http://lamda.nju.edu.cn

Submodular Maximization Under the Intersection of Matroid and Knapsack Constraints

Yu-Ran Gu, Chao Bian, Chao Qian
Email: guyr@lamda.nju.edu.cn

LAMDA Group, Nanjing University, China

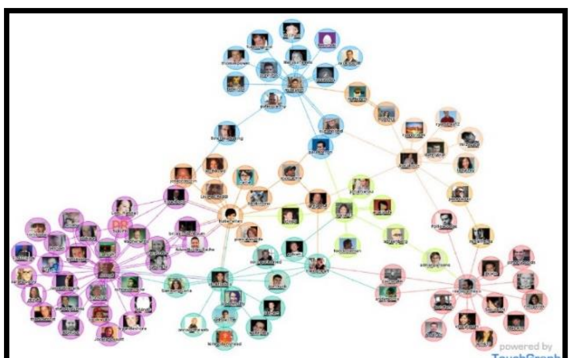
Submodular maximization problem (SMP)

The problem: given a finite set $\mathcal{N}$ and a **submodular** objective function $f: 2^{\mathcal{N}} \to \mathrm{R}^{+}$, to find $arg\ max_{S \subseteq \mathcal{N}}\ f(S)$.

$$\forall X \subseteq Y, v \notin Y: f(X \cup \{v\}) - f(X) \geq f(Y \cup \{v\}) - f(Y)$$

Many applications:

- Movie Recommendation [Mirzasoleiman et al., ICML'16] [Badanidiyuru et al., NeurIPS'20]
- Data summarization [Lin et al., ACL'11] [Sipos et al., CIKM'12] [Dasgupta et al., ACL'13]
- Influence maximization [Kempe et al., KDD'03]



Influence maximization

NP-hard in general!

Common constraints in SMP

Independent system: 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 $A \in \mathcal{I}$.

Matroid Constraint: 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 $e \in B \backslash \mathrm{A}$ such that $A \cup e \in \mathcal{I}$. A set $S \subseteq \mathcal{N}$ satisfies the matroid constraint if and only if $S \in \mathcal{I}$.

Knapsack Constraint: given a modular cost function c, a set $S \subseteq \mathcal{N}$ satisfies the knapsack constraint if and only if $c(S) \leq 1$.

SMP under matroid and knapsack constraints

The problem studied: given a finite set $\mathcal{N}$ and a submodular objective function $f: 2^{\mathcal{N}} \rightarrow \mathrm{R}^+$, a m-knapsack constraint with cost functions $c_1, \cdots, c_m$, and a k-matroid $\mathcal{M}(\mathcal{N}, \cap_{i=1}^{k} \mathcal{I}_i)$, to find $arg\ max_{S \subseteq \mathcal{N}}\ f(S)$ such that $S \in \cap_{i=1}^{k} \mathcal{I}_i$ and $\forall i \in [m], c_i(S) \leq 1$.

Comparison of the state-of-the-art algorithms:

Algorithm	Approximation	Running Time
FANTOM [Mirzasoleiman et al., ICML'16]	$(1+\epsilon)(2k+(2+2/k)m + O(1)$	$\tilde{O}(n^2/\epsilon)$
DENSITYSEARCHSGS [Feldman et al., arXiv'20]	$(1+\epsilon)(k+2m)+O(\sqrt{m})$	$\tilde{O}(n/\epsilon)$
SPROUT(**This Paper**)	$(1+\epsilon)(k+m)+O(\sqrt{m})$	$\tilde{O}(n^2/\epsilon)$

Our algorithm: 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 $\forall i \in [m], c_i(S) \leq 1$

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

Partial Enumeration By Reducing Problem Instance:
To be more robust in practice and achieve better guarantee

Incorporating a **partial enumeration technique** [Badanidiyuru et al., NeurIPS'20] into the **simultaneous greedy framework** [Feldman et al., arXiv'20]

