Maximizing Submodular Functions: From Theory to Practice

Bolin Ding
bding3@uiuc.edu

Zhenhui Li
zli28@uiuc.edu

CS598 2010 Spring

Abstract

We review a collection of techniques for maximizing submodular functions under different constraints. In general, we want to maximize a function $f : 2^X \to \mathbb{R}$ ($\max_{A \subseteq X} f(A)$), under certain constraints, e.g., cardinality/budget constraint ($|A| \leq k$), matroid constraint (A is independent), and graph structure constraint (A is a path or a circle in a graph). These problems are usually NP-hard, so we mostly focus on approximation algorithms.

The submodularity implies that adding an element to a small set A helps more than adding it to a large set $B \supseteq A$. This so-called *diminishing returns property* has been found in a number of practical problems, such as *observation selection* problems in machine learning and *influential maximization* problems in social networks. We will also review the applications of optimizing submodular functions in machine learning, AI, social networks, and so on.

Contents

1 Introduction

In this survey, we consider the problem of *maximizing submodular functions* (possibly under certain constraints) and its applications in AI, social networks, databases, and so on.

Submodular functions. *Submodularity* is a discrete analog of convexity. Formally, given a finite ground set S, a *set function* $f : 2^X \to \mathbb{R}$ is *submodular* if $f(A)+f(B) \geq f(A \cap B)+f(A \cup B)$, $\forall A, B \subseteq X$; or equivalently, $f(A + e) - f(A) \geq f(B + e) - f(B)$, $\forall A \subseteq B \subseteq X$ and $e \in X \setminus B$. Another equivalent definition is $f(A+e_1)+f(A+e_2) \geq f(A)+f(A+e_1+e_2)$, $\forall A \subseteq X$ and distinct $e_1, e_2 \in X \setminus A$. Throughout this paper, for a set A, we abbreviate $A \cup \{x\}$ by $A + i$, and let $f_A(i) = f(A \cup \{i\}) - f(A)$, and $f_A(B) = f(A \cup B) - f(A)$.

Outline. This survey consists of two parts. In the first part, we will review a collection of techniques for maximizing submodular functions. Unlike submodular function minimization, maximizing submodular functions is usually NP-hard. Thus, we will mostly focus on approximation algorithms. We will introduce the *cardinality constraint* [25], *path constraint* [4], and *matroid constraint* [2] in maximizing monotone submodular functions in Section 2. Then, we will consider the unconstrained maximization of non-monotone submodular functions [8] in Section 3. We will also have a brief rview of other related techniques in Section 2-3.

In the second part, we will discuss the application of submodular function in different "practical" problems. As many techniques for submodular function maximization can be efficiently implemented, they have been broadly applied in different areas, such as machine learning, AI, social networks, databases and so on. We will review some of them in Section 4.

Objectives. Maximizing submodular functions is of importance in terms of both theory and practice. The goal of this survey is to introduce a collection of recent results on this theoretical topic and review how they are applied in "practical" problems. Simple and efficient algorithms are preferred in solving practical problems. So this survey gives introduction to both sides (theoretical results and practical problems), and want to motivate the design of efficient approximation algorithms for specific problems encountered in other areas.

2 Maximizing Monotone Submodular Functions

A set function $f : 2^X \to \mathbb{R}$ is *non-negative* if $f(A) \geq 0$ $\forall A \subseteq X$. f is *monotone* if $f(A) \leq f(B)$ $\forall A \subseteq B$. In this section, we consider non-negative monotone submodular functions and assume $f(\emptyset) = 0$, as $f(\emptyset) > 0$ usually results in better approximation ratio.

2.1 Cardinality Constraint

A simple and extensively-studied submodular function is the *set coverage* function, i.e., the size of set union. The Max-k-Cover problem is to choose k sets (from a collection of sets) whose union is as large as possible. It is well-known that a greedy algorithm provides a $(1 - 1/e)$-approximation for this problem and this is optimal unless P = NP [7].

This problem is naturally generalized to the problem of *maximizing submodular function under cardinality constraints*, i.e., for a non-negative monotone submodular function $f : 2^X \to \mathbb{R}$, to find $\max_{S \in X}\{f(S) \mid |S| \leq k\}$. The greedy algorithm for the Max-k-Cover problem can be also generalized with the same approximation ratio preserved [25, 24]. [29] considers the minimum

weighted submodular set cover problem. It is also worthy to mention that continuous greedy algorithms are used to deal with the extension of submodular set functions to a continuous domain [29, 30]. Later in Section 2.2.1, we will discuss how a continuous greedy process is applied in maximizing continuous submodular function within a matroid polytope [2].

We now present the discrete greedy algorithm for maximizing submodular function under cardinality constraints, and a simple analysis of the approximation ratio.

2.1.1 Greedy Algorithm

$\underline{\textsc{Greedy}(f, X, k)}$:
Let $S \leftarrow \emptyset$.
Repeat k times: Pick $x \in X \setminus S$ s.t. $f(S+x) - f(S)$ is maximized.
Output S.

Theorem 2.1 GREEDY *gives a* $(1-1/e)$*-approximation for the problem of maximizing submodular function under cardinality constraint.*

Proof. Let x_i be the element picked in the i^{th} iteration of GREEDY. Let $S_i = \{x_1, x_2, \ldots, x_i\}$. Suppose $S^* = \{x_1^*, x_2^*, \ldots, x_k^*\}$ is the optimal solution, and define $r_i = f(S^*) - f(S_i) = \text{OPT} - f(S_i)$. If we can prove $f(S_{i+1}) - f(S_i) \geq \frac{r_i}{k}$, then $\text{OPT} - f(S_{i+1}) \leq (1 - \frac{1}{k}) \cdot (\text{OPT} - f(S_i))$, from which we can conclude that $\text{OPT} - f(S_k) \leq (1 - \frac{1}{k})^k \cdot \text{OPT}$, i.e., $f(S_k) \geq (1 - 1/e)\text{OPT}$.

