Submodular Maximization under the Intersection of Matroid and Knapsack Constraints

Yu-Ran Gu, Chao Bian, Chao Qian

State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
{guyr, bianc, qianc}@lamda.nju.edu.cn

Abstract

Submodular maximization arises in many applications, and has attracted a lot of research attentions from various areas such as artificial intelligence, finance and operations research. Previous studies mainly consider only one kind of constraint, while many real-world problems often involve several constraints. In this paper, we consider the problem of submodular maximization under the intersection of two commonly used constraints, i.e., $k$-matroid constraint and $m$-knapsack constraint, and propose a new algorithm SPROUT by incorporating partial enumeration into the simultaneous greedy framework. We prove that SPROUT can achieve a polynomial-time approximation guarantee better than the state-of-the-art algorithms. Then, we introduce the random enumeration and smooth techniques into SPROUT to improve its efficiency, resulting in the SPROUT++ algorithm, which can keep a similar approximation guarantee. Experiments on the applications of movie recommendation and weighted max-cut demonstrate the superiority of SPROUT++ in practice.

Introduction

Submodular maximization, i.e., maximization of a set function which satisfies the diminishing returns property under some constraints, arises in many applications, e.g., influence maximization (Kempe, Kleinberg, and Tardos 2003; Qian et al. 2018), data summarization (Lin and Bilmes 2011; Sapos et al. 2012; Dassgupta, Kumar, and Ravi 2013), sparse regression (Das and Kempe 2011; Qian, Yu, and Zhou 2015), sensor placement (Krause and Guestrin 2005), adversarial attack (Liu et al. 2021), and human assisted learning (De et al. 2020; Liu, Mu, and Qian 2023). This problem is generally NP-hard, and the design of polynomial-time approximation algorithms has attracted much attention.

The pioneering work of Nemhauser, Wolsey, and Fisher (1978) and Fisher, Nemhauser, and Wolsey (1978) revealed the good performance of the classic greedy algorithm which achieves an approximation ratio of $(1 - 1/e)^{-1}$ under the cardinality constraint and a $(k + 1)$-approximation under the more general $k$-matroid constraint when maximizing a monotone submodular function. When the objective function is non-monotone, Lee, Sviridenko, and Vondrak (2010) provided a $(k + 1 + 1/(k - 1) + e)$-approximation for the $k$-matroid by local search requiring $\text{poly}(n) \cdot \exp(k, e)$ running time, where $n$ is the problem size and $e > 0$. These studies mainly focus on submodular maximization under one kind of constraint, while real-world problems often deal with multiple constraints simultaneously, e.g., movie recommendation under cardinality and rating constraints (Mirzasoleiman, Badanidiyuru, and Karbasi 2016), and Twitter text summarization under cardinality and date constraints (Badanidiyuru et al. 2020).

For submodular maximization under matroid and knapsack constraints, Chekuri, Vondrak, and Zenklusen (2010) proposed an algorithm which achieves a $(k/0.19 + e)$-approximation for $k$-matroid and $m$-knapsack constraints with the time complexity of $\text{poly}(n) \cdot \exp(k, m, e)$, which, however, may be unacceptable. Mirzasoleiman, Badanidiyuru, and Karbasi (2016) developed the first practical algorithm, FANTOM, offering a $((1 + e)/(1 + 1/k)(2k + 2m + 1))$-approximation for the intersection of the very general $k$-system and $m$-knapsack constraints with $O(n^2/e)$ oracle calls. For the sake of simplicity, we ignore poly-logarithmic factors by using the $O$ notation. More recently, Feldman, Harshaw, and Karbasi (2020) designed DENSITYSEARCHSGS based on the simultaneous greedy algorithmic framework to solve this problem, which achieves an approximation ratio of $(1 + e)(k + 2m) + O(\sqrt{k + m})$ with $O(n/e)$ oracle calls. The approximation ratio becomes $(1 + e)(k + 2m) + O(\sqrt{m})$ for the intersection of $k$-extendible and $m$-knapsack constraints and $(1 + e)(k + 2m + 1)$ for a monotone objective function, where $k$-extendible is a subclass of $k$-system.

Note that the constraints (i.e., $k$-system) considered in previous studies may be so general that the proposed algorithms may not perform well under some important subclasses of these constraints. In this paper, we consider the (not necessarily monotone) submodular maximization problem under the intersection of $k$-matroid and $m$-knapsack constraints, which arises in numerous applications, e.g., vertex cover (Delong et al. 2012), weighted max-cut (Feldman, Harshaw, and Karbasi 2017; Habadi et al. 2020), video summarization (Gygli, Grabner, and Gool 2015; Feldman, Karbasi, and Kazemi 2018), image summarization and revenue maximization (Mirzasoleiman, Badanidiyuru, and Karbasi 2016). We propose a Simultaneous and Partial enumeration algorithm, called SPROUT++.
SPROUT, by incorporating the partial enumeration technique into the simultaneous greedy algorithmic framework. SPROUT offers an opportunity of balancing the approximation guarantee with the running time by a parameter $C$, i.e., as $C$ increases, the approximation ratio improves while the time complexity raises. In particular, when $C = 1$, SPROUT achieves a $O(n^2/\epsilon)$ approximation using $\tilde{O}(n^2/\epsilon)$ oracle calls,

1 which is better than the state-of-the-art algorithms, as shown in Table 1. When the objective function is monotone, the approximation ratio improves to $(1 + \epsilon)(k + m + 1)$. Since the partial enumeration used in SPROUT may be too time-consuming and some good solutions in binary search are ignored, we propose a more practical and efficient algorithm, i.e., SPROUT++, by introducing the random enumeration and smooth techniques into SPROUT. We prove that SPROUT++ can achieve a similar approximation guarantee to SPROUT. Experiments are conducted on the applications of movie recommendation and weighted max-cut, demonstrating the superior performance of SPROUT++ in practice.

