Theoretically Grounded Pruning of Large Ground Sets for Constrained, Discrete Optimization

Ankur Nath **Alan Kuhnle**
Department of Computer Science and Engineering, Texas A&M University

Abstract

Modern instances of combinatorial optimization problems often exhibit billion-scale ground sets, which have many uninformative or redundant elements. In this work, we develop light-weight pruning algorithms to quickly discard elements that are unlikely to be part of an optimal solution. Under mild assumptions on the instance, we prove theoretical guarantees on the fraction of the optimal value retained and the size of the resulting pruned ground set. Through extensive experiments on real-world datasets for various applications, we demonstrate that our algorithm, QuickPrune, efficiently prunes over 90% of the ground set and outperforms state-of-the-art classical and machine learning heuristics for pruning.

1 INTRODUCTION

In many data science and machine learning tasks, the optimization of a discrete function is required. For example, *subset selection* problems (Wei et al., 2015; Kim and Boukouvala, 2020), such as selecting the most informative features from a dataset for model training. As another example, consider the problem of selecting a subset of items to display in a recommendation system (Mehrotra and Vishnoi, 2023; Ko et al., 2022), where the goal is to maximize user engagement.

In this work, we consider maximization of an objective function f defined on subsets of $\mathcal{U}$, subject to a knapsack constraint: a modular cost function c on the elements is restricted to be at most the budget κ. Denote by $f_\kappa(X) = \max_{S \subseteq X : c(S) \leq \kappa} f(S)$, where $X \subseteq \mathcal{U}$. For many applications of this problem, the ground set $\mathcal{U}$ is massive, with size n in the billion-scale or larger. On the other hand, the budget constraints are frequently such that an optimal set has only a few elements. For example, in viral marketing campaigns (Kempe et al., 2003a), a company may have a budget to promote only a limited number of products while still aiming to maximize overall sales or customer reach. Intuitively, in these cases, the vast majority of elements of $\mathcal{U}$ are irrelevant to finding $f_\kappa(\mathcal{U})$.

Therefore, rather than solving $f_\kappa(\mathcal{U})$ directly, it may be beneficial to produce a small set $\mathcal{U}' \subseteq \mathcal{U}$ of relatively promising candidate elements. Once $\mathcal{U}'$ is obtained, an expensive heuristic or exact algorithm may be employed to produce a feasible solution. Further, since elements not in $\mathcal{U}'$ are discarded, we desire a range of budgets $[\kappa_{\min}, \kappa_{\max}]$ to be supported, so that the pruned ground set has value beyond a single use. Formally, we have the following problem definition.

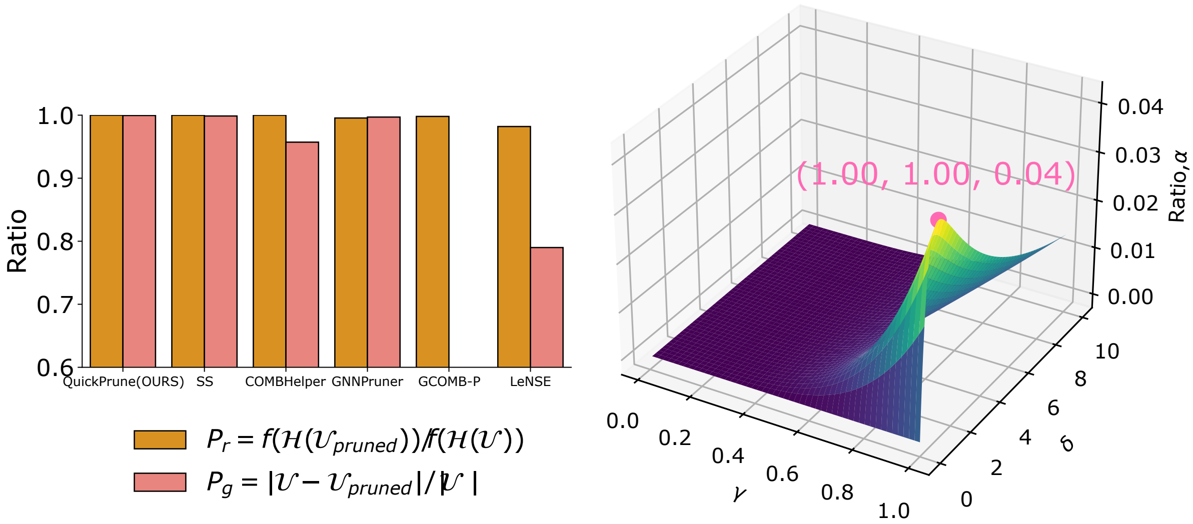
Problem definition (Pruning). Given an objective function $f : 2^{\mathcal{U}} \to \mathbb{R}_+$, modular cost function $c : \mathcal{U} \to \mathbb{R}_+$, and budget range $[\kappa_{\min}, \kappa_{\max}]$, produce $\mathcal{U}' \subseteq \mathcal{U}$, such that

- $|\mathcal{U}'| = \mathcal{O}\left(F(\kappa_{\max}, \kappa_{\min}, c)\,\text{polylog}(n)\right)$, where F is a function and $n = |\mathcal{U}|$;
- there exists $\alpha \in [0, 1]$, such that, for any $\tau \in [\kappa_{\min}, \kappa_{\max}]$, it holds that $f_\tau(\mathcal{U}') \geq \alpha f_\tau(\mathcal{U})$.

