Fast Semidifferential-based Submodular Function Optimization

Rishabh Iyer RKIYER@U.WASHINGTON.EDU
University of Washington, Seattle, WA 98195, USA

Stefanie Jegelka STEFJE@EECS.BERKELEY.EDU
University of California, Berkeley, CA 94720, USA

Jeff Bilmes BILMES@U.WASHINGTON.EDU
University of Washington, Seattle, WA 98195, USA

Abstract

We present a practical and powerful new framework for both unconstrained and constrained submodular function optimization based on discrete semidifferentials (sub- and super-differentials). The resulting algorithms, which repeatedly compute and then efficiently optimize submodular semigradients, offer new and generalize many old methods for submodular optimization. Our approach, moreover, takes steps towards providing a unifying paradigm applicable to both submodular minimization and maximization, problems that historically have been treated quite distinctly. 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 minimization and maximization, and show that many state-of-the-art maximization algorithms are special cases. Lastly, we complement our theoretical analyses with supporting empirical experiments.

1. Introduction

In this paper, we address minimization and maximization problems of the following form:

$$\text{Problem 1: } \min_{X \in \mathcal{C}} f(X), \qquad \text{Problem 2: } \max_{X \in \mathcal{C}} f(X)$$

where $f : 2^V \rightarrow \mathbb{R}$ 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. A function $f : 2^V \rightarrow \mathbb{R}$ is said to be *submodular* (Fujishige, 2005) if for all subsets $S, T \subseteq V$, it holds that $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$. Defining $f(j|S) \triangleq f(S \cup j) - f(S)$ as the gain of $j \in V$ with respect to $S \subseteq V$, then f is submodular if and only if $f(j|S) \geq f(j|T)$ for all $S \subseteq T$ and $j \notin T$. Traditionally, submodularity has been a key structural property for problems in combinatorial optimization, and for applications in econometrics, circuit and game theory, and operations research. More recently, submodularity's popularity in machine learning has been on the rise.

On the other hand, 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. Algorithms for submodular maximization are very different in nature from their submodular minimization cohorts, and their complexity too varies depending on the problem. In any case, 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.

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

In this paper, we address the issue of scalability and simultaneously draw connections across the apparent gap between minimization and maximization problems. We demonstrate that many algorithms for submodular maximization may be viewed as special cases of a generic minorize-maximize framework that relies on discrete semidifferentials. This framework encompasses state-of-the-art greedy and local search techniques, and provides a rich class of very practical algorithms. In addition, we show that any approximate submodular maximization algorithm can be seen as an instance of our framework.

We also present a complementary majorize-minimize framework for submodular minimization that 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.

As a whole, the semidifferential framework offers a new unifying perspective and basis for treating submodular minimization and maximization problems in both the constrained and unconstrained case. While it has long been known (Fujishige, 2005) that submodular functions have tight subdifferentials, our results rely on a recently discovered property (Iyer & Bilmes, 2012a; Jegelka & Bilmes, 2011b) showing that submodular functions also have superdifferentials. Furthermore, our approach is entirely combinatorial, thus complementing (and sometimes obviating) related relaxation methods.

2. Motivation and Background

Submodularity's escalating popularity in machine learning is due to its natural applicability. Indeed, instances of Problems 1 and 2 are seen in many forms, to wit:

MAP inference/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 & Jolly, 2001). A richer higher-order model can be induced for which MAP inference corresponds to Problem 1 where V is a set of edges in a graph, and $\mathcal{C}$ is a set of cuts in this graph — this was shown to significantly improve many image segmentation results (Jegelka & Bilmes, 2011b). Moreover, Delong et al. (2012) efficiently solve MAP inference in a sparse higher-order graphical model by restating the problem as a submodular vertex cover, i.e., Problem 1 where $\mathcal{C}$ is the set of all vertex covers in a graph.

Clustering: Variants of submodular minimization have been successfully applied to clustering problems (Narasimhan et al., 2006; Nagano et al., 2010).

Limited Vocabulary Speech Corpora: The problem of finding a maximum size speech corpus with bounded vocabulary (Lin & Bilmes, 2011a) can be posed as submodular function minimization subject to a size constraint. Alternatively, cardinality can be treated as a penalty, reducing the problem to unconstrained submodular minimization (Jegelka et al., 2011).

