Towards Practical Constrained Monotone Submodular Maximization

Wenxin Li  
Department of ECE  
The Ohio State University  
li.7328@osu.edu

Ness Shroff  
Department of ECE and CSE  
The Ohio State University  
shroff.11@osu.edu

April 24, 2018

Abstract

In this work, we design new approximation algorithms for the problem of maximizing a monotone non-negative submodular function $f(\cdot) : 2^E \rightarrow \mathbb{R}^+$ under various constraints, which improve the state-of-the-art results in terms of the time complexity and/or performance guarantee. Specifically, we first investigate the cardinality constrained monotone submodular maximization problem $\max\{f(S) : |S| \leq k\}$, which has been widely studied for about four decades. We first design an $(1 - 1/e - \varepsilon)$-approximation algorithm which makes $O(n \cdot \max\{\varepsilon^{-1}, \log \log k\})$ value oracle queries to function $f(\cdot)$. To the best of our knowledge, this is the fastest currently known algorithm which achieve almost optimal performance guarantee. Further, we answer the open problem on finding a lower bound on the number of queries. Specifically, we prove that, no (randomized) algorithm could achieve an approximation better than $(\frac{1}{2} + \Theta(1))$ with $o\left(\frac{n}{\log n}\right)$ queries to $f(\cdot)$.

The acceleration in the algorithm for the cardinality constrained problem is achieved through our Adaptive Decreasing Threshold (ADT) algorithm. Based on this technique, we also study the problem of maximizing a monotone submodular function under $p$-system constraint and $d$ knapsack constraints. We show that an $(1/(p + \frac{d}{2} + 1) - \varepsilon)$-approximate solution could be computed using $O\left(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon} \max\{\log \frac{1}{\varepsilon}, \log \log n\}\right)$ oracle queries. Note that our algorithm improves the state of the art in both time complexity and approximation ratio. As a direct consequence, we obtain an $(1/(\frac{d}{2}d + 1) - \varepsilon)$-approximate algorithm under $d$-knapsack constraint. Furthermore we show how to improve this approximation when $d = 1$ while making $O(n \cdot \max\{\varepsilon^{-1}, \log \log k\})$ function evaluations. Finally we investigate the problem of maximizing a monotone submodular function with bounded curvature $\kappa \in [0, 1]$ under a single matroid constraint. We show that we can obtain the almost optimal $(1 - \frac{\kappa}{2} - \varepsilon)$-approximate solution using $\tilde{O}(nk)$ value oracle queries, where $k$ denotes the matroid rank.

We argue that our ADT technique, which provides a faster way of “guessing”, could be utilized to obtain faster algorithms in other problems. Moreover, to establish the randomized query complexity lower bound result, we introduce a general characterization between randomized complexity and deterministic complexity of approximation algorithms by utilizing Yao’s Minimax Principle, we hope that this characterization could be utilized in other problems and may thus be interesting in its own right.
1 Introduction

A set function $f(\cdot) : 2^E \rightarrow \mathbb{R}^+$ is submodular if for all subsets $S, T \subseteq E$, the inequality $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ holds. Submodular functions form a natural class of set functions that have numerous applications. For example, many machine learning problems, which are inherently discrete, such as feature selection [10], document summarization [13, 9, 12, 15] or news recommendation [23], can be cast as submodular maximization problems. Because of this interest, there has been a considerable amount of literature about submodular maximization subject to diverse types of constraints [3, 4, 7, 8].

In this paper, we first consider the canonical problem of maximizing a non-negative monotone submodular function subject to a cardinality constraint. A seminal and simple approach towards this problem is the celebrated greedy algorithm [17]—at each stage, the algorithm selects one element from the available candidates that have maximum marginal gain with respect to the current solution set. The standard greedy algorithm makes $O(k)$ selections, while each such operation requires $O(n)$ marginal gain computations and comparisons, these facts lead to $O(nk)$ time complexity for the greedy algorithm. However, even $O(nk)$ is still not practical for large scale combinatorial problems, especially when evaluating the submodular function $f(\cdot)$ is expensive.

To accelerate the standard greedy paradigm, Badanidiyuru and Vondrák [1] proposed the first and current fastest deterministic algorithm with provable guarantees in both oracle query complexity and approximation ratio in 2013. Instead of choosing the element with the maximum marginal increment, they accept elements whose marginal value is no less than some threshold to avoid the $O(n)$ comparison operations, and the threshold decreases multiplicatively by a factor of $1 - \varepsilon$, to ensure that it achieves the $(1 - \frac{1}{e} - \varepsilon)$ approximation ratio. This algorithm has a query complexity of $O(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon})$. A natural question that arises is: Is $O(\frac{n}{\varepsilon} \log \frac{n}{\varepsilon})$ the query complexity barrier for almost optimal deterministic algorithms, i.e., does there exists a faster deterministic algorithm that achieves the $(1 - \frac{1}{e} - \varepsilon)$ approximation ratio? We answer this question in the affirmative by designing a faster deterministic $(1 - \frac{1}{e} - \varepsilon)$-approximate algorithm in Section 3.1. It is worth-noting that Minoux’s Accelerated Greedy (lazy greedy) [14] algorithm, exploits submodularity to obtain enormous speedups compared with standard greedy, achieves an exact $(1 - \frac{1}{e})$-approximation. However, while Lazy Greedy works well in practice, it does not provide better time-complexity guarantees over the standard greedy algorithm.

On the other hand, in spite of all the attention that this classic problem has received, the fine-grained query complexity remains open [6]—No query complexity lower bound is known [6]. In Section 3.2 we answer this question by proving the first deterministic and randomized query complexity lower bound for the simple cardinality constrained problem $\max \{ f(S) : |S| \leq k \}$. Note that our lower bound results of course hold for the more general types of constraints, such as knapsack constraint, matroid constraint and $p$-system constraint.

Actually one well-known harness result on the approximation ratio is—Nemhauser and Wolsey [17] proved that no algorithm with polynomial number of evaluations to $f(\cdot)$ can achieve an approximation ratio better than $1 - \frac{1}{e}$, which shows that the $1 - \frac{1}{e}$ approximation ratio of standard greedy is essentially tight. However, it has been observed that in practice, the performance of greedy is often better than this tight theoretical result, and even close to optimal citesharma2015greedy. To explain and quantify this phenomenon, the notation of curvature is widely used to obtain quantification analysis, which is introduced by Conforti and Cornuédjols [5]. The curvature of a monotone
submodular function is formally defined as \( \kappa = 1 - \min_{e \in E} \frac{f(E) - f(E \setminus \{e\})}{f(e)} \). They prove that the greedy algorithm actually gives a \( \frac{1 - e^{-\kappa}}{\kappa} \)-approximation solution for monotone submodular objective function with curvature \( \kappa \), which was the state-of-the-art \cite{21} until a modified continuous greedy and local search was proposed by Sviridenko, Vondrák and Ward to achieve a factor of \( 1 - \frac{\kappa}{e} - \varepsilon \) approximation \cite{20}. The latter result is indeed the best possible in the value oracle model. However, compared with the simple greedy algorithm, these two algorithms are computationally expensive. Moreover, it is known that the notation of curvature exhibits a transition between modular functions (easy) and submodular functions (hard), one may not expect a complicated optimal algorithm for the class of submodular functions with curvature information, which are naturally expected to be easier to deal with. Hence another question that arises is, whether it is possible to obtain the almost optimal \( (1 - \frac{\kappa}{e} - \varepsilon) \)-approximation solution in a faster way? We also give an affirmative answer to this question by presenting a more practical almost optimal algorithm.

Our results are formally summarized in the following Section 1.1.

### 1.1 Main Results

We first present an Adaptive Threshold Decreasing (ADT) Algorithm for the cardinality constraint with the following performance guarantee.

**Theorem 1.** Let \( f : 2^E \to \mathbb{R}^+ \) be a non-negative monotone submodular function and \( k \) be an integer such that \( k \leq |E| = n \). There is a deterministic algorithm that finds a solution \( S \subseteq E \) with size \( |S| \leq k \) such that \( f(S) \geq (1 - \frac{1}{e} - \varepsilon) \max_{T \subseteq E, |T| \leq k} f(T) \) and the algorithm performs \( O(n \cdot \max\{\frac{1}{\varepsilon}, \log \log k\}) \) value oracle queries.

In addition, we prove some harness of approximation results for this classic problem, which answers the open problem in \cite{6}. By applying some adversary arguments, we are able to exhibit the following performance barrier.

**Theorem 2.** For the problem of maximizing a monotone submodular function \( f(\cdot) \) under a cardinality constraint: \( \max_{T \subseteq E, |T| \leq k} f(S) \), any (possible randomized) algorithm \( A \) with an approximation ratio of \( (\frac{1}{2} + \Theta(1)) \) must make \( \Omega(\frac{n}{\log n}) \) queries to function \( f(\cdot) \).

Moreover, we reveal a general relationship in query complexity between deterministic and randomized algorithms with good approximation guarantees, which enables us to extend the lower bound to the randomized setting. Note that this conclusion relies heavily on one precondition on the input instance, which we define it as a scale free property in Section 3.2.2.

**Theorem 3.** For any problem \( \mathcal{P} \), suppose that there exists a scale-free input instance set \( \mathcal{I}_s \), on which the worst case time complexity of any \( \alpha \)-approximate deterministic algorithm is at least \( T(\mathcal{P}) \). Then for any \( (\alpha + \delta) \)-approximate randomized algorithm \( A \) of problem \( \mathcal{P} \), its time complexity is in the order of \( \Omega(T(\mathcal{P})) \), where \( \delta = \Theta(1) \) is an arbitrary positive number such that \( \alpha + \delta \leq 1 \).

On the other hand, based on the proposed ADT algorithm and another idea we develop called the Backtracking Threshold (BT) algorithm, we are able to obtain an algorithm for the \( p \)-system+d knapsack constraints, which achieves improvements both in time complexity and approximation ratio compared with current state-of-the-art-result—an \( (\frac{1}{p+2d+1} - \varepsilon) \)-approximate algorithm using \( O(\frac{n}{\varepsilon} \log^2 \frac{n}{\varepsilon}) \) queries \cite{1}. 

2
Theorem 4. There is an \((\frac{1}{p+\frac{d}{2}} - \varepsilon)\)-approximation algorithm for maximizing a non-negative monotone submodular function subject to a \(p\)-system and \(d\) knapsack constraints, which performs \(O(\frac{2}{\varepsilon} \log \frac{2}{\varepsilon} \max\{\log \frac{1}{\varepsilon}, \log \log n\})\) oracle queries.

Finally we show that it is possible to achieve the almost optimal approximation \((1 - \frac{2}{\varepsilon} - \varepsilon)\) making \(\tilde{O}(nk)\) value oracle queries.

Theorem 5. There exists an algorithm that for any non-negative monotone submodular function \(f(\cdot) : 2^E \rightarrow \mathbb{R}^+\) with curvature \(\kappa\) and a matroid \(M = (\mathcal{N}, \mathcal{I})\) of rank \(k \in [0,1]\), \(A'\) could compute a solution \(S \in M\) such that \(f(S) \geq (1 - \frac{2}{\varepsilon}) \max\{f(T) : T \in M\}\) by performing \(\tilde{O}(nk)\) value oracle queries.