Existing methods for the pruning task (Zhou et al., 2017a; Manchanda et al., 2020a; Ireland and Montana, 2022a; Tian et al., 2024a) are heuristics that 1) are only formulated to solve one instance of size-constrained maximization; 2) provide no guarantee on the size of $\mathcal{U}'$; and 3) provide no guarantee on the fraction of the optimal value retained in $\mathcal{U}'$ after the pruning process. Thus, in this work, we are motivated by the following questions:

Is it possible to develop pruning algorithms with theoretical guarantees that are useful beyond solving one problem instance? If so, what assumptions are required on the objective function and the problem instance? Are the resulting algorithms practical and competitive with existing heuristics?

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258. Copyright 2025 by the author(s).



(a) (b)

Figure 1: **(a):** Typical empirical results of QuickPrune versus competing methods on an instance of the MaximumCover problem: QuickPrune retains 99.99% of the optimal value while pruning 99.95% of the ground set. **(b):** Plot of the pruning ratio $\alpha(\varepsilon, \delta, \gamma)$ of Theorem 1, as a function of the parameter δ of QuickPrune and the γ-submodularity of the objective function f. Here, ε is fixed to 0.01.

Contributions. We introduce QuickPrune, a lightweight pruning method that prunes the original ground set $\mathcal{U}$ for a range of budgets in a single pass through the ground set. We emphasize that lightweight pruning methods are needed since an expensive pruning process defeats the purpose of pruning. In this regard, our algorithm processes elements one-by-one – once an element is rejected, it is never re-considered and may be safely discarded. QuickPrune makes at most $\mathcal{O}\left(\log(\kappa_{\max}/\kappa_{\min})\right)$ queries to f per element processed. Moreover, we prove theoretical bounds on the quality of the pruned ground set $\mathcal{U}'$ of QuickPrune, and also on its size.

Specifically, given budget range $\kappa_{\min} \leq \kappa_{\max}$, we show that the output of QuickPrune satisfies $|\mathcal{U}'| = \mathcal{O}\left(\log\left(\frac{\kappa_{\max}}{\kappa_{\min}}\right) \cdot \left(\frac{\kappa_{\max}}{c_{\min}}\right) \log(n)\right)$, where $c_{\min} = \min_{u \in \mathcal{U}} c(u)$. Further, if the objective function f is γ-weakly submodular[1], and if a mild assumption[2] on the costs of elements in an optimal solution of $f_\tau(\mathcal{U})$ holds, we show that $f_\tau(\mathcal{U}') \geq \alpha(\varepsilon, \delta, \gamma) \cdot f_\tau(\mathcal{U})$, for any $\tau \in [\kappa_{\min}, \kappa_{\max}]$, where ε, δ are parameters of the algorithm. A plot of α is shown in Fig. 1(b); If $\gamma = 1$ and $\delta = 1$, $f_\tau(\mathcal{U}') \geq \left(\frac{1}{24} - \epsilon\right) f_\tau(\mathcal{U})$. To the best of our knowledge, all previous methods for this problem are heuristics with no performance guarantee, either on the size of the pruned universe or on the fraction of the optimal value retained after pruning. We give a comprehensive discussion of related work in Section 1.1.

To show the practical effectiveness of our algorithm, we evaluate it against several state-of-the-art classical and machine learning heuristics for the pruning problem across four different optimization contexts. The methods are evaluated by two metrics, the fraction P_g of the original ground set that is pruned (higher is better); and the fraction P_r of the value of a solution by a given algorithm $\mathcal{H}$ for the problem that is retained (higher is better); *i.e.* $P_r = f(\mathcal{H}(\mathcal{U}'))/f(\mathcal{H}(\mathcal{U}))$. Empirically, as shown in Fig. 1, our algorithm outperforms competing methods on both metrics, and achieves substantial reductions in ground set size (typically over 90%) while nearly preserving the value of f_κ across multiple budgets.

Organization. The rest of this paper is organized as follows. In Section 1.1, we describe the relationship of our contribution to existing work; in Section 1.2, we introduce preliminaries and notation. In Section 2, we describe our QuickPrune algorithm and prove Theorem 1, which summarizes its theoretical properties. In Section 3, we conduct our empirical evaluation and comparison to existing heuristics for the pruning problem. In Section 4, we conclude the paper and discuss future directions. In Appendices, we describe omitted details and proofs from the main text.

[1] Submodularity is a diminishing-returns condition ubiquitous to many applications, discussed further in Section 1.2.

[2] Namely, that no single element uses almost all of the budget. See Section 2 for the precise formulation.

1.1 Related Work

(Weakly) Submodular Optimization. *Submodularity* is a notion of diminishing returns that is satisfied or partially satisfied by many, varied objective functions (Dem, 2021; Feige and Vondrak; Feige and Kilian, 1996; Feige et al., 2011a,b; Feige and Izsak, 2013; Gharan and Vondrák, 2010, 2011a,b,c; Gupta et al., 2010; Khot, 2001; Kothawade et al., 2022; Lee et al., 2009; Nemhauser et al., 1978a; Uziahu and Feige, 2023; Vondrák, 2013). Constrained non-submodular maximization has proven useful for data summarization (*e.g.* (Dem, 2021; Kothawade et al., 2022)), feature selection (Elenberg et al., 2018), reduction of training set size for deep learning methods (Killamsetty et al., 2023). Submodularity occupies a role for discrete optimization analagous to the role of convexity in continuous optimization. The typical model (*e.g.* (Feige and Vondrak; Gharan and Vondrák, 2011b; Gupta et al., 2010)) is that the function f is available to an algorithm as a black-box orcale that returns $f(S)$ when queried with set S. However, these function evaluations are very expensive, so we desire to minimize the *query complexity* of an algorithm.