To prove $f(S_{i+1}) - f(S_i) \geq \frac{r_i}{k}$, consider the monotonicity and submodularity of f, we have

$$r_i = f(S^*) - f(S_i) \leq f(S^* \cup S_i) - f(S_i) \leq \sum_{j=1}^{k} \left(f(S_i + x_j^*) - f(S_i) \right).$$

So there is at least one x_j^* s.t. $f(S_i + x_j^*) - f(S_i) \geq \frac{r_i}{k}$. □

2.2 Matroid Constraint

A natural generalization of the cardinality constraint is also of much importance in applications: Suppose X is partitioned into $X_1, X_2, \ldots, X_l$ with associated integers $k_1, k_2, \ldots, k_l$. A set $A \subseteq X$ is *independent* iff $|A \cap X_i| \leq k_i$ for any i, and all such independent sets form a *partition matroid* $\mathcal{M} = (X, \mathcal{I})$, where $\mathcal{I} \subseteq 2^X$ is the collection of independent sets. Our goal is to find $\max\{f(S) \mid S \in \mathcal{I}\}$.

We are now mainly interested in the general *matroid* constraint, which contains the partition matroid constraint as a special case. It has been shown that a greedy algorithm produces an $\frac{1}{2}$-approximation under a matroid constraint, and produces an $\frac{1}{p+1}$-approximation for the intersection of p matroids (a *feasible* set is independent in p matroids) [9]. A natural question is whether finding an $(1 - 1/e)$-approximation in the *value oracle* model under a matroid constraint (one matroid) is possible, which had been an open problem for a long time until [1] applied the pipage rounding to obtain an $(1-1/e)$-approximation algorithm for a special class of submodular function. [28, 2] later generalizes the algorithm for all nonnegative monotone submodular functions using the combination of a *continuous greedy process* and *pipage rounding technique*.

In the following part, we will present the main algorithm in [2]. It consists of two steps: In the first step (Theorem 2.3), we use the continuous greedy process to approximation $\max\{F(y) \mid y \in$

$P(\mathcal{M})\}$ with a factor of $1-1/e$, where F is a smoothed version of f in the continuous domain, and $P(\mathcal{M})$ the matroid polytope of $\mathcal{M}$. In the second step (Theorem 2.6), we use the pipage technique to round $y \in P(\mathcal{M})$ to a set $S \in \mathcal{I}$ with $f(S) \geq F(y) \geq (1-1/e)\mathrm{OPT}$.

Multilinear extension $F(y)$. For a nonnegative monotone submodular set function $f: 2^X \to \mathbb{R}_+$, a canonical extension $F: [0,1]^X \to \mathbb{R}_+$ can be obtained as follows: For $y \in [0,1]^X$, let $\hat{y}$ be a random vector $\{0,1\}^X$ where each coordinate is independently rounded to 1 with probability y_i or 0 otherwise. Let $R \subseteq X$ be a set whose indicator vector is $\hat{y}$. Then, we define

$$F(y) = \mathbf{E}\left[f(\hat{y})\right] = \sum_{R \subseteq X} f(R) \prod_{i \in R} y_i \prod_{j \notin R} (1-y_i).$$

A function $F: [0,1]^X \to \mathbb{R}_+$ is *smooth monotone submodular* if (i) F has second partial derivatives everywhere; (ii) for each $j \in X$, $\frac{\partial F}{\partial y_j} \geq 0$ everywhere (monotone); and (iii) for any $i, j \in X$, $\frac{\partial^2 F}{\partial y_i \partial y_j} \leq 0$ everywhere (submodular).

Let F be the canonical extension of a monotone submodular function f, we can verify that

$$\frac{\partial F}{\partial y_j} = \mathbf{E}\left[f(\hat{y}) \mid \hat{y}_j = 1\right] - \mathbf{E}\left[f(\hat{y}) \mid \hat{y}_j = 0\right] \geq 0$$

and

$$\begin{aligned}\frac{\partial^2 F}{\partial y_i \partial y_j} &= \left(\mathbf{E}\left[f(\hat{y}) \mid \hat{y}_i = 1, \hat{y}_j = 1\right] - \mathbf{E}\left[f(\hat{y}) \mid \hat{y}_i = 1, \hat{y}_j = 0\right]\right) \\ &\quad - \left(\mathbf{E}\left[f(\hat{y}) \mid \hat{y}_i = 0, \hat{y}_j = 1\right] - \mathbf{E}\left[f(\hat{y}) \mid \hat{y}_i = 0, \hat{y}_j = 0\right]\right) \leq 0.\end{aligned}$$

So we can conclude that

Proposition 2.2 *Let F be the canonical extension of a monotone submodular function f. Then F is a smooth monotone submodular function.*

2.2.1 Continuous Greedy Process

To approximate $\max\{F(y) \mid y \in P(\mathcal{M})\}$ with a factor of $1-1/e$, the Greedy algorithm (and its analysis in Theorem) can be generalized in a continuous way.

ContinuousGreedy$(f, \mathcal{M})$:
Start with $t = 0$ and $y(0) = \mathbf{0}$ (analogue to $S = \emptyset$ in Greedy). Select a proper step length δ.
In each step, while $t < 1$:
  Let $R(t)$ contain each $j \in X$ with probability $y_j(t)$ and $w_j(t) = \mathbf{E}\left[f_{R(t)}(j)\right]$.
  Let $I(t)$ be the maximum-weight independent set in $\mathcal{M}$ according to the weights $w_j(t)$ (easily obtained using a greedy algorithm).
  Let $y(t+\delta) = y(t) + \delta \cdot \mathbf{1}_{I(t)}$ (analogue to the greedy selection in Greedy).
Output $y(1)$.

Theorem 2.3 *In* ContinuousGreedy *algorithm, we have that $y(1) \in P(\mathcal{M})$ and, with high probability, $F(y(1)) \geq (1-1/e)\mathrm{OPT}$.*

It is easy to see $y(1) \in P(\mathcal{M})$ since $y(t) = \sum_t \delta \cdot \mathbf{1}_{I(t)}$. For the approximation ratio, as an analogy to GREEDY, if we can prove that $F(y(t+\delta)) - F(y(t)) \geq \delta(\mathrm{OPT} - F(y(t)))$, then

$$\mathrm{OPT} - F(y(t+\delta)) \leq (1-\delta)(\mathrm{OPT} - F(y(t))) \quad \Rightarrow \quad \mathrm{OPT} - F(y(1)) \leq (1-\delta)^{\delta}\mathrm{OPT},$$

i.e., $F(y(1)) \geq (1-1/e)\mathrm{OPT}$. Technique details are formalized in the following two lemmas.