Additional Results. We believe that our techniques have far more applications or consequences than the aforementioned cases. Here we list two problems as examples.

Corollary 6. There exists an \((p(\frac{p}{2} + \frac{d}{2} + 1) - \varepsilon)\)-approximation algorithm for maximizing a non-negative (non-monotone) submodular function subject to the intersection of \(p\)-system and \(d\) knapsack constraints, which performs \(O(nr\max\{\log \frac{1}{\varepsilon}, \log \log n\})\) value oracle queries and independence oracle queries, where \(r \leq n\) denotes the size of the largest feasible solution.

Corollary 6 is achieved by our ADT algorithm, which improved the current fastest algorithm for this problem. Please refer to our technique report [11].

Corollary 7. There is an \((0.377 - \Omega(\varepsilon))\)-approximation algorithm for maximizing monotone submodular function under knapsack constraint, whose time complexity is in the order of \(O(n \cdot \{\varepsilon^{-1}, \log \log n\})\).

1.2 Techniques and Intitutions

In this subsection we briefly introduce our key techniques and ideas.

Adaptively decreasing threshold: This is the main technique that enables us to obtain faster algorithms, in which we can adaptively adjust the threshold to get a constant approximation of \(OPT\). To understand the motivation and design philosophy behind our Adaptive Threshold Method, we start with the following game with the goal of guessing a number.

Two player number guessing game. The first player called Alice, privately chooses a real number \(x_A \in [1, \bar{x}]\). The goal facing the second player Bob is to win the game by (approximately) discovering the secret number \(x_A\), i.e., identify a real number \(x_B\) such that

\[ x_B \in N_\varepsilon(x_A) = \left\{ x \in \mathbb{R}^+ \mid \frac{1}{1 + \varepsilon} \leq \frac{x}{x_A} \leq 1 + \varepsilon \right\} \]

while using as few queries as possible. A query is simply a real number \(x_B^{(i)}\) and the answer of Alice is \(1_{\{x_B^{(i)} \in N_\varepsilon(x_A)\}}(\delta \leq \varepsilon)\).

We are interested in the minimum number of queries that Bob should make to win the game. First we show that \(\Omega(\frac{\log \bar{x}}{\log (1 + \varepsilon)})\) number of queries are necessary for Bob to win the game, which is essentially in the order of \(\Omega(\frac{\log \bar{x}}{\varepsilon})\) when \(\varepsilon \to 0\). Suppose that Bob makes queries \(\{x^{(i)}\}(1 \leq i \leq q_B),\)
where WLOG we assume that \( x^{(i)} \leq x^{(i+1)} \), \( x^{(0)} = x \) and \( x^{(q_B+1)} = \bar{x} \). Note that if \( q_B \leq \frac{\log \bar{x}}{4 \log(1+\varepsilon)} - 2 \), then we can obtain the following conclusion utilizing the AM-GM inequality [19].

\[
\max_i \frac{x^{(i+1)}}{x^{(i)}} > \frac{1}{q_B + 1} \sum_{i=0}^{q_B} \frac{x^{(i+1)}}{x^{(i)}} \geq \bar{x}^{q_{B+1}} \geq (1 + \varepsilon)^4
\]

Let \( i^* = \arg \max_i (x^{(i+1)}/x^{(i)}) \). Now consider the case when Alice’s strategy is to answer “No” until it forces Bob to obtain an \( \varepsilon \)-accurate solution successfully. Based on (11), we know that

\[
[x^{(i^*)}, x^{(i^*+1)}] \cap \left\{ ((1 + \varepsilon)x^*, +\infty) \cup \left( 0, \frac{x^*}{1+\varepsilon} \right) \right\} \neq \emptyset
\]

which implies that Alice is always able to let Bob lose the game. On the other hand, we claim that the aforementioned lower bound is actually tight because Bob could adopt the following strategy—Guessing numbers in the set

\[
\left\{ x \mid x = x(1 + \varepsilon)^i \, , \, i \in \mathbb{Z}^+ \right\} \cap [\bar{x}, x]
\]

It is not hard to see that Bob can win the game by making \( O\left( \frac{\log \bar{x}}{\log(1+\varepsilon)} \right) \) queries. When \( \varepsilon \) is a constant, we have shown a \( \Omega(\log \bar{x}) \) lower bound on the number of queries. Are there any possible methods to beat the barrier of \( \Omega(\log \bar{x}) \)? A promising solution is to allow Bob make a two-dimensional query—A query is a pair of real numbers \( (x^{(i)}, \delta_i) \) and the corresponding answer of Alice is \( \mathbb{1}_{\{x^{(i)} \in N_{\delta_i}(x_A)\}} \).

Intuitively Bob could win the game in a faster way since the radius of his query is not necessary to be fixed. Actually we can show that Bob cannot obtain a constant approximation of \( x_A \) using \( o(\log \log \bar{x}) \) number of queries. Here we prove this lower bound and in Section 3.1 we will see how this guessing number game is related to our problem and how to achieve this new lower bound.

Suppose that Bob makes a query sequence \( \{(x^{(1)}, \delta_1), (x^{(2)}, \delta_2), \ldots, (x^{(q_B)}, \delta_{q_B})\} \). For ease of presentation, we color the interval \( [x^{(i)}/(1+\delta_i), (1 + \delta_i)x^{(i)}] \) by red if Alice’s answer to the \( i \)-th query is “No”, and we define the length of interval \( (a, b) \) as \( \frac{b - a}{\alpha} \). We use \( \mathcal{I}_i \subseteq [1, \bar{x}] \) to denote the longest uncolored interval after the \( i \)-th query, then based on the trivial observation that \( |\mathcal{I}_{i+1} \cap \left\{ x^{(i)}/(1+\delta_i), (1 + \delta_i)x^{(i)} \right\} | \leq 2 \), and again using the AM-GM inequality, we know that the length of \( \mathcal{I}_i \) decreases by at most the following speed,

\[
|\mathcal{I}_i| \geq |\mathcal{I}_{i-1}|^{1/3}
\]

It hence follows that

\[
|\mathcal{I}_{Q_B}| \geq \bar{x}^{\frac{1}{3q_B}}
\]

According to the similar argument above, we know that \( |\mathcal{I}_{Q_B}| \leq (1+c)^4 \), where constant \( c \) represents that approximation of Bob’s answer. Therefore we know that \( q_B = \Omega(\log \log \bar{x}) \).

In Section 3.1 we will proceed to formalize the intuition obtained from the second type of game into our ADT algorithm.
Backtracking Threshold (BT) algorithm. The ADT algorithm above could also lead to complexity reduction in $p$-system and $d$-knapsack constrained maximization problem. As for the improvements in approximation performance, we propose the Backtracking Threshold (BT) algorithm. The idea is simple but effective, when some knapsack constraints are violated, we recursively find a set with guaranteed function value, while the down-closed property of $p$-system and knapsack constraint ensures that the solution is feasible.

Rounding the marginal gains. For maximizing a monotone submodular function with bounded curvature under the matroid constraint, we first utilize the Lagrangian relaxation to enable us to exploit the fact that, the maximum weight independent set in a matroid polytope could be computed by greedy algorithm. Next by taking advantage of the fact that the cover constraint is soft, we round the marginal gains to obtain a simpler input instance, which ensures that we can find the optimal solution to the rounded problem fast, while losing a factor of $1 - \Omega(\varepsilon)$ in the objective function.

1.3 Additional Related Work

There is a large body of literature on submodular maximization, thus we mention only a few which is most relevant to our work. Besides the aforementioned results, there are also some other well-known results towards more practical algorithm design. For the simple cardinality constraint, there is also a stochastic greedy algorithm which uses $O(n \log \frac{1}{\varepsilon})$ value oracle queries while achieving $(1 - \frac{1}{e} - \varepsilon)$-approximation \[10\]. As for the general matroid constraint, Badanidiyuru and Vondrak \[1\] also proposed an accelerated continuous greedy algorithm which uses $O(\frac{n^2}{\varepsilon^2} \log \frac{2}{\varepsilon})$ value oracle queries and $O(\frac{n}{\varepsilon^2} \log \frac{n}{\varepsilon} + \frac{1}{\varepsilon} r^2)$ matroid independence queries. Later Buchbinder et al. \[2\] improve this result while exhibit an tradeoff between the value oracle query and independence query. They also give a $(\frac{1}{e} - \varepsilon)$-approximation algorithm for cardinality-constrained non-monotone submodular maximizing problem, which requires $O(\frac{\alpha}{\varepsilon} \log \frac{1}{\varepsilon})$ function value query.

2 Preliminaries

Notations. We use $[n]$ to denote the set \{1, 2, \ldots, n\} and $\mathbb{1}_A$ be the indicator variable of event $A$, i.e., $\mathbb{1}_A = 1$ if $A$ is true and 0 otherwise.

Submodular function. A set function $f : 2^E \rightarrow \mathbb{R}$ is called submodular iff $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ ($\forall A, B \subseteq E$), which is equivalent to another characterization which is based on diminishing returns: $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$ ($\forall A \subseteq B \subseteq E$). It is called monotone iff $f(A) \leq f(B)$ ($\forall A \subseteq B \subseteq E$) and normalized iff $f(\emptyset) = 0$.

Knapsack constraint. Knapsack constraint $K \subseteq 2^E$ implies a collection of sets defined as $K = \{ S \subseteq E | \sum_{e \in S} c(e) \leq 1 \}$, in which $c(e)$ denote the weight of element $e$ and without loss of generality we assume the capacity is 1.

Matroid constraint. For a set $E$ of elements and $I \subseteq 2^E$, we say $\mathcal{M}(E, I)$ is a matroid, if for all $S \in I$ and $T \subseteq S, T \in I$. And for all $S, T \in I$ such that $|S| < |T|$ there is an element $e \in T \setminus S$ such that $S \cup \{e\} \in I$. $S$ is called an independent set of $\mathcal{M}$ if $S \in I$. If there are no independent set of size larger than $S$, then $S$ is a base of $\mathcal{M}$.
Multilinear Extension. Let $F : [0, 1]^n \to \mathbb{R}_+$ denote the multilinear extension of $f$:

$$F(x) = \mathbb{E}[f(R(x))] = \sum_{S \subseteq V} f(S) \prod_{i \in S} x_i \prod_{i \in V \setminus S} (1 - x_i),$$

where $R(x)$ denotes a random subset of $E$, and each $i \in E$ is selected independently at random with probability $x_i$. Let $\nabla F$ denote the gradient of $F$ and $\frac{\partial F}{\partial x_i}$ represent the $i$-th coordinate of the gradient of $F$.

3 Faster Algorithm for Cardinality Constraint

In this section, we consider the most typical of constraint—cardinality constraint. In the following, we first introduce a deterministic $(1 - \frac{1}{e} - \varepsilon)$-approximate algorithm in Section 3.1 whose time complexity is almost linear in $n$. After that, we prove the evaluation oracle lower bound results in Section 3.2.

