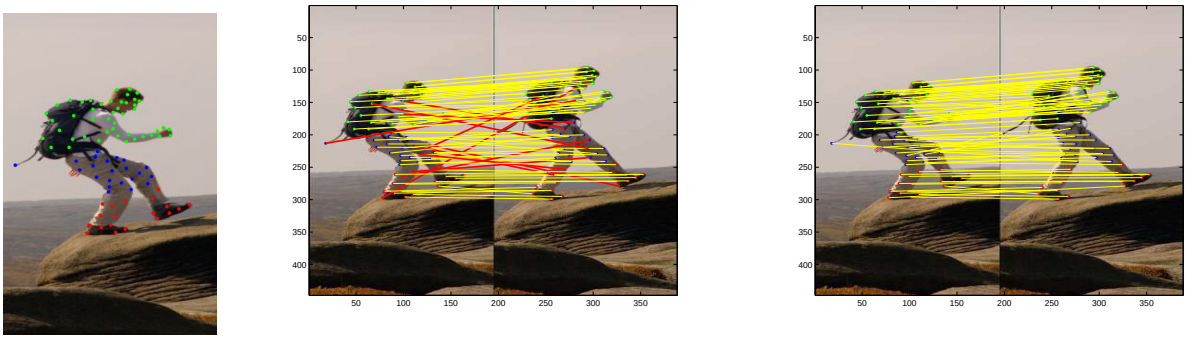

Figure 3.5: The figure on the left shows the clustering used in cooperative matching, while the middle and the right ones show the results with cooperative matching and bipartite matching respectively. The source of this image is from [Jegelka et al., 2013b].

sparse graphs, we consider grid like graphs in the form of square grids, grids with diagonals and cubic grids. For dense graphs, we sparsely connect a few dense cluster subgraphs. For matchings, we restrict ourselves to bipartite graphs, and consider both sparse and dense variants of these.

First, we observe that in many cases, MMin clearly outperforms MU. This suggests the practical utility of more than one iteration. Second, despite its simplicity, MMin performs comparably to EA, and sometimes even better. In summary, the experiments suggest that the complex EA only gains on a few worst-case instances, whereas in many (average) cases, MMin yields near-optimal results (factor 1–2). In terms of running time, MMin is definitely preferable: on small instances (for example $n = 40$), our Matlab implementation of MMin takes 0.2 seconds, while EA takes an hour. On larger instances ($n = 100$), the running times differ on the order of seconds versus couple of hours.

3.7.3 Real world Experiments: Cooperative Matchings

The problem of matching key-points in images, also called image correspondence, is a very important problem in computer vision [Ogale and Aloimonos, 2005]. The simplest model for this problem is to do matching with linear scores, i.e., bipartite matching [Jegelka et al.,

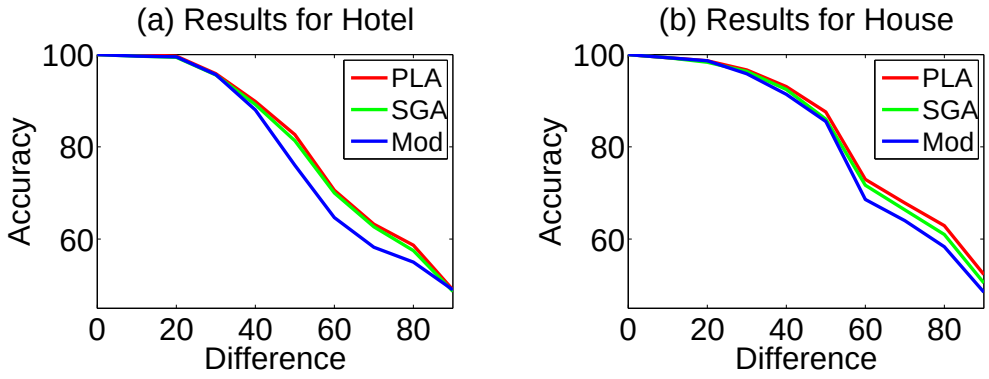

Figure 3.6: (a) and (b) give the results on the House and Hotel dataset, showing that MMin performs comparably to an exact algorithm PLA [Iyer and Bilmes, 2014a]. Moreover, both submodular methods perform much better than standard matching (Mod).

2013b], called a *linear assignment*. This model does not represent interaction between the pixels. For example, we see many obviously spurious matches in figure 3.5b. Many models try to capture this, via, for example via quadratic assignments [Caetano et al., 2009]. Instead of just looking at the best linear assignment, the quadratic models try to incorporate pairwise constraints. This is also called graph matching. We introduce a new and different model here. First, we cluster key-points, separately in each of the two images, into k clusters. Figure 3.5a shows a particular clustering of an image into $k = 3$ groups. The clustering can be performed based on the pixel color map, or simply the distance of the key-points. That is, each image has k clusters. Let $\{V_i^{(1)}\}_{i=1}^k$ and $\{V_i^{(2)}\}_{i=1}^k$ be the two sets of clusters. We then compute the linear assignment problem, letting $\mathcal{M} \subseteq \mathcal{E}$ be the resulting maximum matching. We then partition the edge set $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \mathcal{E}_k \cup \mathcal{E}'$ where $\mathcal{E}_i = \mathcal{M} \cap (V_\ell^{(1)} \times V_s^{(2)})$ for $\ell, s \in \{1, 2, \ldots, k\}$ corresponding to the i'th largest intersection, and $\mathcal{E}' = \{\mathcal{E} \setminus \cup_{i=1}^k \mathcal{E}_i\}$ are the remaining edges either that were not matched or that did not lie within a frequently associated pair of image key-point clusters. We then define a submodular function as follows:

$$f(S) = \sum_{i=1}^{k} \psi_i(w(S \cap \mathcal{E}_i)) + w(S \cap \mathcal{E}'), \tag{3.24}$$

which provides an additional discount to the edges $\{\mathcal{E}_i\}_{i=1}^{k}$ corresponding to key-points that were frequently associated in the initial pass. Figures 3.5b and 3.5c shows how the submodular matchings gains viz-a-viz the simple bipartite matching, with $k = 3$. The minimum matching approach obtains many spurious matches between clusters (shown in red), while the cooperation described above reduces these spurious matches. The cooperative matching improves the performance over the modular method on these images, by about 20%.

We also test the performance of our algorithms on the CMU House and Hotel dataset [Caetano et al., 2009]. The house dataset has 111 images, while the hotel dataset has 101 images. We consider all possible pairs of images, with differences between the two images ranging from $0 : 10 : 90$ in both cases. We consider three algorithms: PLA (which is an almost exact algorithm for this problem [Iyer and Bilmes, 2014a], MMin and the simple modular bipartite matching as a baseline (Mod). Man and PLA both use the submodular function from Equation (3.24). Again, we set $k = 3$. The results are shown in Figure 3.5(d-e) where we observe that the submodular methods beat Mod by about $3 - 5\%$ on average. Moreover, we observe that MMin performs comparably to PLA, while being computationally much less expensive. We do not compare the ellipsoidal approximation algorithm (EA) [Goemans et al., 2009], mainly because it is too slow to run on real world problems.

3.8 Discussion

In this chapter, we introduced a general MMin framework for submodular minimization algorithms. This framework is akin to the class of algorithms for other submodular optimization problems we shall consider in this thesis (i.e submodular maximization, difference of submodular minimization, and submodular minimization subject to submodular constraints. Moreover, our framework may be viewed as a special case of a proximal minimization algorithm that uses Bregman divergences derived from submodular functions [Iyer et al., 2012]. To our knowledge this is the first generic and unifying framework of algorithms for submodular minimization.

Following the conference version of our chapter [Iyer et al., 2013b], which proposed

the superdifferential based MMin and MMax scheme of algorithms, several papers have built on the techniques proposed in this chapter. In particular, the MMin paradigm of algorithms have been used for several real world problems, as an extremely scalable and practical class of algorithms. [Jegelka and Bilmes, 2011d] studied MMin in the context of the cooperative cut problem, and observed that MMin scaled very well for image segmentation. Similarly, in this thesis, we consider the problem of cooperative matching [Iyer and Bilmes, 2014a], and again notice that MMin performs comparably to more complicated algorithms. [Khalil et al., 2014] investigate the problem of diffusion models in social networks, and show how the problem of facilitating desirable spreads by adding edges in a social network is a constrained submodular minimization under cardinality constraints. They use our Majorization-Minimization framework, and observe that the algorithm scales almost linearly with time, and correspondingly can be used for very large networks. Finally [Heng et al., 2014] use our algorithm for the information path planning algorithm, where they observed that the previously proposed techniques were prohibitively slow. Our framework on the other hand, scaled to very large scale graphs.

A number of papers have built on the theoretical contributions of this work. [Djolonga and Krause, 2014] use the idea of sub and super-differentials, to provide upper and lower bound modular functions as bounds on the partition function for Log-Submodular distributions. They also use our curvature dependent bound on the modular upper bound (Lemma 3.10) to provide a bound on the approximation on the Log-Partition function. Similarly, [Iyer and Bilmes, 2014b] use the supergradients and curvature to provide bounds on the partition function on a class of submodular distributions. [Niepert et al., 2014] study notions of conditional independence for set functions, and investigate the effect of curvature in this connection. Similarly, [Kusner, 2014] extend some of the results here to adaptive submodular functions, and investigate the role of supergradients and parameters like the curvature and submodularity ratio in this setting.

Chapter 4

SUBMODULAR FUNCTION MAXIMIZATION

4.1 Introduction

In this chapter, we investigate submodular function maximization:

$$\text{Problem 2: } \max_{X \in \mathcal{C}} f(X)$$

where $f : 2^V \to \mathbb{R}$ is a submodular set function on subsets of a ground set $V = \{1, 2, \cdots, n\}$, and $\mathcal{C} \subseteq 2^V$ is a family of feasible solution sets. The set $\mathcal{C}$ could express, for example, that solutions must be an independent set in a matroid, a limited budget knapsack, or a cut (or spanning tree, path, or matching) in a graph.

This chapter provides a Minorization-Maximization scheme (which we denote by MMax) for submodular maximization problems, based on the submodular semigradients. This class of algorithms are very similar in spirit, and somewhat complementary, to the MMin framework proposed in Chapter 3. We demonstrate that many recently studied combinatorial algorithms may be viewed as special cases of a generic minorize-maximize framework that relies on discrete subdifferentials. This framework encompasses state-of-the-art greedy and local search techniques, and provides a rich class of very practical algorithms for submodular maximization. As a whole, this framework offers a new unifying perspective and basis for treating submodular maximization problems in both the constrained and unconstrained case. Similar to submodular minimization, we investigate the role of complexity parameters like the curvature, monotonicity ratio and submodularity ratio in the approximation guarantees we provide in this chapter.

4.2 Motivating Applications

Submodular maximization arises naturally in many subset extraction problems. From a modeling perspective, they naturally model aspects of coverage, diversity and information which seem relevant in many subset selection problems. We list some of these motivating applications below:

Sensor placement: A number of models for sensor placement have been closely related and naturally modeled as instances of submodular maximization [Krause et al., 2008b, 2006].

Summarization: A number of summarization problems, including document and news summarization [Lin and Bilmes, 2010, 2011c, 2012], image collection summarization [Simon et al., 2007] etc. are becoming increasingly important machine learning problems. Natural properties of summaries are that they are diverse and simultaneously cover the information from the collection. A number of submodular functions naturally model these properties, and not surprisingly they have empirically provided large improvements in performance.

Diverse web search and retrieval: The problem of finding a diverse set of articles in information retrieval and web search can be modeled as submodular maximization [He et al., 2012]. These can also be extended to obtaining diverse collections for image retrieval.

Data subset selection: Present day machine learning applications tend to draw large amounts of training data. Larger data sets come with increased demands on computational resources; moreover, they tend to include redundant information as their size increases. Therefore, the performance gain curves of large-scale systems with respect to the amount of training data often show diminishing return. Submodular functions naturally capture this notion, and pose as promising models for these problems. Moreover, these models have shown promising results, for example in the context of speech data subset selection [Lin and Bilmes, 2009, Wei et al., 2013].

Influence maximization: Influence maximization has been an important problem in social network theory. Interestingly, a number of models for this problem are instances of submodular maximization [Kempe et al., 2003].

Determinantal Point Processes: The Determinantal Point Processes (DPPs) which have found numerous applications in machine learning [Kulesza and Taskar, 2012] have been shown to naturally model diversity. They are closely related to submodular functions, and are in fact log-submodular distributions. In particular, the MAP inference problem is a form of non-monotone submodular maximization.

4.3 Combinatorial Algorithmic Framework

4.3.1 The Minorization-Maximization (MMax) framework

With the submodular subgradients (defined in Section 1.4.1), we can define a generic MMax algorithm. In each iteration, the algorithm maximizes a modular lower formed via the current solution Y.

$$m_f^Y(X) = f(Y) + h_Y(X) - h_Y(Y) \ \leq f(X). \tag{4.1}$$

These bounds are tight at the current solution, i.e they satisfy $m_f^Y(Y) = f(Y)$, and $m_f^Y(X) \leq f(X), \forall X \subseteq V$. In almost all cases, optimizing the modular approximation is much faster than optimizing the original cost function f.

Algorithm 2 Subgradient Ascent Algorithm for Problem 2

Start with an arbitrary X^0.

while *until convergence* $(X^{i-1} = X^i)$ **do**

- Pick a subgradient h_{X^t} at X^t
- $X^{t+1} := \text{argmax}_{X \in \mathcal{C}}\, m_f^{X^t}(X)$
- $t \leftarrow t + 1$

Algorithm 2 show our discrete MMax scheme for submodular maximization, for both

constrained and unconstrained settings. Since we are maximizing a tight lower bound, the algorithm must make progress in each iteration.

Lemma 4.1. *Algorithm 2 monotonically increases the objective function value at every iteration for Problems 2, as long as a linear function can be exactly optimized over $\mathcal{C}$.*

Proof. Note that $f(X^{t+1}) \geq m_f^{X^t}(X^{t+1})$. Since X^{t+1} minimizes $m_f^{X^t}$, it follows that

$$f(X^{t+1}) \geq m_f^{X^t}(X^{t+1}) \geq m_f^{X^t}(X^t) = f(X^t). \tag{4.2}$$

□

Note that unlike subgradient descent used in convex analysis, this is a discrete algorithm. Moreover, note that we use the subgradients for submodular maximization (recall that in the case of submodular minimization, we used the supergradients), and hence we call this Subgradient Ascent.

Just like for minimization, for submodular maximization too we obtain a family of algorithms where each member is specified by a distinct schedule of subgradients in Algorithm 2. We will only select subgradients that are vertices of the subdifferential, i.e., each subgradient corresponds to a permutation of V. For any of those choices, Algorithm 2 (which we shall henceforth call MMax) converges quickly. To bound the running time, we assume that we proceed only if we make sufficient progress, i.e., if $f(X^{t+1}) \geq (1+\eta)f(X^t)$.

Lemma 4.2. *MMax with $X^0 = \operatorname{argmax}_j f(j)$ runs in time $O(T \log_{1+\eta} n)$, where T is the time for maximizing a modular function subject to $X \in \mathcal{C}$.*

Proof. Let X^* be the optimal solution, then

$$f(X^*) \leq \sum_{i \in X^*} f(j) \leq n \max_{j \in V} f(j) = n f(X^0). \tag{4.3}$$

Furthermore, we know that $f(X^t) \geq (1+\eta)^t f(X^0)$. Therefore, we have reached the maximum function value after at most $(\log n)/\log(1+\eta)$ iterations. □

Constraints	Property	Subgradient	Approx. bound	Lower bound
Unconstrained	non-monotone	Random (RA/RP)	$1/4$	$1/2$
Unconstrained	symmetric	Random (RA/RP)	$1/2$	$1/2$
Unconstrained	non-monotone	Local Search based (RLS/DLS)	$1/3 - \eta$	$1/2$
Unconstrained	symmetric	Local Search based (RLS/DLS)	$1/2 - \eta$	$1/2$
Unconstrained	non-monotone	Bi-directional greedy (BG)	$1/3$	$1/2$
Unconstrained	non-monotone	Rand. bi-directional greedy (RBG)	$1/3$	$1/2$
Unconstrained	non-monotone	Greedy (GR)	$1/n$	$1/2$
Cardinality	monotone	Random (RA/RP)	k/n	$\frac{1}{\kappa_f}(1 - e^{-\kappa_f})$
Cardinality	non-monotone	Random (RA/RP)	$k(n-k)/n(n-1)$	$1/2$
Cardinality	symmetric	Random (RA/RP)	$2k(n-k)/n(n-1)$	$1/2$
Cardinality	monotone	Greedy (GR)	$\frac{1}{\kappa_f}(1 - e^{-\kappa_f})$	$\frac{1}{\kappa_f}(1 - e^{-\kappa_f})$
Cardinality	non-monotone	Greedy (GR)	$1/k$	$1/2$
Cardinality	approx-monotone	Greedy (GR)	$(1 - 1/e)/\nu_f^{k,2k}$	$1 - 1/e$
Cardinality	non-monotone	Random-Greedy (RG)	$1/e$	$1/2$
Matroid	monotone	Greedy (GR)	$1/(1 + \kappa_f)$	$\frac{1}{\kappa_f}(1 - e^{-\kappa_f})$
Knapsack	monotone	Greedy (GR)	$1 - 1/e$	$1 - 1/e$

Table 4.1: Summary of the approximation factors obtained through specific subgradients for various forms of submodular maximization (see text for details).

In practice, we observe that MMax terminates within 3-10 iterations. We next consider specific subgradients and their theoretical implications. Our results rely on the observation that many maximization algorithms actually compute a specific subgradient and run MMax with this subgradient. To our knowledge, this observation is new.

4.4 Unconstrained Submodular Maximization

For unconstrained problems, we assume the submodular function to be non-monotone (the problem is trivial for monotone submodular functions). The approximation guarantees obtained via different choices of subgradients are summarized in Table 4.1.

4.4.1 Random Permutation (RA/RP).

In iteration t, we randomly pick a permutation σ^t that defines a subgradient at X^{t-1}, i.e., X^{t-1} is assigned to the first $|X^{t-1}|$ positions. This corresponds to randomly choosing an extreme subgradient in subdifferential of X^t. At $X^0 = \emptyset$, this can be any permutation. Stopping after the first iteration we can achieve an approximation factor of $1/4$ in expectation, and $1/2$ for symmetric functions. We call this procedure Random Permutation (RP). Making further iterations only improves the solution. We call this Random Adaptive (RA).

Lemma 4.3. *When running Algorithm RP (or RA) with $X^0 = \emptyset$, it holds after one iteration that $\mathbf{E}(f(X^1)) \geq \frac{1}{4} f(X^*)$ if f is a general non-negative submodular function, and $\mathbf{E}(f(X^1)) \geq \frac{1}{2} f(X^*)$ if f is symmetric.*

Proof. Each permutation has the same probability $1/n!$ of being chosen. Therefore, it holds that

$$\mathbf{E}(f(X^1)) = \mathbf{E}_\sigma(\max_{X \subseteq V} h^\sigma_\emptyset(X)) \tag{4.4}$$

$$= \frac{1}{n!} \sum_\sigma \max_{X \subseteq V} h^\sigma_\emptyset(X) \tag{4.5}$$

Let $\emptyset \subseteq S_1^\sigma \subseteq S_2^\sigma \cdots S_n^\sigma = V$ be the chain corresponding to a given permutation σ. We can bound

$$\max_{X \subseteq V} h_\emptyset^\sigma(X) \geq \sum_{k=0}^{n} \frac{\binom{n}{k}}{2^n} f(S_k^\sigma) \tag{4.6}$$

because $\max_{X \subseteq V} h_\emptyset^\sigma(X) \geq f(S_k^\sigma), \forall k$ and $\sum_{k=0}^{n} \frac{\binom{n}{k}}{2^n} = 1$. Together, Equations (4.5) and (4.6) imply that

$$\begin{aligned}
\mathbf{E}(f(X^1)) &\geq \mathbf{E}_\sigma(\max_{X \subseteq V} h_\emptyset^\sigma(X)) \\
&\geq \sum_\sigma \sum_{k=0}^{n} \frac{\binom{n}{k}}{2^n} f(S_k^\sigma) \frac{1}{n!} \\
&= \sum_{k=0}^{n} \frac{\binom{n}{k}}{n! 2^n} \sum_\sigma f(S_k^\sigma) \\
&= \sum_{k=0}^{n} \frac{\binom{n}{k}}{n! 2^n} k!(n-k)! \sum_{S:|S|=k} f(S) \\
&= \sum_S \frac{f(S)}{2^n} \\
&= \mathbf{E}_S(f(S))
\end{aligned}$$

By $\mathbf{E}_S(f(S))$, we denote the expected function value when the set S is sampled uniformly at random, i.e., each element is included with probability 1/2. [Feige et al., 2011b] shows that $\mathbf{E}_S(f(S)) \geq \frac{1}{4} f(X^*)$. For symmetric submodular functions, the factor is 1/2. □

A key observation which proves the result above is that RP guarantees a solution, which in expectation, is no worse than a random subset (RS). Moreover, Algorithm RA further improves upon RP and hence RS. In practice, moreover, we find that this difference is often significant.

4.4.2 Randomized local search (RLS) based subgradient.

Instead of using a completely random subgradient at every iteration as in RA, we fix the positions of two elements: the permutation must satisfy that $\sigma^t(|X^t| + 1) \in \text{argmax}_j f(j|X^t)$

and $\sigma^t(|X^t|-1) \in \text{argmin}_j\, f(j|X^t \backslash j)$. The remaining positions are assigned randomly. An η-approximate version of MMax with such subgradients returns an η-approximate local maximum that achieves an improved approximation factor of $1/3 - \eta$ in $O(\frac{n^2 \log n}{\eta})$ iterations.

Lemma 4.4. *Algorithm RLS returns an η-approximate local maximum X that satisfies $\max\{f(X), f(V \backslash X)\} \geq (\frac{1}{3} - \eta) f(X^*)$ in $O(\frac{n^2 \log n}{\eta})$ iterations. For a symmetric function, that factor is $1/2 - \eta$.*

Proof. At termination ($t = T$), it holds that $\max_j f(j|X^T) \leq 0$ and $\min_j f(j|X^T \setminus j) \geq 0$; this implies that the set X^t is local optimum.

To show local optimality, recall that the subgradient $h^{\sigma^T}_{X^T}$ satisfies $h^{\sigma^T}_{X^T}(X^T) = f(X^T)$, and $h^{\sigma^T}_{X^T}(Y) \geq h^{\sigma^T}_{X^T}(X^T)$ for all $Y \subseteq V$. Therefore, it must hold that $max_{j \notin X^T} f(j|X^T) = \max_{j \notin X^T} h^{\sigma^T}_{X^T}(j) \leq 0$, and $\min_{j \in X^T} f(j|X^T \backslash j) = h^{\sigma^T}_{X^T}(j) \geq 0$, which implies that the set X^T is a local maximum.

We now use a result by [Feige et al., 2011b] showing that if a set X is a local optimum, then $f(X) \geq \frac{1}{3} f(X^*)$ if f is a general non-negative submodular set function and $f(X) \geq \frac{1}{2} f(X^*)$ if f is a symmetric submodular function. If the set is an η-approximate local optimum, we obtain a $\frac{1}{3} - \eta$ approximation (and $1/2 - \eta$ for symmetric submodular) [Feige et al., 2011b]. A complexity analysis similar to Lemma 4.2 reveals that the worst case complexity of this algorithm is $O(\frac{n^2 \log n}{\eta})$. □

Note that even finding an exact local maximum is hard for submodular functions [Feige et al., 2011b], and therefore it is necessary to resort to an η-approximate version, which converges to an η-approximate local maximum.

4.4.3 Deterministic local search (DLS) based subgradient.

A completely deterministic variant of RLS defines the permutation by an entirely greedy ordering. We define permutation σ^t used in iteration t via the chain $\emptyset = S^{\sigma^t}_0 \subset S^{\sigma^t}_1 \subset \ldots \subset S^{\sigma^t}_n$ it will generate. The initial permutation is $\sigma^0(j) = \text{argmax}_{k \notin S^{\sigma^0}_{j-1}} f(k|S^{\sigma^0}_{j-1})$ for $j = 1, 2, \ldots$.

In subsequent iterations t, the permutation σ^t is

$$\sigma^t(j) = \begin{cases} \sigma^{t-1}(j) & \text{if } t \text{ even}, j \in X^{t-1} \\ \text{argmax}_k \, f(k|S^{\sigma^t}_{j-1}) & \text{if } t \text{ even}, j \notin X^{t-1} \\ \text{argmin}_k \, f(k|S^{\sigma^t}_{j+1} \backslash k) & \text{if } t \text{ odd}, j \in X^{t-1} \\ \sigma^{t-1}(j) & \text{if } t \text{ odd}, j \notin X^{t-1}. \end{cases}$$

Algorithm 3 Deterministic Local Search Algorithm for Unconstrained Submodular Maximization

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

  $Y_1 = \text{argmax}_{v \in V} \, f(v)$;

  % Greedily add elements

  while $f(Y_{n+1}) \geq (1 + \eta) f(Y_n)$ **do**

    $y = \text{argmax}_{v \in V \backslash Y_n} \, f(v|Y_n)$;

    $Y_{n+1} = Y_n \cup y$;

    $n \leftarrow n + 1$;

  % Greedily remove elements

  while $f(Y_{n+1}) \geq (1 + \eta) f(Y_n)$ **do**

    $y = \text{argmax}_{v \in Y_n} \, f(v|Y_n \backslash v)$;

    $Y_{n+1} = Y_n \backslash y$;

    $n \leftarrow n + 1$;

until *convergence* ($Y_n = Y_{n-1}$);

return the better amongst Y_n and $V \backslash Y_n$.

This schedule is very similar to the deterministic local search (DLS) algorithm by [Feige et al., 2011b]. This scheme is shown in Algorithm 3. Moreover, similar to RLS, we can show that choosing the better amongst X and $V \backslash X$ (where X is the solution obtained on

convergence) achieves a $1/3 - \eta$ approximation (and $1/2 - \eta$ for symmetric). This algorithm essentially iterates over two greedy algorithms, viz. greedily adding elements and greedily removing elements until convergence. Moreover, note that each inner greedy iteration (which iterates between adding and removing elements) corresponds to one subgradient iteration of MMax.

4.4.4 Greedy Subgradient

A very common algorithm used often for submodular maximization is the greedy algorithm [Nemhauser et al., 1978].

Algorithm 4 Greedy Algorithm for Unconstrained Submodular Maximization

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

  $y = \text{argmax}_{v \in V \setminus Y_n} f(v|Y_n)$;

  $Y_{n+1} = Y_n \cup y$;

  $n \leftarrow n + 1$;

until $f(Y_n) \leq f(Y_{n-1})$;

return Y_{n-1}.

While this algorithm has strong guarantees for constrained monotone submodular functions (we shall revisit this later), it retains very weak guarantees for non-monotone submodular functions.

Lemma 4.5. *The greedy algorithm admits a worst case factor of $\theta(1/n)$ for non-monotone submodular maximization, which is tight.*

Proof. Assume $\max_{j \in V} f(j) > 0$, otherwise, the greedy algorithm returns the empty set and achieves the optimum. Let S^G denote the greedy solution, then we have $f(S^G) \geq \max_{j \in V} f(j)$.

Then, for any $S \subseteq V$, the following chain of inequalities hold

$$f(S) \leq \sum_{j \in S} f(j) \leq |S| \max_{j \in S} f(j) \leq |S| \max_{j \in V} f(j) \leq |S| f(S^G).$$

Therefore, $f(S^G) \geq 1/|S| f(S)$, for any $S \subseteq V$, and hence $f(S^G) \geq 1/n f(OPT)$.

To show the tightness, consider the following function. Define f such that, $f(S) = 1$, if $x \in S$ for some item x, and $f(S) = |S|$ otherwise. The submodularity of the function f can be easily checked by the definition. The greedy algorithm, in the worst case, returns item x, while the optimum is achieved by $V \setminus x$. In this case, the approximation factor is $1/(n-1)$. □

It is interesting to note however, that though the greedy algorithm can be really bad for non-monotone submodular functions, the Deterministic Local Search Algorithm above which uses the greedy algorithm multiple times (potentially also improving upon it) admits constant factor guarantees. Moreover, one could also visualize an algorithm which rather than starting at the empty set and adding elements, starts with the ground set and removes elements. Though this technique is equally reasonable, it too can be $\theta(1/n)$. In the next two sections, we shall see bidirectional variants of the greedy algorithm, proposed by [Buchbinder et al., 2012] provide optimal constant factor approximation guarantees. In some sense, they exploit this drawback of the unidirectional nature of the greedy algorithm and suggest a bidirectional greedy approach.

4.4.5 Bi-directional greedy (BG) subgradient.

The procedures above indicate that greedy and local search algorithms implicitly define specific chains and thereby subgradients. Likewise, the deterministic bi-directional greedy algorithm by [Buchbinder et al., 2012] induces a distinct permutation of the ground set (the algorithm is shown in Algorithm 5 for completeness). This algorithm effectively overcomes the difficulties of the simple greedy algorithm, and achieves an approximation factor of 1/3. This improves upon the local search techniques by removing η, and unlike the local search, the 1/3 approximation holds in just $O(n)$ time.

This algorithm is equivalent to MMax with a specific induced subgradient. In particular, the entire scheme of Algorithm 5 can be seen as a single subgradient in MMax. Given an initial ordering τ, the bi-directional greedy algorithm (Algorithm 5) by [Buchbinder et al., 2012] generates a chain of sets. Let σ^τ denote the permutation defined by this chain, obtainable by mimicking the algorithm.

Algorithm 5 Bidirectional Greedy Algorithm of [Buchbinder et al., 2012] for Unconstrained Submodular Maximization

Start with $A_0 = \emptyset, B_0 = V$, and an initial ordering of $V : \tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$

for $i = 1$ *to* n **do**

  $a_i \leftarrow f(A_{i-1} \cup \tau_i) - f(A_{i-1})$

  $b_i \leftarrow f(B_{i-1} \backslash \tau_i) - f(B_{i-1})$

  if $a_i \geq b_i$ **then**

    $A_i = A_{i-1} \cup \tau_i, B_i = B_{i-1}$

  else

    $A_i = A_{i-1}, B_i = B_{i-1} \backslash \tau_i$

return A_n (or B_n)

Lemma 4.6. *The set X^1 obtained by MMax with the subgradient σ^τ equivalent to BG satisfies that $f(X) \geq \frac{1}{3} f(X^*)$.*

Proof. We run MMax with the corresponding subgradient σ^τ. By construction, the set S^τ returned by the bi-directional greedy algorithm is contained in the chain. Therefore, it holds that

$$\begin{aligned} f(X^1) &\geq \max_{X \subseteq V} h_\emptyset^{\sigma^\tau}(X) \\ &\geq \max_k f(S_k^{\sigma^\tau}) \\ &\geq f(S^\tau) \\ &\geq \frac{1}{3} f(X^*). \end{aligned}$$

The first inequality follows since the subgradient is tight for all sets in the chain. For the second inequality, we used that S^τ belongs to the chain, and hence $S^\tau = S_j^{\sigma^\tau}$ for some j. The last inequality follows from the approximation factor satisfied by S^τ [Buchbinder et al., 2012]. We can continue the algorithm, using any one of the adaptive schedules above to get a locally optimal solution. This can only improve the solution. □

It is worth pointing out here, that the ordering σ^τ is obtained only by running Algorithm 5. In particular, having obtained the sets X_n, Y_n (note that $X_n \cup Y_n = V$, we can define a ordering via the elements in X_n and Y_n. The subgradient is formed via this natural ordering.

4.4.6 Randomized bi-directional greedy (RBG) subgradient.

Like its deterministic variant, the randomized bi-directional greedy algorithm (shown in Algorithm 6 for completeness) by [Buchbinder et al., 2012] can be shown to run MMax with a specific subgradient. Starting from $\emptyset$ and V, it implicitly defines a random chain of subsets and thereby (random) subgradients. A simple analysis shows that this subgradient leads to the best possible approximation factor of 1/2 in expectation.

Algorithm 6 Randomized Bidirectional Greedy Algorithm of [Buchbinder et al., 2012] for Unconstrained Submodular Maximization

Start with $A_0 = \emptyset, B_0 = V$, and an initial ordering of $V : \tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$

for $i = 1$ *to* n **do**

- $a_i \leftarrow f(A_{i-1} \cup \tau_i) - f(A_{i-1})$
- $b_i \leftarrow f(B_{i-1} \backslash \tau_i) - f(B_{i-1})$
- $a_i' \leftarrow \max\{a_i, 0\}, b_i' \leftarrow \max\{b_i, 0\}$
- **with probability** $\frac{a_i'}{a_i' + b_i'}$: $A_i = A_{i-1} \cup \tau_i, B_i = B_{i-1}$
- **else with probability** $\frac{b_i'}{a_i' + b_i'}$: $A_i = A_{i-1}, B_i = B_{i-1} \backslash \tau_i$

return A_n (or B_n)

Like its deterministic counterpart, the Randomized bi-directional Greedy algorithm (RBG)

by [Buchbinder et al., 2012] induces a (random) permutation σ^τ based on an initial ordering τ.

Lemma 4.7. *The set X^1 obtained by running MMax using the randomized bi-directional greedy subgradient σ^τ satisfies $\mathbf{E}(f(X^1)) \geq \frac{1}{2} f(X^*)$, where the expectation is taken over the randomness in σ^τ.*

Proof. The permutation σ^τ is obtained by a randomized algorithm, but once σ^τ is fixed, the remainder of MMax is deterministic. By an argumentation similar to that in the proof of Lemma 4.6, it holds that

$$\begin{aligned}
\mathbf{E}(f(X)) &\geq \mathbf{E}(\max_X h_\emptyset^{\sigma^\tau}(X)) \\
&\geq \mathbf{E}(\max_k f(S_k^{\sigma^\tau})) \\
&\geq \mathbf{E}(f(S^{\sigma^\tau})) \\
&\geq \frac{1}{2} f(X^*)
\end{aligned}$$

The last inequality follows from a result in [Buchbinder et al., 2012]. □

Similar to the deterministic case, the (random) subgradient here is initialized only by running the randomized algorithm.

4.5 Constrained Submodular Maximization

In this final section, we analyze subgradients for maximization subject to the constraint $X \in \mathcal{C}$. We consider both monotone and non-monotone submodular functions. Table 4.1 lists the results obtained with specific choices of subgradients under different settings. Some of these results are also plotted in Figure 4.1.

4.5.1 Random Subgradient (RA/RP)

Similar to the unconstrained setting, the simplest subgradient is to just choose a random one at every iteration: we randomly pick a permutation σ that defines a subgradient at X^{t-1}, i.e.,

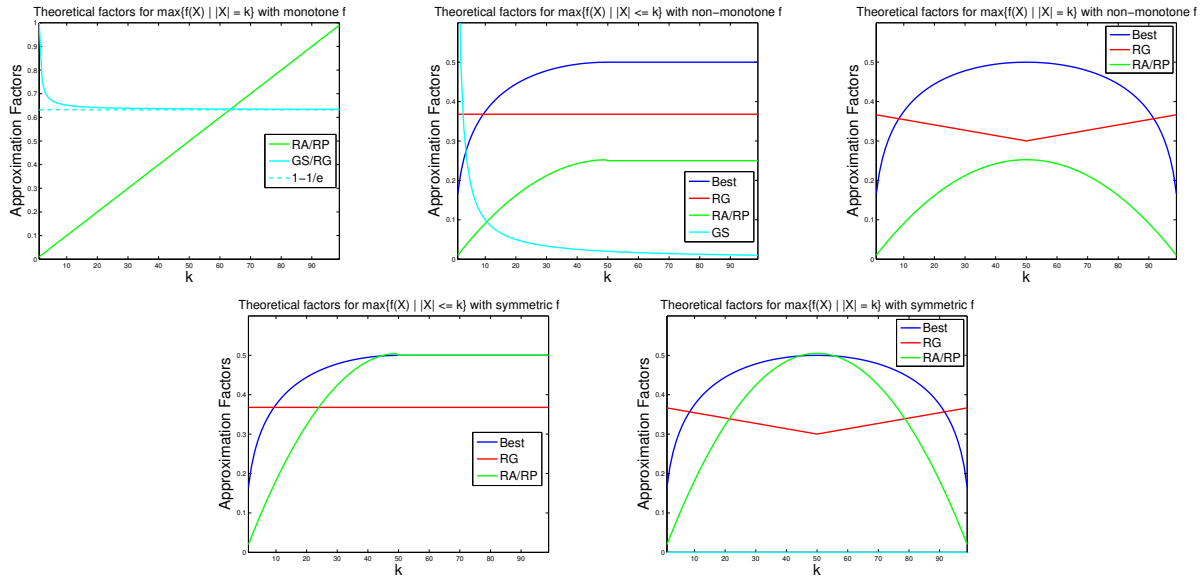

Figure 4.1: Theoretical Approximation guarantees for cardinality constrained submodular maximization under various settings. The top left figure corresponds to monotone submodular functions, the top middle figure corresponds to the problem $\max\{f(X)||X| \leq k\}$, while the top right figure corresponds to $\max\{f(X)||X| = k\}$, both with non-monotone submodular functions. The bottom left figure corresponds to $\max\{f(X)||X| \leq k\}$, while the bottom right refers to $\max\{f(X)||X| = k\}$, both with symmetric submodular functions. For the different subgradient schemes in the legends, refer to the text. The method *Best* refers to the best algorithm for the corresponding problem, based on a continuous double greedy algorithm [Buchbinder et al., 2014]

X^{t-1} is assigned to the first $|X^{t-1}|$ positions. At $X^0 = \emptyset$, this can be any permutation. We iteratively optimize the corresponding modular upper bound, subject to the constraints.This simple technique achieves bounded approximation factors.

Lemma 4.8. *Given a monotone submodular function and a parameter k, running MMax with a random permutation obtains a worst case approximation factor of k/n for the problem $\max\{f(X)||X| \leq k\}$. With a non-monotone submodular function, running MMax with a random permutation provides an approximation factor of $\frac{k(n-k)}{n(n-1)}$ for the problem $\max\{f(X)||X| = k\}$, and a factor of $\frac{k(n-k)}{n(n-1)}$, for $k \leq n/2$ and $1/4 + o(1)$ for $k \geq n/2$ for the problem $\max\{f(X)||X| \leq k\}$. When f is symmetric, the factors can be improved to $\frac{2k(n-k)}{n(n-1)}$.*

Proof. First we consider the case of $\max\{f(X)||X| = k\}$. Each permutation has the same probability $1/n!$ of being chosen. Therefore, it holds that

$$\begin{aligned}\mathbf{E}(f(X^1)) &= \mathbf{E}_\sigma(\max_{X\subseteq V} h^\sigma_\emptyset(X)) \\ &= \frac{1}{n!}\sum_\sigma \max_{X\subseteq V} h^\sigma_\emptyset(X)\end{aligned}$$

Let $\emptyset \subseteq S^\sigma_1 \subseteq S^\sigma_2 \cdots S^\sigma_n = V$ be the chain corresponding to a given permutation σ. We then have that, $\max_{X\subseteq V:|X|=k} h^\sigma_\emptyset(X) \geq f(S^\sigma_k)$. Hence, we have,

$$\begin{aligned}\mathbf{E}(f(X^1)) &\geq \mathbf{E}_\sigma(\max_{X\subseteq V} h^\sigma_\emptyset(X)) \\ &\geq \frac{1}{n!}\sum_\sigma f(S^\sigma_k) \\ &= \frac{(n-k)!k!}{n!}\sum_{S:|S|=k} f(S) \\ &= \sum_{S:|S|=k} \frac{f(S)}{\binom{n}{k}} \\ &= \mathbf{E}_{S\in\binom{[n]}{k}}(f(S))\end{aligned}$$

By $\mathbf{E}_{S\in\binom{[n]}{k}}(f(S))$, we denote the expected function value when the set S is sampled uniformly at random amongst sets of size k. [Filmus, 2013] shows that choosing a random set of size k

obtains a guarantee of $\frac{k}{n}$ and $\frac{n(n-k)}{n(n-1)}$ for the problem $\max\{f(X)||X| = k\}$ with monotone and non-monotone submodular functions respectively.

We now handle the problem of $\max\{f(X)||X| \leq k\}$. When f is monotone, this problem is equivalent to $\max\{f(X)||X| = k\}$. When f is non-monotone, we can provide an algorithm as follows. If $k \leq n/2$, select a random set of size k, else, select a random set of size $n/2$. This simple technique provides a worst case guarantee of $\frac{k(n-k)}{n(n-1)}$, for $k \leq n/2$ and $1/[4(1-1/n)] = 1/4 + o(1)$ for $k \geq n/2$.

