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; Sipos et al. 2012; Dasgupta, 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 Vondrák (2010) provided a $(k+1+1/(k-1)+\epsilon)$-approximation for the k-matroid by local search requiring $\text{poly}(n)\cdot\exp(k,\epsilon)$ running time, where n is the problem size and $\epsilon > 0$. 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, Vondrák, and Zenklusen (2010) proposed an algorithm which achieves a $(k/0.19+\epsilon)$-approximation for k-matroid and m-knapsack constraints with the time complexity of $\text{poly}(n)\cdot\exp(k,m,\epsilon)$, which, however, may be unacceptable. Mirzasoleiman, Badanidiyuru, and Karbasi (2016) developed the first practical algorithm, FANTOM, offering a $((1+\epsilon)(1+1/k)(2k+2m+1))$-approximation for the intersection of the very general k-system and m-knapsack constraints with $\tilde{O}(n^2/\epsilon)$ oracle calls. For the sake of simplicity, we ignore polylogarithmic factors by using the $\tilde{O}$ 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 $(1+\epsilon)(k+2m)+O(\sqrt{k+m})$ with $\tilde{O}(n/\epsilon)$ oracle calls. The approximation ratio becomes $(1+\epsilon)(k+2m)+O(\sqrt{m})$ for the intersection of k-extendible and m-knapsack constraints and $(1+\epsilon)(k+2m+1)$ 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; Haba 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 cOnstrained sUbmodular maximizaTion algorithm, called

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

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.

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 $((1+\epsilon)(k+m+3+2\sqrt{m+1}))$-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 $\mathcal{N}$, we study the submodular functions $f: 2^{\mathcal{N}} \to \mathbb{R}$, i.e., set functions satisfying the diminishing returns property. Specifically, a set function is submodular if $f(e|A) \geq f(e|B)$, $\forall A \subseteq B \subseteq \mathcal{N}$ and $e \notin B$, where $f(S|A) \triangleq f(S \cup A) - f(A)$ 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 $(\mathcal{N}, \mathcal{I})$ where $\mathcal{I} \subseteq 2^{\mathcal{N}}$, $(\mathcal{N}, \mathcal{I})$ is called an independence system if (1) $\emptyset \in \mathcal{I}$; (2) $\forall A \subseteq B \subseteq \mathcal{N}$, if $B \in \mathcal{I}$ then $A \in \mathcal{I}$. An independence system is called a matroid $\mathcal{M}(\mathcal{N}, \mathcal{I})$ if $\forall A, B \in \mathcal{I}$ and $|A| < |B|$, there is $e \in B \backslash A$ such that $A \cup e \in \mathcal{I}$. We present k-matroid in Definition 1.

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

Besides, given a modular cost function $c: 2^{\mathcal{N}} \to \mathbb{R}$, i.e., $\forall A \subseteq \mathcal{N}, c(A) = \sum_{e \in A} c(e)$, 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 $S \subseteq \mathcal{N}$ 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 $f: 2^{\mathcal{N}} \to \mathbb{R}$, a m-knapsack constraint with cost functions $c_1, c_2, \cdots, c_m$, and a k-matroid $\mathcal{M}(\mathcal{N}, \bigcap_{i=1}^{k} \mathcal{I}_i)$, to find $\arg\max_{S \subseteq \mathcal{N}} f(S)$ 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 $\mathcal{A}$ with a size of C as the first part of solution set, and then selects the set $S_{\mathcal{A}}$ from the remaining elements using the KNAPSACKSGS subroutine as presented in Algorithm 2. Finally, it returns the best solution $\mathcal{A} \cup S_{\mathcal{A}}$ over all feasible $\mathcal{A}$. In the following, we describe in more details about SPROUT. First, we define a new objective function $z_{\mathcal{A}}$ in line 2 for the enumerated feasible set $\mathcal{A} \subseteq \mathcal{N}$ in line 1. Then, a reduced ground set $\mathcal{N}'$ is derived by removing elements belonging to $\mathcal{A}$ or whose value of $z_{\mathcal{A}}$ is larger than $f(\mathcal{A})/C$ from $\mathcal{N}$ in line 3. After selecting the set $\mathcal{A}$, we need

[1] As in (Feldman, Harshaw, and Karbasi 2020), the dependence on k and m is suppressed from the running time.