3.1 Main Algorithm and Analysis

In our ADT algorithm, we design a new algorithm paradigm which approximates the optimal value in a faster way. In some sense, the main idea of our algorithm is to “train” our estimation in a self-adaptive fashion—The “training” procedure of our ADT consists of $\ell$ iterations, in each iteration we adaptively adjust our lower and upper estimations of $OPT$ utilizing the set we obtained from last iteration, and finally we are able to obtain a constant approximation of $OPT$ after the first for
Algorithm 1: Adaptive Decreasing Threshold (ADT) Algorithm

1 Input: Submodular function \( f(\cdot) : 2^E \to \mathbb{R}^+ \), cardinality upper bound \( k \).
2 Output: Set \( U \subseteq E \) such that \( |U| \leq k \).
3 Initialization: \( \bar{\omega}_1 \leftarrow \max_{e \in E} f(e) \), \( \bar{\omega}_1 \leftarrow k \cdot \omega_1 \), \( U \leftarrow \emptyset \), \( \ell \leftarrow \log \left( \frac{\log k}{\log(1 + c)} \right) \).

4 for \( i = 1 : \ell \) do
5 \( \alpha_i = \exp(\log k \cdot e^{-i}) - 1 \)
6 \( \theta = \bar{\omega}_1 \)
7 while \( \theta \leq \bar{\omega}_1 \) do
8 \( S_{\theta}^{(i)} \leftarrow \emptyset \)
9 for \( e \in E \) do
10 if \( f(S_{\theta}^{(i)} + e) - f(S_{\theta}^{(i)}) \geq \theta / 2k \) and \( |S_{\theta}^{(i)}| \leq k \) then
11 \( S_{\theta}^{(i)} \leftarrow S_{\theta}^{(i)} + e \)
12 \( \theta \leftarrow \theta(1 + \alpha_i) \)
13 \( \omega_{i+1} \leftarrow \max_{\theta} f(S_{\theta}^{(i)}) \)
14 \( \bar{\omega}_{i+1} \leftarrow \omega_{i+1}(1 + \alpha_i) \)
15 \( \tau \leftarrow \frac{(1 + c) \omega_{q}}{k} \)
16 while \( \tau \geq \frac{\bar{\omega}_{q}}{k} \) do
17 for \( e \in E \) do
18 if \( f(U + e) - f(U) \geq \tau \) and \( |U| \leq k \) then
19 \( U \leftarrow U + e \)
20 \( \tau \leftarrow \tau - \frac{\bar{\omega}_{q}}{k} \)
21 return \( U \)

3.1.1 Performance Analysis of ADT

In this subsection we derive the performance guarantee of Algorithm 1. We first make the following vital observation.

Lemma 8.

\[ \bar{w}_i \leq f(O^*) \leq \bar{w}_i, \forall i \in [\ell] \]  
(3)

Proof: When \( i = 1 \), inequality (3) is equivalent to

\[ \max_{e} f(e) \leq f(O^*) \leq \max_{e} f(e) \]

this follows from the submodularity of \( f(\cdot) \) and the fact that \( \arg\max_{e} f(e) \) is a feasible solution. Note that for \( \bar{w}_q = \max_{\theta} f(S_{\theta}^{(q-1)}) \) \( (\forall i \geq 2) \), where \( |S_{\theta}^{(i-1)}| \leq k \), the LHS of (3) follows from the definition of \( O^* \). In the following we prove the RHS by induction.

Induction Step. Suppose that \( f(O^*) \leq \bar{w}_i \) holds for \( i = q \). We consider the iteration when \( i = q \), in which the value of \( \theta \) takes values in the set

\[ \Theta_q = \left\{ \bar{w}_q, \bar{w}_q(1 + \alpha_q), \ldots, \bar{w}_q \left( \frac{\log \bar{w}_1 / \bar{w}_q}{\log(1 + c)} \right) \right\} \]
combined with the induction assumption, we know that there must exists some \( \theta^* \in \Theta_q \) such that

\[
\theta^* \leq f(O^*) \leq (1 + \alpha_q)\theta^*.
\]  

(4)

In the following, we lower bound the value of \( f(S_{\theta^*}^{(q)}) \). According to the threshold accepting rule, we know that if \( |S_{\theta^*}^{(q)}| = k \), then \( f(S^{(q)}) \) must be no less than its size multiply the threshold:

\[
f(S_{\theta^*}^{(q)}) \geq |S_{\theta^*}^{(q)}| \cdot \frac{\theta^*}{2k} = \frac{\theta^*}{2} \geq \frac{f(O^*)}{2(1 + \alpha_q)}
\]

(5)

where the last inequality follows from (4). Otherwise if \( |S_{\theta^*}^{(q)}| < k \), which implies that the marginal gains of elements in \( O^* \) are small, based this we can obtain the following inequality:

\[
f(O^*) - f(S_{\theta^*}^{(q)}) \leq f(O^* \cup S_{\theta^*}^{(q)}) - f(S_{\theta^*}^{(q)})
\]

\[
\leq \sum_{e \in O^*} \left[ f(S_{\theta^*}^{(q)} + e) - f(S_{\theta^*}^{(q)}) \right]
\]

\[
\leq \sum_{e \in O^*} \frac{\theta^*}{2k} = \frac{\theta^*}{2} \leq \frac{f(O^*)}{2}
\]

(6)

which is equivalent to

\[
f(S_{\theta^*}^{(q)}) \geq \frac{f(O^*)}{2}
\]

Combining (5) and (6), we obtain that

\[
f(S_{\theta^*}^{(q)}) \geq \min \left\{ \frac{f(O^*)}{2}, \frac{f(O^*)}{2(1 + \alpha_q)} \right\} = \frac{f(O^*)}{2(1 + \alpha_q)}
\]

(7)

based on which we have

\[
\tilde{w}_{q+1} = (1 + \alpha_q)\tilde{w}_{q+1} = (1 + \alpha_q) \max_{\theta} f(S_{\theta}^{(q)})
\]

\[
\geq (1 + \alpha_q) f(S_{\theta^*}^{(q)}) \geq f(O^*)
\]

where the equalities are based on the definition of \( w_i \) and \( \tilde{w}_i \), the last inequality comes from (7). Hence we have shown that the lemma holds for \( i = q + 1 \). The proof is complete.

As a consequence, we have the following corollary.

Corollary 9.

\[
\frac{f(O^*)}{1 + c} \leq \tilde{w}_r \leq f(O^*)
\]

Proof: The corollary follows from Lemma 8 and the following equation:

\[
\tilde{w}_r = (1 + \alpha_r)\tilde{w}_r = \exp \left( \log k \cdot e^{-\log \left( \frac{\log k}{\log(1 + c)} \right)} \right) \tilde{w}_r = (1 + c)\tilde{w}_r
\]

\[
\square
\]

8
We first explain our motivation behind the threshold decreasing procedure. Recall the analysis of standard greedy algorithm, to achieve an \((1 - \frac{1}{e})\)-approximate solution, it suffices to select an element whose marginal gain is no less than the average deficit \(\frac{f(O^*) - f(U)}{k}\). Note that WLOG we can assume that

\[
\frac{f(O^*) - f(U)}{k} \in \left[ \frac{f(O^*)}{ek}, \frac{f(O^*)}{k} \right]
\]

(8)

Observe that inequality (8) bound the term \(f(O^*) - f(U)\) in a small interval, which motivates us to guess the average deficit of each selection utilizing the following Geometric sequence

\[
G = \left\{ \frac{f(O^*)}{k}, \frac{f(O^*)}{k}(1 - \varepsilon), \frac{f(O^*)}{k}(1 - \varepsilon)^2, \ldots, \frac{f(O^*)}{ek} \right\}
\]

Before proving the main results, we first give the following definition.

**Definition 10.** (Effective Threshold) We call \(\tau \in G\) is effective if there exists at least one element that is added into \(U\) when using \(\tau\) as a threshold, otherwise call it ineffective. Ineffective element appears.

Now we proceed to formalize our intuition.

**Corollary 11.** Algorithm 4 is a \((1 - \frac{1}{e} - \varepsilon)\)-approximation algorithm.

**Proof:** Denote \(U = \{u_1, u_2, \ldots, u_r\}\), where \(u_i\) represents the \(i\)-th element added into \(U\) and let \(U^{(i)} = \{u_1, u_2, \ldots, u_i\}\) (\(U^{(0)} = \emptyset\)). We first claim that

\[
f(u_1) \geq (1 - \varepsilon)\frac{f(O^*)}{k}
\]

(9)

We prove (9) by contradiction, suppose that (9) is false. Note that \(\tau\) is initialized to \(\tau_0 = \frac{(1+\varepsilon)w_i}{k}\), it can be seen from Corollary 9 that \(\tau_0 \geq \frac{f(O^*)}{k}\). Hence there must exists one iteration in which \(\tau \in \left[ (1 - \varepsilon)\frac{f(O^*)}{k}, \frac{f(O^*)}{k} \right]\). According to the assumption, we know that \(U = \emptyset\) still holds after this iteration. However, \(\text{argmax}_e f(e)\) must satisfy the threshold requirement and \(U\) cannot be empty after this iteration, which leads to a contradiction and therefore (9) is true.

Let \(\tau_{\text{min}} = \min\{\tau \in G : \tau \geq \frac{f(O^*)}{ek}\}\). We claim that WLOG we can assume that \(\tau_{\text{min}}\) is ineffective. Because otherwise, consider the iteration when the threshold \(\tau = \tau_{\text{min}}\), after this iteration ends we obtain set \(U' \subseteq U\) and we have

\[
f(U' + e) - f(U') < \tau_{\text{min}}, \quad \forall e \in E \setminus U'
\]

(10)

Note that the assumption that \(\tau_{\text{min}}\) is effective is necessary to let (10) hold. Based on (10), we could further know that

\[
f(O^*) - f(U') \leq \sum_{e \in O^*} [f(U' + e) - f(U')]
\]

\[
\leq |O^*|\tau_{\text{min}} \leq \frac{1}{e(1 - \varepsilon)}f(O^*)
\]

9
where the last inequality is based on the definition of $\tau_{\min}$. Thus
\[
f(U) \geq f(U') \geq (1 - \frac{1}{e(1 - \varepsilon)})f(O^*) \geq (1 - \frac{1}{e} - \frac{3}{e})f(O^*) = (1 - \frac{1}{e} - \Omega(\varepsilon))OPT
\]
where the last inequality is based on the elementary inequality $\frac{1}{1 - \varepsilon} \leq 1 + 3\varepsilon$ for $\varepsilon \leq 1 - \frac{1}{e} < \frac{2}{3}$. Therefore we could assume that $\tau_{\min}$ is ineffective, which implies that there are exactly $k$ elements in $U$ when our algorithm ends.

