# Data-Dependent Evaluations for Budgeted Submodular Maximization

**Lejian Zhang**
College of Computing and Data Science, Nanyang Technological University, Singapore

**Xueyan Tang**
College of Computing and Data Science, Nanyang Technological University, Singapore

**Jing Tang**
Data Science and Analytics Thrust, The Hong Kong University of Science and Technology (Guangzhou), China

## Abstract

Submodular maximization is an important building block for developing algorithms in many areas such as machine learning and data mining. Due to the NP-hardness of the problem, analysis of submodular maximization algorithms typically provides pessimistic worst-case approximation factors only. It is not easy to evaluate how close a produced solution is to an optimal one for a given problem instance. In this paper, we develop new data-dependent upper bounds for submodular maximization with a knapsack constraint. We theoretically prove that they dominate the optimal solution and empirically demonstrate their advantages in certifying how close to optimal a solution is through experiments with real-world datasets.

**2012 ACM Subject Classification** Mathematics of computing → Submodular optimization and polymatroids; Theory of computation → Approximation algorithms analysis

**Keywords and phrases** submodular maximization, knapsack constraint, approximation guarantee

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2026.4

**Related Version** *Extended Version*: https://arxiv.org/abs/2607.05759

**Funding** This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award MOE-T2EP20122-0007). Jing Tang's work is also partially supported by National Key R&D Program of China under Grant No. 2024YFA1012700, by the National Natural Science Foundation of China (NSFC) under Grant No. 62402410, and by Guangdong Provincial Project (No. 2023QN10X025).

## 1 Introduction

A great number of optimization problems in different areas can be modeled as submodular maximization problems, including facility location [21], feature selection [8], recommendation system [18], influence maximization [26], document summarization [20], maximum coverage [10], sensor placement [16], exemplar sampling [12] and market expansion [9].

Due to the NP-hard nature of submodular maximization, many approximation algorithms have been developed, attempting to find good solutions in polynomial time. Approximation algorithms are typically evaluated by the *approximation factor*, which refers to the worst-case ratio between the function value of an output solution and that of an optimal solution. Nemhauser and Wolsey [22] introduced a simple greedy algorithm which guarantees an approximation factor of $1-1/e$ for the problem of Monotone Submodular Maximization with a Cardinality constraint (MSMC) (finding a set $S$ of a given size which maximizes $f(S)$). But the greedy algorithm does not guarantee any positive approximation factor for the more general problem of Monotone Submodular Maximization with a Knapsack constraint (MSMK) (finding a set $S$ of total cost at most a given budget which maximizes $f(S)$, assuming elements


have costs). Wolsey [27] improved the simple greedy algorithm by a small modification and showed that it achieves an approximation factor of 0.357. Khuller et al. [15] attempted to derive an approximation factor of $1 - 1/\sqrt{e} \approx 0.393$ for the modified greedy algorithm, but their analysis was found flawed [29]. Tang et al. [25] proved that this modified greedy algorithm actually guarantees an approximation factor of 0.405. Subsequently, more careful approximation analysis showed that the exact approximation factor of the modified greedy algorithm is between 0.427 and 0.42945 [11, 17]. Both the simple and modified greedy algorithms run in $O(n^2)$ time, where $n$ is the size of the ground set. Sviridenko [24] proposed a $(1 - 1/e)$-approximation algorithm at the expense of increasing the time complexity to $O(n^5)$. The time complexity of the algorithm was later reduced to $O(n^4)$ without sacrificing the approximation factor [11, 17]. Yaroslavtsev et al. [28] developed a 0.5-approximation algorithm called Greedy+Max that runs in $O(n^2)$ time. Feldman et al. [11] augmented Greedy+Max with a single guess to achieve an approximation factor of 0.6174 in $O(n^3)$ running time. The running times of these algorithms can be reduced from $O(n^i)$ to $\tilde{O}(n^{i-1}/\epsilon)$ (ignoring poly-logarithmic terms) at the cost of losing an $\epsilon$ in the approximation factors using the threshold technique of [2].

Although the above algorithms provide approximation factors, these constant factors usually leave a significant gap between the solution obtained and the optimal solution of a particular problem instance because they consider only the worst-case scenarios. It is challenging to measure how close to optimal an output solution is for a specific problem instance in practice. One possible method is to derive upper bounds on the optimal function values on a per-instance basis. We refer to such bounds as *data-dependent bounds*. Unlike the constant approximation factors which are fixed and independent of problem instances, data-dependent bounds can be used to characterize the actual quality of the solutions constructed on different problem instances in an instance-aware manner. Recently, a few studies have attempted this method. Balkanski et al. [3] and Chakrabarty and Cote [7] introduced instance-specific upper bounds for the MSMC problem, but these bounds cannot be applied to the MSMK problem directly. Tang et al. [25] presented a naive data-dependent upper bound for the MSMK problem.

In this paper, we construct and evaluate new data-dependent upper bounds on the optimal solution for the MSMK problem. Our contributions can be summarized as follows:

- We develop two strategies, a slicing strategy and a removing strategy, to derive data-dependent upper bounds. We theoretically prove that the constructed bounds dominate the optimal solution, and they are tighter than the existing bounds.
- We transform these strategies into linear programs so that multiple base sets can be incorporated into the derivation of data-dependent upper bounds, which further tightens the established bounds.
- We conduct extensive experiments with several real-world applications, including maximum coverage, revenue maximization, and feature selection, to demonstrate empirically the advantages of our proposed bounds in certifying the quality of solutions to the MSMK problem.

The rest of this paper is organized as follows. Section 2 gives a formal definition of the MSMK problem and some preliminaries. Section 3 elaborates the design and analysis of our data-dependent upper bounds. Section 4 discusses the experiments. Finally, Section 5 concludes the paper.

## 2 Preliminaries

Given a ground set $V$, a set function $f\colon 2^V \to \mathbb{R}$ is *submodular* if for any two sets $A, B \subseteq V$, it holds that

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B).$$

An equivalent definition of a submodular set function $f$ is that for any two sets $A \subseteq B \subseteq V$ and any element $v \in V \setminus B$, it holds that

$$f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B).$$

The latter definition describes the *diminishing return* property of a submodular set function.

In this paper, we focus on monotone submodular maximization, where a set function $f$ is *monotone* (non-decreasing) if $f(A) \leq f(B)$ for any two sets $A \subseteq B \subseteq V$. The input to the MSMK problem includes a ground set $V$, a non-negative monotone submodular set function $f : 2^V \to \mathbb{R}_{\geq 0}$, a non-negative modular cost function $c : 2^V \to \mathbb{R}_{\geq 0}$ to measure the cost of selecting elements (where $c(S) = \sum_{v \in S} c(v)$ and $c(v)$ is the cost of element $v$), and a budget $b$. The objective of the MSMK problem is to find a set that maximizes the function value $f$ among all the sets of cost at most $b$, i.e., $\arg\max_{S \subseteq V,\, c(S) \leq b} f(S)$. Without loss of generality, we assume that for each element $v \in V$, $c(v) \leq b$.

We define the *marginal gain* of an element $v$ with respect to a set $S$ as $f_S(v) = f(S \cup \{v\}) - f(S)$, i.e., the increase in the function value by adding $v$ to $S$. We define the *marginal density* of an element $v$ with respect to a set $S$ as $d_S(v) = \frac{f_S(v)}{c(v)}$, i.e., the marginal gain normalized by the element cost. Due to the diminishing return property of a submodular set function, the marginal gain and density of an element $v$ are the highest when $S = \emptyset$ and are the lowest when $S = V \setminus \{v\}$ (among the sets $S$ that do not contain $v$). We refer to the marginal density in the latter case, i.e., $d_{V\setminus\{v\}}(v)$, as the *cutoff density*. For notational convenience, we shall use $f_S(T)$ to denote the marginal gain of a set $T$ with respect to another set $S$, i.e., $f_S(T) = f(S \cup T) - f(S)$.

Let $OPT$ denote an optimal solution to the MSMK problem. For any set $S \subseteq V$, we have

$$\begin{aligned} f(OPT) &= f(OPT \cup S) - f_{OPT}(S) \\ &= f(S) + f_S(OPT) - f_{OPT}(S) \\ &\leq f(S) + \max_{T \subseteq V,\, c(T) \leq b} (f_S(T) - f_T(S)) \\ &\leq f(S) + \max_{T \subseteq V,\, c(T) \leq b} f_S(T), \end{aligned} \tag{1}$$

where the first inequality follows from the fact that $c(OPT) \leq b$. Our main approach is to construct an upper bound $\Lambda(S)$ on the last term $\max_{T \subseteq V,\, c(T) \leq b}(f_S(T) - f_T(S))$ or $\max_{T \subseteq V,\, c(T) \leq b} f_S(T)$ in Equation (1). Given any set $S \subseteq V$, we can then derive an upper bound on $f(OPT)$ by computing $f(S) + \Lambda(S)$, where $S$ will be termed the *base set*. Technically, we develop two strategies: a slicing strategy that constructs an upper bound on $\max_{T \subseteq V,\, c(T) \leq b} f_S(T)$, and a removing strategy that derives an upper bound on $\max_{T \subseteq V,\, c(T) \leq b}(f_S(T) - f_T(S))$ from an upper bound on $\max_{T \subseteq V,\, c(T) \leq b} f_S(T)$. We further transform these strategies into linear programs to facilitate dealing with multiple base sets.

For simplicity of presentation, we define the ground set $V := \{1, 2, \ldots, n\}$. Without loss of generality, we assume that $S = \{1, 2, \ldots, p\}$ when discussing a base set $S \subseteq V$. In addition, we also assume that the elements in $S$ are arranged in non-descending order of cutoff density, i.e., $d_{V\setminus\{1\}}(1) \leq d_{V\setminus\{2\}}(2) \leq \cdots \leq d_{V\setminus\{p\}}(p)$. The elements in the remaining set $V \setminus S$ are arranged in non-ascending order of marginal density with respect to $S$, i.e., $d_S(p+1) \geq d_S(p+2) \geq \cdots \geq d_S(n)$.

# 3 Design and analysis of upper bounds

## 3.1 Removing strategy

It is easy to infer that $\max_{T\subseteq V,\, c(T)\leq b} f_S(T) = \max_{T\subseteq V\setminus S,\, c(T)\leq b} f_S(T)$, because $T$ should include only elements from $V\setminus S$ in order to maximize the marginal gain $f_S(T)$. To construct an upper bound on $\max_{T\subseteq V\setminus S,\, c(T)\leq b} f_S(T)$, an intuitive idea is to find the lowest index $r$ in the remaining set $V\setminus S = \{p+1, p+2, \ldots, n\}$ satisfying $\sum_{i=p+1}^{r} c(i) > b$ [25]. The upper bound is defined as follows:

$$\Lambda^0(S) = \sum_{i=p+1}^{r-1} f_S(i) + \left(b - \sum_{i=p+1}^{r-1} c(i)\right) \cdot d_S(r). \tag{2}$$

In essence, $\Lambda^0$ takes elements in the remaining set greedily according to their marginal density with respect to the base set $S$ until the budget $b$ is fulfilled. Owing to the submodularity of $f$, it is easy to see that $\Lambda^0(S)$ is an upper bound on $\max_{T\subseteq V\setminus S,\, c(T)\leq b} f_S(T) = \max_{T\subseteq V,\, c(T)\leq b} f_S(T)$. Given a base set $S$, the time complexity to compute $\Lambda^0(S)$ is $O(n\log n) = O(|V|\log|V|)$ due to the sorting of the elements.