Algorithm 1: SPROUT

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

Parameter: Error params δ, ϵ, correction params β, γ, enumeration param C and number ℓ of solutions

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

1: **for** each feasible $\mathcal{A} \subseteq \mathcal{N}$ with C elements **do**
2: $z_{\mathcal{A}}(S) \triangleq f(S|\mathcal{A})$.
3: $\mathcal{N}' \triangleq \{e \in \mathcal{N} | e \notin \mathcal{A} \wedge C \cdot z_{\mathcal{A}}(e) \leq f(\mathcal{A})\}$.
4: $\mathcal{M}_i'(\mathcal{N}', \mathcal{I}_i') \triangleq$ contraction of $\mathcal{M}_i(\mathcal{N}, \mathcal{I}_i)$ by $\mathcal{A}$.
5: $\mathcal{I}' \triangleq \bigcap_{i=1}^{k} \mathcal{I}_i'$.
6: Decrease knapsack budgets by $c_i(\mathcal{A})$ and normalize each of them to 1.
7: Let $S_0 = \emptyset$, and $\mathcal{V}$ be the maximum $z_{\mathcal{A}}$ value of a single feasible element in $\mathcal{N}'$.
8: Let $b_1 = 1$ and $b_0 = \lceil \log |\mathcal{N}'|/\delta \rceil$.
9: **while** $|b_1 - b_0| > 1$ **do**
10: $\rho = \beta \mathcal{V}(1+\delta)^{\lfloor (b_1+b_0+1)/2 \rfloor} + \gamma f(\mathcal{A})/C$.
11: $S_K =$ KNAPSACKSGS$(z_{\mathcal{A}}, \mathcal{N}', \mathcal{I}', \{c_i\}_{i=1}^m, \ell, \rho, \epsilon)$.
12: Add S_K to S_0.
13: $b_E = \lfloor (b_1 + b_0 + 1)/2 \rfloor$.
14: **end while**
15: $S_{\mathcal{A}} = \arg\max_{S \in S_0} f(S)$.
16: **end for**
17: $\mathcal{A}^* = \arg\max_{\mathcal{A}} f(\mathcal{A} \cup S_{\mathcal{A}})$ over all feasible $\mathcal{A} \subseteq \mathcal{N}$.
18: **return** $\mathcal{A}^* \cup S_{\mathcal{A}^*}$

Algorithm 2: KNAPSACKSGS: Subroutine of SPROUT (Feldman, Harshaw, and Karbasi 2020)

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

Parameter: Error param ϵ, number ℓ of solutions, density ratio ρ

Output: A set S s.t. $S \in \bigcap_{i=1}^{k} \mathcal{I}_i'$ and $\forall i \in [m], c_i'(S) \leq 1$

1: Let $S_0^i = \emptyset$ for all $i \in [\ell]$ and $\mathcal{N}_0' = \mathcal{N}'$.
2: Let $t = 1$ and $\tau = \mathcal{V}$.
3: **while** $\tau > \epsilon \mathcal{V}/n$ **do**
4: **for** each pair (a, i) with $a \in \mathcal{N}_{t-1}', i \in [\ell]$ **do**
5: **if** $S_{t-1}^i \cup a \in \mathcal{I}'$ and $z(a|S_{t-1}^i) \geq \max\{\tau, \rho \cdot \sum_{j=1}^{m} c_j'(a)\}$ **then**
6: **if** $c_j'(a) \leq 1, \forall j \in [m]$ **then**
7: Let $a_t = a$ and $i_t = i$.
8: For $i' \in [\ell]$, $S_t^{i'} = \begin{cases} S_{t-1}^{i_t} \cup a_t, \text{if } i' = i_t \\ S_{t-1}^{i}, \text{if } i' \neq i_t \end{cases}$.
9: $\mathcal{N}_t' = \mathcal{N}_{t-1}' \backslash a_t$.
10: $t = t + 1$.
11: **end if**
12: **end if**
13: **end for**
14: $\tau = (1 - \epsilon)\tau$.
15: **end while**
16: **return** $\arg\max_{S \in \{S_{t-1}^i\}_{i=1}^{\ell}} z(S)$

to contract matroids from $\mathcal{M}_i(\mathcal{N}, \mathcal{I}_i)$ to $\mathcal{M}_i'(\mathcal{N}', \mathcal{I}_i')$ by $\mathcal{A}$ in line 4, where $\mathcal{I}_i'$ is the set $\{S \subseteq \mathcal{N}' : S \cup \mathcal{A} \in \mathcal{I}_i\}$ 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 $\mathcal{A}$ in line 6.

We start with the subroutine, i.e., KNAPSACKSGS (Feldman, Harshaw, and Karbasi 2020) as presented in Algorithm 2, which utilizes the density threshold in its simultaneous greedy framework. At a high level, KNAPSACKSGS simultaneously maintains ℓ 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 ρ in line 5. Moreover, KNAPSACKSGS decreases threshold τ 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 ρ^*, where β is used to ensure that ρ^* is included in the range of search. In this procedure, each S_K generated by KNAPSACKSGS is added to S_0, and E in b_E indicates whether the knapsack constraints are violated in line 6 of KNAPSACKSGS during the execution, i.e., $E = 0$ if the subroutine never violates them and $E = 1$ otherwise. For each $\mathcal{A}$, $S_{\mathcal{A}}$ with the maximal f value is selected from S_0 in line 15. Finally, the union of $\mathcal{A}$ and its corresponding $S_{\mathcal{A}}$ 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 $S_{\mathcal{A}} \cup \mathcal{A}$ obtained by SPROUT is represented by $\mathcal{S}$.