### Preliminaries

Given a ground set $N$, we study the submodular functions $f : 2^N \to \mathbb{R}$, i.e., set functions satisfying the diminishing returns property. Specifically, a set function is submodular if $f(e \cup A) \geq f(e \cup B)$, $\forall A \subseteq B \subseteq N$ and $e \notin B$, where $f(S) = f(S \cup A) - f(A)$ means the marginal gain of adding a set $S$ to $A$. Note that we do not distinguish the element $e$ and the corresponding single-element set $\{e\}$ for convenience. Without loss of generality, we assume the functions are non-negative.

Now we introduce the considered constraints. Given a set system $(N, \mathcal{I})$ where $\mathcal{I} \subseteq 2^N$, $(N, \mathcal{I})$ is called an independence system if (1) $\emptyset \in \mathcal{I}$; (2) $\forall A \subseteq B \subseteq N$, if $B \in \mathcal{I}$ then $A \in \mathcal{I}$. An independence system is called a matroid $\mathcal{M}(N, \mathcal{I})$ if $\forall A, B \in \mathcal{I}$ and $|A| < |B|$, there is $e \in B \setminus A$ such that $A \cup e \in \mathcal{I}$. We present $k$-matroid in Definition 1.

**Definition 1** ($k$-Matroid). Given $k$ matroids $\mathcal{M}_i(N, \mathcal{I}_i)$ defined on the ground set $N$, $k$-matroid is a matroid $\mathcal{M}(N, \mathcal{I})$, where $\mathcal{I} = \bigcap_{i=1}^{k} \mathcal{I}_i$.

Besides, given a modular cost function $c : 2^N \to \mathbb{R}$, i.e., $\forall A \subseteq N$, $c(A) = \sum_{e \in A} c(e)$, and a budget $B$, the knapsack constraint means that the cost of a subset should be upper bounded by $B$, i.e., $c(S) \leq B$. Without loss of generality, the budget $B$ is normalized to 1. Next, we present $m$-knapsack constraint in Definition 2. We use $[m]$ (where $m$ is a positive integer) to denote the set $\{1, 2, \ldots, m\}$.

**Definition 2** ($m$-Knapsack Constraint). Given $m$ modular cost functions $c_1, c_2, \ldots, c_m$, a set $S \subseteq N$ satisfies the $m$-knapsack constraint if and only if $\forall i \in [m], c_i(S) \leq 1$.

Instead of meeting one of these constraints, real-world applications often involve them simultaneously. In this paper, we study the problem of submodular maximization under the intersection of $k$-matroid and $m$-knapsack constraints.

**Definition 3** (Submodular Maximization Under the Intersection of $k$-Matroid and $m$-Knapsack Constraints). Given a submodular function $f : 2^N \to \mathbb{R}$, a $m$-knapsack constraint with cost functions $c_1, c_2, \ldots, c_m$, and a $k$-matroid $\mathcal{M}(N, \bigcap_{i=1}^{k} \mathcal{I}_i)$, to find $\max_{S \subseteq N} f(S)$ such that $S \in \bigcap_{i=1}^{k} \mathcal{I}_i$ and $\forall i \in [m], c_i(S) \leq 1$.

Mirzasoleiman, Badanidiyuru, and Karbasi (2016) proposed FANTOM which can find a good solution by employing density threshold greedy multiple times. If the solutions generated-so-far are not good, there must exist high-quality solutions in the elements that never being chosen before, and density threshold greedy will continue to search. Feldman, Harshaw, and Karbasi (2020) provided DENSITYSEARCHSGS achieving a better approximation guarantee with lower time complexity by utilizing the simultaneous greedy framework. However, it seems that DENSITYSEARCHSGS may be so delicate that it achieves previously the best approximation guarantee while behaving not that well in practice, as mentioned in (Feldman, Harshaw, and Karbasi 2020). Aimed to get around it, we propose the algorithms, SPROUT and SPROUT++.

### SPROUT Algorithm

The basic idea of SPROUT is incorporating a partial enumeration technique (Badanidiyuru et al. 2020) into the simultaneous greedy framework (Feldman, Harshaw, and Karbasi 2020), such that it can be more robust in practice and provide a better approximation ratio. Specifically, SPROUT as presented in Algorithm 1 enumerates the feasible set $A$ with a size of $C$ as the first part of solution set, and then selects the set $S_A$ from the remaining elements using the KNAPSACKSGS subroutine as presented in Algorithm 2. Finally, it returns the best solution $A \cup S_A$ over all feasible $A$. In the following, we describe in more details about SPROUT. First, we define a new objective function $z_A$ in line 2 for the enumerated feasible set $A \subseteq N$ in line 1. Then, a reduced ground set $N'$ is derived from removing elements belonging to $A$ or whose value of $z_A$ is larger than