By Equation (1), $f(S) + \Lambda^0(S)$ gives an upper bound on $f(OPT)$. Nevertheless, it "consumes" a total cost of $b + c(S)$. On the other hand, it is impossible for an optimal solution to spend any cost exceeding the budget $b$. We would like to tighten $\Lambda^0$ by removing some elements and make the total cost equal to $b$, i.e., to construct an upper bound on $\max_{T\subseteq V, c(T)\leq b}(f_S(T) - f_T(S))$.

We start by defining a continuous version of the marginal gain function with respect to $S$ based on the remaining set $V\setminus S = \{p+1, p+2, \ldots, n\}$:

$$G_+(x) = \sum_{i=p+1}^{r-1} f_S(i) + \left(x - \sum_{i=p+1}^{r-1} c(i)\right) \cdot d_S(r), \tag{3}$$

where $r$ is the lowest index satisfying $\sum_{i=p+1}^{r} c(i) > x$.

Note that $\Lambda^0(S) = G_+(b)$. $G_+(x)$ calculates an upper bound on the marginal gain generated by selecting elements from $V\setminus S$ with a total cost of $x$, i.e., for any $T\subseteq V\setminus S$,

$$G_+(c(T)) \geq \sum_{i\in T} f_S(i) \geq f_S(T). \tag{4}$$

Obviously, $G_+(x)$ is non-decreasing. Intuitively, we would like to pick $G_+(b - c(S))$ to fill up the budget $b$. However, $G_+(b - c(S))$ is not necessarily an upper bound on $f_S(OPT\setminus S)$, because we cannot guarantee that $c(OPT\setminus S) \leq b - c(S)$.

We propose to first choose elements to consume a total cost of $b + x$ and then remove elements from $S$ to reduce the total cost to $b$. To do so, we define another continuous function $G_-(x)$ to calculate a lower bound on the loss in the function value $f$ caused by removing elements with a total cost of $x$. Function $G_-(x)$ is defined as:

$$G_-(x) = \sum_{i=1}^{s-1} f_{V\setminus\{i\}}(i) + \left(x - \sum_{i=1}^{s-1} c(i)\right) \cdot d_{V\setminus\{s\}}(s), \tag{5}$$

where $s$ is the lowest index in $S = \{1, 2, \ldots, p\}$ satisfying $\sum_{i=1}^{s} c(i) > x$. That is, $G_-(x)$ takes elements greedily according to their cutoff density until an accumulated cost $x$ is reached.

Clearly, $G_-(x)$ is also non-decreasing. It is easy to see that for any set $T \subseteq S \setminus OPT$, we have

$$G_-(c(T)) \leq \sum_{i \in T} f_{V \setminus \{i\}}(i) \leq f_{V \setminus T}(T) \leq f_{OPT}(T),$$

where the first inequality is by the definition of $G_-(x)$ and the last two inequalities are due to the submodularity of $f$.

To maintain a total cost equal to $b$, if $G_+(b - c(S) + x)$ is picked, we can remove $G_-(x)$ from it. Ideally, we would like to set

$$x^* = \max\{c(OPT \cup S) - b, 0\}.$$

Then, it holds that $b - c(S) + x^* \geq b - c(S) + c(OPT \cup S) - b = c(OPT \cup S) - c(S)$ and $x^* \leq c(OPT \cup S) - c(OPT)$. Hence, we have

$$\begin{aligned}
&G_+(b - c(S) + x^*) - G_-(x^*) \\
&\geq G_+(c(OPT \cup S) - c(S)) - G_-(c(OPT \cup S) - c(OPT)) \\
&= G_+(c(OPT \setminus S)) - G_-(c(S \setminus OPT)) \\
&\geq f_S(OPT \setminus S) - f_{OPT}(S \setminus OPT) \\
&= f(OPT) - f(S).
\end{aligned}$$

However, $c(OPT \cup S)$ is not known as it is dependent on the optimal solution $OPT$. Note that $x^* \leq \max\{c(OPT) + c(S) - b, 0\} \leq c(S)$ always holds. Thus, we can pick the maximal value of $G_+(b - c(S) + x) - G_-(x)$ among all $x \in [0, c(S)]$. Hence, an upper bound incorporating the removing strategy is defined as

$$\Lambda^1(S) = \max_{x \in [0, c(S)]} \{G_+(b - c(S) + x) - G_-(x)\}.$$

Since $G_+(x)$ is a non-decreasing function, we have

$$\begin{aligned}
\Lambda^0(S) = G_+(b) &\geq \max_{x \in [0, c(S)]} G_+(b - c(S) + x) \\
&\geq \max_{x \in [0, c(S)]} (G_+(b - c(S) + x) - G_-(x)) = \Lambda^1(S),
\end{aligned}$$

which shows that $\Lambda^1$ is at least as good as $\Lambda^0$.

### 3.1.1 $\Lambda^1$ in linear program presentation

We can also present the calculation of $\Lambda^1$ as a linear program shown in Equation (6), where $\langle x_1, x_2, \ldots, x_n \rangle \in [0,1]^n$ is an indicator vector.[1]

$$\begin{aligned}
\max \quad & \sum_{i \in V \setminus S} f_S(i) \cdot x_i - \sum_{i \in S} f_{V \setminus \{i\}}(i) \cdot (1 - x_i) \\
\text{subject to} \quad & \sum_{i \in V} c(i) \cdot x_i \leq b.
\end{aligned} \tag{6}$$

We prove the equivalence between the optimum of Equation (6) and $\Lambda^1(S)$ in Appendix A.

[1] All linear programs in this paper will have variables $x_1, x_2, \ldots, x_n$ forming an indicator vector. We shall omit the constraint $\forall i \in V, 0 \leq x_i \leq 1$ in the presentations of linear programs for conciseness.

According to Equation (6), we can arrange elements in $V$ in descending order according to their "weights" in the objective function. That is, for each $i \in S$, its weight is $f_{V\setminus\{i\}}(i)$; for each $i \in V \setminus S$, its weight is $f_S(i)$. We can then pick elements greedily from the beginning and stop when the budget is fulfilled. Given a set $S$, the time complexity to compute $\Lambda^1(S)$ is $O(n \log n) = O(|V| \log |V|)$ due to the sorting of the elements. Thus, $\Lambda^1(S)$ has the same time complexity to compute as $\Lambda^0(S)$.

Similarly, we can also write the calculation of $\Lambda^0$ as a linear program shown in Equation (7).

$$\begin{aligned} \max \quad & \sum_{i \in V\setminus S} f_S(i) \cdot x_i \\ \text{subject to} \quad & \sum_{i \in V} c(i) \cdot x_i \leq b. \end{aligned} \tag{7}$$

Notice that Equation (6) differs from Equation (7) in only one item: $-\sum_{i\in S} f_{V\setminus\{i\}}(i) \cdot (1 - x_i)$. We refer to this item as the "removing item", which is in essence the core of the removing strategy.

## 3.2 Slicing strategy

We develop a slicing strategy to derive a data-dependent upper bound for the MSMK problem, which is inspired by Balkanski et al.'s work [3]. Their method upper bounds the optimal MSMC solution by lower bounding the dual objective of finding the set $S$ of minimum size that has a given function value. We introduce it from a different perspective to motivate our approach to deal with the MSMK problem.

Recall that the remaining set $V \setminus S = \{p+1, p+2, \ldots, n\}$ is sorted in non-ascending order of marginal density (or equivalently marginal gain in the case of MSMC) with respect to $S$. To simplify presentation, we define $V_i := \{1, 2, \ldots, i\}$ and $V_0 := \emptyset$.

Let $k$ be the cardinality constraint. If the upper bound $\Lambda^0$ is applied, we pick the top $k$ elements $p+1, p+2, \ldots, p+k$ from $V \setminus S$ which have the largest marginal densities with respect to $S$. Now we arrange the elements in $OPT \setminus S$ in non-ascending order of marginal density with respect to $S$ and obtain a sorted sequence $\langle o_1, o_2, \ldots, o_k \rangle$ where $o_1 < o_2 < \cdots < o_k$. Similarly, we define $O_i := \{o_1, o_2, \ldots, o_i\}$ and $O_0 := \emptyset$. Let us first notice two properties of $\langle o_1, o_2, \ldots, o_k \rangle$ as follows:

- for each $i \in \{1, 2, \ldots, k\}$, $f_{S\cup O_{i-1}}(o_i) \leq f_S(o_i)$, which is due to the submodularity of $f$;
- for each $i \in \{1, 2, \ldots, k\}$, $f_{S\cup O_{i-1}}(o_i) = f_S(O_i) - f_S(O_{i-1}) \leq f_S(\{p+1, p+2, \ldots, o_i\}) - \sum_{j=1}^{i-1} f_{S\cup O_{j-1}}(o_j) = f_S(V_{o_i}) - \sum_{j=1}^{i-1} f_{S\cup O_{j-1}}(o_j)$, which is due to the monotonicity of $f$.

Balkanski et al.'s method [3] replaces each $f_{S\cup O_{i-1}}(o_i)$ with a variable $v_i$. Each variable $v_i$ corresponds to a distinct element in $V \setminus S$ because $o_i$ can be any element in $V \setminus S$. It then maximizes the total value of the sequence $v_1, v_2, \ldots, v_k$ satisfying the above conditions. Since $f_S(o_1), f_{S\cup O_1}(o_2), \ldots, f_{S\cup O_{k-1}}(o_k)$ is also such a sequence satisfying these conditions, the maximum total value of $v_1, v_2, \ldots, v_k$ is an upper bound for it. Thus, $\sum_{i=1}^k v_i \geq \sum_{i=1}^k f_{S\cup O_{i-1}}(o_i) = f_S(OPT \setminus S)$, and $f(S) + \sum_{i=1}^k v_i$ is an upper bound on $f(OPT)$. The elements picked by this method do not necessarily have successive indexes. Hence, it produces an upper bound at least as good as (no larger than) $\Lambda^0$.