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, $f(\mathcal{A} \cup S_K) \geq \min\{\rho + (1 - 1/C)f(\mathcal{A}), \frac{1-\epsilon}{p+1}((1 - 1/\ell - \epsilon)z_{\mathcal{A}}(S'_{OPT}) - \rho m) + f(\mathcal{A})\}$ for each generated ρ 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_t^i and an element a satisfying that

$S_t^i \cup a$ obeys the k-matroid constraint and violates the m-knapsack constraint, i.e., $S_t^i \cup a \in \mathcal{I}'$ and $c'_j(S_t^i \cup a) > 1$ for some $j \in [m]$. We use A to represent $S_t^i \cup a$ and show that $z_{\mathcal{A}}(A) > \rho$, despite the infeasibility of A. Considering Corollary 54 in (Feldman, Harshaw, and Karbasi 2020), we permutate the elements of A in the order of being added to S_t^i, which means $A = \{a_i\}_{i=1}^k$ and a_k is a. Let $A_i = \{a_j\}_{j=1}^i$ for $i \in [k]$ and $A_0 = \emptyset$. As a result, we derive that $z_{\mathcal{A}}(A) = \sum_{i=1}^k z_{\mathcal{A}}(a_i|A_{i-1}) \geq \sum_{i=1}^k \rho \sum_{j=1}^m c'_j(a_i) = \rho \sum_{j=1}^m c'_j(A) > \rho$, since all of the elements in A are added with a density ratio no less than ρ, and A violates the m-knapsack constraint. By line 16 of Algorithm 2, we know that $z_{\mathcal{A}}(S_K) = \max_{S \in \{S_T^i\}_{i=1}^\ell} z_{\mathcal{A}}(S)$, where T is the number of iterations executed in KNAPSACKSGS. By line 3 of Algorithm 1, we have $C \cdot z_{\mathcal{A}}(a) \leq f(\mathcal{A})$. Thus, we obtain $f(\mathcal{A} \cup S_K) = f(\mathcal{A}) + z_{\mathcal{A}}(S_K) \geq f(\mathcal{A}) + z_{\mathcal{A}}(S_t^i) = f(\mathcal{A}) + z_{\mathcal{A}}(A \setminus a) \geq f(\mathcal{A}) + z_{\mathcal{A}}(A) - z_{\mathcal{A}}(a) \geq \rho + (1 - 1/C) f(\mathcal{A})$, where the second inequality is by the submodularity of $z_{\mathcal{A}}$.

When $E = 0$, we apply Corollary 54 in (Feldman, Harshaw, and Karbasi 2020) to KNAPSACKSGS, which yields $z_{\mathcal{A}}(S_K) = \max_{S \in \{S_T^i\}_{i=1}^\ell} z_{\mathcal{A}}(S) \geq \sum_{i=1}^\ell z_{\mathcal{A}}(S_T^i)/\ell \geq \frac{1-\epsilon}{p+1} \left(\sum_{i=1}^\ell z_{\mathcal{A}}(S'_{\text{OPT}} \cup S_T^i)/\ell - \epsilon \mathcal{V} - \rho m \right)$, where the set S'_{OPT} 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 $z_{\mathcal{A}}$ is monotone, this conclusion can improve to

$$z_{\mathcal{A}}(S_K) \geq (1 - \epsilon) \left(z_{\mathcal{A}}(S'_{\text{OPT}}) - \epsilon \mathcal{V} - \rho m \right) / (p + 1). \quad (1)$$

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

Theorem 1. *For the problem in Definition 3, when the error parameters $\delta = \epsilon$, and the number ℓ of solutions is $P+1$, SPROUT achieves an approximation ratio of roughly*[2] $\left(\frac{1-\epsilon}{k+m+3+2\sqrt{m+1}} + \frac{(1-\epsilon)C}{r} \right)^{-1}$ *using $\tilde{O}(Pn^{C+1}/\epsilon)$ oracle calls and $\tilde{O}(Pmn^{C+1}/\epsilon)$ arithmetic operations, where $P = \max\{\lceil \sqrt{1+m} \rceil, k\}$ and r is the size of S_{OPT}.*

When $C = 1$, the approximation ratio becomes $(1+\epsilon)(k+m+3+2\sqrt{m+1})$, and the number of oracle calls becomes $\tilde{O}(Pn^2/\epsilon)$. If the objective function f is monotone, the approximation ratio improves to $(1 + \epsilon)(k + m + 1)$.

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

In particular, we consider the case that $\mathcal{A}$ 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 $S_{\text{OPT}} \setminus \mathcal{A}$ are kept in the reduced ground set, because they have a marginal value of at most $f(\mathcal{A})/C$ due to the greedy characteristic during its generation. Thus, $S_{\text{OPT}} \setminus \mathcal{A}$ is feasible for the reduced instance.