Coresets. Our notion of pruning is related to the idea of a coreset (Braverman et al., 2022; Feldman, 2020; Indyk et al., 2014; Kogan and Krauthgamer, 2015; Liu et al., 2019; Mirrokni and Zadimoghaddam, 2015; Mirzasoleiman et al., 2020; Tukan et al., 2020, 2022; Yang et al., 2023; Zhang et al., 2022); in which a small set is to be selected to best summarize a set of points with respect to a desired set of queries; for example, construct a weighted graph that approximately preserves the values of every cut in an original graph (Kogan and Krauthgamer, 2015). The concept of coreset does not have a precise definition, and thus our terminology *pruned universe* could be recast as coreset. Coresets have been applied to *e.g.* graph summarization (Liu et al., 2019), data-efficient training of ML models (Mirzasoleiman et al., 2020; Yang et al., 2023), pruning neurons from neural networks (Tukan et al., 2022). For the types of optimization problem we consider, *i.e.* constrained maximization with objective function available as a value query oracle, there have been only a small number of attempts to produce a coreset (Indyk et al., 2014; Mirrokni and Zadimoghaddam, 2015). The closest to our setting is the method of Mirrokni and Zadimoghaddam (2015). Their notion of *randomized composable coreset* is used for distributed computation and is only applicable to a single size constraint. Also, their notion of coreset is much stronger than what we require in this work.

Pruning via supervised learning methods. There have been multiple ML-based heuristics proposed to accomplish the pruning task for a single instance of a combinatorial optimization problem. The first line of work (Lauri and Dutta, 2019; Lauri et al., 2023; Sun et al., 2021a,b; Zhang and Ajwani, 2022) seeks to train a binary classification algorithm to predict whether a data element belongs to an optimal solution. These approaches consider graph-based optimization problems and train a linear classifer on vertices by hand-crafting a set of local features for each vertex, such as centrality measures, solutions to linear programming relaxations, and so on; a classifier, such as random forests, is then trained on small instances that can be exactly solved. The goal is to correctly predict when a vertex does *not* belong to an optimal solution, and hence can be safely pruned. These methods rely on hand-crafted features customized to each problem; which themselves may be superlinear to compute, such as a solution to an LP relaxation of the problem or the eigenvector centrality on the original (unpruned) graph. A recent generalization (COMBHelper) of this approach uses a GCN (graph convolution network) in place of a linear classifer (Tian et al., 2024a). We compare to COMBHelper empirically in Section 3.

Other heuristics for pruning. GCOMB (Manchanda et al., 2020a) uses a weighted degree heuristic with a probabilistic greedy algorithm, combined with a supervised learning approach, to produce a pruned ground set. On the other hand, LeNSE (Ireland and Montana, 2022a) trains an RL agent to navigate from an initial subgraph to a pruned subgraph that most likely contains the optimal solution for a specific optimization problem via a local search procedure on subgraphs. Another heuristic is that of Zhou et al. (2017a), which uses ideas from submodularity to extract a promising pruned universe. We empirically compare our algorithm to each of these methods in Section 3. In contrast to our work, all of these are formulated for a single instance of size-constrained maximization and have no theoretical guarantees of any kind, *including that a nonempty feasible solution remains after the pruning process.*

1.2 Preliminaries and Notation

For natural number n, we use the notation $[n] = \{0, 1, \ldots, n-1\}$, and for a function f understood from context: $\Delta\left(T|S\right) = f(S \cup T) - f(S)$ is the *marginal gain* of set T to set S. A non-negative set function on ground set $\mathcal{U}$, $f : 2^{\mathcal{U}} \to \mathbb{R}^+$, is *submodular* iff $\forall S \subseteq \mathcal{U}, \forall T \subseteq \mathcal{U}, f(S \cap T) + f(S \cup T) \leq f(S) + f(T)$. Roughly speaking, this means the whole is not greater than the sum of its parts. An equivalent characterization is the following notion of diminishing-returns: $\forall S \subseteq T \subseteq \mathcal{U}, \forall x \notin T, \Delta\left(x|T\right) \leq \Delta\left(x|S\right)$. A non-negative, set function f is *monotone* iff $\forall S \subseteq T \subseteq \mathcal{U}, f(S) \leq f(T)$. A submodular function is not necessarily monotone.

In this work, we consider the following relaxation of submodularity: the monotone function f is γ-submodular iff γ is the maximum value in $[0,1]$ such that $\forall S \subseteq T \subseteq \mathcal{U}, \forall x \notin T, \gamma\Delta\left(x|T\right) \leq \Delta\left(x|S\right)$. Observe that f is submodular iff $\gamma = 1$. Occasionally in the literature, γ is termed the *diminishing-returns ratio* of a function Bian et al. (2017); Kuhnle et al. (2018). We remark that monotonicity is needed for the definition of γ-submodular, and throughout our analysis, we assume that the objective function f is monotone.

2 QUICKPRUNE: A PRUNING ALGORITHM WITH GUARANTEES

In this section, we describe the pruning algorithms introduced in this paper. In Section 2.1, we describe the pruning algorithm for a single knapsack constraint. The main pruning algorithm (QuickPrune, Alg. 2) runs multiple copies of the single constraint algorithm in parallel, as detailed in Section 2.2.

To prove theoretical guarantees for QuickPrune, we make the following assumption on the costs. Intuitively, the assumption says that no element of an optimal solution consumes a very large fraction of the total budget.

Assumption 1 (No Huge Items (NHI))**.** Given an instance (f, c, κ) of SM-K, the instance satisfies the assumption with $\eta > 0$ if there exists an optimal solution O to the instance, such that, for all $o \in O$, $c(o) \leq \kappa(1 - \eta)$.