Minimum Power Assignment: In wireless networks, one seeks a connectivity structure that maintains connectivity at a minimum energy 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 & Berman, 1994). Similar costs have been considered for spanning tree and matching problems.

Summarization/Sensor placement: Submodular maximization also arises in many subset extraction problems. Sensor placement (Krause et al., 2008), document summarization (Lin & Bilmes, 2011b) and speech data subset selection (Lin & Bilmes, 2009), for example, are instances of submodular maximization.

Determinantal Point Processes: The Determinantal Point Processes (DPPs) which have found numerous applications in machine learning (Kulesza & Taskar, 2012) are known to be log-submodular distributions. In particular, the MAP inference problem is a form of non-monotone submodular maximization.

Indeed, there is strong motivation for solving Problems 1 and 2 but, as mentioned above, these problems come not without computational difficulties. Much work has therefore been devoted to developing optimal or near optimal algorithms. Among the several algorithms (McCormick, 2005) for the unconstrained variant of Problem 1, where $\mathcal{C} = 2^V$, the best complexity to date is $O(n^5\gamma + n^6)$ (Orlin, 2009) (γ is the cost of evaluating f). This has motivated studies on faster, possibly special case or approximate, methods (Stobbe & Krause, 2010; Jegelka et al., 2011). Constrained

minimization problems, even for simple constraints such as a cardinality lower bound, are mostly NP-hard, and not approximable to within better than a polynomial factor. Approximation algorithms for these problems with various techniques have been studied in (Svitkina & Fleischer, 2008; Iwata & Nagano, 2009; Goel et al., 2009; Jegelka & Bilmes, 2011a). Unlike submodular minimization, all forms of submodular maximization are NP-hard. Most such problems, however, admit constant-factor approximations, which are attained via very simple combinatorial algorithms (Nemhauser et al., 1978; Buchbinder et al., 2012).

Majorization-minimization (MM)[1] algorithms are known to be useful in machine learning (Hunter & Lange, 2004). Notable examples include the EM algorithm (McLachlan & Krishnan, 1997) and the convex-concave procedure (Yuille & Rangarajan, 2002). Discrete instances have been used to minimize the difference between submodular functions (Narasimhan & Bilmes, 2005; Iyer & Bilmes, 2012b), but these algorithms generally lack theoretical guarantees. This paper shows, by contrast, that for submodular optimization, MM algorithms have strong theoretical properties and empirically work very well.

3. Submodular semi-differentials

We first briefly introduce submodular semidifferentials. Throughout this paper, we assume normalized submodular functions (i.e., $f(\emptyset) = 0$). The subdifferential $\partial_f(Y)$ of a submodular set function $f : 2^V \to \mathbb{R}$ for a set $Y \subseteq V$ is defined (Fujishige, 2005) analogously to the subdifferential of a continuous convex function:

$$\partial_f(Y) = \{y \in \mathbb{R}^n : f(X) - y(X) \geq f(Y) - y(Y) \text{ for all } X \subseteq V\} \quad (1)$$

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

$$h^\sigma_Y(\sigma(i)) = f(S^\sigma_i) - f(S^\sigma_{i-1}). \quad (2)$$

Surprisingly, we can also define superdifferentials $\partial^f(Y)$ of a submodular function (Jegelka & Bilmes, 2011b; Iyer & Bilmes, 2012a) at Y:

$$\partial^f(Y) = \{y \in \mathbb{R}^n : f(X) - y(X) \leq f(Y) - y(Y); \text{for all } X \subseteq V\} \quad (3)$$

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 $\hat{g}_Y$ ("grow"), $\check{g}_Y$ ("shrink") and $\bar{g}_Y$ as follows (Iyer & Bilmes, 2012a):

$$\underbrace{\begin{array}{l}\hat{g}_Y(j) = f(j \mid V \setminus \{j\}) \\ \check{g}_Y(j) = f(j \mid Y \setminus \{j\}) \\ \bar{g}_Y(j) = f(j \mid V \setminus \{j\})\end{array}}_{\text{for } j \in Y} \qquad \underbrace{\begin{array}{l}\hat{g}_Y(j) = f(j \mid Y) \\ \check{g}_Y(j) = f(j \mid \emptyset) \\ \bar{g}_Y(j) = f(j \mid \emptyset)\end{array}}_{\text{for } j \notin Y.}$$