Remember that we would like to construct an upper bound on $\max_{T\subseteq V,\, c(T)\leq b} f_S(T) = \max_{T\subseteq V\setminus S,\, c(T)\leq b} f_S(T)$ for the MSMK problem. The main challenge of MSMK is that the elements have non-uniform costs so that the number of elements to pick and fill up the budget $b$ is not predetermined. A naive approach to address it is to choose a fixed value $\epsilon$ and divide the budget $b$ into slices of cost $\epsilon$ each, as illustrated in Figure 1. Then, to fill

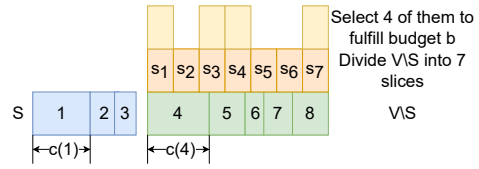



**Figure 1** Naive slicing strategy.

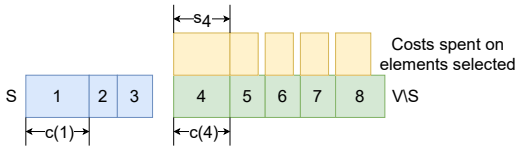



**Figure 2** Slicing strategy producing $\Lambda^2$.

up the budget $b$, we pick $\frac{b}{\epsilon}$ slices. However, the element costs are not necessarily multiples of $\epsilon$. Hence, it is possible for a slice to involve a number of elements, where the number is not fixed and the first as well as last elements may be partially involved. Note that all the elements involved in a slice would have successive indexes. In general, it would not guarantee that the maximum total value of all slices is an upper bound on $f_S(OPT \setminus S)$, because the elements in $OPT \setminus S$ may not have successive indexes.

To address this issue, we can choose $\epsilon \to 0$ so that each slice would involve only one element and thus share the same marginal density with the element it belongs to. The drawback is that setting $\epsilon \to 0$ would increase the number of slices to pick towards infinity and hence may significantly increase the time complexity of computing the upper bound. To guarantee efficiency, we can pick all the slices belonging to the same element at once.

Motivated by these thoughts, we present a novel and elegant design to construct an upper bound $\Lambda^2$ for the MSMK problem.

We start by defining the concept of a valid partition for the MSMK problem.

▶ **Definition 1.** *Given a budget $b$ and a set $S$, a sequence of values $v_1, v_2, \ldots, v_n \in \mathbb{R}_{\geq 0}$ form a valid partition of a value $v \in \mathbb{R}_{\geq 0}$ with respect to $S$ if*

**(i)** $\sum_{i=1}^{n} v_i = v$;

*and there exists a sequence of costs $s_1, s_2, \ldots, s_n \in \mathbb{R}_{\geq 0}$ where $s_i \leq c(i)$ represents the cost spent on element $i$ such that*

**(ii)** $\sum_{i=1}^{n} s_i \leq b$;

**(iii)** *for each $i \in \{1, 2, \ldots, n\}$, $d_S(i) \cdot s_i = \frac{f_S(i)}{c(i)} \cdot s_i \geq v_i$;*

**(iv)** *for each $i \in \{1, 2, \ldots, n\}$, $f_S(V_i) \geq \sum_{j=1}^{i} v_j$.*

In Definition 1, $s_i$ can be interpreted as the total cost of the slices we pick involving element $i$, as illustrated in Figure 2. Condition (ii) states that the total cost of all the slices picked is capped by $b$. Note that for each element $i \in S$, $d_S(i) = 0$. By condition (iii), we always have $v_i = 0$ for each $i \in \{1, 2, \ldots, p\}$.

First, we prove the following property.

▶ **Lemma 2.** *For any set $T \subseteq V \setminus S$ with $c(T) \leq b$, $f_S(T)$ has a valid partition with respect to $S$.*

**Proof Sketch.** We construct a valid partition of $f_S(T)$ to prove this lemma. Please refer to Appendix B for details. ◀

Lemma 2 implies that the maximum value that has a valid partition with respect to $S$ is an upper bound on $\max_{T \subseteq V \setminus S,\, c(T) \leq b} f_S(T)$. We remark that Definition 1 and Lemma 2 actually do not require the remaining set $V \setminus S = \{p+1, p+2, \ldots, n\}$ to be sorted in non-ascending order of marginal density with respect to $S$. On the other hand, having $V \setminus S$ sorted allows us to develop an efficient algorithm to compute the maximum value that has a valid partition.

To find this maximum value, for each element $i \in V \setminus S$, we would like to spend as little cost $s_i$ as possible while maximizing $v_i$. By conditions (iii) and (iv) of Definition 1, in general, the cost to spend on each element $a_i$ should be $s_i = \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i)$, so that the increase from $f_S(V_{i-1})$ to $f_S(V_i)$ is the same as $d_S(i) \cdot s_i$. Thus, we iterate through the sorted sequence $\langle p+1, p+2, ..., n\rangle$ until encountering an element $t$ where the remaining budget $b - \sum_{i=p+1}^{t-1} \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i) \leq \frac{d_{S\cup V_{t-1}}(t)}{d_S(t)} \cdot c(t)$. For each element $i$ $(p+1 \leq i < t)$, a cost of $s_i = \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i)$ is spent on $i$ and we set $v_i = d_S(i) \cdot s_i = f_{S\cup V_{i-1}}(i)$. For the element $t$, a cost of $s_t = b - \sum_{i=p+1}^{t-1} \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i)$ is spent on $t$ and we set $v_t = d_S(t) \cdot s_t$. For all the remaining elements $i$ $(1 \leq i \leq p$ or $t+1 \leq i \leq n)$, we set $s_i = 0$ and $v_i = 0$. Then, we define

$$\Lambda^2(S) = \sum_{i=1}^{n} v_i = \sum_{i=p+1}^{t} v_i = f_S(V_{t-1}) + \left(b - \sum_{i=p+1}^{t-1} \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i)\right) \cdot d_S(t). \tag{8}$$

The idea of constructing $\Lambda^2$ is illustrated in Figure 2. Given a set $S$, the time complexity to compute $\Lambda^2(S)$ is $O(n \log n) = O(|V| \log |V|)$ due to the sorting of the elements. Obviously, $\Lambda^2(S)$ has a valid partition as demonstrated by the cost and value sequences above. Next, we prove that $\Lambda^2(S)$ is the maximum $v$ that has a valid partition and hence an upper bound on $\max_{T\subseteq V\setminus S,\, c(T)\leq b} f_S(T) = \max_{T\subseteq V,\, c(T)\leq b} f_S(T)$.

Notice that $\Lambda^2(S) = \sum_{i=p+1}^{t} v_i = \sum_{i=p+1}^{t} d_S(i) \cdot s_i$. Since $d_S(p+1) \geq d_S(p+2) \geq \cdots \geq d_S(n)$ and $\sum_{i=p+1}^{t} s_i = b$, we have $\sum_{i=p+1}^{t} d_S(i) \cdot s_i \leq \sum_{i=p+1}^{r-1} d_S(i) \cdot c(i) + \left(b - \sum_{i=p+1}^{r-1} c(i)\right) \cdot d_S(r) = \Lambda^0(S)$, where $r$ is the lowest index satisfying $\sum_{i=p+1}^{r} c(i) > b$ as defined in Equation (2). This implies that $\Lambda^2(S) \leq \Lambda^0(S)$ and hence, $\Lambda^2$ is at least as good as $\Lambda^0$.

▶ **Lemma 3.** *The maximum value $v$ that has a valid partition is given by $\Lambda^2(S)$.*

**Proof Sketch.** Let the cost sequence generated for computing $\Lambda^2(S)$ be $s_1, s_2, \ldots, s_n$ and the corresponding value sequence be $v_1, v_2, \ldots, v_n$. If $s_1, s_2, \ldots, s_n$ do not use up the budget $b$, the claim is straightforward since $\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} f_{S\cup V_{i-1}}(i) = f_S(V_n)$ must be the maximum possible due to property (iv) of Definition 1 (by setting $i = n$). Now assume that $\sum_{i=1}^{n} s_i = b$. We prove the lemma by contradiction. Let $\tilde{v}$ be the maximum value that has a valid partition and its cost sequence be $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$. Assume that $\tilde{v} > \Lambda^2(S)$. We check the difference between the two sequences and find the lowest-indexed element $a_q$ where $s_q \neq \tilde{s}_q$. We then derive contradictions for the cases of $\tilde{s}_q > s_q$ and $\tilde{s}_q < s_q$ respectively to complete the proof. Please refer to Appendix C for details. ◀

### 3.2.1 $\Lambda^2$ in linear program presentation

Like $\Lambda^1$, we can write the calculation of $\Lambda^2$ as a linear program shown in Equation (9).

$$\begin{aligned} \max \quad & \sum_{i\in V\setminus S} f_S(i) \cdot x_i \\ \text{subject to} \quad & \forall i \in V, \quad \sum_{j=1}^{i} f_S(j) \cdot x_j \leq f_S(V_i), \\ & \sum_{i\in V} c(i) \cdot x_i \leq b. \end{aligned} \tag{9}$$

We prove in Appendix D that the optimum of Equation (9) is equal to $\Lambda^2(S)$.

Though we have assumed $S = \{1, 2, \ldots, p\}$ and $V \setminus S = \{p+1, p+2, \ldots, n\}$, Equation (9) actually does not require the elements of $S$ to be arranged before the elements of $V \setminus S$ in the ground set. The indexes of their elements can be arbitrarily interleaved. This is because in the first constraint of Equation (9), $f_S(j) = 0$ for any element $j \in S$ and thus it would not increase the value of the left-hand side. On the right-hand side, $f_S(V_i)$ never decreases as $i$ increases. Thus, if the first constraint holds for all $i \in V \setminus S$, it would also hold for all $i \in S$. This observation will be useful when later we introduce a unified upper bound based on multiple base sets $S$, where it is difficult to separate the indexes of $S$ and $V \setminus S$ for all base sets.

## 3.3 Combining removing and slicing strategies

We can combine the removing strategy and slicing strategy, bringing forth a tighter upper bound $\Lambda^3$. Given a set $S$, we denote the upper bound $\Lambda^2(S)$ generated with a budget $x \in [0, b]$ as $\Phi(x)$. The new upper bound $\Lambda^3$ is defined as follows:

$$\Lambda^3(S) = \max_{x \in [0, c(S)]} (\Phi(b - c(S) + x) - G_-(x)),$$

where $G_-(x)$ is defined in Equation (5).

According to the property of $\Lambda^2(S)$, $\Phi(x)$ is a non-decreasing function and $\Phi(x) \geq \max_{T \subseteq V \setminus S,\, c(T) \leq x} f_S(T)$.

