# An Efficient Evolutionary Algorithm for Subset Selection with General Cost Constraints

**Chao Bian,[1] Chao Feng,[2] Chao Qian,[1]* Yang Yu[1]**
[1]National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
[2]School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China
chaobian12@gmail.com, chaofeng@mail.ustc.edu.cn, {qianc, yuy}@lamda.nju.edu.cn

**Abstract**

In this paper, we study the problem of selecting a subset from a ground set to maximize a monotone objective function $f$ such that a monotone cost function $c$ is bounded by an upper limit. State-of-the-art algorithms include the generalized greedy algorithm and POMC. The former is an efficient fixed time algorithm, but the performance is limited by the greedy nature. The latter is an anytime algorithm that can find better subsets using more time, but without any polynomial-time approximation guarantee. In this paper, we propose a new anytime algorithm EAMC, which employs a simple evolutionary algorithm to optimize a surrogate objective integrating $f$ and $c$. We prove that EAMC achieves the best known approximation guarantee in polynomial expected running time. Experimental results on the applications of maximum coverage, influence maximization and sensor placement show the excellent performance of EAMC.

## Introduction

The subset selection problem is a general NP-hard problem with many applications, such as maximum coverage (Feige 1998), influence maximization (Kempe, Kleinberg, and Tardos 2003) and sensor placement (Krause, Singh, and Guestrin 2008), to name a few. The goal is to select a subset of size at most $B$ from a ground set of $n$ items for maximizing some given monotone submodular function $f$, i.e.,

$$\arg\max_{X\subseteq V} f(X) \quad \text{s.t.} \quad |X| \leq B,$$

where $f : 2^V \to \mathbb{R}$ is monotone submodular. Note that submodularity is an attractive property encoding a natural diminishing returns condition. It is known that the greedy algorithm, which iteratively adds one item with the largest marginal gain on $f$, achieves the optimal polynomial-time approximation guarantee of $(1-1/e)$ (Nemhauser, Wolsey, and Fisher 1978; Nemhauser and Wolsey 1978).

Since there are also many applications involving complex constraints, much attention has been drawn to the problem with general cost constraints $c(X) \leq B$. The linear cost constraint (i.e., $c$ is a positive linear function) was first considered. Khuller *et al.* (1999) showed that the greedy algorithm fails to guarantee a bounded approximation ratio. Krause and Guestrin (2005) proved that the generalized greedy algorithm can achieve an approximation ratio of $(1/2)(1-1/e)$. In each iteration, the generalized greedy algorithm adds one item with the largest ratio of the marginal gain on $f$ and $c$. The subset found by this rule is compared with the best single item, and the better one is returned as the final output. Its approximation ratio can be improved to $(1-1/e)$ by a partial enumeration heuristic (Krause and Guestrin 2005), which, however, leads to impractical computation time.

Later, the monotone submodular cost constraint (i.e., $c$ is a monotone submodular function) was considered. Note that a positive linear function must be monotone submodular. Iyer and Bilmes (2013) proposed several algorithms with bounded approximation guarantees by using suitable surrogate functions for $f$ and $c$ to optimize over. Considering that the cost function $c$ can be non-submodular in some real applications such as mobile robotic sensing and door-to-door marketing, Zhang and Vorobeychik (2016) studied the more general case where $c$ is only required to be monotone. They proved that the generalized greedy algorithm can achieve an approximation ratio of $(1/2)(1-1/e)$, but compared with the optimal solution for a budget slightly smaller than $B$.

Besides relaxing the constraints, non-submodular objective functions have also been studied, e.g., (Bian et al. 2017; Elenberg et al. 2018; Bogunovic, Zhao, and Cevher 2018; Qian et al. 2019). They have many applications, such as Bayesian experimental design (Krause, Singh, and Guestrin 2008), dictionary selection (Krause and Cevher 2010), sparse regression (Das and Kempe 2011), and unsupervised feature selection (Feng, Qian, and Tang 2019).

Recently, Qian *et al.* (2017) considered the very general problem of maximizing monotone functions with monotone cost constraints, i.e.,

$$\arg\max_{X\subseteq V} f(X) \quad \text{s.t.} \quad c(X) \leq B, \tag{1}$$

where both $f$ and $c$ are monotone, but not necessarily submodular. They proved that the generalized greedy algorithm is still a good approximation solver, achieving an approxi-

*This work was supported by the NSFC (61603367, 61876077) and the JiangsuSF (BK20170013). Chao Qian is the corresponding author.

mation ratio of $(\alpha_f/2)(1-e^{-\alpha_f})$, where $\alpha_f$ is the submodularity ratio measuring how close $f$ is to submodularity.

It is also well known that the performance of the generalized greedy algorithm may be limited by the greedy nature. Qian *et al.* (2017) thus proposed an anytime algorithm POMC which can use more time to find better solutions. Note that the generalized greedy algorithm is a fixed time algorithm. By adopting the Pareto optimization technique (Friedrich and Neumann 2015; Qian, Yu, and Zhou 2015), POMC employs a simple multi-objective evolutionary algorithm (EA) to maximize $f$ and minimize $c$ simultaneously. POMC can also achieve the approximation ratio of $(\alpha_f/2)(1-e^{-\alpha_f})$. Furthermore, Roostapour *et al.* (2019) showed the advantage of POMC in dynamic environments. When the budget $B$ changes dynamically, POMC can maintain the $(\alpha_f/2)(1-e^{-\alpha_f})$-approximation ratio, while the generalized greedy algorithm cannot.

However, we find from the theoretical results in (Qian et al. 2017; Roostapour et al. 2019) that either to achieve the approximation ratio of $(\alpha_f/2)(1-e^{-\alpha_f})$ in static environments or to maintain this approximation ratio in dynamic environments, the required expected running time of POMC is unbounded, which can be exponential. This implies that POMC is not a polynomial-time approximation algorithm, making it undesirable for the concerned problem Eq. (1).

In this paper, we propose a new anytime algorithm called EAMC, which employs a simple EA to maximize the surrogate objective $f(X)/(1-e^{-\alpha_f c(X)/B})$. We prove that EAMC can achieve the $(\alpha_f/2)(1-e^{-\alpha_f})$-approximation ratio in at most $2en^2(n+1)$ expected running time, where $n$ is the size of the ground set $V$. We empirically evaluate the performance of EAMC on the applications of maximum coverage, influence maximization and sensor placement. The linear cost constraint as well as the routing constraint (where the cost function is monotone non-submodular) are considered. Experimental results show that compared with the generalized greedy algorithm, EAMC can always find better solutions using a little more running time, providing an alternative for solving the problem Eq. (1) as more computational resources are available nowadays.