Subroutine from [Feldman et al., arXiv'20]

Indicator-based Binary Search:
To find solution with approximately best guarantee

Indicates whether the knapsack constraints are violated in line 6 of KNAPSACKSGS during the execution

Theoretical analysis

Theorem 1. SPROUT achieves an approximation ratio of **roughly** $\left(\frac{1-\epsilon}{k+m+3+2\sqrt{m+1}}+\frac{(1-\epsilon)C}{r}\right)^{-1}$ **using $\tilde{O}(\frac{Pn^{C+1}}{\epsilon})$ oracle calls and $\tilde{O}(\frac{Pmn^{C+1}}{\epsilon})$ arithmetic operations**, where $P = \{\lceil\sqrt{1+m}\rceil, k\}$ and r is the size of S_{OPT}.

Time complexity:
Composed by arithmetic operations and oracle calls

Computational cost of arithmetic operations is much less than that of oracle calls!

Lemma 1. In SPROUT, $f(\mathcal{A} \cup S_K) \geq \min\{\rho + \left(1-\frac{1}{C}\right) f(\mathcal{A}), \frac{(1-\epsilon)}{p+1}\left(\left(1-\frac{1}{\ell}-\epsilon\right) \mathcal{Z}_{\mathcal{A}}(S'_{OPT}) - \rho m\right) + f(\mathcal{A})\}$ 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 basic idea of Theorem 1:

Using Lemma 1 ➢ Proved based on the value of E

⇩

$$f(\mathcal{S}) \geq \min\{\rho + \left(1-\frac{1}{C}\right)f(\mathcal{A}), \frac{(1-\epsilon)}{p+1}\left(\left(1-\frac{1}{\ell}-\epsilon\right)\mathcal{Z}_{\mathcal{A}}(S'_{OPT}) - \rho m\right) + f(\mathcal{A})\}$$

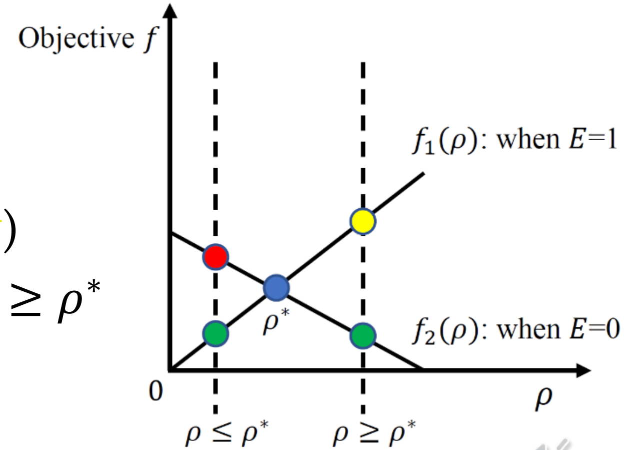
Best density ratio ρ^* ⇩

$$\rho^* = \frac{(1-\epsilon)\left(1-\frac{1}{\ell}-\epsilon\right)\mathcal{Z}_{\mathcal{A}}(S'_{OPT}) + \frac{(p+1)f(\mathcal{A})}{C}}{p+1+m(1-\epsilon)}$$

To find ρ' $s.t.$ $(1-\delta)\rho^* \leq \rho' \leq \rho^*$ ⇩

Consider all three cases for ρ and E in search

1) In one iteration of search, $\rho \leq \rho^*$ and $E = 0$.(**red**)
2) In one iteration of search, $\rho \geq \rho^*$ and $E = 1$.(**yellow**)
3) $\rho \leq \rho^*$ implies E = 1, then increase b_1 to $\lfloor\frac{b1+b0+1}{2}\rfloor$; $\rho \geq \rho^*$ implies $E = 0$, then decrease b_0 to $\lfloor\frac{b_1+b_0+1}{2}\rfloor$. (**green**)