Letting $x = \max\{c(OPT \cup S) - b, 0\}$, similar to the derivation in $\Lambda^1(S)$, we have

$$\begin{aligned}
&\Phi(b - c(S) + x) - G_-(x) \\
&\geq \Phi(c(OPT \cup S) - c(S)) - G_-(c(OPT \cup S) - c(OPT)) \\
&= \Phi(c(OPT \setminus S)) - G_-(c(S \setminus OPT)) \\
&\geq f_S(OPT \setminus S) - f_{OPT}(S \setminus OPT) \\
&= f(OPT) - f(S),
\end{aligned}$$

which implies that $f(S) + \Lambda^3(S) \geq f(OPT)$.

In addition, we have

$$\begin{aligned}
\Lambda^2(S) = \Phi(b) &\geq \max_{x \in [0, c(S)]} \Phi(b - c(S) + x) \\
&\geq \max_{x \in [0, c(S)]} (\Phi(b - c(S) + x) - G_-(x)) = \Lambda^3(S),
\end{aligned}$$

which shows that $\Lambda^3$ is at least as good as $\Lambda^2$.

The calculation of $\Lambda^3$ can also be written as a linear program by adding the removing item to Equation (9). By similar arguments to the computational complexity of $\Lambda^1(S)$ in Section 3.1.1, the time complexity to compute $\Lambda^3(S)$ is also $O(n \log n) = O(|V| \log |V|)$, the same as $\Lambda^0(S)$, $\Lambda^1(S)$ and $\Lambda^2(S)$. Please refer to Appendix E for details.

## 3.4 Enhancing bounds with multiple base sets

If we calculate an upper bound with one base set $S$ only (e.g., an empty set or the output solution of an algorithm), the result may not be satisfying. To tighten the bound, we can compute multiple upper bounds with different base sets (such as all the intermediate sets generated by a greedy algorithm for the MSMK problem) and choose the lowest upper bound

obtained as the final result. This enhancement can be conducted on all the proposed new upper bounds ($\Lambda^1$ to $\Lambda^3$). We denote the final result for such an enumeration based on $\Lambda^i$ as $\Lambda^{i+}$, i.e., $\Lambda^{i+} = \min_{S\in\{S_1,S_2,\dots\}} f(S) + \Lambda^i(S)$.

In the above approach, one upper bound is computed for each base set separately. Alternatively, we can compute a shared upper bound for multiple base sets, producing an even tighter bound.

### 3.4.1 Unified augmentation for $\Lambda^0$ and $\Lambda^1$

We first construct a linear program as follows:

$$\begin{aligned} \max \quad & z \\ \text{subject to} \quad & \forall S \subseteq V, \quad z \le f(S) + \Lambda(S, \mathbf{x}), \\ & \sum_{i\in V} c(i)\cdot x_i \le b, \end{aligned} \tag{10}$$

where $\mathbf{x} = \langle x_1, x_2, \dots, x_n\rangle \in [0,1]^n$ is an indicator vector. Let $\mathbf{x^{OPT}}$ be the indicator vector of the optimal solution $OPT$ to the MSMK problem (where $x_i^{OPT} = 1$ if $i \in OPT$, and $x_i^{OPT} = 0$ otherwise). If function $\Lambda(S, \mathbf{x})$ satisfies $f(S) + \Lambda(S, \mathbf{x^{OPT}}) \ge f(OPT)$ for every set $S \subseteq V$, it is easy to see that the optimal value of Equation (10) must be no less than $f(OPT)$.

Equation (10) cannot be solved efficiently because there is an exponential number of possible sets $S$ (or constraints). However, we can relax Equation (10) by considering the constraints relevant to a collection of base sets $\{S_1, S_2, \dots, S_m\}$ only.

$$\begin{aligned} \max \quad & z \\ \text{subject to} \quad & \forall i \in \{1,2,\dots,m\}, \quad z \le f(S_i) + \Lambda(S_i, \mathbf{x}), \\ & \sum_{i\in V} c(i)\cdot x_i \le b. \end{aligned} \tag{11}$$

The optimal value of this linear program is not less than that of Equation (10), and thus guaranteed to be an upper bound on $f(OPT)$. We call this procedure the "unified" augmentation method, since it forces each base set $S_i$ to share the same indicator vector $\mathbf{x}$.

For $\Lambda^0$ and $\Lambda^1$, we can design $\Lambda(S, \mathbf{x})$ according to the objective functions in their linear program representations (Equation (7) and Equation (6)).

Specifically, we define $\Lambda^0(S, \mathbf{x})$ and $\Lambda^1(S, \mathbf{x})$ as follows:

$$\Lambda^0(S, \mathbf{x}) = \sum_{i\in V\setminus S} f_S(i)\cdot x_i, \tag{12}$$

$$\Lambda^1(S, \mathbf{x}) = \sum_{i\in V\setminus S} f_S(i)\cdot x_i - \sum_{i\in S} f_{V\setminus\{i\}}(i)\cdot(1-x_i). \tag{13}$$

Apparently, $\Lambda^0(S, \mathbf{x}) \ge \Lambda^1(S, \mathbf{x})$. For any set $S \subseteq V$, we have

$$\begin{aligned} & f(S) + \Lambda^1(S, \mathbf{x^{OPT}}) \\ = \;& f(S) + \sum_{i\in V\setminus S} f_S(i)\cdot x_i^{OPT} - \sum_{i\in S} f_{V\setminus\{i\}}(i)\cdot(1-x_i^{OPT}) \\ = \;& f(S) + \sum_{i\in OPT\setminus S} f_S(i) - \sum_{i\in S\setminus OPT} f_{V\setminus\{i\}}(i) \\ \ge \;& f(S) + f_S(OPT) - f_{OPT}(S) = f(OPT). \end{aligned} \tag{14}$$

Thus, we can instantiate $\Lambda(S, \mathbf{x})$ with $\Lambda^0(S, \mathbf{x})$ or $\Lambda^1(S, \mathbf{x})$ in Equation (11), and solve Equation (11) to obtain an upper bound on $f(OPT)$. We denote the resulting bounds as $\Lambda^{0*}$ and $\Lambda^{1*}$.

### 3.4.2 Unified augmentation for $\Lambda^2$ and $\Lambda^3$

To calculate a unified upper bound with multiple base sets $S_1, S_2, \ldots, S_m$ for $\Lambda^2$, we construct the following linear program, where the second and third constraints correspond to conditions (iii) and (iv) of Definition 1, and $V_j := \{1, 2, \ldots, j\}$:

$$\begin{aligned} \max \quad & z && (15)\\ \text{subject to} \quad & \forall i \in \{1,2,\ldots,m\}, \quad z \le f(S_i) + \sum_{j \in V} v_j^i, \\ & \forall i \in \{1,2,\ldots,m\} \text{ and } j \in V, \quad v_j^i \le f_{S_i}(j) \cdot x_j, \\ & \forall i \in \{1,2,\ldots,m\} \text{ and } j \in V, \quad \sum_{k=1}^{j} v_k^i \le f_{S_i}(V_j), \\ & \sum_{i \in V} c(i) \cdot x_i \le b. \end{aligned}$$

Equation (15) can be viewed as an extension of Equation (9) to multiple base sets (recall that Equation (9) allows the indexes of the elements in $S$ and $V \setminus S$ to be interleaved arbitrarily). It is easy to prove that the optimum of Equation (15) dominates $f(OPT)$, since setting $\langle x_1, x_2, \ldots, x_n \rangle = \mathbf{x}^{\mathbf{OPT}}$ (the indicator vector of $OPT$) and $z = f(OPT)$ can satisfy all the constraints therein.

To calculate a unified upper bound for $\Lambda^3$, we can further add the removing item to the first constraint of Equation (15) as follows:

$$\begin{aligned} \max \quad & z && (16)\\ \text{subject to} \quad & \forall i \in \{1,2,\ldots,m\}, \quad z \le f(S_i) + \sum_{j \in V} v_j^i - \sum_{j \in S_i} f_{V \setminus \{j\}}(j) \cdot (1 - x_j), \\ & \forall i \in \{1,2,\ldots,m\} \text{ and } j \in V, \quad v_j^i \le f_{S_i}(j) \cdot x_j, \\ & \forall i \in \{1,2,\ldots,m\} \text{ and } j \in V, \quad \sum_{k=1}^{j} v_k^i \le f_{S_i}(V_j), \\ & \sum_{i \in V} c(i) \cdot x_i \le b. \end{aligned}$$

Equation (16) can be viewed as an extension of Equation (17) to multiple base sets. Similarly, the optimum of Equation (16) dominates $f(OPT)$, since setting $\langle x_1, x_2, \ldots, x_n \rangle = \mathbf{x}^{\mathbf{OPT}}$ and $z = f(OPT)$ can satisfy all the constraints therein.

We denote the upper bounds obtained from Equations (15) and (16) as $\Lambda^{2*}$ and $\Lambda^{3*}$ respectively.

## 4 Experiments

We carry out experiments on three different applications to demonstrate the advantage of our proposed bounds ($\Lambda^1$, $\Lambda^2$, $\Lambda^3$ and their augmentations) over the previous bounds $\Lambda^0$ and $\Lambda^{0+}$ in [25]. The experiments are conducted on a Windows machine with a 3.8GHz Intel Xeon W-2235 CPU and 32GB RAM with code written in Python.

**Feature selection.** Feature selection is a procedure widely used in machine learning [1, 4, 8]. Given a feature set $V$ which contains all features that a data point in a training set may have, the goal is to select a subset $S \subseteq V$, with the highest utility subject to a limited budget, to construct a classification model. We adopt the following entropy function [3] to determine the utility value of a feature set $S \subseteq V$:

$$f(S) = -\sum_{x \in X_S} \sum_{y \in Y} p(x, y) \log p(x, y),$$

where $X_S$ is the feature matrix indexed by $S$, $Y$ is the set of labels, and $p(x, y)$ is the joint distribution on $(X_S, Y)$. We experiment with **Adult Income**, a real-world dataset from [5]. This dataset contains 32561 individuals with a variety of features. We retrieve 111 binary features from the dataset as in [13] and then randomly generate 20 ground sets each containing 100 features by uniformly sampling from these 111 binary features.

**Maximum coverage.** Maximum coverage is a classical optimization problem with numerous practical applications, such as influence maximization [26, 14] and sensor placement [16, 18]. It addresses the challenge of using limited resources to achieve the highest utility. Given a graph $G = (V, E)$ where each vertex $v \in V$ has an associated cost $c(v)$, the objective is to find a vertex subset $S \subseteq V$ with limited total cost that maximizes a coverage function:

$$f(S) = |N_G(S)|,$$

where $N_G(S)$ is the graph neighborhood function that returns a set containing all vertices in $S$ and all vertices adjacent to any vertex in $S$ in the graph $G$. We experiment with **ego-facebook** and **com-youtube**, two real-world datasets from [19]. **ego-facebook** contains 4039 nodes and 88234 edges; **com-youtube** (top 5000 communities) contains 39841 nodes and 224234 edges. For each dataset, we randomly construct 20 ground sets each containing 1000 nodes by uniformly sampling from the dataset.