## Preliminaries

Let $\mathbb{R}$ and $\mathbb{R}^+$ denote the set of reals and non-negative reals, respectively. Given a ground set $V=\{v_1,v_2,\ldots,v_n\}$, we study the functions $f: 2^V \to \mathbb{R}$ over subsets of $V$. A set function $f$ is monotone if $\forall X \subseteq Y$, $f(X) \leq f(Y)$, which implies that adding more items to a set never decreases the function value. Assume w.l.o.g. that monotone functions are normalized, i.e., $f(\emptyset)=0$. A set function $f$ is submodular (Nemhauser, Wolsey, and Fisher 1978) if for any $X \subseteq Y \subseteq V$ and $v \notin Y$, $f(X\cup\{v\})-f(X) \geq f(Y\cup\{v\})-f(Y)$, which intuitively represents the diminishing returns property, i.e., adding an item to a set $X$ gives a larger benefit than adding the same item to a superset $Y$ of $X$.

For a general set function $f$, the notion of submodularity ratio as presented in Definition 1 is used to measure to what extent $f$ has the submodular property. When $f$ is monotone, it holds that (1) $0 \leq \alpha_f \leq 1$, and (2) $f$ is submodular iff $\alpha_f = 1$. Note that there are also other notions, e.g., in (Krause and Cevher 2010; Das and Kempe 2011; Zhou and Spanos 2016), to characterize the closeness of $f$ to submodularity.

**Definition 1** (Submodularity Ratio (Zhang and Vorobeychik 2016))**.** *The submodularity ratio of a non-negative set function $f$ is defined as*

$$\alpha_f = \min_{X\subseteq Y, v\notin Y} \frac{f(X\cup\{v\})-f(X)}{f(Y\cup\{v\})-f(Y)}. \quad (2)$$

The total curvature as presented in Definition 2 characterizes how close a monotone submodular function $f$ is to modularity. It is easy to verify that $1 \geq 1-\kappa_f \geq 0$. In this paper, we use $\kappa_f$ for a general monotone function $f$ as in (Zhang and Vorobeychik 2016; Bian et al. 2017; Qian et al. 2017). Without the submodular property, it holds that $1/\alpha_f \geq 1-\kappa_f \geq 0$ by Eq. (2).

**Definition 2** (Total Curvature (Conforti and Cornuéjols 1984; Vondrák 2010))**.** *Let $f$ be a monotone submodular set function. The total curvature of $f$ is*

$$\kappa_f = 1 - \min_{v\in V: f(\{v\})>0} \frac{f(V)-f(V\setminus\{v\})}{f(\{v\})}.$$

The concerned problem is presented in Definition 3, which is to maximize a monotone objective function $f$ such that a monotone cost function $c$ is upper bounded by a budget $B$. Both $f$ and $c$ are not necessarily submodular. Assume that $f$ is given by a value oracle, i.e., for any subset $X$, an algorithm can query an oracle to obtain the value of $f(X)$. As in (Zhang and Vorobeychik 2016; Qian et al. 2017), we assume that instead of the exact cost function $c$, only an $\psi(n)$-approximation $\hat{c}$ can be obtained, where $\forall X \subseteq V: c(X) \leq \hat{c}(X) \leq \psi(n)\cdot c(X)$. This is because the exact computation of $c$ may require exponential time in some real-world applications.

**Definition 3** (The General Problem)**.** *Given a monotone objective function $f: 2^V \to \mathbb{R}^+$, a monotone cost function $c: 2^V \to \mathbb{R}^+$ and a budget $B$, to find*

$$\arg\max_{X\subseteq V} f(X) \quad s.t. \quad c(X) \leq B. \quad (3)$$

Here are three applications with monotone submodular objective functions, that will be empirically studied in this paper. Given a family of sets that cover a universe of elements, maximum coverage (Feige 1998) is to select some sets whose union is maximal under a cost budget. It is easy to verify that $f$ is monotone and submodular.

**Definition 4** (Maximum Coverage)**.** *Given a set $U$ of elements, a collection $V=\{S_1,S_2,\ldots,S_n\}$ of subsets of $U$, a monotone cost function $c: 2^V \to \mathbb{R}^+$ and a budget $B$, to find*

$$\arg\max_{X\subseteq V} f(X) = \left|\bigcup_{S_i\in X} S_i\right| \quad s.t. \quad c(X) \leq B.$$

Influence maximization is to identify a set of influential users in social networks. Let a directed graph $G=(V,E)$ represent a social network, where each node is a user and each edge $(u,v)\in E$ has a probability $p_{u,v}$ representing the influence strength from user $u$ to $v$. Given a budget $B$, influence maximization is to find a subset $X$ of $V$ such that the

expected number of nodes activated by propagating from $X$ is maximized (Kempe, Kleinberg, and Tardos 2003). A fundamental propagation model is Independence Cascade (IC). It uses a set $A_t$ to record the nodes activated at time $t$, and at time $t+1$, each inactive neighbor $v$ of $u \in A_t$ becomes active with probability $p_{u,v}$. This process is repeated until no nodes get activated at some time. The set of nodes activated by propagating from $X$ is denoted as $IC(X)$, which is a random variable. The objective $\mathbb{E}[|IC(X)|]$ called influence spread is monotone and submodular. Note that $\mathbb{E}[\cdot]$ denotes the expectation of a random variable.

**Definition 5** (Influence Maximization)**.** *Given a directed graph $G = (V, E)$, edge probabilities $p_{u,v}$ where $(u, v) \in E$, a monotone cost function $c : 2^V \to \mathbb{R}^+$ and a budget $B$, to find*

$$\arg\max_{X \subseteq V} f(X) = \mathbb{E}[|IC(X)|] \quad s.t. \quad c(X) \leq B.$$

Sensor placement (Krause, Singh, and Guestrin 2008) is to decide where to place a limited number of sensors such that the uncertainty is mostly reduced. Let $o_j$ denote a random variable representing the observations collected from location $v_j$ by installing a sensor. Note that the conditional entropy (i.e., remaining uncertainty) of a total set $U$ of random variables having observed a subset $S$ is $H(U \mid S) = H(U) - H(S)$, where $H(\cdot)$ denotes the entropy. Thus, the goal is to select a subset $X$ of locations maximizing the entropy of $\{o_j \mid v_j \in X\}$. It is known that the entropy $H(\cdot)$ is monotone and submodular.