Interestingly, MMax directly and automatically achieves this. If $k \leq n/2$, we can use exactly the above proof to obtain the guarantee (since maximizing the random modular lower bound is better than the expected value of a random subset of size k). When $k \geq n/2$, we can slightly modify the above proof as follows.

$$\begin{aligned}
\mathbf{E}(f(X^1)) &\geq \mathbf{E}_\sigma(\max_{X \subseteq V} h^\sigma_\emptyset(X)) \\
&= \frac{1}{n!} f(S^\sigma_{n/2}) \\
&= \frac{1}{n!} \sum_\sigma f(S^\sigma_{n/2}) \\
&= \mathbf{E}_{S \in \binom{[n]}{n/2}}(f(S))
\end{aligned}$$

For symmetric submodular functions, the factor is $\frac{2n(n-k)}{n(n-1)}$, which follows by combining the above proof technique with the guarantee of [Filmus, 2013] for symmetric submodular functions. In particular, the problem $\max\{f(X)||X| = k\}$ achieves a factor of $\frac{2n(n-k)}{n(n-1)}$, while the problem $\max\{f(X)||X| \leq k\}$ obtains a factor of $\frac{2n(n-k)}{n(n-1)}$, for $k \leq n/2$ and $1/2 + o(1)$ for $k \geq n/2$. □

Note that the guarantee of the random subgradient approach holds just after the first iteration, and continuing the algorithm until convergence will only improve the objective. We also saw how many different settings can very easily be integrated into a unified algorithm of MMax, which naturally takes care of the constraints given. This is in contrast to the random subset algorithm, which needs to adapt based on the value of k, for example, in the setting of $\max\{f(X)||X| \leq k\}$.

Also note, that the problem $\max\{f(X)||X| \leq k\}$ is equivalent to unconstrained submodular maximization when $k = n$. Correspondingly, MMax yields a slightly improved approximation factor of $1/[4(1 - 1/n)]$ (thanks to the Lemma above), even for the unconstrained submodular maximization with a random subgradient. This is a slight improvement over Lemma 4.3.

4.5.2 Greedy Subgradient (GS)

We investigate the role of a greedy subgradient, which yields a greedy algorithm for submodular maximization. In this section, we shall study the role of many of the constructs defined in section 1.5 in the approximation guarantees.

Monotone submodular maximization: An important subgradient results from the greedy permutation σ^g, defined as

$$\sigma^g(i) \in \operatorname*{argmax}_{j \notin S_{i-1}^{\sigma^g} \text{ and } S_{i-1}^{\sigma^g} \cup \{j\} \in \mathcal{C}} f(j|S_{i-1}^{\sigma^g}). \tag{4.7}$$

This definition might be partial; we arrange any remaining elements arbitrarily. When using the corresponding subgradient h^{σ^g}, we recover a number of approximation results already after one iteration:

Algorithm 7 Greedy Algorithm for $\max_{X \in \mathcal{C}} f(X)$

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

  $y = \operatorname{argmax}_{v \in V \setminus Y_n} f(v|Y_n)$;

  $Y_{n+1} = Y_n \cup y$;

  $n \leftarrow n + 1$;

until $Y_n \notin \mathcal{C}$;

return Y_{n-1}.

Lemma 4.9. *Using h^{σ^g} in iteration 1 of MMax yields the following approximation bounds for X^1:*

- $\frac{1}{\kappa_f}(1-(1-1/k)^{k\kappa_f}) \geq \frac{1}{\kappa_f}(1-e^{-\kappa_f})$, *if* $\mathcal{C} = \{X \subseteq V : |X| \leq k\}$
- $\frac{1}{p+\kappa_f}$, *for the intersection* $\mathcal{C} = \cap_{i=1}^{p} \mathcal{I}_i$ *of* p *matroids*
- $\frac{1}{\kappa_f}(1-(\frac{K-\kappa_f}{K})^k)$, *for any down-monotone constraint* $\mathcal{C}$, *where* K *and* k *are the maximum and minimum cardinality of the maximal feasible sets in* $\mathcal{C}$.

We remark that the above choice of the subgradient essentially ensures that MMax mimics the Greedy algorithm (Algorithm 7) in its first iteration. This result provides guarantees for a large class of constraints (which include cardinality, matroids etc.)

Proof. We prove the first result for cardinality constraints. The proofs for the matroid and general down-monotone constraints are analogous. By the construction of σ^g, the set $S_k^{\sigma^g}$ is exactly the set returned by the greedy algorithm. This implies that

$$\begin{aligned} f(X^1) &\geq \operatorname*{argmax}_{X:|X|\leq k} h_{\emptyset}^{\sigma^g}(X) \\ &\geq h_{\emptyset}^{\sigma^g}(S_k^{\sigma^g}) \\ &= f(S_k^{\sigma^g}) \\ &\geq \frac{(1-e^{-\kappa_f})}{\kappa_f} f(X^*). \end{aligned}$$

The last inequality follows from [Nemhauser et al., 1978, Conforti and Cornuejols, 1984]. □

Remark 4.1. *Note the role of curvature in the guarantees above. [Nemhauser et al., 1978] showed that the greedy algorithm obtains a worst case approximation factor of $1-1/e$ for cardinality constraints, and $1/(p+1)$ for p matroid constraints (in particular, this gives a $1/2$ approximation for submodular maximization subject to a single matroid constraint). [Conforti and Cornuejols, 1984] improved these factors to $\frac{1}{\kappa_f}(1-e^{-\kappa_f})$ and $\frac{1}{p+\kappa_f}$ respectively. Also note that the greedy subgradient is tight for the cardinality constrained case. In the case of matroidal constraints however, the greedy algorithm is sub-optimal [Calinescu et al., 2007].*

A very similar construction of a greedy permutation provides bounds for budget constraints, i.e., $c(S) \triangleq \sum_{i \in S} c(i) \leq B$ for some given nonnegative costs c. The problem of interest here is,

$$\max\{f(S)|c(S) \leq B\} \tag{4.8}$$

Algorithm 8 Greedy Algorithm for $\max_{c(S)\leq B} f(X)$

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

$y = \text{argmax}_{v \in V \setminus Y_n} \frac{f(v|Y_n)}{c(v)}$;

$Y_{n+1} = Y_n \cup y$;

$n \leftarrow n + 1$;

until $c(Y_n) \geq B$;

return Y_{n-1}.

In particular, define a permutation as:

$$\sigma^g(i) \in \underset{j \notin S^{\sigma^g}_{i-1}, c(S^{\sigma^g}_{i-1} \cup \{j\}) \leq B}{\text{argmax}} \frac{f(j|S^{\sigma^g}_{i-1})}{c(j)}. \tag{4.9}$$

The following result then follows from [Lin and Bilmes, 2010, Krause and Guestrin, 2005a, Sviridenko, 2004].

Lemma 4.10. *Using σ^g in MMax under the budget constraints yields:*

$$\max\{\max_{i:c(i)\leq B} f(i), f(X^1)\} \geq \frac{1}{2}(1 - 1/e)f(X^*). \tag{4.10}$$

Let σ^{ijk} be a permutation with i, j, k in the first three positions, and the remaining arrangement greedy. Running $O(n^3)$ restarts of MM yields sets X_{ijk} (after one iteration) with

$$\max_{i,j,k \in V} f(X_{ijk}) \geq (1 - 1/e)f(X^*). \tag{4.11}$$

The proof is analogous to that of Lemma 4.9. This Lemma essentially mimics two variants of the greedy algorithm. One is a faster variant due to [Lin and Bilmes, 2010, Krause and Guestrin, 2005a] (Algorithm 8). The second variant is a more computationally complex algorithm, and enumerates all subsets of size 3 and augments them with the cost sensitive greedy procedure shown above. This algorithm was shown to give a $1 - 1/e$ approximation [Sviridenko, 2004].

Fast/Lazy variants of Greedy Algorithm: Finally we note that the greedy algorithm for monotone submodular functions can be accelerated via the accelerated greedy algorithm [Minoux, 1978]. This algorithm effectively avoids the $O(n^2)$ computation using submodularity. In practice, moreover, it performs much faster [Leskovec et al., 2007]. Since we need to compute the greedy ordering to run MMax, the accelerated greedy procedure serves as a much more effective means. [Badanidiyuru and Vondrák, 2014] recently propose a fast though approximate lazy scheme for obtaining an ordering. This scheme also obtains an ordering like the lazy greedy algorithm. Unlike the lazy greedy, however, it is only approximate, and obtains a factor of $1 - 1/e - \epsilon$. It does so in $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$, which is asymptotically faster than $O(n^2)$ of the greedy algorithm. Finally, a multistage version of a greedy algorithm was proposed by [Wei et al., 2014a], where they use proxy functions to define the greedy ordering. Since all these algorithms are variants of the greedy algorithm, and essentially induce an ordering, they can all be seen as inducing an ordering with MMax.

Non-monotone submodular maximization: The greedy algorithm provides constant factor (and sometimes optimal) approximation factors for monotone submodular functions. In the context of non-monotone submodular functions however, the greedy algorithm can perform really badly.

Lemma 4.11. *The greedy algorithm obtains a worst case approximation guarantee of $\theta(1/k)$, which is tight, for the problem $\max\{f(X)||X| \leq k\}$ when the submodular function f is non-monotone.*

Note that this greedy algorithm is slightly different from Algorithm 7. Along with checking for the cardinality constraint being satisfied, we also check that best gain at any iteration is non-negative (since the function is no longer monotone). Hence this approach works only for the problem $\max\{f(X)||X| \leq k\}$ and not the cardinality equality setting $\max\{f(X)||X| = k\}$. The proof of Lemma 4.11 is similar to that of Lemma 4.5.

Fortunately, if the function is approximately monotone till sets of size $2k$ (i.e $\nu_f^{k,2k} \approx 1$), we can retain guarantees close to $1 - 1/e$.

Lemma 4.12. *Given an approximately monotone submodular function f, the greedy algorithm obtains a worst case factor of $\frac{1}{\nu_f^{k,2k}}(1 - 1/e)$ for the problem $\max\{f(X)||X| \leq k\}$.*

Proof. This result follows directly from the proof of [Nemhauser et al., 1978]. Let S^* be the optimal solution and let S^G be the greedy solution. Using the proof technique of [Nemhauser et al., 1978], one can show that $f(S^G) \geq (1 - 1/e)f(S^* \cup S^G)$. This part just requires submodularity of f and the greedy algorithm. Through the notion of approximate monotonicity, we know that $f(S^* \cup S^G) \geq \frac{f(S^*)}{\nu_f^{k,2k}}$ (by definition of $\nu_f^{k,2k}$). The result then directly follows from these two observations. □

An alternate way of quantifying approximate monotonicity is by looking at the gains of sets till size l. In particular, define a parameter ϵ_f^l such that,

$$\epsilon_f^l = -\min_{T \subseteq V, |T| \leq l} f(j|T) \tag{4.12}$$

This quantity was defined by [Krause et al., 2008b]. If f is monotone, $\epsilon_f^l \leq 0, \forall l$.

Note that ϵ_f^l also provides a notion of approximation, and in particular, an additive approximation factor rather than a multiplicative one.

Lemma 4.13. *The greedy algorithm admits an additive approximation factor of satisfying:*

$$f(S^G) \geq (1 - 1/e)f(S^*) - k\epsilon_f^{2k} \tag{4.13}$$

Proof. This Lemma was also shown in [Krause et al., 2008b]. For completeness, we have it here. It is easy to see that,

$$f(T) \geq f(S) + (k-l)f(j|T) \geq f(S) - (l-k)\epsilon_f^l,$$

for any sets $T \subseteq S$ satisfying $|T| \leq k$ and $|S| = l$. This follows directly from submodularity, and the definition of ϵ_f^l. Hence,

$$f(S^* \cup S^G) \geq f(S^*) - k\epsilon_f^{2k}$$

Following the proof of Lemma 4.12, we have

$$f(S^G) \geq (1-1/e)f(S^* \cup S^G) \geq (1-1/e)f(S^*) - k\epsilon_f^{2k}. \tag{4.14}$$

□

A number of submodular functions, though non-monotone still are *close* to monotone, and the above result holds for these functions. One example of this is the symmetric mutual information $I(X_A; X_{V \setminus A}$, which has been used effectively in sensor placement problems [Krause et al., 2008b]. While this function is non-monotone, it is approximately monotone in the above sense, and obtains approximation guarantees of the form $(1-\epsilon)(1-1/e)$ for a reasonable choice of parameters, and small ϵ. Another example, is a function of the form $f(X) = \sum_{i \in V} \sum_{j \in X} s_{ij} - \lambda \sum_{i,j \in X} s_{ij}$. This function has effectively been used in summarization and subset selection problem for choosing diverse sets of objects [Lin and Bilmes, 2010, 2009]. This function also is approximately monotone under reasonable assumptions [Lin and Bilmes, 2010], and satisfies $\nu_f^{2k,k} = 1$ with high probability.

Approximately submodular functions: [Das and Kempe, 2011] introduced the notion of approximately submodular to improve the bounds of the greedy algorithm for approximately submodular functions.

Lemma 4.14. *[Das and Kempe, 2011] Given a approximately submodular function f, the greedy algorithm obtains a set S^G with worst case guarantee $1 - e^{-\gamma_f^{S^G;k}}$.*

For a submodular function, $\gamma_f^{S^G;k} = 1$, and in general, $\gamma_f^{S^G;k} \leq 1$. We can in fact, seamlessly combine Lemmata 4.14 and 4.12 to obtain the following guarantee for a approximately monotone and approximately submodular function.

Lemma 4.15. *Given a approximately submodular and approximately monotone function f, the greedy algorithm obtains a set S^G with worst case guarantee $\frac{1-e^{-\gamma_f^{S^G;k}}}{\nu_f^{2k,k}}$.*

The proof of this Lemma, follows again by adapting the notion of approximate monotonicity in the proof of [Das and Kempe, 2011].

4.5.3 Random Greedy (RG) Subgradient

The greedy algorithm fails to obtain constant factor approximation guarantees for general non-monotone submodular functions. A simple randomized variant of the greedy algorithm, however, ensures constant factor guarantees [Buchbinder et al., 2014].

Algorithm 9 Random Greedy Algorithm under Cardinality Constraints

Start with $Y_0 = \emptyset$

for $i = 1$ *to* k **do**

  Let $M_i = \text{argmax}_{X \subseteq V \setminus Y_{i-1}, |X|=k} \sum_{v \in X} f(v|Y_{i-1})$;

  Choose y as a uniformly random element in M_i;

  $Y_i = Y_{i-1} \cup y$;

return Y_k.

It is easy to see that this random greedy algorithm (Algorithm 9) also implicitly defines a subgradient, which when used within MMax, essentially mimics the algorithm.

Lemma 4.16. *Using the random greedy subgradient within MMax, provides a worst case approximation factor of $1 - 1/e$ for monotone submodular maximization under cardinality constraints, a factor of $1/e$ with non-monotone submodular functions for the problem $\max\{f(X)||X| \leq k\}$ and $\frac{1-k/en}{e} - \epsilon$ for the problem $\max\{f(X)||X| = k\}$ with non-monotone submodular functions.*

Note that Algorithm 9 requires k to be such that $n \geq 2k$. As shown in [Buchbinder et al., 2014], this assumption can be made w.l.o.g through some simple reductions. Also notice the similarity between the greedy algorithm and the random greedy algorithm. While the greedy algorithm picks the best element at every iteration, the random greedy randomly picks one of the top k elements at every iteration (both with respect to the current marginal gains). The proof of Lemma 4.16 is similar in lines to the previous results.

4.6 Generality of MMax and Choosing Optimal Subgradients

We list a number of algorithms above, which *naturally* can be seen as instances of MMax. In particular, these algorithms naturally define permutations, or sequences of permutations corresponding to MMax. We can only expect this to work with combinatorial algorithms. However, a number of submodular maximization algorithms [Chekuri et al., 2011, Calinescu et al., 2011, Buchbinder et al., 2014, Feldman et al., 2011, Calinescu et al., 2007] rely on continuous relaxations and rounding. These algorithms are not combinatorial, but often provide tighter guarantees than their discrete counterparts. For example, for the problem of monotone submodular maximization subject to a single matroid constraints, a continuous greedy algorithm [Calinescu et al., 2011, 2007] obtains a $1 - 1/e$ approximation while the discrete greedy algorithm provides only a $1/2$ approximation. In fact, [Vondrák, 2010] improves this to a curvature dependent factor of $\frac{1}{\bar{\kappa_f}(S^*)}(1 - e^{-\bar{\kappa_f}(S^*)}) \geq \frac{1}{\kappa_f}(1 - e^{-\kappa_f})$. Similarly for cardinality constrained non-monotone submodular maximization, a continuous double greedy algorithm [Buchbinder et al., 2014] provides tighter guarantees than the random greedy algorithm (see Figure 4.1, where this method is referred to as *Best*). These continuous greedy algorithms do not yield natural subgradients or orderings. Certain combinatorial algorithms also do not readily fit into our algorithmic framework. For example, [Lee et al., 2009a,b] propose local search based techniques to obtain constant factor approximations for non-monotone submodular maximization under knapsack and matroid constraints. Unfortunately, these algorithms require swap operations along with inserting and deleting elements. We do not currently know how to phrase these swap operations via our framework.

While many of these algorithms do not explicitly define a sequence of orderings and permutations, we show that any polynomial time approximation algorithm for unconstrained or constrained variants of submodular optimization can be ultimately seen as an instance of our algorithm, via polynomial-time computable subgradients. In particular we show that there exists polynomial time computable subgradients, which when instantiated within MMax, provides a solution comparable to the given approximation algorithm. This result moreover, holds for both constrained and unconstrained versions.

Theorem 4.1. *For any polynomial-time unconstrained submodular maximization algorithm that achieves an approximation factor α, there exists a schedule of subgradients (obtainable in polynomial time) that, if used within MMax, leads to a solution with the same approximation factor α.*

The proof relies on the following observation.

Lemma 4.17. *Any submodular function f satisfies*

$$\max_{X \in \mathcal{C}} f(X) = \max_{X \in \mathcal{C}, h \in \mathcal{P}_f} h(X) = \max_{X \in \mathcal{C}, \sigma \in \sigma} h^{\sigma}_{\emptyset}(X). \tag{4.15}$$

Lemma 4.17 implies that there exists a permutation (and equivalent subgradient) with which MMax finds the optimal solution in the first iteration. Known hardness results [Feige, 1998] imply that this permutation may not be obtainable in polynomial time.

Proof. (Lemma 4.17) The first equality in Lemma 4.17 follows from the fact that any submodular function f can be written as

$$f(X) = \max_{h \in P_f} h(X). \tag{4.16}$$

For the second equality, we use the fact that a linear program over a polytope has a solution at one of the extreme points of the corresponding polytope. □

We can now prove Theorem 4.1

Proof. (Thm. 4.1) Let Y be the set returned by the approximation algorithm; this set is polynomial-time computable by definition. Let τ be an arbitrary permutation that places the elements in Y in the first $|Y|$ positions. The subgradient h^τ defined by τ is a subgradient both for $\emptyset$ and for Y. Therefore, using $X^0 = \emptyset$ and h^τ in the first iteration, we obtain a set X^1 with

$$f(X^1) \geq h^\tau_\emptyset(X^1) \geq h^\tau_\emptyset(Y) = f(Y) \geq \alpha f(X^*). \tag{4.17}$$

The equality follows from the fact that Y belongs to the chain of τ. □

While the above theorem shows the optimality of MMax in the unconstrained setting, a similar result holds for the constrained case:

Corollary 4.1. *Let $\mathcal{C}$ be any constraint such that a linear function can be exactly maximized over $\mathcal{C}$. For any polynomial-time algorithm for submodular maximization over $\mathcal{C}$ that achieves an approximation factor α, there exists a schedule of subgradients (obtainable in polynomial time) that, if used within MMax, leads to a solution with the same approximation factor α.*

The proof of Corollary 4.1 follows directly from the Theorem 4.1.

Lastly, we pose the question of selecting the optimal subgradient in each iteration. An optimal subgradient h would lead to a function m_h whose maximization yields the largest improvement. Unfortunately, obtaining such an "optimal" subgradient is impossible:

Theorem 4.2. *The problem of finding the optimal subgradient $\sigma^{OPT} = \text{argmax}_{\sigma, X \subseteq V} h^\sigma_{X^t}(X)$ in Step 4 of Algorithm 2 is NP-hard even when $\mathcal{C} = 2^V$. Given such an oracle, however, MMax using subgradient σ^{OPT} returns a global optimizer.*

Proof. Lemma 4.17 implies that an optimal subgradient at $X^0 = \emptyset$ or $X^0 = V$ is a subgradient at an optimal solution. An argumentation as in Equation (4.17) shows that using this subgradient in MM leads to an optimal solution. Since this would solve submodular maximization (which is NP-hard), it must be NP-hard to find such a subgradient.

To show that this holds for arbitrary X^t (and correspondingly at every iteration), we use that the submodular subdifferential can be expressed as a direct product between a submodular polyhedron and an anti-submodular polyhedron [Fujishige, 2005]. Any problem involving an optimization over the subdifferential, can then be expressed as an optimization over a submodular polyhedron (which is a subdifferential at the empty set) and an anti-submodular polyhedron (which is a subdifferential at V) [Fujishige, 2005]. Correspondingly, Equation (4.15) can be expressed as the sum of two submodular maximization problems. □

4.7 Experiments

We now empirically compare variants of MMax with different subgradients for various settings of non-monotone and monotone submodular functions and constraints.

4.7.1 Unconstrained Submodular Maximization

First consider unconstrained submodular maximization. As a test function, we use the objective of [Lin and Bilmes, 2009], $f(X) = \sum_{i \in V} \sum_{j \in X} s_{ij} - \lambda \sum_{i,j \in X} s_{ij}$, where λ is a redundancy parameter. This non-monotone function was used to find the most diverse yet relevant subset of objects in a large corpus. We use the objective with both synthetic and real data. We generate 10 instances of random similarity matrices $\{s_{ij}\}_{ij}$ and vary λ from 0.5 to 1. Our real-world data is the Speech Training data subset selection problem [Lin and Bilmes, 2009] on the TIMIT corpus [Garofolo et al., 1993], using the string kernel metric [Rousu and Shawe-Taylor, 2006] for similarity. We use $20 \leq n \leq 30$ so that the exact solution can still be computed with the algorithm of [Goldengorin et al., 1999].

We compare the following choices of subgradients: (a) Deterministic Local search (DLS), (b) Bidirectional Greedy (BG), (c) Randomized Bidirectional Greedy (RBG), (d) Randomized local search (RLS), (e) Random Adaptive (RA) and (f) Random subgradient (RP), and (g) a baseline Random Set algorithm (RS) that picks a set uniformly at random. RS achieves a 1/4 approximation in expectation [Feige et al., 2011b]. For random algorithms, we select the best solution out of 5 repetitions. Figure 4.2 shows that DLS, BG, RBG and RLS dominate.

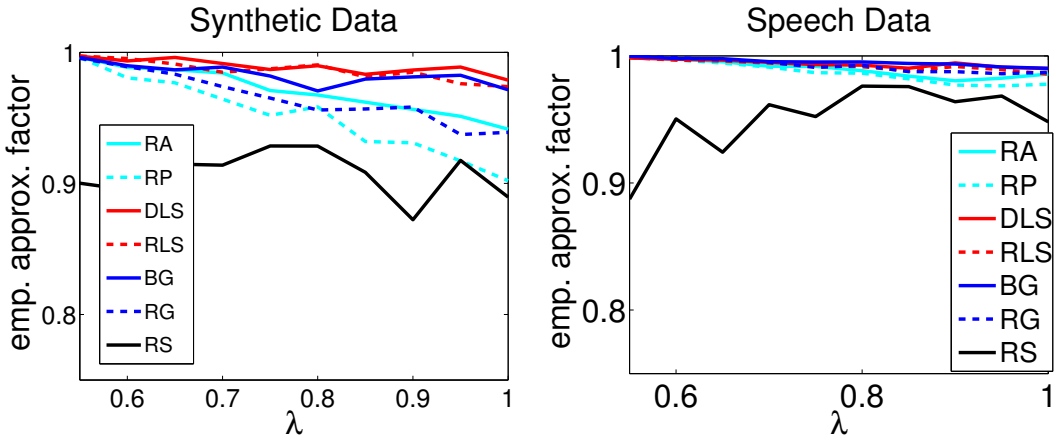

Figure 4.2: Empirical approximation factors for variants of MMax. See Section 4.4 for legend details.

Even though RBG has the best theoretical worst-case bounds, it performs slightly poorer than the local search ones and BG. Moreover, MMax with random subgradients (RP) is much better than choosing a set uniformly at random (RS). In general, the empirical approximation factors are much better than the theoretical worst-case bounds. Importantly, the MMax variants are extremely fast, about 200-500 times faster than the exact branch and bound technique of [Goldengorin et al., 1999].

4.7.2 Constrained Submodular Maximization

We now look at constrained versions. We consider both monotone and non-monotone submodular functions. We use the facility location objective, $f(X) = \sum_{i \in V} \max_{j \in X} s_{ij}$ as a test function with monotone functions. This function has been used as a coverage function in many application settings [Lin and Bilmes, 2011c, 2010, Wei et al., 2013]. In the non-monotone case, we use the same test function as the unconstrained case, i.e $f(X) = \sum_{i \in V} \sum_{j \in X} s_{ij} - \lambda \sum_{i,j \in X} s_{ij}$. We choose the tradeoff parameter $\lambda = 0.9$. In both cases, we consider synthetic and real data. Similar to the unconstrained setting, the synthetic data is generated from random similarity matrices, while the real data comes from speech data

subset selection. Again, we restrict n to lie between 20 and 30.

We compare the following choices of subgradients (a) Greedy Subgradient (GR), (b) Randomized Greedy subgradient (RG), (c) Random Adaptive (RA), (d) Random subgradient (RP), and (e) a baseline random set algorithm (RS), which picks a set uniformly at random of size k. We plot these results for k between 1 and $n/2$. Theoretically, RA, RP and RS achieve the same worst case guarantee, while RG achieves the tightest guarantee amongst these choices for non-monotone submodular functions. With monotone functions, RG and GR achieve the same performance guarantee.

We see that the Greedy algorithm (GR) achieves the best performance for both non-monotone and monotone functions (the results for the monotone setting are shown in Figure 4.3, while those for the non-monotone function are in Figure 4.4). While this is intuitive for monotone functions, the results are somewhat surprising for non-monotone functions (since GR can achieve factors of $\theta(1/n)$, while RG achieves constant factor guarantees). However, our choice of the non-monotone submodular function, is in fact approximately monotone [Lin and Bilmes, 2010], which explains the good empirical performance of GR.

We observe that MMax with a random permutation (RP) and its adaptive variant (RA) perform much better than the random set baseline. Moreover, the empirical factors seem to match the theory very well. For example, the random variants (RS, RP and RA) all achieve a factor of k/n (which grows with k), with monotone submodular functions. We observe that these algorithm all perform better with larger values of k.

4.8 Discussion

In this chapter, we introduced a Minorization-Maximization framework for submodular optimization algorithms. To our knowledge this is the first generic and unifying framework of combinatorial algorithms for submodular maximization. We show how several recently proposed combinatorial algorithms for submodular maximization, occur as instances of MMax, with specific subgradients. Empirically, we compare these different choices of subgradients for several submodular functions. A number of papers compare the maximization algorithms

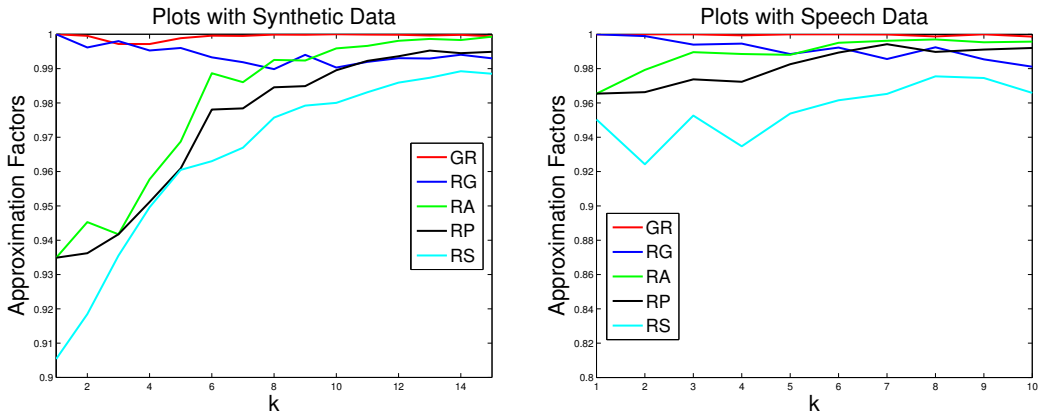

Figure 4.3: Empirical approximation factors for variants of MMax under cardinality constraints, and with the monotone submodular function as the facility location. See Section 4.5 for legend details.

we study in this chapter for other real world problems. For example, [Gillenwater, 2014] study these algorithms for MAP inference in Determinantal Point Processes, and [Reed and Ghahramani, 2013] study the MAP inference problem in latent feature models, both of which are instances of unconstrained submodular maximization. In both these cases, the authors compare the algorithms for unconstrained submodular maximization, and find conclusions similar to ours. Similarly, the greedy algorithm for submodular maximization has been used in several real world applications including document summarization, image summarization, sensor placement etc. [Lin, 2012, Tschiatschek et al., 2014, Krause and Guestrin, 2005b]. In all these cases, the greedy algorithm has been observed to perform considerably better than the pessimistic worst case approximation factors known so far.

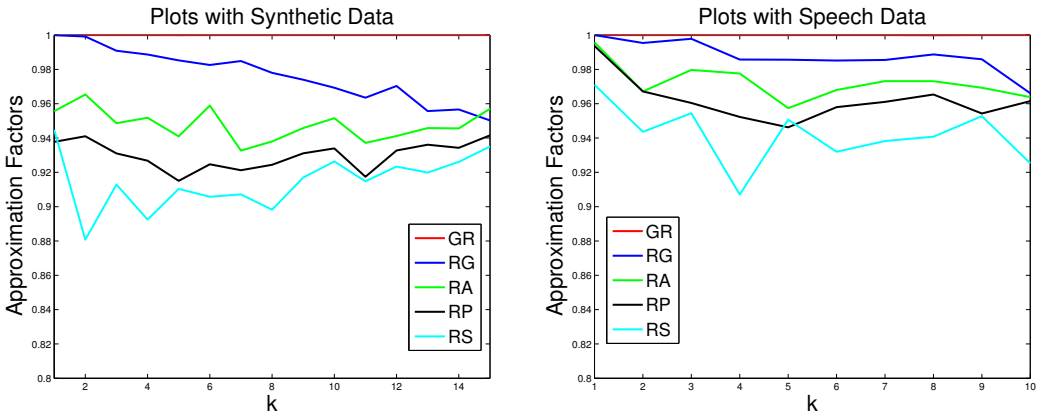

Figure 4.4: Empirical approximation factors for variants of MMax under cardinality constraints, and with a non-monotone submodular function. See Section 4.5 for legend details.

Chapter 5

DIFFERENCE OF SUBMODULAR OPTIMIZATION

5.1 Introduction

Given two submodular functions f and g, consider the following optimization problem:

$$\text{Problem 3: } \min_{X \subseteq V}[v(X)] \equiv \min_{X \subseteq V}[f(X) - g(X)] \tag{5.1}$$

where $v(X) \triangleq f(X) - g(X)$. This problem is equivalent to Problem 3′ which is maximizing the difference of submodular functions:

$$\text{Problem 3': } \max_{X \subseteq V}[-v(X)] \equiv \max_{X \subseteq V}[g(X) - f(X)] \tag{5.2}$$

We also note that Problem 3 is also equivalent to the problems of minimizing and maximizing the difference of two supermodular functions, since $\min(f-g) \equiv \min((-g)-(-f))$.

5.2 Motivating Applications

A number of machine learning problems involve minimization over a difference between submodular functions. The following are some examples:

- **Sensor placement with submodular costs:** The problem of choosing sensor locations A from a given set of possible locations V can be modeled [Krause and Guestrin, 2005b, Krause et al., 2008b] by maximizing the mutual information between the chosen variables A and the unchosen set $V \backslash A$ (i.e., $f(A) = I(X_A; X_{V \backslash A})$). Alternatively, we may wish to maximize the mutual information between a set of chosen sensors X_A and a fixed quantity of interest C (i.e., $f(A) = I(X_A; C)$) under the assumption that the set of features X_A are conditionally independent given C [Krause and Guestrin, 2005b]. These objectives are submodular and thus the problem becomes maximizing a

submodular function subject to a cardinality constraint. Often, however, there are costs $c(A)$ associated with the locations that naturally have a diminishing returns property. For example, there is typically a discount when purchasing sensors in bulk. Moreover, there may be diminished cost for placing a sensor in a particular location given placement in certain other locations (e.g., the additional equipment needed to install a sensor in, say, a precarious environment could be re-used for multiple sensor installations in like environments). Hence, along with maximizing mutual information, we also want to simultaneously minimize the cost and this problem can be addressed by minimizing the difference between submodular functions $f(A) - \lambda c(A)$ for tradeoff parameter λ.

- **Discriminatively structured graphical models and neural computation:** An application suggested in [Narasimhan and Bilmes, 2005] and the initial motivation for this problem is to optimize the EAR criterion to produce a discriminatively structured graphical model. EAR is basically a difference between two mutual information functions (i.e., a difference between submodular functions). [Narasimhan and Bilmes, 2005] shows how classifiers based on discriminative structure using EAR can significantly outperform classifiers based on generative graphical models. Note also that the EAR measure is the same as "synergy" in a neural code [Brenner et al., 2000], widely used in neuroscience.

- **Feature selection:** Given a set of features $X_1, X_2, \cdots, X_{|V|}$, the feature selection problem is to find a small subset of features X_A that work well when used in a pattern classifier. This problem can be modeled as maximizing the mutual information $I(X_A; C)$ where C is the class. Note that $I(X_A; C) = H(X_A) - H(X_A|C)$ is always a difference between submodular functions. Under the naïve Bayes model, this function is submodular [Krause and Guestrin, 2005b]. It is not submodular under general classifier models such as support vector machines (SVMs) or neural networks. Certain features, moreover, might be cheaper to use given that others are already being computed. For example, if a subset $S_i \subseteq V$ of the features for a particular information source i are spectral in nature, then once a particular $v \in S_i$ is chosen, the remaining features $S_i \setminus \{v\}$ may be relatively

inexpensive to compute, due to grouped computational strategies such as the fast Fourier transform. Therefore, it might be more appropriate to use a submodular cost model $c(A)$. One such cost model might be $c(A) = \sum_i \sqrt{m(A \cap S_i)}$ where $m(j)$ would be the cost of computing feature j. Another might be $c(A) = \sum_i c_i \min(|A \cap S_i|, 1)$ where c_i is the cost of source i. Both offer diminishing cost for choosing features from the same information source. Such a cost model could be useful even under the naïve Bayes model, where $I(X_A; C)$ is submodular. Feature selection becomes a problem of maximizing $I(X_A; C) - \lambda c(A) = H(X_A) - [H(X_A|C) + \lambda c(A)]$, the difference between two submodular functions.

- **Probabilistic Inference:** A typical instance of probabilistic inference is the following: We are given a distribution $p(x) \propto \exp(-v(x))$ where $x \in \{0,1\}^n$ and v is a pseudo-Boolean function [Boros and Hammer, 2002]. It is desirable to compute $\text{argmax}_{x \in \{0,1\}^n} p(x)$ which means minimizing $v(x)$ over x, the most-probable explanation (MPE) problem [Pearl, 1988]. If p factors with respect to a graphical model of tree-width k, then $v(x) = \sum_i v_i(x_C)$ where C_i is a bundle of indices such that $|C| \leq k+1$ and the sets $\mathcal{C} = \{C_i\}_i$ form a junction tree, and it might be possible to solve inference using dynamic programming. If k is large and/or if hypertree factorization does not hold, then approximate inference is typically used [Wainwright and Jordan, 2008]. On the other hand, defining $x(X) = \{x \in \{0,1\}^n : x_i = 1 \text{ whenever } i \in X\}$, if the set function $\bar{v}(X) = v(x(X))$ is submodular, then even if p has large tree-width, the MPE problem can be solved exactly in polynomial time [Jegelka and Bilmes, 2011c]. This, in fact, is the basis behind inference in many computer vision models where v is often not only submodular but also has limited sized $|C_i|$. For example, for submodular v and if $|C_i| \leq 2$ then graph-cuts can solve the MPE problem extremely rapidly [Kolmogorov and Zabih, 2004] and even some cases with v non-submodular [Kolmogorov and Rother, 2007]. An important challenge is to consider non-submodular v that can be minimized efficiently and for which there are approximation guarantees, a problem recently addressed in [Jegelka and Bilmes, 2011d]. On the other hand, if v can be expressed as a difference between two submodular func-

tions (which it can, see Lemma 5.1), or if such a decomposition can be computed (which it sometimes can, see Lemma 6.2), then a procedure to minimize the difference between two submodular functions offers new ways to solve probabilistic inference. As an example, a large class of rich higher potentials can be expressed as [Gallagher et al., 2011]:

$$f(x) = \sum_{C \in \mathcal{C}} w_C \prod_{i \in C} x_i \tag{5.3}$$

$\mathcal{C}$ here stands for a set of sets, possibly with higher-order terms (i.e there exist $C \in \mathcal{C} : |C| > 2$). If $w_C \leq 0, \forall C \in \mathcal{C}$, then f is submodular. If $|\mathcal{C}|$ is not large (say polynomial in n), we can efficiently find a decomposition into submodular components (which will contain the sets $C \in \mathcal{C} : w_C \leq 0$) and the supermodular terms (which contain sets $C \in \mathcal{C} : w_C \geq 0$). These can potentially represent a rich class of potential functions for a number of applications, particularly in vision.

Previously, [Narasimhan and Bilmes, 2005] proposed an algorithm inspired by the convex-concave procedure [Yuille and Rangarajan, 2002] to address Equation (5.2). This algorithm iteratively minimizes a submodular function by replacing the second submodular function g by it's modular lower bound. They also show that any set function can be expressed as a difference between two submodular functions and hence every set function optimization problem can be reduced to minimizing a difference between submodular functions. They show that this process converges to a local minima, however the convergence rate is left as an open question.

The road-map of this chapter is as follows. We first provide a constructive procedure for finding the submodular functions f and g for any arbitrary set function v. Although our construction is NP hard in general, we show how for certain classes of set functions v, it is possible to find the decompositions f and g in polynomial time. In Section 5.4, we propose two new algorithms both of which are guaranteed to monotonically reduce the objective at every iteration and which converge to a local minima. Further we note that the per-iteration cost of our algorithms is in general much less than [Narasimhan and Bilmes, 2005], and empirically verify that our algorithms are orders of magnitude faster on real data. We show

that, unlike in [Narasimhan and Bilmes, 2005], our algorithms can be extended to easily optimize equation (5.2) under cardinality, knapsack, and matroid constraints. Moreover, one of our algorithms can actually handle complex combinatorial constraints, such as spanning trees, matchings, cuts, etc. We then give a hardness result (Section 5.5) that there does not exist any polynomial time algorithm with any polynomial time multiplicative approximation guarantees, even when it is easy to find or when we are given the decomposition f and g, thus justifying the need for heuristic methods to solve this problem. We show, however, that it is possible to get additive bounds by showing polynomial time computable upper and lower bound on the optima. We also provide computational bounds for all our algorithms (including the submodular-supermodular procedure), a problem left open in [Narasimhan and Bilmes, 2005]. Finally we perform a number of experiments on the feature selection problem under various cost models, and show how our algorithms used to maximize the mutual information perform better than greedy selection (which would be near optimal under the naïve Bayes assumptions) and with less cost.

5.3 Generality of DS Functions

In this section, we show that any set function can be expressed as a DS function using suitable submodular functions as shown below. The result was first shown in [Narasimhan and Bilmes, 2005] using the Lovász extension. We here give a new combinatorial proof, which avoids Hessians of polyhedral convex functions and which provides a way of constructing (a non-unique) pair of submodular functions f and g for an arbitrary set function v.

Before providing the main result, define α_v as the *distance* from submodularity of the set function v as,

$$\alpha_v = \min_{j,X,Y:X\subset Y\subseteq V\setminus j} [v(j|X) - v(j|Y)] \tag{5.4}$$

For a non-submodular function v, $\alpha_v < 0$.

Lemma 5.1. *[Narasimhan and Bilmes, 2005] Given any set function v, it can be expressed as a DS functions $v(X) = f(X) - g(X), \forall X \subseteq V$ for some submodular functions f and g.*

Proof. Consider any (strictly) submodular function g, i.e., one having $\alpha_g = \min_{X \subset Y \subseteq V \setminus j}[g(j|X) - g(j|Y)] > 0$. Define $f'(X) = v(X) + \frac{|\alpha'|}{\alpha_g} g(X)$ with any $\alpha' \leq \alpha_v$. Now it is easy to see that f' is submodular since $\min_{X \subset Y \subseteq V \setminus j} f'(j|X) - f'(j|Y) \geq \alpha_v + |\alpha'| \geq 0$. Hence $v(X) = f'(X) - \frac{|\alpha'|}{\alpha_g} g(X)$, is a difference between two submodular functions. □

The above proof requires the computation of α_v and α_g which has, in general, exponential complexity. Using the construction above, however, it is easy to find the decomposition f and g under certain conditions on v.