Our algorithm: 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 $\forall i \in [m]$, $c_i(S) \leq 1$

```
Let e* be the feasible element e ∈ N maximizing f(e).
while t_c > 0 do
   Randomly select a feasible single-element set A ⊆ N
   never being chosen before.
   if f(A) ≥ (1 − α) f(e*) then
      z_A(S) ≜ f(S|A).
      N' ≜ {e ∈ N | e ∉ A}
      M'_i(N', I'_i) ≜ contraction of M_i(N, I_i) by A.
      I' ≜ ⋂_{i=1}^k I'_i.
      Decrease knapsack budgets by c_i(A) and
      normalize each of them to 1.
      Let S_0 = ∅, and V be the maximum z_A value of
      a single feasible element in N'.
      Let b_1 = 1 and b_0 = ⌈log |N'|/δ⌉.
      while |b_1 − b_0| > 1 do
         b = ⌊(b_1 + b_0 + 1)/2⌋.
         ρ = βV(1 + δ)^b + γ f(A).
         S_K = KNAPSACKSGS(z_A, N', I', {c_i}_{i=1}^m, ℓ, ρ, ε).
         Add S_K to S_0.
         b_E = b + (1 − 2E)(1 − 1/μ)|b_E − b|.
      end while
      S_A = arg max_{S∈S_0} f(S).
      t_c = t_c − 1.
   end if
end while
A* = arg max_A f(A ∪ S_A) over all feasible A ⊆ N.
return A* ∪ S_{A*}
```

Random Sampling and Threshold Filtering:
To get more valuable elements more efficiently

Delete No Extra Elements:
To maintain high-quality elements

Aim to BE MORE EFFICIENT!!!

Smooth Technique:
To avoid search range shrinking so fast that many good solutions may be missed

Smooth parameter

Theoretical analysis

Theorem 2. Suppose that $\forall a \in S_{OPT}, (1+\alpha)f(a) \geq f(e^*)$, where e^* is a feasible max-value element in $\mathcal{N}$ and $\alpha \leq \frac{(1-\epsilon)(p+1-(1-\epsilon)^2)}{\epsilon(p+1)+m(1-\epsilon)}$. SPROUT++ offers an approximation ratio of $\boldsymbol{(1+\epsilon)(k+m+3+2\sqrt{m+1})}$ with probability at least $1-e^{\frac{-rt_C}{n}}$ **using $\tilde{O}(\frac{\log^{-1}\{\frac{2\mu}{2\mu-1}\}t_C Pn}{\epsilon})$ oracle calls and $\tilde{O}(\frac{\log^{-1}\{\frac{2\mu}{2\mu-1}\}t_C Pmn}{\epsilon})$ arithmetic operations.**

Proof can be easy!
Just consider the event that any element in S_{OPT} is sampled by SPROUT++!

SPROUT++ can achieve a similar guarantee to SPROUT with a high probability using much less time (depending on tc) under an assumption!

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 $S_{OPT}\backslash e$ is large enough by the submodularity, e.g., selecting small subsets from a relatively large set.

Experiment – movie recommendation

- Movie recommendation [Mirzasoleiman et al., ICML'16] : select a subset of representative movies from MovieLens Dataset.

Formally stated: given a set $\mathcal{N}$ with n movies and a non-monotone submodular objective function $f(S) = \frac{\sum_{i\in\mathcal{N}}\sum_{j\in S} s_{i,j} - \sum_{i\in S}\sum_{j\in S} s_{i,j}}{n}$ [Lin et al., ACL'11], where $s_{i,j} = \exp(-\lambda \cdot dist(v_i, v_j))$ is the similarity between movies i and j [Badanidiyuru et al., NeurIPS'20],