**Definition 6** (Sensor Placement)**.** *Given $n$ locations $V = \{v_1, v_2, \ldots, v_n\}$, a monotone cost function $c : 2^V \to \mathbb{R}^+$ and a budget $B$, to find*

$$\arg\max_{X \subseteq V} H(\{o_j \mid v_j \in X\}) \quad s.t. \quad c(X) \leq B.$$

For these applications, the cost constraint can be a simple size constraint, i.e., $c(X) = |X|$, or a general linear cost constraint, i.e., $c(X) = \sum_{i:v_i \in X} c_i$. In some situations, it, however, can be more complex, e.g., a routing constraint, which is monotone non-submodular. For example, in mobile robotic sensing domains, the costs of moving between locations as well as that of making measurements at locations need to be counted. Let a graph $G = (V, E)$ characterize the routing network of all locations, where $c_e$ denotes the cost of traversing an edge $e \in E$ and $c_v$ denotes the cost of visiting a node $v \in V$. The cost function is $c(X) = c_R(X) + \sum_{v \in X} c_v$ (Zhang and Vorobeychik 2016), where $c_R(X)$ is the cost of the shortest walk to visit each node in $X$ at least once, which is generally non-submodular (Herer 1999) and cannot be exactly computed in polynomial time. In the experiments, both linear cost and routing constraints will be used.

## Previous Algorithms

In this section, we introduce two state-of-the-art algorithms for the concerned problem, i.e., maximizing monotone functions with monotone cost constraints.

### The Generalized Greedy Algorithm

As shown in Algorithm 1, the generalized greedy algorithm selects one item maximizing the ratio of the marginal gain on $f$ and $\hat{c}$ in each iteration. After examining all items (i.e., $V' = \emptyset$), the found subset is compared with the best single item (i.e., $u^*$ in line 9), and the better one is returned.

**Algorithm 1** Generalized Greedy Algorithm

**Input**: a monotone objective function $f$, a monotone approximate cost function $\hat{c}$, and a budget $B$
**Output**: a solution $X \subseteq V$ with $\hat{c}(X) \leq B$
**Process**:

1: Let $X = \emptyset$ and $V' = V$;
2: **repeat**
3: $\quad v^* \in \arg\max_{v \in V'} \frac{f(X \cup \{v\}) - f(X)}{\hat{c}(X \cup \{v\}) - \hat{c}(X)}$;
4: $\quad$ **if** $\hat{c}(X \cup \{v^*\}) \leq B$ **then**
5: $\quad\quad X = X \cup \{v^*\}$
6: $\quad$ **end if**
7: $\quad V' = V' \setminus \{v^*\}$
8: **until** $V' = \emptyset$
9: Let $u^* \in \arg\max_{u \in V : \hat{c}(\{u\}) \leq B} f(\{u\})$
10: **return** $\arg\max_{S \in \{X, \{u^*\}\}} f(S)$

Zhang and Vorobeychik (2016) first proved that for the problem in Definition 3 where $f$ is submodular, the generalized greedy algorithm can obtain a subset $X$ satisfying

$$f(X) \geq (1/2)(1 - 1/e) \cdot f(\tilde{X}),$$

where

$$f(\tilde{X}) = \max\Big\{f(X) \mid c(X) \leq B \cdot \frac{\alpha_{\hat{c}}(1 + \alpha_c^2(K_c - 1)(1 - \kappa_c))}{\psi(n) K_c}\Big\}, \tag{4}$$

and $K_c = \max\{|X| \mid c(X) \leq B\}$, i.e., the largest size of a subset satisfying the constraint. As $0 \leq \alpha_{\hat{c}}, \alpha_c \leq 1$, $1 - \kappa_c \leq 1/\alpha_c$ and $\psi(n) \geq 1$, it holds that

$$\frac{\alpha_{\hat{c}}(1 + \alpha_c^2(K_c - 1)(1 - \kappa_c))}{\psi(n) K_c} \leq 1.$$

Thus, $\tilde{X}$ is actually an optimal solution of the problem Eq. (3) with a slightly smaller budget constraint. Qian *et al.* (2017) extended the analysis to the general situation where $f$ is not necessarily submodular, and proved the approximation ratio of $(\alpha_f/2)(1 - e^{-\alpha_f})$ w.r.t. $f(\tilde{X})$, as shown in Theorem 1.

**Theorem 1.** *(Qian et al. 2017) For the problem in Definition 3, the generalized greedy algorithm finds a subset $X \subseteq V$ with*

$$f(X) \geq (\alpha_f/2) \cdot (1 - e^{-\alpha_f}) \cdot f(\tilde{X}),$$

*where $f(\tilde{X})$ is defined in Eq.* (4)*.*

### The POMC Algorithm

Though the generalized greedy algorithm is an efficient fixed time algorithm, its performance may be limited due to the greedy nature. Based on Pareto Optimization (Friedrich and Neumann 2015; Qian, Yu, and Zhou 2015), Qian *et al.* (2017) proposed an anytime algorithm POMC for

maximizing Monotone functions with monotone Cost constraints. POMC can use more time to find better subsets.

A subset $X \subseteq V$ can be naturally represented by a Boolean vector $\boldsymbol{x} \in \{0,1\}^n$, where the $i$-th bit $x_i = 1$ iff $v_i \in X$. For notational convenience, we will not distinguish $\boldsymbol{x} \in \{0,1\}^n$ and its corresponding subset $X$. The idea of POMC is to reformulate the constrained problem Eq. (3) as a bi-objective maximization problem

$$\arg\max_{\boldsymbol{x}\in\{0,1\}^n} \big(f_1(\boldsymbol{x}),\ f_2(\boldsymbol{x})\big), \tag{5}$$

where $f_1(\boldsymbol{x}) = f(\boldsymbol{x})$ and $f_2(\boldsymbol{x}) = -\hat{c}(\boldsymbol{x})$. In other words, POMC tries to maximize the objective function $f$ and minimize the approximate cost function $\hat{c}$ simultaneously. Note that the solutions with large constraint violation (e.g., $\hat{c}(\boldsymbol{x}) \geq 2B$ in (Qian et al. 2017)) can be excluded from the optimization process by setting their $f_1$ values to $-\infty$.

The domination relationship is used to compare solutions in the bi-objective setting. For two solutions $\boldsymbol{x}$ and $\boldsymbol{x}'$, $\boldsymbol{x}$ *weakly dominates* $\boldsymbol{x}'$, denoted as $\boldsymbol{x} \succeq \boldsymbol{x}'$, if $f_1(\boldsymbol{x}) \geq f_1(\boldsymbol{x}') \wedge f_2(\boldsymbol{x}) \geq f_2(\boldsymbol{x}')$; $\boldsymbol{x}$ *dominates* $\boldsymbol{x}'$, denoted as $\boldsymbol{x} \succ \boldsymbol{x}'$, if $\boldsymbol{x} \succeq \boldsymbol{x}'$ and either $f_1(\boldsymbol{x}) > f_1(\boldsymbol{x}')$ or $f_2(\boldsymbol{x}) > f_2(\boldsymbol{x}')$; they are *incomparable* if neither $\boldsymbol{x} \succeq \boldsymbol{x}'$ nor $\boldsymbol{x}' \succeq \boldsymbol{x}$.