In terms of Lemma 1, we derive $f(\mathcal{S}) \geq \min\{\rho + (1 - 1/C) f(\mathcal{A}), \frac{1-\epsilon}{p+1} \left((1 - 1/\ell - \epsilon) z_{\mathcal{A}}(S'_{\text{OPT}}) - \rho m \right) + f(\mathcal{A})\}$. To maximize this lower bound, the ideal density ratio is

$$\rho^* = \frac{(1-\epsilon)(1 - 1/\ell - \epsilon) z_{\mathcal{A}}(S'_{\text{OPT}}) + (p+1) f(\mathcal{A})/C}{p + 1 + m(1-\epsilon)},$$

and the corresponding value of the lower bound is

$$f^*(\mathcal{S}) = \frac{(1-\epsilon)(1 - 1/\ell - \epsilon)}{p + 1 + (1-\epsilon)m} z_{\mathcal{A}}(S'_{\text{OPT}}) + \left(1 - \frac{(1-\epsilon)m}{C\,(p + 1 + (1-\epsilon)m)} \right) f(\mathcal{A}).$$

However, ρ^* is hard to be derived in practice because it contains the item $z_{\mathcal{A}}(S'_{\text{OPT}})$. 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 ρ^*.

SPROUT sets $\rho = \beta \mathcal{V} (1 + \delta)^{\lfloor (b_1 + b_0 + 1)/2 \rfloor} + \gamma f(\mathcal{A})/C$, 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

$$\rho = \frac{(1-\epsilon)(1 - 1/\ell - \epsilon) \mathcal{V} (1 + \delta)^b + (p+1) f(\mathcal{A})/C}{p + 1 + m(1-\epsilon)},$$

where $b = \lfloor (b_1 + b_0 + 1)/2 \rfloor$. Considering the submodularity of $z_{\mathcal{A}}$, we have $\mathcal{V} \leq z_{\mathcal{A}}(S'_{\text{OPT}}) \leq |\mathcal{N}'| \mathcal{V}$. Thus, we set $b_1 = 1$ and $b_0 = \lceil \log |\mathcal{N}'|/\delta \rceil$, to make the initial search range of ρ contain ρ^*. 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 ρ^* in the range of search (i.e., the range of ρ where $b \in [b_1, b_0]$). We consider three cases for ρ 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 generated a solution $\mathcal{S}$ satisfying $f(\mathcal{S}) \geq \frac{1-\epsilon}{p+1} \left((1 - 1/\ell - \epsilon) z_{\mathcal{A}}(S'_{\text{OPT}}) - \rho m \right) + f(\mathcal{A}) \geq \frac{1-\epsilon}{p+1} \left((1 - 1/\ell - \epsilon) z_{\mathcal{A}}(S'_{\text{OPT}}) - \rho^* m \right) + f(\mathcal{A}) = f^*(\mathcal{S})$, 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 generated a solution $\mathcal{S}$ satisfying $f(\mathcal{S}) \geq \rho + (1 - 1/C) f(\mathcal{A}) \geq \rho^* + (1 - 1/C) f(\mathcal{A}) = f^*(\mathcal{S})$, 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, ρ^* is always contained in the range of binary search (i.e., the range of ρ where $b \in [b_1, b_0]$), and the final ρ found by SPROUT satisfies $(1-\delta)\rho^* \leq \rho \leq \rho^*$.

[2] The precise approximation ratio is shown in Eq. (2).

Combining the analyses of the above three cases, we have

$$f(\mathcal{S}) \geq \frac{(1-\delta)(1-\epsilon)(1-1/\ell-\epsilon)}{p+1+m(1-\epsilon)} z_{\mathcal{A}}(S'_{\text{OPT}}) + \left(1 - \frac{\delta(p+1)+m(1-\epsilon)}{C\,(p+1+m(1-\epsilon))}\right) f(\mathcal{A}).$$

Because $S_{\text{OPT}} \backslash \mathcal{A}$ is feasible for the reduced instance and S'_{OPT} is an optimal one, we have $z_{\mathcal{A}}(S'_{\text{OPT}}) \geq z_{\mathcal{A}}(S_{\text{OPT}} \backslash \mathcal{A}) = \text{OPT} - f(\mathcal{A})$. Furthermore, the greedy choice of $\mathcal{A}$ implies $f(\mathcal{A}) \geq C \cdot \text{OPT}/|S_{\text{OPT}}| = C \cdot \text{OPT}/r$. Since $\delta = \epsilon$ and $1 - 1/\ell - \epsilon \geq (1-1/\ell)(1-2\epsilon)$, we have

$$f(\mathcal{S}) \geq \frac{(1-\epsilon)^2(1-2\epsilon)(1-1/\ell)}{p+1+m} \cdot \text{OPT} + \left(\frac{C}{r} - \frac{\epsilon(p+1)+m(1-\epsilon)+C(1-\epsilon)^3}{r\,(p+1+m(1-\epsilon))}\right) \cdot \text{OPT}.$$