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Approximation</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FANTOM (Mirzasoleiman, Badanidiyuru, and Karbasi 2016)</td>
<td>$(1 + \epsilon)(2k + (2 + 2/k)m) + O(1)$</td>
<td>$\tilde{O}(n^2/\epsilon)$</td>
</tr>
<tr>
<td>DENSITYSEARCHSGS (Feldman, Harshaw, and Karbasi 2020)</td>
<td>$(1 + \epsilon)(k + 2m) + O(\sqrt{m})$</td>
<td>$\tilde{O}(n/\epsilon)$</td>
</tr>
<tr>
<td>SPROUT (this paper)</td>
<td>$(1 + \epsilon)(k + m) + O(\sqrt{m})$</td>
<td>$\tilde{O}(n^2/\epsilon)$</td>
</tr>
</tbody>
</table>

Table 1: Comparison of the state-of-the-art algorithms for submodular maximization under the intersection of $k$-matroid and $m$-knapsack constraints. For the proposed SPROUT algorithm, the parameter $C = 1$ is used here.
Algorithm 1: SPROUT

**Input:** Objective function $f : 2^\mathcal{N} \to \mathbb{R}_+$, $k$ matroids $\mathcal{M}_i(\mathcal{N}, T_i)$ and $m$ cost functions $c_i : \mathcal{N} \to \mathbb{R}_+$

**Parameter:** Error params $\delta, \epsilon$, correction params $\beta, \gamma$, enumeration param $C$ and number $f$ of solutions

**Output:** A set $S$ s.t. $S \in \bigcap_{i=1}^k T_i$ and $\forall i \in [m], c_i(S) \leq 1$

1. for each feasible $A \subseteq \mathcal{N}$ with $C$ elements do
   2. $z_A(S) \triangleq f(S|A)$.
   3. $\mathcal{N}' \triangleq \{ e \in \mathcal{N} : e \notin A \land C \cdot z_A(e) \leq f(A) \}$.
   4. $\mathcal{M}'(\mathcal{N}', T'_i) = \text{contraction of } \mathcal{M}_i(\mathcal{N}, T_i) \text{ by } A$.
   5. $T_i' \triangleq \bigcap_{i=1}^k T'_i$.
   6. Decrease knapsack budgets by $c_i(A)$ and normalize each of them to 1.
   7. Let $b_1 = 1$ and $V$ be the maximum $z_A$ value of a single feasible element in $\mathcal{N}'$.
   8. while $|b_1 - b_0| > 1$ do
      9. $\rho = \beta V(1 + \delta)(|b_1 + b_0 + 1|/2) + \gamma f(A)/C$.
      10. $S_K = \text{KNAPSACKGS}(z_A, \mathcal{N}', T'_i, (c_i)_{i=1}^m, \rho, \epsilon)$.
      11. Add $S_K$ to $S_0$.
      12. $b_E = [(b_1 + b_0 + 1)/2]$.
   13. end while
   14. $S_A = \text{arg max } S \in S_0, f(S)$.
   15. end for
   16. $A^* = \text{arg max } A, f(A \cup S_A)$ over all feasible $A \subseteq \mathcal{N}$.
   17. return $A^* \cup S_{A^*}$

$f(A)/C$ from $\mathcal{N}$ in line 3. After selecting the set $A$, we need to contract matroids from $\mathcal{M}_i(\mathcal{N}, T_i)$ to $\mathcal{M}'_i(\mathcal{N}', T'_i)$ by $A$ in line 4, where $T'_i$ is the set $\{S \subseteq \mathcal{N}' : S \cup A \subseteq T'_i \}$ based on the concept of matroid contraction (White and White 1986). For knapsack constraints, the budget for each cost function is decreased by the corresponding cost of $A$ in line 6.

We start with the subroutine, i.e., KNAPSACKGS (Feldman, Harshaw, and Karbasi 2020) as presented in Algorithm 2, which utilizes the density threshold in its simultaneous greedy framework. At a high level, KNAPSACKGS simultaneously maintains $\ell$ disjoint candidate solution sets and inserts an element to one of them at a time by a selection criterion utilizing the density threshold. More concretely, it adds an element to one candidate solution set $S$ when its density, i.e., the marginal gain divided by the sum of the knapsack costs, is not smaller than the density ratio $\rho$ in line 5. Moreover, KNAPSACKGS decreases threshold $\tau$ in line 14 to limit the number of iterations.

Since the density ratio is key to SPROUT, it is essential to derive an appropriate value. Inspired by (Feldman, Harshaw, and Karbasi 2020), SPROUT uses binary search in lines 8–14 to approximate the best density ratio $\rho^*$, where $\beta$ is used to ensure that $\rho^*$ is included in the range of search. In this procedure, each $S_K$ generated by KNAPSACKGS is added to $S_0$, and $E$ in $b_E$ indicates whether the knapsack constraints are violated in line 6 of KNAPSACKGS during the execution, i.e., $E = 0$ if the subroutine never violates them and $E = 1$ otherwise. For each $A$, $S_A$ with the maximal $f$ value is selected from $S_0$ in line 15. Finally, the union of $A$ and its corresponding $S_A$ maximizing $f$ is returned.

**Theoretical Analysis**

We prove in Theorem 1 that SPROUT can achieve an approximation ratio of $(1 + \epsilon)(k + m + 3 + 2\sqrt{m + 1})$ using $\tilde{O}(n^2/\epsilon)$ oracle calls, and this ratio improves to $(1 + \epsilon)(k + m + 1)$ when the objective function is monotone.

The time complexity is mainly measured by the number of oracle calls and arithmetic operations since we evaluate the objective function and the constraints through a value and a membership oracle, respectively. Because the computational cost of arithmetic operations is much less than that of oracle calls in most applications, we focus more on the number of oracle calls. For the ease of presentation, we use $S_{OPT}$ and OPT to denote an optimal solution and its objective value of the problem in Definition 3, respectively. Besides, the solution $S_A \cup A$ obtained by SPROUT is represented by $S$.