**Revenue maximization.** In social advertising, we would like to choose seed users who will advertise products to their neighbors on social networks. A social network is modeled by a graph $G = (V, E)$, where each node $v \in V$ represents a user, and each edge $(u, v) \in E$ has a weight $w_{uv}$ describing the influence of $u$ on $v$. Each user $v$ will be paid a cost $c(v)$ for advertising products if chosen as a seed user. The goal is to select a subset $S \subseteq V$ of seed users within a budget to maximize product revenue. We adopt the expected product revenue function defined in [6]:

$$f(S) = \sum_{v \in V} \Big( \sum_{u \in S} w_{uv} \Big)^{0.9},$$

where the exponent 0.9 measures diminishing returns. We experiment with a real-world dataset **Caltech36** from [23] containing 769 users. We randomly generate 20 ground sets each containing 100 nodes by uniformly sampling from the dataset.

**Cost and budget setting.** All the datasets lack a cost function. When conducting the experiments for each ground set, we randomly assign a cost to each element in three different ways: (a) all element costs are generated from $U(1, 5)$, where $U(x, y)$ represents a uniform distribution between $x$ and $y$; (b) 80% element costs are generated from $U(1, 5)$ and 20% element costs are generated from $U(5, 20)$ (most costs are small while some are large); (c)

80% element costs are generated from $U(1,5)$ and 20% element costs are generated from $U(0,1)$ (most costs are large while some are small). Similar performance trends are observed for these three distributions. We present the results for (a) here and defer the results for (b) and (c) to the extended version. For cost setting (a), we conduct experiments for all ground sets generated with different budgets from 6 to 40 (step size 1).

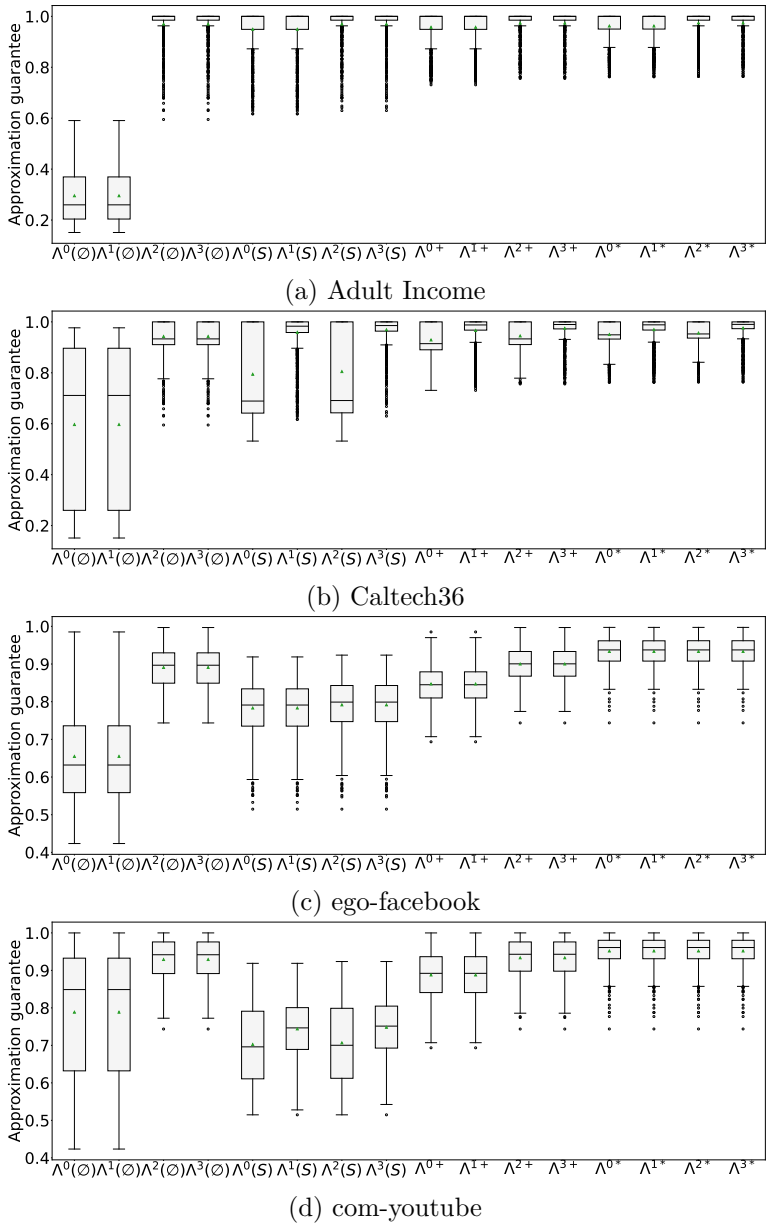
**Algorithms and intermediate sets.** For each problem instance, we run the modified greedy algorithm **MGreedy** [27] and the **Greedy+Max** algorithm [28]. The experimental results for the two algorithms show similar trends. Due to space limitations, we focus on presenting the results of **MGreedy** in this paper. **MGreedy** starts with an empty solution set and employs a greedy heuristic. In each iteration, the greedy heuristic finds the element with the highest marginal density among all elements that can fit into the remaining budget, and adds it to the solution set. The procedure repeats until no more element can be added to the solution set $S$ due to budget violation. **MGreedy** then finds an element $v^*$ which maximizes $f(\{v^*\})$ and compares it with $f(S)$ and outputs the one with a larger function value. The approximation factor of **MGreedy** is between 0.427 and 0.42945 [11, 17].

We calculate the upper bounds $\Lambda^0$, $\Lambda^1$, $\Lambda^2$, $\Lambda^3$ using the empty set $\emptyset$ or the algorithm's output solution $S$ as the base set, and their enumerated and unified augmentations $\Lambda^{i+}$ and $\Lambda^{i*}$ based on all intermediate sets generated by the algorithm (i.e., those produced by the greedy heuristic as well as the singleton set $\{v^*\}$). $\Lambda^i$ and $\Lambda^{i+}$ are calculated by the formulas presented. $\Lambda^{i*}$ is calculated by solving the linear program presented, for which we use the SciPy library in Python. Then, we divide the **MGreedy** solution by the upper bound to derive its approximation guarantee. For each dataset, there are 700 problem instances (20 ground sets $\times$ 35 different budgets from 6 to 40). We present the box plot to illustrate the experimental results across all instances. We use a common box plot showing the first quartile (25th percentile) and the third quartile (75th percentile) as a box, and the boundaries of the whiskers based on 1.5 times the interquartile range (the distance between the third and first quartiles). In addition, the green triangle presents the mean value.

Figure 3 presents the approximation guarantees derived from different upper bounds for the **MGreedy** algorithm. We can make the following observations from the results. First, data-dependent upper bounds typically certify much higher approximation guarantees than the worst-case theoretical result (recall that **MGreedy**'s approximation factor is between 0.427 and 0.42945). The unified upper bounds $\Lambda^{i*}$ confirm that **MGreedy** solutions are quite close to optimal (within 10% in most cases). Second, $\Lambda^2/\Lambda^{2+}/\Lambda^{2*}$ are significantly tighter than $\Lambda^0/\Lambda^{0+}/\Lambda^{0*}$ for most problem instances, which demonstrates the advantage of the slicing strategy. $\Lambda^3/\Lambda^{3+}/\Lambda^{3*}$ (resp. $\Lambda^1/\Lambda^{1+}/\Lambda^{1*}$) are at least as good as $\Lambda^2/\Lambda^{2+}/\Lambda^{2*}$ (resp. $\Lambda^0/\Lambda^{0+}/\Lambda^{0*}$) as proved. As seen from Figure 3(b), the augmentations $\Lambda^{3+}/\Lambda^{3*}$ (resp. $\Lambda^{1+}/\Lambda^{1*}$) are much tighter than $\Lambda^{2+}/\Lambda^{2*}$ (resp. $\Lambda^{0+}/\Lambda^{0*}$) for the **Caltech36** dataset, which will be further analyzed below. Third, upper bounds calculated with only one base set $\emptyset$ or $S$ can be rather loose (e.g., $\Lambda^0(\emptyset)$ and $\Lambda^1(\emptyset)$ for the **Adult Income** dataset). The relative performance of $\Lambda^i(\emptyset)$ and $\Lambda^i(S)$ varies with the dataset. Upper bounds calculated with multiple base sets are normally considerably tighter. The unified upper bounds $\Lambda^{i*}$ are remarkably tighter than the enumerated upper bounds $\Lambda^{i+}$ in many cases.

We present two reasons accounting for the superior performance of the removing strategy on the **Caltech36** dataset.

First, an element $e$ in the **Caltech36** dataset is more likely to have a higher cutoff density $d_{V\setminus\{e\}}(e)$ compared with the other three datasets, which makes the removing strategy more effective. In Figure 4, we present the $d_{V\setminus\{e\}}(e)$ values of elements from different ground sets. The plot contains data from 20000/20000/2000/2000 elements from the 20 ground sets of the **ego-facebook**/**com-youtube**/**Caltech36**/**Adult Income** datasets, respectively.




(a) Adult Income

(b) Caltech36

(c) ego-facebook

(d) com-youtube

**Figure 3** Actual approximation guarantee plots for different upper bounds.

Second, recall that the enumerated augmentation $\Lambda^{i+}$ is the lowest upper bound among all those obtained from taking the intermediate sets of **MGreedy** as base sets. If $\Lambda^{1+}$ or $\Lambda^{3+}$ is given by the upper bound obtained from an empty set (the initial greedy solution), the removing strategy does not further tighten the bound because

$$\begin{aligned}\Lambda^1(\emptyset) &= \max_{x\in[0,c(\emptyset)]} \{G_+(b-c(\emptyset)+x) - G_-(x)\} \\ &= G_+(b-c(\emptyset)+0) - G_-(0) \\ &= G_+(b) \\ &= \Lambda^0(\emptyset),\end{aligned}$$

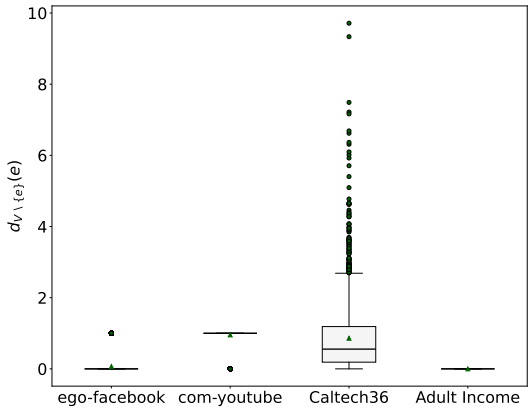



**Figure 4** Cutoff densities of elements in four datasets.

and a similar argument also holds for $\Lambda^{3+}$. $\Lambda^{3+}$ is never equal to $\Lambda^3(\emptyset)$ for the **Caltech36** dataset, while it is common for the other three datasets as illustrated in Table 1. This observation explains the reason why the **com-youtube** dataset fails to have a better $\Lambda^{3+}$ than $\Lambda^{2+}$ despite its cutoff densities $d_{V\setminus\{e\}}(e)$ not being close to 0. For similar reasons, the unified augmentations $\Lambda^{1*}/\Lambda^{3*}$ have the same performance tendencies as the enumerated augmentations $\Lambda^{1+}/\Lambda^{3+}$.