Next we prove the following recursion inequality, which plays an crucial role in the analysis of standard greedy:
\[
f(U^{(i+1)}) - f(U^{(i)}) \geq (1 - \varepsilon)\frac{f(O^*) - f(U^{(i)})}{k}
\]
(11)

It states that the increment at each step is at least the average deficit, which is the start-point of our aforementioned intuition. Let $\tau^{(i)}$ denote the corresponding threshold when $u_i$ is selected. Note that we have proven the base case when $i = 0$, it is essentially inequality (9). For $i \geq 1$, since $e_i^* = \arg\max \varepsilon [f(U^{(i)} + e) - f(U^{(i)})]$ is not selected, then
\[
\frac{f(O^*) - f(U^{(i)})}{k} \leq f(U^{(i)} + e_i^*) - f(U^{(i)}) \leq \frac{\tau^{(i)}}{1 - \varepsilon} \leq \frac{f(U^{(i+1)}) - f(U^{(i)})}{1 - \varepsilon}
\]
which implies that (11) is true. Similar as the analysis standard greedy, we know that $f(U) \geq (1 - \frac{1}{e} - \Omega(\varepsilon))OPT$
and the corollary holds.

\[\Box\]

#### 3.1.2 Time Complexity Analysis of ADTA

In this subsection we analyze the time complexity of our ADT algorithm. We first make the following observation.

**Observation 12.** The time complexity of line (4) – (14) is $O(n \log \log k)$.

**Proof:** Note that the time complexity of the $i$-th iteration of the first for loop in line is
\[
O\left(n \cdot \frac{\log \lambda_i}{\log(1 + \alpha_i)}\right) = O\left(n \cdot \frac{\log(1 + \alpha_{i-1})}{\log(1 + \alpha_i)}\right)
\]
then the total running time is in the order of
\[
O\left(n \cdot \sum_{i=1}^{\ell} \frac{\log(1 + \alpha_{i-1})}{\log(1 + \alpha_i)}\right) = O(\ell \cdot n) = O(n \log \log n)
\]
where the equality can be easily verified by the definition of $\alpha_i$ and $\ell$. \[\Box\]

Notice that for the second while loop starting from line 16, there are $O(\frac{1}{\varepsilon})$ loops and the time complexity of each loop is $O(n)$. It follows that the time complexity of this while loop is $O(n\varepsilon^{-1})$. Hence we obtain the following corollary.

**Corollary 13.** The time complexity of our ADT algorithm is $O(n \cdot \max\{\varepsilon^{-1}, \log \log n\})$.

Then Corollary 13 and Corollary 11 complete the proof of Theorem 11.
3.2 Lower Bounds on Query Complexity

3.2.1 Deterministic Lower bound

The main idea of our proof is to construct some hard input instances, on which the algorithm cannot achieve good enough performance guarantee. WLOG we assume that $n$ is even.

Consider the following sequence $\{a_i\}_{i \in [n]}
\[ a_i = 1 \quad (i \in \left[\frac{n}{2}\right]), \quad a_i = M \quad (i > \frac{n}{2}). \]
where $M$ is a large positive integer.

Now consider all the $N = n!$ permutations of sequence $\{a_j\}$. Let $\sigma^{(i)}$ be the $i$-th ($1 \leq i \leq N$) permutation and let $\sigma^{(i)}(j)$ be the value of the $j$-th ($1 \leq j \leq n$) element in permutation $\sigma^{(i)}$. We define a modular function $f_i(\cdot) : 2^E \rightarrow \mathbb{Z}^+$ as:

\[ f_i(S) = \sum_{j \in S} \sigma^{(i)}(j) \]

Note that for $\forall S \subseteq E$, its corresponding function is in the form of $i + j \cdot M$, where $i, j \in \left[\frac{n}{2}\right]$, thus $f(\cdot)$ takes value in a set of at most $\frac{n^2}{4}$ distinct elements, i.e. $|U| \leq \frac{n^2}{4}$ where $U$ denotes the codomain of function $f(\cdot)$.

Now consider the following adversary argument. Suppose that the adversary keeps a list $L$ of functions $f_i(\cdot)$ ($1 \leq i \leq N$) and update the list in the following manner. Initially all the functions $f_i(\cdot)$ ($1 \leq i \leq N$) appears in list $L$. When the algorithm makes query $S_i$, the adversary partitions the functions on list $L$ according to their function values, i.e., functions with the same value on set $S_i$ will be partitioned into the same sub-list. The adversary always keeps the longest sub-list and returns the associated function value as the answer to query $S_i$.

Let $N^{(i)}$ be the number of functions after then $i$-th query, then the following inequality follows from our previous argument.

\[ N^{(i+1)} \geq \frac{N^{(i+1)}}{n^2} \]

Since there are at most $\Omega(n^2)$ different function values and each time the adversary maintains the longest sub-list. Hence, the number of functions remaining on the list after $Q$ queries is at least

\[ N^{(Q)} \geq \frac{N}{n^{2Q}} = \frac{n!}{n^{2Q}} \]

Suppose that the algorithm chooses set $R \subseteq E$ as its output, WLOG we assume that $|R| = k = \frac{n}{2}$, otherwise we choose any set $R'$ such that $R \subseteq R'$ and $|R| = k = \frac{n}{2}$, our analysis can still be applied to $R'$. To achieve an $c$-approximation, $R$ must satisfy that $f(R) \geq c \cdot OPT = \frac{cMn}{2}$. When $M$ is large enough, we have

\[ \left| \left\{ f_i(\cdot) \mid f_i(R) \geq \frac{cMn}{2} \right\} \right| = \sum_{r=\frac{n}{4}}^{\frac{n}{2}} \left( \frac{n}{2} \right) \cdot \left( \frac{n}{2} - r \right) \cdot (n/2)! \cdot (n/2)! \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]

\[ \geq N^{(Q)} \geq \frac{n!}{n^{2Q}} \]
which is equivalent to
\[ n^2 Q \geq \frac{\binom{n}{n/2}}{\sum_{r=\frac{n}{2}}^{n/2} \binom{n/2}{r}} \geq \frac{\binom{n}{n/2}}{n \cdot \binom{n/2}{r}^2} \left( c = \frac{1}{2} + \Theta(1) \right) \quad (14) \]
which implies that \( Q \geq \frac{n}{\log n} \). Otherwise the algorithm will fail to distinguish the functions in the list and thus cannot provide a \( c \)-approximation solution.

### 3.2.2 Randomized Lower bound

In this subsection we establish a very general result to lower bound the time complexity of randomized algorithms with given approximation ratio requirement. Recall that a common idea to prove time complexity lower bound for deterministic algorithms is to figure out a set of “bad” instances, and then argue via various techniques that for any deterministic algorithm, there always exists a specific instance to prevent the algorithm from finding the desired answer in a fast way. Now our result indicates that, if the aforementioned “bad” set satisfies the following property we called “scale-free”, then the complexity lower bounds for deterministic algorithms also holds for randomized algorithms whose approximation ratio is better than the that of deterministic algorithm by an arbitrary small \( \Theta(1) \) constant.

**Definition 14.** (Scale-free Instance Set) A finite instance set \( \mathcal{I} \) is called scale-free, if there exists a measure \( \mu \) defined on \( \mathcal{I} \) such that
\[
\frac{\max_{I \in \mathcal{I}} T(A, I)}{\max_{I \in \mathcal{I}} T(A, I)} = \Theta(1)
\]
holds for any deterministic algorithm \( A \) and any set \( \mathcal{I}' \subseteq \mathcal{I} \) such that \( \frac{\mu(\mathcal{I}')}{\mu(\mathcal{I})} = \Theta(1) \), where \( T(A, I) \) denotes the time that running time of algorithm \( A \) on instance \( I \).

With this definition, now we are able to establish the following lemma characterizing an interesting relation between the deterministic complexity and randomized complexity.

**Theorem 15.** For any problem \( \mathcal{P} \), suppose that there exists a scale-free input instance set \( \mathcal{I}_s \), on which the worst case time complexity of any \( \alpha \)-approximate deterministic algorithm is at least \( T(\mathcal{P}) \). Then any \((\alpha + \delta)\)-approximate randomized algorithm \( A \) of problem \( \mathcal{P} \), its time complexity is in the order of \( \Omega(T(\mathcal{P})) \), where \( \delta = \Theta(1) \) is an arbitrary positive number such that \( \alpha + \delta \leq 1 \).

**Proof:** Let \( \mathcal{P} \) be a problem with a scale-free input instance set \( \mathcal{I}_s \) and a finite set \( \mathcal{A} \) of deterministic approximation algorithms. Let \( \mathcal{R}^{(c)} \) represents the set of randomized algorithms whose expected approximation ratio is no less than \( c \) (\( c \in [0, 1] \)), then \( \mathcal{R}(\mathcal{P}, c) \), the Randomized Complexity of problem \( \mathcal{P} \) over \( \mathcal{R}^{(c)} \), could be lower bounded as:
\[
\mathcal{R}(\mathcal{P}, c) = \min_{R \in \mathcal{R}^{(c)}} \max_{I \in \mathcal{I}} \mathbb{E}_{A \sim \lambda_R} T(A, I) 
\geq \min_{R \in \mathcal{R}^{(c)}} \max_{I \in \mathcal{I}_s} \mathbb{E}_{A \sim \lambda_R} T(A, I) \quad (15)
\]
where \( \mathcal{I} \) denote the entire input set and the inequality holds because \( \mathcal{I}_s \subseteq \mathcal{I} \).
Note that every randomized algorithm \( R \) could be represented as a distribution \( \lambda_R \) over the set of deterministic algorithms. Let \( C_R = \max_I E_{A \sim \lambda_R} T(A, I) \) and \( S(A, I) \) be the solution computed by algorithm \( A \) on instance \( I \). According to the definition of approximation ratio, we know that the expected (objective function) value, \( E[f(S(A, I))] \), of the approximate solution \( S(A, I) \), will be no less than a factor \( (\alpha + \delta) \) times the optimal solution value \( \text{OPT}_I \):

\[
E_{A \sim \lambda_R}[f(S(A, I))] \geq (\alpha + \delta) \cdot \text{OPT}_I, \quad \forall I \in I
\]  

(17)

Notice that

\[
E_{A \sim \lambda_R}[f(S(A, I))] \leq \mathbb{P}(f(S(A, I)) \leq \alpha \cdot \text{OPT}_I) \cdot (\alpha \cdot \text{OPT}_I) + \left[ 1 - \mathbb{P}(f(S(A, I)) \leq \alpha \cdot \text{OPT}_I) \right] \cdot \text{OPT}_I
\]

Combined with (17), it follows that

\[
\mathbb{P}_{A \sim \lambda_R}(f(S(A, I)) \leq \alpha \cdot \text{OPT}_I) \leq 1 - \frac{\delta}{1 - \alpha}, \quad \forall I \in I
\]  

(18)

We could further know that

\[
E_{A \sim \lambda_R} \left[ 1\{f(S(A, I)) \leq \alpha \cdot \text{OPT}_I\} \cup \{T(A, I) > \frac{2(1 - \alpha)}{\delta} C_R\} \right]
\]

\[
\leq E_{A \sim \lambda_R} \left[ 1\{f(S(A, I)) \leq \alpha \cdot \text{OPT}_I\} \right] + E_{A \sim \lambda_R} \left[ 1\{T(A, I) > \frac{2(1 - \alpha)}{\delta} C_R\} \right]
\]

\[
\leq \left( 1 - \frac{\delta}{1 - \alpha} \right) + \frac{\delta}{2(1 - \alpha)} = 1 - \frac{\delta}{2(1 - \alpha)}
\]  