The proof of Theorem 1 relies on Lemma 1, which shows the approximation of solutions obtained in the binary search procedure of SPROUT. Our proof is mainly attributed to the analysis in (Feldman, Harshaw, and Karbasi 2020).

**Lemma 1.** In SPROUT, $f(A \cup S_K) \geq \min\{\rho + (1 - 1/C)f(A), \frac{1}{\rho}(1 - 1/\ell - 1/\epsilon)z_A(S_{OPT} - pm) + f(A)\}$ for each generated $\rho$ in line 10 and corresponding $S_K$, where $S_{OPT}$ refers to an optimal solution for the reduced instance, and $p = \max\{\ell - 1, k\}$.

**Proof.** The following analysis includes two parts based on the value of the indicator $E$. When $E = 1$, there is a
candidate solution set $S_i$ and an element $a$ satisfying that $S_i \cup a$ obeys the $k$-matroid constraint and violates the $m$-knapsack constraint, i.e., $S_i \cup a \in T'$ and $c_j(S_i \cup a) > 1$ for some $j \in [m]$. We use $A$ to represent $S_i \cup a$ and show that $z_A(A) > \rho$, despite the infeasibility of $A$. Considering Corollary 54 in (Feldman, Harshaw, and Karbasi 2020), we permute the elements of $A$ in the order of being added to $S_i$, which means $A = \{a_1\}_{j=1}^k$ and $a_k$ is $a$. Let $A_i = \{a_i\}_{j=1}^k$ for $i \in [k]$ and $A_0 = \emptyset$. As a result, we derive that $z_A(A) = \sum_{k=1}^m z_A(a_i|A_{i-1}) \geq \sum_{k=1}^m \rho \sum_{j=1}^m c_j(a_i) = \rho \sum_{j=1}^m c_j(A) > \rho$, since all of the elements in $A$ are added with a density constraint no less than $\rho$, and $A$ violates the $m$-knapsack constraint. By line 16 of Algorithm 2, we know that $z_A(S_K) = \max_{S \in \ell(S_i) \subseteq S_K} z_A(S)$, where $T$ is the number of iterations executed in KNAPSACKS. By line 3 of Algorithm 1, we have $C \cdot z_A(a) \leq f(A)$. Thus, we obtain $f(A \cup S_K) = f(A) + z_A(S_K) \geq f(A) + z_A(S_i) = f(A) + z_A(A \setminus a) \geq f(A) + z_A(A) \geq \rho + (1 - 1/C) f(A)$, where the second inequality is by the submodularity of $z_A$.

When $E = 0$, we apply Corollary 54 in (Feldman, Harshaw, and Karbasi 2020) to KNAPSACKS, which yields $z_A(S_K) = \max_{S \subseteq \ell(S_i) \subseteq S_K} z_A(S) \geq \sum_{i=1}^\ell z_A(S_i) / \ell \geq 1 + \frac{1}{\ell + 1} \left(\sum_{i=1}^\ell z_A(S_i) / \ell - \ell \right)$, where the set $S_0$ refers to an optimal solution for the reduced instance, and $p = \max(\ell, 1, k)$ as stated by Proposition 8 in (Feldman, Harshaw, and Karbasi 2020). Besides, if $\mathbf{z}_A$ is monotone, this conclusion can improve to

$$z_A(S_K) \geq (1 - \epsilon) (z_A(S_0) - \ell - \ell (p + 1)) / (p + 1).$$

Let $S_u$ be a set selected from the disjoint sets $\{S_i\}_{i=1}^\ell$ uniformly at random. Obviously, each element in $S_u$ appears in $S_u$ with probability at most $1 / \ell$. We have $\sum_{i=1}^\ell z_A(S_i) / \ell = \mathbb{E}[z_A(S_i) \cup S_u] \geq (1 - 1 / \ell) z_A(S_0)$, where the inequality holds by Lemma 2.2 in (Buchbinder et al. 2014). Thus, $f(A \cup S_K) = f(A) + z_A(S_K) \geq \frac{1}{\ell + 1} \left((1 - 1 / \ell) z_A(S_0) - \ell \right) f(A)$.

**Theorem 1.** For the problem in Definition 3, when the error parameters $\delta = \epsilon$, and the number $\ell$ of solutions is $P + 1$, SPROUT achieves an approximation ratio of roughly $\frac{1}{\log(1/C)}$ oracle calls and $\tilde{O}(\text{Ppm}(C+1) / \epsilon)$ arithmetic operations, where $P = \max([\sqrt{1 + m}], k)$ and $r$ is the size of $S_0$. $P$.

**Proof.** SPROUT enumerates each feasible set $A$ of size $C$ in line 1, thus it can find a set $A$ with max-value elements in the optimal solution. In the following, we are to show that the subroutine KNAPSACKS can find a corresponding set $S_A$ such that $S_A \cup A$ is good enough.

In particular, we consider the case that $A$ is the subset of the optimal solution $S_{OPT}$, containing the first $C$ elements of $S_{OPT}$ in the greedy ordering with respect to the objective function $f$, inspired by (Badanidiyuru et al. 2020). All elements of $S_{OPT} \setminus A$ are kept in the reduced ground set, because they have a marginal value of at most $f(A) / C$ due to the greedy characteristic during its generation. Thus, $S_{OPT} \setminus A$ is feasible for the reduced instance.