To solve the transformed bi-objective maximization problem Eq. (5), POMC employs a simple multi-objective EA, i.e., lines 1-9 of Algorithm 2, inspired from the GSEMO algorithm (Laumanns, Thiele, and Zitzler 2004). Starting from the special solution $0^n$ which represents the empty set (line 1), it iteratively uses bit-wise mutation and domination-based comparison to improve the solutions maintained in the population $P$ (lines 2-9). In each iteration, a parent solution $\boldsymbol{x}$ is first selected from $P$ uniformly at random (line 3); then an offspring solution $\boldsymbol{x}'$ is generated by applying the bit-wise mutation operator to $\boldsymbol{x}$ (line 4); finally the generated offspring solution $\boldsymbol{x}'$ is used to update the population $P$ according to the domination-based comparison (lines 5-7), making $P$ always contain non-dominated solutions generated-so-far. After running $T$ iterations, the solution with the largest $f$ value while satisfying the cost constraint in $P$ is output as the final solution (line 10).

The bit-wise mutation operator as presented in Definition 7 is a global search operator, which can generate any solution $\boldsymbol{x}' \in \{0,1\}^n$ from a solution $\boldsymbol{x}$, with probability $(1/n)^{H(\boldsymbol{x},\boldsymbol{x}')}(1-1/n)^{n-H(\boldsymbol{x},\boldsymbol{x}')}$, where $H(\cdot,\cdot)$ denotes the Hamming distance.

**Definition 7** (Bit-wise Mutation)**.** *For a solution $\boldsymbol{x} \in \{0,1\}^n$, the bit-wise mutation operator generates a new offspring solution by flipping each bit of $\boldsymbol{x}$ independently with probability $1/n$.*

Qian *et al.* (2017) proved that POMC can achieve the approximation ratio of $(\alpha_f/2)(1-e^{-\alpha_f})$ w.r.t. $f(\tilde{X})$ by using at most $enBP_{\max}/\delta_{\hat{c}}$ iterations in expectation, as shown in Theorem 2. Note that the population size of POMC will increase by 1 in one iteration if the newly generated solution is incomparable with the solutions in $P$. Thus, if any two solutions in $\{0,1\}^n$ are incomparable, e.g., when $f(\boldsymbol{x}) = \hat{c}(\boldsymbol{x})$ and each $\boldsymbol{x}$ has a unique $f$ value, $P$ will continue to increase, resulting in an exponential population size. Furthermore, $B$ and $1/\delta_{\hat{c}}$ can also be exponentially large w.r.t. $n$. Thus, the running time of POMC is not polynomially upper bounded.

**Algorithm 2** POMC Algorithm

**Input**: a monotone objective function $f$, a monotone approximate cost function $\hat{c}$, and a budget $B$
**Parameter**: the number $T$ of iterations
**Output**: a solution $\boldsymbol{x} \in \{0,1\}^n$ with $\hat{c}(\boldsymbol{x}) \leq B$
**Process**:

1: Let $\boldsymbol{x} = 0^n$, $P = \{\boldsymbol{x}\}$ and $t = 0$;
2: **while** $t < T$ **do**
3: Select $\boldsymbol{x}$ from $P$ uniformly at random;
4: Generate $\boldsymbol{x}'$ by applying bit-wise mutation to $\boldsymbol{x}$;
5: **if** $\nexists \boldsymbol{z} \in P$ such that $\boldsymbol{z} \succ \boldsymbol{x}'$ **then**
6: $P = (P \setminus \{\boldsymbol{z} \in P \mid \boldsymbol{x}' \succeq \boldsymbol{z}\}) \cup \{\boldsymbol{x}'\}$
7: **end if**
8: $t = t + 1$
9: **end while**
10: **return** $\arg\max_{\boldsymbol{x}\in P:\hat{c}(\boldsymbol{x})\leq B} f(\boldsymbol{x})$

**Theorem 2.** *(Qian et al. 2017) For the problem in Definition 3, POMC with $\mathbb{E}[T] \leq enBP_{\max}/\delta_{\hat{c}}$ finds a subset $X \subseteq V$ with*

$$f(X) \geq (\alpha_f/2) \cdot (1 - e^{-\alpha_f}) \cdot f(\tilde{X}),$$

*where $P_{\max}$ denotes the largest size of the population $P$ during the running of POMC, $\delta_{\hat{c}} = \min\{\hat{c}(X \cup \{v\}) - \hat{c}(X) \mid X \subseteq V, v \notin X\}$ denotes the minimum increment on $\hat{c}$ by adding a single item, and $f(\tilde{X})$ is defined in Eq.* (4)*.*

As more and more computational resources are available, anytime algorithms are appealing now. It, however, still requires polynomial-time approximation guarantees to make the algorithms reliable. Our focus of this work is to develop an anytime algorithm with polynomial-time bounded approximation guarantees for solving the problem Eq. (3).

## The EAMC Algorithm

In this section, we propose a simple EA for maximizing Monotone functions with monotone Cost constraints, called EAMC. Instead of maximizing $f$ in Eq. (3), EAMC introduces a surrogate objective $g$ to be maximized, which can be formally defined as

$$g(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & |\boldsymbol{x}| = 0, \\ f(\boldsymbol{x})/(1 - e^{-\alpha_f \hat{c}(\boldsymbol{x})/B}) & |\boldsymbol{x}| \geq 1. \end{cases} \tag{6}$$

It can be seen that $g$ takes into account both the original objective $f$ and the cost $\hat{c}$. A smaller $\hat{c}$ value or a larger $f$ value will result in a larger $g$ value.

During the optimization procedure, EAMC maintains a population $P$ of solutions, and a newly generated solution $\boldsymbol{x}'$ will be compared only with the solutions in $\texttt{bin}(|\boldsymbol{x}'|)$, which can be formally defined as

$$\texttt{bin}(|\boldsymbol{x}'|) = \{\boldsymbol{z} \in P \mid |\boldsymbol{z}| = |\boldsymbol{x}'|\}.$$

That is, $\boldsymbol{x}'$ is only compared with the solutions in $P$ which have the same size as $\boldsymbol{x}'$. Such a setting can make the population $P$ have a large diversity, and thus, improve the search ability of the algorithm. As the problem Eq. (3) requires to

**Algorithm 3** EAMC Algorithm

**Input**: a monotone objective function $f$, a monotone approximate cost function $\hat{c}$, and a budget $B$
**Parameter**: the number $T$ of iterations
**Output**: a solution $\boldsymbol{x} \in \{0,1\}^n$ with $\hat{c}(\boldsymbol{x}) \leq B$
**Process**:

1: Let $\boldsymbol{u}^0 = \boldsymbol{v}^0 = \boldsymbol{x} = 0^n$, $P = \{\boldsymbol{x}\}$ and $t = 0$;
2: **while** $t < T$ **do**
3: Select $\boldsymbol{x}$ from $P$ uniformly at random;
4: Generate $\boldsymbol{x}'$ by applying bit-wise mutation to $\boldsymbol{x}$;
5: **if** $\hat{c}(\boldsymbol{x}') \leq B$ **then**
6: Let $i = |\boldsymbol{x}'|$;
7: **if** $\texttt{bin}(i) = \emptyset$ **then**
8: $P = P \cup \{\boldsymbol{x}'\}$;
9: $\boldsymbol{u}^i = \boldsymbol{v}^i = \boldsymbol{x}'$
10: **else**
11: **if** $g(\boldsymbol{x}') \geq g(\boldsymbol{u}^i)$ **then**
12: $\boldsymbol{u}^i = \boldsymbol{x}'$
13: **end if**
14: **if** $f(\boldsymbol{x}') \geq f(\boldsymbol{v}^i)$ **then**
15: $\boldsymbol{v}^i = \boldsymbol{x}'$
16: **end if**
17: $P = (P \setminus \texttt{bin}(i)) \cup \{\boldsymbol{u}^i\} \cup \{\boldsymbol{v}^i\}$
18: **end if**
19: **end if**
20: $t = t + 1$
21: **end while**
22: **return** $\arg\max_{\boldsymbol{x} \in P} f(\boldsymbol{x})$

maximize $f$ while satisfying the budget constraint, EAMC only considers $\boldsymbol{x}'$ with $\hat{c}(\boldsymbol{x}') \leq B$; during the running process, besides the solution with the largest $g$ value, each $\texttt{bin}$ maintains the solution with the largest $f$ value generated-so-far.

The procedure of EAMC is described in Algorithm 3. Starting from the empty set $0^n$ (line 1), it repeatedly tries to improve the $g$ value of solutions in each $\texttt{bin}$ (lines 2-21). In each iteration, a new solution $\boldsymbol{x}'$ is generated by randomly flipping bits of an archived solution $\boldsymbol{x}$ selected from the current $P$ (lines 3-4), and only $\boldsymbol{x}'$ satisfying the constraint can be included into $P$ (line 5). If $\texttt{bin}(|\boldsymbol{x}'|) = \emptyset$, $\boldsymbol{x}'$ is added into $P$, and $\boldsymbol{u}^{|\boldsymbol{x}'|}, \boldsymbol{v}^{|\boldsymbol{x}'|}$ are used to maintain the two solutions of size $|\boldsymbol{x}'|$ with the largest $g$ and $f$ value generated-so-far, respectively (lines 7-9); otherwise, $\boldsymbol{x}'$ is compared with the solutions in $\texttt{bin}(|\boldsymbol{x}'|)$, and $\texttt{bin}(|\boldsymbol{x}'|)$ will be updated if the largest $g$ or $f$ value generated-so-far is improved (lines 10-18). It can be seen that each $\texttt{bin}(i)$ contains only the solutions $\boldsymbol{u}^i$ and $\boldsymbol{v}^i$, which can be the same. After $T$ iterations, the solution with the largest $f$ value in $P$ is output (line 22). Note that all solutions in $P$ must satisfy the budget constraint due to line 5.

EAMC requires to know the submodularity ratio $\alpha_f$, which is used in the surrogate objective $g$. For submodular $f$, $\alpha_f = 1$. For non-submodular $f$, the exact computation of $\alpha_f$ may require exponential time, and a lower bound on $\alpha_f$ can be used instead. Note that lower bounds on $\alpha_f$ have been derived for some monotone non-submodular applications, e.g., Bayesian experimental design (Chamon and Ribeiro 2017; Qian, Yu, and Tang 2018; Hashemi et al. 2019) and determinantal function maximization (Qian, Yu, and Tang 2018).

## Theoretical Analysis

In this section, we prove the general approximation bound of EAMC, as shown in Theorem 3. Compared with Theorem 2, EAMC achieves the same approximation guarantee, i.e., $(\alpha_f/2)(1 - e^{-\alpha_f})$, as POMC, but the expected running time is polynomially upper bounded. The proof relies on Lemma 1, which intuitively means that for any subset, the inclusion of some specific item can improve $f$ by at least a quantity proportional to the current distance to the optimum. As in (Zhang and Vorobeychik 2016; Qian et al. 2017), we can assume that $\forall v \in V : \hat{c}(\{v\}) \leq B$, because any $v$ with $\hat{c}(\{v\}) > B$ can be directly removed before optimization.

**Lemma 1.** *(Qian et al. 2017) For any $X \subseteq V$, let $v^* \in \arg\max_{v \notin X} \frac{f(X \cup \{v\}) - f(X)}{\hat{c}(X \cup \{v\}) - \hat{c}(X)}$. It holds that*

$$f(X \cup \{v^*\}) - f(X) \geq \alpha_f \frac{\hat{c}(X \cup \{v^*\}) - \hat{c}(X)}{B} \cdot (f(\tilde{X}) - f(X)),$$

*where $f(\tilde{X})$ is defined in Eq.* (4).

**Theorem 3.** *For the problem in Definition 3, EAMC with $\mathbb{E}[T] \leq 2en^2(n+1)$ finds a subset $X \subseteq V$ with*

$$f(X) \geq (\alpha_f/2) \cdot (1 - e^{-\alpha_f}) \cdot f(\tilde{X}),$$

*where $f(\tilde{X})$ is defined in Eq.* (4).

*Proof.* The theorem is proved by analyzing the increase of a quantity $J_{\max}$, which is defined as

$$J_{\max} = \max\{i \mid \exists \boldsymbol{x} \in \texttt{bin}(i) : g(\boldsymbol{x}) \geq f(\tilde{X})\}.$$

Let $P_{\max}$ denote the largest size of the population $P$ during the running of EAMC. We first show that $J_{\max} \geq 1$ after at most $enP_{\max}$ expected number of iterations. It is easy to see that the solution $0^n$ will always be in $P$, because only the solutions with the same size can be compared and $0^n$ is the unique solution with size 0. By Lemma 1, flipping a specific 0-bit of $0^n$ (i.e., adding a specific item) can generate a new solution $\boldsymbol{x}'$ such that

$$f(\boldsymbol{x}') \geq \alpha_f \frac{\hat{c}(\boldsymbol{x}')}{B} \cdot f(\tilde{X}) \geq (1 - e^{-\alpha_f \hat{c}(\boldsymbol{x}')/B}) \cdot f(\tilde{X}), \quad (7)$$

where the last inequality holds by $\forall r \in \mathbb{R} : 1 - r \leq e^{-r}$. In each iteration, $\boldsymbol{x}'$ can be generated with probability at least $\frac{1}{P_{\max}} \cdot \frac{1}{n}(1 - \frac{1}{n})^{n-1} \geq \frac{1}{enP_{\max}}$, where $\frac{1}{P_{\max}}$ is a lower bound on the probability of selecting $0^n$ from the population and $\frac{1}{n}(1 - \frac{1}{n})^{n-1}$ is the probability of flipping a specific bit of $0^n$ while keeping the other bits unchanged. Thus, the expected number of iterations to generate $\boldsymbol{x}'$ is at most $enP_{\max}$. Note that $g(\boldsymbol{x}') \geq f(\tilde{X})$ by Eqs. (6) and (7). If $\boldsymbol{x}'$ is included into $P$, we have $J_{\max} \geq 1$; otherwise, there must exist one solution in $\texttt{bin}(1)$ with a larger $g$ value, implying $J_{\max} \geq 1$.