4. The discrete MM framework

With the above semigradients, we can define a generic MM algorithm. In each iteration, the algorithm optimizes a modular approximation formed via the current solution Y. For minimization, we use an upper bound

$$m^{g_Y}(X) = f(Y) + g_Y(X) - g_Y(Y) \geq f(X), \quad (4)$$

and for maximization a lower bound

$$m_{h_Y}(X) = f(Y) + h_Y(X) - h_Y(Y) \leq f(X). \quad (5)$$

Both these bounds are tight at the current solution, satisfying $m_{g_Y}(Y) = m_{h_Y}(Y) = f(Y)$. In almost all cases, optimizing the modular approximation is much faster than optimizing the original cost function f.

Algorithm 1 Subgradient ascent [descent] algorithm for submodular maximization [minimization]

1: Start with an arbitrary X^0.
2: **repeat**
3: Pick a semigradient h_{X^t} [g_{X^t}] at X^t
4: $X^{t+1} := \text{argmax}_{X \in \mathcal{C}} m_{h_{X^t}}(X)$
[$X^{t+1} := \text{argmin}_{X \in \mathcal{C}} m^{g_{X^t}}(X)$]
5: $t \leftarrow t + 1$
6: **until** we have converged ($X^{i-1} = X^i$)

Algorithm 1 shows our discrete MM scheme for maximization (MMax) [and minimization (MMin)] , and for both constrained and unconstrained settings. Since we are minimizing a tight upper bound, or maximizing a tight lower bound, the algorithm must make progress.

Lemma 4.1. *Algorithm 1 monotonically improves the objective function value for Problems 1 and 2 at every iteration, as long as a linear function can be exactly optimized over $\mathcal{C}$.*

Contrary to standard continuous subgradient descent schemes, Algorithm 1 produces a feasible solution at

[1] MM also refers to minorization-maximization here.

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, 2011). Nevertheless, Algorithm 1 still relies on the choice of the semigradients defining the bounds. Therefore, we next analyze the effect of certain choices of semigradients.

5. Submodular function minimization

For minimization problems, 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 to submodular minimization.

5.1. Unconstrained Submodular Minimization

We begin with unconstrained minimization, where $\mathcal{C} = 2^V$ in Problem 1. 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 $\mathcal{L} = [A, B]$ 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 5.1. *(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 5.1 has been used to prune down the search space of the minimum norm point algorithm (Bach, 2011; Fujishige & Isotani, 2011). Indeed, A and B may be obtained by using MMin-III:

Lemma 5.2. *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 $(X^0 \cap B) \cup A$.*

Lemma 5.2 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 5.3. *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 5.3 generalizes part of Lemma 3 in (Jegelka et al., 2011).

Theorem 5.3 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. 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 5.3 states that both MMin-I and II converge to a local minimum. 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., 2007).

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: $A_+ = B_+ = X^* = [1, 6, 7, 8, 10]$.

The MMin algorithms can be extended to arbitrary initializations and other supergradients (Iyer et al., 2013).

5.2. Constrained submodular minimization

MMin straightforwardly generalizes to constraints more complex than $\mathcal{C} = 2^V$, and Theorem 5.3 still holds for more general lattices or ring family constraints.

Beyond lattices, MMin applies to any set of constraints $\mathcal{C}$ as long as we have an efficient algorithm at hand that minimizes a nonnegative 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.

As opposed to unconstrained submodular minimization, almost all cases of constrained submodular minimization are very hard (Svitkina & Fleischer, 2008; Jegelka & Bilmes, 2011a; Goel et al., 2009), and admit at most approximate solutions in polynomial time. The next theorem states an upper bound on the approximation factor achieved by MMin-I for nonnegative, nondecreasing cost functions. An important ingredient in the bound is the *curvature* (Conforti & Cornuejols, 1984) of a monotone submodular function f, defined as

$$\kappa_f = 1 - \min_{j \in V} f(j \mid V \backslash j) \,/\, f(j) \qquad (6)$$

Theorem 5.4. *Let $X^* \in \operatorname{argmin}_{X \in \mathcal{C}} f(X)$. The solution $\widehat{X}$ returned by MMin-I satisfies*

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

If the minimization in Step 4 is done with approximation factor β, then $f(\widehat{X}) \leq \beta/(1 - \kappa_f) f(X^*)$.