Let $Q = (1-1/\ell)/(p+1+m)$, and we have $p = P$ by setting $\ell = P+1$. If $k > \lceil\sqrt{1+m}\rceil$, $Q = (k+m+2+\frac{m+1}{k})^{-1} \geq (k+m+2+\sqrt{m+1})^{-1}$. Otherwise, $Q = (m+2+\lceil\sqrt{1+m}\rceil+\frac{m+1}{\lceil\sqrt{1+m}\rceil})^{-1} \geq (m+3+2\sqrt{m+1})^{-1}$. Thus, SPROUT will return a solution $\mathcal{S}$ satisfying

$$f(\mathcal{S}) \geq \left(\frac{(1-\epsilon)^2(1-2\epsilon)}{k+m+3+2\sqrt{m+1}} + \frac{(C-\epsilon)(P+1)+(C-1)m(1-\epsilon)-C(1-\epsilon)^3}{r\,(P+1+m(1-\epsilon))}\right)\text{OPT}, \tag{2}$$

i.e., a roughly $\left(\frac{1-\epsilon}{k+m+3+2\sqrt{m+1}}+\frac{(1-\epsilon)C}{r}\right)^{-1}$ 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 $z_{\mathcal{A}}$. 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 $\mathcal{A}$ is fixed, we first need to reduce the ground set which costs $O(n)$ oracle calls and arithmetic operations, and the most time-consuming procedure is the calls to the subroutine KNAPSACKSGS, each of which requires $\tilde{O}(\ell|\mathcal{N}'|/\epsilon)$ oracle calls and $\tilde{O}(m\ell|\mathcal{N}'|/\epsilon)$ arithmetic operations according to Observation 22 in (Feldman, Harshaw, and Karbasi 2020). As the binary search method is employed, the number of calls is $O(\log(b_0/b_1)) = O(\log(\log|\mathcal{N}'|/\delta)) = \tilde{O}(1)$. Thus, we can conclude that SPROUT requires $\tilde{O}(Pn^{C+1}/\epsilon)$ oracle calls and $\tilde{O}(Pmn^{C+1}/\epsilon)$ 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 $f : 2^{\mathcal{N}} \to \mathbb{R}_+$, k matroids $\mathcal{M}_i(\mathcal{N}, \mathcal{I}_i)$ and m cost functions $c_i : \mathcal{N} \to \mathbb{R}_+$
Parameter: Error params δ, ϵ, correction params β, γ, acceleration param α, smooth param μ, counter t_c and number ℓ of solutions
Output: A set S s.t. $S \in \bigcap_{i=1}^k \mathcal{I}_i$ and $\forall i \in [m]$, $c_i(S) \leq 1$

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

As presented in Algorithm 3, SPROUT++ specifies C as 1 and randomly picks t_c feasible sets (i.e., feasible single elements) in line 3 instead of enumerating over all possible $\mathcal{A}$. 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 μ 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_c) 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 $S_{\text{OPT}} \backslash e$ 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*

$$\forall a \in S_{OPT}, (1+\alpha)\cdot f(a) \geq f(e^*), \tag{3}$$

where e^* is a feasible max-value element in $\mathcal{N}$ and $\alpha \leq (1-\epsilon)(p+1-(1-\epsilon)^2)/(\epsilon(p+1)+m(1-\epsilon))$. 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 $P=\max\{\lceil\sqrt{1+m}\rceil,k\}$, r is the size of S_{OPT}, and μ 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 $\log^{-1}(2\mu/(2\mu-1))$. In the following, we use $\mathcal{A}_1,\cdots,\mathcal{A}_{t_c}$ to denote the chosen t_c feasible single-element sets.

Consider that there is $\mathcal{A}\in\{\mathcal{A}_1,\cdots,\mathcal{A}_{t_c}\}$ such that $\mathcal{A}\subseteq S_{\text{OPT}}$. According to the proof of Lemma 1, when $E=1$, $f(\mathcal{S})\geq f(\mathcal{A})+z_{\mathcal{A}}(A)-z_{\mathcal{A}}(a)\geq f(\mathcal{A})+\rho-z_{\mathcal{A}}(a)\geq\rho-\alpha f(\mathcal{A})$, where the last inequality holds by $z_{\mathcal{A}}(a)\leq f(e^*)\leq(1+\alpha)f(\mathcal{A})$ due to the assumption in Eq. (3); when $E=0$, we have $f(\mathcal{S})\geq\frac{1-\epsilon}{p+1}((1-1/\ell-\epsilon)z_{\mathcal{A}}(S'_{\text{OPT}})-\rho m)+f(\mathcal{A})$. Thus, we can conclude that $f(\mathcal{S})\geq\min\{\rho-\alpha f(\mathcal{A}),\frac{1-\epsilon}{p+1}((1-1/\ell-\epsilon)z_{\mathcal{A}}(S'_{\text{OPT}})-\rho m)+f(\mathcal{A})\}$. By following the proof of Theorem 1 with $C=1$ and

$$\rho^*=\frac{(1-\epsilon)(1-1/\ell-\epsilon)z_{\mathcal{A}}(S'_{\text{OPT}})+(1+\alpha)(p+1)f(\mathcal{A})}{p+1+m(1-\epsilon)},$$

we can derive

$$f(\mathcal{S})\geq\frac{(1-\delta)(1-\epsilon)(1-1/\ell-\epsilon)}{p+1+m(1-\epsilon)}z_{\mathcal{A}}(S'_{\text{OPT}})+\frac{(1-\delta)(p+1)-\alpha(\delta(p+1)+m(1-\epsilon))}{p+1+m(1-\epsilon)}f(\mathcal{A}).$$