For example, in the case of size constraint, this assumption is satisfied if $\kappa \geq 2$ and $\eta \leq 1/2$.

Our theoretical guarantees for the main pruning algorithm are summarized in Theorem 1, which is proved in Section 2.2. Observe that we show a constant factor C of the optimal value is retained for all budgets in the range $[\kappa_{\min}, \kappa_{\max}]$; the desired range of budgets to support by the user. The constant depends mainly on γ, the submodularity parameter of f, and the input parameter δ; C is optimized with $\delta = 1$ at the value $1/24 - \varepsilon$. Also, we show a size bound on the size of $\mathcal{U}'$, the pruned ground set, which depends logarithmically on the size n of $\mathcal{U}$, as well as the values of the maximum and minimum budgets and the minimum cost of an element.

Theorem 1. *Let $\kappa_{\min} < \kappa_{\max}$, let $0 < \eta \leq 1/2$, and let f, c be γ-submodular and modular functions, respectively. Suppose that for all $\kappa' \in [\kappa_{\min}, \kappa_{\max}]$, the instance (f, c, κ') satisfies the* NHI *assumption with η. Let $\mathcal{U}'$ be the output of* QuickPrune *(Alg. 2) with parameters $(f, c, \kappa_{\min}, \kappa_{\max}, \delta, \varepsilon, \eta)$. Then, for all $\kappa' \in [\kappa_{\min}, \kappa_{\max}]$, it holds that $f_{\kappa'}(\mathcal{U}') \geq \frac{\delta\gamma^5(1-\varepsilon\gamma^{-1})}{6(\delta\gamma^2+1)(1+\gamma^{-1}\delta)} f_{\kappa'}(\mathcal{U})$. And, $|\mathcal{U}'| = \mathcal{O}\left(\log(\frac{\kappa_{\max}}{\kappa_{\min}})\left(1 + \frac{\kappa_{\max}}{\delta c_{min}}\right)\log(n/\varepsilon)\right)$.*

2.1 Pruning for a Single Knapsack Constraint

First, we target the case of a single constraint. Given a single budget value κ, we formulate an algorithm to prune for the instance $(f, \mathcal{U}, \kappa, c)$. That is, we produce an algorithm that, for some $\alpha > 0$, produces $\mathcal{U}'$ such that $f_\kappa(\mathcal{U}') \geq \alpha f_\kappa(\mathcal{U})$ and $|\mathcal{U}'| \leq \mathcal{O}\left(F(\kappa, c)\,\mathrm{polylog}(n)\right)$.

The algorithm QuickPrune-Single is given in Alg. 1. In addition to the problem instance, it takes parameter $\delta > 0$, which impacts the size of the resulting pruned set $\mathcal{U}'$ by adjusting the condition to add elements to $\mathcal{U}'$; and parameter $\varepsilon > 0$, which controls how aggressively the algorithm deletes elements. In overview, the algorithm works by taking one pass through the ground set $\mathcal{U}$ and processing elements one-by-one. An element is added to $\mathcal{U}'$ if it meets the condition on Line 7: $\Delta\left(e|A\right) \geq \frac{\delta c(e) f(A)}{\kappa}$; intuitively, this means the element should increase the value of A (the pruned set) an amount proportional to $c(e)/\kappa$ to be worth retaining.

To ensure the size of $\mathcal{U}'$ doesn't grow too large, a deletion condition is checked on Line 11, which intuitively asks if the value of A has increased by a large factor from the previous checkpoint; if so, the original elements of A are discarded. Submodularity is used to bound the amount of value lost from the deletion, and the condition on element addition bounds the maximal number of elements required for the increased value.

The theoretical properties are summarized in the following theorem, proven in Section 2.1.1.

Theorem 2. *Let Alg. 1 be run on instance $(\mathcal{U}, c, f, \kappa)$, with parameters $\varepsilon, \delta > 0$, such that f is γ-submodular. Then, Alg. 1 produces pruned universe $\mathcal{U}'$, such that $|\mathcal{U}'| < 2\left(1 + \frac{\kappa}{\delta c_{min}}\right)\log(n/\varepsilon) + 3$, and there exists $A' \subseteq \mathcal{U}'$, such that $c(A') \leq \kappa$ and $f(A') \geq \frac{\delta\gamma^4(1-\varepsilon\gamma^{-1})}{2(\delta\gamma^2+1)(1+\gamma^{-1}\delta)}\mathrm{OPT}$.*

2.1.1 Proof of Theorem 2

Due to space constraints, omitted proofs are provided in Appendix A. We require the following fact about geometrically increasing sequences of real numbers. For our purposes, the sequence (y_i) will assume the values of $f(A)$ as elements are added.

Fact 1. Let $(y_i)_{i=1}^m$ be a sequence of positive real

Algorithm 1 QuickPrune-Single: Pruning for a single constraint.

```
1: Input: Instance $(f, \mathcal{U}, c, \kappa)$, size-control parameter $\delta > 0$, deletion parameter $\varepsilon > 0$
2: Output: Pruned ground set, $\mathcal{U}'$
3: Initialize: $A \leftarrow \emptyset$, $a^* \leftarrow \emptyset$, $A_s \leftarrow \emptyset$
4: for $e \in \mathcal{U}$ do
5:     if $c(e) > \kappa$ then
6:         continue
7:     if $\Delta(e|A) \geq \frac{\delta c(e) f(A)}{\kappa}$ then
8:         $A \leftarrow A + e$
9:     if $f(e) > f(a^*)$ then
10:        $a^* \leftarrow e$
11:    if $f(A) > \frac{n}{\varepsilon} f(A_s)$ then
12:        $A \leftarrow A \setminus A_s$
13:        $A_s \leftarrow A$
14: return $\mathcal{U}' \leftarrow A + a^*$
```

numbers, such that, for some $\beta > 0$, it holds that $y_i \geq (1+\beta) y_{i-1}$, for all $i \in [m]$. Let $\gamma > 0$. Then, if $m \geq \frac{\beta+1}{\beta} \log \gamma^{-1}$, it holds that $y_m \geq y_1/\gamma$.