A similar, slightly looser bound was shown for cuts in (Jegelka & Bilmes, 2011b), by using a weaker notion of curvature. Note that the bound in Theorem 5.4 is at most $\frac{n}{1+(n-1)(1-\kappa_f)}$, where $n = |V|$ is the dimension of the problem. We prove Theorem 5.4 in (Iyer et al., 2013).

In the worst case, when $\kappa_f = 1$, our approximation bounds are identical to prior work (Goel et al., 2009; Svitkina & Fleischer, 2008). Matroid rank functions have $\kappa_f = 1$, implying that they are difficult instances for MMin. But several practically relevant submodular functions do satisfy $\kappa_f > 0$. In such case, Theorem 5.4 replaces known polynomial bounds by an improved factor depending on κ_f. An example for such functions are concave over modular functions used in (Stobbe & Krause, 2010; Lin & Bilmes, 2011b; Jegelka & Bilmes, 2011b). These comprise, for instance, functions of the form $f(X) = (w(X))^a$, for some $a \in [0, 1]$ and a nonnegative weight vector w, whose curvature is $\kappa_f \approx 1 - a(\frac{\min_j w(j)}{w(V)})^{1-a} < 1$. Another example is $f(X) = \log(1 + w(X))$ with $\kappa_f \approx 1 - \frac{\min_j w(j)}{w(V)}$. Several applications use a sum of such functions, each with bounded support (Jegelka & Bilmes, 2011b; Iyer & Bilmes, 2012b). This further reduces the curvature.

The bounds of Theorem 5.4 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 5.4, 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.

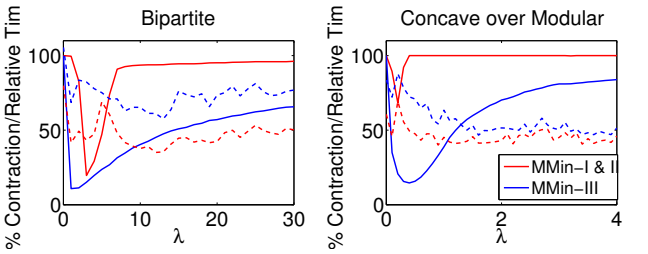

Figure 1. Lattice reduction (solid line), and runtime (%) of MMin+min-norm relative to unadorned min-norm (dotted).

Lemma 5.5. *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 $X \in \mathcal{C}$.*

5.3. Experiments

We will next see that, apart from its theoretical properties, MMin is in practice competitive to more complex algorithms. We implement and compare algorithms using Matlab and the SFO toolbox (Krause, 2010).

Unconstrained minimization We first study the results in Section 5.1 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 & 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 & 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 1. 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 & 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.

Constrained minimization. For constrained minimization, we compare MMin-I to two methods: a simple algorithm (MU) that minimizes the upper bound

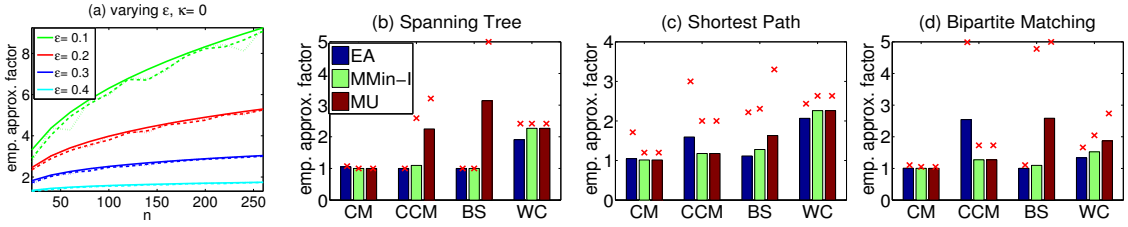

Figure 2. Constrained minimization for worst-case (a) and average-case (b-d) instances. In (a), Dashed lines: MMin, dotted lines: EA, solid lines: theoretical bound. In (b - d), bars are average approximation factors and crosses worst observed results. CM - Concave over Mod., CCM - Clust. Concave Mod., BS - Best Set and WC - Worst Case

$g(X) = \sum_{i \in X} f(i)$ (Goel et al., 2009) (this is identical to the first iteration of MMin-I), and a more complex algorithm (EA) that computes an approximation to the submodular polyhedron (Goemans et al., 2009) and in many cases yields a theoretically optimal approximation. MU has the theoretical bounds of Theorem 5.4, while EA achieves a worst-case approximation factor of $O(\sqrt{n} \log n)$. We show two experiments: the theoretical worst-case and average-case instances. Figure 2 illustrates the results.