Lemma 5.2. *Given a set function v, if α_v or at least a lower bound on α_v, can be computed in polynomial time, the functions f and g corresponding to v can obtained in polynomial time.*

Proof. Define g as $g(X) = \sqrt{|X|}$. Then $\beta = \min_{X \subset Y \subseteq V \setminus j}[\sqrt{|X|+1} - \sqrt{|X|} - \sqrt{|Y|+1} + \sqrt{|Y|}] = \min_{X \subset V \setminus j}[\sqrt{|X|+1} - \sqrt{|X|} - \sqrt{|X|+2} + \sqrt{|X|+1}] = 2\sqrt{n-1} - \sqrt{n} - \sqrt{n-2}$. The last inequality follows since the smallest difference in gains will occur at $|X| = n-2$. Hence α_g is easily computed, and given a lower bound on α, from Lemma 5.1 the decomposition can be obtained in polynomial time. A similar argument holds for g being other concave functions over $|X|$. □

5.4 Algorithms for DS Optimization

In this section we investigate the general Majorization-Minimization class of algorithms for DS optimization. These algorithms use the modular upper and modular lower bounds of a submodular function, and reduce the general optimization problem to a sequence of simpler optimization problems at every iteration. These modular upper and lower bounds are defined via sub and supergradients of a submodular function (see Section 1.4.1 for a definition of these). This framework subsumes the Submodular-Supermodular procedure [Narasimhan and Bilmes, 2005], and includes two new algorithms to minimize DS functions. All these algorithms are guaranteed to monotonically reduce the objective at every iteration and converge to local minima. We describe these algorithms in the subsections below.

Algorithm 10 The submodular-supermodular (SubSup) procedure [Narasimhan and Bilmes, 2005]

1: $X^0 = \emptyset$; $t \leftarrow 0$;
2: **while** not converged (i.e., $(X^{t+1} \neq X^t)$) **do**
3: Randomly choose a permutation σ^t whose chain contains the set X^t.
4: $X^{t+1} := \text{argmin}_X \, f(X) - h^g_{X^t,\sigma^t}(X)$
5: $t \leftarrow t + 1$
6: **end while**

5.4.1 The Submodular-Supermodular Procedure

We now review the submodular-supermodular procedure [Narasimhan and Bilmes, 2005] to minimize functions expressible as a difference between submodular functions (DS functions).

The submodular supermodular (SubSup) procedure is given in Algorithm 10. At every step of the algorithm, we minimize a submodular function which can be performed in strongly polynomial time [Orlin, 2009, Schrijver, 2000] although the best known complexity is $O(n^5\eta + n^6)$ where η is the cost of a function evaluation. The idea here, is to replace g by its modular lower bound, thereby obtaining a submodular upper bound of the difference $f(X) - g(X)$. To make the context clear, we shall denote the subgradient of g, induced by the ordering σ^t as $h^g_{X^t,\sigma^t}$.

Algorithm 10 is guaranteed to converge to a local minima and moreover the algorithm monotonically decreases the function objective at every iteration, as we show below.

Lemma 5.3. *[Narasimhan and Bilmes, 2005] Algorithm 10 is guaranteed to decrease the objective function at every iteration. Further, the algorithm is guaranteed to converge to a local minima by checking at most $O(n)$ permutations at every iteration.*

Proof. The objective reduces at every iteration since:

$$\begin{aligned} f(X^{t+1}) - g(X^{t+1}) &\overset{a}{\leq} f(X^{t+1}) - h^g_{X^t,\sigma^t}(X^{t+1}) \\ &\overset{b}{\leq} f(X^t) - h^g_{X^t,\sigma^t}(X^t) \\ &\overset{c}{=} f(X^t) - g(X^t) \end{aligned}$$

Where (a) follows since $h^g_{X^t,\sigma^t}(X^{t+1}) \leq g(X^{t+1})$, and (b) follows since X^{t+1} is the minimizer of $f(X) - h^g_{X^t,\sigma^t}(X)$, and (c) follows since $h^g_{X^t,\sigma^t}(X^t) = g(X^t)$ from the tightness of the modular lower bound.

Further note that, if there is no improvement in the function value by considering $O(n)$ permutations each with different elements at $\sigma^t(|X^t| - 1)$ and $\sigma^t(|X^t| + 1)$, then this is equivalent to a local minima condition on v since $h^g_{X^t,\sigma^t}(S^\sigma_{|X^t|+1}) = f(S^\sigma_{|X^t|+1})$ and $h^g_{X^t,\sigma^t}(S^\sigma_{|X^t|-1}) = f(S^\sigma_{|X^t|-1})$. □

Algorithm 10 requires performing a submodular function minimization at every iteration which while polynomial in n is (due to the complexity described above) not practical for large problem sizes. So while the algorithm reaches a local minima, it can be costly to find it. A desirable result, therefore, would be to develop new algorithms for minimizing DS functions, where the new algorithms have the same properties as the SubSup procedure but are much faster in practice. We give this in the following sections.

5.4.2 The Supermodular-Submodular (SupSub) procedure

In the submodular-supermodular procedure we iteratively minimized $f(X) - g(X)$ by replacing g by it's modular lower bound at every iteration. We can instead replace f by it's modular upper bound as is done in Algorithm 11, which leads to the *supermodular-submodular* procedure.

In the SupSub procedure, at every step we perform submodular maximization which, although NP complete to solve exactly, admits a number of fast constant factor approximation algorithms [Buchbinder et al., 2012, Feige et al., 2011a]. Notice that we have two modular

Algorithm 11 The supermodular-submodular (SupSub) procedure

1: $X^0 = \emptyset$; $t \leftarrow 0$;
2: **while** not converged (i.e., $(X^{t+1} \neq X^t)$) **do**
3: $\quad X^{t+1} := \text{argmin}_X \, m^f_{X^t}(X) - g(X)$
4: $\quad t \leftarrow t + 1$
5: **end while**

upper bounds and hence there are a number of ways we can choose between them. One way is to run both maximization procedures with the two modular upper bounds at every iteration in parallel, and choose the one which is better. Here by better we mean the one in which the function value is lesser. Alternatively we can alternate between the two modular upper bounds by first maximizing the expression using the first modular upper bound, and then maximize the expression using the second modular upper bound. Notice that since we perform approximate submodular maximization at every iteration, we are not guaranteed to monotonically reduce the objective value at every iteration. If, however, we ensure that at every iteration we take the next step only if the objective v does not increase, we will restore monotonicity at every iteration. Also, in some cases we converge to local optima as shown in the following theorem.

Theorem 5.1. *Both variants of the supermodular-submodular procedure (Algorithm 11) monotonically reduces the objective value at every iteration. Moreover, assuming a submodular maximization procedure in line 3 that reaches a local maxima of $m^f_{X^t}(X) - g(X)$, then if Algorithm 11 does not improve under both modular upper bounds then it reaches a local optima of v.*

Proof. For either modular upper bound, we have:

$$\begin{aligned} f(X^{t+1}) - g(X^{t+1}) &\overset{a}{\leq} m^f_{X^t}(X^{t+1}) - g(X^{t+1}) \\ &\overset{b}{\leq} m^f_{X^t}(X^t) - g(X^t) \\ &\overset{c}{=} f(X^t) - g(X^t), \end{aligned}$$

where (a) follows since $f(X^{t+1}) \leq m^f_{X^t}(X^{t+1})$, and (b) follows since we assume that we take the next step only if the objective value does not increase and (c) follows since $m^f_{X^t}(X^t) = f(X^t)$ from the tightness of the modular upper bound.

To show that this algorithm converges to a local minima, we assume that the submodular maximization procedure in line 3 converges to a local maxima. Then observe that if the objective value does not decrease in an iteration under both upper bounds, it implies that $m^f_{X^t}(X^t) - g(X^t)$ is already a local optimum in that (for both upper bounds) we have $m^f_{X^t}(X^t \cup j) - g(X^t \cup j) \geq m^f_{X^t}(X^t) - g(X^t), \forall j \notin X^t$ and $m^f_{X^t}(X^t \backslash j) - g(X^t \backslash j) \geq m^f_{X^t}(X^t) - g(X^t), \forall j \in X^t$. Note that $m^f_{X^t,1}(X^t \backslash j) = f(X^t) - f(j|X^t \backslash j) = f(X^t \backslash j)$ and $m^f_{X^t,2}(X^t \cup j) = f(X^t) + f(j|X^t) = f(X^t \cup j)$ and hence if both modular upper bounds are at a local optima, it implies $f(X^t) - g(X^t) = m^f_{X^t,1}(X^t) - g(X^t) \leq m^f_{X^t,1}(X^t \backslash j) - g(X^t \backslash j) = f(X^t \backslash j) - g(X^t \backslash j)$. Similarly $f(X^t) - g(X^t) = m^f_{X^t,2}(X^t) - g(X^t) \leq m^f_{X^t,2}(X^t \cup j) - g(X^t \cup j) = f(X^t \cup j) - g(X^t \cup j)$. Hence X^t is a local optima for $v(X) = f(X) - g(X)$, since $v(X^t) \leq v(X^t \cup j)$ and $v(X^t) \leq v(X^t \backslash j)$. □

To ensure that we take the largest step at each iteration, we can use the recently proposed tight (1/2)-approximation algorithm in [Buchbinder et al., 2012] for unconstrained non-monotone submodular function maximization — this is the best possible in polynomial time for the class of submodular functions independent of the P=NP question. The algorithm is a form of bi-directional randomized greedy procedure and, most importantly for practical considerations, is linear time [Buchbinder et al., 2012]. In practice we just use a combination of a form of a simple greedy procedure, and the bi-directional randomized algorithm, by picking the best amongst the two at every iteration. Since the randomized greedy algorithm is 1/2 approximate, the combination of the two procedures also will be 1/2 approximate.

Lastly, note that this algorithm is closely related to a local search heuristic for submodular maximization [Feige et al., 2011a]. In particular, if instead of using the greedy algorithm entirely at every iteration, we take only one local step, we get a local search heuristic. Hence, via the SupSub procedure, we may take larger steps at every iteration as compared to a local

search heuristic.

5.4.3 The Modular-Modular (ModMod) procedure

The submodular-supermodular procedure and the supermodular-submodular procedure were obtained by replacing g by it's modular lower bound and f by it's modular upper bound respectively. We can however replace both of them by their respective modular bounds, as is done in Algorithm 12.

Algorithm 12 Modular-Modular (ModMod) procedure

1: $X^0 = \emptyset$; $t \leftarrow 0$;
2: **while** not converged (i.e., $(X^{t+1} \neq X^t)$) **do**
3: Choose a permutation σ^t whose chain contains the set X^t.
4: $X^{t+1} := \text{argmin}_X \, m^f_{X^t}(X) - h^g_{X^t,\sigma^t}(X)$
5: $t \leftarrow t + 1$
6: **end while**

In this algorithm at every iteration we minimize only a modular function which can be done in $O(n)$ time, so this is extremely easy (i.e., select all negative elements for the smallest minimum, or all non-positive elements for the largest minimum). Like before, since we have two modular upper bounds, we can use any of the variants discussed in the subsection above. Moreover, we are still guaranteed to monotonically decrease the objective at every iteration and converge to a local minima.

Theorem 5.2. *Algorithm 12 monotonically decreases the function value at every iteration. If the function value does not increase on checking $O(n)$ different permutations with different elements at adjacent positions and with both modular upper bounds, then we have reached a local minima of v.*

Proof. Again we can use similar reasoning as the earlier proofs and observe that:

$$\begin{aligned} f(X^{t+1}) - g(X^{t+1}) &\leq m^f_{X^t}(X^{t+1}) - h^g_{X^t,\sigma^t}(X^{t+1}) \\ &\leq m^f_{X^t}(X^t) - h^g_{X^t,\sigma^t}(X^t) \\ &= f(X^t) - g(X^t) \end{aligned}$$

We see that considering $O(n)$ permutations each with different elements at $\sigma^t(|X^t|-1)$ and $\sigma^t(|X^t|+1)$, we essentially consider all choices of $g(X^t \cup j)$ and $g(X^t \backslash j)$, since $h^g_{X^t,\sigma^t}(S_{|X^t|+1}) = f(S_{|X^t|+1})$ and $h^g_{X^t,\sigma^t}(S_{|X^t|-1}) = f(S_{|X^t|-1})$. Since we consider both modular upper bounds, we correspondingly consider every choice of $f(X^t \cup j)$ and $f(X^t \backslash j)$. Note that at convergence we have that $m^f_{X^t}(X^t) - h^g_{X^t,\sigma^t}(X^t) \leq m^f_{X^t}(X) - h^g_{X^t,\sigma^t}(X), \forall X \subseteq V$ for $O(n)$ different permutations and both modular upper bounds. Correspondingly we are guaranteed that (since the expression is modular) $\forall j \notin X^t, v(j|X^t) \geq 0$ and $\forall j \in X^t, v(j|X^t \backslash j) \geq 0$, where $v(X) = f(X) - g(X)$. Hence the algorithm converges to a local minima. □

An important question is the choice of the permutation σ^t at every iteration X^t. We observe experimentally that the quality of the algorithm depends strongly on the choice of permutation. Observe that $f(X) - g(X) \leq m^f_{X^t}(X) - h^g_{X^t,\sigma^t}(X)$, and $f(X^t) - g(X^t) = m^f_{X^t}(X^t) - h^g_{X^t,\sigma^t}(X^t)$. Hence, we might obtain the greatest local reduction in the value of v by choosing permutation $\sigma^* \in \text{argmin}_\sigma \min_X (m^f_{X^t}(X) - h^g_{X^t,\sigma^t}(X))$, or the one which maximizes $h^g_{X^t,\sigma^t}(X)$. We in fact might expect that choosing σ^t ordered according to greatest gains of g, with respect to X^t, we would achieve greater descent at every iteration. Another choice is to choose the permutation σ based on the ordering of gains of v (or even $m^f_{X^t}$). Through the former we are guaranteed to at least progress as much as the local search heuristic. Indeed, we observe in practice that the first two of these heuristics performs much better than a random permutation for both the ModMod and the SubSup procedure, thus addressing a question raised in [Narasimhan and Bilmes, 2005] about which ordering to use. Practically for the feature selection problem, the second heuristic seems to work the best.

5.4.4 Constrained minimization of a difference between submodular functions

In this section we consider the problem of minimizing the difference between submodular functions subject to constraints. We first note that the problem of minimizing a submodular function under even simple cardinality constraints in NP hard and also hard to approximate [Svitkina and Fleischer, 2008]. Since there does not yet seem to be a reasonable algorithm for constrained submodular minimization at every iteration, it is unclear how we would use Algorithm 10. However the problem of submodular maximization under cardinality, matroid, and knapsack constraints though NP hard admits a number of constant factor approximation algorithms [Nemhauser et al., 1978, Lee et al., 2009a] and correspondingly the cardinality constraints can be easily introduced in Algorithm 11. Moreover, since a non-negative modular function can be easily, directly and even exactly optimized under cardinality, knapsack and matroid constraints [Jegelka and Bilmes, 2011a], Algorithm 12 can also easily be utilized. In addition, since problems such as finding the minimum weight spanning tree, min-cut in a graph, etc., are polynomial time algorithms in a number of cases, Algorithm 12 can be used when minimizing a non-negative function v expressible as a difference between submodular functions under combinatorial constraints. If v is non-negative, then so is its modular upper bound, and then the ModMod procedure can directly be used for this problem — each iteration minimizes a non-negative modular function subject to combinatorial constraints which is easy in many cases [Jegelka and Bilmes, 2011a, 2010].

5.5 Hardness of Approximation

The results in this section are mostly negative, in that they demonstrate theoretically how complex a general problem such as $\min_X[f(X) - g(X)]$ is, even for submodular f and g. In this chapter, rather than consider these hardness results pessimistically, we think of them as providing justification for the heuristic procedures given in Section 5.4. In many cases, inspired heuristics can yield good quality and hence practically useful algorithms for real-world problems. For example, the ModMod procedure (Algorithm 12) and even the

SupSub procedure (Algorithm 11) can scale to very large problem sizes, and thus can provide useful new strategies for the applications listed in Section 5.1.

Note that general set function minimization is inapproximable and there exist a large class of functions where all (adaptive, possibly randomized) algorithms perform arbitrarily poorly in polynomial time [Trevisan, 2004]. Since every set function can be written as a difference of submodular functions, it seems to suggest directly that Problem 3 is also inapproximable. However, note that given a general set function v, finding the corresponding submodular functions f and g, such that $v(X) = f(X) - g(X)$ could require exponential complexity. Hence, it is not immediate that DS optimization is also hard to approximate.

However, in this section, we show that DS optimization is NP hard to approximate, even when we are given oracle access to the functions f and g. Hence any algorithm trying to find the global optimum for this problem [Byrnes, 2009, Kawahara and Washio, 2011] can only be exponential in the worst case.

The first result (Theorem 5.3) shows that Problem 3 (minimizing DS functions) is multiplicatively inapproximable (upto any polynomial factor) unless P=NP. The second result (Theorem 5.4) provides an information theoretical proof of constant factor multiplicative inapproximability. We then improve this result, to show an information theoretic hardness result of a polynomial factor $\Omega(\sqrt{n}/\log n)$ (Theorem 5.5). Unlike the earlier two hardness results, this result holds even for a difference of Matroid rank functions [Kobayashi, 2014], and hence, the difference of $\mathrm{M}^\natural$ concave functions [Maheara and Murota, 2014]. Finally, Theorem 5.6 provides a polynomial factor multiplicative inapproximability for Problem 3′, i.e maximizing the difference between submodular functions. Again, this result also holds to the problems of maximizing the difference of matroid rank functions, and the difference of $\mathrm{M}^\natural$ concave functions.

5.5.1 P = NP Hardness for DS Minimization

We first provide a P=NP hardness result for DS minimization.

Theorem 5.3. *Unless $P = NP$, there cannot exist any polynomial time approximation algorithm for* $\min_X v(X)$ *where* $v(X) = [f(X) - g(X)]$ *is a positive set function and f and g are given submodular functions. In particular, let n be the size of the problem instance, and $\alpha(n) > 0$ be any positive polynomial time computable function of n. If there exists a polynomial-time algorithm which is guaranteed to find a set $X' : f(X') - g(X') < \alpha(n)OPT$, where OPT=*$\min_X f(X) - g(X)$*, then $P = NP$.*

Proof. We prove this by reducing this to the *subset sum* problem. Given a positive modular function m and a positive constant t, is there a subset $S \subseteq V$ such that $m(S) = t$? First we choose a random set C (unknown to the algorithm), and define $t = m(C)$. Define a set function v, such that $v(S) = 1$, if $m(S) = t$ and $v(S) = \frac{1}{\alpha(n)} - o(1)$ otherwise. Observe that $\min_S v(S) = \frac{1}{\alpha(n)} - o(1)$, since $\alpha(n) > 1$. Note that $\alpha = \min_{X \subset Y \subseteq V \setminus j} v(j|X) - v(j|Y) \geq 2(\frac{1}{\alpha(n)} - 1)$. Hence we can easily compute a lower bound on α and hence from lemma 6.2 we can directly compute the decomposition f and g. In fact notice that the decomposition is directly computable since both α and β are known.

Now suppose there exists a polynomial time algorithm for this problem with an approximation factor of $\alpha(n)$. This implies that the algorithm is guaranteed to find a set S, such that $v(S) < 1$. Hence this algorithm will solve the subset sum problem in polynomial time, which is a contradiction unless P = NP. □

5.5.2 Information Theoretic Hardness for DS Minimization

In this section, we investigate an information theoretic hardness result, which is independent of the $P = NP$ question. We first show that there cannot exist a sub-exponential time algorithm for this problem with any constant factor approximation. The theorem below gives information theoretic hardness for this problem. Unlike the P=NP proof above though, this is restricted to constant factor inapproximability, rather than the more general result of Theorem 5.3 which includes polynomial approximation factors.

Theorem 5.4. *For any constant $0 < \epsilon < 1$, there cannot exist any deterministic (or possibly*

randomized) algorithm for $\min_X[f(X)-g(X)]$ *(where* f *and* g *are given submodular functions), that always finds a solution which is at most* $\frac{1}{\epsilon}$ *times the optimal, in fewer than* $e^{\epsilon^2 n/8}$ *queries.*

Proof. For showing this theorem, we use the same proof technique as in [Feige et al., 2011a]. Define two sets C and D, such that $V = C \cup D$ and $|C| = |D| = n/2$. We then define a set function $v(S)$ which depends only on $k = |S \cap C|$ and $l = |S \cap D|$. In particular define $v(S) = \frac{1}{\epsilon},$ if $|k-l| \leq \epsilon n$ and $v(S) = 1,$ if $|k-l| > \epsilon n$. Again, we have a trivial bound on α_v here since $v(j|X) \geq \frac{1}{\epsilon} - 1$ and $v(j|Y) \leq 1 - \frac{1}{\epsilon}$. Hence, $\alpha_v = \min_{X \subset Y \subseteq V \setminus j}[v(j|X) - v(j|Y)] > 2|1 - \frac{1}{\epsilon}|$. Thus, for this set function, a decomposition $v = f - g$ can easily be obtained (Lemma 6.2).

Now, let the partition (C, D) be taken uniformly at random and unknown to the algorithm. The algorithm issues some queries S to the value oracle. Call S "unbalanced" if $|S \cap C|$ differs from $|S \cap D|$ by more than ϵn. Recall the Chernoff bounds [Alon and Spencer, 2000]: Let $Y_1, Y_2, \cdots, Y_t$ be independent random variables in $[-1, 1]$, such that $\mathbb{E}[Y_i] = 0$, then:

$$Pr[\sum_{i=1}^{t} Y_i > \lambda] \leq 2e^{-\lambda^2/2t}. \tag{5.5}$$

Define $Y_i = I(i \in S)[I(i \in C) - I(i \in D)]$. Clearly $Y_i \in [-1, 1]$, and we can use the bounds above. Hence for any query S, the probability that S is unbalanced is at most $2e^{-\epsilon^2 n/2}$. Thus, we can see that even after $e^{\epsilon^2 n/4}$ number of queries, the probability that the resulting set is unbalanced is still $2e^{-\epsilon^2 n/4}$. Hence any algorithm will query only balanced sets regardless of C and D, and consequently with high probability the algorithm will obtain $\frac{1}{\epsilon}$ as the minimum, while the actual minimum is 1. Thus, such an algorithm will never be able to achieve an approximation factor better than $\frac{1}{\epsilon}$. □

We improve the above result by showing an inapproximability upto a polynomial factor of $\Omega(\sqrt{n}/\log n)$.

Theorem 5.5. *There cannot exist a polynomial time approximation algorithm for* $\min_X v(X)$*, where* $v(X) = [f(X) - g(X)]$ *is a positive set function and* f *and* g *are given submodular functions, which achieves a factor better than* $\Omega(\sqrt{n/\log n}$*. This result is independent of*

P=NP, and holds even when the functions f and g are monotone submodular, and matroid rank functions.

Proof. We use the hardness construction from [Goemans et al., 2009, Svitkina and Fleischer, 2008]. The main idea is to construct two submodular functions $f(X)$ and $f_R(X)$ that with high probability are indistinguishable. Thus, also with high probability, no algorithm can distinguish between the two functions and the gap in their values provides a lower bound on the approximation. Then, define the DS function as,

$$v(X) = \kappa - f(X) + f_R(X) \tag{5.6}$$

Define the two monotone submodular functions $f(X) = \min\{|X|, \kappa\}$ and $f_R(X) = \min\{\beta + |X \cap \bar{R}|, |X|, \kappa\}$, where $R \subseteq V$ is a random set of cardinality κ. Let κ and β be integers such that $\kappa = x\sqrt{n}/5$ and $\beta = x^2/5$ for an $x^2 = \omega(\log n)$. A Chernoff bound analysis very similar to [Svitkina and Fleischer, 2008] reveals that any algorithm that uses a polynomial number of queries can distinguish f and f_R with probability only $n^{-\omega(1)}$, and therefore cannot reliably distinguish the functions with a polynomial number of queries.

Correspondingly, any algorithm which maximizes $v(X)$ cannot obtain a solution better than κ. The minimum solution on the other hand, is no bigger than the value at $X = R$, which is β. This means that the worst case factor is at least $\kappa/\beta = \Omega(\sqrt{n/\log n}$. □

This result is still somewhat weaker than the P=NP result (Theorem 5.3, which shows that DS minimization is inapproximable upto any polynomial factor. Whether there exists a similar information theoretic hardness result is an open problem.

The result above, however, holds even for a subclass of DS minimization problems, which is the Difference of Matroid Rank (DR) minimization problems [Kobayashi, 2014], since the submodular functions above are Matroid Rank functions. Correspondingly, the hardness factor of $\Omega(\sqrt{n/\log n})$ holds for DR minimization, and correspondingly, for the difference of $\mathrm{M}^\natural$ concave functions [Maheara and Murota, 2014]. [Kobayashi, 2014] show that DR minimization is NP hard. The result above further strengthens this to show a hardness of approximation upto a factor of $\Omega(\sqrt{n/\log n})$.

5.5.3 Information Theoretic Hardness for DS Maximization

Finally, we show an inapproximability result for maximizing the difference of submodular functions. This result is independent of P=NP, and is a general polynomial factor inapproximability result.

Theorem 5.6. *There cannot exist any polynomial time approximation algorithm for* $\max_X v(X)$ *where* $v(X) = [f(X) - g(X)]$ *is a positive set function and* f *and* g *are given submodular functions. In particular, let* n *be the size of the problem instance, and* $\alpha(n) > 0$ *be any positive polynomial time computable function of* n*. There cannot exists a polynomial-time algorithm which is guaranteed to find a set* $X' : f(X') - g(X') > \alpha(n)OPT$*, where* OPT=$\max_X f(X) - g(X)$*. This result is independent of P=NP.*

Proof. We prove this result using the hardness construction from [Goemans et al., 2009, Svitkina and Fleischer, 2008]. The main idea of their proof technique is to construct two submodular functions $f(X)$ and $f_R(X)$ that with high probability are indistinguishable. Thus, also with high probability, no algorithm can distinguish between the two functions and the gap in their values provides a lower bound on the approximation. Then, define the DS function as,

$$v(X) = f(X) - f_R(X) + 1/\alpha(n) \tag{5.7}$$

Define the two monotone submodular functions $f(X) = \min\{|X|, \kappa\}$ and $f_R(X) = \min\{\beta + |X \cap \bar{R}|, |X|, \kappa\}$, where $R \subseteq V$ is a random set of cardinality κ. Let κ and β be an integer such that $\kappa = x\sqrt{n}/5$ and $\beta = x^2/5$ for an $x^2 = \omega(\log n)$. A Chernoff bound analysis very similar to [Svitkina and Fleischer, 2008] reveals that any algorithm that uses a polynomial number of queries can distinguish f and fR with probability only $n^{-\omega(1)}$, and therefore cannot reliably distinguish the functions with a polynomial number of queries.

Correspondingly, any algorithm which maximizes $v(X)$ cannot obtain a solution better than $1/\alpha(n)$. The maximum solution on the other hand, is no worse than the value at $X = R$, which is $\kappa - \beta + 1/\alpha(n)$. This means that the worst case factor is at least $\alpha(n)(\kappa - \beta) > \alpha(n)$.

Hence, any algorithm which achieves a factor better than $\alpha(n)$ must distinguish between f and f_R, which is a contradiction. □

The result above holds for DR maximization as well, since the submodular functions above are Matroid Rank functions. Hence the problems of maximizing the difference of Matroid Rank functions, and correspondingly, correspondingly, for the difference of $\mathrm{M}^\natural$ concave functions [Maheara and Murota, 2014], are both inapproximable upto any polynomial factor.

5.5.4 Hardness Results for Optimizing the Difference of Monotone Submodular Functions

All the hardness results above hold even when the submodular functions f and g are monotone. This follows from the following Lemma:

Lemma 5.4. *Given (not necessarily monotone) submodular functions f and g, there exists monotone submodular functions f' and g' such that,*

$$f(X) - g(X) = f'(X) - g'(X), \forall X \subseteq V \tag{5.8}$$

Proof. The proof of this Lemma follows from a simple observation. The decomposition theorem of [Cunningham, 1983] shows that any submodular function can be decomposed into a modular function plus a monotone non-decreasing and *totally normalized* polymatroid rank function. Specifically, given submodular f, g we have

$$f'(X) \triangleq f(X) - \sum_{j \in X} f(j|V\backslash j) \tag{5.9}$$

and

$$g'(X) \triangleq g(X) - \sum_{j \in X} g(j|V\backslash j) \tag{5.10}$$

f', g' are then totally normalized polymatroid rank functions. Hence we have: $v(X) = f'(X) - g'(X) + k(X)$, with modular $k(X) = \sum_{j \in X} v(j|V\backslash j)$. The idea is then to add $v(j)$

to f' if $v(j) \geq 0$ or add it to g' other-wise. In particular, let $V^+ = \{j : v(j) \geq 0\}$ and $V^- = \{j : v(j) < 0\}$. Notice that $V^+ \cup V^- = V$. Then,

$$v(X) = f'(X) + k(X \cap V^+) - \{g' - k(X \cap V^-)\} \tag{5.11}$$

Notice above that $f'(X) + k(X \cap V^+)$ and $g' - k(X \cap V^-)$ are both monotone non-decreasing. Hence proved. □

This then implies the following corollary.

Corollary 5.1. *Given submodular functions f and g such that $v(X) = f(X) - g(X) \geq 0$, the problem $\min_{X \subseteq V} v(X)$ is inapproximable, even if both f and g are monotone non-decreasing submodular.*

5.6 Additive Approximation Results and Complexity Bounds

In this section, we analyze the computational and approximation bounds for this problem. For simplicity we assume that the function v is normalized, i.e $v(\emptyset) = 0$. Hence we assume that v achieves it minima at a negative value (in the case of DS minimization) and maxima at a positive value (in the case of DS maximization).

5.6.1 Additive Approximation Bounds

Since any submodular function can be decomposed into a modular function plus a monotone non-decreasing and *totally normalized* polymatroid rank function [Cunningham, 1983], we have: $v(X) = f'(X) - g'(X) + k(X)$, with modular $k(X) = \sum_{j \in X} v(j|V \backslash j)$ and f' and g' being the totally normalized polymatroid functions.

The algorithms in the previous sections are all based on repeatedly finding upper bounds for v. The following lower bound on $\min_X v(X)$, provides an additive approximation bound for DS minimization:

Theorem 5.7. *We have the following two lower bounds on the minimizers of* $v(X) = f(X) - g(X)$*:*

$$\min_X v(X) \geq \min_X f'(X) + k(X) - g'(V)$$
$$\min_X v(X) \geq f'(\emptyset) - g'(V) + \sum_{j \in V} \min(k(j), 0)$$

Proof. Notice that

$$\begin{aligned}\min_X f(X) - g(X) &= \min_X f'(X) - g'(X) + k(X) \\ &\geq \min_X (f'(X) + k(X)) - \max_X g'(X) \\ &= \min_X f'(X) + k(X) - g'(V)\end{aligned}$$

To get the second result, we start from the bound above and loosen it as:

$$\begin{aligned}&\min_X f'(X) + k(X) - g'(V) \\ &\geq \min_X f'(X) + \min_X k(X) - g'(V) \\ &= f'(\emptyset) + \sum_{j \in V} \min(v(j|V \backslash j), 0) - g'(V) \\ &= f'(\emptyset) + \sum_{j \in V} \min(k(j), 0) - g'(V) \qquad (5.12)\end{aligned}$$

□

We can also provide an additive bound for DS maximization.

Theorem 5.8. *We have the following two upper bounds on the maximizers of* $v(X) = f(X) - g(X)$*:*

$$\max_X v(X) \leq f'(V) - \min_X \{g'(X) - k(X)\}$$
$$\max_X v(X) \leq f'(V) - g'(\emptyset) + \sum_{j \in V} \max(k(j), 0)$$

Proof. The proof of this follows along the same lines as the proof above. □

The above bounds essentially provide bounds on the minima (or maxima) of the objective and thus can be used to obtain an additive approximation guarantee. The algorithms described in this chapter are all polynomial time algorithms (as we show below) and correspondingly from the bounds above we can get an estimate on how far we are from the optimal.

5.6.2 Computational Bounds

We now provide computational bounds for ϵ-approximate versions of our algorithms. Note that this was left as an open question in [Narasimhan and Bilmes, 2005]. Finding the local minimizer of DS functions is PLS complete since it generalizes the problem of finding the local optimum of the MAX-CUT problem [Schäffer, 1991]. Note that this trivially generalizes the MAX-CUT problem since if we set $f(X) = 0$ and $g(X)$ is the cut function, we get the max cut problem.

However we show that an ϵ-approximate version of this algorithm will converge in polynomial time.

Definition 5.1. *An ϵ-approximate version of an iterative monotone non-decreasing algorithm for minimizing a set function v is defined as a version of that algorithm, where we proceed to step $t+1$ only if $v(X^{t+1}) \leq v(X^t)(1+\epsilon)$.*

Note that the ϵ-approximate versions of algorithms 10, 11 and 12, are guaranteed to converge to ϵ-approximate local optima. An ϵ-approximate local optima of a function v is a set X, such that $v(X \cup j) \geq v(X)(1+\epsilon)$ and $v(X \backslash j) \geq v(X)(1+\epsilon)$. W.l.o.g., assume that $X^0 = \emptyset$. Then we have the following computational bounds:

Theorem 5.9. *The ϵ-approximate versions of algorithms 10, 11 and 12 have a worst case complexity of $O(\frac{\log(|M|/|m|)}{\epsilon}T))$, where $M = f'(\emptyset) + \sum_{j \in V} \min(v(j|V \backslash j), 0) - g'(V)$, $m = v(X^1)$ and $O(T)$ is the complexity of every iteration of the algorithm (which corresponds to respectively the submodular minimization, maximization, or modular minimization in algorithms 10, 11 and 12).*

Proof. Observe that $m = v(X^1) \leq v(X^0) = 0$. Correspondingly if $v(X^1) = 0$, it implies that the algorithm has converged, and cannot improve (since we are assuming our algorithms are $\epsilon-$approximate. Hence in this case the algorithm will converge in one iteration. Consider then the case of $m < 0$. Note also from Theorem 5.8 that $M = f'(\emptyset) + \sum_{j \in V} \min(v(j|V \backslash j), 0) - g'(V) < 0$ and that $\min_X f(X) - g(X) \geq M$. Since we are guaranteed to improve by a factor by at least $1 + \epsilon$ at every iteration we have that in k iterations: $|m|(1+\epsilon)^k \leq |M| \Rightarrow k = O(\frac{\log(|M|/|m|)}{\epsilon})$. Also since we assume that the complexity at every iteration is $O(T)$ we get the above result. □

The bound above is for DS minimization. A very similar guarantee can be provided for DS maximization (using an upper bound on the maximizer of the DS function). Observe that for the algorithms we use, $O(T)$ is strongly polynomial in n. The best strongly polynomial time algorithm for submodular function minimization is $O(n^5\eta + n^6)$ [Orlin, 2009] (the lower bound is currently unknown). Further the worst case complexity of the greedy algorithm for maximization is $O(n^2)$ while the complexity of modular minimization is just $O(n)$. Note finally that these are worst case complexities and actually the algorithms run much faster in practice.

5.7 Experiments

We test our algorithms on the feature subset selection problem in the supervised setting. Given a set of features $X_V = \{X_1, X_2, \cdots, X_{|V|}\}$, we try to find a subset of these features A which has the most information from the original set X_V about a class variable C under constraints on the size or cost of A. Normally the number of features $|V|$ is quite large and thus the training and testing time depend on $|V|$. In many cases, however, there is a strong correlation amongst features and not every feature is novel. We can thus perform training and testing with a much smaller number of features $|A|$ while obtaining (almost) the same error rates.

The question is how to find the most representative set of features A. The mutual information between the chosen set of features and the target class C, $I(X_A; C)$, captures

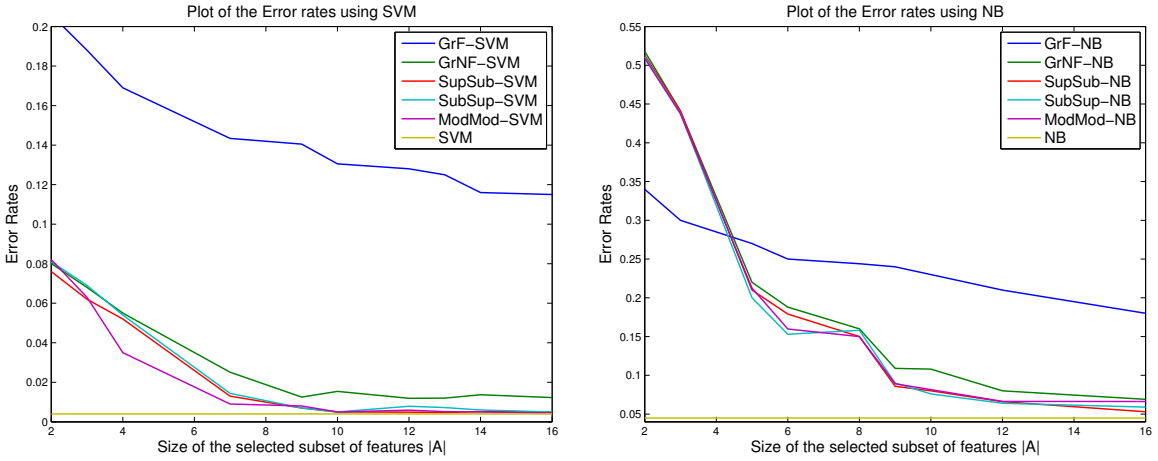

Figure 5.1: Plot showing the accuracy rates vs. the number of features on the Mushroom data set.

the relevance of the chosen subset of features. In most cases the selected features are not independent given the class C so the naïve Bayes assumption is not applicable, meaning this is not a pure submodular optimization problem. As mentioned in Section 5.1, $I(X_A; C)$ can be exactly expressed as a difference between submodular functions $H(X_A)$ and $H(X_A|C)$.

5.7.1 Modular Cost Feature Selection

In this subsection, we look at the problem of maximizing $I(X_A; C) - \lambda|A|$, as a regularized feature subset selection problem. Note that a mutual information $I(X_A; C)$ query can easily be estimated from the data by just a single sweep through this data. Further we have observed that using techniques such as Laplace smoothing helps to improve mutual information estimates without increasing computation. In these experiments, therefore, we estimate the mutual information directly from the data and run our algorithms to find the representative subset of features.

We compare our algorithms on two data sets, i.e., the Mushroom data set [Iba et al., 1988] and the Adult data set [Kohavi, 1996] obtained from [Frank and Asuncion, 2010]. The

Mushroom data set has 8124 examples with 112 features, while the Adult data set has 32,561 examples with 123 features. In our experiments we considered subsets of features of sizes between 5%-20% of the total number of features by varying λ. We tested the following algorithms for the feature subset selection problem. We considered two formulations of the mutual information, one under naïve Bayes, where the conditional entropy $H(X_A|C)$ can be written as $H(X_A|C) = \sum_{j \in A} H(X_i|C)$ and another where we do not assume such factorization. We call these two formulations *factored* and *non-factored* respectively. We then considered the simple greedy algorithm, of iteratively adding features at every step to the factored and non-factored mutual information, which we call GrF and GrNF respectively. Lastly, we use the new algorithms presented in this chapter on the non-factored mutual information.

We then compare the results of the greedy algorithms with those of the three algorithms for this problem, using two pattern classifiers based on either a linear kernel SVM (using [Chang and Lin, 2011]) or a naïve Bayes (NB) classifier. We call the results obtained from the supermodular-submodular heuristic as "SupSub", the submodular-supermodular procedure [Narasimhan and Bilmes, 2005] as "SubSup", and the modular-modular objective as "ModMod." In the SubSup procedure, we use the minimum norm point algorithm [Fujishige and Isotani, 2011] for submodular minimization, and in the SubSup procedure, we use the optimal algorithm of [Buchbinder et al., 2012] for submodular maximization. We observed that the three heuristics generally outperformed the two greedy procedures, and also that GRF can perform quite poorly, thus justifying our claim that the naïve Bayes assumption can be quite poor. This also shows that although the greedy algorithm in that case is optimal, the features are correlated given the class and hence modeling it as a difference between submodular functions gives the best results. We also observed that the SupSub and ModMod procedures perform comparably to the SubSup procedure, while the SubSup procedure is *much* slower in practice. Comparing the running times, the ModMod and the SupSub procedure are each a few times slower then the greedy algorithm (ModMod is slower due computing the modular semigradients), while the SubSup procedure is around 100 times slower. The SubSup

procedure is slower due to general submodular function minimization which can be quite slow.

The results for the Mushroom data set are shown in Figure 5.1. We performed a 10 fold cross-validation on the entire data set and observed that when using all the features SVM gave an accuracy rate of 99.6% while the all-feature NB model had an accuracy rate of 95.5%. The results for the Adult database are in Figure 5.2. In this case with the entire set of features the accuracy rate of SVM on this data set is 83.9% and NB is 82.3%.