Using Fact 1, we bound the number of elements in $\mathcal{U}'$. Intuitively, the geometric increase in the value of $f(A)$ yields a bound on the maximum number of elements until the deletion condition on Line 11 is triggered.

Proposition 1. *The size of $\mathcal{U}'$, as returned by Alg. 1, satisfies $|\mathcal{U}'| < 2\left(1 + \frac{\kappa}{\delta c_{min}}\right) \log(n/\varepsilon) + 3$, where $c_{min} = \min_{a \in \mathcal{U}} c(a)$.*

Next, we bound how much value may have been lost from deletion throughout the execution of the algorithm. In the following proposition, $\hat{A}$ is all elements ever added to A, and $\dot{A}$ is the actual set obtained by the algorithm after all deletions. We show at most an $\varepsilon\gamma^{-1}$-fraction of value is lost.

Proposition 2. *Suppose f is γ-submodular. Let A_i be the value of A after the execution of iteration i of the* ***for*** *loop, for $i = 1$ to n, and $A_0 = \emptyset$. Let $\hat{A} = \bigcup A_j, \dot{A} = A_n$. Then $f(\dot{A}) \geq (1 - \gamma^{-1}\varepsilon) f(\hat{A})$.*

So far, we have established a size bound on $\mathcal{U}'$ and bounded the value of $\mathcal{U}'$ lost from deletion. Next, we turn to showing that there exists a feasible set inside $\mathcal{U}'$ that has a constant fraction of the optimal value. We start by showing in Lemma 1 and Proposition 3 that a set A^* exists within $\mathcal{U}'$ that has a constant fraction of $f(\mathcal{U}')$.

Lemma 1. *Let f be γ-submodular. Let $\delta, \kappa > 0$. Suppose $A_i = \{a_1, \dots a_i\}$ is a sequence of sets satisfying (1) $c(a_i) \leq \kappa$, and (2) $\Delta(a_i | A_{i-1}) \geq \gamma \frac{\delta c(a_i) f(A_{i-1})}{\kappa}$, for each i from 1 to m. Let $f(a^*) \geq \max_{i \in [m]} f(a_i)$, and $c(a^*) \leq \kappa$. Then there exists $A^* \subseteq A_m + a^*$, such that $c(A^*) \leq \kappa$, and $f(A^*) \geq \frac{\delta\gamma^4}{2(\delta\gamma^2+1)} f(A_m)$.*

Algorithm 2 QuickPrune: The Pruning Algorithm.

```
Input: Instances $(f, \mathcal{U}, c, [\kappa_{\min}, \kappa_{\max}])$, parameters $\delta > 0$, $\varepsilon > 0$, $0 < \eta \leq 1/2$.
Output: Pruned ground set, $\mathcal{U}'$
$\mathcal{B} = \{\tau_i = \kappa_{\max}(1-\eta)^i : i \in \mathbb{Z}, (1-\eta)\kappa_{\min} \leq \tau_i \leq \kappa_{\max}\}$
Initialize a copy of $\mathcal{QP}_\tau$ of QuickPrune-Single for each $\tau \in \mathcal{B}$, with parameters $\varepsilon, \delta$.
for $e \in \mathcal{U}$ do
    Pass $e$ to $\mathcal{QP}_\tau$ for all $\tau \in \mathcal{B}$.
return the union of all sets returned by all instances $\mathcal{QP}_\tau$
```

Proposition 3. *Let $\mathcal{U}' = \dot{A} + a^*$ have its value at termination of* QuickPrune-Single. *There exists $A' \subseteq \mathcal{U}' = \dot{A} + a^*$, such that $f(A') \geq \frac{\delta\gamma^4}{2(\delta\gamma^2+1)} f(\dot{A})$ and $c(A') \leq \kappa$.*

Next, we relate the value of $f(\mathcal{U}')$ to $\mathrm{OPT} = f_\kappa(\mathcal{U})$ in Proposition 4.

Proposition 4. *Suppose Alg. 1 is run on instance $(\mathcal{U}, c, f, \kappa)$ with parameters $\delta, \varepsilon > 0$. Let $\hat{A} = \bigcup_{j \in [n]} A_j$ be all elements added to A. Then $f(\hat{A}) \geq \mathrm{OPT}/(1 + \gamma^{-1}\delta)$, where $\mathrm{OPT} = \max_{S \subseteq \mathcal{U} : c(S) \leq \kappa} f(S)$.*

Proof of Theorem 2. Finally, we are ready to put all the pieces together. Assume the hypotheses of the theorem statement. The size bound on $\mathcal{U}'$ follows by Proposition 1. Let A_j be the value of the set A at the beginning of the jth iteration of QuickPrune-Single. Let $\dot{A} = A_{n+1}$ be the final value of A, $\hat{A} = \bigcup_{i \in [n+1]} A_i$. Let A' be as guaranteed by Proposition 3. Then$f(A') \overset{(a)}{\geq} \frac{\delta\gamma^4}{2(\delta\gamma^2+1)} f(\dot{A}) \overset{(b)}{\geq} \frac{\delta\gamma^4(1-\varepsilon\gamma^{-1})}{2(\delta\gamma^2+1)} f(\hat{A})$, where (a) is by Prop. 3, (b) is by Prop. 2, and the result follows from Prop. 4. □

2.2 The Pruning Algorithm: QuickPrune

In this section, we describe the main pruning algorithm QuickPrune (Alg. 2) and prove Theorem 1.