Lemma 2.4 *In* CONTINUOUSGREEDY *algorithm,* $F(y(t+\delta)) - F(y(t)) \geq \delta(1-d\delta)\sum_{j\in I(t)} \mathbf{E}\left[f_{R(t)}(j)\right]$.

Proof. Let $R(t)$ contain $j \in X$ with probability $y_j(t)$ and $D(t)$ contains $j \in I(t)$ with probability δ. Let $\Delta_j(t) = \delta$ if $j \in I(t)$ and 0 otherwise. Since $\mathbf{Pr}\left[j \in R(t+\delta)\right] = y_j(t) + \Delta_j(t)$ and $\mathbf{Pr}\left[j \in R(t) \cup D(t)\right] = 1 - (1-y_j(t))(1-\Delta_j(t)) \leq \mathbf{Pr}\left[j \in R(t+\delta)\right]$, for the monotonicity of f, we have $\mathbf{E}\left[f(R(t+\delta))\right] \geq \mathbf{E}\left[f(R(t) \cup D(t))\right]$. Therefore, we have

$F(y(t+\delta)) - F(y(t)) \geq \mathbf{E}\left[f(R(t) \cup D(t)) - f(R(t))\right] \geq \sum_j \mathbf{Pr}\left[D(t) = \{j\}\right] \mathbf{E}\left[f_{R(t)}(j)\right]$
$= \sum_{j\in I(t)} \delta(1-\delta)^{|I(t)|-1}\mathbf{E}\left[f_{R(t)}(j)\right] \geq \delta(1-d\delta)\mathbf{E}\left[f_{R(t)}(j)\right]$. □

If we can prove $\sum_{j\in I(t)} \mathbf{E}\left[f_{R(t)}(j)\right] \geq \mathrm{OPT} - F(y(t))$, then the proof of Theorem 2.3 is completed. In fact, suppose the optimal solution is S^*, from f is monotone and submodular, we have $\mathrm{OPT} = f(S^*) \leq f(R(t) \cup S^*) \leq f(R(t)) + \sum_{j\in S^*} f_{R(t)}(j)$. Taking expectation and from how $I(t)$ is selected in CONTINUOUSGREEDY, we have that $\mathrm{OPT} \leq F(y(t)) + \sum_{j\in I(t)} \mathbf{E}\left[f_{R(t)}(j)\right]$.

Unfortunately, in a value oracle model of f, we can only get an estimation $w_j(t)$ of $\mathbf{E}\left[f_{R(t)}(j)\right]$ instead of the precise value. So more work needs to be done. Let $w_j(t)$ be an estimation of $\mathbf{E}\left[f_{R(t)}(j)\right]$ by taking the average of $\frac{10}{\delta^2}(1+\ln n)$ independent samples of $R(t)$. We have:

Lemma 2.5 *In* CONTINUOUSGREEDY *algorithm, with high probability,*

$$\sum_{j\in I(t)} \mathbf{E}\left[f_{R(t)}(j)\right] \geq (1-2d\delta)\mathrm{OPT} - F(y(t)).$$

The above lemma can be proved using Chernoff bounds. And then, from Lemma 2.4 and 2.5, we can complete the proof of Theorem 2.3.

2.2.2 Pipage Rounding

Now the remaining task is to round the fractional solution $y \in P(\mathcal{M})$ returned by CONTINUOUSGREEDY to an integral solution $S \in \mathcal{I}$ with $\mathbf{E}\left[f(S)\right] \geq F(y)$.

Consider the $y \in P(\mathcal{M})$ returned by CONTINUOUSGREEDY, for the monotonicity of F, we can assume $y(X) = r_{\mathcal{M}}(X)$, i.e., $y \in B(\mathcal{M})$. So if y is not integral, there must exist at least two fractional variables y_i and y_j. Let $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, where only the i^{th} entry is 1, and $y_{ij}(t) = y + t(\mathbf{e}_i - \mathbf{e}_j)$. Consider the function $F_{ij}^y(t) = F(y_{ij}(t))$. From Proposition 2.2, we can verify that $F_{ij}^y(t)$ is convex, and thus, either $F_{ij}^y(t)$ or $F_{ji}^y(t)$ is nondecreasing. The intuition of pipage rounding is that we select the direction of $F_{ij}^y(t)$ or $F_{ji}^y(t)$ (whichever is nondecreasing) and modify y as $y' \leftarrow y_{ij}(t')$ or $y_{ji}(t')$ for some t' to eliminate at least one fractional variable each time. It should be noted that y' may violate the matroid constraint, so t' is selected carefully, and a fractional variable may be eliminated in more than one iteration.

To ensure that after a polynomial number of iteration, the pipage rounding can terminate, we will work with *tight sets* in the rounding process. A set $A \subseteq X$ is *tight* if $y(A) = r_{\mathcal{M}}(A)$. In

a tight set, there are always either at least two fractional variables or none. Initially, X is tight ($y(X) = r_{\mathcal{M}}(X)$). We will argue that after each iteration of rounding, either the size of the working tight set decreases, or a fractional variable is eliminated.

HitConstraint$(y, \mathcal{M}, i, j)$:
Define $\mathcal{A} = \{A \subseteq X \mid i \in A, j \notin A\}$.
Find $\delta = \min_{A \in \mathcal{A}}(r_{\mathcal{M}}(A) - y(A))$ and A^* to achieve this.
Case 1: (if $y_j < \delta$) $y_i \leftarrow y_i + y_j$, $y_j \leftarrow 0$, $A_{\text{hit}} \leftarrow \{j\}$.
Case 2: (if $y_j \leq \delta$) $y_i \leftarrow y_i + \delta$, $y_j \leftarrow y_j - \delta$, $A_{\text{hit}} \leftarrow A^*$.
Output (y, A_{hit}).

After the calling of HitConstraint, we will either eliminate a fractional variable (Case 1) or get a tight set A_{hit} which excludes j (Case 2). It is also possible that $\delta = 0$ (y is unchanged), but even in this case, A_{hit} still excludes j. In both case, y returned by HitConstraint is feasible (in $P(\mathcal{M})$) for the selection of δ. The main procedure of pipage rounding is as follows.