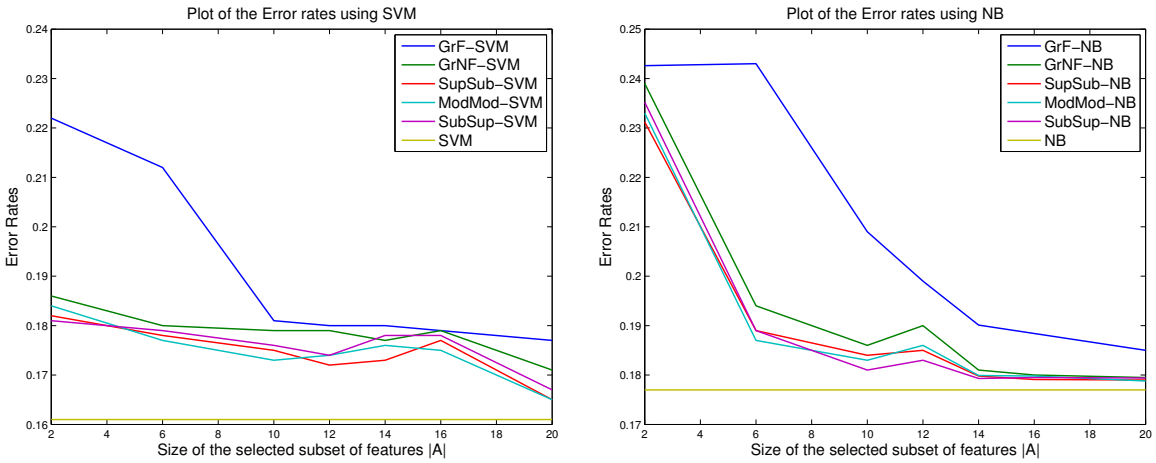

Figure 5.2: Plot showing the accuracy rates vs. the number of features on the Adult data set.

In the mushroom data, the SVM classifier significantly outperforms the NB classifier and correspondingly GrF performs much worse than the other algorithms. Also, in most cases the three algorithms outperform GrNF. In the adult data set, both the SVM and NB perform comparably although SVM outperforms NB. However in this case also we observe that our algorithms generally outperform GrF and GrNF.

5.7.2 Submodular cost feature selection

We perform synthetic experiments for the feature subset selection problem under submodular costs. The cost model we consider is $c(A) = \sum_i \sqrt{m(A \cap S_i)}$. We partitioned V into sets $\{S_i\}_i$ and chose the modular function m randomly. In this set of experiments, we compare the

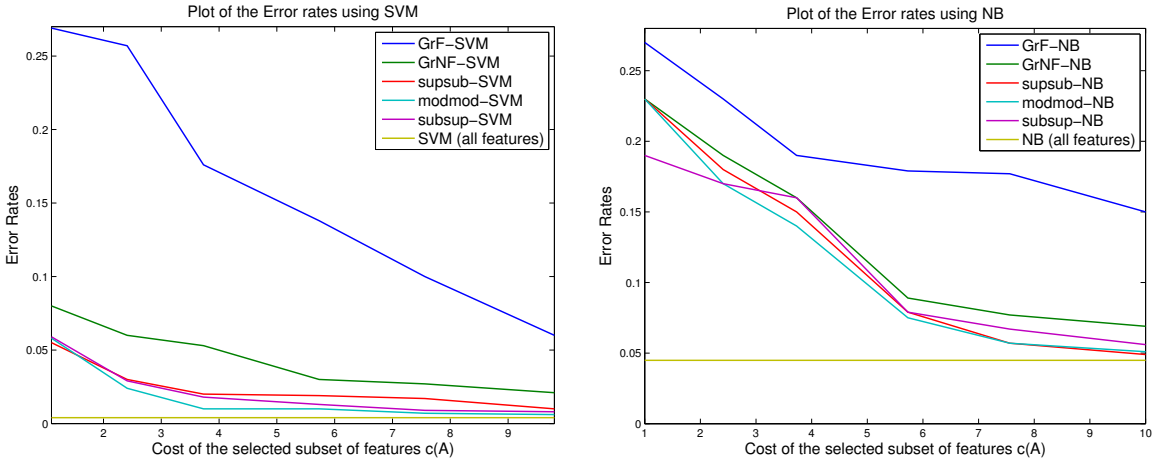

Figure 5.3: Plot showing the accuracy rates vs. the cost of features for the Mushroom data set

accuracy of the classifiers vs. the *cost* associated with the choice of features for the algorithms. Recall, with simple (modular) cardinality costs the greedy algorithms performed decently in comparison to our algorithms in the adult data set, where the NB assumption is reasonable. However with submodular costs, the objective is no longer submodular even under the NB assumption and thus the greedy algorithms perform much worse. This is unsurprising since the greedy algorithm is approximately optimal only for monotone submodular functions. This is even more strongly evident from the results of the mushrooms data-set (Figure 5.3)

5.8 Discussion

We have introduced new algorithms for optimizing the difference between two submodular functions, provided new theoretical understanding that provides some justification for heuristics, have outlined applications that can make use of our procedures, and have tested in the case of feature selection with modular and submodular cost features. Our new ModMod procedure is fast at each iteration and experimentally does about as well as the SupSub and SubSup procedures. The ModMod procedure, moreover, can also be used under various com-

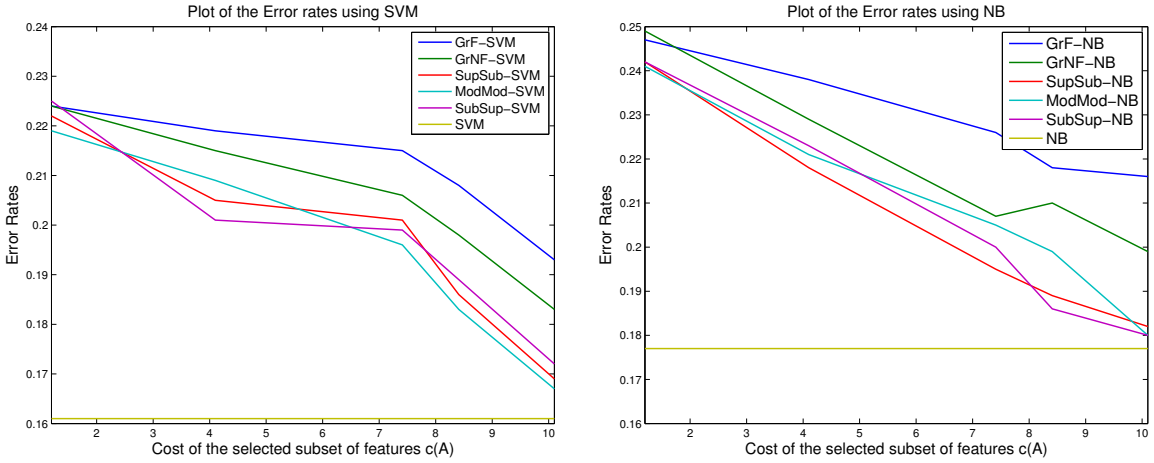

Figure 5.4: Plot showing the accuracy rates vs. the cost of features for the Adult data set

binatorial constraints, and therefore the ModMod procedure may hold the greatest promise as a practical heuristic.

Another algorithm which works for general DS optimization, is the work of [Kawahara and Washio, 2011], which though exact, employs a branch and bound technique, which is inefficient in practice (the timing analysis from [Kawahara and Washio, 2011] also depicts that). Hence this approach is not likely to scale to large scale optimization and machine learning problems.

A number of papers build upon our work. [Maheara and Murota, 2014], extend our framework, to handle differences of $\mathrm{L}^\natural$ and $\mathrm{M}^\natural$ concave functions (which generalize submodular set functions). Similarly, a number of papers have used our algorithms for real world problems. In the feature subset selection application considered in this chapter, we observe that using the exact objective function with a heuristic algorithm, performs better than an approximate objective function (which is the naïve Bayes assumption) and a more accurate algorithm. This fact was also observed in the context of Learning submodular functions for summarization applications [Lin, 2012, Tschiatschek et al., 2014]. The subproblems involved in the algorithm of learning submodular functions, naturally occur as a difference of submodular functions,

when one uses the true loss function for learning. [Tschiatschek et al., 2014] observed that using the true loss function along with the DS optimization algorithms from this chapter, works better using an approximate loss function (where one can then use the standard greedy algorithm with theoretical guarantees) [Tschiatschek et al., 2014, Lin, 2012]. Again, we notice that using the *true* objective function, can perform better than an approximate objective, even though the approximate one admits theoretically tighter guarantees. Finally, [Pasumarthi et al., 2014] investigate the problem of targeted influence maximization, and show that the problem occurs as a maximization problem over a difference of submodular functions. They use our DS optimization framework, and observe that the algorithms perform much better than baselines which approximate the objective function.

Chapter 6

SUBMODULAR OPTIMIZATION SUBJECT TO SUBMODULAR CONSTRAINTS

6.1 Introduction

In this chapter, we study a new class of discrete optimization problems that have the following form:

Problem 4 (SCSC): $\min\{f(X) \,|\, g(X) \geq c\}$, and Problem 5 (SCSK): $\max\{g(X) \,|\, f(X) \leq b\}$,

where f and g are monotone non-decreasing submodular functions that also, w.l.o.g., are normalized ($f(\emptyset) = g(\emptyset) = 0$)[1], and where b and c refer to budget and cover parameters respectively. The corresponding constraints are called the submodular cover [Wolsey, 1982] and submodular knapsack [Atamtürk and Narayanan, 2009] respectively and hence we refer to Problem 4 as *Submodular Cost Submodular Cover* (henceforth SCSC) and Problem 5 as *Submodular Cost Submodular Knapsack* (henceforth SCSK). Our motivation stems from an interesting class of problems that require minimizing a certain submodular function f while simultaneously maximizing another submodular function g. We shall see that these naturally occur in applications like sensor placement, data subset selection, and many other machine learning applications. A standard approach used in literature [Iyer and Bilmes, 2012b, Narasimhan and Bilmes, 2005, Kawahara and Washio, 2011] has been to transform these problems into minimizing the difference between submodular functions (DS optimization, which we studied in Chapter 5):

$$\text{Problem 3: } \min_{X \subseteq V} \big(f(X) - g(X)\big). \tag{6.1}$$

Unfortunately, we saw that Problem 3 is NP-hard and inapproximable (in the worst case), even when f and g are monotone. Although an exact branch and bound algorithm has been provided

[1] A monotone non-decreasing normalized ($f(\emptyset) = 0$) submodular function is called a polymatroid function.

for this problem [Kawahara and Washio, 2011], its complexity can be exponential in the worst case. While a number of well motivated heuristics in the form of majorization-minimization style algorithms were investigated in Chapter 5, in this chapter, we show that reformulations of Problem 3, in the form of Problems 4 and 5 admit algorithms with bounded approximation guarantees. On the other hand, in many applications, one of the submodular functions naturally serves as part of a constraint, in which case, Problems 4 and 5 are fitting models.

Problems 4 and 5 generalize a number of well-studied discrete optimization problems. For example the *Submodular Set Cover* problem (henceforth SSC) [Wolsey, 1982] occurs as a special case of Problem 4, with f being modular and g is submodular. Similarly the *Submodular Cost Knapsack* problem (henceforth SK) [Sviridenko, 2004] is a special case of Problem 5 again when f is modular and g submodular. Both these problems subsume the *Set Cover* and *Max k-Cover* problems [Feige, 1998]. When both f and g are modular, Problems 4 and 5 are called *knapsack problems* [Kellerer et al., 2004]. Furthermore, Problem 4 also subsumes the cardinality constrained submodular minimization problem [Svitkina and Fleischer, 2008] and more generally the problem of minimizing a submodular function subject to a knapsack constraints. It also subsumes the problem of minimizing a submodular function subject to a matroid span constraint (by setting g as the rank function of the matroid, and c as the rank of the matroid). This in turn subsumes the minimum submodular spanning tree problem [Goel et al., 2009], when f is monotone submodular.

The following are some of our contributions. We show that Problems 4 and 5 are intimately connected, in that any approximation algorithm for either problem can be used to provide guarantees for the other problem as well. We then provide a framework of combinatorial algorithms based on optimizing, sometimes iteratively, subproblems that are easy to solve. These subproblems are obtained by computing either upper or lower bound approximations of the cost functions or constraining functions. We also show that many combinatorial algorithms like the greedy algorithm for SK [Sviridenko, 2004] and SSC [Wolsey, 1982] (both of which seemingly use different techniques) also belong to this framework and provide the first constant-factor bi-criterion approximation algorithm for SSC [Wolsey, 1982] and hence

the general set cover problem [Feige, 1998]. We then show how with suitable choices of approximate functions, we can obtain a number of bounded approximation guarantees and show the hardness for Problems 4 and 5, which in fact match some of our approximation guarantees up to log-factors. Our guarantees and hardness results depend on the *curvature* of the submodular functions f and g. We observe a strong asymmetry in the results that the factors change polynomially based on the curvature of f but only by a constant-factor with the curvature of g, hence making the SK and SSC much easier compared to SCSK and SCSC. Finally we empirically evaluate the performance of our algorithms showing that the typical case behavior is much better than these worst case bounds.

6.2 Motivating Applications

The utility of Problems 4 and 5 become apparent when we consider how they occur in real-world applications and how they subsume a number of important optimization problems.

Sensor Placement and Feature Selection: Often, the problem of choosing sensor locations A from a given set of possible locations V can be modeled [Krause et al., 2008b, Iyer and Bilmes, 2012b] by maximizing the mutual information between the chosen variables A and the unchosen set $V \backslash A$ (i.e.,$f(A) = I(X_A; X_{V \backslash A})$). Note that, while the symmetric mutual information is not monotone, it can be shown to approximately monotone [Krause et al., 2008b]. Alternatively, we may wish to maximize the mutual information between a set of chosen sensors X_A and a quantity of interest C (i.e., $f(A) = I(X_A; C)$) assuming that the set of features X_A are conditionally independent given C [Krause et al., 2008b, Iyer and Bilmes, 2012b]. Both these functions are submodular. Since there are costs involved, we want to simultaneously minimize the cost $g(A)$. Often this cost is submodular [Krause et al., 2008b, Iyer and Bilmes, 2012b]. For example, there is typically a discount when purchasing sensors in bulk (economies of scale). Moreover, there may be diminished cost for placing a sensor in a particular location given placement in certain other locations (e.g., the additional equipment needed to install a sensor in, say, a precarious environment could be re-used for

multiple sensor installations in similar environments). Hence this becomes a form of Problem 5 above. An alternate view of this problem is to find a set of sensors with minimal cooperative cost, under a constraint that the sensors cover a certain fraction of the possible locations, naturally expressed as Problem 4.

Data subset selection: A data subset selection problem in speech and NLP involves finding a limited vocabulary which simultaneously has a large coverage. This is particularly useful, for example in speech recognition and machine translation, where the complexity of the algorithm is determined by the vocabulary size. The motivation for this problem is to find the subset of training examples which will facilitate evaluation of prototype systems [Lin and Bilmes, 2011a]. This problem then occurs in the form of Problem 4, where we want to find a small vocabulary subset (which is often submodular [Lin and Bilmes, 2011a]), subject to a constraint that the subset acoustically spans the entire data set (which is also often submodular [Lin and Bilmes, 2009, 2010]). This can also be phrased as Problem 5, where we ask for maximizing the acoustic coverage and diversity subject to a bounded vocabulary size constraint.

Privacy Preserving Communication: Given a set of random variables $X_1, \cdots, X_n$, denote $\mathfrak{I}$ as an information source, and $\mathfrak{P}$ as private information that should be filtered out. Then one way of formulating the problem of choosing a information containing but privacy preserving set of random variables can be posed as instances of Problems 4 and 5, with $f(A) = H(X_A|\mathfrak{I})$ and $g(A) = H(X_A|\mathfrak{P})$, where $H(\cdot|\cdot)$ is the conditional entropy. An alternative strategy would be to formulate the problem with $f(A) = H(X_A) + H(X_A|\mathfrak{I})$ and $g(A) = H(X_A) + H(X_A|\mathfrak{P})$.

Machine Translation : Another application in machine translation is to choose a subset of training data that is optimized for given test data set, a problem previously addressed with modular functions [Moore and Lewis, 2010]. Defining a submodular function with ground

set over the union of training and test sample inputs $V = V_{\text{tr}} \cup V_{\text{te}}$, we can set $f : 2^{V_{\text{tr}}} \to \mathbb{R}_+$ to $f(X) = f(X|V_{\text{te}})$, and take $g(X) = |X|$, and $b \approx 0$ in Problem 5 to address this problem. We call this the *Submodular Span* problem.

6.3 Combinatorial Algorithmic Framework

In this chapter, we provide a framework of algorithms based on choosing appropriate surrogate functions for f and g to optimize over. This framework is represented in Algorithm 13. We would like to choose surrogate functions $\hat{f}_t$ and $\hat{g}_t$ such that using them, Problems 4 and 5 become easier. If the algorithm is just single stage (not iterative), we represent the surrogates as $\hat{f}$ and $\hat{g}$. The surrogate functions we consider in this chapter are in the forms of upper and lower bounds, defined via sub and supergradients (see Section 1.4.1 for a definition of these), and approximations (for example, the Ellipsoidal Approximation). Our algorithms belong to the Majorization-Minimization framework of algorithms, similar to the ones investigated for Problems 1, 2 and 3. In the case of submodular minimization, maximization and DS optimization, this framework subsumes a large class of known combinatorial algorithms and provides a generic recipe for different forms of submodular function optimization. In this chapter, we extend these ideas to Problems 4 and 5, to obtain a fast family of algorithms. The other type of surrogate functions we consider are those obtained from other approximations of the functions. One such classical approximation is the ellipsoidal approximations [Goemans et al., 2009]. While computing this approximation is time consuming, it turns out to provide the tightest theoretical guarantees.

Algorithm 13 General algorithmic framework to address both Problems 4 and 5

1: **for** $t = 1, 2, \cdots, T$ **do**
2: Choose surrogate functions $\hat{f}_t$ and $\hat{g}_t$ for f and g respectively, tight at X^{t-1}.
3: Obtain X^t as the optimizer of Problem 4 or 5 with $\hat{f}_t$ and $\hat{g}_t$ instead of f and g.
4: **end for**

6.3.1 Majorization-Minimization Algorithms

Using the modular upper and lower bounds above in Algorithm 13, provide a class of Majorization-Minimization (MM) algorithms, akin to the algorithms proposed in Chapters 3, 4 and 5. An appropriate choice of the bounds ensures that the algorithm always improves the objective values for Problems 4 and 5. In particular, choosing $\hat{f}_t$ as a modular upper bound of f tight at X^t, i.e $\hat{f}_t(X) = m^f_{X^t}(X)$, or $\hat{g}_t$ as a modular lower bound of g tight at X^t. This ensures that the objective value of Problems 4 and 5 always improves at every iteration as long as the corresponding surrogate problem can be solved exactly. Unfortunately, Problems 4 and 5 are NP-hard even if f or g (or both) are modular [Feige, 1998], and therefore the surrogate problems themselves cannot be solved exactly. Fortunately, the surrogate problems are often much easier than the original ones and can admit log or constant-factor guarantees. In practice, moreover, these factors are almost 1. In order to guarantee improvement from a theoretical stand-point however, the iterative schemes can be slightly modified using the following trick. Notice that the only case when the true valuation of X^t is better than X^{t+1} is when the surrogate valuation of X^t is better than X^{t+1} (since X^{t+1} is only near-optimal and not optimal). In such case, we terminate the algorithm at X^t. What is also fortunate and perhaps surprising, is that unlike the case of DS optimization (where the problem is inapproximable in general, the constrained forms of optimization (Problems 4 and 5) do have approximation guarantees. It is also interesting to note that the algorithms proposed in this chapter, are very similar to the majorization-minimization algorithms we proposed for DS optimization (Problem 3), and for Submodular Minimization and Maximization (Problems 1 and 2).

6.3.2 Ellipsoidal Approximation

We also consider ellipsoidal approximations (EA) of f. Recall that we can compute this surrogate function $\hat{f}$, by approximating the submodular polyhedron by an ellipsoid. This resulting function is of the form $\sqrt{w^f(X)}$ for a certain modular weight vector $w^f \in \mathbb{R}^V$, and

satisfies, $\sqrt{w^f(X)} \leq f(X) \leq O(\sqrt{n}\log n)\sqrt{w^f(X)}, \forall X \subseteq V$.

Similar to the procedure in Chapter 3, we compute a curve-normalized version of f:

$$f^\kappa(X) \triangleq \frac{f(X) - (1-\kappa_f)\sum_{j\in X} f(j)}{\kappa_f}. \tag{6.2}$$

The function f^κ essentially contains the zero curvature component of the submodular function f and the modular upper bound $\sum_{j\in X} f(j)$ contains all the linearity. The main idea is to then approximate only the polymatroidal part and retain the linear component. This simple trick improves many of the approximation bounds. Given a polymatroid function f with a curvature $\kappa_f < 1$, the submodular function $f^{\text{ea}}(X) = \kappa_f\sqrt{w^{f^\kappa}(X)} + (1-\kappa_f)\sum_{j\in X} f(j)$ satisfies:

$$f^{\text{ea}}(X) \leq f(X) \leq O\left(\frac{\sqrt{n}\log n}{1 + (\sqrt{n}\log n - 1)(1-\kappa_f)}\right) f^{\text{ea}}(X), \forall X \subseteq V \tag{6.3}$$

f^{ea} is multiplicatively bounded by f by a factor depending on $\sqrt{n}$ and the curvature. The dependence on the curvature is evident from the fact that when $\kappa_f = 0$, we get a bound of $O(1)$, which is not surprising since a modular f is exactly represented as $f(X) = \sum_{j\in X} f(j)$. We shall use the result above in providing approximation bounds for Problems 4 and 5. In particular, the surrogate functions $\hat{f}$ or $\hat{g}$ in Algorithm 13 can be the ellipsoidal approximations above, and the multiplicative bounds transform into approximation guarantees for these problems.

6.4 Relation between SCSC and SCSK

In this section, we show a precise relationship between Problems 4 and 5. From the formulation of Problems 4 and 5, it is clear that these problems are duals of each other. Indeed, in this section we show that the problems are polynomially transformable into each other.

We first introduce the notion of bi-criteria algorithms. An algorithm is a $[\alpha, \beta]$ bi-criterion algorithm for Problem 4 if it is guaranteed to obtain a set X such that $f(X) \leq \alpha f(X^*)$ (approximate optimality) and $g(X) \geq c' = \beta c$ (approximate feasibility), where X^* is an optimizer of Problem 4. Similarly, an algorithm is a $[\beta, \alpha]$ bi-criterion algorithm for Problem

5 if it is guaranteed to obtain a set X such that $g(X) \geq \beta g(X^*)$ and $f(X) \leq b' = \alpha b$, where X^* is the optimizer of Problem 5. In a bi-criterion algorithm for Problems 4 and 5, typically $\alpha \geq 1$ and $\beta \leq 1$. We call these type of approximation algorithms, bi-criterion approximation algorithms of type 1.

We can also view the bi-criterion approximations from another angle. We can say that for Problem 4, $\hat{X}$ is a feasible solution (i.e., it satisfies the constraint), and is a $[\alpha, \beta]$ bi-criterion approximation, if $f(\hat{X}) \leq \alpha f(\hat{X}^*)$, where $\hat{X}^*$ is the optimal solution to the problem $\min\{f(X)|g(X) \geq c/\beta\}$. Similarly for Problem 5, we can say that $\hat{X}$ is a feasible solution (i.e., it satisfies the constraint), and is a $[\beta, \alpha]$ bi-criterion approximation, if $g(\hat{X}) \geq \beta g(\hat{X}^*)$. Here $\hat{X}^*$ is the optimal solution to the problem $\max\{g(X)|f(X) \leq b\alpha\}$. We call these the bi-criterion approximation algorithms of type 2.

It is easy to see that these algorithms can easily be transformed into each other. For example, in the case of problem 4, a bi-criterion algorithm of type-I can obtain a guarantee of type-II if we run it till a covering constraint of c'/β (where c' in this case, is the approximate covering constraint which the type-I algorithm needs to satisfy – note that it need not be the actual covering constraint of the problem). Similarly an algorithm of type-II can obtain a guarantee of type-I if run till a covering constraint of βc (in this case, c is the actual 'covering' constraint, since a type-II approximate algorithm provides a feasible set). We can similarly transform these guarantees for Problem 5. In particular, a bi-criterion algorithm of type-I can be used to obtain a guarantee of type-II if we run it till a budget of $b'\alpha$ (again, here b' is the approximate budget of the type-I algorithm). Similarly an algorithm of type-II can obtain a guarantee of type-I if run till a covering constraint of b/α (in this case, b is the budget of the original problem).

Algorithm 14 Approx. algorithm for SCSK using an approximation algorithm for SCSC using Linear search.

Input: An SCSK instance with budget b, an $[\alpha, \beta]$ approx. algo. for SCSC, & $\epsilon \in [0, 1)$.
Output: $[(1-\epsilon)\beta, \alpha]$ approx. for SCSK.
$c \leftarrow g(V), \hat{X}_c \leftarrow V$.
while $f(\hat{X}_c) > \alpha b$ **do**
    $c \leftarrow (1-\epsilon)c$
    $\hat{X}_c \leftarrow [\alpha, \beta]$ approx. for SCSC using c.
end while
Return $\hat{X}_c$

Algorithm 15 Approx. algorithm for SCSC using an approximation algorithm for SCSK using Linear search.

Input: An SCSC instance with cover c, an $[\beta, \alpha]$ approx. algo. for SCSK, & $\epsilon > 0$.
Output: $[(1+\epsilon)\alpha, \beta]$ approx. for SCSC.
$b \leftarrow \text{argmin}_j f(j), \hat{X}_b \leftarrow \emptyset$.
while $g(\hat{X}_b) < \beta c$ **do**
    $b \leftarrow (1+\epsilon)b$
    $\hat{X}_b \leftarrow [\beta, \alpha]$ approx. for SCSK using b.
end while
Return $\hat{X}_b$.

Figure 6.1: Bi-criterion transformation algorithms for SCSC and SCSK using Linear search.

Though both type-I and type-II guarantees are easily transformable into each other, through the rest of this chapter whenever we refer to bi-criterion approximations, we shall consider only the type-I approximations.

A *non-bicriterion* algorithm for Problem 4 is when $\beta = 1$ and a *non-bicriterion* algorithm

for Problem 5 is when $\alpha = 1$. Algorithms 14 and 15 provide the schematics for using an approximation algorithm for one of the problems for solving the other.

The main idea of Algorithm 14 is to start with the ground set V and reduce the value of c (which governs SCSC), until the valuation of f just falls below αb. At that point, we are guaranteed to get a $((1-\epsilon)\beta, \alpha)$ solution for SCSK. Similarly in Algorithm 15, we increase the value of b starting at the empty set, until the valuation at g falls above βc. At this point we are guaranteed a $((1+\epsilon)\alpha, \beta)$ solution for SCSC. In order to avoid degeneracies, we assume that $f(V) \geq b \geq \min_j f(j)$ and $g(V) \geq c \geq \min_j g(j)$, else the solution to the problem is trivial.

Theorem 6.1. *Algorithm 14 is guaranteed to find a set $\hat{X}_c$ which is a $[(1-\epsilon)\beta, \alpha]$ approximation of SCSK in at most $\log_{1/(1-\epsilon)}[g(V)/\min_j g(j)]$ calls to the $[\alpha, \beta]$ approximate algorithm for SCSC. Similarly, Algorithm 15 is guaranteed to find a set $\hat{X}_b$ which is a $[(1+\epsilon)\alpha, \beta]$ approximation of SCSC in $\log_{1+\epsilon}[f(V)/\min_j f(j)]$ calls to a $[\beta, \alpha]$ approximate algorithm for SCSK.*

Proof. We start by proving the first part, for Algorithm 14. Notice that Algorithm 14 converges when $f(\hat{X}_c)$ just falls below αb. Hence $f(\hat{X}_c) \leq \alpha b$ (is approximately feasible) and at the previous step $c' = c/(1-\epsilon)$, we have that $f(\hat{X}_{c'}) > \alpha b$. Denoting $X^*_{c'}$ as the optimal solution for SCSC at c', we have that $f(X^*_{c'}) > b$ (a fact which follows from the observation that $\hat{X}_c$ is a $[\alpha, \beta]$ approximation of SCSC at c). Hence if X^* is the optimal solution of SCSK, it follows that $g(X^*) < c'$. The reason for this is that, suppose, $g(X^*) \geq c'$. Then it follows that X^* is a feasible solution for SCSC at c' and hence $f(X^*) \geq f(X^*_{c'}) > b$. This contradicts the fact that X^* is an optimal solution for SCSK (since it is then not even feasible).

Next, notice that $\hat{X}_c$ satisfies that $g(\hat{X}_c) \geq \beta c$, using the fact that $\hat{X}_c$ is obtained from a (α, β) bi-criterion algorithm for SCSC. Hence,

$$g(\hat{X}_c) \geq \beta c = \beta(1-\epsilon)c' > \beta(1-\epsilon)g(X^*) \tag{6.4}$$

Hence the Algorithm 14 is a $((1-\epsilon)\beta, \alpha)$ approximation for SCSK.

Algorithm 16 Binary Search Conversion from SCSC to SCSK.

Input: An SCSK instance with budget b, an $[\alpha, \beta]$ approx. algo. for SCSC, & $\epsilon \in [0, 1)$.
Output: $[(1-\epsilon)\beta, \alpha]$ approx. for SCSK.
$c_{\min} \leftarrow \min_j g(j), c_{\max} \leftarrow g(V)$
while $c_{\max} - c_{\min} \geq \epsilon c_{\max}$ **do**
  $c \leftarrow [c_{\max} + c_{\min}]/2$
  $\hat{X}_c \leftarrow [\alpha, \beta]$ approx. for SCSC using c.
  if $f(\hat{X}_c) > \alpha b$ **then**
    $c_{\max} \leftarrow c$
  else
    $c_{\min} \leftarrow c$
  end if
end while
Return $\hat{X}_{c_{\min}}$

Algorithm 17 Binary search conversion from SCSK to SCSC.

Input: An SCSC instance with cover c, an $[\beta, \alpha]$ approx. algo. for SCSK, & $\epsilon > 0$.
Output: $[(1+\epsilon)\alpha, \beta]$ approx. for SCSC.
$b_{\min} \leftarrow \text{argmin}_j f(j), b_{\max} \leftarrow f(V)$.
while $b_{\max} - b_{\min} \geq \epsilon b_{\min}$ **do**
  $b \leftarrow [b_{\max} + b_{\min}]/2$
  $\hat{X}_b \leftarrow [\beta, \alpha]$ approx. for SCSK using b
  if $g(\hat{X}_b) < \beta c$ **then**
    $b_{\min} \leftarrow b$
  else
    $b_{\max} \leftarrow b$
  end if
end while
Return $\hat{X}_{b_{\max}}$

Figure 6.2: Bicriterion transformation algorithms for SCSC and SCSK using Binary search.

In order to show the converge rate, notice that $c \geq \min_j g(j) > 0$. Since $\alpha \geq 1$ and $b \geq \min_j f(j)$, we can guarantee that this algorithm will stop before c reaches $\min_j g(j)$. The reason is that, when $c = \min_j g(j)$, the minimum value of $f(X)$ such that $g(X) \geq c$ is $\min_j f(j)$, which is smaller than b. Moreover, since $\alpha > 1$, it implies that the algorithm would have terminated before this point.

The proof for the second part of the statement, for Algorithm 15, is omitted since it is shown using a symmetric argument. □

Theorem 6.1 implies that the complexity of Problems 4 and 5 are identical, and a solution to one of them provides a solution to the other. Furthermore, as expected, the hardness of Problems 4 and 5 are also almost identical. When f and g are polymatroid functions, moreover, we can provide bounded approximation guarantees for both problems, as shown in the next section.

Alternatively we can also do a binary search instead of a linear search to transform Problems 4 and 5. This essentially turns the factor of $O(1/\epsilon)$ into $O(\log 1/\epsilon)$. Algorithms 16 and 17 show the transformation algorithms using binary search. While the binary search also ensures the same performance guarantees, it does so more efficiently.

Theorem 6.2. *Algorithm 16 is guaranteed to find a set $\hat{X}_c$ which is a $[(1-\epsilon)\beta, \alpha]$ approximation of SCSK in at most $\log_2 \frac{[g(V)/\min_j g(j)]}{\epsilon}$ calls to the $[\alpha, \beta]$ approximate algorithm for SCSC. Similarly, Algorithm 17 is guaranteed to find a set $\hat{X}_b$ which is a $[(1+\epsilon)\alpha, \beta]$ approximation of SCSC in $\log_2 \frac{[f(V)/\min_j f(j)]}{\epsilon}$ calls to a $[\beta, \alpha]$ approximate algorithm for SCSK. When f and g are integral, moreover, Algorithm 16 obtains a $[\beta, \alpha]$ bi-criterion approximate solution for SCSK in $\log_2 g(V)$ iterations, and similarly Algorithm 17 obtains a $[\alpha, \beta]$ bi-criterion approximate solution for SCSC in $\log_2 g(V)$ iterations.*

Proof. To show this theorem, we use the result from Theorem 6.1. Let $c = c_{\min}$ and $c' = c_{\max}$. An important observation is that throughout the algorithm, the values of $c_{\min}$ satisfy $f(\hat{X}_{c_{\min}}) \leq \alpha b$ and $f(\hat{X}_{c_{\max}}) > \alpha b$. Hence, $f(\hat{X}_c) \leq \alpha b$ and $f(\hat{X}_{c'}) > \alpha b$. Moreover,

notice that $c'/c = c_{\max}/c_{\min} = 1/(1-\epsilon)$. Hence using the proof of Theorem 6.1 the approximation guarantee follows.

In order to show the complexity, notice that $c_{\max} - c_{\min}$ is decreasing throughout the algorithm. At the beginning, $c_{\max} - c_{\min} \leq g(V)$ and at convergence, $c_{\max} - c_{\min} \geq \epsilon c_{\max}/2 \geq \epsilon \min_j g(j)/2$. The bound at convergence holds since, let c'_{max} and c'_{min} be the values at the previous step. It holds that $c'_{\max} - c'_{\min} \geq \epsilon c'_{\max}$. Moreover, $c_{max} - c_{min} = (c'_{\max} - c'_{\min})/2 \geq \epsilon c'_{\max}/2 \geq \epsilon c_{\max}$. Hence the number of iterations is bounded by $\log_2 \frac{[2g(V)/\min_j g(j)]}{\epsilon}$. Moreover, when f and g are integral, the analysis is much simpler. In particular, notice that once $c_{max} - c_{min} = 1$, the algorithm will stop at the next iteration (this is because at this point, $c = (c_{max} + c_{min})/2$ is equivalent to $c = c_{max}$). Hence, the number of iterations is bounded by $\log_2 g(V)$, and we can exactly obtain a $[\beta, \alpha]$ bi-criterion approximation algorithm.

The proof for the second part of the statement, for Algorithm 15, similarly follows using a symmetric argument.

□

When f and g are integral, this removes the ϵ dependence on the factors and could potentially be much faster in practice. We also remark that a specific instance of such a transformation has been used [Krause et al., 2008a], for a specific class of functions f and g. We shall show in section 6.7 that their algorithm is in fact a special case of Algorithm 16 through a specific construction to convert the non-submodular problem into an instance of a submodular one.

Algorithm 14 and Algorithm 15 indeed provide an interesting theoretical result connecting the complexity of Problems 4 and 5. In the next section however, we provide distinct algorithms for each problem when f and g are polymatroid functions — this turns out to be faster than having to resort to the iterative reductions above.

6.5 Approximation Algorithms for SCSC

We first describe our approximation algorithms designed specifically for SCSC, leaving to §6.6 the presentation of our algorithms slated for SCSK. We first investigate a special case, the submodular set cover (SSC), and then provide two algorithms, one of them (ISSC) is very practical with a weaker theoretical guarantee, and another one (EASSC) which is slow but has the tightest guarantee.

6.5.1 Submodular Set Cover (SSC)

Algorithm 18 Greedy Algorithm for $\max\{\sum_{i\in X} f(i) | g(X) \geq c\}$ [Wolsey, 1982]

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

$y = \text{argmin}_{v \in V \setminus Y_n} \frac{f(v)}{g(v|Y_n)}$;

$Y_{n+1} = Y_n \cup y$;

$n \leftarrow n + 1$;

until $g(Y_n) > B$;

return Y_{n-1}.

We start by considering a classical special case of SCSC (Problem 4) where f is already a modular function and g is a submodular function. This problem occurs naturally in a number of problems related to active/online learning [Guillory and Bilmes, 2010] and summarization [Lin and Bilmes, 2010, 2011b]. This problem was first investigated by Wolsey [Wolsey, 1982], wherein he showed that a simple greedy algorithm (shown in Algorithm 18) achieves bounded (in fact, log-factor) approximation guarantees. We show that this greedy algorithm can naturally be viewed in the framework of our Algorithm 13 by choosing appropriate surrogate functions $\hat{f}_t$ and $\hat{g}_t$. The idea is to use the modular function f as its own surrogate $\hat{f}_t$ and choose the function $\hat{g}_t$ as a modular lower bound of g. Akin to the framework of algorithms in [Iyer et al., 2013b], the crucial factor is the choice of the lower bound (or subgradient).

Define the *greedy subgradient* (equivalently the *greedy permutation*) as:

$$\sigma(i) \in \operatorname{argmin} \left\{ \frac{f(j)}{g(j|S^\sigma_{i-1})} \;\middle|\; j \notin S^\sigma_{i-1}, g(S^\sigma_{i-1} \cup j) < c \right\}. \tag{6.5}$$

Once we reach an i where the constraint $g(S^\sigma_{i-1} \cup j) < c$ can no longer be satisfied by any $j \notin S^\sigma_{i-1}$, we choose the remaining elements for σ arbitrarily. Let the corresponding subgradient be referred to as h^σ. Let N be the minimum i such that $g(S^\sigma_i) \geq c$ and $\theta_i = \min_{j \notin S^\sigma_{i-1}} \frac{f(j)}{g(j|S^\sigma_{i-1})}$. Then we have the following lemma, which is an extension of [Wolsey, 1982]:

Lemma 6.1. *Choosing the surrogate function $\hat{f}$ as f and $\hat{g}$ as h^σ (with σ defined in Eqn. (6.5)) in Algorithm 13, at the end of the first iteration, we are guaranteed to obtain a set X^g such that*

$$\frac{f(X^g)}{f(X^*)} \leq 1 + \log_e \min\{\lambda_1, \lambda_2, \lambda_3\} \tag{6.6}$$

where $\lambda_1 = \max\{\frac{1}{1-\kappa_g(S^\sigma_i)} |\ i : c(S^\sigma_i) < 1\}$, $\lambda_2 = \frac{\theta_N}{\theta_1}$ and $\lambda_3 = \frac{g(V)-g(\emptyset)}{g(V)-g(S^\sigma_{N-1})}$. Furthermore if g is integral, $\frac{f(X^g)}{f(X^*)} \leq H(\max_j g(j))$, where $H(d) = \sum_{i=1}^d \frac{1}{i}$ for a positive integer d.

Proof. The permutation σ is chosen based on a greedy ordering associated with the submodular set cover problem, and therefore $h^\sigma(S^\sigma_N) = g(S^\sigma_N) \geq c$ (where the inequality follows from the definition of N), and thus S^σ_N is a feasible solution in the surrogate problem (where g is replaced by h^σ). The resulting knapsack problem can be addressed using the greedy algorithm [Kellerer et al., 2004], but this exactly corresponds to the greedy algorithm of submodular set cover [Wolsey, 1982] and hence the guarantee follows from Theorem 1 in [Wolsey, 1982]. □

The surrogate problem in this instance is a simple knapsack problem that can be solved nearly optimally using dynamic programming [Vazirani, 2004]. As stated in the proof, the greedy algorithm for the submodular set cover problem [Wolsey, 1982] is in fact equivalent to using the greedy algorithm for the knapsack problem [Kellerer et al., 2004], which is in fact suboptimal. When g is integral, the guarantee of the greedy algorithm is $H_g \triangleq H(\max_j g(j))$, where $H(d) = \sum_{i=1}^d \frac{1}{i}$ [Wolsey, 1982] (henceforth we will use H_g for this quantity). This

factor is tight up to lower-order terms [Feige, 1998]. Furthermore, since this algorithm directly solves SSC, we call it the *primal greedy*. We could also solve SSC by looking at its *dual*, which is SK [Sviridenko, 2004]. Although SSC does not admit any constant-factor approximation algorithms [Feige, 1998], we can obtain a constant-factor *bi-criterion* guarantee:

Lemma 6.2. *Using the greedy algorithm for SK [Sviridenko, 2004] as the approximation oracle in Algorithm 15 provides a* $[1+\epsilon, 1-e^{-1}]$ *bi-criterion approximation algorithm for SSC, for any* $\epsilon > 0$.

Proof. We call this the *dual greedy*. This result follows immediately from the guarantee of the submodular cost knapsack problem [Sviridenko, 2004] and Theorem 6.1. □

We remark that we can also use a simpler version of the greedy iteration at every iteration [Lin and Bilmes, 2010, Krause and Guestrin, 2005a] and we obtain a guarantee of $(1+\epsilon, 1/2(1-e^{-1}))$. In practice, however, both these factors are almost 1 and hence the simple variant of the greedy algorithm suffices.

An interesting connection between the greedy algorithm and the induced orderings, allows us to further simplify this dual algorithm. A nice property of the greedy algorithm for the submodular knapsack problem is that it can be completely parameterized by the chain of sets – this holds for the greedy algorithm of [Lin and Bilmes, 2010, Krause and Guestrin, 2005a] for knapsack constraints, and the basic greedy algorithm of [Nemhauser and Wolsey, 1978] under cardinality constraints. In particular, having computed the greedy chain of sets, and given a value of b or the budget, we can easily find the corresponding set in $O(\log n)$ time using binary search. Moreover, this also implies that we can do the transformation algorithms by just iterating through the chain of sets once. In particular, the linear search over the different values of ϵ is equivalent to the linear search over the different chain of sets. Moreover, we could also do the much faster binary search. Hence the complexity of the dual greedy algorithm is almost identical to the primal greedy one for the submodular set cover problem.