(19)

Utilizing union bound we know the first inequality is true; The second inequality is based on (18) and the following consequence from Markov’s inequality:

\[
E_{A \sim \lambda_R} \left[ 1\{T(A, I) > \frac{2(1 - \alpha)}{\delta} C_R\} \right] = \mathbb{P}_{A \sim \lambda_R}(T(A, I) > \frac{2(1 - \alpha)}{\delta} C_R)
\]

\[
\leq \frac{\delta \cdot E_{A \sim \lambda_R} T(A, I)}{2(1 - \alpha) C_R}
\]

\[
\leq \frac{\delta}{2(1 - \alpha)}
\]

where the last inequality is based on the definition of \( C_R \). Applying Yao’s Minimax Principle \[22\], we know that for \( \forall \mu \), there exists a deterministic algorithm \( A^*_\mu \) such that

\[
E_{I \sim \mu} \left[ 1\{f(S(A^*_\mu, I)) \leq \alpha \cdot \text{OPT}_I\} \cup \{T(A^*_\mu, I) > \frac{2(1 - \alpha)}{\delta} C_R\} \right] \leq 1 - \frac{\delta}{2(1 - \alpha)}
\]  

(20)

Now we define a new input instance set \( I^*_\mu \subseteq I_s \) as

\[
I^*_\mu = \left\{ I \in I_s \mid f(S(A^*_\mu, I)) > \alpha \cdot \text{OPT}_I, T(A^*_\mu, I) \leq \frac{2(1 - \alpha)}{\delta} C_R \right\}
\]  

(21)

Based on (20), it can be seen that under probability measure \( \mu \),

\[
\mathbb{P}_{I \sim \mu}(I \in I^*_\mu) > \frac{\delta}{2(1 - \alpha)} > \frac{\delta}{2} = \Theta(1)
\]  

(22)

13
Recall that for any deterministic algorithm $A$, its worst-case time complexity on $\mathcal{I}_s$ is at least $T(P)$, combined with the fact that $\mathcal{I}_s$ is a scale-free set and inequality (22), we know that

$$\max_{I \in \mathcal{I}_s} T(A^{\mu}_{\mathcal{I}_s}, I) \in \Omega(T(P))$$  \hspace{1cm} (23)

On the other hand, from the definition of $\mathcal{I}^*_\mu$, we know that every input instance in $\mathcal{I}_s$ satisfies that

$$T(A^{\mu}_{\mathcal{I}_s}, I) \leq \frac{2(1 - \alpha)}{\delta} C_R, \hspace{0.5cm} \forall I \in \mathcal{I}_s$$  \hspace{1cm} (24)

Combining (24) and (23), $C_R$ is then proven to be in the order of $T(P)$. Notice that the arguments above holds for arbitrary randomized algorithm in $R^{(\alpha+\delta)}$, it follows that

$$R(P, \alpha + \delta) \in \Omega(T(P))$$

\hspace{1cm} □

4 Algorithm for the $p$-system and $d$ knapsack constraints

We consider the following problem

$$\max_S f(S)$$  \hspace{1cm} (25)

subject to $S \in (\cap_{i=1}^d K_i) \cap \mathcal{I}_0$ where $\mathcal{I}_0$ denote the $p$-system constraint and

$$K_i = \{S \subseteq E | C_i(S) \leq 1\}$$

Here $C_i(\cdot) : 2^E \rightarrow \mathbb{R}^+$ denote the modular cost function in the $i$-th knapsack constraint, i.e., $C(S) = \sum_{e \in S} C_i(e)$. In the rest of this section, we first present the Backtracking Threshold (BT) Algorithm algorithm for problem (25) in Section 4.1. Then we discuss possible extensions on the knapsack constraint problem in Section 4.2.
4.1 \( p \)-system+\( d \)-knapsack problem

\[\textbf{Algorithm 2:} \text{Backtracking Threshold (BT) Algorithm for } p \text{-system+}\!d \text{ knapsack constraints}\]

1. \textbf{Input:} submodular function \( f(\cdot) : 2^E \rightarrow \mathbb{R}^+ \), knapsack function \( C_i(\cdot) : 2^E \rightarrow \mathbb{R}^+ (i \in [d]) \).
2. \textbf{Output:} Set \( S^* \in (\cap_{i=1}^d K_i) \cap \mathcal{I}_0 \).
3. Guess the value of \( |O^*_d| \)
4. \( M \leftarrow \max_e f(e) \)
5. \( \Delta \leftarrow M \)
6. while \( \Delta \geq \frac{\epsilon}{\alpha} \) do
7. \hspace{1em} for \( e \in E \) do
8. \hspace{2em} if \( \sum_{i=1}^d C_i(e) \leq \frac{1}{2} \), \( \sum_{i=1}^d C_i(e) \Delta \) and \( S + e \in \mathcal{I}_0 \) then
9. \hspace{3em} \( S \leftarrow S + e; \)
10. \hspace{2em} else
11. \hspace{3em} \( \hat{S} \leftarrow \{e, \arg\max_{e \in S} \sum_{i=1}^d C_i(e)\}; \)
12. \hspace{3em} for \( e \in S \setminus \hat{S} \) do
13. \hspace{4em} if \( C_i(S + e) \leq 1 (\forall i \in [d]) \) then
14. \hspace{5em} \( \hat{S} \leftarrow \hat{S} + e; \)
15. \hspace{1em} end
16. \( e^* \leftarrow \arg\max_{e \in E} f(e); \)
17. \( S^* \leftarrow \arg\max_{T \in \{S^*, \hat{S}, \{e^*\}\}} f(T) \)

\textbf{Remark:} Note that removing the big elements is necessary. Consider the following case: \( \{\frac{1}{2} + t\epsilon, \frac{1}{2} - \epsilon, \epsilon, \ldots, \epsilon\} \).

4.1.1 Performance Analysis of Backtracking Threshold (BT) Algorithm

Before presenting the analysis, we first give some basic definitions and notations that are used in our proof. An important ingredient in our algorithm is the distinction between elements by their costs.

\textbf{Definition 16.} (Big element) An element is called big if there exists some \( i \in [d] \) such that \( C_i(e) > 1/2 \). Otherwise it is called small element.

Here are some useful notations.

\textbf{Notation.} In the following, we use \( \tilde{e}_1 \) to represent the element which triggers the else iteration, i.e., it is not added into set \( S \) because of some knapsack constraints. Let \( \tilde{e}_2 \) denote the element with highest total cost in \( S \), i.e., \( \tilde{e}_2 = \arg\max_{e \in S} \sum_{i=1}^d C_i(e) \). WLOG we assume that \( S = \{e_1, e_2, \ldots, e_s\} \), where \( e_i \) is the \( i \)-th element added into \( S \). Since \( \hat{S} \) is a subset of \( S \), we denote \( \hat{S} \) as \( \hat{S} = \{e_{i_1}, e_{i_2}, \ldots, e_{i_{s'}}\} \), where \( i_t \in [s] \) for \( \forall t \leq s' \leq s \). Further we let \( S^{(i)} = \{e_1, e_2, \ldots, e_i\} (i \in [s]) \) and \( S^{(i)} \) is defined in a similar way.

Note that a simple but crucial consequence of Definition 16 is the following fact, which upper bounds the number of big elements in the \( O^* \).

\textbf{Fact 17.} There are at most \( d \) big elements in \( O^* \).
In fact if there are more than \( d \) big elements in \( O^* \), then there must exist some dimension \( i \), in which the cost of \( O^* \) is strictly larger than 1, thus the fact is true. In the following we use \( O^*_b \) to denote the collection of those big elements in \( O^* \).

In the rest of this section, we derive out the performance of RT algorithm in two cases, which divided according the existence of \( \tilde{e}_1 \).

**Case 1:** \( \tilde{e}_1 \) exists.

In this case, we first show the following fact that indicates the feasibility of several candidate solution set.

**Fact 18.** \( \tilde{S}, \{\tilde{e}_1, \tilde{e}_2\} \in (\cap_{i=1}^{d} K_i) \cap I_0 \)

**Proof:** On the one hand, for set \( \tilde{S} \), note that \( \tilde{S} \subseteq S \cup \{\tilde{e}_1\} \in I_0 \), by the down-closed property of \( I_0 \), we know that \( \{\tilde{e}_1, \tilde{e}_2\} \subseteq \tilde{S} \in I_0 \). On the other hand, notice that \( \tilde{S} \) is initialized to \( \{\tilde{e}_1, \tilde{e}_2\} \), in which both \( \tilde{e}_1 \) and \( \tilde{e}_2 \) are small elements, which implies that

\[
f_t(\{\tilde{e}_1, \tilde{e}_2\}) \leq \sum_{i=1}^{d} C_i(\{\tilde{e}_1, \tilde{e}_2\}) = 2 \sum_{j=1}^{2} C_i(\tilde{e}_j) \leq \frac{1}{2} + \frac{1}{2} = 1, \quad \forall t \in [d] \tag{26}
\]

from which we know that \( \{\tilde{e}_1, \tilde{e}_2\} \) must be a feasible solution and the initialization of set \( \tilde{S} \) is always valid, then \( \tilde{S} \subseteq \cap_{i=1}^{d} K_i \). \( \square \)

According to the density threshold rule, we know that

\[
f(S(j+1)) - f(S(j)) \geq \rho \sum_{e \in [d]} C_i(e_{j+1})
\]

based on which we could lower bound \( f(\tilde{S}) \) as

\[
f(\tilde{S}) = \sum_{j=1}^{s'} [f(\tilde{S}(j)) - f(\tilde{S}(j-1))] \\
\geq \sum_{j=1}^{s'} [f(S(i,j)) - f(S(i,j-1))] \\
\geq \rho \sum_{j=1}^{s'} \sum_{t=1}^{d} C_i(e_{ij}) \tag{27}
\]

We claim that \( S \setminus \tilde{S} \neq \emptyset \), otherwise set \( \tilde{S} \) will be equal to \( S \cup \{\tilde{e}_1\} \), this contradicts the fact that \( \tilde{S} \) is a feasible solution. Hence there exists some \( e_b \in S \setminus \tilde{S} \) such that \( \tilde{S} \cup \{e_b\} \) violates some knapsack constraint \( C_{ib}(\cdot) \). It follows that

\[
\sum_{i=1}^{d} \sum_{e \in S \cup \{e_b\}} C_i(e) \geq \sum_{e \in S \cup \{e_b\}} C_{ib}(e) > 1 \tag{28}
\]

Combining (27) and (28) and interchanging the order of summation, we know that

\[
f(\tilde{S}) \geq \rho \left( 1 - \sum_{i=1}^{d} C_i(e_{ib}) \right) \geq \rho \left( 1 - \sum_{i=1}^{d} C_i(\tilde{e}_2) \right) \tag{29}
\]
where the second inequality is based on the fact that $\tilde{e}_2 = \arg\max_{e \in S} \sum_{i=1}^d C_i(e)$. On the other hand, using similar argument of (27), we know that

$$ f(\tilde{S}) \geq f(\{\tilde{e}_1, \tilde{e}_2\}) \geq \rho \left( \sum_{j=1}^2 \sum_{i=1}^d C_i(\tilde{e}_j) \right) $$

Moreover, we have

$$ f(S) \geq \theta \left( \sum_{e \in S} \sum_{i=1}^d C_i(e) \right) \geq \theta \left( 1 - \sum_{i=1}^d C_i(\tilde{e}_1) \right) $$

(31)

Combining (29), (30) and (31), we know that

$$ \max \left\{ f(S), f(\tilde{S}) \right\} \geq \frac{f(S) + 2f(\tilde{S})}{3} $$

(32)

$$ \geq \frac{1}{3} \left[ \theta \left( 1 - \sum_{i=1}^d C_i(\tilde{e}_1) \right) + \rho \left( \sum_{j=1}^2 \sum_{i=1}^d C_i(\tilde{e}_j) \right) + \theta \left( 1 - \sum_{i=1}^d C_i(\tilde{e}_2) \right) \right] $$

(33)

Case 2: $\tilde{e}_1$ does not exist.