In terms of Lemma 1, we derive $f(S) \geq \min\{\rho + (1 - 1/C) f(A), \rho + (1 - 1/C) \rho, (1 - 1/C) \rho + (1 - 1/C) f(A)\}$. To maximize this lower bound, the ideal density ratio is $\rho^* = (1 - \epsilon) (1 - 1 / \ell - \epsilon) z_A(S_{OPT}) + (p + 1) f(A) / C$, and the corresponding value of the lower bound is $f^*(S) = \frac{(1 - \epsilon) (1 - 1 / \ell - \epsilon) z_A(S_{OPT})}{p + 1 + m (1 - \epsilon)} + \frac{1 - (1 - \epsilon) (1 - 1 / \ell - \epsilon) z_A(S_{OPT}) + (p + 1) f(A) / C}{p + 1 + m (1 - \epsilon)} f(A)$. However, $\rho^*$ is hard to be derived in practice because it contains the item $z_A(S_{OPT})$. Next, based on (Feldman, Harshaw, and Karbasi 2020), we will show that SPROUT can use binary search in lines 8–14 to obtain a good estimation of $\rho^*$. SPROUT sets $\rho = \beta \mathcal{V}(1 + \delta) / ((b_1 + b_0 + 1)/2) + \gamma f(A) / C$, as shown in line 10 of Algorithm 1. By setting the correction parameter $\beta = (1 - \epsilon) (1 - 1 / \ell - \epsilon) / (p + 1 + m (1 - \epsilon))$ and $\gamma = (p + 1) / (p + 1 + m (1 - \epsilon))$, we have $\rho = (1 - \epsilon) (1 - 1 / \ell - \epsilon) \mathcal{V}(1 + \delta) \mathcal{V} + (p + 1) f(A) / C$, where $b = \lfloor (b_1 + b_0 + 1)/2 \rfloor$. Considering the submodularity of $z_A$, we have $\mathcal{V} \leq z_A(S_{OPT}) \leq N' \mathcal{V}$. Thus, we set $b_1 = 1$ and $b_0 = \lceil \log N' / \delta \rceil$, to make the initial search range of $\rho$ contain $\rho^*$. We will show that during the binary search procedure, SPROUT can either get a solution with good approximation guarantee directly or maintain the best density ratio $\rho^*$ in the range of search (i.e., the range of $\rho$ where $b \in [b_1, b_0]$). We consider three cases for $\rho$ and $E$ as follows.

1. There exists one iteration of binary search, such that $\rho \leq \rho^*$ and $E = 0$. According to the analysis of the case $E = 0$ in the proof of Lemma 1, we know that SPROUT has got a solution $S$ satisfying $f(S) \geq \frac{1}{p + 1} ((1 - 1 / \ell - \epsilon) z_A(S_{OPT}) - \rho m) + f(A) \geq \frac{1}{p + 1} ((1 - 1 / \ell - \epsilon) z_A(S_{OPT}) - \rho m) + f(A) = f^*(S)$, where the second inequality holds by $\rho \leq \rho^*$.

2. There exists one iteration of binary search, such that $\rho \geq \rho^*$ and $E = 1$. According to the analysis of the case $E = 1$ in the proof of Lemma 1, we know that SPROUT has got a solution $S$ satisfying $f(S) \geq \rho + (1 - 1/C) f(A) \geq \rho^* + (1 - 1/C) f(A) = f^*(S)$, where the second inequality holds by $\rho \geq \rho^*$.

3. If the above cases have not occurred, then $\rho \leq \rho^*$ implies $E = 1$, which will increase $b_1$ to $\lfloor (b_1 + b_0 + 1)/2 \rfloor$; and $\rho \geq \rho^*$ implies $E = 0$, which will decrease $b_0$ to $\lfloor (b_1 + b_0 + 1)/2 \rfloor$. Thus, $\rho^*$ is always contained in the range of binary search (i.e., the range of $\rho$ where $b \in [b_1, b_0]$), and the final $\rho$ found by SPROUT satisfies $(1 - \delta) \rho^* \leq \rho^*$.
Combining the analyses of the above three cases, we have

\[
f(S) \geq \frac{(1 - \delta)(1 - \epsilon)(1 - 1/\ell - \epsilon)}{p + 1 + m(1 - \epsilon)} z_A(S_{OPT}^+, S_{OPT}^-) + \left(1 - \frac{\delta(p + 1) + m(1 - \epsilon)}{C(p + 1 + m(1 - \epsilon))}\right) f(A).
\]

Because \( S_{OPT}^+ \backslash A \) is feasible for the reduced instance and \( S_{OPT}^- \) is an optimal one, we have \( z_A(S_{OPT}^+) \geq z_A(S_{OPT} \backslash A) = OPT - f(A) \). Furthermore, the greedy choice of \( A \) implies \( f(A) \geq C \cdot OPT / S_{OPT} = C \cdot OPT / r \).