It is also important to put the bicriterion result into perspective. Notice that the bicriterion guarantee suggests that we find only an approximate feasible solution. In particular, the dual greedy algorithm provides a type-I bi-criterion approximation, – that the solution obtained by running the algorithm with a cover constraint of $(1 - 1/e)c$ is competitive to the optimal solution with a cover constraint of c. However, we can also obtain a type-II bi-criterion approximation by running the dual greedy algorithm until it satisfies the cover constraint of c. In this case, we would obtain a feasible solution. The guarantee, however, would say that the resulting solution would be competitive to the optimal solution obtained with a cover constraint of $c/(1 - e^{-1})$. Furthermore, these factors are in practice close to 1, and the primal and dual greedy algorithms would both perform very well empirically.

6.5.2 Iterated Submodular Set Cover (ISSC)

We next investigate an algorithm for the general SCSC problem when both f and g are submodular. The idea here is to iteratively solve the submodular set cover problem which can be done by replacing f by a modular upper bound at every iteration. In particular, this can be seen as a variant of Algorithm 13, where we start with $X^0 = \emptyset$ and choose $\hat{f}_t(X) = m^f_{X^t,2}(X)$ at every iteration (alternatively, we can choose $\hat{f}_t(X) = m^f_{X^t,1}(X)$). At the first iteration with $X^0 = \emptyset$, either variant then corresponds to the set cover problem with the simple modular upper bound $f(X) \leq m^f_\emptyset(X) = \sum_{j \in X} f(j)$ where $m^f_{X^t}$ refers to either variant. The surrogate problem at each iteration becomes

$$\begin{aligned} &\text{minimize } m^f_{X^t}(X) \\ &\text{subject to } g(X) \geq c. \end{aligned}$$

This scheme is depicted in Algorithm 19.

Hence, each iteration is an instance of SSC and can be solved nearly optimally using the greedy algorithm. We can continue this algorithm for T iterations or until convergence. An analysis very similar to the ones in Chapters 3, 4 and 5 will reveal polynomial time convergence. Since each iteration is only the greedy algorithm, this approach is also highly practical and

Algorithm 19 Iterated Submodular Set Cover Algorithm for SCSC (Problem 4)

Start with an arbitrary X^0.

while *until convergence* $(X^{i-1} = X^i)$ **do**

Pick a supergradient and modular upper bound $m^f_{X^t}$ at X^t

$X^{t+1} := \text{argmin}\{m^f_{X^t}(X) \mid g(X) \geq c\}$

$t \leftarrow t + 1$

scalable. Since there are two approaches to solve the set cover problem (the primal approach of Lemma 6.1 and the dual greedy approach of Lemma 6.2), we have two forms of ISSC, the *primal ISSC* and the *dual ISSC*. The following shows the resulting theoretical guarantees:

Theorem 6.3. *The primal ISSC algorithm obtains an approximation factor of* $\frac{K_g H_g}{1+(K_g-1)(1-\hat{\kappa_f}(S^*))} \leq \frac{n}{1+(n-1)(1-\hat{\kappa_f}(S^*))} H_g$ *where* $K_g = 1 + \max\{|X| : g(X) < c\}$ *and* H_g *is the approximation factor of the submodular set cover using* g*. Similarly the dual ISSC obtains a bi-criterion guarantee of* $\left[\frac{(1+\epsilon)K_g}{1+(K_g-1)(1-\kappa_f)}, 1 - e^{-1}\right]$.

Proof. The first part of the result follows directly from Lemma 3.10. In particular, we obtain a guarantee of

$$\frac{\gamma|X^*|}{1 + (|X^*| - 1)(1 - \hat{\kappa_f}(S^*))} \tag{6.7}$$

for the problem of $\min\{f(X)|X \in \mathcal{C}\}$ where γ is the approximation guarantee of solving a modular function over $\mathcal{C}$ where $\mathcal{C}$ is the feasible set. In this case, $\gamma = H_g$ and $|X^*| \leq K_g$.

When using the dual greedy approach at every iteration, we can use a similar form of the result in the bi-criterion sense. Consider only the first iteration of this algorithm (due to the monotonicity of the algorithm, we will only improve the objective value). We are then guaranteed to obtain a set $\hat{X}$ such that (denote X^1 as the solution after the first iteration)

$$f(\hat{X}) \leq f(X^1) \leq m^f_\emptyset(X^1) \leq (1+\epsilon) m^f_\emptyset(X^*) \leq \frac{K_g(1+\epsilon)}{1 + (K_g - 1)(1 - \kappa_f)} f(X^*) \tag{6.8}$$

The inequalities above follow from the fact that the modular upper bound $m^f_\emptyset(X)$ satisfies,

$$m^f_\emptyset(X) \leq f(X) \leq \frac{|X|}{1 + (|X| - 1)(1 - \kappa_f)} f(X) \tag{6.9}$$

and the fact that X^1 which is the solution obtained through the dual set cover with a cover constraint $(1 - e^{-1})c$ satisfies $m^f_\emptyset(X^1) \leq (1 + \epsilon) m^f_\emptyset(X^*)$. In the above, X^* is the optimal solution to the problem $\min\{f(X) | g(X) \geq c\}$. We could also run the dual set cover algorithm to obtain a feasible solution (i.e., a type-II guarantee). In this case, the guarantee would compete with the optimal solution satisfying a cover constraint of $c/(1 - e^{-1})$. □

From the above, it is clear that $K_g \leq n$. Notice also that H_g is essentially a log-factor. We also see an interesting effect of the curvature κ_f of f. When f is modular ($\kappa_f = 0$), we recover the approximation guarantee of the submodular set cover problem. Similarly, when f has restricted curvature, the guarantees can be much better. For example, using square-root over modular function $f(X) = \sum_{i=1}^k \sqrt{w_i(X)}$, which is common model used in applications [Iyer and Bilmes, 2012b, Lin, 2012, Jegelka and Bilmes, 2011b], the worst case guarantee is $H_g \sqrt{K_g}$ (see Corollary 3.3). Moreover, the approximation guarantee already holds after the first iteration, so additional iterations can only further improve the objective.

We remark here that a special case of ISSC, using only the first iteration (i.e., the simple modular upper bound of f) was considered in [Wan et al., 2010, Du et al., 2011]. Our algorithm not only possibly improves upon theirs, but our approximation guarantee is also more explicit than theirs. In particular, they show a guarantee of $\nu_f H_g$, where $\nu_f = \min\{\frac{\sum_{i \in X} f(i)}{f(X)} | g(X) = g(V)\}$. Since this factor ν_f itself involves an optimization problem, it is not clear how to efficiently compute this factor. Moreover given a submodular function f, it is also not evident how good this factor is. While our guarantee is an upper bound of $\nu_f H_g$, it is much more explicit in its dependence on the parameters of the problem. It can also be computed efficiently and has an intuitive significance related to the curvature of the function. Furthermore, our bound is also tight since with, for example, $f(X) = \min\{|X|, 1\}$, our exactly matches the bound of [Wan et al., 2010, Du et al., 2011]. Lastly our algorithm also potentially improves upon theirs thanks to its iterative nature.

For another variant of this algorithm, we can replace g with its greedy modular lower bound at every iteration. Then, rather than solving every iteration through the greedy algorithm, we can solve every iteration as a knapsack problem (minimizing a modular function over a modular lower bound constraint) [Kellerer et al., 2004], using say, a dynamic programming based approach. This could potentially improve over the greedy variant, but at a potentially higher computational cost.

6.5.3 Ellipsoidal Approximation based Submodular Set Cover (EASSC)

In this setting, we use the ellipsoidal approximation discussed in §6.3. We can compute the κ_f-normalized version of f (f^κ, see §6.3), and then compute its ellipsoidal approximation $\sqrt{w^{f^\kappa}}$. We then define the function $\hat{f}(X) = f^{\text{ea}}(X) = \kappa_f \sqrt{w^{f^\kappa}(X)} + (1 - \kappa_f) \sum_{j \in X} f(j)$ and use this as the surrogate function $\hat{f}$ for f. We choose $\hat{g}$ as g itself. The surrogate problem becomes:

$$\min \left\{ \kappa_f \sqrt{w^{f^\kappa}(X)} + (1 - \kappa_f) \sum_{j \in X} f(j) \;\middle|\; g(X) \geq c \right\}. \tag{6.10}$$

While function $\hat{f}(X) = f^{\text{ea}}(X)$ is not modular, it is a weighted sum of a concave over modular function and a modular function. Fortunately, we can use the result from [Nikolova, 2010], where they show that any function of the form of $\sqrt{w_1(X)} + w_2(X)$ can be optimized over any polytope $\mathcal{P}$ with an approximation factor of $\gamma(1 + \epsilon)$ for any $\epsilon > 0$, where γ is the approximation factor of optimizing a modular function over $\mathcal{P}$. The complexity of this algorithm is polynomial in n and $\frac{1}{\epsilon}$. The main idea of their proof is to reduce the problem of minimizing $\sqrt{w_1(X)} + w_2(X)$, into $\log n$ problems of minimizing a modular function over the polytope. We use their algorithm to minimize $f^{\text{ea}}(X)$ over the *submodular set cover* constraint and hence we call this algorithm EASSC. Again we have the two variants, *primal EASSC* and *dual EASSC*, which essentially use at every iteration the primal and dual forms of set cover.

Theorem 6.4. *The primal EASSC obtains a guarantee of* $O(\frac{\sqrt{n} \log n H_g}{1 + (\sqrt{n} \log n - 1)(1 - \kappa_f)})$*, where* H_g

is the approximation guarantee of the set cover problem. Moreover, the dual EASSC obtains a bi-criterion approximation of $\left[O(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n-1)(1-\kappa_f)}), 1-e^{-1}\right]$.

Proof. The idea of the proof is to use the result from [Nikolova, 2010] where they show that any function of the form $\lambda_1\sqrt{m_1(X)}+\lambda_2 m_2(X)$ where $\lambda_1 \geq 0, \lambda_2 \geq 0$ and m_1 and m_2 are positive modular functions has a FPTAS, provided a modular function can easily be optimized over $\mathcal{C}$. Note that our function is exactly of that form. Hence, $f^{\text{ea}}(X)$ can be approximately optimized over $\mathcal{C}$. It now remains to show that this translates into the approximation guarantee. From Lemma 3.2, we know that the Ellipsoidal Approximation f^{ea} satisfies $f^{\text{ea}}(X) \leq f(X) \leq \gamma(n) f^{\text{ea}}(X), \forall X$ where

$$\gamma(n) = O\left(\frac{\sqrt{n}\log n}{(\sqrt{n}\log n - 1)(1-\kappa_f)+1)}\right). \tag{6.11}$$

Then, if $\hat{X}$ is the $1+\epsilon$ approximately optimal solution for minimizing f^{ea} over $\{X : g(X) \geq c\}$, we have that:

$$f(\hat{X}) \leq \gamma(n) f^{\text{ea}}(\hat{X}) \leq H_g \gamma(n)(1+\epsilon) f^{\text{ea}}(X^*) \leq H_g \gamma(n)(1+\epsilon) f(X^*), \tag{6.12}$$

where X^* is the optimal solution. We can set ϵ to any constant, say 1, and we get the result.

The dual guarantee again follows in a very similar manner thanks to the guarantee for the dual SSC. □

If the function f has $\kappa_f = 1$, we can use a much simpler algorithm. In particular, since the ellipsoidal approximation is of the form of $f^{\text{ea}}(X) = \sqrt{w^f(X)}$, we can minimize $(f^{\text{ea}}(X))^2 = w^f(X)$ at every iteration, giving a surrogate problem of the form

$$\min\{w^f(X) | g(X) \geq c\}. \tag{6.13}$$

This is directly an instance of SSC, and in contrast to EASSC, we just need to solve SSC once. We call this algorithm EASSCc. This guarantee is tight up to log factors when $\kappa_f = 1$.

Corollary 6.1. *The primal EASSCc obtains an approximation guarantee of* $O(\sqrt{n}\log n\sqrt{H_g})$. *Similarly, the dual EASSCc obtains a bicriterion guarantee of* $[O(\sqrt{n}\log n), 1-e^{-1}]$.

Proof. Let $\hat{X}$ be a set such that,

$$w(\hat{X}) \geq H_g \min\{w(X)|g(X) \geq c\} \tag{6.14}$$

Then denote $a(n) = O(\sqrt{n}\log n)$.

$$f(\hat{X}) \leq a(n)\sqrt{w(\hat{X})} \leq \sqrt{H_g}a(n)\sqrt{w(X^*)} \leq \sqrt{H_g}a(n)f(X^*) \tag{6.15}$$

In the above, $X^* = \text{argmin}\{f(X)|g(X) \geq c\}$.

The result for the dual variant can also be similarly shown. □

6.6 Approximation Algorithms for SCSK

In this section, we describe our approximation algorithms for SCSK. We note the dual nature of the algorithms in this current section to those given in §6.5. We first investigate a special case, the submodular knapsack (SK), and then provide three algorithms, two of them (Gr and ISK) being practical with slightly weaker theoretical guarantee, and another one (EASK) which is not scalable but has the tightest guarantee.

6.6.1 Submodular Cost Knapsack (SK)

We start with a special case of SCSK (Problem 5), where f is a modular function and g is a submodular function. In this case, SCSK turns into the SK problem for which the greedy algorithm with partial enumeration provides a $1 - e^{-1}$ approximation [Sviridenko, 2004]. The greedy algorithm can be seen as an instance of Algorithm 13 with $\hat{g}$ being the modular lower bound of g and $\hat{f}$ being f, which is already modular. In particular, we then get back the framework of [Iyer et al., 2013b], where the authors show that choosing a permutation based on a greedy ordering, exactly analogous to Eqn. (6.5), provides the bounds. In particular, define:

$$\sigma(i) \in \text{argmax}\left\{ \frac{g(j|S^\sigma_{i-1})}{f(j)} \;\middle|\; j \notin S^\sigma_{i-1}, f(S^\sigma_{i-1} \cup \{j\}) \leq b \right\}, \tag{6.16}$$

where the remaining elements are chosen arbitrarily. A slight catch however is that for the analysis to work, [Sviridenko, 2004] needs to consider $\binom{n}{3}$ instances of such orderings (partial enumeration), chosen by fixing the first three elements in the permutation [Iyer et al., 2013b]. We can however just choose the simple greedy ordering in one stage, to get a slightly worse approximation factor of $1 - e^{-1/2}$ [Iyer et al., 2013b, Lin and Bilmes, 2010, Krause and Guestrin, 2005a]. This simple greedy algorithm is shown in Alg. 20.

Algorithm 20 Greedy Algorithm for $\max\{g(X) \mid \sum_{i \in X} f(i) \leq b\}$ [Wolsey, 1982]

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

$y = \text{argmax}_{v \in V \setminus Y_n} \frac{g(v|Y_n)}{f(v)}$;

$Y_{n+1} = Y_n \cup y$;

$n \leftarrow n + 1$;

until $g(Y_n) > B$;

return Y_{n-1}.

Lemma 6.3. *Choosing the surrogate function $\hat{f}$ as f and $\hat{g}$ as h^{σ} in Algorithm 13 yields a set X^g:*

$$\max\{\max_{i:f(i)\leq b} g(i), g(X^g)\} \geq 1/2(1-1/e)g(X^*). \tag{6.17}$$

Let σ^{ijk} be a permutation with i, j, k in the first three positions, and the remaining arrangement greedy. Running $O(n^3)$ restarts of one iteration of Algorithm 13 yields sets X_{ijk} with

$$\max_{i,j,k \in V} f(X_{ijk}) \geq (1-1/e)f(X^*). \tag{6.18}$$

6.6.2 Greedy (Gr) Algorithms

In this section, we extend the greedy algorithm to the general SCSK problem, with submodular f and g. Here we consider two variants of the greedy algorithm. The first variant, proceeds at every iteration i, by choosing an element $j \notin S_{i-1} : f(S^{\sigma}_{i-1} \cup \{j\}) \leq b$ which maximizes

$g(j|S_{i-1})$. In terms of Algorithm 13, this is analogous to choosing a permutation, σ such that:

$$\sigma(i) \in \text{argmax}\{g(j|S^{\sigma}_{i-1})|j \notin S^{\sigma}_{i-1}, f(S^{\sigma}_{i-1} \cup \{j\}) \leq b\}. \tag{6.19}$$

This algorithm is agnostic to the cost, and hence we call it the cost-agnostic greedy algorithm.

Algorithm 21 Cost Agnostic Greedy Algorithm for $\max\{g(X) \mid f(X) \leq b\}$

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

  $y = \text{argmax}_{v \in V \setminus Y_n} g(v|Y_n)$;

  $Y_{n+1} = Y_n \cup y$;

  $n \leftarrow n + 1$;

until $g(Y_n) > B$;

return Y_{n-1}.

Theorem 6.5. *The cost agnostic greedy algorithm for SCSK obtains an approx. factor of* $\frac{1}{\kappa_g}(1 - (\frac{K_f - \kappa_g}{K_f})^{k_f}) \geq \frac{1}{K_f}$, *where* $K_f = \max\{|X| : f(X) \leq b\}$ *and* $k_f = \min\{|X| : f(X) \leq b \;\&\; \forall j \in X, f(X \cup j) > b\}$.

Proof. The proof of this result follows directly from [Conforti and Cornuejols, 1984, Iyer et al., 2013b]. In particular, it holds for any down monotone constraint. It is easy to see that the constraint $\{f(X) \leq b\}$ is down-monotone when f is a monotone submodular function. □

In the worst case, $k_f = 1$ and $K_f = n$, in which case the guarantee is $1/n$. The bound above follows from a simple observation that the constraint $\{f(X) \leq b\}$ is down-monotone for a monotone function f. Note that, in this variant, we do not use any specific information about the cost function f, and hence, we call this the cost agnostic greedy algorithm. Moreover, the bound above holds for maximizing a submodular function g over any down monotone constraint [Conforti and Cornuejols, 1984].

The second variant of the greedy algorithm proceeds by considering both f and g. In particular, at every round, we choose an item $j \notin S_{i-1} : f(S^{\sigma}_{i-1} \cup \{j\}) \leq b$ which maximizes $\frac{g(j|S_{i-1})}{f(j|S_{i-1})}$. In terms of Algorithm 13, this is analogous to choosing a permutation, σ such that:

$$\sigma(i) \in \text{argmax}\{\frac{g(j|S_{i-1})}{f(j|S_{i-1})} | j \notin S^{\sigma}_{i-1}, f(S^{\sigma}_{i-1} \cup \{j\}) \leq b\}. \tag{6.20}$$

Since this variant of the greedy algorithm is sensitive to the cost function f, we call it the cost sensitive greedy algorithm. This variant is shown in Algorithm 22.

Algorithm 22 Cost Sensitive Greedy Algorithm for $\max\{g(X) \mid f(X) \leq b\}$

Start with $Y_0 = \emptyset, n \leftarrow 0$

repeat

  $y = \text{argmax}_{v \in V \setminus Y_n} \frac{g(v|Y_n)}{f(v|Y_n)}$;

  $Y_{n+1} = Y_n \cup y$;

  $n \leftarrow n + 1$;

until $g(Y_n) > B$;

return Y_{n-1}.

Theorem 6.6. *The cost sensitive greedy algorithm provides a worst case guarantee of* $1 - e^{-\frac{(1-\hat{\kappa_f}(S^*))g(S^G)}{B}}$*, where* S^G *is the greedy solution and* S^* *is the optimal solution.*

Proof. Denote $S_1, S_2, \cdots S_k$ as the chain of sets obtained by the greedy algorithm. Then $S^G = S_k$ is the solution from the greedy algorithm. For $i = 1, 2, \cdots, k$, we have that,

$$g(S^*) \leq g(S_{i-1}) + \sum_{j \in S^* \setminus S_{i-1}} g(j|S_{i-1}) \tag{6.21}$$

$$\leq f(S_{i-1}) + \sum_{j \in S^* \setminus S_{i-1}} \frac{g(s_i|S_{i-1})}{f(s_i|S_{i-1})} f(j|S_{i-1}) \tag{6.22}$$

where the last inequality follows from the fact that $\frac{g(j|S_{i-1})}{f(j|S_{i-1})} \leq \frac{g(s_i|S_{i-1})}{f(s_i|S_{i-1})}$, from the cost sensitive

greedy algorithm. From the definition of the curvature, it follows that,

$$\sum_{j\in S^*\setminus S_{i-1}} f(j|S_{i-1}) \leq \sum_{j\in S^*} f(j) \tag{6.23}$$

$$\leq frac1(1-\hat{\kappa_f}(S^*)) \sum_{j\in S^*} f(j|S^*\backslash j) \tag{6.24}$$

The above holds following the definition of the curvature. Hence, we have that,

$$g(S^*) - g(S_{i-1}) \leq f(S_{i-1}) + \frac{g(s_i|S_{i-1})}{f(s_i|S_{i-1})} \frac{1}{(1-\hat{\kappa_f}(S^*))} \sum_{j\in S^*} f(j|S^*\backslash j) \tag{6.25}$$

$$\leq \frac{g(s_i|S_{i-1})}{f(s_i|S_{i-1})} \frac{1}{(1-\hat{\kappa_f}(S^*))} f(S^*) \tag{6.26}$$

where the last inequality follows since $f(S^*) \geq \sum_{j\in S^*} f(j|S^*\backslash j)$. Moreover, since $f(S^*) \leq B$, we have that,

$$g(S^*) - g(S_{i-1}) \leq \frac{g(S_i) - g(S_{i-1})}{f(S_i) - f(S_{i-1})} \frac{1}{(1-\hat{\kappa_f}(S^*))} B \tag{6.27}$$

Denote $\theta_i = g(S^*) - g(S_i)$. Also, denote $\beta_i = f(S_i) - f(S_{i-1})$. We then obtain the following recursion,

$$\theta_{i-1} \leq \frac{\theta_{i-1} - \theta_i}{\beta_i} \frac{1}{(1-\hat{\kappa_f}(S^*))} B \tag{6.28}$$

which implies that,

$$\theta_i \leq \theta_{i-1}(1 - \frac{(1-\hat{\kappa_f}(S^*))\beta_i}{B}) \tag{6.29}$$

Solving this inequality, yields,

$$f(S^*) - f(S_i) \leq \prod_{i=1}^{k} (1 - \frac{(1-\hat{\kappa_f}(S^*))\beta_i}{B}) f(S^*) \tag{6.30}$$

Finally, using $1 - x \leq e^{-x}$ and the fact that $\sum_i \beta_i = f(S_i)$, we obtain $f(S_i) \geq (1 - e^{-\frac{f(S^G)(1-\hat{\kappa_f}(S^*))}{B}})$. □

The above result depends on the curvature $\hat{\kappa_f}$ of f at the optimal set S^*. If f is modular, (i.e $\hat{\kappa_f}(S^*) = 0$. we obtain a worst case guarantee of $(1 - e^{-\frac{f(S^G)}{B}})$. When $\hat{\kappa_f}(S^*) = 0$, this is

unbounded. However, combining the cost sensitive greedy and the cost agnostic version (by taking the best solution of both), we can obtain bounded approximation guarantees. While the cost agnostic greedy has bounded worst case guarantee, the cost sensitive greedy performs much better on real world applications. One reason for this is that the submodular function f may not have a worst case curvature for several of the classes of submodular function used in applications.

6.6.3 Iterated Submodular Cost Knapsack (ISK)

Here, we choose $\hat{f}_t(X)$ as a modular upper bound of f, tight at X^t. Let $\hat{g}_t = g$. Then at every iteration, we solve:

$$\max\{g(X)|m^f_{X^t}(X) \leq b\}, \tag{6.31}$$

which is a submodular maximization problem subject to a knapsack constraint (SK). As mentioned above, greedy can solve this nearly optimally. We start with $X^0 = \emptyset$, choose $\hat{f}_0(X) = \sum_{j \in X} f(j)$ and then iteratively continue this process until convergence (note that this is an ascent algorithm). This scheme is shown in Algorithm 23.

Algorithm 23 Iterated Submodular Knapsack Algorithm for SCSK (Problem 5)

Start with an arbitrary X^0.

while *until convergence* $(X^{i-1} = X^i)$ **do**

- Pick a supergradient and modular upper bound $m^f_{X^t}$ at X^t
- $X^{t+1} := \text{argmax}\{g(X) \ \mid \ m^f_{X^t}(X) \leq b\}$
- $t \leftarrow t + 1$

We have the following theoretical guarantee:

Theorem 6.7. *Algorithm ISK obtains a set X^t such that $g(X^t) \geq (1 - e^{-1})g(\tilde{X})$, where $\tilde{X}$ is the optimal solution of* $\max\left\{g(X) \mid f(X) \leq \frac{b(1+(K_f-1)(1-\kappa_f)}{K_f}\right\}$ *and where* $K_f = \max\{|X| : f(X) \leq b\}$.

Proof. We are given that $\tilde{X}$ is the optimal solution to the problem:

$$\max\left\{g(X) \mid f(X) \leq \frac{b(1+(K_f-1)(1-\kappa_f)}{K_f}\right\} \tag{6.32}$$

$\tilde{X}$ is also a feasible solution to the problem:

$$\max\left\{g(X) \mid \sum_{j\in X} f(j) \leq b\right\} \tag{6.33}$$

The reason for this is that:

$$\sum_{j\in X} f(j) \leq \frac{K_f}{1+(K_f-1)(1-\kappa_f)} f(X) \leq \frac{K_f}{1+(K_f-1)(1-\kappa_f)} \frac{b(1+(K_f-1)(1-\kappa_f))}{K_f} \leq b$$

Now, at the first iteration we are guaranteed to find a set X^1 such that $g(X^1) \geq (1-1/e)g(\tilde{X})$. The further iterations will only improve the objective since this is a ascent algorithm. □

It is worth pointing out that the above bound holds even after the first iteration of the algorithm. It is interesting to note the similarity between this approach and the iterated submodular set cover ISSC. Notice that the bound above is a form of a bicriterion approximation factor of type-II. In particular, we obtain a feasible solution at the end of the algorithm. We can obtain a type-I bicriterion approximation bound, by running this for a larger budget constraint. In particular, we run the above algorithm with a budget constraint of $\frac{bK_f}{1+(K_f-1)(1-\kappa_f)}$ instead of b. The following guarantee then follows.

Lemma 6.4. *The ISK algorithm of type-I is guaranteed to obtain a set X which has a bicriterion approximation factor of $[1-e^{-1}, \frac{K_f}{1+(K_f-1)(1-\kappa_f)}]$, where $K_f = \max\{|X| : f(X) \leq b\}$.*

Proof. This result directly follows from the Lemma above, and the transformation from a type-II approximation algorithm to a type-I one. □

6.6.4 Ellipsoidal Approximation based Submodular Cost Knapsack (EASK)

Choosing the Ellipsoidal Approximation f^{ea} of f as a surrogate function, we obtain a simpler problem:

$$\max\left\{g(X) \;\middle|\; \kappa_f\sqrt{w^{f^\kappa}(X)} + (1-\kappa_f)\sum_{j\in X} f(j) \leq b\right\}. \tag{6.34}$$

In order to solve this problem, we look at its dual problem (i.e., Eqn. (6.10)) and use Algorithm 14 to convert the guarantees. Recall that Eqn. (6.10) itself admits two variants of the ellipsoidal approximation, which we had referred to as the primal EASSC and the dual EASSC. Hence, we call the combined algorithms, primal and dual EASK, which admit the following guarantees:

Lemma 6.5. *The primal EASK obtains a guarantee of* $\left[1+\epsilon, O(\frac{\sqrt{n}\log nH_g}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})\right]$. *Similarly, the dual EASK obtains a guarantee of* $\left[(1+\epsilon)(1-1/e), O(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})\right]$.

Proof. This Lemma directly follows from Theorem 6.1 and Theorem 6.4. □

These factors are akin to those of Theorem 6.4, except for the additional factor of $1+\epsilon$. Unlike the greedy algorithm, ISSC and ISK, note that both EASK and EASSC are enormously costly and complicated algorithms. Hence, it also seems at first thought that EASK would need to run multiple versions of EASSC at each conversion round of Algorithm 14. Fortunately, however, we need to compute the Ellipsoidal Approximation just once and the algorithm can reuse it for different values of c. Furthermore, since construction of the approximation is often the bottleneck, this scheme is likely to be as costly as EASSC in practice.

In the case when the submodular function has a curvature $\kappa_f = 1$, we can actually provide a simpler algorithm without needing to use the conversion algorithm (Algorithm 14). In this case, we can directly choose the ellipsoidal approximation of f as $\sqrt{w^f(X)}$ and solve the surrogate problem:

$$\max\{g(X) : w^f(X) \leq b^2\}. \tag{6.35}$$

This surrogate problem is a submodular cost knapsack problem, which we can solve using the greedy algorithm. We call this algorithm EASKc. This guarantee is tight up to log factors if $\kappa_f = 1$.

Corollary 6.2. *Algorithm EASKc obtains a bi-criterion guarantee of* $[1-e^{-1}, O(\sqrt{n}\log n)]$.

Proof. Let $\hat{X}$ be the approximate optimizer of Eqn. (6.35). First we show that $g(\hat{X}) \geq (1-1/e)g(X^*)$ where X^* is the optimizer of problem 5. Also notice that $f(X^*) \leq b$ since it feasible. Also, $\sqrt{w^f(X^*)} \leq f(X^*) \leq b$ and hence X^* is feasible in Eqn. (6.35). Hence it holds that $g(\hat{X}) \geq (1-1/e)g(X^*)$.

We now show that $f(\hat{X}) \leq b\sqrt{n}\log n$. Notice that,

$$f(\hat{X}) \leq \sqrt{n}\log n\sqrt{w^f(\hat{X})} \leq b\sqrt{n}\log n \tag{6.36}$$

□

6.7 Extensions beyond SCSC and SCSK

SCSC is in fact more general and can be extended to more flexible and complicated constraints which can arise naturally in many applications [Krause et al., 2008a, Guillory and Bilmes, 2011]. Notice first that

$$\{g(X) \geq c\} \Leftrightarrow \{g'(X) = g'(V)\} \tag{6.37}$$

where $g'(X) = \min\{g(X), c\}$. We can also have "and" constraints as $g_1(X) = g_1(V)$ and $g_2(X) = g_2(V)$. These have a simple equivalence:

$$\{g_1(X) = g_1(V) \wedge g_2(X) = g_2(V)\} \Leftrightarrow \{g(X) = g(V)\} \tag{6.38}$$

when $g(X) = g_1(X) + g_2(X)$ [Krause et al., 2008a]. Moreover, we can also handle k 'and' constraints, by defining $g(X) = \sum_{i=1}^{k} g_i(X)$.

Similarly we can have "or" constraints, i.e., $g_1(X) = g_1(V)$ or $g_2(X) = g_2(V)$. These also have a nice equivalence:

$$\{g_1(X) = g_1(V) \vee g_2(X) = g_2(V)\} \Leftrightarrow \{g(X) = g(V)\} \tag{6.39}$$

by defining $g(X) = g_1(X)g_2(V) + g_2(X)g_1(V) - g_1(X)g_2(X)$ [Guillory and Bilmes, 2011]. We can also extend these recursively to multiple 'or' constraints. Hence our algorithms can directly solve all these variants of SCSC.

SCSK can also be extended to handle more complicated forms of functions g. In particular, consider the function

$$g(X) = \min\{g_1(X), g_2(X), \ldots, g_k(X)\} \tag{6.40}$$

where the functions $g_1, g_2, \ldots, g_k$ are submodular. Although $g(X)$ in this case is not submodular, this scenario occurs naturally in many applications, particularly sensor placement [Krause et al., 2008a]. The problem in [Krause et al., 2008a] is in fact a special case of Problem 5, using a modular function f. Often, however, the budget functions involve a cooperative cost, in which case f is submodular. Using Algorithm 14, however, we can easily solve this by iteratively solving the dual problem. Notice that the dual problem is in the form of Problem 4 with a non-submodular constraint $g(X) \geq c$. It is easy to see that this is equivalent to the constraint $g_i(X) \geq c, \forall i = 1, 2, \ldots, k$, which can be solved thanks to the techniques above.

We can also handle multiple constraints in SCSK. In particular, consider multiple 'and' constraints – $\{f_i(X) \leq b_i, i = 1, 2, \cdots, k\}$, for monotone submodular functions f_i and g. A first observation is that the greedy algorithm can almost directly extended to these cases, since we do not use any specific property of the constraints, while providing the approximation guarantees. Hence Theorem 6.5 can directly be extended to this case. We can also provide bi-criterion approximation guarantees with 'and' constraints. The approximate feasibility for a $[\beta, \alpha]$ bi-criterion approximation in this setting would be to have a set X such that $f_i(X) \leq \alpha b_i$. Algorithm ISK can easily then be used in this scenario, and at every iteration we would solve a monotone submodular maximization problem subject to multiple linear (or knapsack constraints). Surprisingly this problem also has a constant factor $(1 - 1/e)$ approximation guarantee [Kulik et al., 2009]. Hence we can retain the same approximation guarantees as in Theorem 6.7. We can also use algorithm EASKc, and obtain a curvature-independent approximation bound for this problem. The reason for this is the Ellipsoidal Approximation is of the form $\sqrt{w^{f_i}(X)}$, for each i and squaring it will lead to knapsack constraints. If we add the curvature terms (i.e., try to implement EASK in this setting), we obtain a much more complicated class of constraints, which we do not currently know how to

handle.

We can also extend SCSC and SCSK to non-monotone submodular functions. In particular, recall that the submodular knapsack has constant factor approximation guarantees even when g is non-monotone submodular [Feige et al., 2011a]. Hence, we can obtain bi-criterion approximation guarantees for the Submodular Set Cover (SSC) problem, by solving the Submodular Knapsack (SK) problem multiple times. We can similarly do SCSC and SCSK when f is monotone submodular and g is non-monotone submodular, by extending ISSC, EASSC, ISK and EASK (note that in all these cases, we need to solve a submodular set cover or submodular knapsack problem with a non-monotone g). These algorithms, however, do not extend if f is non-monotone, and we do not currently know how to implement these.

6.8 Hardness of SCSC and SCSK

In this section, we provide the hardness for Problems 4 and 5. The lower bounds serve to show that the approximation factors above are almost tight.

Theorem 6.8. *For any $\kappa > 0$, there exists submodular functions with curvature κ such that no polynomial time algorithm for Problems 4 and 5 achieves a bi-criterion factor better than $\frac{\alpha}{\beta} = \frac{n^{1/2-\epsilon}}{1+(n^{1/2-\epsilon}-1)(1-\kappa)}$ for any $\epsilon > 0$.*

Proof. We prove this result using the hardness construction from [Goemans et al., 2009, Svitkina and Fleischer, 2008]. The main idea of their proof technique is to construct two submodular functions $f(X)$ and $f_R(X)$ that with high probability are indistinguishable. Thus, also with high probability, no algorithm can distinguish between the two functions and the gap in their values provides a lower bound on the approximation. We shall see that this lower bound in fact matches the approximation factors up to log factors and hence this is the hardness of the problem.

Define two monotone submodular functions $f(X) = \kappa_f \min\{|X|, \eta\} + (1 - \kappa_f)|X|$ and $f_R(X) = \kappa_f \min\{\gamma + |X \cap \bar{R}|, |X|, \eta\} + (1-\kappa_f)|X|$, where $R \subseteq V$ is a random set of cardinality η. Let η and γ be an integer such that $\eta = x\sqrt{n}/5$ and $\gamma = x^2/5$ for an $x^2 = \omega(\log n)$. Both

f and f_R have curvature κ_f. We also assume a very simple function $g(X) = |X|$. Given an arbitrary $\epsilon > 0$, set $x^2 = n^{2\epsilon} = \omega(\log n)$. Then the ratio between $f_R^{\kappa_f}(R)$ and $g^{\kappa_f}(R)$ is $\frac{n^{1/2-\epsilon}}{1+(n^{1/2-\epsilon}-1)(1-\kappa_f)}$. A Chernoff bound analysis very similar to [Svitkina and Fleischer, 2008] reveals that any algorithm that uses a polynomial number of queries can distinguish h^κ and f_R^κ with probability only $n^{-\omega(1)}$, and therefore cannot reliably distinguish the functions with a polynomial number of queries.

We first prove the first part of this theorem (i.e., for SCSC). In this case, we consider the problem

$$\min\{h(X)||X| \geq \eta\} \tag{6.41}$$

with h chosen as f and f_R respectively. It is easy to see that R is the optimal set in both cases. However the ratio between the two is

$$\gamma(n) = \frac{n^{1/2-\epsilon}}{1 + (n^{1/2-\epsilon} - 1)(1 - \kappa_f)} \tag{6.42}$$

Now suppose there exists an algorithm which is guaranteed to obtain an approximation factor better than $\gamma(n)$. Then by running this algorithm, we are guaranteed to find different answers for Eqn (6.41) above. This must imply that this algorithm can distinguish between f_R and g which is a contradiction. Hence no algorithm for Problem 4 can obtain an approximation guarantee $\frac{n^{1/2-\epsilon}}{1+(n^{1/2-\epsilon}-1)(1-\kappa_f)}$.

In order to extend the result to the bi-criterion case, we need to show that no bi-criterion approximation algorithm can obtain a factor better than $\frac{\alpha}{\beta} = \gamma(n)$. Assume there exists a bi-criterion algorithm with a factor $[\alpha, \beta]$ such that:

$$\frac{\alpha}{\beta} < \gamma(n) = \frac{n^{1/2-\epsilon}}{1 + (n^{1/2-\epsilon} - 1)(1 - \kappa_f)} \tag{6.43}$$

Then, we are guaranteed to obtain a set S in equation (6.41) such that $h(S) \leq \alpha OPT$ and $|S| \geq \beta\eta$, where $\alpha \geq 1$ and $\beta \leq 1$. Now we run this algorithm with $h = f$ and $h = f_R$ respectively. With $h = f$, it is easy to see that the algorithm obtains a set S_1 such that $f(S_1) \leq \alpha\eta$ and $|S_1| \geq \beta\eta$. Similarly, with $h = f_R$, the algorithm finds a set S_2 such that $f(S_2) \leq \alpha(\kappa\gamma + (1 - \kappa)\eta)$ and $|S_2| \geq \beta\eta$. Since f_R and f are indistinguishable, $S_1 = S_2 = S$.

We first assume that $|S| < \eta$. Then $f(S) = |S| \geq \beta\eta$. We then have that,

$$f_R(S) \leq \alpha(\kappa\gamma + (1-\kappa)\eta \tag{6.44}$$

$$\leq \alpha n^{2\epsilon}(1 + (n^{1/2-\epsilon} - 1)(1 - \kappa_f) \tag{6.45}$$

$$< \beta\gamma(n)n^{2\epsilon}(1 + (n^{1/2-\epsilon} - 1)(1 - \kappa_f) \tag{6.46}$$

$$< \beta n^{1/2+\epsilon} \tag{6.47}$$

$$< \beta\eta \tag{6.48}$$

This implies that the algorithm can distinguish between f_R and f which is a contradiction. Now consider the second case when $|S| \geq \eta$. In this case, $f(S) = \eta$. Again the chain of inequalities above shows that $f_R(S) < \beta\eta < \eta$ since $\beta \leq 1$. This again implies that the algorithm can distinguish between f_R and f which is a contradiction. Hence no bi-criterion approximation algorithm can obtain a factor better than $\gamma(n)$.

To show the second part (for SCSK), we can simply invoke Theorem 6.1 and argue that any $[\beta, \alpha]$ bi-criterion approximation algorithm for Problem 5 with

$$\frac{\alpha}{\beta} < \frac{\gamma(n)}{1+\epsilon} \tag{6.49}$$

can be used in algorithm 3, to provide a $[\alpha, \beta]$ bi-criterion algorithm with $\frac{\alpha}{\beta} < \gamma(n)$. This contradicts the first part above and hence the same hardness applies to Problem 5 as well. □

The above result shows that EASSC and EASK meet the bounds above to log factors. We see an interesting curvature-dependent influence on the hardness. We also see from our approximation guarantees also that the curvature of f plays a more influential role than the curvature of g on the approximation quality. In particular, as soon as f becomes modular, the problem becomes easy, even when g is submodular. This is not surprising since the submodular set cover problem and the submodular cost knapsack problem both have constant factor guarantees. In particular, we know from the results from Chapter 4, that we can obtain constant factor approximation bounds when f is modular (depending on the curvature of g). We summarize the results in Table 6.2.

6.9 Experiments

In this section, we compare the performance of the various algorithms proposed in this chapter on two real world problems, namely speech data subset selection, and sensor placement.

6.9.1 Limited Complexity Speech Data Subset Selection

Here we consider the speech data subset selection application [Lin and Bilmes, 2009, 2011a, Jegelka et al., 2011, Wei et al., 2014b] with the submodular function f encouraging limited complexity while g tries to achieve acoustic variability (or coverage).

In this scenario, we test two different functions f (one being submodular, and another modular). As a modular function, we define $f_1(X) = |X|$, which represents the simplest notion of complexity of the corpus, viz. the number of utterances. We also define a submodular complexity parameter, as the size of the vocabulary induced by the set of utterances. This function is of the form $f_2(X) = |\Gamma(X)|$, where $\Gamma(X)$ is the neighborhood function on a bipartite graph constructed between the utterances and the words [Lin and Bilmes, 2011a].