PipageRound$(y, \mathcal{M})$:
While y is not integral do:
  $T \leftarrow X$ (an initial tight set).
  While T contains fractional variables do:
    Pick $i, j \in T$ with fractional y_i, y_j.
    $(y^+, A^+) \leftarrow$ HitConstraint$(y, \mathcal{M}, i, j)$.
    $(y^-, A^-) \leftarrow$ HitConstraint$(y, \mathcal{M}, j, i)$.
    If $y^+ = y^- = y$ then $T \leftarrow T \cap A^+$;
    Else: $p \leftarrow ||y^+ - y||/||y^+ - y^-||$
      With probability p, $y \leftarrow y^-$, $T \leftarrow T \cap A^-$; Otherwise, $y \leftarrow y^+$, $T \leftarrow T \cap A^+$.
Output y.

First, we argue that T is always a tight set in PipageRound, except when $T = A^+$ (or A^-) $= \{j\}$ with $y_j = 0$ (A^+ or A^- is returned by Case 1 of HitConstraint), the inner loop of PipageRound terminates immediately. This is because A^+ (or A^-) in Case 2 of HitConstraint is tight and T is tight (by induction), and thus $T \cap A^+$ or $T \cap A^-$ is tight (not hard to prove the intersection/union of two tight sets is also tight). Note that X is always tight, from how y is updated in HitConstraint.

Second, in each iteration of the inner loop, we have either, $|T|$ decreases by at least one (A^+ or A^- excludes j or i, respectively), or a fractional variable is eliminated (Case 1 of HitConstraint).

Third, after the inner loop terminates, at least one fractional variable is eliminated, because in each iteration of the inner loop, at least one of fractional variables i, j remains in T.

From the above analysis, PipageRound$(y^*, \mathcal{M})$ uses at most $O(|X|^2)$ iterations to get an integral solution S. As now the direction $F^y_{ij}(t)$ or $F^y_{ji}(t)$ is selected randomly, we may not have $f(S) \geq F(y^*)$ for sure, but we can prove that $\mathbf{E}[f(S)] \geq F(y^*)$.

Consider two consecutive iteration of the inner loop, suppose y is updated as y'. Let $y^+ = y + t^+(\mathbf{e}_i - \mathbf{e}_j)$, $y^- = y + t^-(\mathbf{e}_j - \mathbf{e}_i)$, using the convexity of F^y_{ij} and Jenson's inequality, we have

$$\mathbf{E}\left[F(y')\right] = pF^y_{ji}(t^-) + (1-p)F^y_{ij}(t^+) = pF^y_{ij}(-t^-) + (1-p)F^y_{ij}(t^+) \geq F^y_{ij}(-pt^- + (1-p)t^+) = F^y_{ij}(0) = F(y).$$

By induction on the number of steps, we have $\mathbf{E}[f(S)] \geq F(y^*)$.

Theorem 2.6 *Given $y^* \in B(\mathcal{M})$,* PipageRound$(y^*, \mathcal{M})$ *outputs in polynomial time a indepdent set $S \in \mathcal{I}$ with $\mathbf{E}[f(S)] \geq F(y^*)$.*

2.3 Path Constraint

Beside matroid constraint, there is still some other useful constraints, like *path constraint* introduced in [4], which can be found in a broad class of applications.

Given an arc-weighted directed graph $G = (V, A, l)$, a nonnegative monotone submodular function $f : 2^V \to \mathbb{R}$ is defined on the set of vertices, with $f(\emptyset) = 0$. We assume G satisfies the *asymmetric triangle inequality*: $\forall u, v, w \in V,\ l(u, v) + l(v, w) \geq l(u, w)$. And thus, we can restrict our attention from walks to paths and cycles. Let $\mathcal{P}(s, t, B)$ be all the s-t walks of length at most B ($\forall P \in \mathcal{P}(s, t, B),\ l(P) \leq B$). Our goal is to find $\max\{f(P) \mid P \in \mathcal{P}(s, t, B)\}$.

In the following part, we will present the quasi-polynomial time algorithm [4] that yields an $O(\log \mathrm{OPT})$-approximation for this problem.

Let $|V| = n$. We can assume $f(V)$ is poly-bounded in n. Otherwise, we can convert f to be poly-bounded with the loss of a $(1 + O(\epsilon))$ factor in the approximation ratio.

The algorithm in [4] is surprisingly simple and clean. The main idea is to guess the middle vertex v of a good solution P and the length B' from s to v on P; then recursively, find a walk P_1 from s to v with budget B' and a walk P_2 from v to t with budget $B - B'$ *to augment* P_1; and finally, output P as the concatenation of P_1 and P_2.

The algorithm above is efficiently mainly because when the middle vertex v and the first-half budget B' are fixed, we will first greedily pick a good path P_1 for the first half, and then a good path P_2 for the second half to augment P_1 (but never consider changing P_1 when selecting P_2: analogue to greedily picking an element in the Greedy algorithm for Max-k-Cover).

GreedyGuess(s, t, B, X, i):
If $l(s, t) > B$ then return Infeasible.
Initialization: Let $P \leftarrow s \cdot t$ and *best* $\leftarrow f_X(P)$.
Base case: If $i = 0$ then return P.
For each $v \in V$ (middle vertex) and each $1 \leq B_1 \leq B$ (first-half budget) do:
  $P_1 \leftarrow$ GreedyGuess$(s, v, B_1, X, i - 1)$.
  $P_2 \leftarrow$ GreedyGuess$(v, t, B - B_1, X \cup V(P_1), i - 1)$.
  If $f_X(P_1 \cdot P_2) >$ *best* then $P \leftarrow P_1 \cdot P_2$ and *best* $\leftarrow f_X(P)$.
Return P.

It is easy to see the running time of GreedyGuess is $O((2nB)^i \cdot T_f)$, where T_f is the time to compute f on a given set.

For the approximation ratio, we have the following main lemma.