Worst case. We use a very hard cost function (Goemans et al., 2009)

$$f(X) = \min\{|X|, |X \cap \bar{R}| + \beta, \alpha\}, \tag{7}$$

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 2 shows results for cardinality lower bound constraints; the results for other, more complex constraints are similar. As ϵ shrinks, the problem becomes harder. In this case, EA and MMin-I achieve about the same empirical approximation factors, which matches the theoretical guarantee of $n^{1/2-\epsilon}$.

Average case. We next compare the algorithms on more realistic functions that occur in applications. Figure 2 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 (7). Functions of type (1) and (2) have been used in speech and computer vision (Lin & Bilmes, 2011b; Jegelka & Bilmes, 2011b; Iyer & 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 more details, please refer to (Iyer et al., 2013).

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 needs about 58 seconds. On larger instances ($n = 500$), the running times differ on the order of seconds versus hours.

6. Submodular maximization

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. 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, 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 6.1. *MMax with $X^0 = \text{argmax}_j f(j)$ runs in time $O(T \log_{1+\eta} n)$, where T is the time for maximizing a modular function subject to $X \in \mathcal{C}$.*

In practice, we observe that MMax terminates within 3-10 iterations. We next consider specific subgradients and their theoretical implications. For unconstrained problems, we assume the submodular function to be non-monotone (the results trivially hold for monotone functions too); for constrained problems, we assume the function f to be monotone nondecreasing. 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. The proofs for the statements in Section 6.1 may be found in (Iyer et al., 2013).

6.1. Unconstrained Maximization

Random Permutation (RA/RP). In iteration t, we randomly pick a permutation σ that defines a subgradient at X^{t-1}, i.e., X^{t-1} is assigned to the first $|X^{t-1}|$ positions. At $X^0 = \emptyset$, this can be any permutation. Stopping after the first iteration (RP) achieves an approximation factor of $1/4$ in expectation, and $1/2$ for symmetric functions. Making further iterations (RA) only improves the solution.

Randomized local search (RLS). Instead of using a completely random subgradient as in RA, we fix the positions of two elements: the permutation must satisfy that $\sigma^t(|X^t|+1) \in \operatorname{argmax}_j f(j|X^t)$ and $\sigma^t(|X^t|-1) \in \operatorname{argmin}_j f(j|X^t \backslash j)$. 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.

Deterministic local search (DLS). 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 $\emptyset = S_0^{\sigma^t} \subset S_1^{\sigma^t} \subset \ldots \subset S_n^{\sigma^t}$ it will generate. The initial permutation is $\sigma^0(j) = \operatorname{argmax}_{k \notin S_{j-1}^{\sigma^0}} f(k|S_{j-1}^{\sigma^0})$ for $j = 1, 2, \ldots$. In subsequent iterations t, the permutation σ^t is

$$\sigma^t(j) = \begin{cases} \sigma^{t-1}(j) & \text{if } t \text{ even}, j \in X^{t-1} \\ \operatorname{argmax}_k f(k|S_{j-1}^{\sigma^t}) & \text{if } t \text{ even}, j \notin X^{t-1} \\ \operatorname{argmin}_k f(k|S_{j+1}^{\sigma^t} \backslash k) & \text{if } t \text{ odd}, j \in X^{t-1} \\ \sigma^{t-1}(j) & \text{if } t \text{ odd}, j \notin X^{t-1}. \end{cases}$$

This schedule is equivalent to the deterministic local search (DLS) algorithm by Feige et al. (2007), and therefore achieves an approximation factor of $1/3 - \eta$.

Bi-directional greedy (BG). 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. It is therefore equivalent to MMax with the corresponding subgradients and achieves an approximation factor of $1/3$. This factor improves that of the local search techniques by removing η. Moreover, unlike for local search, the $1/3$ approximation holds already after the first iteration.

Randomized bi-directional greedy (RG). Like its deterministic variant, the randomized bi-directional greedy algorithm 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.