For the coverage function g, we use two types of coverage functions. One is a facility location function

$$g_1(X) = \sum_{i \in V} \max_{j \in X} s_{ij} \tag{6.50}$$

while the other one is a saturated sum function

$$g_2(X) = \sum_{i \in V} \min\{\sum_{j \in X} s_{ij}, \eta \sum_{j \in V} s_{ij}\}. \tag{6.51}$$

Both these functions are defined in terms of a similarity matrix $\mathbf{S} = \{s_{ij}\}_{i,j \in V}$, using the string kernel metric [Rousu and Shawe-Taylor, 2006] for similarity.

Submodular Coverage and Modular Complexity: In this setting, we choose the modular complexity function $f(X) = |X|$ (i.e the size of the corpus), and the submodular coverage function as the Facility Location function and the Saturated Coverage function.

We compare the greedy algorithm for the submodular coverage and submodular knapsack. This problem then becomes a simple speech data subset selection problem [Wei et al., 2014b]. Perhaps unsurprisingly, both algorithms perform comparable, and are much better than a random subset baseline (which just randomly selects items of different sizes) – the results are shown in the top panel of Figure 6.3.

Submodular Coverage and Submodular Complexity We next consider the g being a submodular coverage function g_1 and g_2, and f being the submodular complexity function f_2. This problem then becomes a limited vocabulary speech corpus selection with acoustic diversity [Lin and Bilmes, 2011a, Jegelka et al., 2011]. We compare all our different algorithms, (ISSC, ISK, EASSC, EASSK and Gr (S and Ag)) for this setting. Again, we vary the coverage and budget parameters in these algorithms through the range of values of f and g. Furthermore, in our experiments, we observe that the neighborhood function f has a curvature $\kappa_f = 1$. Thus, it suffices to use the simpler versions of algorithm EA (i.e., algorithm EASSCc and EASKc). The results are shown in the bottom panel of Figure 6.3. Since EA is computationally very expensive, we restrict ourselves to $n = 50$ utterances. We observe for this problem, that EASSC, ISSC, ISK and Gr-S perform comparably. This implies, moreover, that the iterative variants, viz. Gr, ISSC and ISK, perform comparably to the more complicated EA-based ones, although EASSC and EASK have better theoretical guarantees. Finally, as expected, the cost agnostic greedy (Gr-Ag) performs much worse than these algorithms, and in fact, is comparable to a simple baseline of choosing random subsets.

6.9.2 Sensor Placement

We next consider an application of sensor placement. The problem here is to maximize the sensor coverage g, while simultaneously minimizing the cost of the sensors f. A number of natural models for sensor coverage are are submodular[Krause and Guestrin, 2009, Krause et al., 2008b]. We consider three different submodular functions as models for sensor coverage. The first one is the symmetric mutual information function, $g_1(A) = I(\mathcal{X}_A; \mathcal{X}_{V \setminus A})$, where A

refers to the set of sensors chosen from $\mathcal{X}$ [Krause and Guestrin, 2009, Krause et al., 2008b]. While this function is not monotone, it can be shown to be approximately monotone as long as the number of sensors is not too large [Krause et al., 2008b]. The second model g_2, is the average variance reduction function [Krause et al., 2008a], which is monotone submodular. Finally, we consider the information gain function $g_3(A) = I(\mathcal{X}_A; \mathcal{X}_U)$, where U are the set of target variables of interest [Krause et al., 2008b, Krause and Guestrin, 2005b]. While this function is not submodular in general, it can be shown to be submodular under certain assumptions (c.f. Chapter 7 in [Krause, 2008]). In particular, we assume that the the measurements from A are noisy versions of the true measurements, with a noise parameter, and they are conditionally independent given the true values [Krause, 2008]. Furthermore, all the above information theoretic quantities are defined on multivariate Gaussians.

We consider two models of cost functions f. One is a simple additive model $f(X) = w(X)$, where the cost of the set of sensors is the sum of the individual costs. Often however, the costs are not additive in practice, and a very natural model to consider is a submodular function [Krause et al., 2008b]:

$$f(X) = \sum_{i=1}^{k} \psi_i(c(X \cap S_i)) \tag{6.52}$$

where ψ_i's are concave, $c(j)$ is the cost of sensor j and $S_1, \cdots, S_k$ are groups of similar sensors. The motivation here is that there is typically a discount when purchasing sensors in bulk (economies of scale). Moreover, there may be a diminished cost for placing a sensor in a particular location given placement in certain other locations (e.g., the additional equipment needed to install a sensor in, say, a precarious environment could be re-used for multiple sensor installations in similar environments). We consider real world data of placing sensors to predict the pH values from the lake of Merced [Krause and Guestrin, 2009].

Submodular Coverage with Additive Costs: In this section, we consider the problem of maximizing the coverage functions g_1, g_2 and g_3, while minimizing an additive cost function. Again, we compare the greedy algorithm for submodular knapsack and submodular set cover,

and observe that both algorithms perform comparably, and much better than the random subset baseline. The results are shown in Figure 6.4.

Submodular Coverage with Submodular Costs: Finally, we compare our algorithms in the case when we have submodular coverage and submodular cost functions. Again, we vary the coverage and budget parameters in these algorithms through the range of values of f and g. The results are shown in the bottom panel of Figure 6.4. Similar to the speech data subset selection application, we observe, that EASSC, ISSC, ISK and Gr-S perform comparably, even though the iterative algorithms ISK, ISSC and Gr-S are much more practical computationally. Moreover, we also see that these three algorithms generally perform much better than the random subset baseline, and often, better than the cost agnostic greedy algorithm.

6.9.3 Discussion on Running times

We investigated the algorithms proposed in this chapter, on two real world experiments. The Ellipsoidal Approximation based techniques (EASSC and EASK), though the tightest algorithms theoretically, are much more computationally expensive compared to the iterative procedures (ISSC, ISK and Gr). For example, computing the Ellipsoidal Approximation (which is the most computationally expensive part) for the Bipartite Neighborhood function with $|V| = 50$ takes about 5 hours while all the iterative variants (i.e., Gr, ISSC and ISK) take less than a second. This difference is much more prominent on larger instances (for example $|V| = 100$). The results above demonstrate that empirically, however, the iterative algorithms perform comparably to the Ellipsoidal Approximation.

6.10 Discussion

In this chapter, we propose a unifying framework for Problems 4 and 5 based on suitable surrogate functions. We provide a number of iterative algorithms which are very practical and scalable (like Gr, ISK and ISSC), and also algorithms like EASSC and EASK, which though more intensive, obtain tight approximation bounds. Finally, we empirically compare

our algorithms, and show that the iterative algorithms compete empirically with the more complicated and theoretically better approximation algorithms.

To our knowledge, this chapter provides the first general framework of approximation algorithms for Problems 4 and 5 for monotone submodular functions f and g. A number of papers, however, investigate related problems and approaches. For example, [Fujishige, 2005] investigates an exact algorithm for solving problem 4, with equality instead of inequality. However, since problem 4 subsumes the problem of minimizing a monotone submodular function subject to a cardinality equality constraint, and is hence NP hard [Nagano et al., 2011]. Hence this algorithm in the worst case, must be exponential. Furthermore, a similar problem was considered in [Krause et al., 2006] with one specific instance of a function f, which is not submodular. They also use, a considerably different algorithm. Also, an algorithm equivalent to the first iteration of ISSC was proposed in [Wan et al., 2010, Du et al., 2011] and ISSC not only generalizes this, but we also provide a more explicit approximation guarantee (we provide an elaborate discussion on this in the section describing ISSC). We also point out that, a special case of SCSK was considered in [Lin, 2012], with f being submodular, and g modular (we called this the *submodular span problem*). The authors there use an algorithm very similar to Algorithm 14, to convert this problem into an instance of minimizing a submodular function subject to a knapsack constraint, for which they use the algorithm of [Svitkina and Fleischer, 2008]. Unfortunately, the algorithm of [Svitkina and Fleischer, 2008] does not scale very well. Our algorithms for this problem, on the other hand, would continue to scale very well in practice.

A number of papers build upon the work we propose in this thesis. For example, [Kusner, 2014] consider versions of SCSC and SCSK with f being adaptive submodular, and show approximation guarantees based on the curvature of f and g. Similarly, [Iyer and Bilmes, 2014a] propose an algorithm for SCSC and SCSK, with f being low rank sums of concave over modular functions. [Xiong et al., 2014] propose an algorithm for mobile crowdsensing, which involves a special case of Problem 5. They use SCSK, and observe that the algorithm converges pretty quickly, and works well empirically. Similarly, **?** investigate connections

between SCSK and the sparse group cover problem, and propose an algorithm very similar to the cost-sensitive greedy algorithm proposed here. Finally, **?** investigate the problem of limited vocabulary speech corpus selection (similar to the ones we consider in the experimental section), and use SCSC and SCSK for corpus selection on real world speech corpora like Switchboard and Fisher. They also demonstrate how SCSK scales to the large scale Fisher dataset, which contains about a million utterances.

We would also like to contrast the results from this chapter, to those we show for Problem 3. While the algorithms proposed in Chapter 5 are scalable and practical, and moreover strikingly similar to the algorithms we propose here, the problem itself is very hard. Moreover, these algorithms are not comparable to ours, since we directly model the hard constraints and our bi-criteria results give worst-case bounds on the deviation from the constraints and the optimal solution. This is often important, since there are hard constraints in many practical applications (in the form of power constraints, or budget constraints). Casting it as Problem 3, however, no longer has guarantees on the deviation from the constraints.

Problem	Algorithm	Approx. factor* Bi-Criterion factor$^\#$	Hardness* (Bi-Criterion $\frac{\alpha}{\beta}$)$^\#$
SSC	Primal Greedy	$H(\max_j f(j))^*$	$\log n^*$
	Dual Greedy	$(1+\epsilon, 1-\frac{1}{e})^\#$	$1-1/e^\#$
SCSC	Primal ISSC	$\frac{nH_g}{1+(n-1)(1-\kappa_f(S^*))}^*$	$\Omega(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})^{*,\#}$
	Dual ISSC	$[\frac{(1+\epsilon)n}{1+(n-1)(1-\kappa_f(S^*))}, 1-1/e]^\#$	
	Primal EASSC	$O(\frac{\sqrt{n}\log nH_g}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})^*$	
	Dual EASSC	$[O(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n-1)(1-\kappa_f)}, 1-1/e]^\#$	
	Primal EASSCc	$O(\sqrt{n}\log n\sqrt{H_g})^*$	
	Dual EASSCc	$[O(\sqrt{n}\log n), 1-1/e]^\#$	
SK	Greedy	$1-1/e^*$	$1-1/e^*$
SCSK	Greedy (CA)	$\frac{1}{\kappa_g}(1-(\frac{K_f-\kappa_g}{K_f})^{k_f})^*$	$\Omega(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})^{*,\#}$
	Greedy (CS)	$[1-e^{-\frac{(1-\hat{\kappa_f}(S^*))g(S^G)}{B}}]^*$	
	ISK	$[1-e^{-1}, \frac{K_f}{1+(K_f-1)(1-\kappa_f(S^*))}]^\#$	
	Primal EASK	$[1+\epsilon, O(\frac{\sqrt{n}\log nH_g}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})]^\#$	
	Dual EASK	$[(1-1/e)(1+\epsilon), O(\frac{\sqrt{n}\log n}{1+(\sqrt{n}\log n-1)(1-\kappa_f)})]^\#$	
	EASKc	$[1-1/e, O(\sqrt{n}\log n)]^\#$	

Table 6.1: Worst case approximation factors, hardness for Problems 4 and 5 and their special cases.

	Modular g	Submodular g	
	$(\kappa_g = 0)$	$(0 < \kappa_g < 1)$	$(\kappa_g = 1)$
Modular f $(\kappa_f = 0)$	FPTAS	$\frac{1}{\kappa_g}(1 - e^{-\kappa_g})$	$1 - 1/e$
Submod f $(0 < \kappa_f < 1)$	$\Omega(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})$	$\Omega(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})$	$\Omega(\frac{\sqrt{n}}{1+(\sqrt{n}-1)(1-\kappa_f)})$
Submod f $(\kappa_f = 1)$	$\Omega(\sqrt{n})$	$\Omega(\sqrt{n})$	$\Omega(\sqrt{n})$

Table 6.2: Summary of Hardness results for SCSC/ SCSK

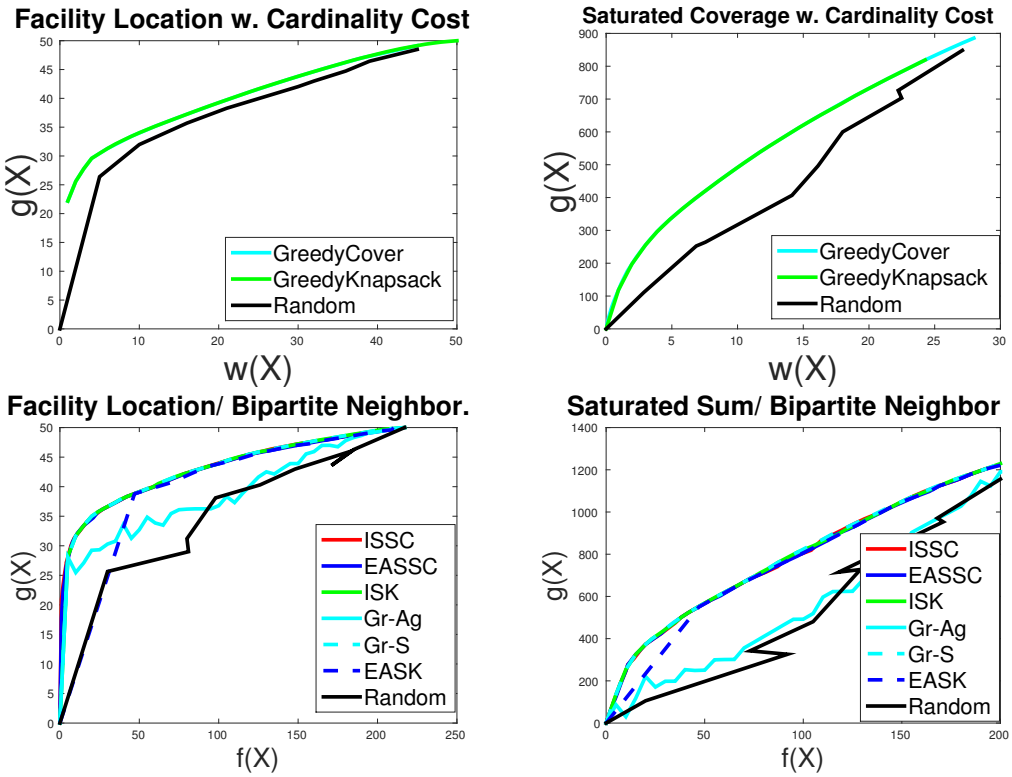

Figure 6.3: The top two figures show the performance of the greedy algorithms for submodular knapsack, and submodular cover respectively, with a modular f and submodular g (being Facility Location and Saturated Coverage respectively) – note that the green and cyan colored lines overlap for this example, since both algorithms perform exactly identically. The bottom two figures show the performance of the algorithms in the text for submodular functions f (vocabulary size) and submodular g (Facility Location and Saturated Coverage).

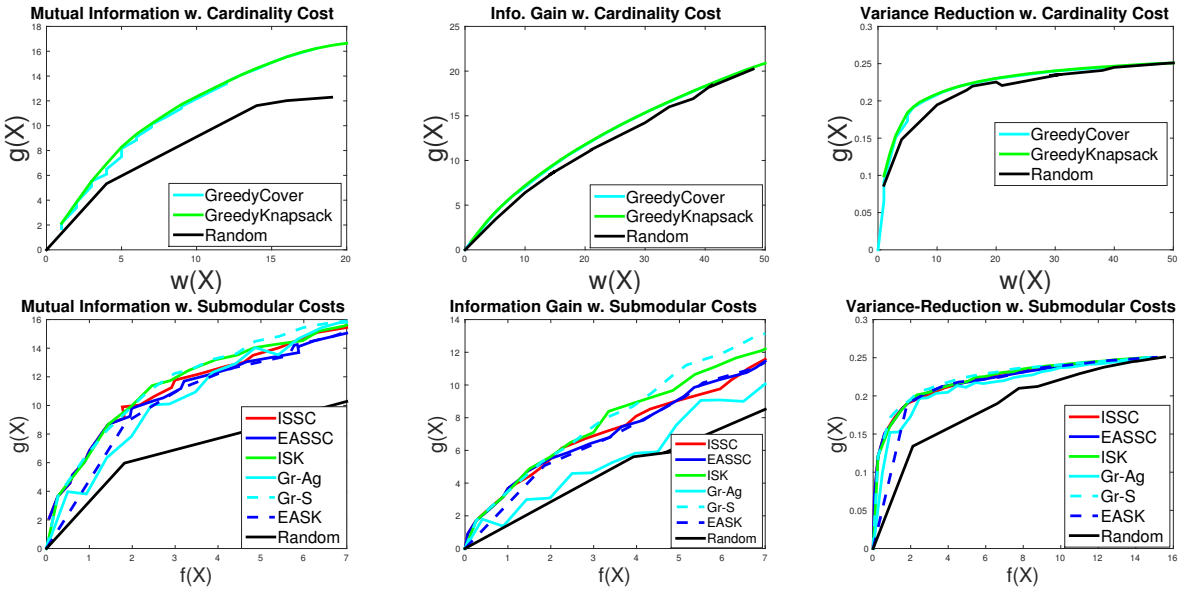

Figure 6.4: The top two figures show the performance of the greedy algorithms for submodular knapsack, and submodular cover respectively, with a additive cost f and submodular coverage g (being Mutual Information, Information Gain and Variance Reduction respectively). The bottom two figures show the performance of the algorithms in the text for submodular cost function f and submodular g.

Chapter 7

CONTINUOUS RELAXATION ALGORITHMS FOR SUBMODULAR OPTIMIZATION

7.1 Introduction

In Chapters 3, 4, 5, and 6, we provided two classes of combinatorial algorithms, one being tight but not practical (the Ellipsoidal Approximation algorithm), while the other, being an extremely scalable one based on the Majorization-Minimization framework. In this chapter, we investigate a complementary continuous relaxation based framework for these optimization problems. Similar to the combinatorial one, we show how this framework is unifying and scalable.

The continuous relaxation techniques known for submodular optimization, have so far either been analyzed for very specific constraints, or when general, are too *slow* to use in practice. For example, in the case of minimization, they were studied only for the specific constraints of covers [Iwata and Nagano, 2009] and cuts [Jegelka and Bilmes, 2011c], and in the case of maximization, the techniques though general have yet to show significant practical impact due to their prohibitive computational costs [Chekuri et al., 2011, Calinescu et al., 2011]. Hence discrete algorithms (for example, the class of Algorithms in Chapter 4) are typically used in applications (e.g., [Lin, 2012]). In this chapter, we develop a unifying continuous relaxation methodology that addresses these issues. We focus mainly on Problems 1 and 2, and show how we can extend these ideas to Problem 3 as well. We summarize our contributions, in comparison to previous work (in Table 7.1), which lists one problem as being still open, and other problems as being unnecessary (given a "fast" approach, the corresponding "slow" approach is unnecessary). Our techniques are not only connective, but also fast and scalable. In the case of constrained minimization, we provide a formulation

applicable for a large class of constraints. In the case of submodular maximization, we show how for a large class of submodular functions of practical interest, the generic slow algorithms can be made fast and scalable. We note, however, that it is still an open problem to provide a fast and scalable algorithmic framework (with theoretical guarantees) based on continuous relaxations for general submodular maximization.

Constraints or	Operation (& speed)	Algorithm Approach	
		Combinatorial	Relaxation
Function Specific	Min (fast)	[Goel et al., 2009, Jegelka and Bilmes, 2011c]	[Iwata and Nagano, 2009, Jegelka and Bilmes, 2011c]
	Min (slow)	[Svitkina and Fleischer, 2008]	Unnecessary
	Max (fast)	[Nemhauser et al., 1978, Buchbinder et al., 2012] etc.	**This thesis**
	Max (slow)	Unnecessary	[Buchbinder et al., 2014, Calinescu et al., 2011]
General	Min (fast)	[Iyer et al., 2013b]	**This thesis**
	Min (slow)	[Goemans et al., 2009]	Unnecessary
	Max (fast)	[Iyer et al., 2013b]	Open
	Max (slow)	Unnecessary	[Chekuri et al., 2011]

Table 7.1: Past work & our contributions (see text for explanation).

Our relaxation viewpoint, moreover, complements and improves on the bounds founds obtained through the combinatorial algorithms described in the previous chapters (viz. the Majorization-Minimization algorithm, and the Ellipsoidal Approximation algorithm). For example, where the MM algorithms may have an approximation bound of k, our results imply a bound of $n - k + 1$, where $n = |V|$, so considering both these, we obtain combined bounds of the form $\min(k, n - k + 1)$ (more specifics are given in Table 7.2). This also holds for maximization – in certain cases discrete algorithms obtain suboptimal results, while relaxation techniques obtain improved, and sometimes optimal guarantees.

7.2 Continuous Relaxation Framework

The idea of our relaxation strategy is as follows: the submodular function $f(S)$, which is defined on the vertices of the n-dimensional hypercube (i.e., characteristic vectors), is

extended to a function defined on $[0,1]^n$. The two functions valuate identically if the vector $x \in [0,1]^n$ is the characteristic vector of a set. We then solve a continuous optimization problem subject to linear constraints, and finally round the obtained fractional solution to a discrete one – i.e. our relaxation framework follows a two-stage procedure:

1. Find the optimal (or approximate) solution $\hat{x}$ to the problem $\min_{x \in \mathcal{P}_\mathcal{C}} \breve{f}(x)$ (or $\max_{x \in \mathcal{P}_\mathcal{C}} \tilde{f}(x)$).

2. Round the continuous solution $\hat{x}$ to obtain the discrete indicator vector of set $\hat{X}$.

Here, $\mathcal{P}_\mathcal{C}$ denotes the polytope corresponding to the family $\mathcal{C}$ of feasible sets – i.e., their convex hull or its approximation, which is a "continuous relaxation" of the constraints $\mathcal{C}$. The final approximation factor is then $f(\hat{X})/f(X^*)$, where X^* is the exact optimizer of f over $\mathcal{C}$.

For minimization, the convex *Lovász extension* is a suitable relaxation of f. Appropriately rounding the resulting optimal continuous solutions leads to a number of approximation guarantees. For maximization, ideally we could utilize a concave extension. Since the tightest concave extension of a submodular function is hard to characterize [Vondrák, 2007], we instead use the *multilinear extension* that behaves like a concave function in certain directions [Chekuri et al., 2011, Calinescu et al., 2011]. Our resulting algorithms often achieve better bounds than discrete greedy approaches. In terms of complexity, the convex relaxation, or the Lovász extension is extremely simple to evaluate. Both the function valuation and its gradient can be evaluated in $O(n)$ complexity. The multilinear extension, and its gradient, however has an exponential number of terms in the sum, and is hard to evaluate directly. One possibility is to approximate this sum via sampling. However, the number of samples needed for (theoretically) sufficient accuracy is polynomial but in many cases still prohibitively large for practical applications [Calinescu et al., 2011] – we discuss this further in Section 7.4. In this chapter, we show that some practically useful submodular functions have alternative, low-complexity formulations of the multilinear extension that circumvent sampling entirely.

An important quantity is the *integrality gap* that measures – over the class $\mathcal{S}$ of all submodular (or monotone submodular) functions – the largest possible discrepancy between

the optimal discrete solution and the optimal continuous solution. For minimization problems, the integrality gap is defined as:

$$\mathcal{I}_{\mathcal{C}}^{\mathcal{S}} \triangleq \sup_{f \in \mathcal{S}} \frac{\min_{X \in \mathcal{C}} f(X)}{\min_{x \in \mathcal{P}_{\mathcal{C}}} \breve{f}(x)} \geq 1. \tag{7.1}$$

For maximization problems, we would take the supremum over the inverse ratio. In both cases, $\mathcal{I}_{\mathcal{C}}^{\mathcal{S}}$ is defined only for non-negative functions.

The integrality gap largely depends on the specific formulation of the relaxation. Intuitively, it provides a lower bound on our approximation factor: we usually cannot expect to improve the solution by rounding, because any rounded discrete solution is also a feasible solution to the relaxed problem. One rather only hopes, when rounding, to not worsen the cost relative to that of the continuous optimum. Indeed, integrality gaps can often be used to show tightness of approximation factors obtained from relaxations and rounding [Chlamtac and Tulsiani, 2012].

One way to see the relationship between the integrality gap and the approximation factor obtained by rounding is as follows. Let ROPT denote the optimal relaxed value, while DOPT denotes the optimal discrete solution. The integrality gap measures the gap between DOPT and ROPT, i.e $\mathcal{I} = \text{DOPT}/\text{ROPT}$. Let RSOL denote the rounded solution obtained from the relaxed optimum. The way one obtains bounds in a rounding scheme, is by bounding the gap between RSOL and ROPT, which naturally is an upper bound on the approximation factor (which is the gap between DOPT and RSOL. However, notice that the gap between RSOL and ROPT is lower bounded by the integrality gap. Hence the integrality gap captures the tightness of the rounding scheme.

7.3 Submodular Minimization

For submodular minimization, the optimization problem in Step 1 is a convex optimization problem, and can be solved efficiently if one can efficiently project onto the polytope $\mathcal{P}_{\mathcal{C}}$. Our second ingredient is rounding. To round, a surprisingly simple thresholding turns out to be quite effective for a large number of constrained and unconstrained submodular minimization problems: choose an appropriate $\theta \in (0, 1)$ and pick all elements with "weights" above θ, i.e.,

$\hat{X}_\theta = \{i : \hat{x}(i) \geq \theta\}$. We call this procedure the *θ-rounding procedure*. In the following sections, we first review relaxation techniques for unconstrained minimization (which are known), and afterwards phrase a generic framework for constrained minimization. Interestingly, both constrained and unconstrained versions essentially admit the same rounding strategy and algorithms.

7.3.1 Unconstrained Submodular Minimization

Continuous relaxation techniques for unconstrained submodular minimization have been well studied [Grötschel et al., 1981, Stobbe, 2013, Bach, 2013, Fujishige, 2005]. In this case, $\mathcal{P}_\mathcal{C} = [0,1]^n$, and importantly, the approximation factor and integrality gap are both 1.

Lemma 7.1. *[Fujishige, 2005] For any submodular function f, it holds that* $\min_{X \subseteq V} f(X) = \min_{x \in [0,1]^n} \breve{f}(x)$. *Given a continuous minimizer $x^* \in \operatorname{argmin}_{x \in [0,1]^n} \breve{f}(x)$, the discrete minimizers are exactly those obtained by θ-rounding x^*, for any $\theta \in (0,1)$. Given a continuous minimizer $x^* \in \operatorname{argmin}_{x \in [0,1]^n} \breve{f}(x)$, the discrete minimizers are exactly the maximal chain of sets $\emptyset \subset X_{\theta_1} \subset \ldots X_{\theta_k}$ obtained by θ-rounding x^*, for $\theta_j \in (0,1)$.*

Since the Lovász extension is a non-smooth convex function, it can be minimized up to an additive accuracy of ϵ in $O(1/\epsilon^2)$ iterations of the subgradient method. This accuracy directly transfers to the discrete solution if we choose the best set obtained with any $\theta \in (0,1)$ [Bach, 2013]. For special cases, such as submodular functions derived from concave functions, smoothing techniques yield a convergence rate of $O(1/t)$ [Stobbe and Krause, 2010].

It can often be faster to instead solve the unconstrained regularized problem $\min_{x \in \mathbb{R}^n} \breve{f}(x) + \frac{1}{2}\|x\|_2^2$. A motivation for this approach is that the problem, $\min_{x \in [0,1]^n} \breve{f}(x)$, can be seen as a specific form of barrier function $\min_{x \in \mathbb{R}^n} \breve{f}(x) + \delta_{[0,1]^n}(x)$ [Stobbe, 2013]. The level sets of the optimal solution to the regularized problem are the solutions of the entire regularization path of $\min_{X \subseteq V} f(X) + \theta|X|$ [Bach, 2013], and therefore a simple rounding at 0 gives the optimal solution. The dual problem $\min_{y \in \mathcal{B}_f} \|x\|_2^2$ is amenable to the Frank-Wolfe algorithm (or conditional gradient) [Frank and Wolfe, 1956]—with a convergence rate of $O(1/\sqrt{t})$, or an

improved fully corrective version known as the minimum norm point algorithm [Fujishige and Isotani, 2011]. The complexity of the improved method is still open. Moreover, regularizers other than the ℓ_2 norm are possible [Stobbe, 2013]. For decomposable functions, reflection methods can also be very effective [Jegelka et al., 2013a].

7.3.2 Constrained Submodular Minimization

We next address submodular minimization under constraints, where rounding affects the accuracy of the discrete solution. By appropriately formulating the problem, we show that θ-rounding applies to a large class of problems. We assume that the family $\mathcal{C}$ of feasible solutions can be expressed by a polynomial number of linear inequalities, or at least that linear optimization over $\mathcal{C}$ can be done efficiently, as is the case for matroid polytopes [Edmonds, 1970].

A straightforward relaxation of $\mathcal{C}$ is the convex hull $\mathcal{P}_\mathcal{C} = \text{conv}(1_X, X \in \mathcal{C})$ of $\mathcal{C}$. Often however, it is not possible to obtain a decent description of the inequalities determining $\mathcal{P}_\mathcal{C}$, even in cases when minimizing a linear function over $\mathcal{C}$ is easy (two examples are the s-t cut and s-t path polytopes [Schrijver, 2003]). In those cases, we relax $\mathcal{C}$ to its *up-monotone* closure $\widehat{\mathcal{C}} = \{X \cup Y \mid X \in \mathcal{C} \text{ and } Y \subseteq V\}$. With $\widehat{\mathcal{C}}$, a set is feasible if it is in $\mathcal{C}$ or a superset of a set in $\mathcal{C}$. The convex hull of $\widehat{\mathcal{C}}$ is the up-monotone extension of $\mathcal{P}_\mathcal{C}$ within the hypercube, i.e. $\mathcal{P}_{\widehat{\mathcal{C}}} = \hat{\mathcal{P}}_\mathcal{C} = (\mathcal{P}_\mathcal{C} + \mathbb{R}^n_+) \cap [0,1]^n$, which is often easier to characterize than $\mathcal{P}_\mathcal{C}$. If $\mathcal{C}$ is already up-monotone, then $\hat{\mathcal{P}}_\mathcal{C} = \mathcal{P}_\mathcal{C}$. The following proposition formalizes this equivalence.

Proposition 7.1. *For any family $\mathcal{C}$ of feasible sets, the relaxed hull $\widehat{\mathcal{P}}$ of $\mathcal{C}$ is the convex hull of $\widehat{\mathcal{C}}$: $\hat{\mathcal{P}}_\mathcal{C} = \mathcal{P}_{\widehat{\mathcal{C}}} = \text{conv}(1_X, X \in \widehat{\mathcal{C}})$. For any up-monotone constraint $\mathcal{C}$, the relaxation is tight: $\hat{\mathcal{P}}_\mathcal{C} = \mathcal{P}_\mathcal{C}$.*

Although this Proposition seems intuitive, we prove it here for completeness.

Proof. Let $\mathcal{P}_{\hat{\mathcal{C}}} = \text{conv-hull}(1_X, X \in \hat{\mathcal{C}})$. We need to show that $\mathcal{P}_{\hat{\mathcal{C}}} = \hat{\mathcal{P}}_\mathcal{C}$.

First, we observe that the characteristic vector 1_X for every set $X \in \hat{\mathcal{C}}$ lies in $\hat{\mathcal{P}}_\mathcal{C}$. This follows because, by definition, for every set $X \in \widehat{\mathcal{C}}$, there exists a set $Z \subseteq V$ such that

$X \backslash Z \in \mathcal{C}$. Since $1_Z \in \mathcal{P}_\mathcal{C}$ and $1_{X \backslash Z} \in \mathbb{R}^n_+$, we conclude that $1_X = 1_Z + 1_{X \backslash Z} \in \hat{\mathcal{P}}_\mathcal{C}$, and therefore, $\mathcal{P}_{\hat{\mathcal{C}}} \subseteq \hat{\mathcal{P}}_\mathcal{C}$.

We next show $\mathcal{P}_{\hat{\mathcal{C}}} \supseteq \hat{\mathcal{P}}_\mathcal{C}$ by investigating the polytope $\hat{\mathcal{P}}_\mathcal{C}$. Since it is an intersection of $\mathcal{P}_\mathcal{C} + \mathbb{R}^n_+$ (which is an integral polyhedron) and $[0,1]^n$, it follows from [Schrijver, 2003] (Theorem 5.19), that $\hat{\mathcal{P}}_\mathcal{C}$ is also integral. Let 1_Z be any extreme point of $\hat{\mathcal{P}}_\mathcal{C}$. We will show that $Z \in \hat{\mathcal{C}}$, and this implies $\mathcal{P}_{\hat{\mathcal{C}}} \supseteq \hat{\mathcal{P}}_\mathcal{C}$. Since $1_Z \in \hat{\mathcal{P}}_\mathcal{C}$, there exists a vector $x \in \mathcal{P}_\mathcal{C}$ and $y \geq 0$ such that $x = 1_Z - y$. This implies that $x = \sum_{i=1}^K \lambda_i 1_{X_i}, X_i \in \mathcal{C}$. Since $y \geq 0$, it must hold that $X_i \subseteq Z$ for all $1 \leq i \leq K$. As Z contains at least one feasible set X_i, $Z \in \hat{\mathcal{C}}$, proving the result. The second statement of the result follows from the first, because an up-monotone constraint satisfies $\mathcal{C} = \hat{\mathcal{C}}$. □

Optimization. The relaxed minimization problem $\min_{x \in \hat{\mathcal{P}}_\mathcal{C}} \breve{f}(x)$ is non-smooth and convex with linear constraints, and therefore amenable to, e.g., projected subgradient methods. We first assume that the submodular function f is monotone non-decreasing (which often holds in applications), and later relax this assumption for a large class of constraints.

For projected (sub)gradient methods, it is vital that the projection on $\hat{\mathcal{P}}_\mathcal{C}$ can be done efficiently. Indeed, this holds with the above assumptions that we can efficiently solve a linear optimization over $\hat{\mathcal{P}}_\mathcal{C}$. In this case, e.g. Frank-Wolfe methods [Frank and Wolfe, 1956] apply. The projection onto matroid polyhedra can also be cast as a form of unconstrained submodular function minimization and is hence polynomial time solvable [Fujishige, 2005].

To apply splitting methods such as the alternating directions method of multipliers (ADMM) [Boyd et al., 2011], we write the problem as $\min_{x,y:x=y} \breve{f}(x) + I(y \in \hat{\mathcal{P}}_\mathcal{C})$. One iteration of ADMM requires (1) computing the proximal operator of f, (2) projecting onto $\mathcal{P}_\mathcal{C}$, and (3) doing a simple dual update step. Computing the proximal operator of the Lovász extension is equivalent to unconstrained submodular minimization, or to solving the minimum norm point problem. In special cases, faster algorithms apply [Nagano and Kawahara, 2013, Stobbe and Krause, 2010, Jegelka et al., 2013a]. An approximate proximal operator for generalized graph cuts [Jegelka et al., 2011] can be obtained via parametric max-flows in

$O(n^2)$ [Nagano and Kawahara, 2013, Chambolle and Darbon, 2009].

Algorithm 24 The constrained θ-rounding scheme

Input: Continuous vector $\hat{x}$, Constraints $\mathcal{C}$

Output: Discrete set $\hat{X}$

1: Obtain the chain of sets $\emptyset \subset X_1 \subset X_2 \subset X_k$ corresponding to $\hat{x}$.
2: **for** $j = 1, 2, \cdots, k$ **do**
3: **if** $\exists \hat{X} \subseteq X_j : \hat{X} \in \mathcal{C}$ **then**
4: Return $\hat{X}$
5: **end if**
6: **end for**

Rounding. Once we have obtained a minimizer $\hat{x}$ of $\breve{f}$ over $\widehat{\mathcal{P}}_{\mathcal{C}}$, we apply simple θ-rounding. Whereas in the unconstrained case, $\hat{X}_\theta$ is feasible for any $\theta \in (0, 1)$, we must now ensure $\hat{X}_\theta \in \widehat{\mathcal{C}}$. Hence, we pick the largest threshold θ such that $\hat{X}_\theta \in \widehat{\mathcal{C}}$, i.e., the smallest $\hat{X}_\theta$ that is feasible. This is always possible since $\widehat{\mathcal{C}}$ is up-monotone and contains V. The threshold θ can be found using $O(\log n)$ checks among the sorted entries of the continuous solution $\hat{x}$ or n checks in the unsorted vector $\hat{x}$. The following lemma states how the threshold θ determines a worst-case approximation:

Lemma 7.2. *For a monotone submodular f and any $\hat{x} \in [0, 1]^V$ and $\theta \in (0, 1)$ such that $\hat{X}_\theta = \{i \mid \hat{x}_i \geq \theta\} \in \hat{\mathcal{C}}$, it holds that $f(\hat{X}_\theta) \leq \frac{1}{\theta}\breve{f}(\hat{x})$.*

$$f(\hat{X}_\theta) \leq \frac{1}{\theta}\breve{f}(\hat{x}). \tag{7.2}$$

If, moreover, $\breve{f}(\hat{x}) \leq \beta \min_{x \in \hat{\mathcal{P}}_{\mathcal{C}}} \breve{f}(x)$, then it holds that $f(\hat{X}_\theta) \leq \frac{\beta}{\theta} \min_{X \in \mathcal{C}} f(X)$.

Proof. The proof follows from the positive homogeneity of $\breve{f}$ and the monotonicity of f and

$\breve{f}$:

$$\theta f(\hat{X}_\theta) = \theta \breve{f}(1_{X_\theta}) \tag{7.3}$$

$$= \breve{f}(\theta 1_{X_\theta}) \tag{7.4}$$

$$\leq \breve{f}(\hat{x}) \tag{7.5}$$

$$\leq \beta \min_{x \in \hat{\mathcal{P}}_\mathcal{C}} \breve{f}(x) \tag{7.6}$$

$$\leq \beta \min_{X \in \hat{\mathcal{C}}} f(X) \tag{7.7}$$

$$\leq \beta \min_{X \in \mathcal{C}} f(X) \tag{7.8}$$

Inequality (7.6) follows from the approximation bound of $\hat{x}$ with respect to $\min_{x \in \hat{\mathcal{P}}_\mathcal{C}} \breve{f}(x)$, and (7.7) uses the observation that the optimizer of the continuous problem is smaller than the discrete one. Finally (7.8) follows from (7.7) since it is optimizing over a smaller set. □

The set $\hat{X}_\theta$ is in $\widehat{\mathcal{C}}$ and therefore guaranteed to be a superset of a solution $\hat{Y}_\theta \in \mathcal{C}$. As a final step, we prune down $\hat{X}_\theta$ to $\hat{Y}_\theta \subseteq \hat{X}_\theta$. Since the objective function is non-decreasing, $f(\hat{Y}_\theta) \leq f(\hat{X}_\theta)$, Lemma 7.2 holds for $\hat{Y}_\theta$ as well. If, in the worst case, $\theta = 0$, then the approximation bound in Lemma 7.2 is unbounded. Fortunately, in most cases of interest we obtain polynomially bounded approximation factors.

In the following, we will see that our $\widehat{\mathcal{P}}_\mathcal{C}$ provides the basis for relaxation schemes under a variety of constraints, and that these, together with θ-rounding, yield bounded-factor approximations. We assume that there exists a family $\mathcal{W} = \{W_1, W_2, \dots\}$ of sets $W_i \subseteq V$ such that the polytope $\hat{\mathcal{P}}_\mathcal{C}$ can be described as

$$\hat{\mathcal{P}}_\mathcal{C} = \Big\{ x \in [0,1]^n \;\Big|\; \sum_{i \in W} x_i \geq b_W \text{ for all } W \in \mathcal{W} \Big\}. \tag{7.9}$$

Analogously, this means that $\hat{\mathcal{C}} = \{X \mid |X \cap W| \geq b_W, \text{ for all } W \in \mathcal{W}\}$. In our analysis, we do not require $\mathcal{W}$ to be of polynomial size, but a linear optimization over $\widehat{\mathcal{P}}_\mathcal{C}$ or a projection onto it should be possible at least within a bounded approximation factor. This is the case for s-t paths and cuts, covering problems, and spanning trees.

This means we are addressing the following class of optimization problems:

$$\begin{aligned} \min_{x} \quad & \breve{f}(x) \\ \text{subject to} \quad & x \in [0,1]^n,\ \sum_{i \in W} x_i \geq b_W,\ \forall W \in \mathcal{W} \end{aligned} \tag{7.10}$$

The following main result states approximation bounds and integrality gaps for the class of problems described by Equation (7.9).

Theorem 7.1. *The θ-rounding scheme for constraints $\mathcal{C}$ whose relaxed polytope $\hat{\mathcal{P}}_{\mathcal{C}}$ can be described by Equation* (7.9) *achieves a worst case approximation bound of* $\max_{W \in \mathcal{W}} |W| - b_W + 1$. *If we assume that the sets in $\mathcal{W}$ are disjoint, the integrality gap for these constraints matches the approximation:* $\mathcal{I}_{\mathcal{C}}^{S} = \max_{W \in \mathcal{W}} |W| - b_W + 1$.