As $\mathcal{A}\subseteq S_{\text{OPT}}$, we still have $z_{\mathcal{A}}(S'_{\text{OPT}})\geq z_{\mathcal{A}}(S_{\text{OPT}}\backslash\mathcal{A})=\text{OPT}-f(\mathcal{A})$. Note that in the proof of Theorem 1, $\mathcal{A}$ is the best element in S_{OPT}, and thus $f(\mathcal{A})\geq\text{OPT}/|S_{\text{OPT}}|=\text{OPT}/r$; while $\mathcal{A}$ here can be only guaranteed from S_{OPT}, but we can still have $f(\mathcal{A})\geq f(e^*)/(1+\alpha)\geq\text{OPT}/(r(1+\alpha))$ by the assumption in Eq. (3). Thus,

$$f(\mathcal{S})\geq\frac{(1-\epsilon)^2(1-2\epsilon)(1-1/\ell)}{p+1+m}\cdot\text{OPT}+\frac{(1-\epsilon)(p+1)-(1-\epsilon)^3-\alpha(\epsilon(p+1)+m(1-\epsilon))}{r(1+\alpha)(p+1+m(1-\epsilon))}\cdot\text{OPT},$$

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

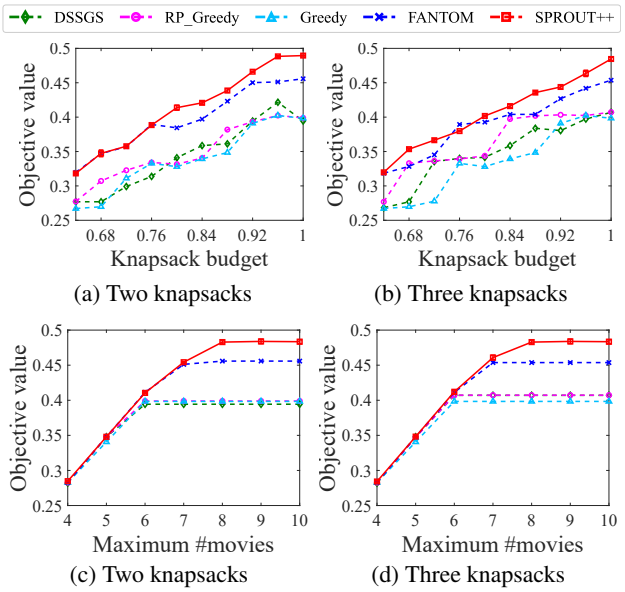
We finally estimate the probability of the event $\mathcal{E}_0$ that $\exists\mathcal{A}\in\{\mathcal{A}_1,\cdots,\mathcal{A}_{t_c}\},\mathcal{A}\subseteq S_{\text{OPT}}$. Let $\mathcal{E}_1$ denote the complement of $\mathcal{E}_0$. By the selection of $\mathcal{A}_i$ in line 3 of Algorithm 3, we have $\Pr[\mathcal{E}_1]=\prod_{i=1}^{t_c}\Pr[\mathcal{A}_i\nsubseteq S_{\text{OPT}}]\leq\prod_{i=1}^{t_c}(1-r/(n-i+1))\leq(1-r/n)^{t_c}\leq e^{-rt_c/n}$, implying $\Pr[\mathcal{E}_0]\geq1-e^{-rt_c/n}$. Thus, the theorem holds. □

Empirical Study

In this section, we empirically compare SPROUT++ with various celebrated algorithms, i.e., the greedy algorithm, DENSITYSEARCHSGS (denoted as DSSGS) (Feldman, Harshaw, and Karbasi 2020), and FANTOM (Mirzasoleiman, Badanidiyuru, and Karbasi 2016), on the applications of movie recommendation and weighted max-cut. We also compare RePeated Greedy (denoted as RP_Greedy), a popular algorithm proposed in (Feldman, Harshaw, and Karbasi 2017). As SPROUT++ is randomized, we repeat its run for 10 times independently, and report the average results and standard deviation. For the sake of fairness, we use the same setting on both problems, i.e., $t_c=n/5$, $\alpha=0.5$, $\mu=1$, $\ell=2$, $\delta=\epsilon=0.25$, $\beta=5\times10^{-4}$, and $\gamma=1\times10^{-6}$. We also perform the sensitivity analysis about the parameters t_c and μ to show their influence, and compare the performance of SPROUT++ and SPROUT in the end.