6.2. Constrained Maximization

In this final section, we analyze subgradients for maximization subject to the constraint $X \in \mathcal{C}$. Here we assume that f is monotone. An important subgradient results from the greedy permutation σ^g, defined as

$$\sigma^g(i) \in \operatorname*{argmax}_{j \notin S_{i-1}^{\sigma^g} \text{ and } S_{i-1}^{\sigma^g} \cup \{j\} \in \mathcal{C}} f(j|S_{i-1}^{\sigma^g}). \tag{8}$$

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:

Lemma 6.2. *Using h^{σ^g} in iteration 1 of MMax yields the following approximation bounds for X^1:*

- $\frac{1}{\kappa_f}(1 - e^{-\kappa_f})$, *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 $\mathcal{C}$.*

A similar result holds for Knapsack constraints. The proof of Lemma 6.2 (Iyer et al., 2013) relies on the observation that the maximizer of the function m_h for the subgradient $h = h^{\sigma^g}$ is never worse than the result of a greedy algorithm. The bounds follow from (Conforti & Cornuejols, 1984).

6.3. Generality

The correspondences between MMax and maximization algorithms hold even more generally:

Theorem 6.3. *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.*

Under mild assumptions, Theorem 6.3 holds even for constrained maximization. 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 6.4. *The problem of finding the optimal subgradient $\sigma^{OPT} = \operatorname{argmax}_{\sigma, X \subseteq V} h^{\sigma}_{X^t}(X)$ in Step 4 of Algorithm 1 is NP-hard even when $\mathcal{C} = 2^V$. Given such*

an oracle, however, MMax using subgradient σ^{OPT} returns a global optimizer.

6.4. Experiments

We now empirically compare variants of MMax with different subgradients. As a test function, we use the objective of Lin & 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. 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 & Bilmes, 2009) on the TIMIT corpus (Garofolo et al., 1993), using the string kernel metric (Rousu & 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 algorithms DLS, BG, RG, RLS, RA and RP, and a baseline RS that picks a set uniformly at random. RS achieves a 1/4 approximation in expectation (Feige et al., 2007). For random algorithms, we select the best solution out of 5 repetitions. Figure 3 shows that DLS, BG, RG and RLS dominate. Even though RG 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).

7. Discussion and Conclusions

In this paper, we introduced a general MM framework for submodular optimization algorithms. This framework is akin to the class of algorithms for minimizing the difference between submodular functions (Narasimhan & Bilmes, 2005; Iyer & Bilmes, 2012b). In addition, it 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 combinatorial algorithms for submodular optimization.

An alternative framework relies on relaxing the discrete optimization problem by using a continuous extension (the Lovász extension for minimization and multilinear extension for maximization). Relaxations have been applied to some constrained (Iwata & Nagano, 2009) and unconstrained (Bach, 2011) minimization problems as well as maximization problems (Buchbinder et al., 2012). Such relaxations, however, rely on a final rounding step that can be challenging — the combinatorial framework obviates this step. Moreover, our results show that in many cases, it yields good results very efficiently.

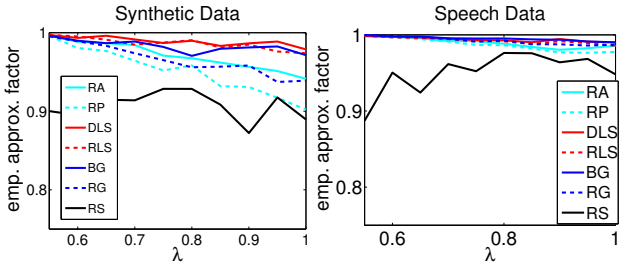

Figure 3. Empirical approximation factors for variants of MMax. See Section 6.1 for legend details.

Acknowledgments: We thank Karthik Mohan, John Halloran and Kai Wei for discussions. This material is based upon work supported by the National Science Foundation under Grant No. IIS-1162606, and by a Google, a Microsoft, and an Intel research award.

References

Averbakh, I. and Berman, O. Categorized bottleneck-minisum path problems on networks. *Operations Research Letters*, 16:291–297, 1994.

Bach, F. Learning with Submodular functions: A convex Optimization Perspective. *Arxiv*, 2011.

Boykov, Y. and Jolly, M.P. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, 2001.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. A tight (1/2) linear-time approximation to unconstrained submodular maximization. *In FOCS*, 2012.