Lemma 2.7 *Let $P^* \in \mathcal{P}(s, t, B)$ such that $P^* = v_0 v_1 \dots v_k$ where $v_0 = s, v_k = t$. Let P be the path returned by* GreedyGuess(s, t, B, X, i). *If $i \geq \lceil 1 + \log k \rceil$, then $f_X(P) \geq f_X(P^*)/\lceil 1 + \log k \rceil$.*

Proof. The proof is by induction on k. Suppose $v = v_{\lceil k/2 \rceil}$ is the middle vertex in P^*. Let P_1^* be the s-v subpath on P^* and P_2^* be the v-t subpath on P^*. Also, let $B_1 = l(P_1^*)$ and $B_2 = l(P_2^*)$. Suppose in some iteration, GreedyGuess(s, t, B, X, i) correctly guesses v and B_1; let P_1 and P_2

be returned by the recursive calls. Let $X' = X \cup V(P_1)$, from the induction, we have

$$f_X(P_1) \geq \frac{1}{\lceil \log k \rceil} f_X(P_1^*) \text{ and } f_{X'}(P_2) \geq \frac{1}{\lceil \log k \rceil} f_{X'}(P_2^*).$$

So, $f_X(P) = f_X(P_1) + f_{X'}(P_2) \geq \frac{1}{\lceil \log k \rceil} (f_X(P_1^*) + f_{X'}(P_2^*))$. Also note that

$$f_{X'}(P_2^*) = f(P_2^* \cup P_1 \cup X) - f(P_1 \cup X) = f_X(P_2^* \cup P_1) - f_X(P_1) \geq f_{X \cup P_1^*}(P_2^*) - f_X(P_1).$$

From the above two inequalities, we can conclude that $f_X(P) \geq \frac{1}{1+\lceil \log k \rceil} f_X(P^*)$. □

We can guess the length k (number of vertices) of the optimum path, and GreedyGuess yields a $O(\log k)$-approximation in $O((2nB)^{\log k})$ time.

FastGreedyGuess **algorithm.** We can improve GreedyGuess to obtain a $(n \log B)^{O(\log n)}$-time algorithm FastGreedyGuess while preserving the approximation ratio $O(\log k)$.

The modification is minor: In each iteration of GreedyGuess, instead of guessing the first-half budget B_1, we guess the first-half profit p_1 ($\leq f(V)$) and we use the binary search to find the minimum B_1 such that FastGreedyGuess$(s, v, B_1, X, i-1)$ returns a path P_1 with $f(P_1) \geq p_1$. The total running time is thus bounded by $O((n \cdot f(V) \cdot \log B)^{\log k}) = (n \log B)^{O(\log n)}$ as $f(V)$ is poly-bounded in n. And, the approximation ratio is not changed.

Theorem 2.8 *Given G, s, t, budget B and submodular function f, there is a quasi-polynomial time algorithm that finds an $O(\log \mathrm{OPT})$ approximation for the problem of maximizing submodular function under path constraint.*

2.4 Other Constraints

Group budget constraint. Given a collection of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ and its partition $\mathcal{S} = \mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \mathcal{G}_d$, we want to select k sets from $\mathcal{S}$ with the additional group budget constraint that at most one set can be selected from each group $\mathcal{G}_i$. [3] gives an 1/2-approximation to this problem. We can observe that it is a special case of maximizing monotone submodular function under the (partition) matroid constraint.

Linear constraints (also called knapsack constraint). Given a ground set S, let $w^1, \ldots, w^k \in \mathbb{R}_+^S$ be k weight vectors and $C^1, C^2, \ldots, C^k \in \mathbb{R}_+$ be capacities. A set S satisfies the k linear constraints iff $\sum_{j \in S} w_j^i \leq C^i \quad \forall i \in \{1, \ldots, k\}$. $(1 - 1/e)$-approximation algorithms have been obtained for maximizing monotone submodular function under one or more linear constraints [27, 20].

Multiple matroid constraints. A greedy algorithm is known to produces an $\frac{1}{p+1}$-approximation for maximizing a monotone submodular function within the intersection of p matroids [9]. After 30 years, [22] improves the approximation ratio to $\frac{1}{p+\epsilon}$ using a local-search algorithm. The algorithm and analysis in [22] can be also applied to a linear objective function and a non-monotone submodular function with approximation ratios $\frac{1}{p-1+\epsilon}$ and $\frac{1}{p+1+1/p+\epsilon}$, respectively.

Combination of linear constraints and matroid constraint. More recently, [5] obtains $(1 - 1/e - \epsilon)$-approximation for the problem of maximizing monotone submodular function subject to one matroid constraint and k linear constraints, which generalizes the result in [20].

3 Maximizing Non-Monotone Submodular Functions

Now we consider the unconstrained maximization of a submodular function $f : 2^X \to \mathbb{R}_+$ which is not necessarily monotone. We only assume the function is nonnegative.

Typical examples of such a problem include Max-Cut and Max-Di-Cut, for which taking a uniformly random vertex subset are $\frac{1}{2}$-approximation and $\frac{1}{4}$-approximation, respectively.

We will present two algorithm in [8] in this section. The first one is, again, taking a uniformly random subset of X, which yields an $\frac{1}{4}$-approximation for submodular function maximization, and $\frac{1}{2}$-approximation for symmetric submodular function. The second one is a deterministic local search algorithm which improves the approximation ratio from $\frac{1}{4}$ to $\left(\frac{1}{3} - \frac{\epsilon}{n}\right)$ for general submodular function. A randomized local search that yields $\left(\frac{2}{5} - o(1)\right)$-approximation is also introduced in [8].

More recently, [21] gives approximation algorithms for maximizing non-monotone submodular function under matroid constraints or linear constraints, with approximation ratios comparable to monotone submodular function maximization.

3.1 Random Set Algorithm

Recall our goal is to maximize a non-monotone submodular function $f : 2^X \to \mathbb{R}_+$. Inspired by the random set algorithm for Max-Cut and Max-Di-Cut, we consider the following algorithm:

RandomSet**:** Return $S = X(1/2)$, i.e., picking each element in X with probability $1/2$.

Theorem 3.1 *In* RandomSet*,* $\mathbf{E}[f(S)] \geq \frac{1}{4}\mathrm{OPT}$*. If* f *is symmetric,* $\mathbf{E}[f(S)] \geq \frac{1}{2}\mathrm{OPT}$*.*

Let $A(p)$ be a random subset of A where each element appears with probability p.