Movie Recommendation. In daily life, 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, Snavely, and Seitz 2007) is considered, i.e., $f(S)=\left(\sum_{i\in\mathcal{N}}\sum_{j\in S}s_{i,j}-\sum_{i\in S}\sum_{j\in S}s_{i,j}\right)/n$, where $s_{i,j}=\exp(-\lambda\cdot dist(\boldsymbol{v}_i,\boldsymbol{v}_j))$ is the similarity between movies i and j (Badanidiyuru et al. 2020). $dist(\boldsymbol{v}_i,\boldsymbol{v}_j)$ denotes the Euclidean distance between movie feature vectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, and λ is set to 4 in our experiments. As for constraints, we set a uniform matroid for each genre limiting the number of movies in this genre to 2, and limit the total 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 $c_1=10-rating$ with a budget of 20. The other two knapsacks are designed to pick movies released close to particular years, i.e., 1995 and 1997, hence the cost functions are $c_2=|1995-year|$ and $c_3=|1997-year|$ 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 such that the number of edges between them is maximized. The weighted max-cut problem considers the more general case, i.e., each edge is associated with a weight and the goal is to maximize the sum of weights of edges in cut. Formally, we study the non-monotone submodular function $f(S)=\sum_{u\in S}\sum_{v\in V\backslash S}w_{u,v}$, where V is the set of vertices and $w_{u,v}$ is the weight of edge (u,v). In the experiments, we use a classic type of random graphs, i.e., Erdos Rény graphs (Erdos, Rényi et al. 1960), where the number of nodes is set to $n=1000$, 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 chosen set size to 10, and set the sum of degree of nodes as its knapsack cost with budget 100. Moreover, we



(a) Two knapsacks (b) Three knapsacks

(c) Two knapsacks (d) Three knapsacks

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

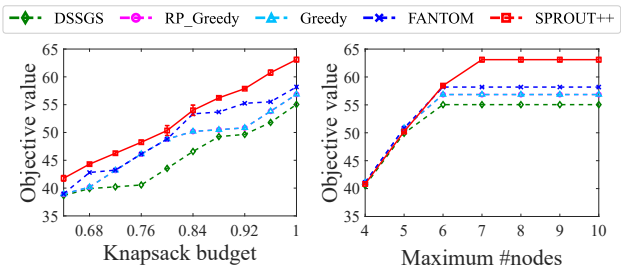


Figure 2: Weighted Max-cut. Left subfigure: obj. value vs. knapsack budget; right subfigure: obj. value vs. maximum number of allowed nodes.

index the nodes and limit the sum of the last digit of nodes to 40 as an extra knapsack constraint.

The results for movie recommendation and weighted max-cut are shown in Figures 1 and 2, respectively. We have normalized the budgets of knapsacks to 1, and then vary them from 0.64 to 1, as shown in Figures 1(a), 1(b) and 2(a). We also vary the maximum number of allowed movies or nodes from 4 to 10 in Figures 1(c), 1(d) and 2(b). For movie recommendation, Figures 1(a) and 1(c) consider two knapsacks c_1 and c_2, while the others consider an extra knapsack constraint c_3. These results show that SPROUT++ outperforms all the celebrated algorithms in respect of the quality of chosen movies and graph-cut. Actually, the generally better performance of RP_Greedy over DSSGS is consistent with the empirical observation in (Feldman, Harshaw, and Karbasi 2020), i.e., RP_Greedy is often better suited for applications than DENSITYSEARCHSGS though the latter algorithm has better worst-case approximation guarantee.

We further study the influence of parameters t_c and μ empirically. As shown in Figures 3(a) and 3(b), we select FANTOM as the baseline and plot the curve of objective value over the ratio of t_c to n and the value of μ, respectively.

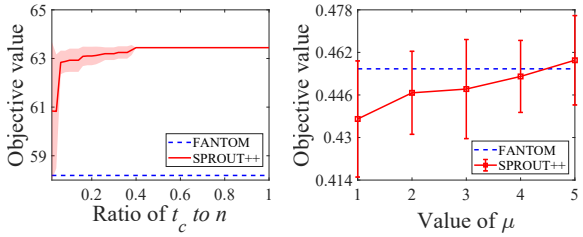


Figure 3: Parametric sensitivity analysis on t_c and μ. Left subfigure: obj. value vs. t_c/n on weighted max-cut; right subfigure: obj. value vs. μ on movie recommendation.

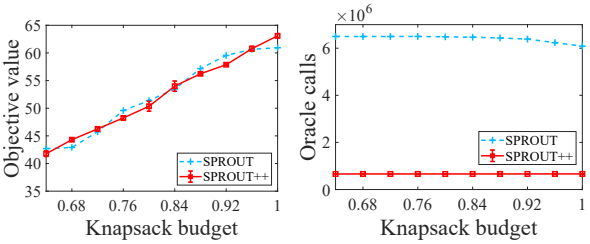


Figure 4: Comparison of SPROUT/SPROUT++ on experiments in Figure 2(a). Left subfigure: obj. value vs. knapsack budget; right subfigure: oracle calls vs. knapsack budget.

Concretely, in Figure 3(a), we test on the max-cut problem where the budget is set to 1 and the maximum number of allowed nodes is set to 10, and vary the ratio of t_c to n from 0.02 to 1. We observe that SPROUT++ immediately overwhelms 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 μ from 1 to 5, Figure 3(b) shows that the performance of SPROUT++ raises as μ 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.