Assume that currently $J_{\max} = i$. According to lines 11-13 and line 17 of Algorithm 3, the largest $g$ value of solutions in $\texttt{bin}(i)$ will not decrease, and thus, $J_{\max}$ will not

decrease. This implies that there always exists $\boldsymbol{x} \in \texttt{bin}(i)$ with $g(\boldsymbol{x}) \geq f(\tilde{X})$.

Next we consider the increase of $J_{\max}$ until a solution with the $f$ value at least $(\alpha_f/2)(1-e^{-\alpha_f}) \cdot f(\tilde{X})$ is found. Let $\boldsymbol{x}$ denote the solution with the largest $g$ value. By Lemma 1, flipping a specific 0-bit of $\boldsymbol{x}$ can generate a solution $\boldsymbol{x}'$ with $|\boldsymbol{x}'| = i+1$ and

$$\begin{aligned} f(\boldsymbol{x}') &\geq \alpha_f \frac{\hat{c}(\boldsymbol{x}') - \hat{c}(\boldsymbol{x})}{B} \cdot f(\tilde{X}) \\ &\quad + \Big(1 - \alpha_f \frac{\hat{c}(\boldsymbol{x}') - \hat{c}(\boldsymbol{x})}{B}\Big)(1 - e^{-\alpha_f \hat{c}(\boldsymbol{x})/B}) f(\tilde{X}) \\ &= \left(1 - \Big(1 - \alpha_f \frac{\hat{c}(\boldsymbol{x}') - \hat{c}(\boldsymbol{x})}{B}\Big) e^{-\alpha_f \hat{c}(\boldsymbol{x})/B}\right) f(\tilde{X}) \\ &\geq (1 - e^{-\alpha_f \hat{c}(\boldsymbol{x}')/B}) \cdot f(\tilde{X}), \end{aligned} \tag{8}$$

where the first inequality holds by $g(\boldsymbol{x}) = f(\boldsymbol{x})/(1 - e^{-\alpha_f \hat{c}(\boldsymbol{x})/B}) \geq f(\tilde{X})$ due to $J_{\max} = i$, and the last inequality holds by $\forall r \in \mathbb{R} : 1 - r \leq e^{-r}$. Thus, we have $g(\boldsymbol{x}') \geq f(\tilde{X})$. In each iteration, $\boldsymbol{x}'$ can be generated with probability at least $\frac{1}{P_{\max}} \cdot \frac{1}{n}(1-\frac{1}{n})^{n-1} \geq \frac{1}{enP_{\max}}$, implying that the expected number of iterations until generating $\boldsymbol{x}'$ with $|\boldsymbol{x}'| = i+1$ and $g(\boldsymbol{x}') \geq f(\tilde{X})$ is at most $enP_{\max}$. We then consider two cases for $\hat{c}(\boldsymbol{x}')$.

(1) If $\hat{c}(\boldsymbol{x}') > B$, Eq. (8) implies $f(\boldsymbol{x}') \geq (1 - e^{-\alpha_f}) f(\tilde{X})$. Let $\boldsymbol{y} \in \arg\max_{u \in V : \hat{c}(\{u\}) \leq B} f(\{u\})$. We have

$$\begin{aligned} f(\boldsymbol{x}') &= f(\boldsymbol{x}) + (f(\boldsymbol{x}') - f(\boldsymbol{x})) \\ &\leq f(\boldsymbol{x}) + f(\boldsymbol{x}' \setminus \boldsymbol{x})/\alpha_f \\ &\leq f(\boldsymbol{x}) + f(\boldsymbol{y})/\alpha_f \\ &\leq (f(\boldsymbol{x}) + f(\boldsymbol{y}))/\alpha_f, \end{aligned} \tag{9}$$

where the first inequality holds by Definition 1, and the last holds by $\alpha_f \in [0,1]$. In each iteration, $\boldsymbol{y}$ can be generated by selecting $0^n$ and flipping a specific 0-bit, occurring with probability at least $\frac{1}{enP_{\max}}$. Thus, $\boldsymbol{y}$ can be generated in at most $enP_{\max}$ expected number of iterations. According to the updating procedure of $P$ in lines 7-10 and 14-17, we know that once $\boldsymbol{y}$ is generated, $P$ will always contain a solution $\boldsymbol{z} \in \texttt{bin}(1)$ with $f(\boldsymbol{z}) \geq f(\boldsymbol{y})$. It can also be verified that there always exists one solution $\boldsymbol{z}' \in \texttt{bin}(i)$ with $f(\boldsymbol{z}') \geq f(\boldsymbol{x})$. By line 22 of Algorithm 3, the solution with the largest $f$ value will be finally returned. Thus, EAMC can output a solution with the $f$ value at least

$$\max\{f(\boldsymbol{x}), f(\boldsymbol{y})\} \geq (\alpha_f/2)(1 - e^{-\alpha_f}) \cdot f(\tilde{X}).$$

(2) If $\hat{c}(\boldsymbol{x}') \leq B$, according to the updating procedure of $P$ in lines 7-13 and 17, we know that $\boldsymbol{x}'$ will be added into $P$, because there does not exist $\boldsymbol{z} \in \texttt{bin}(i+1)$ with $g(\boldsymbol{z}) \geq g(\boldsymbol{x}')$; otherwise, $g(\boldsymbol{z}) \geq g(\boldsymbol{x}') \geq f(\tilde{X})$, contradicting with the definition of $J_{\max}$. Now, $J_{\max} = i+1$.

Repeating the above analysis, EAMC will output a solution $\boldsymbol{z}$ with $f(\boldsymbol{z}) \geq (\alpha_f/2)(1 - e^{-\alpha_f}) \cdot f(\tilde{X})$, which achieves the desired approximation guarantee, or $J_{\max}$ will continue to increase until reaching the maximum value $n$. The latter case implies $\hat{c}(1^n) \leq B$, and EAMC will output $1^n$, which is optimal as $f$ is monotone.