In this case we partition the optimal solution $O^*$ into three subsets: $O^* = O^*_1 \cup O^*_2 \cup O^*_b$, where

$$ O^*_1 = \{ e \in O^* \setminus O^*_b | f_S(e) \leq \theta \} $$

and $O^*_2 = O^* \setminus (O^*_b \cup O^*_1)$. Then we could lower bound $f(S)$ in the following manner,

$$ f(O^*) - f(S) \leq f(S \cup O^*) - f(S) \leq \left[ f(S \cup O^*_b) - f(S) \right] + \left[ f(S \cup O^*_1) - f(S) \right] + \left[ f(S \cup O^*_2) - f(S) \right] = R_b + R_1 + R_2 $$

(34)

For $R_b$, a direct consequence of submodularity is

$$ R_b \leq \sum_{e \in O^*_b} f(e) = |O^*_b| \max_e f(e) $$

(35)

As for $R_1$, we could upper bound it in the following manner:

$$ R_1 = \sum_{e \in O^*_1} \left[ f(S \cup \{e\}) - f(S) \right] \leq \theta \sum_{e \in O^*_1} \sum_{i=1}^d C_i(e) \leq \theta \left( d - \sum_{e \in O^*_b} \sum_{i=1}^d C_i(e) \right) \leq \theta \left( d - \frac{1}{2} |O^*_b| \right) $$

(36)
where (a) is based on the threshold accepting rule and the definition of $O^*_b$; Since $O^*$ is a feasible solution with respect to $K_i (\forall i \in [d])$, thus $\sum_{e \in O^*} \sum_{i=1}^{d} C_i(e) \leq 1$, and the total cost of each element in $O^*_b$ is at least $\frac{1}{\pi}$, thus (b) and (c) follows.

Finally, to upper bound $R_3$, the proof is indeed the same as the analysis of standard greedy on $p$-system constraint. We provide the proof for completeness.

Lemma 19. $R_2 \leq (p + \Omega(\varepsilon))f(S)$

Proof: Please refer to our technique report [11].

Combining Lemma 19 and (34)-(36), we can obtain that

$$\max \{ f(S), \max_{e \in E} f(e) \} \geq \frac{f(O^*) - \rho(d - \frac{1}{2}|O^*_b|)}{p + |O^*_b| + 1} - \Omega(\varepsilon)f(O^*) \tag{37}$$

Now it is time to put the two cases above together. According to (32) and (37), the quality of our final solution is at least

$$f(S^*) \geq \max \{ f(S), f(\tilde{S}), \max_{e \in E} f(e) \} \geq \min \left\{ \frac{2}{3} \theta, \frac{f(O^*) - \rho(d - \frac{1}{2}|O^*_b|)}{|O^*_b| + 1} \right\} - \Omega(\varepsilon)f(O^*) \tag{38}$$

Note that for

$$\theta^* = \frac{f(O^*)}{d + 2/3 + |O^*_b|/6 + \frac{2}{3p}} \in \Theta(1) f(O)$$

(38) attains its maximum value, and there exists one iteration in which $\theta \in [(1 - \Omega(\varepsilon)), \theta^*, \theta^*]$. Hence

$$f(S^*) \geq \left( \frac{1}{p + \frac{3}{2}d + \frac{|O^*_b|}{4} + 1} - \Omega(\varepsilon) \right)OPT \geq \left( \frac{1}{p + \frac{3}{4}d + 1} - \Omega(\varepsilon) \right)OPT$$

4.1.2 Time complexity

Please refer to our technique report [11] for details

4.2 Algorithm for knapsack constraint

As a direct consequence of the Backtracking Threshold(BT) algorithm, we could obtain an $\left( \frac{1}{1 + \frac{3}{4}d - \varepsilon} \right)$-approximation algorithm. In this subsection, we show that we could actually obtain better
approximation guarantee based our proposed techniques. We take \( d = 1 \) as an example.

**Algorithm 3: Algorithm for \( d \) knapsack constraints**

1. **Input:** submodular function \( f(\cdot) : 2^E \to \mathbb{R}^+ \), knapsack function \( C_i(\cdot) : 2^E \to \mathbb{R}^+ (i \in [d]) \).
2. **Output:** Set \( S^* \in \cap_{i=1}^d K_i \).
3. \( \lambda \leftarrow \epsilon \)-approximation of \( f(O^*) \)
4. \( e^* \leftarrow \arg\max_{e \in E} f(e) \);
5. \( S, \tilde{S} \leftarrow \emptyset \)
6. \( \tau = \frac{\lambda}{cd} \)
7. **while** \( \tau \leq \frac{\epsilon}{cd} \) **do**
8. **for** \( e \in E \) **do**
9. **if** \( f_S(e) \geq \tau \sum_{i=1}^d C_i(e) \) and \( \sum_{i=1}^d C_i(e) \leq \frac{1}{2} \) **then**
10. **if** \( C_i(S + e) \leq 1 \) (\( \forall i \in [d] \)) **then**
11. \( S \leftarrow S + e; \)
12. **else**
13. \( \tilde{S} \leftarrow \{e, \arg\max_{e \in S} \text{Cost}(e)\}; \)
14. **for** \( e \in S \setminus \tilde{S} \) **do**
15. **if** \( C_i(\tilde{S} + e) \leq 1 \) (\( \forall i \in [d] \)) **then**
16. \( \tilde{S} \leftarrow \tilde{S} + e; \)
17. **return** \( S^* \leftarrow \arg\max_{T \in \{S^*, S, \{e^*\}\}} f(T) \)
18. \( \tau \leftarrow (1 - \epsilon)\tau \)
19. **return** \( S^* \leftarrow \arg\max_{T \in \{S^*, S, \{e^*\}\}} f(T) \)

### 4.2.1 Analysis of the algorithm

Note that Algorithm 3 stops only when at least one of the following condition happens: (1) No elements could be added into the solution set because of low marginal gain. (2) At least one knapsack constraint is violated.

If (1) happens, then \( f(S) \) must be of high value:

\[
\begin{align*}
f(O^*) - f(S) & \leq d \cdot \frac{\epsilon \lambda}{d} \leq \epsilon f(O^*) \\
& \implies f(S) \geq (1 - \epsilon)f(O^*)
\end{align*}
\]

Hence we just need to consider case (2), and we assume that \( \tau = \tau_s \) when the algorithm stops. Using the same notation and arguments as section 4.1.1 we are able to establish the following results:

\[
\begin{align*}
f(S) \geq \tau_s \sum_{e \in E} \sum_{i=1}^d C_i(e) & \geq \tau_s \left( 1 - \sum_{i=1}^d C_i(\bar{e}_1) \right) \quad (39) \\
f(\tilde{S}) \geq \tau_s \left( 1 - \sum_{i=1}^d C_i(\bar{e}_2) \right) \quad (40) \\
f(\{\bar{e}_1, \bar{e}_2\}) \geq \tau_s \sum_{j=1}^2 \sum_{i=1}^d C_i(e) \quad (41)
\end{align*}
\]
In addition, consider the threshold $\tau' = \frac{\tau_s}{1 - \varepsilon}$ and let $S' \subseteq S$ denote the final solution set in the iteration $\tau = \tau'$. Then

$$f(O^* \setminus O_b^*) - f(S) \leq \sum_{e \in O^* \setminus (O_b^* \cup S)} f(S + e) - f(S) \leq \sum_{e \in O^* \setminus (O_b^* \cup S)} f(S' + e) - f(S) \leq \frac{\tau_s}{1 - \varepsilon} \left( d - \frac{1}{2} |O_b^*| \right)$$

$$\implies f(S) \geq \frac{OPT - (d - \frac{1}{2} |O_b^*|) \tau_s}{|O_b^*| + 1} - \Omega(\varepsilon)OPT$$ (42)

Combining the inequalities above, we can obtain

$$f(S^*) \geq \max \left\{ \frac{1 + \sum_{i=1}^{d} C_i(e)}{2} \frac{OPT - (d - \frac{1}{2} |O_b^*|) \tau_s}{|O_b^*| + 1} \right\} - \Omega(\varepsilon)OPT$$ (43)

$$\geq \frac{[1 + \sum_{i=1}^{d} C_i(\tilde{e}_1)]}{[1 + \sum_{i=1}^{d} C_i(\tilde{e}_1)](1 + d) + d} OPT - \Omega(\varepsilon)OPT$$ (44)

On the other hand, similar as the analysis of greedy, we could obtain

$$f(S) \geq \left( 1 - e^{-\sum_{i=1}^{d} C_i(\tilde{e}_1)} \right) f(O^* \setminus O_b^*)$$

when $d = 1$, our algorithm actually compute a solution such that

$$\frac{f(S^*)}{OPT} \geq \max \left\{ \frac{1 + x}{3 + 2x}, \frac{1 - e^{2x-2}}{2 - e^{2x-2}} \right\} - \Omega(\varepsilon) \geq 0.377 - \Omega(\varepsilon)$$

Here we use $x$ to denote $\sum_{i=1}^{d} C_i(\tilde{e}_1)$.

Finally we have the following corollary

**Corollary 20.** Algorithm 3 is an $(0.377 - \Omega(\varepsilon))$-approximation algorithm for maximizing monotone submodular function under knapsack constraint, whose time complexity is in the order of $O(n \cdot \{\varepsilon^{-1}, \log \log n\})$.

One note here is, we could assume that $\tau'$ is not the maximum threshold, which guarantee that $\tau'$ is used as a threshold in the algorithm. Otherwise we will be able obtain an $\frac{1}{2}$-approximation solution since $\max\{f(\tilde{e}), f(S)\} \geq \frac{OPT}{2}$.

## 5 Faster algorithm for matroid constraint and monotone submodular function with curvature

In this section we present a faster algorithm for maximizing monotone submodular function $f(\cdot)$ with bounded curvature under a matroid constraint. The general outline of our algorithm follows the modified continuous greedy algorithm [20], which indeed regards the problem as a multi-objective maximization problem and computes an approximate Pareto-optimal front.