Since \( \delta = \epsilon \) and \( 1 - 1/\ell - \epsilon \geq (1 - 1/\ell)(1 - 2\epsilon) \), we have

\[
f(S) \geq \frac{(1 - \epsilon)^2(1 - 2\epsilon)(1 - 1/\ell)}{p + 1 + m} + \left(1 - \frac{\epsilon(p + 1) + m(1 - \epsilon) + C(1 - \epsilon)^3}{r(p + 1 + m(1 - \epsilon))}\right) OPT.
\]

Thus, SPROUT will return a solution \( S \) satisfying

\[
f(S) \geq \frac{(1 - \epsilon)^2(1 - 2\epsilon)(1 - 1/\ell)}{p + 1 + m} + \frac{(C - \epsilon)(P + 1) + (C - 1)m(1 - \epsilon) - C(1 - \epsilon)^3}{r(P + 1 + m(1 - \epsilon))} \tag{2}
\]

i.e., a roughly \( \left(\frac{1 - \epsilon}{k + m + 3 + 2\sqrt{m + 1}}\right)^{-1} \) approximation ratio. When \( C = 1 \), the approximation ratio becomes \((1 + \epsilon)(k + m + 3 + 2\sqrt{m + 1})\). Moreover, when \( f \) is monotone, SPROUT can provide a \((1 - \epsilon)^{-3}(k + m + 1)\)-approximation solution according to Eq. (1) by setting \( \ell \leq k + 1 \), since the monotonicity of \( f \) leads to the monotonicity of \( z_A \). As the factor \((1 - \epsilon)^3\) is in the order of \( 1 + O(\epsilon) \), this ratio can be rewritten as \((1 + \epsilon)(k + m + 1)\).

We then analyze the time complexity of SPROUT. It is easy to find that SPROUT has at most \( n^C \) iterations. When \( A \) is fixed, we first need to reduce the ground set which costs \( O(n) \) oracle calls and arithmetic operations, and the most-time-consuming part is the calls to the subroutine \( KNAPSACK(G, S) \), each of which requires \( O(|N|^2) / \epsilon \) oracle calls and \( O(m|N|^2) / \epsilon \) arithmetic operations according to Observation 22 in (Feldman, Harshaw, and Karbasi 2020). As the binary search method is employed, the number of calls is \( O(\log (b_0/b_1)) = O(\log (\log |N| / \delta)) = O(1) \). Thus, we can conclude that SPROUT requires \( O(P_{mn}C^+ / \epsilon) \) oracle calls and \( O(P_{mn}C^+ / \epsilon) \) arithmetic operations.

SPROUT++ Algorithm

Though SPROUT can achieve the best approximation guarantee, the exhaustive enumeration may be too time-consuming. Thus, we propose SPROUT++, an accelerated version of SPROUT. In a nutshell, SPROUT++ improves the efficiency by only randomly enumerating good single elements. It also introduces a smooth technique into the binary search procedure to make the algorithm more robust.

Algorithm 3: SPROUT++

**Input:** Objective function \( f : 2^N \to \mathbb{R}_+ \), \( k \) matroids \( \mathcal{M}_i(N, I_i) \) and \( m \) cost functions \( c_i : N \to \mathbb{R}_+ \)

**Parameter:** Error params \( \delta, \epsilon \), correction params \( \beta, \gamma \), acceleration param \( \alpha \), smooth param \( \mu \), counter \( t_e \) and number \( \ell \) of solutions

**Output:** A set \( S \) s.t. \( S \in \bigcap_{i=1}^k I_i \) and \( \forall i \in [m], c_i(S) \leq 1 \)

1. Let \( e^* \) be the feasible element \( e \in N \) maximizing \( f(e) \).
2. while \( t_e > 0 \) do
   3. Randomly select a feasible single-element set \( A \subseteq N \) never being chosen before.
   4. if \( f(A) \geq (1 - \alpha)f(e^*) \) then
      5. \( z_A(S) \geq f(S[A]) \).
      6. \( N^e \triangleq \{ e \in N \vert e \notin A \} \).
      7. \( M_i'(N^e, I_i') \triangleq \) contraction of \( M_i(N, I_i) \) by \( A \).
      8. \( I^e \triangleq \bigcap_{i=1}^k I_i' \).
      9. Decrease knapsack budgets by \( c_i(A) \) and normalize each of them to 1.
     10. Let \( S_0 = \emptyset \), and \( V \) be the maximum \( z_A \) value of a single feasible element in \( N^e \).
     11. Let \( b_1 = 1 \) and \( b_0 = \log |N^e| / \delta \).
     12. while \( |b_1 - b_0| > 1 \) do
        13. \( b = (b_1 + b_0 + 1)/2 \).
        14. \( \rho = \beta \sqrt{(1 + \delta)^b + \gamma f(A)} \).
        15. \( S_K = \text{KNAPSACK}(S_A, N^e, I^e, \{ c_i \}_{i=1}^m, b, \rho, \epsilon) \).
        16. Add \( S_K \) to \( S_0 \).
        17. \( b_E = b + (1 - 2b)(1 - 1/\mu) |b_E - b| \).
     18. end while
     19. \( S_A = \arg \max_{S \in S_0} f(S) \).
     20. \( t_e = t_e - 1 \).
     21. end if
     22. end while
     23. \( A^* = \arg \max_{A \subseteq N \vert A \subseteq N^e} f(A \cup S_A) \) over all feasible \( A \subseteq N \).
     24. return \( A^* \cup S_A \).

As presented in Algorithm 3, SPROUT++ specifies \( C \) as 1 and randomly picks \( t_e \) feasible sets (i.e., feasible single elements) in line 3 instead of enumerating over all possible \( A \). On top of that, SPROUT++ omits the elements with low value in line 4, and does not delete extra elements from the ground set in line 6. Besides, it employs a smooth technique in line 17 in the search procedure, since the search range shrinks so fast that many density ratios corresponding to good solutions may be missed. Technically, we use a parameter \( \mu \) to decide the search range in the next iteration instead of sharply reducing it by half.