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

Badanidiyuru, A.; Karbasi, A.; Kazemi, E.; and Vondrák, J. 2020. Submodular maximization through barrier functions. In *Advances in Neural Information Processing Systems 33 (NeurIPS'20)*, 524–534. Virtual.

Buchbinder, N.; Feldman, M.; Naor, J.; and Schwartz, R. 2014. Submodular maximization with cardinality constraints. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, 1433–1452. Portland, OR.

Chekuri, C.; Vondrák, J.; and Zenklusen, R. 2010. Dependent randomized rounding via exchange properties of combinatorial structures. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, 575–584. Las Vegas, NV.

Das, A.; and Kempe, D. 2011. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, 1057–1064. Bellevue, WA.

Dasgupta, A.; Kumar, R.; and Ravi, S. 2013. Summarization through submodularity and dispersion. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL'13)*, 1014–1022. Sofia, Bulgaria.

De, A.; Koley, P.; Ganguly, N.; and Gomez-Rodriguez, M. 2020. Regression under human assistance. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI'20)*, 2611–2620. New York, NY.

Delong, A.; Veksler, O.; Osokin, A.; and Boykov, Y. 2012. Minimizing sparse high-order energies by submodular vertex-cover. In *Advances in Neural Information Processing Systems 25 (NeurIPS'12)*, 971–979. Lake Tahoe, NV.

Edwards, C. S. 1973. Some extremal properties of bipartite subgraphs. *Canadian Journal of Mathematics*, 25(3): 475–485.

Erdos, P.; Rényi, A.; et al. 1960. On the evolution of random graphs. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1): 17–60.

Feldman, M.; Harshaw, C.; and Karbasi, A. 2017. Greed is good: Near-optimal submodular maximization via greedy optimization. In *Proceedings of the 30th Conference on Learning Theory (COLT'17)*, 758–784. Amsterdam, The Netherlands.

Feldman, M.; Harshaw, C.; and Karbasi, A. 2020. How do you want your greedy: Simultaneous or repeated? *arXiv:2009.13998*.

Feldman, M.; Karbasi, A.; and Kazemi, E. 2018. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems 31 (NeurIPS'18)*, 730–740. Montreal, Canada.

Fisher, M. L.; Nemhauser, G. L.; and Wolsey, L. A. 1978. An analysis of approximations for maximizing submodular set functions—II. *Polyhedral Combinatorics*, 73–87.

Gygli, M.; Grabner, H.; and Gool, L. V. 2015. Video summarization by learning submodular mixtures of objectives. In *Proceedings of the IEEE 28th Conference on Computer Vision and Pattern Recognition (CVPR'15)*, 3090–3098. Boston, MA.

Haba, R.; Kazemi, E.; Feldman, M.; and Karbasi, A. 2020. Streaming submodular maximization under a k-set system constraint. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, 3939–3949. Virtual.

Kempe, D.; Kleinberg, J.; and Tardos, É. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, 137–146. Washington, DC.

Krause, A.; and Guestrin, C. 2005. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI'05)*, 324–331. Edinburgh, Scotland.

Lee, J.; Sviridenko, M.; and Vondrák, J. 2010. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4): 795–806.

Lin, H.; and Bilmes, J. 2011. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL'11)*, 510–520. Portland, OR.

Lindgren, E. M.; Wu, S.; and Dimakis, A. G. 2015. Sparse and greedy: Sparsifying submodular facility location problems. In *NIPS Workshop on Optimization for Machine Learning*. Montreal, Canada.

Liu, D.-X.; Mu, X.; and Qian, C. 2023. Human assisted learning by evolutionary multi-objective optimization. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI'23)*, to appear. Washington, DC.

Liu, S.; Lu, N.; Chen, C.; and Tang, K. 2021. Efficient combinatorial optimization for word-level adversarial textual attack. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30: 98–111.

Mirzasoleiman, B.; Badanidiyuru, A.; and Karbasi, A. 2016. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33rd International Conference on Machine Learning (ICML'16)*, 1358–1367. New York City, NY.

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

Qian, C.; Shi, J.-C.; Tang, K.; and Zhou, Z.-H. 2018. Constrained monotone k-submodular function maximization using multiobjective evolutionary algorithms with theoretical guarantee. *IEEE Transactions on Evolutionary Computation*, 22(4): 595–608.

Qian, C.; Yu, Y.; and Zhou, Z.-H. 2015. Subset selection by Pareto optimization. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)*, 1765–1773. Montreal, Canada.

Simon, I.; Snavely, N.; and Seitz, S. M. 2007. Scene summarization for online image collections. In *Proceedings of the IEEE 11th International Conference on Computer Vision (ICCV'07)*, 1–8. Rio de Janeiro, Brazil.

Sipos, R.; Swaminathan, A.; Shivaswamy, P.; and Joachims, T. 2012. Temporal corpus summarization using submodular word coverage. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*, 754–763. Maui, HI.

White, N.; and White, N. M. 1986. *Theory of Matroids*. 26. Cambridge University Press.