Proof. To show the first part of the theorem, we invoke Lemma 7.2. The constraints of Equation (7.9) demand that for every set $W \in \mathcal{W}$, at least $b_W \leq |W|$ elements need to be chosen, or "covered". Consequently, to round a vector $x \in \hat{\mathcal{P}}_{\mathcal{C}}$, it is sufficient to choose $\theta = \min_{W \in \mathcal{W}} x_{[b_W, W]}$ as the rounding threshold, where $x_{[k,A]}$ denotes the k^{th} largest entry of x in a set A. The worst case scenario is that the $b_W - 1$ entries of x with indices in the set W are all 1, and the remaining mass of 1 is equally distributed over the remaining elements in W. In this case, the value of $x_{[b_W, W]}$ is $1/(|W| - b_W + 1)$. Since the constraint requires $\sum_{i \in W} x_i \geq b_W$, it must hold that $x_{[b_W, W]} \geq 1/(|W| - b_W + 1)$. The approximation factor then follows with Lemma 7.2.

To analyze the integrality gap, we first show the following Lemma.

Lemma 7.3. *If $\hat{\mathcal{P}}_{\mathcal{C}}$ is described by Equation 7.9, then the integrality gap can be as large as* $\mathcal{I}_{\mathcal{C}}^{S} \geq \max_{W \in \mathcal{W}} |W| - b_W + 1$. *Moreover, if $\mathcal{C}$ and θ are such that θ-rounding provides a valid set in $\widehat{\mathcal{C}}$, the integrality gap is never larger than that:* $\mathcal{I}_{\mathcal{C}}^{S} \leq \frac{\beta}{\theta}$.

To show the lower bound on the gap, we construct a simple example. Assume $W' = \text{argmax}_{W \in \mathcal{W}} |W| - b_W + 1$ satisfies $W \cap W' = \emptyset, \forall W \in \mathcal{W}, W \neq W'$. This is true since the $\mathcal{W}$ consists of disjoint sets. Let B' be a subset of W' such that $|B'| = |W'| - b_{W'} + 1$. In other

words, every feasible solution must intersect B' Now we define $f(X) = \min\{|X \cap B'|, 1\}$. The Lovász extension of this function is $\breve{f}(x) = \max_{j \in X \cap B'} x_j$. An optimal continuous solution for $\breve{f}$ and $\mathcal{W}$ has entries $x^*_j = 1$ for $j \notin B'$ and $x^*_j = 1/(|W'| - b_{W'} + 1)$ for $j \in B'$. In this case, the integrality gap is $f(X^*)/\breve{f}(x^*) = 1/(W'| - b_{W'} + 1)^{-1} = W'| - b_{W'} + 1$.

The upper bound on the gap follows from the approximation factor:

$$\begin{aligned} I^S_{\mathcal{C}} &= \max_f \frac{f(X^*)}{\min_{x \in \mathcal{P}_{\mathcal{C}}} \breve{f}(x)} \\ &\leq \max_f \frac{f(\hat{X}_\theta)}{\min_{x \in \mathcal{P}_{\mathcal{C}}} \breve{f}(x)} \\ &= \frac{\beta}{\theta} \end{aligned}$$

where the second inequality follows from the fact that $f(X^*) \leq \hat{X}_\theta$ and the last one from Lemma 7.2. □

Note that the integrality gap matches the approximation factor, thus showing the tightness of the rounding strategy for a large class of constraints. In particular, for the class of constraints we consider below, we can provide instances where the integrality gap matches the approximation factors.

A result similar to Theorem 7.1 was shown in [Koufogiannakis and Young, 2013] for a different, greedy algorithmic technique. While their result also holds for a large class of constraints, for the constraints in Equation (7.9) they obtain a factor of $\max_{W \in \mathcal{W}} |W|$, which is worse than Theorem 7.1 if $b_W > 1$. This is the case, for instance, for matroid span constraints, cardinality constraints, trees and multiset covers.

Pruning. The final piece of the puzzle is the pruning step, where we reduce the set $\hat{X}_\theta \in \hat{\mathcal{C}}$ to a final solution $\hat{Y}_\theta \subseteq \hat{X}_\theta$ that is feasible: $\hat{Y}_\theta \in \mathcal{C}$. This is important when the true constraints $\mathcal{C}$ are not up-monotone, as is the case for cuts or paths. Since we have assumed that the function f is monotone, pruning can only reduce the objective value. The pruning step means finding *any* subset of $\hat{X}_\theta$ that is in $\mathcal{C}$, which is often not hard. We propose the following heuristic for this: if $\mathcal{C}$ admits (approximate) linear optimization, as is the case for

all the constraints considered here, then we may improve over a given rounded subset by assigning additive weights: $w(i) = \infty$ if $i \notin \hat{X}_\theta$, and otherwise use either uniform ($w(i) = 1$) or non-uniform ($w(i) = 1 - \hat{x}(i)$) weights.

We then solve $\hat{Y}_\theta \in \operatorname{argmin}_{Y \in \mathcal{C}} \sum_{i \in Y} w(i)$. Uniform weights lead to the solution with minimum cardinality, and non-uniform weights will give a bias towards elements with higher certainty in the continuous solution. Truncation via optimization works well for paths, cuts, matchings or matroid constraints.

Non-monotone submodular functions. We can easily extend the relaxation paradigm above to non-monotone functions, in case the constraints are up-monotone. The strategy exactly follows from the results from Section 3.6 and Lemma 3.15. Examples of up-monotone constraints are matroid spans (Sec. 7.3.2) and covers (Sec. 7.3.2).

Down-monotone constraints. For down-monotone constraints, the up-monotone closure $\hat{C}$ will be the entire power set 2^V. Here, a different construction is needed. Define a monotone extension $f^d(X) = \min_{Y : Y \subseteq V \setminus X} f(Y)$. Also, define $\mathcal{C}' = \{X : V \setminus X \in \mathcal{C}\}$. It is easy to see that $\mathcal{C}'$ is up-monotone. Notice that if we assume f to be normalized and nonnegative, then the empty set will always be a great solution for down-monotone constraints, and this extension may not make sense. However in general the submodular function may not necessarily be normalized, and all we need for this is that the function f^d is normalized and non-negative. In other words, this implies that the function f itself be non-negative and $\min_{X \subseteq V} f(X) = 0$.

Lemma 7.4. *Given a submodular function f, the function f^d is monotone non-decreasing submodular, and can be evaluated in polynomial time. It holds that*

$$\min_{X \in \mathcal{C}} f(X) = \min_{X \in \mathcal{C}'} f^d(X). \tag{7.11}$$

Moreover, minimizing f^d over $\mathcal{C}'$ is equivalent to minimizing f over $\mathcal{C}$ in terms of the approximation factor.

	Matroid Constraints		Set Covers		Paths, Cuts and Matchings		
	Cardinality	Trees	Vertex Covers	Edge Covers	Cuts	Paths	Matchings
CR.	$n-k+1$	$m-n+1$	2	$deg(G) \leq n$	$P_{max} \leq n$	$C_{max} \leq m$	$deg(G) \leq n$
MMin	k	n	$\|VC\| \leq n$	$\|EC\| \leq n$	$C_{max} \leq m$	$P_{max} \leq n$	$\|M\| \leq n$
EA	$\sqrt{n}$	$\sqrt{m}$	$\sqrt{n}$	$\sqrt{m}$	$\sqrt{m}$	$\sqrt{m}$	$\sqrt{m}$
Integrality Gaps	$\Omega(n-k+1)$	$\Omega(m-n+1)$	2	$\Omega(n)$	$\Omega(n)$	$\Omega(m)$	$\Omega(n)$
Hardness[1]	$\Omega(\sqrt{n})$	$\Omega(n)$	$2-\epsilon$	$\Omega(n)$	$\Omega(\sqrt{m})$	$\Omega(n^{2/3})$	$\Omega(n)$

Table 7.2: Comparison of the results of the continuous relaxation framework (CR) with the semigradient MMin framework, the Ellipsoidal Approximation (EA) algorithm of [Goemans et al., 2009], hardness [Goel et al., 2009, Iwata and Nagano, 2009, Svitkina and Fleischer, 2008], and the integrality gaps of the corresponding constrained submodular minimization problems. Note the complementarity between CR and SG. See text for further details. In this case, we investigate worst case results with $\kappa_f = 0$.

Proof. The proof of this result is very similar to the previous Lemma. To show submodularity, note that the function $g(Z) = \min_{Y:Y\subseteq Z} f(Y)$ as a function of Z is submodular [Fujishige, 2005]. Then, $f^d(X) = g(V\backslash X)$ is also submodular. We also observe that f^d is monotone.

If the constraints $\mathcal{C}$ are down-monotone, then $\mathcal{C}'$ is up-monotone. Let X^{d*} be the exact minimizer of f^d over $\mathcal{C}'$. This implies that $V\backslash X^{d*} \in \mathcal{C}$. By the definition of f^d, it implies that $\exists X \subseteq V\backslash X^{d*}$ such that $f^d(X^{d*}) = f(X)$. Moreover, since $\mathcal{C}$ is down monotone, $X \in \mathcal{C}$ (since $V\backslash X^{d*} \in \mathcal{C}$). Hence $\min_{X\in\mathcal{C}} f(X) \leq f(X) = \min_{X\in\mathcal{C}'} f^d(X)$.

Conversely, let X^* be the minimizer of f under $\mathcal{C}$. Then $f^d(V\backslash X^*) = \min_{X\subseteq X^*} f(X) \leq f(X^*)$. Hence $\min_{X\in\mathcal{C}'} f^d(X) \leq f^m(X^*) = \min_{X\in\mathcal{C}} f(X)$. The approximation factor follows similarly. □

To demonstrate the utility of Theorem 7.1, we apply it to a variety of problems. Many of the constraints below are based on a graph $G = (\mathcal{V}, \mathcal{E})$, and in that case the ground set is the set $\mathcal{E}$ of graph edges. When the context is clear, we overload notation and refer to $n = |\mathcal{V}|$ and $m = |\mathcal{E}|$. Results are summarized in Table 7.2.

Matroid Constraints

An important class of constraints are matroid span or base constraints (both are equivalent since f is monotone), with cardinality constraints (uniform matroids) and spanning trees (graphic or cycle matroids) as special cases. A matroid $\mathcal{M} = (\mathcal{I}_\mathcal{M}, r_\mathcal{M})$ is defined by its down-monotone family of independent sets $\mathcal{I}_\mathcal{M}$ or its rank function $r_\mathcal{M} : 2^V \to \mathbb{R}$. A set Y is a *spanning set* if its rank is that of V: $r_M(Y) = r_M(V)$. It is a *base* if $|Y| = r_M(Y) = r_M(V)$. Hence, the family of all spanning sets is the up-monotone closure of the family of all bases (e.g., supersets of spanning trees of a graph in the case of a graphic matroid). See [Schrijver, 2003] for more details on matroids. Let $\mathcal{S}_\mathcal{M}$ denote the spanning sets of matroid $\mathcal{M}$, and set $k = r_\mathcal{M}(V)$. It is then easy to see that with $\mathcal{C} = \mathcal{S}_\mathcal{M}$, the polytope $\mathcal{P}_\mathcal{C}$ is the matroid span polytope, which can be described as $\mathcal{P}_\mathcal{C} = \{x \in [0,1]^n, x(S) \geq r_\mathcal{M}(V) - r_\mathcal{M}(V \backslash S), \forall S \subseteq V\}$ [Schrijver, 2003]. This is clearly in the form of Eqn. 7.9. Although this polytope is described via an exponential number of inequalities, a linear program can still be solved efficiently over it [Edmonds, 1970]. Furthermore, projecting onto this polytope is also easy, since it corresponds to submodular function minimization [Fujishige, 2005].

Corollary 7.1. *Let $\hat{Y}_\theta$ be the rounded and pruned solution obtained from minimizing the Lovász extension over the span polytope. Then $f(\hat{Y}_\theta) \leq (n - k + 1) f(X^*)$. The integrality gap is also $n - k + 1$.*

Proof. This result follows directly as a Corollary from Theorem 7.1, by observing that the approximation factor in this case is $\max_{S \subseteq V} |S| - r_\mathcal{M}(V) + r_\mathcal{M}(V \backslash S) - 1$. Then notice that $r_{\mathcal{M}^*}(S) = |S| - r_\mathcal{M}(V) + r_\mathcal{M}(V \backslash S)$, where $\mathcal{M}^*$ is the dual matroid of $\mathcal{M}$ [Schrijver, 2003]. Correspondingly the approximation factor can be expressed as $\max_{S \subseteq V} r_{\mathcal{M}^*}(S) - 1 = r_{\mathcal{M}^*}(V) - 1 = n - k + 1$. To show the integrality gap, we use Lemma 7.3. Consider the simple uniform matroid, with $\mathcal{W} = \{V\}$. A straightforward application of Lemma 7.3 then reveals the integrality gap. □

In general, the rounding step will only provide an $\hat{X}_\theta$ that is a spanning set, but not a base. We can prune it to a base by greedily finding a maximum weight base among the elements of

$\hat{X}_\theta$. The worst-case approximation factor of $n-k+1$ complements other known results for this problem [Iyer et al., 2013b, Goemans et al., 2009]. The semi-gradient framework of [Iyer et al., 2013b] guarantees a bound of k, while more complex (and less practical) sampling based methods [Svitkina and Fleischer, 2008] and approximations [Goemans et al., 2009] yield factors of $O(\sqrt{n})$. The factor k of [Iyer et al., 2013b] is the best for small k, while our continuous relaxation works well when k is large. Moreover, for a monotone function under matroid span constraints, the pruning step will always find a set in the base, and hence a matroid span constraint is, for all intents and purposes, identical to a base constraint.

These results also extend to non-monotone cost functions, with approximation bounds of $n-k+1$ (Cor. 7.1) and k (using [Iyer et al., 2013b]) for matroid span constraints. We can also handle matroid independence constraints. Note that

$$\min_{X \in \text{Indep}(\mathcal{M})} f(X) = \min_{X \in \text{Span}(\mathcal{M}^*)} f'(X), \tag{7.12}$$

where $f'(X) = f(V \backslash X)$ is a submodular function and Span($\mathcal{M}$) and Indep($\mathcal{M}$) refer to the Span and Independence sets of a Matroid $\mathcal{M}$, and $\mathcal{M}^*$ is the dual matroid of $\mathcal{M}$ [Schrijver, 2003]. Recall that the rank function of the dual Matroid satisfies $r_{\mathcal{M}^*}(V) = n - r_{\mathcal{M}}(V)$. Hence, the approximation factors for the matroid independence constraints are $k+1$ for our relaxation based framework and $n-k$ for the discrete framework of [Iyer et al., 2013b]. Again, we see how our results complement those of [Iyer et al., 2013b]. Moreover, the algorithm of [Goemans et al., 2009] achieves an approximation factor of $O(\sqrt{n})$, in both cases.

Cardinality Constraints. This is a special class of matroid, called the uniform matroid. Since it suffices to analyze monotone submodular functions, the constraint of interest is $\mathcal{C} = \{X : |X| = k\}$. In this case, the corresponding polytope takes a very simple form: $\mathcal{P}_\mathcal{C} = \{x \in [0,1]^n : \sum_i x_i = k\}$. The projection onto this polyhedron can be easily performed using bisection [Adams et al., 2012]. Furthermore, the rounding step in this context is very intuitive. It corresponds to choosing the elements with the k largest entries in the continuous solution $\hat{x}$.

Spanning Trees. Here, the ground set $V = \mathcal{E}$ is the edge set in a graph and $\mathcal{C}$ is the set of all spanning trees. The corresponding polytope $\mathcal{P}_{\mathcal{C}}$ is then the spanning tree polytope. The bound of Corollary 7.1 becomes $|\mathcal{E}| - |\mathcal{V}| + 1 = m - n + 1$. The discrete algorithms of [Iyer et al., 2013b, Goel et al., 2009] achieve a complementary bound of $|\mathcal{V}| = n$. For dense graphs, the discrete algorithms admit better worst case guarantees, while for sparse graphs (e.g., embeddable into r-regular graphs for small r), our guarantees are better.

Set Covers

A fundamental family of constraints are set covers. Given a universe $\mathcal{U}$, and a family of sets $\{S_i\}_{i \in V}$, the task is to find a subset $X \subseteq V$ that covers the universe, i.e., $\bigcup_{i \in X} S_i = \mathcal{U}$, and has minimum cost as measured by a submodular function $f : 2^{\mathcal{S}} \to \mathbb{R}$. The set cover polytope is up-monotone, constitutes the set of fractional covers, and is easily represented by Eqn. (7.9) as $\mathcal{P}_{\mathcal{C}} = \{x \in [0,1]^{|V|} \mid \sum_{i:u \in S_i} x(i) \geq 1, \forall u \in \mathcal{U}\}$. The following holds for minimum submodular set cover:

Corollary 7.2. *The approximation factor of our algorithm, and the integrality gap for the minimum submodular set cover problem is* $\gamma = \max_{u \in \mathcal{U}} |\{i : u \in S_i\}|$.

Proof. The proof of this corollary follows from the expression of the set cover polytope and Theorem 7.1. To show the integrality gap, notice that the sets $\mathcal{W}$ here are $W_u = \{i : u \in S_i\}$. Consider an instance of the set cover problem when these sets are disjoint. A direct application of Lemma 7.3 then provides the integrality gap (since the sets in $\mathcal{W}$ are disjoint). □

The approximation factor in Corollary 7.2 (without the integrality gap) was first shown in [Iwata and Nagano, 2009]. The quantity γ corresponds to the maximum frequency of the elements in $\mathcal{U}$.

A generalization of set cover is the multi-set cover problem [Rajagopalan and Vazirani, 1998], where every element u is to be covered multiple (c_u) times. The multi-cover constraints can be formalized as $\mathcal{P}_{\mathcal{C}} = \{x \in [0,1]^{|\mathcal{S}|} \mid \sum_{i:u \in S_i} x(i) \geq c_u, \forall u \in \mathcal{U}\}$.

Corollary 7.3. *The approximation factor and integrality gap of the multi-set cover problem is* $\max_{u \in \mathcal{U}} |\{i : u \in S_i\}| - c_u + 1$.

This result also implies the bound for set cover (with $c_u = 1$). Since the rounding procedure above yields a solution that is already a set cover (or a multi set cover), a subsequent pruning step is not necessary.

Vertex Cover. A vertex cover is a special case of a set cover, where $\mathcal{U}$ is the set of edges in a graph, V is the set of vertices, and S_v is the set of all edges incident to $v \in V$. Corollary 7.2 immediately provides a 2-approximation algorithm for the minimum submodular vertex cover. The 2-approximation for the special case of vertex was also shown in [Goel et al., 2009, Iwata and Nagano, 2009]. The corresponding integrality gap is 2, too. To show this gap, consider a graph such that each vertex is incident to exactly one edge. The sets $\mathcal{W}$ are the sets of vertices for each edge and, in this case, are disjoint. As a result, Theorem 7.1 implies that the integrality gap is exactly 2. We may even take a complete graph and use the modular function $f(X) = |X|$. This shows that the integrality gap is 2 even when the function is modular (linear). In fact, no polynomial-time algorithm can guarantee an approximation factor better than $2 - \epsilon$, for any $\epsilon > 0$ [Goel et al., 2009].

Edge Cover. In the Edge Cover problem, $\mathcal{U}$ is the set of vertices in a graph, V is the set of edges and S_v contains the two vertices comprising edge v. We aim to find a subset of edges such that every vertex is covered by some edge in the subset. It is not hard to see that the approximation factor we obtain is the maximum degree of the graph $deg(G)$, which is upper bounded by $|\mathcal{V}|$, but is often much smaller. The algorithm in [Iyer et al., 2013b] has an approximation factor of the size of the edge cover $|EC|$, which is also upper bounded by $O(|\mathcal{V}|)$. These factors match the lower bound shown in [Goel et al., 2009].

Cuts, Paths and Matchings

Even though Eqn. (7.9) is in the form of covering constraints, it can help solve problems with apparently very different types of constraints. The covering generalization works if we relax $\mathcal{C}$ to its up-monotone closure: $\widehat{\mathcal{C}}$ demands that a feasible set must contain (or "cover") a set in $\mathcal{C}$. To go from $\widehat{\mathcal{C}}$ back to $\mathcal{C}$, we prune in the end.

Cuts and Paths. Here, we aim to find an edge set $X \subseteq \mathcal{E}$ that forms an s-t path (or an s-t cut), and that minimizes the submodular function f. Both the s-t path and s-t cut polytopes are hard to characterize. However, their up-monotone extension $\hat{\mathcal{P}}_{\mathcal{C}}$ can be easily described. Furthermore, both these polytopes are intimately related to each other as a blocking pair of polyhedra (see [Schrijver, 2003]). The extended polytope for s-t paths can be described as a *cut-cover* [Schrijver, 2003] (i.e., any path must hit every cut at least once): $\hat{\mathcal{P}}_{\mathcal{C}} = \{x \in [0,1]^{|\mathcal{E}|} \mid \sum_{e \in C} x(e) \geq 1, \text{ for every s-t cut } C \subseteq \mathcal{E}\}$. The closure of the s-t path constraint (or the cut-cover) is also called s-t connectors [Schrijver, 2003]. Conversely, the extended s-t cut polytope can be described as a *path-cover* [Schrijver, 2003, Jegelka and Bilmes, 2011c]: $\hat{\mathcal{P}}_{\mathcal{C}} = \{x \in [0,1]^{|\mathcal{E}|} \mid \sum_{e \in P} x(e) \geq 1, \text{ for every s-t path } P \subseteq \mathcal{E}\}$.

Corollary 7.4. *The relaxation algorithm yields an approximation factor of $P_{max} \leq |\mathcal{V}|$ and $C_{max} \leq |\mathcal{E}|$ for minimum submodular s-t path and s-t cut, respectively (P_{max} and C_{max} refer to the maximum size simple s-t path and s-t cut). These match the integrality gaps for both problems.*

Proof. The approximation factors directly follow from Theorem 7.1. In order to show the integrality gaps, we need to construct a set $\mathcal{W}$ of s-t paths and cuts that are disjoint. It is easy to construct such graphs. For example, in the case of s-t cuts, consider a graph with m/P parallel paths between s and t, each of length P. The integrality gap in this setting is exactly P, which matches the approximation factor. Similarly, for the s-t path case, consider a graph of m/C cuts in series. In other words, construct a chain of m/C vertices. Connect each adjacent vertex with C edges. We have m/C disjoint cuts each of size C. The integrality

gap and approximation factor both are C in this setting. $\square$

While the description of the constraints as covers reveals approximation bounds, it does not lead to tractable algorithms for minimizing the Lovász extension. However, the extended cut and the extended path polytopes can be described exactly by a linear number of inequalities [Papadimitriou and Steiglitz, 1998, Schrijver, 2003]. For example, the convex relaxation corresponding to the extended cut polytope can be described as a convex optimization problem subject to $m+1$ linear inequality constraints [Papadimitriou and Steiglitz, 1998, Jegelka and Bilmes, 2011c]. In particular, the relaxed optimization problem can be expressed as:

$$\begin{aligned} & \min_x \breve{f}(x) \\ \text{subject to } & x \in [0,1]^{|\mathcal{E}|}, \pi \in [0,1]^{|\mathcal{V}|} \\ & \pi(v) - \pi(u) + x(e) \geq 0, \forall e = (u,v) \in \mathcal{E} \\ & \pi(s) - \pi(t) \geq 1 \end{aligned} \tag{7.13}$$

In the above, the variables π are additional variables that intuitively represent which vertices are reachable from s. Similarly, the extended path polytope is equivalent to the set of s-t flows with $x \leq 1$ [Schrijver, 2003, §13.2a]. This polytope can be described via $n+1$ linear inequalities.

$$\begin{aligned} & \min_x \breve{f}(x) \\ \text{subject to } & x \in [0,1]^{|\mathcal{E}|} \\ & \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0, \forall v \in \mathcal{V}, v \neq s,t \\ & \sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(s)} x_e = 1, \\ & \sum_{e \in \delta^+(t)} x_e - \sum_{e \in \delta^-(t)} x_e = -1 \end{aligned} \tag{7.14}$$

where $\delta^+(v)$ represents the set of edges entering the vertex v and $\delta^-(v)$ represents the set of edges leaving the vertex v.

The pruning step for paths and covers becomes a shortest path or minimum cut problem, respectively. As in the other cases, the approximations obtained from relaxations complement the bounds of P_{max} for paths and C_{max} for cuts shown in [Iyer et al., 2013b].

Perfect Matchings. Given a graph $G = (\mathcal{V}, \mathcal{E})$, the goal is to find a set of edges $X \subseteq \mathcal{E}$, such that X is a perfect matching in G and minimizes the submodular function f. For a bipartite graph, the polytope $\hat{\mathcal{P}_\mathcal{C}}$ can be characterized as $\mathcal{P}_\mathcal{C} = \{x \in [0,1]^{|\mathcal{E}|} \mid \sum_{e \in \delta(v)} x(e) = 1 \text{ for all } v \in \mathcal{V}\}$, where $\delta(v)$ denotes the set of edges incident to v. Similar to the case of Edge Cover, Theorem 7.1 implies an approximation factor of $deg(G) \leq |\mathcal{V}|$, which matches the lower bound shown in [Goel et al., 2009, Iyer et al., 2013a].

7.4 Submodular Maximization

To relax submodular maximization, we use the multilinear extension and the concave extension.

We first show that this extension can be efficiently computed for a large subclass of submodular functions. As above, $\mathcal{C}$ denotes the family of feasible sets, and $\mathcal{P}_\mathcal{C}$ the polytope corresponding to $\mathcal{C}$. For maximization, it makes sense to consider $\mathcal{C}$ to be down-monotone (particularly when the function is monotone). Such a down-monotone $\mathcal{C}$ could represent, for example, matroid independence constraints, or upper bounds on the cardinality $\mathcal{C} = \{X : |X| \leq k\}$. Analogous to the case of minimization, an approximation algorithm for down-monotone constraints can be extended to up-monotone constraints, by using $f'(X) = f(V \backslash X)$.

The relaxation algorithms use the multilinear extension (Eqn. (1.14)) which in general requires repeated sampling and can be very expensive to compute. Below, we show how this can be computed efficiently and exactly for many practical and useful submodular functions.

Weighted Matroid Rank Functions. A common class of submodular functions are sums of weighted matroid rank functions, defined as:

$$f(X) = \sum_i \max\{w_i(A) | A \subseteq X, A \in \mathcal{I}_i\}, \tag{7.15}$$

for linear weights $w_i(j)$. These functions form a rich class of coverage functions for summarization tasks [Lin, 2012]. Interestingly, the concave extension of this class of functions is efficiently computable [Calinescu et al., 2011, Vondrák, 2007]. Moreover, for a number of matroids, the multilinear extension can also be efficiently computed. In particular, consider the weighted uniform matroid rank function, $f(X) = \sum_i \max\{w_i(A)|A \subseteq X, |A| \leq k\}$. The multilinear extension takes a nice form:

Lemma 7.5. *The multilinear extension corresponding to the weighted uniform matroid rank function, $f(X) = \max\{w(A)|A \subseteq X, |A| \leq k\}$, for a weight vector w can be expressed as (without loss of generality, assume that the vector w is ordered as $w_1 \geq w_2 \cdots \geq w_n$),*

$$\tilde{f}(x) = \sum_{i \in V} w_i x_i \sum_{l=1}^{\min(i,k)} P(x_1, \cdots, x_{i-1}, l-1) \tag{7.16}$$

where

$$P(x_1, \cdots, x_i, l) = \sum_{Z \subseteq S^i, |Z|=l} \prod_{s \in Z} x_s \prod_{t \in S^i \setminus Z} (1 - x_t).$$

and $S^i = \{1, 2, \cdots, i\}$.

Proof. Recall that the multilinear extension of f is

$$\tilde{f}(x) = \sum_{X \subseteq V} f(X) \prod_{s \in X} x_s \prod_{t \in V \setminus X} (1 - x_t), \tag{7.17}$$

where $f(X) = \max\{w(A)|A \subseteq X, |A| \leq k\}$. We can rewrite this sum in terms of the weights. Any $i \in V$ is only counted if it is among the k elements in X that have largest weight. Formally, let $\mathcal{L}_l^i = \{X : i \text{ occurs as the } l^{\text{th}} \text{ largest element in } X\}$. Then

$$\tilde{f}(x) = \sum_{i \in V} w_i \sum_{l=1}^{\min(i,k)} \sum_{X \in \mathcal{L}_l^i} \prod_{s \in X} x_s \prod_{t \in V \setminus X} (1 - x_t). \tag{7.18}$$

This sum has a nice form. We can break the sets $X \in \mathcal{L}_l^i$ into $Y \cup Z$, where Y is a subset of

$\{1, 2, \cdots, i-1\}$ and Z is a subset of $V \backslash \{1, 2, \cdots, i-1\}$. With this, we may rewrite

$$\begin{aligned}
&\sum_{X \in \mathcal{L}_l^i} \prod_{s \in X} x_s \prod_{t \in V \backslash X} (1 - x_t) \\
&= \sum_{\substack{Y \subseteq S^{i-1}, \\ |Y| = l-1, \\ Z \subseteq V \backslash S^{i-1}}} \prod_{s \in Y} x_s \prod_{t \in S^{i-1} \backslash Y} (1 - x_t) \prod_{u \in Z} x_u \prod_{v \in S_i' \backslash Z} (1 - x_v),
\end{aligned}$$

where we wrote $S_i' = V \backslash S^{i-1}$. Note that $\sum_{Z \subseteq S_i'} \prod_{u \in Z} x_u \prod_{v \in S_i' \backslash Z} (1 - x_v) = 1$ and hence,

$$\sum_{X \in \mathcal{L}_l^i} \prod_{s \in X} x_s \prod_{t \in V \backslash X} (1 - x_t) = \sum_{\substack{Z \subseteq S^i, \\ |Z| = l-1}} \prod_{s \in Z} x_s \prod_{t \in S^i \backslash Z} (1 - x_t).$$

□

Interestingly, $P(x_1, \cdots, x_i, l)$ admits a nice recursive relationship, and can be computed efficiently, and therefore also the multilinear extension of the weighted matroid rank function.

Lemma 7.6. *$P(x_1, \cdots, x_i, l)$ admits the following relationship,*

$$\begin{aligned}
P(x_1, \cdots, x_i, l) = x_i P(x_1, \cdots, x_{i-1}, l-1) \\
+ (1 - x_i) P(x_1, \cdots, x_i, l-1).
\end{aligned}$$

Moreover, for every $i \in V$ and $l \in [1, n]$, the matrix of values of $P(x_1, \cdots, x_i, l)$, and hence the multilinear extension of $\tilde{f}$ can be computed in $O(n^2)$ time. Moreover, the gradient $\nabla^a \tilde{f}(x)$ can be computed in $O(n^3)$ time.

Proof. The proof of this follows directly from the fact that we divide the possible sets into those containing i and those not containing i. Given this recursion, it is easy to see that the entire matrix of values of $P(x_1, \cdots, x_i, l)$ can be obtained for all values of i and l in $O(n^2)$ iterations. Moreover, given this matrix, we can obtain the expression of $\tilde{f}$ also in $O(n^2)$. □

The above results immediately provide an expression of the multilinear extension of the Facility Location function.

Corollary 7.5. *The multilinear extension corresponding of the Facility Location function* $f(X) = \sum_{i \in V} \max_{j \in X} s_{ij}$*, for a similarity matrix* s_{ij}*, can be expressed as*

$$\tilde{f}(x) = \sum_{i \in V} \sum_{l=1}^{n} s_{ij_i^l} x_{j_i^l} \prod_{m=1}^{l-1} (1 - x_{j_i^m}), \tag{7.19}$$

where $j_i^1, j_i^2, \cdots, j_i^n$ *denote the elements closest to* $i \in V$*, sorted in decreasing order by their similarity* s_{ij}*. The function* $\tilde{f}(x)$ *can be computed in* $O(n^2 \log n)$ *time, and likewise the alternate gradient.*

Proof. The expression for the multilinear extension follows immediately from Lemma 7.5 and 7.6. The extension can be computed by, for each $i \in V$, sorting the elements by s_{ij}, computing the products in one linear pass for each l, and then the sum over l in another pass. Repeating this for each $i \in V$ results in a running time of $O(n(n + n \log n))$.

We next consider the gradient $\nabla^a \tilde{f}(x)$. For simplicity, we focus on the max function, i.e $f(X) = \max_{i \in X} s_i$. Assume w.l.o.g. that $s_1 \geq s_2 \geq \cdots \geq s_n$. The gradient of this function is

$$\nabla_j^a \tilde{f}(x) = s_j \prod_{l=1}^{j-1} (1 - x_l) - \sum_{i=j+1}^{n} s_i x_i \prod_{l=1}^{i-1} (1 - x_l).$$

This follows since $\nabla_j^a \tilde{f}(x) = \tilde{f}(x|x_j = 1) - \tilde{f}(x)$, where $\tilde{f}(x|x_j = 1)$ is the expression of $\tilde{f}(x)$ with x_j set to be 1. The multilinear extension for the max-function is $\tilde{f}(x) = \sum_{i=1}^{n} s_i x_i \prod_{l=1}^{i-1} (1 - x_l)$, and hence,

$$\tilde{f}(x|x_j = 1) = \sum_{i=1}^{j-1} s_i x_i \prod_{l=1}^{i-1} (1 - x_l) + s_j \prod_{l=1}^{j-1} (1 - x_l).$$

Plugging both into the expression of the gradient, we get the above.

Then for every j, we precompute $M(x, j) = \prod_{l=1}^{j-1} (1 - x_l)$ and store it (this can be done in $O(n)$, and $O(n \log n)$ for sorting). Then

$$\nabla_j^a \tilde{f}(x) = s_j M(x, j) - \sum_{i=j+1}^{n} s_i x_i M(x, i). \tag{7.20}$$

Hence the entire alternate gradient can be computed in $O(n \log n)$ time for the max function, and correspondingly in $O(n^2 \log n)$ for facility location. □

Lemma 7.5 provides an expression for the partial alternate gradient (since it is useful for maximizing monotone functions). For the complete gradient $\nabla^a \tilde{f}(x)$, there is a similar expression. The complete gradient is required for non-monotone submodular maximization, when the facility location function is used together with a non-monotone submodular function.

We can rewrite the gradients for some other matroids too, using similar techniques as above. For example, consider partition matroids. Let $B_1, B_2, \cdots, B_m$ be m blocks of a partition of the ground set such that $B_i \cap B_j = \emptyset, \forall i, j$ and $\cup_i B_i = V$. Also let $d_1, \cdots, d_m$ be numbers such that $d_i \leq |B_i|, \forall i$. A set X is independent in a partition matroid if and only if $|X \cap B_i| \leq d_i, \forall i$. The weighted rank function corresponding to the partition matroid is,

$$\begin{aligned} f(X) &= \max\{w(Y) | \forall i, Y \cap B_i \subseteq X \cap B_i, |Y \cap B_i| \leq d_i\} \\ &= \max\{\sum_{i=1}^{m} w(Y_i), Y_i \subseteq X \cap B_i, |Y_i| \leq d_i\} \\ &= \sum_{i=1}^{m} \max\{w(Y_i), Y_i \subseteq X \cap B_i, |Y_i| \leq d_i\}. \end{aligned}$$

The last equality holds since the B_i's are disjoint. Hence, the weighted matroid rank function for a partition matroid is a sum of rank functions corresponding to uniform matroids defined over the subsets B_i. Hence Lemma 7.17 directly provides an expression for the multilinear extension.

Set Cover function. This function is widely used in applications, capturing notions of coverage [Lin, 2012]. Given a collection of sets $\{\mathcal{S}_1, \cdots, S_n\}$ and the universe $\mathcal{U} = \cup_i \mathcal{S}_i$, define $f(X) = w(\cup_{i \in X} \mathcal{S}_i)$, where w_j denotes the weight of item $j \in \mathcal{U}$. This setup can alternatively be expressed via a neighborhood function $\Gamma : 2^V \to 2^{\mathcal{U}}$ such that $\Gamma(X) = \cup_{i \in X} \mathcal{S}_i$. Then $f(X) = w(\Gamma(X))$. Let $\Gamma^{-1}(j) = \{i \in V : j \in \Gamma(i)\}$. Then the multilinear extension has a simple form:

Lemma 7.7. *The multilinear extension corresponding to the set cover function* $f(X) =$

$w(\Gamma(X))$ *is*

$$\tilde{f}(x) = \sum_{j \in \mathcal{U}} w_j [1 - R(x, j)], \tag{7.21}$$

where $R(x, j) = \prod_{i \in \Gamma^{-1}(j)} (1 - x_i)$. *The multilinear extension can be computed in* $O(n^2)$ *time. Moreover, the gradient* $\nabla_k^a \tilde{f}(x)$ *is*

$$\nabla_k^a \tilde{f}(x) = \sum_{j \notin \Gamma(k)} w_j R(x, j), \tag{7.22}$$

and the entire vector $\nabla^a \tilde{f}(x)$ *can be computed in* $O(n^2)$ *time.*

Proof. Again, we express this sum in terms of the weights w. In particular,

$$\begin{aligned}
\tilde{f}(x) &= \sum_{j \in \mathcal{U}} w_j \sum_{X : j \in \Gamma(X)} \prod_{s \in X} x_s \prod_{t \notin X} (1 - x_t) \\
&= \sum_{j \in \mathcal{U}} w_j \sum_{X : X \cap \Gamma^{-1}(j) \neq \emptyset} \prod_{s \in X} x_s \prod_{t \notin X} (1 - x_t) \\
&= \sum_{j \in \mathcal{U}} w_j [1 - \sum_{X : X \subseteq V \setminus \Gamma^{-1}(j)} \prod_{s \in X} x_s \prod_{t \notin X} (1 - x_t)] \\
&= \sum_{j \in \mathcal{U}} w_j [1 - \prod_{t \in \Gamma^{-1}(j)} (1 - x_t)]
\end{aligned}$$

where the last inequality follows from the fact that

$$\begin{aligned}
&\sum_{X : X \subseteq V \setminus \Gamma^{-1}(j)} \prod_{s \in X} x_s \prod_{t \in V \setminus X} (1 - x_t) = \\
&\prod_{u \in \Gamma^{-1}(j)} (1 - x_u) \sum_{X : X \subseteq V \setminus \Gamma^{-1}(j)} \prod_{s \in X} x_s \prod_{t \in \{V \setminus \Gamma^{-1}(j)\} \setminus X} (1 - x_t) \\
&= \prod_{t \in \Gamma^{-1}(j)} (1 - x_t)
\end{aligned}$$

The second part also directly follows from the first, as

$$\begin{aligned}\nabla^a_k \tilde{f}(x) &= \tilde{f}(x|x_k = 1) - \tilde{f}(x) \\ &= \sum_{j:k\in\Gamma^{-1}(j)} w_j[1 - R(x,j)] + \sum_{j:k\notin\Gamma^{-1}(j)} w_j - \tilde{f}(x) \\ &= \sum_{j:k\notin\Gamma^{-1}(j)} w_j R(x,j) \\ &= \sum_{j\notin\Gamma(k)} w_j R(x,j)\end{aligned}$$

In order to compute the vector $\nabla^a \tilde{f}(x)$, first we precompute $R(x,j)$ for all j. This can be done in $O(n^2)$ time. Then using the values of $R(x,j)$, we can compute $\nabla^a_k \tilde{f}(x), \forall k$ also in $O(n^2)$. □

The gradient $\nabla \tilde{f}(x)$ can be computed analogously.

Probabilistic Coverage Functions. A probabilistic generalization of covering functions of the form $f(X) = \sum_{i\in\mathcal{U}} w_i[1 - \prod_{j\in X}(1 - p_{ij})]$ has been used for summarization problems [El-Arini et al., 2009]. If p_{ij} is binary (either i covers j or not), we obtain a standard set cover function. The multilinear extension of probabilistic coverage functions is also efficiently computable

$$\tilde{f}(x) = \sum_{i\in\mathcal{U}} w_i[1 - \prod_{j\in V}(1 - p_{ij}x_j)]. \tag{7.23}$$

This form follows from Proposition 7.2 (below), since this is a pseudo-Boolean function.

Graph Cut functions. Graph cuts are a widely used class of functions. Their multilinear extension also a admits closed form representation. Graph cut functions are of the form

$$f(X) = \sum_{i\in X, j\notin X} s_{ij}. \tag{7.24}$$

Its multilinear extension is also easily expressed:

Lemma 7.8. *The multilinear extension of the graph cut function and its gradient have closed form expressions*

$$\tilde{f}(x) = \sum_{i,j \in V} s_{ij} x_i (1 - x_j), \;\; \nabla_i \tilde{f}(x) = \sum_{j \in V} s_{ij}(1 - 2x_j)$$

For asymmetric cut functions $f(X) = \sum_{i \in V, j \in X} s_{ij}$ *, the expressions are*

$$\tilde{f}(x) = \sum_{i,j \in V} s_{ij} x_j, \quad \nabla_i \tilde{f}(x) = \sum_{j \in V} s_{ij} \tag{7.25}$$

In both cases above, the multilinear extension and the corresponding gradient can be computed in $O(n^2)$ *time.*

Proof. We rewrite the multilinear extension in terms of the s_{ij} to obtain the quadratic polynomial

$$\tilde{f}(x) = \sum_{i,j \in V} s_{ij} \sum_{X : i \in X, j \notin X} \prod_{s \in X} x_s \prod_{t \notin X} (1 - x_t) \tag{7.26}$$

$$= \sum_{i,j \in V} s_{ij} x_s (1 - x_t) \tag{7.27}$$

We can similarly derive the expression for the asymmetric graph cut function and the gradients for both expressions. □

These quadratic functions have been widely used in computer vision.