In Theorem 2, we prove that SPROUT++ can achieve a similar approximation guarantee to SPROUT with a high probability using much less time (depending on \( t_e \)) under an assumption in Eq. (3). This assumption intuitively means that the objective value of each element in \( S_{OPT} \) is relatively large, which can hold if the marginal gain of adding each element \( e \) to \( S_{OPT} \) is large enough by the submodularity. It often appears in the problems of selecting small subsets from a relatively large ground set, where a small chosen subset may represent only part of the ground set and
there are elements which can still contribute enough. Since SPROUT++ is for acceleration, such requirement naturally meets its large-scale application.

**Theorem 2.** For the problem in Definition 3, suppose that \(v_u \in S_{OPT}, (1 + \alpha) \cdot f(A) \geq f(e^*)\),

\[
(3)
\]

where \(e^*\) is a feasible max-value element in \(N\) and \(\alpha \leq (1 - \epsilon)(p + 1 - 1(1 - \epsilon^2))/\epsilon(p + 1) + m(1 - \epsilon)\). Then SPROUT++ offers an approximation ratio of \((1 + \epsilon)/(k + m + 3 + 2\sqrt{m + 1})\) with probability at least \(1 - e^{-rt_c}/n\) using \(\tilde{O}(\log^{-1}(2\mu/(2\mu - 1))t_cPn/\epsilon)\) oracle calls and \(\tilde{O}(\log^{-1}(2\mu/(2\mu - 1))t_cPmn/\epsilon)\) arithmetic operations, where \(P = \max\{\sqrt{1 + m}, k\}\). \(r\) is the size of \(S_{OPT}\), and \(\mu\) is the smooth parameter.

**Proof.** Compared with SPROUT, we find that the shrinking ratio of the search range is now reduced to \((2\mu - 1)/2\mu\) during the search procedure of SPROUT++, implying that the time complexity is multiplied by a factor of \(\log^{-1}(2\mu/(2\mu - 1))\), \(\mu\) and \(\rho\) to denote the chosen \(t_c\) feasible single-element sets.

Consider that there is \(A \in \{A_1, \ldots, A_t\}\) such that \(A \subseteq S_{OPT}\). According to the proof of Lemma 1, when \(E = 1\), \(f(S) \geq f(A) + z(A) - z(A) \geq f(A) + \rho - \alpha f(A)\), where the last inequality holds by \(z(A) \leq f(e^*)(1 + \epsilon)f(A)\) due to the assumption in Eq. (3); when \(E = 0\), we have \(f(S) \geq 1/(1 - 1/\ell - \epsilon)z(A)(S_{OPT} - \rho m) + f(A)\). Thus, we can conclude that \(f(S) \geq \min(\rho - \alpha f(A), 1/(1 - 1/\ell - \epsilon)z(A)(S_{OPT} - \rho m) + f(A))\). By following the proof of Theorem 1 with \(C = 1\) and \(\rho^* = (1 - \epsilon)(1 - 1/\ell - \epsilon)x_{\alpha}(S_{OPT} + (1 + \alpha)(p + 1))\),

\[
\frac{\rho^* + 1 + m(1 - \epsilon)}{p + 1 + m(1 - \epsilon)}\]

we can derive

\[
f(S) \geq \frac{(1 - \delta)(1 - \epsilon)(1 - 1/\ell - \epsilon)}{p + 1 + m(1 - \epsilon)}z(A(S_{OPT} - \rho m) + f(A).
\]

as \(A \subseteq S_{OPT}\), we still have \(z(A)(S_{OPT} - \rho m) = OPT - f(A)\). Note that in the proof of Theorem 1, \(A\) is the best element in \(S_{OPT}\), and thus \(f(A, OPT)\) is \(OPT/r\); while \(A\) here can be only guaranteed from \(S_{OPT}\), but we can still have \(f(A) \geq f(e^*)(1 + \alpha) \geq OPT/(r(1 + \alpha))\) by the assumption in Eq. (3). Thus,

\[
f(S) \geq \frac{(1 - \delta)(1 - \epsilon)(1 - 1/\ell - \epsilon)}{r(1 + \alpha)}OPT + (1 - \delta)(1 - \epsilon)(1 - 1/\ell - \epsilon)OPT/\rho \\
\]

implying a \((1 + \epsilon)/(k + m + 3 + 2\sqrt{m + 1})\)-approximation ratio, since \(\alpha \leq (1 - \epsilon)(p + 1 - 1(1 - \epsilon^2))/\epsilon(p + 1) + m(1 - \epsilon)\).

We finally estimate the probability of the event \(E_0\) that \(\exists A \in \{A_1, \ldots, A_t\}, A \subseteq S_{OPT}\). Let \(E_0\) denote the complement of \(E_0\). By the selection of \(A_i\) in line 3 of Algorithm 3, we have \(Pr[E_1] = \prod_{i=1}^{\epsilon}Pr[A_i \not\subseteq S_{OPT}] \leq \prod_{i=1}^{\epsilon}1 - r/(n - i + 1) \leq (1 - r)^{\epsilon}/n^\epsilon \leq e^{-rt_c/n}\), implying \(Pr[E_0] \geq 1 - e^{-rt_c/n}\). Thus, the theorem holds. \(\square\)

**Empirical Study**

In this section, we empirically compare SPROUT++ with celebrated algorithms, i.e., the greedy algorithm, DENSITYSEARCHSGS (denoted as DSSGS) (Feldman, Harshaw, and Karbasi 2020), FANTOM (Mirzazoeimein, Badanidiyury, and Karbasi 2016) and RePeated Greedy (denoted as RP_Greedy) (Feldman, Harshaw, and Karbasi 2017), on the applications of movie recommendation and weighted max-cut. As SPROUT++ is randomized, we repeat its run for 10 times independently and report the average and standard deviation. We always the same setting (i.e., \(t_c = n/5, \alpha = 0.5, \mu = 1, \ell = 2, \delta = \epsilon = 0.25, \beta = 5 \times 10^{-4}\), and \(\gamma = 1 \times 10^{-6}\) and perform the sensitivity analysis on \(t_c\) and \(\mu\) to show their influence, and finally compare the performance of SPROUT++ and SPROUT.