We first claim that $\mathbf{E}[f(A(p))] \geq (1-p)f(\emptyset) + pf(A)$. This can be proved by induction on $|A|$. Suppose $A' = A - \{x\}$ for some $x \in A$. By submodularity, we have $f(A(p)) - f(A(p) - \{x\}) \geq f(A(p) \cup A') - f(A')$. And therefore, $\mathbf{E}[f(A(p))] \geq \mathbf{E}[f(A(p) - \{x\})] + p(f(A) - f(A'))$. From the induction hypothesis, $\mathbf{E}[f(A(p) - \{x\}] \geq (1-p)f(\emptyset) + pf(A')$. And thus,

$$\mathbf{E}[f(A(p))] \geq (1-p)f(\emptyset) + pf(A') + p(f(A) - f(A')) = (1-p)f(\emptyset) + pf(A).$$

Now consider two independent random sets $A(p), B(q)$. Condition on $A(p) = R$ and taking expectation on $B(q)$, we have $\mathbf{E}[f(R \cup B(q))] \geq (1-q)f(R) + qf(R \cup B)$. Now taking expectation on $A(p) = R$, we have $\mathbf{E}[f(A(p) \cup B(q))] \geq (1-p)((1-q)f(\emptyset) + qf(B)) + p((1-q)f(A) + qf(A \cup B)) = (1-p)(1-q)f(\emptyset) + p(1-q)f(A) + (1-p)qf(B) + pqf(A \cup B)$.

Suppose the optimal solution for maximizing f is S^* and let $\bar{S}^* = X \setminus S^*$. We can rewrite $S = X(1/2)$ as $S^*(1/2) \cup \bar{S}^*(1/2)$, and applying the above claim, we have

$$\mathbf{E}[f(S)] \geq \frac{1}{4}f(\emptyset) + \frac{1}{4}f(S^*) + \frac{1}{4}f(\bar{S}^*) + \frac{1}{4}f(X).$$

Therefore, $\mathbf{E}[f(S)] \geq \frac{1}{4}\mathrm{OPT}$ and $\mathbf{E}[f(S)] \geq \frac{1}{2}\mathrm{OPT}$ if $f(S^*) = f(\bar{S}^*)$.

3.2 Local Search Algorithm

We consider the natural local search algorithm for maximizing f. In each iteration, we try to add or delete an element from S to increase $f(S)$ until this is impossible. However, finding such a

local optimum turns out to be hard, and thus we consider an alternative approximate local search algorithm: S is an $(1+\alpha)$-*approximate local optimum* if $(1+\alpha)f(S) \geq f(S \setminus \{x\})$ for any $x \in S$ and $(1+\alpha)f(S) \geq f(S \cup \{x\})$ for any $x \notin S$.

ApproxLocalSearch(α):
1: Let $S \leftarrow \{s\}$ where $f(s) = \max_{x \in S} f(x)$.
2: If there is $x \in S$ such that $f(S \setminus \{x\}) > (1+\alpha)f(S)$, then let $S \leftarrow S \setminus \{x\}$ and goto 2.
3: If there is $x \notin S$ such that $f(S \cup \{x\}) > (1+\alpha)f(S)$, then let $S \leftarrow S \cup \{x\}$ and goto 2.
4: Return the maximum of $f(S)$ and $f(X \setminus S)$.

Of course, ApproxLocalSearch(α) returns an $(1+\alpha)$-approximate local optimum.

Moreover, suppose S is an $(1+\alpha)$-approximate local optimum, for any $I \subseteq S$ or $I \supseteq S$, we have $f(I) \leq (1+n\alpha)f(S)$, where $|S| = n$. Let $I = T_1 \subseteq T_2 \subseteq \ldots \subseteq T_k = S$, where $T_i \setminus T_{i-1} = \{a_i\}$. Then $f(T_i) - f(T_{i-1}) \geq f(S) - f(S \setminus \{a_i\}) \geq -\alpha f(S)$. Summing them up, we have $f(I) = f(T_1) \leq (1+k\alpha)f(T_k) = (1+k\alpha)f(S)$. The case of $I \supseteq S$ is similar.

Now suppose S^* is the optimal solution. We have $f(S^* \cup S) \leq (1+n\alpha)f(S)$ and $f(S^* \cap S) \leq (1+n\alpha)f(S)$. From the submodularity of f, we have $f(S^* \cup S) + f(X \setminus S) \geq f(X) + f(S^* \setminus S) \geq f(S^* \setminus S)$. Putting them together, we have

$$2(1+n\alpha)f(S) + f(X \setminus S) \geq f(S^* \cap S) + f(S^* \cup S) + f(X \setminus S) \geq f(S^* \cap S) + f(S^* \setminus S) \geq f(S^*).$$

Therefore, picking $\alpha = \frac{\epsilon}{n^2}$, we have either $f(S) \geq \left(\frac{1}{3} - \frac{\epsilon}{n}\right)\mathrm{OPT}$ or $f(X \setminus S) \geq \left(\frac{1}{3} - \frac{\epsilon}{n}\right)\mathrm{OPT}$.

For the running time, in the local search process, we improve $f(s)$ to $f(S) \leq nf(s)$ with a factor $1 + \frac{\epsilon}{n^2}$ in each iteration. So there are at most $O(\frac{n^2}{\epsilon}\log n)$ iterations required.

Theorem 3.2 ApproxLocalSearch *is an* $\left(\frac{1}{3} - \frac{\epsilon}{n}\right)$*-approximation algorithm for the problem of non-monotone submodular function maximization.*

4 Applications of Submodular Function Maximization

In this section, we are mainly interested in applications of submodular functions maximization in the areas of machine learning and artificial intelligence. Krause and Guestrin give a good summary of related works before year 2007 [15]. We will include some works introduced in [15] and also survey some recent works after year 2007. The major applications we are interested in are observation selection, sensor placement, social network.

4.1 Observation selection and sensor placement

In many artificial intelligence applications, we need to effectively collect information in order to make best decisions under uncertainty. In this setting, we usually need to trade off the informativeness of the observation and the cost of acquiring the information. When monitoring spatial phenomena using sensor networks, for example, we can decide where to place sensors. Since we have a limited budget, we want to place the sensors only at the most informative locations. Hence we want to select a set $\mathcal{A} \subseteq \mathcal{V}$ of locations, and want to maximize some objective $F(\mathcal{A})$ measuring the informativeness of the selected locations, subject to a constraint on the number of sensors we can place, i.e., $|\mathcal{A}| \leq k$.

There are many variations of the problem to accommodate different scenarios:

- Cardinality and Budget Constraints. Consider the problem with cardinality and budget constraint, i.e., each location $s \in \mathcal{V}$ has a fixed positive cost $c(s)$, and the cost of an observation selection $\mathcal{A} \subseteq \mathcal{V}$, $c(\mathcal{A})$ is defined as $c(\mathcal{A}) = \sum_{s \in \mathcal{A}} c(s)$. The problem then is to solve the following optimization problem:

$$\mathcal{A}^* = \arg\max_{\mathcal{A}} F(\mathcal{A}) \text{ subject to } c(\mathcal{A}) \leq B,$$

  for some nonnegative budget B. The special case $c(\mathcal{A}) = |\mathcal{A}|$ is called the unit cost case.