The algorithm QuickPrune works as follows. As input, it takes instances with a range of budgets $[\kappa_{\min}, \kappa_{\max}]$, the parameters ε, δ for QuickPrune-Single, and a parameter $\eta > 0$, such that all of the instances satisfy Assumption NHI with η. The algorithm then runs $\log(\kappa_{\max}/\kappa_{\min})$ copies of QuickPrune-Single in parallel – one each for budget $\tau_i = \kappa_{max}(1-\eta)^i$ in the range $[(1-\eta)\kappa_{\min}, \kappa_{\max}]$.

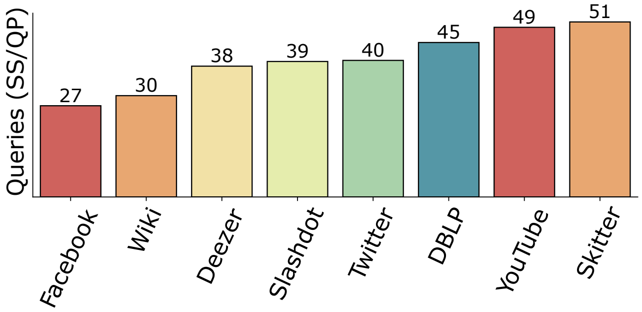


Figure 2: Comparison of the number of oracle calls between SS and QuickPrune.

It returns the union of all pruned sets returned from each copy of QuickPrune-Single.

To establish Theorem 1, we first show the following proposition, with implications to how Assumption NHI can relate an optimal solution of a given budget to one of the instances handled by one of the copies of QuickPrune-Single.

Proposition 5. *Suppose instance* (f, c, κ) *satisfies the* NHI *assumption with* $0 < \eta \leq 1/2$. *Then, there exists an optimal solution* O *to the instance, such that* O *can be partitioned into at most three sets* $\{O_i : i \in [3]\}$, *such that* $c(O_i) \leq \kappa(1 - \eta)$, *for each* $i \in [3]$.

Proof of Theorem 1. Assume the hypotheses of the theorem, and let $\kappa' \in [\kappa_{\min}, \kappa_{\max}]$. By the choice of the set $\mathcal{B}$ on Line 3, there exists a $\tau \in \mathcal{B}$, such that $\kappa'(1 - \eta) \leq \tau \leq \kappa'$. Let $O \subseteq \mathcal{U}$ be an optimal solution to $f_{\kappa'}(\mathcal{U})$ satisfying Assumption NHI. By Proposition 5, O can be partitioned into at most three sets $\{O_i\}$, such that $c(O_i) \leq \kappa(1 - \eta) \leq \tau$.

Moreover, by Theorem 2, there exists $X \subseteq \mathcal{U}'$, such that 1) $c(X) \leq \tau$, and 2) $f(X) \geq \alpha f(O_i)$, with $\alpha = \frac{\delta\gamma^4(1-\varepsilon\gamma^{-1})}{2(\delta\gamma^2+1)(1+\gamma^{-1}\delta)}$. Therefore, $3f(X) \geq \alpha \sum_{i=1}^{3} f(O_i) \geq \alpha\gamma f(O)$, where the last inequality follows from submodularity. Thus, $f(X) \geq \alpha\gamma f(O)/3$. Finally, the size bound on $\mathcal{U}'$ follows from Prop. 1 and the size of $\mathcal{B}$. □

3 EMPIRICAL EVALUATION

In this section, we compare the empirical performance of QuickPrune to existing methods for the pruning problem across four different objective functions, for both size and knapsack constraints. The size constraint is the special case of the knapsack constraint when $c(u) = 1$ for all $u \in \mathcal{U}$ – as the previous methods for pruning are typically formulated for size constraint, this allows for the fairest comparison with existing work. We provide the code and detailed instructions for reproducing the experiments in the supplementary material.

Summary of results. For both size and knapsack constraints, we demonstrate that QuickPrune typically prunes over 90% of the ground set and sacrifices very little in terms of objective value, achieving the highest combined metric (defined below) of all pruning methods in the majority of instances. The algorithm that is most competitive with QuickPrune is the Submodular Sparsification (SS) algorithm of Zhou et al. (2017b). However, SS requires 30 times more oracle queries than QuickPrune, as shown in Fig. 2. For the ML-based pruning methods, a baseline GnnPruner that we introduce outperforms the methods of GCOMB-P (Manchanda et al., 2020b), LeNSE (Ireland and Montana, 2022b), and COMBHelper (Tian et al., 2024b) on nearly every instance that we evaluated. Finally, in Section 3.3, we show that QuickPrune improves over QuickPrune-Single by 5-10% in objective value retained, while increasing the size of the reduced ground set by less than 1% percent.

Evaluation metrics. We evaluate the algorithms using three performance metrics, where higher values indicate better performance in all cases. The first metric is the *pruning approximation ratio* P_r, defined as the ratio of the objective value obtained from the pruned ground set $\mathcal{U}'$ to the objective value from the original ground set $\mathcal{U}$, both computed by a heuristic $\mathcal{H}$. Specifically, $P_r = f(\mathcal{H}(\mathcal{U}'))/f(\mathcal{H}(\mathcal{U}))$. The second metric is the *pruned fraction* P_g of the original ground set that is pruned. Finally, since an algorithm could trivially obtain the highest in one metric by totally disregarding the other (pruning nothing or pruning everything), we consider the *combined metric*, $C = P_r P_g$.