**Movie Recommendation.** Movie recommendation is a popular task aiming to pick representative movies. We use the set of 10473 movies from the MovieLens Dataset, where the rating, release year, genres and feature vector (Lindgren, Wu, and Dimakis 2015), of each movie are included. To select diverse movies, a non-monotone submodular objective function (Lin and Bilmes 2011; Simon, Savely, and Seitz 2007) is considered, i.e., \(f(S) = \sum_{i \in N} j \in s_{ij} - \sum_{i \in s_{ij}} j \in n\), where \(s_{ij} = \exp(-\lambda \cdot \text{dist}(v_i, v_j))\) denotes the similarity between movies \(i\) and \(j\) (Badanidiyury et al. 2020). \(\text{dist}(v_i, v_j)\) denotes the Euclidean distance between movie feature vectors \(v_i\) and \(v_j\), and \(\lambda\) is set to 4 in our experiments. As for constraints, we set a uniform matroid for each genre limiting the number of movies in it to 2, and limit the number of chosen movies to 10. Moreover, we define three knapsack constraints. The first one leads to movies with higher ratings and its cost function is \(c_1 = 10 - \text{rating}\) with a budget of 20. The other two aim to pick movies released close to particular years, i.e., 1995 and 1997, hence the cost functions are \(c_2 = |1995 - \text{year}|\) and \(c_3 = |1997 - \text{year}|\) with a budget of 30.

**Weighted Max-cut.** The max-cut problem is one of the most well-known problems in combinatorial optimization (Edwards 1973). It aims to partition the vertices of a graph into two sets so that the number of edges between them is maximized. The weighted max-cut problem considers that each edge is associated with a weight and the goal is maximizing the sum of weights of edges in cut. Formally, we study the non-monotone submodular function \(f(S) = \sum_{u \in S} \sum_{v \in V \setminus S} w_{u,v}\), where \(V\) is the set of vertices and \(w_{u,v}\) is the weight of edge \((u, v)\). Concretely, we use a classic type of random graphs, i.e., Erdos Rény graphs (Erdos, Rényi et al. 1960), where the number of nodes is \(n = 1000\), and the probability of each potential edge to be involved in the graph is set to 0.01. For each edge, we generate its weight uniformly at random from \([0, 1]\). As for constraints, we use a uniform matroid to limit the set size to 10, and set the sum of degree of nodes as its knapsack cost with budget 100. Moreover, we index the nodes and limit the sum of the last digit of nodes to 40 as an extra knapsack constraint.

The results for experiments are shown in Figures 1 and 2, respectively. We have normalized the budgets of knapsacks.
We further study the influence of parameters \( t_c \) and \( \mu \). We observe that SPROUT++ immediately outperforms the runner-up FANTOM; then the objective value returned by SPROUT++ soars with a sharp drop of the standard deviation as the ratio \( t_c/n \) increases, and eventually stabilizes at a high level. Meanwhile, we perform experiments on movie recommendation with three knapsack constraints where the budget is set to 1 and the maximum number of allowed movies is set to 10. By setting \( t_c = 5 \) and varying \( \mu \) from 1 to 5, Figure 3(b) shows that the performance of SPROUT++ raises as \( \mu \) increases, and can catch up with FANTOM, even \( t_c \) is extremely small. These results imply that in many cases SPROUT++ can be more efficient and effective in practice than in theoretical analysis.

Finally, we compare the performance of SPROUT++ and SPROUT. We consider the experiment in Figure 2(a) for examination, and plot their objective value and number of oracle calls in Figure 4. It comes out that SPROUT++ achieves competitive performance to SPROUT using much less oracle calls, showing the efficiency of SPROUT++ in practice.

### Conclusion

This paper considers the submodular maximization problem under the intersection of \( k \)-matroid and \( m \)-knapsack constraints, and proposes the SPROUT algorithm which achieves a polynomial approximation guarantee better than the best known one. Furthermore, we provide an efficient variant of SPROUT, i.e., SPROUT++, which can still achieve a similar approximation guarantee to SPROUT. Experiments on the applications of movie recommendation and weighted max-cut demonstrate the superiority of SPROUT++ over state-of-the-art algorithms.

---

Figure 1: Movie Recommendation. (a) and (b): obj. value vs. knapsack budget; (c) and (d): obj. value vs. maximum #allowed movies.

Figure 2: Weighted Max-cut. Left: obj. value vs. knapsack budget; right: obj. value vs. maximum #allowed nodes.

Figure 3: Parametric sensitivity analysis on \( t_c \) and \( \mu \). Left: obj. value vs. \( t_c/n \) on weighted max-cut; right: obj. value vs. \( \mu \) on movie recommendation.

Figure 4: Comparison of SPROUT/SPROUT++ on experiments in Figure 2(a). Left: obj. value vs. knapsack budget; right: oracle calls vs. knapsack budget.
Acknowledgments

This work was supported by the NSFC (62022039, 62276124), the Jiangsu NSF (BK20201247), and the project of HUAWEI-LAMDA Joint Laboratory of Artificial Intelligence. Chao Qian is the corresponding author.

References