Now we examine the total expected number of iterations. To make $J_{\max} \geq 1$, the expected number of iterations is at most $enP_{\max}$; to increase $J_{\max}$ from 1 to $n$, the expected number of iterations is at most $(n-1) \cdot enP_{\max}$; to generate $\boldsymbol{y}$, the expected number of iterations is at most $enP_{\max}$. Because there are at most two solutions in $\texttt{bin}(i)$, where $1 \leq i \leq n-1$ (line 17 of Algorithm 3), and at most one solution in $\texttt{bin}(0)$ and $\texttt{bin}(n)$, we have $P_{\max} \leq 2n$. Thus, the total expected number of iterations is at most $(n+1) \cdot enP_{\max} \leq 2en^2(n+1)$. □

When the exact computation of $\alpha_f$ is difficult, a lower bound (denoted by $\alpha$) on $\alpha_f$ can be used in the surrogate objective $g$, and EAMC achieves the approximation ratio of $(\alpha_f/2) \cdot (1 - e^{-\alpha})$, as shown in Corollary 1. In the proof of Theorem 3, the factor containing the first $\alpha_f$ (i.e., $\alpha_f/2$) in the approximation ratio is derived from Eq. (9), which does not depend on $g$. Thus, this factor keeps unchanged. The factor containing the second $\alpha_f$ (i.e., $1 - e^{-\alpha_f}$) is derived from Eq. (8), which applies the definition of $g$. As $\alpha$ is used in $g$ now, the $\alpha_f$ in this factor changes to $\alpha$ accordingly.

**Corollary 1.** *For the problem in Definition 3, when a lower bound (denoted by $\alpha$) on $\alpha_f$ is applied to the surrogate objective $g$ in Eq.* (6)*, EAMC with $\mathbb{E}(T) \leq 2en^2(n+1)$ finds a subset $X \subseteq V$ with*

$$f(X) \geq (\alpha_f/2) \cdot (1 - e^{-\alpha}) \cdot f(\tilde{X}),$$

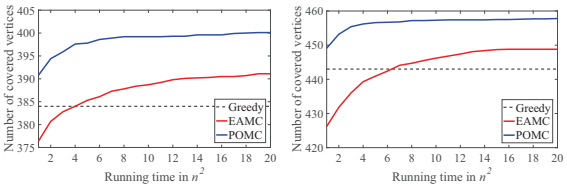
*where $f(\tilde{X})$ is defined in Eq.* (4)*.*

## Empirical Study

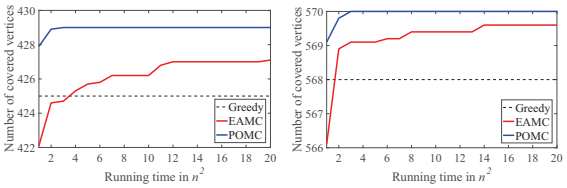
In this section, we empirically examine the performance of EAMC on the applications of maximum coverage, influence maximization, and sensor placement. For each tested instance of each application, we select the generalized greedy algorithm for the baseline, and plot the curve of $f$ over the running time for EAMC. As EAMC is a randomized algorithm, we repeat the run 10 times independently and report the average results.

**Maximum Coverage.** We use two real-world data sets *frb30-15-mis-1* and *frb35-17-mis-1*, from (https://turing.cs.hbg.psu.edu/txn131/vertex_cover.html). Both of them are graphs. The former contains 450 vertices and 17,827 edges, and the latter contains 595 vertices and 27,856 edges. For each vertex, we generate a set which contains the vertex itself and its adjacent vertices. Linear cost constraints are considered, where $c(X) = \sum_{v \in X} c_v$. We use two settings of $c_v$: (1) as in (Harshaw et al. 2019), $c_v = 1 + \max\{d(v) - q, 0\}$, where $d(v)$ is the out-degree of vertex $v$ and $q$ is a constant (which is set to 6 in our experiment); (2) $c_v$ is assigned a random number from $(0, 1]$. They are called "out-degree" and "random" linear cost constraints, respectively. The budgets $B$ are set to 500 and 1, respectively, for these two kinds of settings. The curves are plotted in Figure 1.

**Influence Maximization.** Another two graphs *random_graphs_with_100_vertices* (100 vertices, 3,465 edges) and *random_graphs_with_200_vertices* (200 vertices, 9,950 edges) from (https://turing.cs.hbg.psu.edu/txn131/vertex_cover.html) are used as the social networks, which are briefly

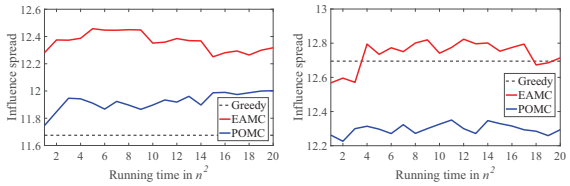



(a) *frb30-15-mis-1*; out-degree (b) *frb35-17-mis-1*; out-degree

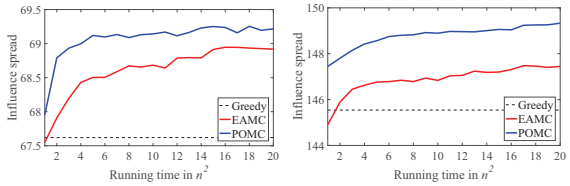



(c) *frb30-15-mis-1*; random (d) *frb35-17-mis-1*; random

Figure 1: Maximum coverage with two kinds of linear cost constraints (out-degree and random). Number of covered vertices: the larger, the better.

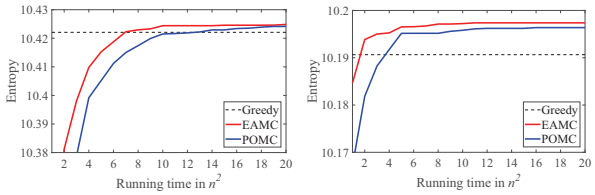



(a) *graph100*; out-degree (b) *graph200*; out-degree




(c) *graph100*; random (d) *graph200*; random

Figure 2: Influence maximization with two kinds of linear cost constraints (out-degree and random). Influence spread: the larger, the better.




(a) *Berkeley* (b) *Beijing*

Figure 3: Sensor placement with the routing constraint. Entropy: the larger, the better.

called *graph100* and *graph200*, respectively. The probability of each edge is set to 0.05. We also consider two settings for the linear cost constraint: out-degree and random. Note that the out-degree linear cost constraint is different from that used in the experiment of maximum coverage. As in (Crawford 2019), $c_v = 1 + (1 + |\xi|) \cdot d(v)$, where $\xi$ is a random number drawn from the normal distribution $\mathcal{N}(0, 0.5^2)$. The budgets $B$ are set to 50 and 3, respectively.

Note that for estimating the influence spread (i.e., the expected number of nodes activated), we simulate the diffusion process 100 times independently and use the average as an estimation. That is, the objective function evaluation is noisy. But for the output solutions of the algorithms, we average over 10,000 times for more accurate estimation. Since the behavior of the greedy algorithm is randomized under noise, we also repeat its run 10 times independently and report the average results. The curves are plotted in Figure 2.