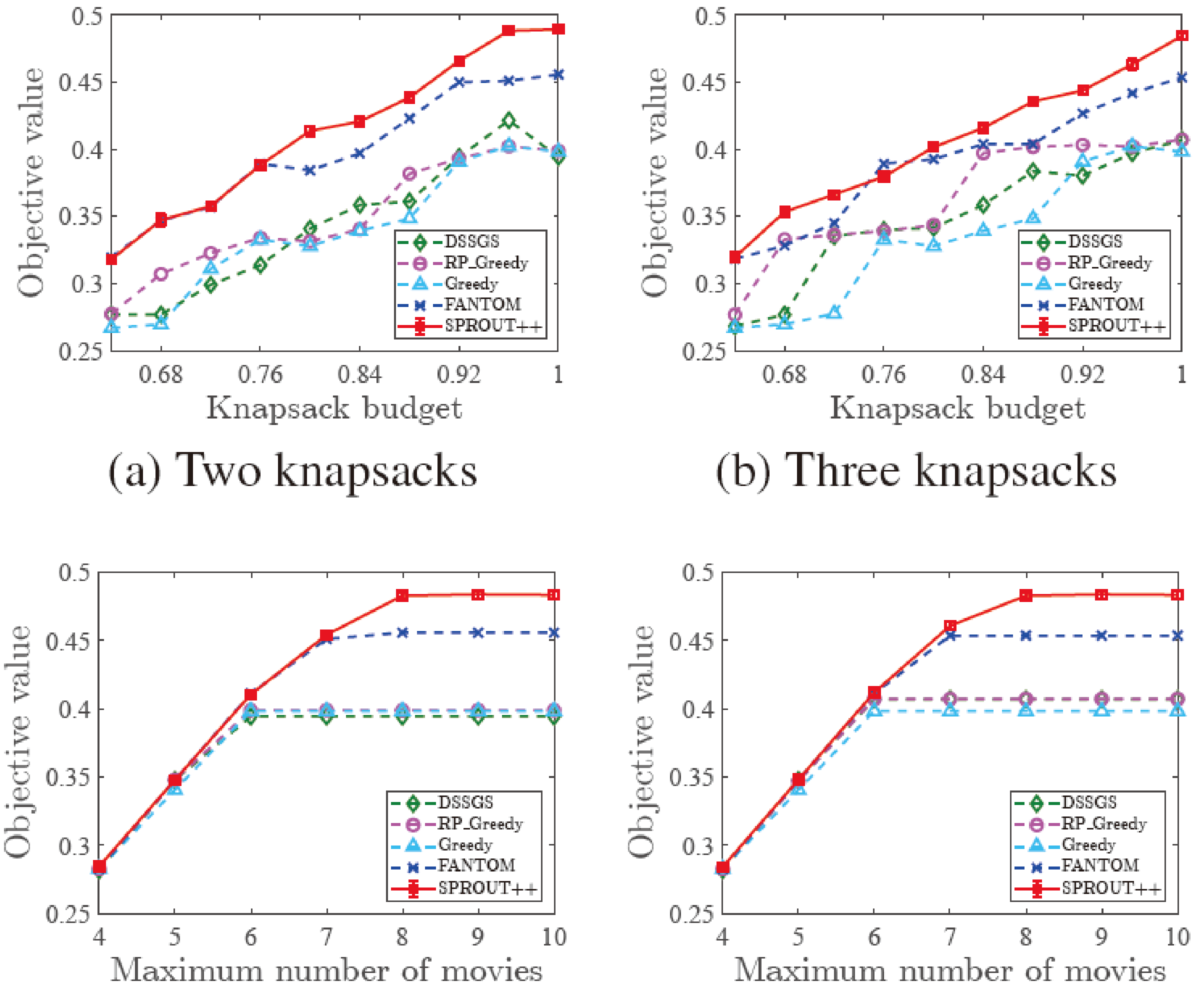
Euclidean distance

$$max_{S\subseteq\mathcal{N}}\ f(S)$$

1. **Knapsack constraints(cost functions):**
   1. **c1 = 10–rating (Budget = 20)**
   2. **c2 = |1995 – year| (Budget = 30)**
   3. **c3 = |1997 – year| (Budget = 30)**
2. **Matroid constraints:**
   1. **a uniform matroid for each genre: limiting the number of movies in this genre to 2**
   2. **a uniform matroid limits the total number of chosen movies to 10**

Constraints

Experiment – movie recommendation



(a) Two knapsacks
(b) Three knapsacks
(c) Two knapsacks
(d) Three knapsacks

Subfigures (a) and (b): obj. value vs. knapsack budget

Subfigures (c) and (d): obj. value vs. maximum number of allowed movies

SPROUT++ is always the best!