**Table 1** Problem instances where $\Lambda^{3+} = \Lambda^3(\emptyset)$.

| Dataset | # of $\Lambda^{3+} = \Lambda^3(\emptyset)$ | Total # | Percentage |
|---|---|---|---|
| Adult Income | 658 | 700 | 94.0% |
| Caltech36 | 0 | 700 | 0.0% |
| ego-facebook | 396 | 700 | 56.8% |
| com-youtube | 700 | 700 | 100% |

To evaluate the computational efficiency of our upper bounds, we present the box plots of computational time in Figure 5. The computational times of $\Lambda^2/\Lambda^{2+}$ and $\Lambda^0/\Lambda^{0+}$ are nearly the same for all the datasets except **Adult Income**.[2] This demonstrates the computational efficiency of the slicing strategy. The computational times of $\Lambda^1/\Lambda^{1+}$ and $\Lambda^0/\Lambda^{0+}$ are also generally close for all the datasets. This shows that the removing strategy does not introduce much additional computational overhead. Comparing enumerated and unified augmentations, the computational times of $\Lambda^{0*}/\Lambda^{1*}$ are similar to $\Lambda^{0+}/\Lambda^{1+}$, whereas the computational times of $\Lambda^{2*}/\Lambda^{3*}$ can be much higher than $\Lambda^{2+}/\Lambda^{3+}$. This is because the number of constraints is independent of the ground set size in the linear program of unified augmentations $\Lambda^{0*}/\Lambda^{1*}$, while it is proportional to the ground set size in the linear program of unified augmentations $\Lambda^{2*}/\Lambda^{3*}$.

For large-scale problems where computational time may be a concern, we recommend using the enumerated augmentation $\Lambda^{3+}$ and the unified augmentation $\Lambda^{1*}$. They often have much lower computational times than the unified augmentations $\Lambda^{2*}$ and $\Lambda^{3*}$ (see **ego-facebook** and **com-youtube** results in Figure 5, where the ground set size is 1000), while the better bound between $\Lambda^{3+}$ and $\Lambda^{1*}$ is generally close to $\Lambda^{2*}$ and $\Lambda^{3*}$ as shown in Figure 3.

[2] Detailed examinations show that the number of elements visited in $V \setminus S$ for computing $\Lambda^2/\Lambda^{2+}$ is much larger than that for computing $\Lambda^0/\Lambda^{0+}$ for the **Adult Income** dataset, while these numbers are similar for other datasets.

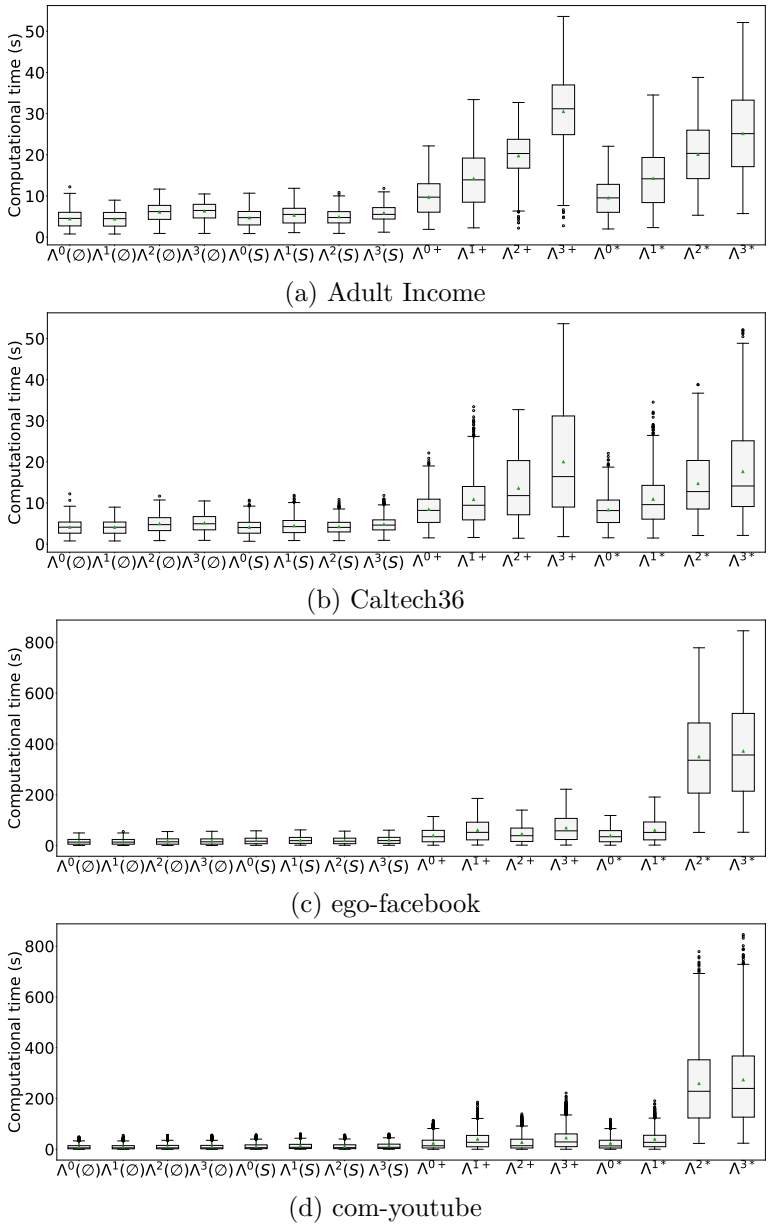



**Figure 5** Computational time plots for different upper bounds.

## 5 Conclusion

In this paper, we have proposed a removing strategy and a slicing strategy for constructing new data-dependent upper bounds for submodular maximization with a knapsack constraint. The upper bounds constructed are shown to be much tighter than existing bounds by extensive experiments. In future work, we would like to find the ways to extend our techniques to submodular maximization problems with other constraints, such as matroid and $p$-system.

## References


1 Magda Amiridi, Nikos Kargas, and Nicholas D Sidiropoulos. Information-theoretic feature selection via tensor decomposition and submodularity. *IEEE Transactions on Signal Processing*, 69:6195–6205, 2021. doi:10.1109/TSP.2021.3125147.

2 Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1497–1514. SIAM, 2014. doi:10.1137/1.9781611973402.110.

3 Eric Balkanski, Sharon Qian, and Yaron Singer. Instance specific approximations for submodular maximization. In *International Conference on Machine Learning*, pages 609–618. PMLR, 2021. URL: http://proceedings.mlr.press/v139/balkanski21a.html.

4 Wei-Xuan Bao, Jun-Yi Hang, and Min-Ling Zhang. Submodular feature selection for partial label learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 26–34, 2022. doi:10.1145/3534678.3539292.

5 Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

6 Adam Breuer, Eric Balkanski, and Yaron Singer. The fast algorithm for submodular maximization. In *International Conference on Machine Learning*, pages 1134–1143. PMLR, 2020. URL: http://proceedings.mlr.press/v119/breuer20a.html.

7 Deeparnab Chakrabarty and Luc Cote. A primal-dual analysis of monotone submodular maximization. *arXiv preprint*, 2023. arXiv:2311.07808.

8 Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint*, 2011. arXiv:1102.3975.

9 Shaddin Dughmi, Tim Roughgarden, and Mukund Sundararajan. Revenue submodularity. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, pages 243–252, 2009. doi:10.1145/1566374.1566409.

10 Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

11 Moran Feldman, Zeev Nutov, and Elad Shoham. Practical budgeted submodular maximization. *Algorithmica*, 85:1332–1371, 2023. doi:10.1007/S00453-022-01071-2.

12 Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 391–398, 2010. URL: https://icml.cc/Conferences/2010/papers/433.pdf.

13 Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *International conference on machine learning*, pages 2544–2553. PMLR, 2018.

14 David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003. doi:10.1145/956750.956769.

15 Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. doi:10.1016/S0020-0190(99)00031-9.

16 Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(2), 2008.

17 Ariel Kulik, Roy Schwartz, and Hadas Shachnai. A refined analysis of submodular greedy. *Operations Research Letters*, 49(4):507–514, 2021. doi:10.1016/J.ORL.2021.04.006.

18 Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 420–429, 2007. doi:10.1145/1281192.1281239.

19 Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

20 Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, 2010. URL: https://aclanthology.org/N10-1134/.

21 Erik Lindgren, Shanshan Wu, and Alexandros G Dimakis. Leveraging sparsity for efficient submodular data summarization. *Advances in Neural Information Processing Systems*, 29, 2016.

22 George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978. doi:10.1287/MOOR.3.3.177.

23 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 4292–4293, 2015. doi:10.1609/AAAI.V29I1.9277.

24 Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004. doi:10.1016/S0167-6377(03)00062-2.

25 Jing Tang, Xueyan Tang, Andrew Lim, Kai Han, Chongshou Li, and Junsong Yuan. Revisiting modified greedy algorithm for monotone submodular maximization with a knapsack constraint. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(1):1–22, 2021. doi:10.1145/3447386.

26 Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. Online processing algorithms for influence maximization. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*, pages 991–1005. ACM, 2018. doi:10.1145/3183713.3183749.

27 Laurence A. Wolsey. Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, 7(3):410–425, 1982. doi:10.1287/MOOR.7.3.410.

28 Grigory Yaroslavtsev, Samson Zhou, and Dmitrii Avdiukhin. "bring your own greedy"+max: near-optimal 1/2-approximations for submodular knapsack. In *International Conference on Artificial Intelligence and Statistics*, pages 3263–3274. PMLR, 2020. URL: http://proceedings.mlr.press/v108/yaroslavtsev20a.html.

29 Ping Zhang, Zhifeng Bao, Yuchen Li, Guoliang Li, Yipeng Zhang, and Zhiyong Peng. Trajectory-driven influential billboard placement. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pages 2748–2757, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3219819.3219946.