**Sensor Placement.** We use the same setting as in (Qian et al. 2017). Two real-world data sets and the routing constraint are tested. One data set (http://db.csail.mit.edu/labdata/labdata.html) is collected from sensors installed at 54 locations of the Intel Berkeley Research lab, and the other (Zheng, Liu, and Hsieh 2013) is air quality data collected from 36 monitoring stations in Beijing. The light and temperature measures are extracted, respectively. A complete graph is used for the routing network, where the cost of each node is assigned a random number drawn from the normal distribution $\mathcal{N}(0.08, 0.1^2)$, and the cost of each edge corresponds to the physical distance between two locations, which is normalized to $[0, 0.01]$. Note that the entropy is calculated using the observed frequency, and the routing cost is computed approximately by the nearest neighbor method (Rosenkrantz, Stearns, and Lewis 1977). The budget $B$ is set to 1. The curves are plotted in Figure 3.

In Figures 1, 2 and 3, the running time is considered in the number of objective function evaluations, and one unit on the $x$-axis corresponds to $n^2$ evaluations, the running time order of the generalized greedy algorithm. We can observe that EAMC can always find better solutions than the generalized greedy algorithm by performing more objective function evaluations, verifying that EAMC, as an anytime algorithm, can use more time to find better solutions. Note that the curves of EAMC may decrease in Figure 2, which is because the better solution may be deleted due to the noise in evaluating the objective influence spread.

We also plot the curve of $f$ over the running time for POMC. It can be observed that the performance of POMC is not very stable. POMC can perform the best in several cases, but can also perform even worse than the generalized greedy algorithm in Figure 2(b). This may be because the population size of POMC can be very large sometimes. For example, in Figure 2(b), the population size of POMC can reach 70 while that of EAMC is at most 23; in Figure 3(b), the largest population size of POMC can be over 7 (137/19) times that of EAMC. Note that an overly large population size will lead to a small probability of selecting a specific solution for mutation, and thus, deteriorate the efficiency of POMC. The empirical observation is actually consistent with the known theoretical result, i.e., Theorem 2 showing that the upper bound on the expected number of iterations until POMC achieves a desired approximation ratio increases linearly with the largest population size.

## Conclusion

In this paper, we propose an anytime algorithm EAMC for the problem of maximizing monotone functions with monotone cost constraints. We prove that EAMC can achieve the best known approximation guarantee, i.e., $(\alpha_f/2)(1 - e^{-\alpha_f})$, in polynomial expected running time, which overcomes the limitation, i.e., no polynomial-time approximation guarantee, of the existing anytime algorithm POMC. The consistently superior performance of EAMC over the fixed time algorithm, i.e., the generalized greedy algorithm, is empirically shown on various applications.

## References


Bian, A. A.; Buhmann, J. M.; Krause, A.; and Tschiatschek, S. 2017. Guarantees for greedy maximization of non-submodular functions with applications. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 498–507.

Bogunovic, I.; Zhao, J.; and Cevher, V. 2018. Robust maximization of non-submodular objectives. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, 890–899.

Chamon, L., and Ribeiro, A. 2017. Approximate supermodularity bounds for experimental design. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 5403–5412.

Conforti, M., and Cornuéjols, G. 1984. 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.

Crawford, V. G. 2019. An efficient evolutionary algorithm for minimum cost submodular cover. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 1227–1233.

Das, A., and Kempe, D. 2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 1057–1064.

Elenberg, E. R.; Khanna, R.; Dimakis, A. G.; and Negahban, S. 2018. Restricted strong convexity implies weak submodularity. *Annals of Statistics* 46(6B):3539–3568.

Feige, U. 1998. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM* 45(4):634–652.

Feng, C.; Qian, C.; and Tang, K. 2019. Unsupervised feature selection by Pareto optimization. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 3534–3541.

Friedrich, T., and Neumann, F. 2015. Maximizing submodular functions under matroid constraints by evolutionary algorithms. *Evolutionary Computation* 23(4):543–558.

Harshaw, C.; Feldman, M.; Ward, J.; and Karbasi, A. 2019. Submodular maximization beyond non-negativity: Guarantees, fast algorithms, and applications. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2634–2643.

Hashemi, A.; Ghasemi, M.; Vikalo, H.; and Topcu, U. 2019. Submodular observation selection and information gathering for quadratic models. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2653–2662.

Herer, Y. 1999. Submodularity and the traveling salesman problem. *European Journal of Operational Research* 114(3):489–508.

Iyer, R., and Bilmes, J. 2013. Submodular optimization with submodular cover and submodular knapsack constraints. In *Advances in Neural Information Processing Systems 26 (NIPS)*, 2436–2444.

Kempe, D.; Kleinberg, J.; and Tardos, É. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 137–146.

Khuller, S.; Moss, A.; and Naor, J. 1999. The budgeted maximum coverage problem. *Information Processing Letters* 70(1):39–45.

Krause, A., and Cevher, V. 2010. Submodular dictionary selection for sparse representation. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 567–574.

Krause, A., and Guestrin, C. 2005. A note on the budgeted maximization of submodular functions. *Technical Report No. CMU-CALD-05-103*.

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

Laumanns, M.; Thiele, L.; and Zitzler, E. 2004. Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions. *IEEE Transactions on Evolutionary Computation* 8(2):170–182.

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

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

Qian, C.; Shi, J.-C.; Yu, Y.; and Tang, K. 2017. On subset selection with general cost constraints. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2613–2619.

Qian, C.; Yu, Y.; Tang, K.; Yao, X.; and Zhou, Z.-H. 2019. Maximizing submodular or monotone approximately submodular functions by multi-objective evolutionary algorithms. *Artificial Intelligence* 275:279–294.

Qian, C.; Yu, Y.; and Tang, K. 2018. Approximation guarantees of stochastic greedy algorithms for subset selection. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 1478–1484.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015. Subset selection by Pareto optimization. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 1765–1773.

Roostapour, V.; Neumann, A.; Neumann, F.; and Friedrich, T. 2019. Pareto optimization for subset selection with dynamic cost constraints. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2354–2361.

Rosenkrantz, D. J.; Stearns, R. E.; and Lewis, II, P. M. 1977. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing* 6(3):563–581.

Vondrák, J. 2010. Submodularity and curvature: The optimal algorithm. *RIMS Kokyuroku Bessatsu B* 23:253–266.

Zhang, H., and Vorobeychik, Y. 2016. Submodular optimization with routing constraints. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, 819–826.

Zheng, Y.; Liu, F.; and Hsieh, H.-P. 2013. U-air: When urban air quality inference meets big data. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1436–1444.

Zhou, Y., and Spanos, C. 2016. Causal meets submodular: Subset selection with directed information. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2649–2657.