Experiment – weighted max-cut

- Weighted max-cut [haba et al., ICML'20] : select a subset of nodes from graph to maximize the max-cut between chosen/unchosen nodes.

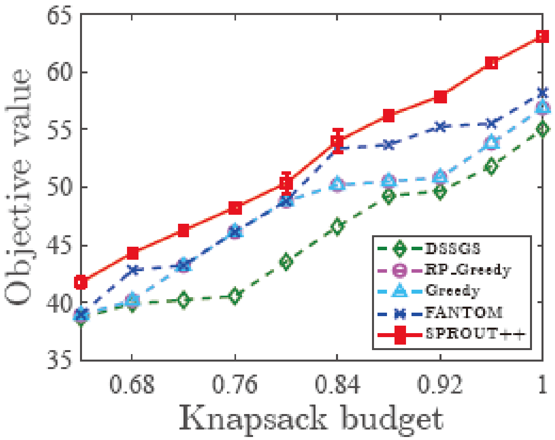
- **Formally stated:** given a node set V and a non-monotone submodular objective function $f(S) = \sum_{u \in S} \sum_{v \in V \setminus S} w_{u,v}$ [haba et al., ICML'11], where V is the set of vertices and $w_{u,v}$ is the weight of edge (u, v),

$$max_{S \subseteq V} \ f(S)$$

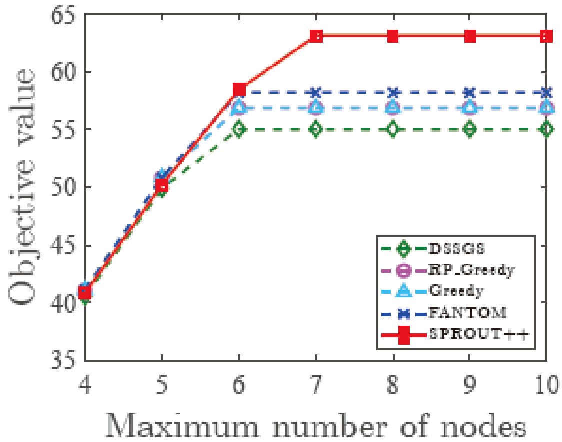
Constraints

1. **Knapsack constraints(cost functions):**
   1. **sum of degree of nodes (Budget = 100)**
   2. **index the nodes and limit the sum of the last digit of nodes (Budget = 40)**
2. **Matroid constraints:**
   1. **a uniform matroid to limit the chosen set size to 10**

Experiment – weighted max-cut



(a)



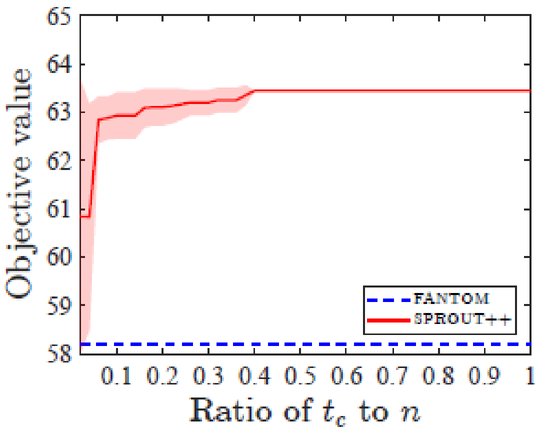
(b)

SPROUT++ is always the best!