## A Equivalence between the optimum of Equation (6) and $\Lambda^1(S)$

▶ **Lemma 4.** *For each value $t \in [0, c(S)]$, there exists an indicator vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in [0,1]^n$ that satisfies $G_+(b - c(S) + t) - G_-(t) = \sum_{i \in V \setminus S} f_S(i) \cdot x_i - \sum_{i \in S} f_{V \setminus \{i\}}(i) \cdot (1 - x_i)$ and $\sum_{i \in V} c(i) \cdot x_i \leq b$.*

**Proof.** We prove this lemma by constructing an $\mathbf{x}$ that satisfies the constraints.

Let $r$ be the index of the last element covered by $G_+(b - c(S) + t)$, i.e., $\sum_{i=p+1}^{r-1} f_S(i) < G_+(b - c(S) + t) \leq \sum_{i=p+1}^{r} f_S(i)$. Similarly, let $s$ be the index of the last element covered by $G_-(t)$, i.e., $\sum_{i=1}^{s-1} f_{V \setminus \{i\}}(i) < G_-(t) \leq \sum_{i=1}^{s} f_{V \setminus \{i\}}(i)$.

We construct an $\mathbf{x}$ as follows:

$$x_i = \begin{cases} 0, & 1 \le i \le s-1, \\ \frac{\sum_{i=1}^{s} c(i) - t}{c(s)}, & i = s, \\ 1, & s+1 \le i \le p, \\ 1, & p+1 \le i \le r-1, \\ \frac{b - c(S) + t - \sum_{i=p+1}^{r-1} c(i)}{c(r)}, & i = r, \\ 0, & r+1 \le i \le n. \end{cases}$$

Then, we have

$$\begin{aligned} G_+(b - c(S) + t) &= \sum_{i=1}^{r-1} f_S(i) + \left(b - c(S) + t - \sum_{i=1}^{r-1} c(i)\right) \cdot \frac{f_S(r)}{c(r)} \\ &= \sum_{i=p+1}^{r-1} f_S(i) \cdot x_i + f_S(r) \cdot x_r = \sum_{i \in V \setminus S} f_S(i) \cdot x_i, \end{aligned}$$

and

$$\begin{aligned} G_-(t) &= \sum_{i=1}^{s-1} f_{V\setminus\{i\}}(i) + \left(t - \sum_{i=1}^{s-1} c(i)\right) \cdot \frac{f_{V\setminus\{s\}}(s)}{c(s)} \\ &= \sum_{i=1}^{s-1} f_{V\setminus\{i\}}(i) \cdot (1 - x_i) + f_{V\setminus\{s\}}(s) \cdot (1 - x_s) = \sum_{i \in S} f_{V\setminus\{i\}}(i) \cdot (1 - x_i). \end{aligned}$$

Thus, $G_+(b - c(S) + t) - G_-(t) = \sum_{i \in V\setminus S} f_S(i) \cdot x_i - \sum_{i \in S} f_{V\setminus\{i\}}(i) \cdot (1 - x_i)$.

Finally,

$$\sum_{i \in V} c(i) \cdot x_i = \sum_{i \in S} c(i) \cdot x_i + \sum_{i \in V\setminus S} c(i) \cdot x_i = c(S) - t + b - c(S) + t = b.$$

Hence, the $\mathbf{x}$ constructed satisfies the two constraints. ◀

▶ **Lemma 5.** *For each indicator vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in [0,1]^n$ that satisfies $\sum_{i \in V} c(i) \cdot x_i \le b$, there exists a value $t \in [0, c(S)]$ such that $G_+(b - c(S) + t) - G_-(t) \ge \sum_{i \in V\setminus S} f_S(i) \cdot x_i - \sum_{i \in S} f_{V\setminus\{i\}}(i) \cdot (1 - x_i)$.*

**Proof.** It suffices to prove the claim for an indicator vector producing the optimal function value of Equation (6), which maximizes the right-hand side of the inequality to be proved. Since the coefficients of all variables $x_i$'s are non-negative, the objective function value of Equation (6) never decreases with increasing $x_i$ values. Thus, without loss of generality, we can assume that the indicator vector producing the optimal function value of Equation (6) satisfies $\sum_{i \in V} c(i) \cdot x_i = b$. Let $t = \sum_{i \in V\setminus S} c(i) \cdot x_i - b + c(S) = c(S) - \sum_{i \in S} c(i) \cdot x_i \ge 0$. We can infer that $\sum_{i \in V\setminus S} f_S(i) \cdot x_i \le G_+(\sum_{i \in V\setminus S} c(i) \cdot x_i)$ since $G_+$ greedily picks elements with the highest marginal densities. Thus,

$$\sum_{i \in V\setminus S} f_S(i) \cdot x_i \le G_+(b - c(S) + t).$$

Similarly, we can infer that $\sum_{i\in S} f_{V\setminus\{i\}}(i)\cdot(1-x_i) \geq G_-(\sum_{i\in S} c(i)\cdot(1-x_i))$ since $G_-$ greedily picks elements with the lowest cutoff densities. Moreover, it follows from $\sum_{i\in V} c(i)\cdot x_i = b$ that $\sum_{i\in S} c(i)\cdot(1-x_i) = c(S) - \sum_{i\in S} c(i)\cdot x_i = c(S) - b + \sum_{i\in V\setminus S} c(i)\cdot x_i$. Therefore,

$$\sum_{i\in S} f_{V\setminus\{i\}}(i)\cdot(1-x_i) \geq G_-(\sum_{i\in S} c(i)\cdot(1-x_i))$$
$$= G_-(c(S) - b + \sum_{i\in V\setminus S} c(i)\cdot x_i) = G_-(t).$$

Hence, we can conclude that $G_+(b-c(S)+t)-G_-(t) \geq \sum_{i\in V\setminus S} f_S(i)\cdot x_i - \sum_{i\in S} f_{V\setminus\{i\}}(i)\cdot(1-x_i)$. ◀

Lemma 4 and Lemma 5 together guarantee the equivalence between the optimum of Equation (6) and $\Lambda^1(S)$.

## B Proof of Lemma 2

▶ **Lemma 2.** *For any set $T \subseteq V \setminus S$ with $c(T) \leq b$, $f_S(T)$ has a valid partition with respect to $S$.*

**Proof.** We construct a valid partition of $f_S(T)$ to prove this lemma. Assume that $T$ has $m$ elements: $T = \{t_1, t_2, \ldots, t_m\}$ where $p+1 \leq t_1 < t_2 < \cdots < t_m \leq n$. To simplify presentation, we define $T_i := \{t_1, t_2, \ldots, t_i\}$ and $T_0 := \emptyset$.

For each element $t_i \in T$, we set $s_{t_i} = \frac{d_{S\cup T_{i-1}}(t_i)}{d_S(t_i)}\cdot c(t_i)$ and $v_{t_i} = f_{S\cup T_{i-1}}(t_i)$. For each element $j \in V\setminus T$, we set $s_j = 0$ and $v_j = 0$. We verify all the conditions of Definition 1.

For condition (i),

$$\sum_{i=1}^{n} v_i = \sum_{i=1}^{m} v_{t_i} = \sum_{i=1}^{m} f_{S\cup T_{i-1}}(t_i) = f_S(T).$$

For condition (ii),

$$\sum_{i=1}^{n} s_i = \sum_{i=1}^{m} s_{t_i} = \sum_{i=1}^{m} \frac{d_{S\cup T_{i-1}}(t_i)}{d_S(t_i)}\cdot c(t_i) \leq \sum_{i=1}^{m} c(t_i) = c(T) \leq b,$$

where the first inequality is due to the submodularity of $f$.

For condition (iii), for each element $t_i \in T$,

$$d_S(t_i)\cdot s_{t_i} = d_{S\cup T_{i-1}}(t_i)\cdot c(t_i) = f_{S\cup T_{i-1}}(t_i) = v_{t_i},$$

and for each element $j \in V\setminus T$,

$$d_S(j)\cdot s_j = 0 = v_j.$$

For condition (iv), for each element $j \in V\setminus T$, let $r$ be the highest index satisfying $t_r \leq j$ (define $r = 0$ and $t_r = 0$ if $j < t_1$), then

$$f_S(V_j) \geq \sum_{k=1}^{r} f_{S\cup T_{k-1}}(t_k) = \sum_{k=1}^{r} v_{t_k} = \sum_{k=1}^{t_r} v_k = \sum_{k=1}^{j} v_k,$$

where the inequality is due to the monotonicity of $f$. ◀

## C Proof of Lemma 3

▶ **Lemma 3.** *The maximum value $v$ that has a valid partition is given by $\Lambda^2(S)$.*

**Proof.** Let the cost sequence generated for computing $\Lambda^2(S)$ be $s_1, s_2, \ldots, s_n$ and the corresponding value sequence be $v_1, v_2, \ldots, v_n$. If $s_1, s_2, \ldots, s_n$ do not use up the budget $b$, i.e., $\sum_{i=p+1}^{n} \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i) < b$, the claim is straightforward since $\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} f_{S\cup V_{i-1}}(i) = f_S(V_n)$ must be the maximum possible due to condition (iv) of Definition 1 (by setting $i = n$). Now assume that $\sum_{i=1}^{n} s_i = b$. Let $t$ be the lowest index satisfying $\sum_{i=1}^{t} s_i = b$. By the procedure for computing $\Lambda^2(S)$,

- for each $p+1 \le i < t$, $s_i = \frac{d_{S\cup V_{i-1}}(i)}{d_S(i)} \cdot c(i)$ and $\sum_{j=1}^{i} v_j = f_S(V_i)$;
- $s_t = \left(b - \sum_{j=p+1}^{t-1} \frac{d_{S\cup V_{j-1}}(j)}{d_S(j)} \cdot c(j)\right) \cdot d_S(t)$ and $\sum_{j=1}^{t} v_j = \Lambda^2(S)$;
- for each $i > t$, $s_i = 0$ and $\sum_{j=1}^{i} v_j = \Lambda^2(S)$.

We prove the lemma by contradiction. Let $\tilde{v}$ be the maximum value that has a valid partition. Assume that $\tilde{v} > \Lambda^2(S)$. There can be multiple valid partitions of $\tilde{v}$. Among all valid partitions of $\tilde{v}$, we pick one with the lowest total cost. If there are multiple valid partitions with the same lowest total cost, we pick one whose cost sequence is the last in the lexicographic order (for any two cost sequences, we can check the lowest index where the costs differ; the sequence with a smaller cost at this index is placed before the other sequence according to the lexicographic order; the lexicographic order is a total order). Let $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$ be the cost sequence and $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_n$ be the value sequence in this valid partition, where $\sum_{i=1}^{n} \tilde{v}_i = \tilde{v}$.

Now we check the difference between the two sequences. We iterate through the sequence $\langle p+1, p+2, \ldots, n\rangle$ to find the first element $q$ where $s_q \neq \tilde{s}_q$. We can assume $q \le t$, since otherwise $\sum_{i=1}^{t} \tilde{s}_i = \sum_{i=1}^{t} s_i = b$ and hence $\tilde{s}_q = 0 = s_q$ must hold.