A related function is a similarity-penalizing function: $f(X) = -\sum_{i,j \in X} s_{ij}$. This function has been used for encouraging diversity [Lin and Bilmes, 2011c, Lin, 2012].

Lemma 7.9. *The multilinear extension and the gradient for the function* $f(X) = -\sum_{i,j \in X} s_{ij}$ *are*

$$\tilde{f}(x) = -\sum_{i,j \in V} s_{ij} x_i x_j, \quad \nabla_i \tilde{f}(x) = -\sum_{j \in V} s_{ij} x_j \tag{7.28}$$

	Fac. Location [2]	Set Cover	Graph Cuts	Diversity I/ II	Concave over card.	Log-Det [3]
Multilinear Closed form	$O(n^2 \log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^3)$
Multilinear Sampling	$O(n^7 \log n)$	$O(n^6)$	$O(n^7)$	$O(n^7)$	$O(n^6)$	$O(n^8)$
Gradient Closed form	$O(n^2 \log n)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^3)$
Gradient Sampling	$O(n^7 \log n)$	$O(n^7)$	$O(n^8)$	$O(n^8)$	$O(n^7)$	$O(n^9)$

Table 7.3: Complexity of evaluating the multilinear extensions and their gradients for both the optimized closed forms given in this chapter and for sampling at high accuracy.

Proof. The lemma follows analogously to the case of graph cuts:

$$\tilde{f}(x) = -\sum_{i,j \in V} s_{ij} \sum_{X : i \in X, j \in X} \prod_{s \in X} x_s \prod_{t \notin X} (1 - x_t) \tag{7.29}$$

$$= -\sum_{i,j \in V} s_{ij} x_i x_j, \tag{7.30}$$

and similarly for the gradient. □

This function is often used with other coverage type functions (for example, graph cut or set cover) [Lin, 2012], and since the multilinear extension is commutative, the above forms provide closed form expressions for a number of objective functions that promote diversity and coverage.

Sparse Pseudo-Boolean functions. For graphical models, in particular in computer vision, set functions are often written as polynomials [Ishikawa, 2009]. Any set function can be written as a polynomial, $p_f(x) = \sum_{T \subseteq V} \alpha_T \prod_{i \in T} x_i$, where $x \in \{0,1\}^n$ is the characteristic vector of a set. In other words, $f(S) = \sum_{T \subseteq S} \alpha_T$. Submodular functions are a subclass of these polynomials. This representation directly gives the multilinear extension as the same polynomial, $\tilde{f}(x) = \sum_{T \subseteq V} \alpha_T \prod_{i \in T} x_i$, and is efficiently computable if the polynomial is *sparse*, i.e., has few nonzero coefficients α_T. This is the case for graph cut like functions above and for the functions considered in [Stobbe and Krause, 2012, Ishikawa, 2009]. This analogy is implicitly known, but we formalize it for completeness.

Proposition 7.2. *The polynomial representation is the multi-linear extension:* $\tilde{f}(x) = p_f(x)$.

Proof. Recall that the multilinear extension is equivalent to the expectation for a product distribution of Bernoulli random variables, $P(S) = \prod_{i \in S} x_i \prod_{j \notin S} x_j$. We see that

$$\tilde{f}(x) = \sum_{S \subseteq V} p(S) \sum_{T \subseteq S} \alpha_T \tag{7.31}$$

$$= \sum_{T \subseteq V} \alpha_T \sum_{S \supseteq T} p(S) \tag{7.32}$$

$$= \sum_{T \subseteq V} \alpha_T \sum_{S \supseteq T} p(T) p(S \setminus T \mid T) \tag{7.33}$$

$$= \sum_{T \subseteq V} \alpha_T \prod_{i \in T} x_i \Big(\sum_{A \subseteq V \setminus T} p(A) \Big) \tag{7.34}$$

$$= \sum_{T \subseteq V} \alpha_T \prod_{i \in T} x_i = p_f(x). \tag{7.35}$$

In the last step, we used that $p(A \mid T) = p(A)$ and that $\sum_{A \subseteq V'} P(A) = 1$ for all $V' \subseteq V$. □

Spectral functions. Diversity can also be encouraged via spectral regularizers [Das et al., 2012]. Given a positive definite matrix $S \in \mathbb{R}^{n \times n}$, define S_X to be the $|X| \times |X|$ sub-matrix of the rows and columns indexed by X. Any scalar function ψ whose derivative is operator-antitone defines a submodular function, $f(X) = \sum_{i=1}^{|X|} \psi(\lambda_i(S_X))$, by applying it to the eigenvalues of S_X [Friedland and Gaubert, 2011]. The resulting class of submodular functions includes the log determinants occurring in DPP inference [Gillenwater et al., 2012], and, more generally, a smoothed log-determinant function $f(X) = \log\det(S_X + \delta I_X) = \sum_{i=1}^{|X|} \log(\lambda_i(S_X) + \delta)$. It is monotone for $\delta \geq 1$, and has an efficiently computable soft-max extension that is similar to the multilinear extension [Gillenwater et al., 2012]. This extension is $\tilde{f}^s(x) = \log(\sum_{X \subseteq V} \exp(f(X)) \prod_{i \in X} x_i \prod_{j \notin X} (1 - x_j)$ and shares several desirable theoretical properties with the multilinear extension[4]. A related function that encourages diversity is $f(X) = -\sum_{i=1}^{|X|} (\lambda_i(S_X) - 1)^2$ [Das et al., 2012].

[4]The results in [Gillenwater et al., 2012] are shown only for $\delta = 0$, i.e log-determinant functions. However, it is easy to see that the soft max extension can be computed for any value of $\delta > 0$ efficiently

Note that $-\sum_{i=1}^{|X|} \lambda_i^2(S_X) = -\text{trace}(S_X^\top S_X) = -\sum_{i,j\in X} s_{ij}^2$. The multilinear extension of this function also takes a nice form.

Lemma 7.10. *The spectral function $f(X) = -\sum_{i=1}^{|X|}(\lambda_i(S_X) - 1)^2$ can also be directly expressed as*

$$f(X) = -\sum_{i,j\in X} s_{ij}^2 + 2\sum_{i\in X} s_{ii} + |X|. \tag{7.36}$$

The multilinear extension and its gradient are

$$\tilde{f}(x) = -\sum_{i,j\in V} s_{ij}^2 x_i x_j + \sum_{i\in V}(2s_{ii} + 1),$$
$$\nabla_i \tilde{f}(x) = -\sum_{j\in V} s_{ij}^2 x_j + 2s_{ii} + 1$$

Both expressions can be computed in $O(n^2)$ time.

Proof. The proof of this lemma follows from Lemma 7.9 and from the fact that the multi-linear extension of a modular function is its linear extension – i.e given a function $f(X) = \sum_{i\in X} s_i$, its multilinear extension is $\tilde{f}(x) = \sum_{i\in V} s_i x_i = \langle s, x\rangle$. □

Concave over modular functions. Finally, we consider functions of the form $f(A) = g(|A|)$ where g is a concave function. Such functions are submodular, and have simple extensions.

Lemma 7.11. *The multilinear extension of the concave over cardinality function $f(A) = g(|A|)$ is*

$$\tilde{f}(x) = \sum_{i=1}^{n} g(i) P(x_1, \cdots, x_n, i), \tag{7.37}$$

where

$$P(x_1, \cdots, x_n, i) = \sum_{Z\subseteq V, |Z|=i} \prod_{s\in Z} x_s \prod_{t\notin Z}(1 - x_t).$$

The term $P(x_1, \cdots, x_n, i)$ can be computed in linear time (excluding the computation of constants), and correspondingly $\tilde{f}(x)$ in $O(n^2)$.

Proof. The proof of this Lemma follows directly from definition of the multilinear extension, and from Lemmas 7.5 and 7.6. □

One may generalize this to sums $f(X) = \sum_i g_i(m_i(X))$ of concave over modular functions, where the g_i are concave and the m_i are modular. This class of functions has a natural concave extension: $\tilde{f}(x) = \sum_i g_i(\langle x, m_i \rangle)$.

Given expressions for the functions above, we can also handle weighted combinations $f(X) = \sum_i \lambda_i f_i(X)$, since its multilinear extension is $\tilde{f}(x) = \sum_i \lambda_i \tilde{f}_i(x)$. In the following sections, we briefly describe relaxation algorithms and rounding schemes for maximization.

7.4.1 Monotone Maximization

We first investigate monotone submodular maximization subject to matroid independence constraints $\mathcal{I}$. The technique for maximizing the multilinear extension is the continuous greedy algorithm [Vondrák, 2008], which is a slight modification of the Frank-Wolfe algorithm [Frank and Wolfe, 1956], with a fixed step size.

- Find $h^t = \text{argmax}_{h' \in \mathcal{P}_\mathcal{C}} \langle h', \nabla^a \tilde{f}(x^t) \rangle$.
- $x^{t+1} = x^t + \delta h^t$, with the step size $\delta = 1/n^2$.

Here ∇^a is the alternate gradient. In each iteration, the algorithm takes a step $x^{t+1} = x^t + \delta h^t$ (with step size $\delta = 1/n^2$) in the direction $h^t = \text{argmax}_{h' \in \mathcal{P}_\mathcal{C}} \langle h', \nabla^a \tilde{f}(x^t) \rangle$ best aligned with the alternate gradient. This *continuous greedy* procedure terminates in $O(n^2)$ iterations, after which we are guaranteed to obtain a point x such that $\tilde{f}(x) \geq (1 - 1/e)\tilde{f}(x^*)$ [Calinescu et al., 2011, Vondrák, 2007]. Moreover, using the pipage rounding technique (in particular, the deterministic variant [Vondrák, 2008]) ensures that we can round the continuous solution to a set in $O(n^2)$ function calls.

A naïve computation of the generic multilinear extension in Eqn. (1.14) or its gradient takes exponential time. To compute these in polynomial time, we can use sampling. For

obtaining an accuracy better than $1/n^2$, we need $O(n^5)$ samples for the multilinear extension or for each coordinate of its gradient [Vondrák, 2008, Vondrák, 2007]. This implies a complexity of $O(n^6)$ function evaluations for the gradient and $O(n^5)$ function evaluations for the extension itself, thus implying the algorithm's complexity as $O(n^8 T_{\nabla f})$, where $T_{\nabla f}$ is the time of evaluating the gain of f. For facility location, this means a running time of $O(n^9 \log n)$, and for set cover functions $O(n^9)$. The specialized expressions in Section 7.4 however lead to algorithms that run several orders of magnitude faster. With $O(n^2)$ iterations, the time becomes $O(n^2 T_{\nabla \tilde{f}})$, where $\nabla \tilde{f}$ is the time to compute the gradient of $\tilde{f}$. Table 7.3 compares the function evaluation times for some practically very useful submodular functions. Moreover, we can use mixtures of these submodular functions, each with efficiently computable multilinear extensions, and compute the resulting multilinear extension also efficiently. While this is still slower than the accelerated greedy algorithm [Minoux, 1978], it gains power for more complex constraints, such as matroid independence constraints, where the discrete greedy algorithm only achieves an approximation factor of $1/2$, whereas the continuous greedy obtains at least a $1 - 1/e$ factor. Similarly, the continuous greedy algorithm achieves a $1 - 1/e$ approximation guarantee for multiple knapsack constraints [Kulik et al., 2009], while the discrete greedy techniques do not yet have such guarantees. Hence, the formulations above make it possible to use the optimal theoretical results with a more manageable running time.

7.4.2 Non-Monotone Maximization

In the non-monotone setting, we must find a local optimum of the multilinear extension. We could use, for example, a Frank-Wolfe style algorithm [Frank and Wolfe, 1956] and run it until it converges to a local optimum. It is easy to see that at convergence, x satisfies $\langle \nabla \tilde{f}(x), y - x \rangle \leq 0, \forall y \in \mathcal{P}_{\mathcal{C}}$ and is a local optimum. Practically, this would mean checking if $\operatorname{argmax}_{y \in \mathcal{P}_{\mathcal{C}}} \langle y, \nabla \tilde{f}(x) \rangle = x$. For simple or no constraints, we could also use a method like L-BFGS. Running this procedure twice, we are guaranteed to obtain a 0.25 approximate solution [Chekuri et al., 2011]. This procedure works for any down-monotone constraint $\mathcal{C}$. Moreover, this procedure with a slightly different extension has been successfully applied in

practice to MAP inference with determinantal point processes [Gillenwater et al., 2012].

A generic rounding strategy for submodular maximization problems was given by [Chekuri et al., 2011], and works for a large class of constraints (including matroid, knapsack constraints, and a combination thereof). Without constraints, this amounts to sampling a set by a distribution based on the continuous solution x — it will satisfy $\mathbf{E}_{X\sim x} f(X) = \tilde{f}(x)$. In practice, however, this may not work well. Since the multilinear extension is linear in any coordinate (holding the other ones fixed), a simpler co-ordinate ascent scheme of choosing the better amongst 0 or 1 for any fractional co-ordinate will guarantee a deterministic procedure of obtaining an integral solution no worse than the continuous one.

The above algorithms and rounding techniques offer a general and optimal framework, even for many complex constraints. Moreover, many of the best algorithms for non-monotone submodular maximization are based on the multilinear extension. For example, the best known algorithm for cardinality constrained non-monotone submodular maximization [Buchbinder et al., 2014] uses a continuous double greedy algorithm on the multilinear extension. However, the practical utility of those algorithms is heavily impaired by computational complexity. In fact, non-monotone functions even require $O(n^7)$ samples [Chekuri et al., 2011]. For DPPs, [Gillenwater et al., 2012] used an extension that is practical and close to the multilinear extension. Since they do not use the multilinear extension, the above rounding schemes do not imply the same approximation bounds as for the multilinear extension, leaving the worst-case approximation quality unknown. The expressions we show above use the multilinear extension and maintain its benefits, demonstrating that for many functions of practical interest, sampling, and hence extremely high complexity, is not necessary. This observation is a step from theory into practice, and allows for the improved approximations to be used in practice.

7.4.3 Integrality Gaps

Surprisingly, the multilinear extension has an integrality gap of 1 for a number of constraints including the matroid and cardinality constraints, since it is easy to round it exactly (using

say, the pipage rounding or contention resolution schemes [Calinescu et al., 2011, Chekuri et al., 2011]). The concave extension however, can have integrality gaps arbitrarily close to $e/(e-1)$ even for simple matroids [Vondrák, 2007]. Hence, even though it is possible to exactly optimize it in certain cases (for example, for weighted matroid rank functions), the rounding only guarantees a $1 - 1/e$ approximation factor.

7.5 Difference Of Submodular (DS) Optimization

Finally, we investigate minimizing the differences between submodular functions. Given submodular functions f and g, we consider the following minimization problem: $\min_{X \in \mathcal{C}} \big(f(X) - g(X)\big)$. In fact, any set function can be represented as a difference between two non-negative monotone submodular functions [Narasimhan and Bilmes, 2005, Iyer and Bilmes, 2012b]. In the unconstrained setting, $\mathcal{C} = 2^V$. A natural continuous relaxation (not necessarily convex) is $\tilde{h}(x) = \breve{f}(x) - \breve{g}(x)$. The continuous problem is a DC programming problem, and can be addressed (often very efficiently) using the convex-concave procedure [Yuille and Rangarajan, 2003]. Moreover, thanks to the special structure of the Lovász extension, there exists a simple rounding scheme for the unconstrained version.

Lemma 7.12. *Given submodular functions f and g, and a continuous vector x, there exists a $\theta \in (0,1)$ such that $f(X_\theta) - g(X_\theta) \geq \breve{f}(x) - \breve{g}(x)$, where $X_\theta = \{x \geq \theta\}$. Moreover, the integrality gap of $\tilde{h}(x)$ (in the unconstrained setting) is equal to 1.*

Proof. Recall that given a point $z \in [0,1]^n$, we can find a chain of sets $\emptyset = Z_0 \subset Z_1 \subset Z_2 \subset \cdots \subset Z_k$ corresponding to z, such that $z = \sum_{j=1}^{k} \lambda_j 1_{Z_j}$. Then the Lovász extension can be written as $\breve{f}(z) = \sum_{j=1}^{k} \lambda_j f(Z_j)$. This chain is independent of the function f and hence, given functions f and g, we have that

$$\tilde{h}(z) = \sum_{j=1}^{k} \lambda_j h(Z_j) \tag{7.38}$$

It is then easy to see that one of $h(Z_j)$ for $j = 1, 2, \cdots, k$ must have a value less than or equal to $\tilde{h}(z)$, thus completing the proof. □

The above lemma shows that in at most $O(n)$, we can round the continuous solution without any loss. Unfortunately, these results do not seem to extend straightforwardly to combinatorial constraints. Although the relaxed difference of convex optimization problem can itself be solved via the convex-concave procedure if the polytope $\mathcal{P}_{\mathcal{C}}$ corresponding to the constraints can be characterized efficiently, the θ-rounding procedure no longer retains any guarantees. However, a procedure like threshold rounding might still provide a feasible solution if the constraints are up-monotone, and taking the best amongst the feasible rounded sets might still work well in practice.

7.6 Discussion

In this work, we have offered a unifying view on continuous relaxation methods for submodular optimization. For minimization problems with various constraints, we provide a generic rounding strategy with new approximation bounds and matching integrality gaps. For maximization, we summarize efficiently computable expressions for many practically interesting submodular functions. This is a useful step towards transferring optimal theoretical results to real-world applications. An interesting question remains whether there exist improved sampling schemes for cases where the multilinear extension is complex. In this chapter, we propose relaxation based algorithms for Problems 1, 2 and 3. An open problem is whether these techniques can be extended to Problems 4 and 5 (i.e. SCSC and SCSK as well). Note that both these problems are special cases of Problems 1 and 2. The main difficulty, however, is to express the constraints in this setting as linear programs. In the cases where the constraints can be expressed through a polynomial number of linear inequalities, however, we could use the techniques proposed in this chapter to get guarantees for both these problems. Another open problem from this chapter, is to investigate the role of complexity parameters like curvature etc. in the approximation bounds. Currently our bounds do not depend on these parameters.

Chapter 8

CONCLUSIONS

This thesis has presented unifying approaches for submodular optimization, while focussing on scalability to large scale data subset selection applications. We show how these approaches unify several existing algorithms, and provide new algorithms for many forms of submodular optimization, including submodular minimization, submodular maximization, difference of submodular optimization, and submodular optimization subject to submodular constraints. We study theoretical properties of these algorithms, as well as the hardness of the underlying optimization problems. We also highlight novel theoretical characterizations, which provide better connections between theory and practice. Finally, we demonstrate the applicability of our algorithms on synthetic data as well as several real world problems including image correspondence, summarization, feature subset selection and limited vocabulary corpus selection etc. We conclude with some open problems.

- In this dissertation (particularly chapters 3 and 4), we investigated several theoretical constructs like the curvature, monotonicity ratio and submodularity ratio, which offer improved results for submodular minimization and maximization. These bounds provide the first step in connecting theory and practice. However, there is still a big gap. Several submodular functions obtain worst case factors close to 1 for these optimization problems, which contrasts the weaker worst case guarantees even if we consider these parameters. An open problem is to investigate other important constructs connected to submodular optimization, which could shed more light into the gap between the approximation factors observed in practice and the worst case ones.

- Another possible research direction could be to investigate the performance of our

algorithms for subclasses of submodular functions, which occur naturally in practice. Note that the worst case approximation bounds hold for several contrived subclasses of submodular functions. Correspondingly, a very important problem is to investigate classes of submodular functions which occur more naturally in applications, and study the worst case behavior for these subclasses of functions.

- In Chapter 4, we unified several submodular maximization schemes from an algorithmic perspective. However, we do not offer new theoretical insights into the guarantees. It might be useful to consider a unifying theoretical perspective, and a proof framework to derive many of the approximation guarantees for these problems. This could help in providing improved theoretical approximation guarantees for many open problems in submodular maximization.

- Most of the algorithms proposed in this work were based on first order (i.e modular) upper and lower bounds, which we have called the semigradients of a submodular function. This is akin to the gradient based algorithms in convex optimization. A natural question is whether second order methods could provide improved guarantees for several of these problems. In particular, one could construct second order (possibly graph-cut) based upper and lower bounds, which might yield better theoretical results.

- Finally, we note that results presented in Chapter 7 were worst case polynomial guarantees, and do not seem to be dependent on the curvature. An interesting open problem, is if these approximation bounds could be naturally extended to include the curvature of a submodular function.

The focus of this thesis was primarily theoretical, with the aim of characterizing a unifying class of algorithms, approximation bounds and hardness results for various flavors of submodular optimization. In every case, we motivated these applications by several real world applications, and also investigated many of these applications in the experiments section of

each chapter. As future work, we would like to investigate these algorithms in many other real world applications, which motivated a lot of this work.

BIBLIOGRAPHY

W. Y. Adams, H. Su, and L. Fei-Fei. Efficient euclidean projections onto the intersection of norm balls. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 433–440, 2012.

S. Ahmed and A. Atamtürk. Maximizing a class of submodular utility functions. *Math. Program., Ser. A*, 2009.

N. Alon and J. H. Spencer. The probabilistic method, 2nd ed. *Wiley-Interscience, New York*, 2000.

A. Atamtürk and V. Narayanan. The submodular knapsack polytope. *Discrete Optimization*, 2009.

I. Averbakh and O. Berman. Categorized bottleneck-minisum path problems on networks. *Operations Research Letters*, 16:291–297, 1994.

F. Bach. Learning with Submodular functions: A convex Optimization Perspective (updated version). *Arxiv*, 2013.

A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. *In SODA*, 2014.

A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of Principles of Database Systems*, pages 155–166. ACM, 2012.

E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete Applied Math.*, 123(1–3):155 – 225, 2002. ISSN 0166-218X. doi: 10.1016/S0166-218X(01)00341-9. URL `http://www.sciencedirect.com/science/article/pii/S0166218X01003419`.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Y. Boykov and M. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, 2001.

Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *TPAMI*, 26(9):1124–1137, 2004.

N. Brenner, S. Strong, R. Koberle, W. Bialek, and R. Steveninck. Synergy in a neural code. *Neural Computation*, 12(7):1531–1552, 2000.

N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. A tight (1/2) linear-time approximation to unconstrained submodular maximization. *In FOCS*, 2012.

N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. Submodular maximization with cardinality constraints. *In SODA*, 2014.

K. Byrnes. Maximizing general set functions by submodular decomposition. *Arxiv preprint arXiv:0906.0120*, 2009.

T. S. Caetano, J. J. McAuley, L. Cheng, Q. V. Le, and A. J. Smola. Learning graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(6): 1048–1058, 2009.

G. Calinescu, C. Chekuri, M. Pal, and J. Vondrák. Maximizing a monotone submodular function under a matroid constraint. *IPCO*, 2007.

G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

D. Chakrabarty, P. Jain, and P. Kothari. Provable submodular minimization using wolfe's algorithm. *arXiv preprint arXiv:1411.0095*, 2014.

A. Chambolle and J. Darbon. On total variation minimization and surface evolution using parametric maximum flows. *Int. Journal of Computer Vision*, 84(3):288–307, 2009.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 2011.

C. Chekuri, J. Vondrák, and R. Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *STOC*, 2011.

E. Chlamtac and M. Tulsiani. Convex relaxations and integrality gaps. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 139–169. Springer, 2012.

M. Conforti and G. Cornuejols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the Rado-Edmonds theorem. *Discrete Applied Mathematics*, 7(3):251–274, 1984.

W. H. Cunningham. Decomposition of submodular functions. *Combinatorica*, 3(1): 53–68, 1983.

A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *ICML*, 2011.

A. Das, A. Dasgupta, and R. Kumar. Selecting diverse features via spectral regularization. In *NIPS*, 2012.

J. Djolonga and A. Krause. From map to marginals: Variational inference in bayesian submodular models. In *Neural Information Processing Systems (NIPS)*, 2014.

H. Du, W. Wu, W. Lee, Q. Liu, Z. Zhang, and D.-Z. Du. On minimum submodular cover with submodular cost. *Journal of Global Optimization*, 50(2):229–234, 2011.

S. Dughmi. Submodular functions: Extensions, distributions, and algorithms. a survey. *arXiv preprint arXiv:0912.0322*, 2009.

J. Edmonds. Submodular functions, matroids and certain polyhedra. *Combinatorial structures and their Applications*, 1970.

K. El-Arini, G. Veda, D. Shahaf, and C. Guestrin. Turning down the noise in the blogosphere. In *KDD*, 2009.

U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 1998.

U. Feige, V. Mirrokni, and J. Vondrák. Maximizing non-monotone submodular functions. *SIAM J. COMPUT.*, 40(4):1133–1155, 2011a.

U. Feige, V. Mirrokni, and J. Vondrák. Maximizing non-monotone submodular functions. *SIAM J. COMPUT.*, 40(4):1133–1155, 2011b.

M. Feldman, J. Naor, and R. Schwartz. A unified continuous greedy algorithm for submodular maximization. In *FOCS*, 2011.

Y. Filmus. Inequalities on submodular functions via term rewriting. *Information Processing Letters*, 2013.

M. Fisher, G. Nemhauser, and L. Wolsey. An analysis of approximations for maximizing submodular set functions—ii. *Polyhedral combinatorics*, pages 73–87, 1978.

L. Fleischer and S. Iwata. A push-relabel framework for submodular function minimization and applications to parametric optimization. *Discrete Applied Mathematics*, 131 (2):311–322, 2003.

A. Frank. An algorithm for submodular functions on graphs. *North-Holland Mathematics Studies*, 66:97–120, 1982.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 1956.

S. Friedland and S. Gaubert. Submodular spectral functions of principal submatrices of a hermitian matrix, extensions and applications. *Linear Algebra and its Applications*, 2011.

S. Fujishige. On the subdifferential of a submodular function. *Mathematical programming*, 29(3):348–360, 1984a.

S. Fujishige. Submodular systems and related topics. *Mathematical Programming at Oberwolfach II*, pages 113–131, 1984b.

S. Fujishige. Theory of submodular programs: A fenchel-type min-max theorem and subgradients of submodular functions. *Mathematical programming*, 29(2):142–155, 1984c.

S. Fujishige. *Submodular functions and optimization*, volume 58. Elsevier Science, 2005.

S. Fujishige and S. Isotani. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7:3–17, 2011.

S. Fujishige and H. Narayanan. Polyhedrally tight set functions and discrete convexity. 2005.

A. C. Gallagher, D. Batra, and D. Parikh. Inference for order reduction in markov random fields. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1857–1864. IEEE, 2011.

J. Garofolo, F. Lamel, L., J. W., Fiscus, D. Pallet, and N. Dahlgren. Timit, acoustic-phonetic continuous speech corpus. In *DARPA*, 1993.

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741, 1984.

J. Gillenwater. *Approximate Inference for Determinantal Point Processes*. PhD thesis, University of Pennsylvania, 2014.

J. Gillenwater, A. Kulesza, and B. Taskar. Near-optimal MAP inference for determinantal point processes. In *NIPS*, 2012.

G. Goel, C. Karande, P. Tripathi, and L. Wang. Approximability of combinatorial problems with multi-agent submodular cost functions. In *FOCS*, 2009.

G. Goel, P. Tripathi, and L. Wang. Combinatorial problems with discounted price functions in multi-agent systems. In *FSTTCS*, 2010.

M. Goemans, N. Harvey, S. Iwata, and V. Mirrokni. Approximating submodular functions everywhere. In *SODA*, pages 535–544, 2009.

B. Goldengorin, G. Tijssen, and M. Tso. *The maximization of submodular functions: Old and new proofs for the correctness of the dichotomy algorithm*. University of Groningen, 1999.

M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

M. Grotschel, L. Lovász, and A. Schrijver. Geometric methods in combinatorial optimization. In *Silver Jubilee Conf. on Combinatorics*, pages 167–183, 1984.

A. Guillory and J. Bilmes. Interactive submodular set cover. *In ICML*, 2010.

A. Guillory and J. Bilmes. Simultaneous learning and covering with adversarial noise. In *ICML*, 2011.

J. He, H. Tong, Q. Mei, and B. Szymanski. Gender: A generic diversified ranking algorithm. In *Neural Information Processing Systems (NIPS)*, pages 1151–1159, 2012.

L. Heng, A. Gotovos, A. Krause, and M. Pollefeys. Efficient visual exploration and coverage with a micro aerial vehicle in unknown environments. 2014.

D. Hunter and K. Lange. A tutorial on MM algorithms. *The American Statistician*, 2004.

W. Iba, J. Wogulis, and P. Langley. Trading off simplicity and coverage in incremental concept learning. In *Proceedings of Fifth International Conference on Machine Learning*, pages 73–79, 1988.

H. Ishikawa. Higher-order clique reduction in binary graph cut. In *CVPR*, 2009.

S. Iwata. A fully combinatorial algorithm for submodular function minimization. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 915–919. Society for Industrial and Applied Mathematics, 2002.

S. Iwata. A faster scaling algorithm for minimizing submodular functions. *SIAM Journal on Computing*, 32(4):833–840, 2003.

S. Iwata and K. Nagano. Submodular function minimization under covering constraints. In *In FOCS*, pages 671–680. IEEE, 2009.

S. Iwata and J. Orlin. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1230–1237. Society for Industrial and Applied Mathematics, 2009.

S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM (JACM)*, 48(4):761–777, 2001.

R. Iyer and J. Bilmes. The submodular Bregman and Lovász-Bregman divergences with applications. In *NIPS*, 2012a.

R. Iyer and J. Bilmes. Algorithms for approximate minimization of the difference between submodular functions, with applications. *In UAI*, 2012b.

R. Iyer and J. Bilmes. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *NIPS*, 2013.

R. Iyer and J. Bilmes. Near Optimal algorithms for constrained submodular programs with discounted cooperative costs. *NIPS workshop on Discrete Optimization and Machine Learning*, 2014a.

R. Iyer and J. Bilmes. Submodular Point Processes. *NIPS workshop on Discrete Optimization and Machine Learning*, 2014b.

R. Iyer, S. Jegelka, and J. Bilmes. Mirror descent like algorithms for submodular optimization. *NIPS Workshop on Discrete Optimization in Machine Learning (DISCML)*, 2012.

R. Iyer, S. Jegelka, and J. Bilmes. Curvature and Optimal Algorithms for Learning and Minimizing Submodular Functions . In *NIPS*, 2013a.

R. Iyer, S. Jegelka, and J. Bilmes. Fast Semidifferential based Submodular function optimization. In *ICML*, 2013b.

R. Iyer, S. Jegelka, and J. Bilmes. Monotone Closure of Relaxed Constraints in Submodular Optimization: Connections Between Minimization and Maximization. In *UAI*, 2014.

S. Jegelka and J. Bilmes. Cooperative cuts: Graph cuts with submodular edge weights. Technical report, Technical Report TR-189, Max Planck Institute for Biological Cybernetics, 2010.

S. Jegelka and J. Bilmes. Online submodular minimization for combinatorial structures. *ICML*, 2011a.

S. Jegelka and J. Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *Computer Vision and Pattern Recognition (CVPR)*, 2011b.

S. Jegelka and J. A. Bilmes. Notes on graph cuts with submodular edge weights. In *Neural Information Processing Society (NIPS) Workshop*, Vancouver, Canada, December 2009. Workshop on Discrete Optimization in Machine Learning: Submodularity, Sparsity & Polyhedra (DISCML).

S. Jegelka and J. A. Bilmes. Approximation bounds for inference using cooperative cuts. In *ICML*, 2011c.

S. Jegelka and J. A. Bilmes. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR*, 2011d.

S. Jegelka, H. Lin, and J. Bilmes. On fast approximate submodular minimization. *In NIPS*, 2011.

S. Jegelka, F. Bach, and S. Sra. Reflections for user-friendly submodular minimization. In *NIPS*, 2013a.

S. Jegelka, A. Kapoor, and E. Horvitz. An interactive approach to solving correspondence problems. *International Journal of Computer Vision*, pages 1–10, 2013b.

Y. Kawahara and T. Washio. Prismatic algorithm for discrete dc programming problems. In *NIPS*, 2011.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer Verlag, 2004.

A. K. Kelmans and B. Kimelfeld. Multiplicative submodularity of a matrix's principal minor as a function of the set of its rows and some combinatorial applications. *Discrete Mathematics*, 44(1):113–116, 1983.

D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, 2003.

E. B. Khalil, B. Dilkina, and L. Song. Scalable diffusion-aware optimization of network topology. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1226–1235. ACM, 2014.

K. Kirchhoff and J. Bilmes. Submodularity for data selection in machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, October 2014.

Y. Kobayashi. The complexity of maximizing the difference of two matroid rank functions. 2014.

R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the second international conference on knowledge discovery and data mining*, volume 7, 1996.

V. Kolmogorov and C. Rother. Minimizing nonsubmodular functions with graph cuts–a review. *IEEE TPAMI*, 29(7):1274–1279, 2007.

V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE TPAMI*, 26(2):147–159, 2004.

C. Koufogiannakis and N. Young. Greedy δ-approximation algorithm for covering with arbitrary constraints and submodular cost. *Algorithmica*, 2013.

A. Krause. *Optimizing Sensing: Theory and Applications*. PhD thesis, Carnegie Mellon University, 2008.

A. Krause. SFO: A toolbox for submodular function optimization. *JMLR*, 11:1141–1144, 2010.

A. Krause and C. Guestrin. A note on the budgeted maximization on submodular functions. Technical Report CMU-CALD-05-103, Carnegie Mellon University, 2005a.

A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of Uncertainity in Artificial Intelligence*. UAI, 2005b.

A. Krause and C. Guestrin. A note on the budgeted maximization of submodular functions, 2005c. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.3314`.

A. Krause and C. Guestrin. Optimizing sensing: From water to the web. Technical report, DTIC Document, 2009.

A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *IPSN*, 2006.

A. Krause, B. McMahan, C. Guestrin, and A. Gupta. Robust submodular observation selection. *Journal of Machine Learning Research (JMLR)*, 9:2761–2801, 2008a.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9:235–284, 2008b.

A. Kulesza and B. Taskar. Determinantal point processes for machine learning. *arXiv preprint arXiv:1207.6083*, 2012.

A. Kulik, H. Shachnai, and T. Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *SODA*, 2009.

M. J. Kusner. Approximately adaptive submodular maximization. *NIPS workshop on Discrete Optimization and Machine Learning*, 2014.

E. Lawler and C. Martel. Computing maximal "polymatroidal" network flows. *Mathematics of Operations Research*, 7(3):334–347, 1982.

J. Lee, V. Mirrokni, V. Nagarajan, and M. Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *STOC*, pages 323–332. ACM, 2009a.

J. Lee, M. Sviridenko, and J. Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *In APPROX*, 2009b.

J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

H. Lin. *Submodularity in Natural Language Processing: Algorithms and Applications*. PhD thesis, University of Washington, Dept. of EE, 2012.

H. Lin and J. Bilmes. How to select a good training-data subset for transcription: Submodular active selection for sequences. In *Interspeech*, 2009.

H. Lin and J. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. *In NAACL*, 2010.

H. Lin and J. Bilmes. Optimal selection of limited vocabulary speech corpora. In *Interspeech*, 2011a.

H. Lin and J. Bilmes. A class of submodular functions for document summarization. In *The 49th Meeting of the Assoc. for Comp. Ling. Human Lang. Technologies (ACL/HLT-2011)*, Portland, OR, June 2011b.

H. Lin and J. Bilmes. A class of submodular functions for document summarization. *In ACL*, 2011c.

H. Lin and J. Bilmes. Learning mixtures of submodular shells with application to document summarization. In *UAI*, 2012.

L. Lovász. Submodular functions and convexity. *Mathematical Programming*, 1983.

O. Macchi. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, pages 83–122, 1975.

T. Maheara and K. Murota. A framework of discrete dc programming by discrete convex analysis. 2014.

S. T. McCormick. Submodular function minimization. *Discrete Optimization*, 12: 321–391, 2005.

G. McLachlan and T. Krishnan. *The EM algorithm and extensions*. New York, 1997.

M. Minoux. Accelerated greedy algorithms for maximizing submodular set functions. *Optimization Techniques*, pages 234–243, 1978.

R. C. Moore and W. Lewis. Intelligent selection of language model training data. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 220–224. Association for Computational Linguistics, 2010.

K. Murota. *Discrete Convex Analysis*. Mathematical Programming, 2003.

K. Nagano and Y. Kawahara. Structured convex optimization under submodular constraints. In *Proc. UAI*, 2013.

K. Nagano, Y. Kawahara, and K. Aihara. Size-constrained submodular minimization through minimum norm base. In *ICML*, 2011.

M. Narasimhan and J. Bilmes. A submodular-supermodular procedure with applications to discriminative structure learning. In *UAI*, 2005.

G. Nemhauser and L. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

M. Niepert, P. Domingos, and J. Bilmes. Generalized conditional independence and decomposition cognizant curvature: Implications for function optimization. *NIPS workshop on Discrete Optimization and Machine Learning*, 2014.

E. Nikolova. Approximation algorithms for offline risk-averse combinatorial optimization, 2010.

A. S. Ogale and Y. Aloimonos. Shape and the stereo correspondence problem. *International Journal of Computer Vision*, 65(3):147–162, 2005.

J. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.

C. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.

R. K. Pasumarthi, R. Narayanam, and B. Ravindran. Targeted influence maximization through a social network. *Discrete Optimization in Machine Learning, NIPS Workshop*, 2014.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2nd printing edition, 1988.

S. Rajagopalan and V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):525–540, 1998.

C. Reed and Z. Ghahramani. Scaling the indian buffet process via submodular maximization. *arXiv preprint arXiv:1304.3285*, 2013.

R. Rockafellar. *Convex analysis*, volume 28. Princeton Univ Pr, 1997.

J. Rousu and J. Shawe-Taylor. Efficient computation of gapped substring kernels on large alphabets. *Journal of Machine Learning Research*, 6(2):1323, 2006.

S. Sahni. Computationally related problems. *SIAM Journal on Computing*, 3(4): 262–279, 1974.

A. Schäffer. Simple local search problems that are hard to solve. *SIAM journal on Computing*, 20:56, 1991.

A. Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Verlag, 2003.

I. Simon, N. Snavely, and S. Seitz. Scene summarization for online image collections. In *ICCV*, 2007.

P. Stobbe. *Convex Analysis for Minimizing and Learning Submodular Set Functions*. PhD thesis, California Institute of Technology, 2013.

P. Stobbe and A. Krause. Efficient minimization of decomposable submodular functions. In *NIPS*, 2010.

P. Stobbe and A. Krause. Learning fourier sparse set functions. In *AISTATS*, 2012.

M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

Z. Svitkina and L. Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS*, pages 697–706, 2008.

L. Trevisan. Inapproximability of combinatorial optimization problems. *The Computing Research Repository*, 2004.

S. Tschiatschek, R. Iyer, H. Wei, and J. Bilmes. Learning Mixtures of Submodular Functions for Image Collection Summarization. In *Neural Information Processing Systems (NIPS)*, 2014.

V. V. Vazirani. *Approximation algorithms*. springer, 2004.

J. Vondrák. *Submodularity in combinatorial optimization*. PhD thesis, Charles University, 2007.

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

J. Vondrák. Submodularity and curvature: the optimal algorithm. *RIMS Kokyuroku Bessatsu*, 23, 2010.

M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305, 2008.

P.-J. Wan, G. Calinescu, X.-Y. Li, and O. Frieder. Minimum-energy broadcasting in static ad hoc wireless networks. *Wireless Networks*, 8:607–617, 2002.

P.-J. Wan, D.-Z. Du, P. Pardalos, and W. Wu. Greedy approximations for minimum submodular cover with submodular cost. *Computational Optimization and Applications*, 45(2):463–474, 2010.

K. Wei, Y. Liu, K. Kirchhoff, and J. Bilmes. Using document summarization techniques for speech data subset selection. In *NAACL-HLT*, 2013.

K. Wei, R. Iyer, and J. Bilmes. Fast multi-stage submodular maximization. In *ICML*, 2014a.

K. Wei, Y. Liu, K. Kirchhoff, C. Bartels, and J. Bilmes. Submodular subset selection for large-scale speech training data. *Proceedings of ICASSP, Florence, Italy*, 2014b.

L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

H. Xiong, D. Zhang, G. Chen, L. Wang, and V. Gauthier. Crowdtasker: Maximizing coverage quality in piggyback crowdsensing under budget constraint. 2014.

A. Yuille and A. Rangarajan. The concave-convex procedure (CCCP). *In NIPS*, 2002.

A. Yuille and A. Rangarajan. The concave-convex procedure. *Neural Computation*, 15 (4):915–936, 2003.

VITA

Rishabh Iyer earned a Bachelor of Technology in Electrical Engineering from the Indian Institute of Technology, Bombay in 2011. At the University of Washington he earned a Master of Science in Electrical Engineering in 2013 and Doctor of Philosophy in Electrical Engineering in 2015.