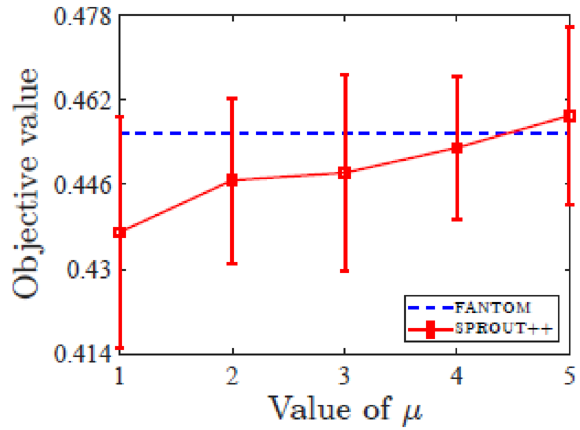
Subfigure (a): obj. value vs. knapsack budget

Subfigure (b): obj. value vs. maximum number of allowed nodes

Experiment – parametric sensitivity analysis



(a)



(b)

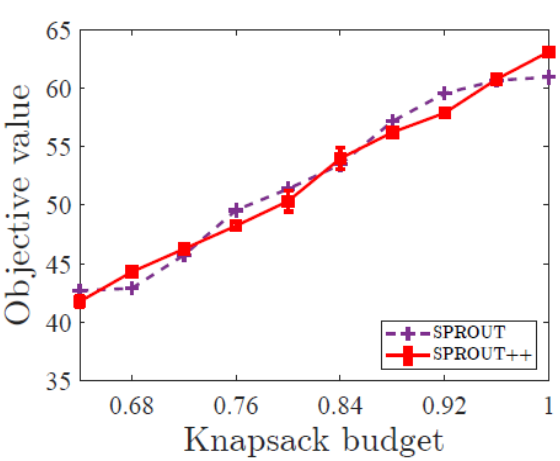
Compared with sub-optimal FANTOM

Subfigure (a): obj. value vs. t_c/n on weighted max-cut

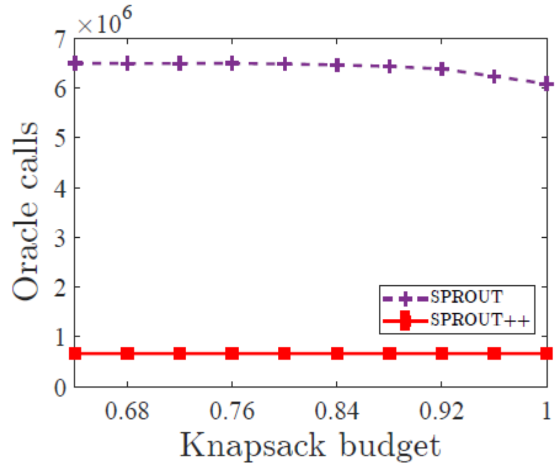
Subfigure (b): obj. value vs. μ on movie recommendation

In SPROUT++, t_c and μ are useful for efficiency!

Experiment – comparison of SPROUTs



(a)



(b)

Compare our propose algorithms

Subfigure (a): obj. value vs. knapsack budget

Subfigure (b): oracle calls vs. knapsack budget

SPROUT++ can compete with SPROUT while being much faster!

Conclusion

- Submodular maximization under knapsack and matroid constraints

$$max_{S \subseteq \mathcal{N}} \ f(S) \quad s.t. \quad c_i(S) \leq 1 \ \forall i \in [m] \ and \ S \in \cap_{j=1}^{k} \mathcal{I}_i$$

Non-monotone Knapsacks Matroids

- **Propose SPROUT algorithm with the SOTA approximation ratio**
- **Propose SPROUT++ algorithm to improve the efficiency of SPROUT**
- **Demonstrate the superior performance of SPROUT & SPROUT++ in practice by extensive experiments**

Thanks