  A natural heuristic which has been frequently used for unit cost case is the greedy algorithm. This algorithm starts with the empty set $\mathcal{A} = \emptyset$, and iteratively, in round j, it finds the location $s_j = \arg\max_s F(\mathcal{A} \cup \{s\}) - F(\mathcal{A})$, i.e., the location which increases the objective the most, and adds it to the current set $\mathcal{A}$. The algorithm stops after B sensors have been chosen. Nemhause *et al.* has proved that this straight-forward greedy solution achieves an objective value that achieves at least a factor $(1 - \frac{1}{e})$ of the optimal score [25]. Readers can refer to Section 2.1 for a review on the algorithm and analysis. When $c(s)$ is arbitrary, a slight modification of the algorithm, combining the greedy selection of the location with the highest benefit-cost ratio, with a partial enumeration scheme achieves the same constant factor $(1 - \frac{1}{e})$ approximation guarantee [27, 12].

- Path Constraints. In the robotic lake monitoring scenario, the observation selection problem is even more complex: Here, the locations $s \in \mathcal{V}$ are nodes in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, d)$, with edge weights d encoding distance. The goal is to find a collection of paths P_i in this graph, one for each of the k robots, providing highest information $F(P_1 \cup \cdots \cup P_k)$, subject to a constraint on the path length. In this setting, the simple greedy algorithm could perform arbitrarily badly. In [26], they show that the multiple robot problem can be reduced to optimizing paths for a single robot. They prove that if we have any single robot algorithm achieving an approximation guarantee of η, a simple sequential allocation strategy - iteratively optimizing one path at a time - achieves (nearly) the same approximation guarantee $(\eta + 1)$ for the case of multiple robots. For single robot algorithm, the approach in [4] can be applied, which is reviewed in Section 2.3.

- Communication Constraints. Often, when placing sensors, the sensors must be able to communicate with each other. In the case of wireless sensor networks (WSN), for example, the sensors need to communicate through lossy links which deteriorate with distance, causing message loss and increased power consumption. We can also model this problem by considering the potential locations as nodes in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, d)$. In Krause 2006, they assign the expected number of retransmissions (i.e., the expected number of times a message has to be resent in order to warrant successful transmission) between locations u and v as edge cost $d(u, v)$. The cost of a set of locations $\mathcal{A} \subset \mathcal{V}$ is then the minimum cost of connecting the locations $\mathcal{A}$ in the graph $\mathcal{G}$. In [13], a randomized approximation algorithm is presented. They provide a sensor placement for which the communication cost is at most a small factor (at worst logarithmically) larger, and for which the expected sensing quality is at most a constant factor worse than the optimal solution.

- Selection Against Adversarial Objectives. These problems can be interpreted as a game: We select a set of observations (sensor locations, experiments), and an adversary selects an

objective function (location to evaluate predictive variance, model parameters etc.) to test us on. Often, the individual objective functions (e.g., the marginal variance at one location, or the information gain for a fixed set of parameters) satisfy submodularity. In [17], it shows that simple myopic algorithm performs arbitrarily badly in the case of an adversarially chosen objective. [17] presents an efficient algorithm for settings where an adversarially-chosen submodular objective function must be optimized. The algorithm guarantees solutions which are at least as informative as the optimal solution, at only a slightly higher cost. They prove that the approximation guarantee is best possible and cannot be improved unless NP-complete problems admit efficient algorithms.

- Gaussian Process. When monitoring spatial phenomena, which can often be modeled as Gaussian processes (GPs), choosing sensor locations is a fundamental task. There are several common strategies to address this task, for example, geometry or disk models, placing sensors at the points of highest entropy (variance) in the GP model. In [19], it tackles the combinatorial optimization problem of maximizing the mutual information between the chosen locations and the locations which are not selected. They prove that the problem of finding the configuration that maximizes mutual information is NP-complete. To address this issue, they describe a polynomial-time approximation that is within $(1 - \frac{1}{e})$ of the optimum by exploiting the submodularity of mutual information. They also show how submodularity can be used to obtain online bounds, and design branch and bound search procedures. The algorithm is extended to exploit lazy evaluations and local structure in the GP, yielding significant speedups.

- Water Sensor Network. In oder to detect the malicious introduction of contaminants, people deploy sensors in a large water distribution network. To catalyze the development of new approaches, the Battle of Water Sensor Networks challenge was organized by Ostfeld *et al.* during the Water Distribution Systems Analysis Symposium 2006. In this challenge, sensor placements were to be designed for two realistic water distribution networks, and several intrusion settings as formalized in Ostfeld *et al.* [6]. The contributed sensor placements were evaluated with respect to four realistic objective functions - the time until an intrusion is detected, the expected population affected by an intrusion, the expected amount of contaminated water consumer, and likelihood of detection. The approach presented in [16] is based on the key observation that the benefit of placing sensors, evaluated according to the BWSN objectives, satisfies submodularity, an intuitive diminishing returns property. [16] uses this property to develop fast algorithms with provable guarantees which can handle all settings defined in BWSN, and actually scale for beyond the problem instances defined in this challenge. They propose a highly efficient algorithm, which scales to networks of tens of thousands of nodes and beyond, and to millions of possible intrusion scenarios. And they prove rigorous theoretical worst-case bounds about the performance: the solutions produced are guaranteed to be within 63% of the optimal solution, within computational time proportional to the number of nodes and scenarios considered. In experiment, they show that the approach also compute online bounds that show that the solutions are usually within 95% of the optimal solution. These online bounds can be applied to the sensor placements returned by any algorithm for BWSN.