If $\tilde{s}_q > s_q$, we must have $q < t$, because $\tilde{s}_t \le b - \sum_{i=1}^{t-1} \tilde{s}_i = \sum_{i=1}^{t} s_i - \sum_{i=1}^{t-1} \tilde{s}_i = s_t$. We can construct a new cost sequence $s_1, \ldots, s_{q-1}, s_q, \tilde{s}_{q+1}, \ldots, \tilde{s}_n$ and a new value sequence $v_1, \ldots, v_{q-1}, \sum_{i=1}^{q} \tilde{v}_i - \sum_{i=1}^{q-1} v_i, \tilde{v}_{q+1}, \ldots, \tilde{v}_n$. We verify that they satisfy all the conditions of Definition 1. Condition (i) holds because the total value remains unchanged. Condition (ii) holds because the total cost $\sum_{i=1}^{q} s_i + \sum_{i=q+1}^{n} \tilde{s}_i < \sum_{i=1}^{q} \tilde{s}_i + \sum_{i=q+1}^{n} \tilde{s}_i \le b$. Condition (iii) obviously holds for any $i \neq q$ as the cost-value pair of index $i$ is inherited from an existing valid partition. For $i = q$, we have $d_S(q) \cdot s_q \ge v_q = \sum_{i=1}^{q} v_i - \sum_{i=1}^{q-1} v_i = f_S(V_q) - \sum_{i=1}^{q-1} v_i \ge \sum_{i=1}^{q} \tilde{v}_i - \sum_{i=1}^{q-1} v_i$. Condition (iv) holds for any $i$ since the total value up to index $i$ is the same as an existing valid partition. Thus, the new cost and value sequences form a valid partition. The total value of the new partition is the same as $\sum_{i=1}^{n} \tilde{v}_i$, but the total cost of the new partition is lower than $\sum_{i=1}^{n} \tilde{s}_i$, contradicting that $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$ have the lowest total cost among all valid partitions.

If $\tilde{s}_q < s_q$, we can also construct a new valid partition contradicting the selection of $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$. Notice that $\sum_{i=1}^{q-1} v_i = \sum_{i=1}^{q-1} d_S(i) \cdot s_i = \sum_{i=1}^{q-1} d_S(i) \cdot \tilde{s}_i \ge \sum_{i=1}^{q-1} \tilde{v}_i$, where the first equality follows from the definition of $\Lambda^2(S)$ and the inequality follows from condition (iii) of Definition 1. Hence,

$$\begin{aligned} f_S(V_q) - \sum_{i=1}^{q-1} \tilde{v}_i &\ge f_S(V_q) - \sum_{i=1}^{q-1} v_i \\ &= f_S(V_q) - f_S(V_{q-1}) \\ &= d_{S\cup V_{q-1}}(q) \cdot c(q) \\ &= d_S(q) \cdot s_q \\ &> d_S(q) \cdot \tilde{s}_q. \end{aligned}$$

Thus, condition (iv) has slack for $i = q$ in the assumed valid partition of $\tilde{v}$. Together with $\tilde{s}_q < s_q$, it implies that $\tilde{s}_q$ can be increased without violating both conditions (iii) and (iv) for $i = q$. It can also be inferred that there is positive cost spent on at least one element in $q+1, q+2, \ldots, n$, because otherwise $\tilde{v} = \sum_{i=1}^{n} \tilde{v}_i = \sum_{i=1}^{q} \tilde{v}_i \leq \sum_{i=1}^{q} d_s(i) \cdot \tilde{s}_i \leq \sum_{i=1}^{q} d_s(i) \cdot s_i \leq \Lambda^2(S)$, contradicting that $\tilde{v} > \Lambda^2(S)$. Let $m$ be the lowest index larger than $q$ satisfying $\tilde{s}_m > 0$. If we move a small amount of cost $\epsilon > 0$ from $\tilde{s}_m$ to $\tilde{s}_q$, $\tilde{v}_q$ can be increased by $d_S(q) \cdot \epsilon$. On the other side, $\tilde{v}_m$ can be decreased by $d_S(q) \cdot \epsilon$ (which does not violate condition (iii) for $i = m$ since $d_S(q) \cdot \epsilon \geq d_S(m) \cdot \epsilon$). Meanwhile, condition (iv) is not violated for any $q < i < m$ because $\tilde{v}_i = 0$ for all such $i$ values. In this way, we have constructed a new valid partition of $\tilde{v}$ with higher cost spent on element $q$, contradicting that $\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_n$ are the last cost sequence in the lexicographic order.

We have identified a contradiction in every case and completed the proof. ◀

## D Equivalence between the optimum of Equation (9) and $\Lambda^2(S)$

▶ **Lemma 6.** *For any value $v$ that has a valid partition $v_1, v_2, \ldots, v_n$, there exists an indicator vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in [0,1]^n$ that satisfies all the constraints of Equation* (9) *and $\sum_{i \in V \setminus S} f_S(i) \cdot x_i = v$.*

**Proof.** We set $x_i = \frac{v_i}{f_S(i)}$ for each element $i \in V \setminus S$, and $x_i = 0$ for each element $i \in S$. It is easy to observe that $\sum_{i \in V \setminus S} f_S(i) \cdot x_i = v$ (recall that $v_i = 0$ for each $i \in S$ in a valid partition). Now we proceed to check if the $\mathbf{x}$ constructed meets all the constraints of Equation (9).

For the first constraint, for each element $i \in V$,

$$\sum_{j=1}^{i} f_S(j) \cdot x_j = \sum_{j=1}^{i} v_i \leq f_S(V_i),$$

where the inequality is due to condition (iv) of Definition 1.

Let the cost sequence of the valid partition be $s_1, s_2, \ldots, s_n$. For the second constraint,

$$\sum_{i \in V} c(i) \cdot x_i = \sum_{i=p+1}^{n} \frac{v_i}{d_S(i)} \leq \sum_{i=p+1}^{n} s_i \leq b.$$

where the two inequalities are due to conditions (ii) and (iii) of Definition 1.

In addition, for each element $i \in V \setminus S$,

$$0 \leq x_i = \frac{v_i}{f_S(i)} \leq \frac{s_i \cdot d_S(i)}{f_S(i)} = \frac{s_i}{c(i)} \leq 1,$$

where the second inequality is due to condition (iii) of Definition 1 and the third inequality is due to the constraint $s_i \leq c(i)$ in Definition 1. ◀

▶ **Lemma 7.** *For each indicator vector $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in [0,1]^n$ that satisfies all the constraints of Equation* (9)*, $\sum_{i \in V \setminus S} f_S(i) \cdot x_i$ has a valid partition.*

**Proof.** Let $s_i = c(i) \cdot x_i$ and $v_i = f_S(i) \cdot x_i$. It is easy to observe that $\sum_{i=1}^{n} v_i = \sum_{i=1}^{n} f_S(i) \cdot x_i = \sum_{i \in V \setminus S} f_S(i) \cdot x_i$. Now we proceed to check if $v_i$ and $s_i$ satisfy all the other conditions of a valid partition in Definition 1.

For condition (ii),

$$\sum_{i=1}^{n} s_i = \sum_{i=1}^{n} c(i) \cdot x_i \leq b,$$

where the inequality is due to the second constraint of Equation (9).

For condition (iii), for each element $i \in V$,

$$d_S(i) \cdot s_i = f_S(i) \cdot x_i = v_i.$$

For condition (iv), for each element $i \in V$,

$$\sum_{j=1}^{i} v_i = \sum_{j=1}^{i} f_S(j) \cdot x_j \leq f_S(V_i),$$

where the inequality is due to the first constraint of Equation (9).

Thus, all the conditions of Definition 1 are satisfied. So, $\sum_{i \in V \setminus S} f_S(i) \cdot x_i$ has a valid partition. ◀

With Lemma 6 and Lemma 7, we can conclude that the optimum of Equation (9) is equal to the maximum value that has a valid partition, i.e., $\Lambda^2(S)$.

## E $\Lambda^3$ in linear program presentation

To write the calculation of $\Lambda^3$ as a linear program, we can further add the removing item to Equation (9).

$$\begin{aligned} \max \quad & \sum_{i \in V \setminus S} f_S(i) \cdot x_i - \sum_{i \in S} f_{V \setminus \{i\}}(i) \cdot (1 - x_i) \\ \text{subject to} \quad & \forall i \in V, \quad \sum_{j=1}^{i} f_S(j) \cdot x_j \leq f_S(V_i), \\ & \sum_{i \in V} c(i) \cdot x_i \leq b. \end{aligned} \tag{17}$$

The optimum of Equation (17) is equal to $\Lambda^3(S)$. This can be proved through a similar analysis to Appendix A.

If the elements in $V \setminus S$ are arranged in non-ascending order of marginal density with respect to $S$, it is easy to see that the optimum of the following linear program is equal to $\Lambda^2(S)$ defined in Equation (8):

$$\begin{aligned} \max \quad & \sum_{i \in V \setminus S} f_S(i) \cdot x_i \\ \text{subject to} \quad & \sum_{i \in V} c(i) \cdot x_i \leq b, \\ & \forall i \in S, \quad 0 \leq x_i \leq 1, \\ & \forall i \in V \setminus S, \quad 0 \leq x_i \leq \frac{f_{S \cup V_{i-1}}(i)}{f_S(i)}. \end{aligned} \tag{18}$$

Thus, Equation (18) is a variation of calculating $\Lambda^2(S)$.

If we add the removing item to this linear program, we can then get a variation of calculating $\Lambda^3(S)$ as follows:

$$\begin{aligned} \max \quad & \sum_{i \in V \setminus S} f_S(i) \cdot x_i - \sum_{i \in S} f_{V \setminus \{i\}}(i) \cdot (1 - x_i) \\ \text{subject to} \quad & \sum_{i \in V} c(i) \cdot x_i \leq b, \\ & \forall i \in S, \quad 0 \leq x_i \leq 1, \\ & \forall i \in V \setminus S, \quad 0 \leq x_i \leq \frac{f_{S \cup V_{i-1}}(i)}{f_S(i)}. \end{aligned} \tag{19}$$

Comparing Equation (19) with Equation (6), the only difference is the feasible range for $x_i$ where $i \in V \setminus S$. Similar to computing $\Lambda^1(S)$ based on Equation (6), we can compute $\Lambda^3(S)$ by picking elements greedily according to their "weights" in the objective function until the budget is fulfilled to get the optimum of Equation (19), where for each $i \in S$, its weight is $f_{V \setminus \{i\}}(i)$; for each $i \in V \setminus S$, its weight is $f_S(i)$. Consequently, the time complexity to compute $\Lambda^3(S)$ is also $O(n \log n) = O(|V| \log |V|)$ due to the sorting of the elements, the same as $\Lambda^0(S)$, $\Lambda^1(S)$ and $\Lambda^2(S)$.