Here we show that the complexity of the modified continuous greedy can be reduced by some rounding operations on the marginal gains that appears in the LP, which enables us to find the optimal solution in a faster way.
Algorithm 4: Faster algorithm for maximizing submodular function with bounded curvature under matroid constraint

1. \( v \leftarrow c\text{-approximation of } \text{OPT} \)
2. \( \beta \leftarrow \frac{r}{\varepsilon(1-\kappa) \max_{e} f(e)} \)
3. \( \delta \leftarrow (1-\kappa) \)
4. \( w(e) \leftarrow (1+\varepsilon)^{\lceil \log_{1+\varepsilon} \beta h(e) \rceil} \)
5. \( w = (w(e) \cdot \mathbb{1}_{(w(e) \geq 1)} \}_{e \in E} \)
6. \( \theta \leftarrow \varepsilon \beta v, \bar{\theta} \leftarrow \min \left\{ \frac{\beta v}{\epsilon}, \beta \max_{S \in \mathcal{I}} (w, \mathbb{1}_{S}) \right\} \)
7. \( \text{while } \theta \leq \bar{\theta} \text{ do} \)
8. \( t \leftarrow 0, y_{\theta}(0) \leftarrow 0 \)
9. \( \text{while } t \leq 1 \text{ do} \)
10. \( \tilde{M}_{\theta}(t) \leftarrow \max_{e} E_{R(t) \sim \mathcal{Y}_{\theta}(t)} \left[ g_{R(t)}(e) \right] \)
11. \( p_{\theta}^e(t) = (1+\varepsilon)^{\log_{1+\varepsilon}\frac{r E_{R(t) \sim \mathcal{Y}_{\theta}(t)} g_{R(t)}(e)}}{\epsilon \tilde{M}_{\theta}(t)} \)
12. \( y_{\theta}(t) \leftarrow (p_{\theta}^e(t) \cdot \mathbb{1}_{(p_{\theta}^e(t) \geq 1)} \}_{e \in E} \)
13. \( x_{\theta}^e(t) \leftarrow \text{LP}(w, p_{\theta}(t), \theta, \mathcal{M}) \)
14. \( y_{\theta}(t+\delta) = y_{\theta}(t) + \delta \cdot x_{\theta}^e(t) \)
15. \( t = t + \delta \)
16. \( \theta \leftarrow \theta(1+\varepsilon) \)
17. \( S_{\theta} \leftarrow \text{Swap-Rounding} \left( y_{\theta}(1), \mathcal{M} \right) \)
18. \( \text{return } S^* \leftarrow \arg\max_{S_{\theta}} f(S_{\theta}) \)

5.1 Analysis of Algorithm

Here function \( f(\cdot) \) is decomposed as \( f(\cdot) = g(\cdot) + h(\cdot) \), where \( h(S) = (1-\kappa_f-\varepsilon) \sum_{e \in E} f(e) \). It’s not hard to check that \( g(\cdot) \) is also a monotone submodular function. Moreover, we could assume that \( 1-\kappa_f \) is bounded away from 1, i.e., \( 1-\kappa_f = \Omega(\varepsilon) \), this is due to the fact that if \( \kappa_f \) is close to 1, then the standard continuous continuous greedy is enough in terms of approximation ratio. As a consequence, the curvature of \( g(\cdot) \) is also bounded away from 1 according to definition of \( h(\cdot) \).

Sub-problem \( \text{LP}(w, p_{\theta}(t), \theta, \mathcal{M}) \). \( x_{\theta}^e(t) \) is indeed the optimal solution of the following linear programming problem:

\[
\max_{x \geq 0} p_{\theta}(t) \cdot x \\
\text{s.t. } w^T x \geq \theta \\
\sum_{e \in X} x_e \leq r(X) \forall X \subseteq E
\]

where \( r(\cdot) : 2^E \rightarrow \mathbb{Z} \), the rank function of matroid \( \mathcal{M} \), is defined as:

\[
r(X) = \max\{|Y| : Y \subseteq X, Y \in \mathcal{I}\}
\]
We introduce the cover constraint into the objective function by Lagrangian relaxation. Define
\[
\phi_\theta(\lambda, t) = \max_{x \geq 0} \left( p_t^T(t) + \lambda w^T \right) \cdot x - \lambda \theta \\
\sum_{e \in X} x_e \leq r(X) \ \forall X \subseteq E
\]  
and let
\[
\Phi_\theta(t) = \min_{\lambda \geq 0} \phi_\theta(\lambda, t)
\]

Some simple facts about \( \phi_\theta(\lambda, t) \). For fixed values of \( t, \theta \) and \( \lambda \), function \( \phi_\theta(\lambda, t) \) can be computed by figuring out \( B_\lambda \), the maximum weight independent set in \( M \), when the weights are given by \( \{ p_t^1(t) + \lambda w_e \}_{e \in E} \). And as long as the ordering of elements does not change, the max-weight base will not change either, which implies that \( \phi_\theta(\cdot) \) is indeed a piecewise linear function with respect to \( \lambda \). For each line segment, the corresponding slope is equal to \( w(B_\lambda) = \sum_{e \in B_\lambda} w_e \). Moreover, \( \phi_\theta(\cdot) \) is indeed a convex function in \( \lambda \), thus the problem of finding the optimal lagrange multiplier \( \lambda^* \), is then reduced to find \( \lambda^* \) such that \( w(B_\lambda) - l(\lambda - \lambda^*) \geq 0 \) holds for all \( \lambda \in [0, +\infty) \).

On the other hand, when \( \lambda = 0 \), elements with larger value of \( p_t^0(t) \) are selected into the solution set first. As \( \lambda \) varies from 0 to +\( \infty \), swaps will occur in the ordered sequence and finally the elements are sorted in decreasing order of \( w_e \). Notice that swaps occur only when the weight of two element become the same, \( \text{i.e., } p_t^{e_1}(t) + \lambda w_{e_1} = p_t^{e_2}(t) + \lambda w_{e_2} \), which implies that \( \lambda = \frac{y - y_0}{w_y - y_0} \). We call such \( \lambda \) turning-point, and let \( T_\theta(t) \) denote the collection of all the turning-points, together with point 0, \( \text{i.e., } T_\theta(t) = \{ \lambda \mid \exists i, j \text{ such that } \lambda = \frac{y - y_0}{w_y - y_0} \} \cup \{0\} \). One remark is, all the turning-point will result in swap, but it may not change \( B_\lambda \).

Evaluating \( \phi_\theta(\lambda, t) \) efficiently. It’s clear to see that \( \phi_\theta(\lambda, t) \) can be computed by applying the greedy algorithm on matroid \( M \). Here we show one trick that could be utilized to compute \( B_\lambda \) in \( O(n + \frac{\log^2(n/\varepsilon)}{\varepsilon^2}) \), which exploits the fact that the number of distinct weights is small after our rounding process. Actually we can sort the elements in the same spirit as the counting sort. In \( O(n) \) time we can compute the frequency histogram for all the possible pairs \((1 + \varepsilon)^j, (1 + \varepsilon)^l\), then performs a prefix sum computation to figure out the sorted position for each element.

Compute the primal optimal solution. Let \( W_\lambda^*(e) = p(e) + \lambda w(e) \) denote the weight of elements in \( [M] \). As we have discussed, after obtaining the value of \( \lambda^* \), we are able to compute \( x^*_e = \arg \max \phi_\theta(\lambda^*, t) \) by greedy. Let \( A_j = \{ e_1, e_2, \ldots, e_j \} \) in which elements are listed in the decreasing order of weight \( W_{\lambda^*}(\cdot) \). In fact we are able to obtain the corresponding dual optimal solution \( y^* = \{ y_X \}_{X \subseteq E} \) according to \( [18] \):
\[
y(A_j) = W_{\lambda^*}(e_j) - W_{\lambda^*}(e_{j+1}), \\
y(X) = 0, \text{if } X \neq A_j \forall j \in [n]
\]
By the Complementary Slackness Condition, we know that the optimal primal solution \( x^* \) satisfies that:
\[
\left( \sum_{e \in A_j} x^*_e - r(A_j) \right) \cdot y(A_j) = 0 \ \forall j \in [n]
\]
Since $\lambda^*$ is a turning-point, there must exist $e_i(\lambda^*) \in B_{\lambda^*}$, $e_i(\lambda^*)+1 \in E \setminus B_{\lambda^*}$ such that $W_{\lambda^*}(e_i(\lambda^*)) = W_{\lambda^*}(e_i(\lambda^*)+1)$, it follows that $y(A_i(\lambda^*)) = 0$. One important remark is, we can actually assume that $y(A_i) \neq 0 \ (\forall i \in [n] \setminus \{i(\lambda^*)\})$. Otherwise we apply small perturbations on $p_{\theta}(t)$—let $p^e_{\theta}(t) = p^e_{\theta}(t) + \varepsilon$ ($\forall e \in E$), where $\varepsilon_i > \varepsilon_{i+1} > 0 \ (\forall i \in [n] \setminus \{i(\lambda^*)\})$ and $e_i(\lambda^*) = e_i(\lambda^*)+1$. We could let $\varepsilon_i \to 0$ to guarantee that the effects of $\varepsilon_i \in E$ on the value of $\phi_{\theta}(\lambda, t)$ and break points are sufficiently small, then $\arg \min_{\lambda \geq 0} \phi_{\theta}(\lambda, t)$ is still equal to $\lambda^*$ and the dual optimal solution remain the same. Hence \((40)\) can be simplified to

$$\sum_{e \in A_j} x^*_e = r(A_j) \ \forall j \in [n] \setminus \{1\} \quad (47)$$

$$w^T x = \theta \quad (48)$$

Based on which, we can derive the following observation about $y_{\theta}(1)$.

**Observation 21. In the primal optimal solution derived obtained in \((44)\), there are at most two fractional components in $x^*$. Moreover, $y_{\theta}(1)$ can be represented as the convex combination of $\frac{2}{3}$ vertices of $\mathcal{P}(\mathcal{M})$.**

**Proof:** Actually $x^*_{e_i(\lambda^*)} = r(A_i(\lambda^*)) - r(A_i(\lambda^*)-1)$ for $\forall i \in [n] \setminus \{i(\lambda^*), i(\lambda^*)+1\}$, thus only $x^*_{e_i(\lambda^*)}$ and $x^*_{e_i(\lambda^*)+1}$ could be fractional valued. Moreover, since $e_i(\lambda^*)+1 \notin B_{\lambda^*}$, we have $r(A_i(\lambda^*)+1) - r(A_i(\lambda^*)-1) = 1$, which implies that $x^*_{e_i(\lambda^*)} + x^*_{e_i(\lambda^*)+1} = 1$. Now let $B^+_{\lambda^*} = B_{\lambda^*} \cup \{e_i(\lambda^*)+1\} \setminus \{e_i(\lambda^*)\}$, then $x^*_i = x^*_{e_i(\lambda^*)} \cdot 1_{B^+_{\lambda^*}} + (1 - x^*_{e_i(\lambda^*)}) \cdot 1_{B^+_{\lambda^*}}$. It follows that $y_{\theta}(1) = \sum_{t=1}^{1/3} x^*_i(t)$, which indicates that $y_{\theta}(1)$ is a convex combination of $\frac{2}{3}$ independent sets. □

### 5.2 Analysis of the algorithm

In this section, we analyze the performance of Algorithm \[.\] We mainly show that the the lost incurred by rounding is $(1 - \Omega(\varepsilon))$.