In Appendix H.1, we summarize the heuristics used for each application, noting that our heuristic-agnostic algorithm is flexible for use with any heuristic. We use only a single attempt to prune the ground set for all algorithms due to the computational cost.

Applications. We evaluate our algorithm on four applications: Maximum Cover (MaxCover), Maximum Cut (MaxCut), Influence Maximization (IM), and information retrieval. For detailed specifications of these applications, see Appendix C.

Baselines and Prior Methods. We compare our algorithm against algorithms that accelerate heuristics by pruning the ground set rather than directly optimizing the objective function. Our evaluation include the following classical and learning based methods: Submodular Sparsification (SS) (Zhou et al., 2017b), GCOMB-P (Manchanda et al., 2020b), LeNSE (Ireland and Montana, 2022b), COMBHelper (Tian et al., 2024b), and GnnPruner, a proposed base-

Table 1: Comparison of pruning algorithms for size constraint experiments (best combined metric in bold); *Values as reported in Ireland and Montana (2022b) and "–" denotes no reasonable result is achieved by the corresponding algorithm under the time constraint.

	QuickPrune			SS			GCOMB-P *			COMBHelper			LeNSE *			GnnPruner		
Graph	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$
Maximum Cover																		
Facebook	1.0000	0.9953	**0.9953**	0.9884	0.9027	0.8922	0.9270	0.0700	0.0649	1.0000	0.3840	0.3840	0.9660	0.0700	0.0676	1.0000	0.7697	0.7697
Wiki	0.9998	0.9422	**0.9420**	0.9559	0.9346	0.8934	0.9900	0.0300	0.0297	1.0000	0.4528	0.4528	1.0940	0.3400	0.3720	1.0000	0.9199	0.9199
Deezer	0.9606	0.9797	0.9411	0.9870	0.9855	**0.9727**	0.9940	0.1300	0.1292	1.0000	0.7278	0.7278	0.9790	0.7500	0.7343	1.0000	0.9151	0.9151
Slashdot	1.0000	0.9925	**0.9925**	0.9824	0.9889	0.9715	1.0000	0.0200	0.0200	1.0000	0.9844	0.9844	0.9790	0.6900	0.6755	1.0000	0.9810	0.9810
Twitter	0.9929	0.9911	**0.9841**	0.9306	0.9893	0.9206	0.9970	0.1700	0.1695	1.0000	0.5654	0.5654	0.9890	0.3300	0.3264	0.9987	0.9793	0.9780
DBLP	0.9951	0.9957	**0.9908**	0.9945	0.9963	**0.9908**	0.9990	0.0300	0.0300	1.0000	0.1818	0.1818	0.9900	0.9000	0.8910	1.0000	0.8705	0.8705
YouTube	1.0000	0.9995	**0.9995**	0.9999	0.9987	0.9986	0.9980	0.0700	0.0699	1.0000	0.9572	0.9572	0.9820	0.7900	0.7758	0.9947	0.9967	0.9914
Skitter	0.9985	0.9997	0.9982	0.9857	0.9991	0.9848	0.9990	0.1000	0.0999	–	–	–	0.9760	0.7000	0.6832	0.9993	0.9891	**0.9884**
Maximum Cut																		
Facebook	0.9886	0.7935	0.7845	0.9928	0.9027	**0.8962**	0.8130	0.9500	0.7723	1.0000	0.1538	0.1538	1.0000	0.0700	0.0700	1.0000	0.6774	0.6774
Wiki	0.9985	0.9037	0.9023	0.9981	0.9346	**0.9328**	0.9200	0.9600	0.8832	1.0000	0.7011	0.7011	0.9810	0.3900	0.3826	1.0000	0.9004	0.9004
Deezer	0.9996	0.9730	0.9726	0.9999	0.9855	**0.9854**	0.8500	0.9900	0.8415	1.0000	0.3754	0.3754	0.9750	0.7400	0.7215	1.0000	0.9188	0.9188
Slashdot	1.0000	0.9904	**0.9904**	1.0000	0.9889	0.9889	0.6320	0.9900	0.6257	0.7935	0.9929	0.7879	0.9900	0.6200	0.6138	1.0000	0.9797	0.9797
Twitter	1.0000	0.9900	**0.9900**	1.0000	0.9893	0.9893	0.6280	0.9900	0.6217	1.0000	0.5542	0.5542	0.9870	0.4800	0.4738	1.0000	0.9740	0.9740
DBLP	1.0000	0.9954	0.9954	1.0000	0.9963	**0.9963**	0.6460	0.9900	0.6395	1.0000	0.1825	0.1825	0.9930	0.9200	0.9136	1.0000	0.8623	0.8623
YouTube	1.0000	0.9994	**0.9994**	1.0000	0.9987	0.9987	0.5360	0.9900	0.5306	0.9990	0.9705	0.9695	0.9870	0.7900	0.7797	0.9936	0.9968	0.9904
Skitter	1.0000	0.9995	**0.9995**	1.0000	0.9991	0.9991	0.4270	0.9900	0.4227	–	–	–	0.9740	0.7100	0.6915	1.0000	0.9884	0.9884
Influence Maximization																		
Facebook	0.9919	0.7450	0.7390	0.9208	0.9027	**0.8312**	0.9510	0.7300	0.6942	1.0039	0.3248	0.3261	0.9790	0.0900	0.0881	0.9938	0.4917	0.4887
Wiki	1.0269	0.8831	0.9069	0.9863	0.9346	**0.9218**	0.9690	0.9000	0.8721	0.9969	0.6066	0.6047	0.9600	0.5100	0.4896	1.0011	0.8964	0.8974
Deezer	0.9384	0.9698	0.9101	1.0262	0.9855	**1.0113**	0.8050	0.5500	0.4428	0.9837	0.1093	0.1075	0.9720	0.7600	0.7387	0.9984	0.8478	0.8464
Slashdot	0.9984	0.9881	0.9865	0.9959	0.9889	0.9848	0.9660	0.9800	0.9467	1.0076	0.9875	**0.9950**	0.9660	0.7700	0.7438	0.9993	0.9797	0.9790
Twitter	1.0045	0.9965	**1.0010**	0.9575	0.9893	0.9473	0.9200	0.9800	0.9016	0.9972	0.4947	0.4933	0.9660	0.4000	0.3864	0.9842	0.9758	0.9604
DBLP	0.8603	0.9946	0.8557	1.0495	0.9963	**1.0456**	0.8630	0.9900	0.8544	0.9873	0.0194	0.0192	0.9690	0.8900	0.8624	0.7558	0.3325	0.2513
YouTube	0.9677	0.9994	0.9671	0.9846	0.9987	0.9833	0.9330	0.9900	0.9237	0.9992	0.9705	0.9697	0.9710	0.7500	0.7282	1.0018	0.9966	**0.9984**
Skitter	0.9813	0.9999	0.9812	1.0032	0.9991	**1.0023**	0.8830	0.9900	0.8742	–	–	–	0.9830	0.7800	0.7667	1.0023	0.9889	0.9912