Conforti, M. and Cornuejols, G. 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.

Delong, A., Veksler, O., Osokin, A., and Boykov, Y. Minimizing sparse high-order energies by submodular vertex-cover. In *In NIPS*, 2012.

Feige, U., Mirrokni, V., and Vondrák, J. Maximizing non-monotone submodular functions. *SIAM J. COMPUT.*, 40(4):1133–1155, 2007.

Fujishige, S. *Submodular functions and optimization*, volume 58. Elsevier Science, 2005.

Fujishige, S. and Isotani, S. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7:3–17, 2011.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallet, D., and Dahlgren, N. Timit, acoustic-phonetic continuous speech corpus. In *DARPA*, 1993.

Goel, G., Karande, C., Tripathi, P., and Wang, L. Approximability of combinatorial problems with multi-agent submodular cost functions. In *FOCS*, 2009.

Goemans, M.X., Harvey, N.J.A., Iwata, S., and Mirrokni, V. Approximating submodular functions everywhere. In *SODA*, pp. 535–544, 2009.

Goldengorin, B., Tijssen, G.A., and Tso, M. *The maximization of submodular functions: Old and new proofs for the correctness of the dichotomy algorithm.* University of Groningen, 1999.

Hunter, D.R. and Lange, K. A tutorial on MM algorithms. *The American Statistician*, 2004.

Iwata, S. and Nagano, K. Submodular function minimization under covering constraints. In *In FOCS*, pp. 671–680. IEEE, 2009.

Iyer, R. and Bilmes, J. The submodular Bregman and Lovász-Bregman divergences with applications. In *NIPS*, 2012a.

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

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

Iyer, R., Jegelka, S., and Bilmes, J. Fast Semidifferential-based Submodular Function Optimization : Extended Version, 2013.

Jegelka, S. and Bilmes, J. A. Approximation bounds for inference using cooperative cuts. In *ICML*, 2011a.

Jegelka, S. and Bilmes, J. A. Submodularity beyond submodular energies: coupling edges in graph cuts. In *CVPR*, 2011b.

Jegelka, S., Lin, H., and Bilmes, J. On fast approximate submodular minimization. *In NIPS*, 2011.

Krause, A. SFO: A toolbox for submodular function optimization. *JMLR*, 11:1141–1144, 2010.

Krause, A., Singh, A., and Guestrin, C. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *JMLR*, 9: 235–284, 2008.

Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *arXiv preprint arXiv:1207.6083*, 2012.

Lin, H. and Bilmes, J. How to select a good training-data subset for transcription: Submodular active selection for sequences. In *Interspeech*, 2009.

Lin, H. and Bilmes, J. Optimal selection of limited vocabulary speech corpora. In *Interspeech*, 2011a.

Lin, H. and Bilmes, J. A class of submodular functions for document summarization. *In ACL*, 2011b.

McCormick, S Thomas. Submodular function minimization. *Discrete Optimization*, 12:321–391, 2005.

McLachlan, G.J. and Krishnan, T. *The EM algorithm and extensions*. New York, 1997.

Nagano, K., Kawahara, Y., and Iwata, S. Minimum average cost clustering. In *NIPS*, 2010.

Narasimhan, M. and Bilmes, J. A submodular-supermodular procedure with applications to discriminative structure learning. In *UAI*, 2005.

Narasimhan, M., Jojic, N., and Bilmes, J. Q-clustering. *NIPS*, 18:979, 2006.

Nemhauser, G.L., Wolsey, L.A., and Fisher, M.L. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14 (1):265–294, 1978.

Orlin, J.B. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, 2009.

Rousu, J. and Shawe-Taylor, J. Efficient computation of gapped substring kernels on large alphabets. *Journal of Machine Learning Research*, 6(2):1323, 2006.

Stobbe, P. and Krause, A. Efficient minimization of decomposable submodular functions. In *NIPS*, 2010.

Svitkina, Z. and Fleischer, L. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS*, pp. 697–706, 2008.

Wan, P.-J., Calinescu, G., Li, X.-Y., and Frieder, O. Minimum-energy broadcasting in static ad hoc wireless networks. *Wireless Networks*, 8:607–617, 2002.

Yuille, A.L. and Rangarajan, A. The concave-convex procedure (CCCP). *In NIPS*, 2002.