**Lemma 22.**

$$G(y_{\theta}(t + \delta)) - G(y_{\theta}(t)) \geq \delta((1 - \Omega(\varepsilon))OPT - G(y_{\theta}(t)))$$

**Proof:** Without loss of generality we assume that $h(S^*) \geq \varepsilon v$, otherwise the standard continuous greedy algorithm could provide an $(1 - \frac{\varepsilon}{\beta} - \Omega(\varepsilon))$-approximation solution. Notice that there must exists an iteration in which $\theta^* \in [(1 - \varepsilon)\beta h(S^*), \beta h(S^*)]$. Since $x_{\theta^*} = \arg \max_{x} p_{\theta^*} \cdot x$, we have

$$\langle p_{\theta^*}(t), x_{\theta^*}(t) \rangle \geq \langle p_{\theta^*}, 1_{O^*} \rangle \quad (49)$$

$$= \sum_{e \in O^*} p^e_{\theta^*}(t) \cdot 1_{\{p^e_{\theta^*}(t) \geq 1\}}$$

$$= \sum_{e \in O^*} p^e_{\theta^*}(t) - \sum_{e \in O^*} p^e_{\theta^*}(t) \cdot 1_{\{p^e_{\theta^*}(t) \leq 1\}}$$

$$\geq \frac{r(1 - \varepsilon)}{\varepsilon M_{\theta^*}(t)} \sum_{e \in O^*} E_{R(t)} y_{\theta^*}(t) \left[ g_{R(t)}(e) \right] - r \quad (50)$$

23
where (a) is due to the fact that \( O^* \) is feasible since \( \theta^* \leq \beta h(S^*) \). Similar as the continuous greedy, further the following inequalities holds.

\[
\begin{align*}
&\geq \frac{r(1-\varepsilon)}{\varepsilon M_{\theta^*}(t)} E_{R(t) \sim y_\theta(t)}[g(R(t) \cup O^*) - g(R(t))] - r \\
&= \frac{r(1-\varepsilon)}{\varepsilon M_{\theta^*}(t)} [G(y_\theta(t) \cup 1_{O^*}) - G(y_\theta(t))] - r \\
&\geq \frac{r(1-\varepsilon)}{\varepsilon M_{\theta^*}(t)} [OPT - G(y_\theta(t))] - r
\end{align*}
\]

On the other hand, the LHS of (49) could be upper bounded as

\[
\langle p_{\theta^*}(t), x_{\theta^*}(t) \rangle \leq \sum_{e \in E} p_{\theta^*}(t) \cdot x_{\theta^*}(t)
\]

\[
\leq \frac{r}{\varepsilon M_{\theta^*}(t)} \sum_{e \in E} x_{\theta^*}(t) \cdot E_{R(t) \sim y_\theta(t)}[g(R(t)(e)]
\]

Notice that \( E_{R(t) \sim y_\theta(t)}[f_R(t)(e)] \leq \frac{\partial E}{\partial x_e}(y_\theta(t)) \), thus

\[
\langle p_{\theta^*}(t), x_{\theta^*}(t) \rangle \leq \frac{r}{\varepsilon M_{\theta^*}(t)} \sum_{e \in E} x_{\theta^*}(t) \cdot \frac{\partial G}{\partial x_e}(y_\theta(t))
\]

\[
= \frac{r}{\varepsilon M_{\theta^*}(t)} \langle \nabla G(y_\theta(t)), x_{\theta^*}(t) \rangle
\]

Put these together, we have

\[
\langle \nabla G(y_\theta(t)), x_{\theta^*}(t) \rangle \geq (1 - \varepsilon)(OPT - G(y_\theta(t))) - \varepsilon \tilde{M}_{\theta^*}(t) \geq (1 - 2\varepsilon)OPT - G(y_\theta(t))
\]

where the last inequality is due to the fact that \( OPT \geq \tilde{M}_{\theta^*}(t) \). Using taylor ex

\[
G(y_\theta(t + \delta)) - G(y_\theta(t)) = G(y_\theta(t) + \delta x_\theta^*(t)) - G(y_\theta(t))
\]

\[
= \delta \langle x_\theta^*(t), \nabla G(y_\theta(t)) \rangle + \frac{1}{2} \delta^2 x_\theta^* T(t) \nabla^2 G(z)x_\theta^*(t)
\]

Since \( |\frac{\partial^2 F(z)}{\partial x \partial x_j}| \leq \min\{f(i), f(j)\} \), then

\[
\delta^2 x_\theta^* T(t) \nabla^2 G(z)x_\theta^*(t) \leq \frac{2rg(O^*)}{1 - \kappa_g}
\]

since \( \delta = O(\frac{\varepsilon (1 - \kappa_g)}{r}) \),

\[
G(y_\theta(t + \delta)) - G(y_\theta(t)) \geq \delta((1 - \Omega(\varepsilon))OPT - G(y_\theta(t)))
\]

\[\square\]

In terms of the loss in modular function \( h(\cdot) \), the proof is similar, please refer to our technique report [11] for more details. Also the \( \tilde{O}(nr) \) value oracle queries is easy to verify.
5.3 An additional note on the notation of curvature

In this subsection, we discuss the notation of total curvature and curvature with respect to the specific set. Let \( f : 2^E \rightarrow R \) be a monotone non-decreasing submodular function. For any set \( T \subseteq E \), we define the associate curvature as:

\[
\kappa(T) = 1 - \min_{i \in T} \frac{f(T) - f(T \setminus \{i\})}{f(\{i\})}
\]

and we use \( \kappa = \kappa(E) \) to denote the total curvature.

For a general constraint \( C \), consider the constrained maximizing problem: maximize \( f(S) \) subject to \( S \in C \). Then we have the following conclusion for Greedy algorithm:

**Proposition 23.** For the maximization problem above, if the approximation ratio of greedy algorithm \( A \) is \( \lambda(\kappa) \), then the ratio can be improved to \( \lambda(\kappa(S_g \cup S^*)) \), where \( S_g \) is the set returned by greedy algorithm and \( S^* \) is the optimal solution.

**Proof:** For notation convenience, we use \( U \) to denote set \( S_g \cup S^* \). Notice that there exists a set function \( h : 2^E \rightarrow R \) satisfies the following rule:

\[
h(S) = f(S \cap U) + (1 - \kappa(U)) \sum_{i \in S \setminus U} f(i)
\]

We claim that function \( h \) is monotone non-decreasing function with total curvature \( \kappa(U) \). Actually \( f(S \cap U) \) is a non-decreasing submodular function with regard to set \( S \) and \( (1 - \kappa(U)) \sum_{i \in S \setminus U} f(i) \) is a non-decreasing modular function with regard to \( S \), thus \( h(S) \) is a non-decreasing submodular function. And the total curvature of \( h \) is:

\[
\kappa_h = 1 - \min_{i \in E} \frac{h(E) - h(E \setminus \{i\})}{h(\{i\})} \\
\leq 1 - \min_{i \in E} \frac{h(E) - h(E \setminus \{i\})}{f(\{i\})}
\]

Notice that

\[
h(E) - h(E \setminus \{i\}) = \begin{cases} f(U) - f(U \setminus \{i\}) & i \in U \\
(1 - \kappa(U))f(\{i\}) & i \notin U \end{cases}
\]

in both cases we have \( h(U) - h(U \setminus \{i\}) \geq (1 - \kappa(U))f(\{i\}) \), which implies that \( \kappa_h \leq \kappa(U) \). Actually there must exists \( j \in U \) such that \( f(U) - f(U \setminus \{j\}) = (1 - \kappa(U))f(j) \), which is equivalent to \( h(U) - h(U \setminus \{j\}) = (1 - \kappa(U))h(j) \) and thus we have \( \kappa_h = \kappa(U) \).

If we apply greedy algorithm to function \( h \), suppose that we obtain set \( G_i \) in the \( i \)-th step \( (1 \leq i \leq r) \). Then we claim that \( G_r = S_g \), actually the set we obtain at each step is the same with the set obtained by applying greedy algorithm to \( f \). This claim can be proved by induction according to the selection rule of greedy algorithm and the following two inequalities:

\[
h(G_i) = f(G_i), \forall 1 \leq i \leq r,
\]

\[
h(S) \leq f(S), \forall S \subseteq E,
\]
Moreover, \( f(S^*) = g(S^*) \), therefore we can conclude that the approximation ratio can be improved to \( \lambda(\kappa(S^* \cup S_g)) \).

\[ \square \]

References

[1] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms, pages 1497–1514. Society for Industrial and Applied Mathematics, 2014.

[2] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing apples and oranges: query tradeoff in submodular maximization. In Proceedings of the twenty-sixth annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1149–1168. Society for Industrial and Applied Mathematics, 2015.

[3] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. SIAM Journal on Computing, 40(6):1740–1766, 2011.

[4] Chandra Chekuri, Jan Vondrak, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on, pages 575–584. IEEE, 2010.

[5] Michele Conforti and Gérard Cornuéjols. Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem. Discrete applied mathematics, 7(3):251–274, 1984.

[6] Moran Feldman. Submodular maximization with cardinality constraints. https://www.openu.ac.il/personal_sites/moran-feldman/selected-presentations/Submodular%20Maximization%20with%20Cardinality%20Constraints.pptx.

[7] Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on, pages 570–579. IEEE, 2011.

[8] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms (SODA), pages 1098–1116. Society for Industrial and Applied Mathematics, 2011.

[9] Michael Gygli, Helmut Grabner, and Luc Van Gool. Video summarization by learning submodular mixtures of objectives. In Proceedings CVPR 2015, pages 3090–3098, 2015.

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

[11] Wenxin Li and Ness B. Shroff. Constrained monotone submodular maximization. Technical Report, 2018.
[12] Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 912–920. Association for Computational Linguistics, 2010.

[13] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies—Volume 1, pages 510–520. Association for Computational Linguistics, 2011.

[14] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In Optimization techniques, pages 234–243. Springer, 1978.

[15] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In International Conference on Machine Learning, pages 1358–1367, 2016.

[16] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. Lazier than lazy greedy. In AAAI, pages 1812–1818, 2015.

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

[18] Alexander Schrijver. Combinatorial optimization: polyhedra and efficiency, volume 24. Springer Science & Business Media, 2003.

[19] J Michael Steele. The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities. Cambridge University Press, 2004.

[20] Maxim Sviridenko, Jan Vondrák, and Justin Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. In Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms (SODA), pages 1134–1148. Society for Industrial and Applied Mathematics, 2015.

[21] Jan Vondrák. Submodularity and curvature: The optimal algorithm (combinatorial optimization and discrete algorithms). 2010.

[22] Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In Foundations of Computer Science, 1977., 18th Annual Symposium on, pages 222–227. IEEE, 1977.

[23] Yisong Yue and Carlos Guestrin. Linear submodular bandits and their application to diversified retrieval. In Advances in Neural Information Processing Systems, pages 2483–2491, 2011.