- Scheduling of Sensors. Previous studies mostly focus on where to locate these sensors to best

predict the phenomenon at the unsensed locations. However, given the power constraints, we also need to determine *when* to selectively activate these sensors in order to maximize the performance while satisfying lifetime requirements. Traditionally, these two problems of sensor placement and scheduling have been considered separately from each other; one first decides where to place the sensors, and then when to activate them. In [18], they present an efficient algorithm to simultaneously optimizes the placement and the schedule. They prove the algorithm provides a constant-factor approximation to the optimal solution of this NP-hard optimization problem. A salient feature of the approach is that it obtains a "balanced" schedules that perform uniformly well over time, rather than only on average. The algorithm applies to complex settings where the sensing quality of a set of sensors is measured, e.g., in the improvement of prediction accuracy (more formally, to situations where the sensing quality function is submodular).They show that, in empirical studies, their approach give drastically improvement on performance compared to separate placement and scheduling (e.g., a 33% improvement in network lifetime on the traffic prediction task).

4.2 Social network

The first work to solve social network influence problem using submodular functions is introduced by Kempe *et al.* [11]. They are motivated by a fundamental algorithmic problem proposed by Domingos and Richardson: if we can try to convince a subset of individuals to adopt a new product or innovation, and the goal is to trigger a large cascade of further adoptions, which set of individuals should we target? In this work, a *Triggering Model* was introduced for modeling the spread of influence in a social network. As the authors show, this model generalizes the Independent Cascade, Linear Threshold and Listen-once models commonly used for modeling the spread of influence. Essentially, this model describes a probability distribution over directed graphs, and the influence is defined as the expected number of nodes reachable from a set of nodes, with respect to this distribution. Kempe *et al.* showed that the problem of selecting a set of nodes with maximum influence is submodular, and hence, the greedy algorithm provides a $(1 - \frac{1}{e})$ approximation.

In Leskovec *et al.* work [23], they generalize the Triggering model to be a special case of the network outbreak detection problem. Outbreak detection can be modeled as selecting nodes (sensor locations, blogs) in a network, in order to detect the spreading of a virus or information as quickly as possible. Outbreak detection problem is fundamentally related to influence problem, one tries to affect as many nodes as possible, while when detecting outbreak, one wants to minimize the effect of an outbreak in the network. In [23], they show that many realistic outbreak detection objectives are submodular, i.e., they exhibit a diminishing returns property: Reading a blog (or placing a sensor) when we have only read a few blogs provides more new information than reading it after we have read many blogs (placed many sensors). They exploit the submodularity of the objective (e.g., detection time) to develop an efficient approximation algorithm, CELF, which achieves near-optimal placements (guaranteeing at least a constant fraction of the optimal solution), providing a novel theoretical result for non-constant node cost functions. CELF is up to 700 times faster than simple greedy algorithm. They also derive novel online bounds on the quality of the placements obtained by any algorithm.

4.3 Other Applications in AI

We discuss several examples of fundamental AI problems which could potentially be addressed using submodularity.

A classical AI problem is probabilistic fault diagnosis. Here, one wants to select a set of tests to probe the system (e.g., a computer network), in order to diagnose the state of unobservable system attributes (e.g., presence of a fault). Krause and Guestrin [14] show that for a wide class of probabilistic graphical models, this criterion F_{IG} is submodular. Often, the cost of a test depends on the tests already chosen (i.e., test A might be cheaper if test B has been chosen). Related applications are test selection for expert systems, e.g., in the medical domain, as well as optimizing test cases for software testing coverage.

In some applications, we have an abundance of information at our disposal, and the scarce resource is the attention of the human, to which this information should be presented. One example is active learning ([10]), where the goal is to learn a classifier, but the training data is initially unlabeled. A human expert can be requested to label a set of examples, but each label is expensive. In some applications, complex constraints are present; e.g., when labeling a stream of video images, labeling a sequence of consecutive images is cheaper than labeling individual images. Hoi [10] show that certain active learning objectives are (approximately) submodular.

5 Conclusions

We review a collection of techniques for maximizing submodular functions, and introduction how they are applied in other areas. In particular, we focus on techniques for maximizing monotone submodular functions under cardinality/budget constraints, matroid constraints, and path constraints, as well as maximizing unconstrained non-monotone submodular functions.

Simple and efficient algorithms are favored by practicers. So that's why the techniques for maximizing monotone submodular functions under cardinality constraints or budget constraints are frequently applied. In the next step, we want to explore the submodularity in database and datamining problems, as well as other constraints that are applicable in these problems.

References

[1] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *IPCO*, pages 182–196, 2007.

[2] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, to appear.

[3] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM*, pages 72–83, 2004.

[4] Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *FOCS*, pages 245–253, 2005.

[5] Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding for matroid polytopes and applications. *CoRR*, abs/0909.4348v2, 2010.

[6] Avi Ostfeld et. al. The battle of the water sensor networks (bwsn): A design challenge for engineers and algorithms. *Journal of Water Resources Planning and Management*, pages 556–568, 2008.

[7] Uriel Feige. A threshold of ln for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

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

[9] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions-ii. *Mathematical Programming Study*, 8:73–87, 1978.

[10] Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *ICML*, pages 417–424, 2006.

[11] David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146, 2003.

[12] A. Krause and C. Guestrin. A note on the budgeted maximization of submodular functions. In *Technical report, CMU-CALD-05-103*, 2005.

[13] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *IPSN*, 2006.

[14] Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *UAI*, pages 324–331, 2005.

[15] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, pages 1650–1654, 2007.

[16] Andreas Krause, Jure Leskovec, Carlos Guestrin, M.ASCE Jeanne VanBriesen, and Christos Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, pages 516–526, 2008.

[17] Andreas Krause, H. Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Selecting observations against adversarial objectives. In *NIPS*, 2007.

[18] Andreas Krause, Ram Rajagopal, Anupam Gupta, and Carlos Guestrin. Simultaneous placement and scheduling of sensors. In *IPSN*, 2009.

[19] Andreas Krause, Ajit Paul Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

[20] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *SODA*, pages 545–554, 2009.

[21] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *STOC*, pages 323–332, 2009.

[22] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. In *APPROX-RANDOM*, pages 244–257, 2009.

[23] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne M. VanBriesen, and Natalie S. Glance. Cost-effective outbreak detection in networks. In *KDD*, pages 420–429, 2007.

[24] George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[25] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions-i. *Mathematical Programming*, 14(1):265–294, 1978.

[26] A. Singh, A. Krause, C. Guestrin, W. Kaiser, and M. Batalin. Efficient planning of informative paths for multiple robots. In *IJCAIR*, 2007.

[27] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[28] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, pages 67–74, 2008.

[29] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

[30] Laurence A. Wolsey. Maximizing real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3):410–425, 1982.