line, which is a modified version of COMBHelper. While the authors of COMBHelper stated that they used GCN (Kipf and Welling, 2016) as their choice of GNN, we observed that the implementation of the algorithm actually employs SAGEConv (Hamilton et al., 2017). To ensure a fair comparison, we have included both GNN options in our evaluation. Empirically, we find that replacing SAGECONV in COMBHelper with GCN significantly improves the performance of COMBHelper, as GCN retains information from all neighbors, unlike SAGECONV. Hence, our proposed baseline, GnnPruner, a modified version of COMBHelper, uses a GCN with fewer layers and uses random numbers as node features, eliminating the need for domain knowledge and extensive feature engineering. Moreover, GnnPruner also outperforms the other ML methods, LeNSE and GCOMB-P.

For the general knapsack constraint, the prior methods do not easily generalize. Since GnnPruner does easily generalize and outperformed the other ML-based methods on size constraints, we only compare to GnnPruner for general knapsack constraints. Instead of using random numbers as node features, we provide the degree, cost, and degree-to-cost ratio of each node as input features for GnnPruner. Additionally, we compare to the Top-k approach, which selects a set (where the size of the set is equal to the size of the pruned ground set by QuickPrune) consisting of the top k degree-to-cost ratios. Further descriptions of each algorithm can be found in Appendix D.

Datasets. We evaluate our approach using real-world datasets from the Stanford Large Network Dataset Collection (Leskovec and Sosič, 2016) for the traditional CO problems on graph, following the experimental setup of Ireland and Montana (2022b). For experiments related to the retrieval system, we use the Beans (Lab, 2020), CIFAR100 (Krizhevsky, 2009), FOOD101 (Bossard et al., 2014) and UCF101 dataset (Soomro et al., 2012). A summary of all datasets used can be found in Appendix E.

3.1 Set 1: Evaluation on Size Constraints

In this section, we evaluate the pruning methods for size constraints; first for classical problems on graphs, and then for the image retrieval application.

Classical Problems on Graph. We consider traditional problems on graphs: Maximum Cover (MaxCover), Maximum Cut (MaxCut), and Influence Maximization (IM), with a budget of $b = 100$, following Ireland and Montana (2022b). From Table 1, we observe that all algorithms achieve similar pruning approximation ratios. However, QuickPrune is able to prune the original ground set by orders of magnitude more than most algorithms. Interestingly, and perhaps surprisingly, the IMM heuristic (Tang et al., 2015) occasionally achieves better solutions for IM on the reduced ground set compared to the unpruned ground set (*e.g.* Twitter, Wiki) – which raises the intriguing possibility of pruning to enhance the performance of a specific algorithm, which is out of the scope of this work.

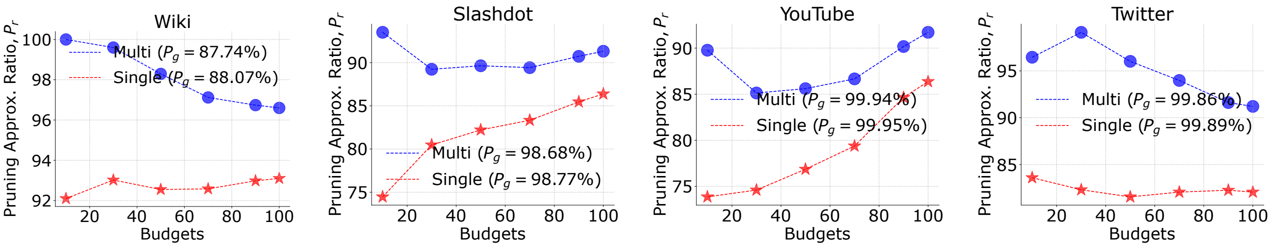


Figure 3: Comparison between QuickPrune and QuickPrune-Single on a selection of data sets for IM.

We observe that SS outperforms our method on some instances, particularly for IM. However, as shown in Fig. 2, SS typically requires over 30 times more oracle calls than QuickPrune. Compared to other algorithms, QuickPrune often surpasses these methods by orders of magnitude in pruning the ground set size while maintaining the same pruning approximation ratio. For example, QuickPrune significantly outperforms GCOMB-P in pruning efficiency for MaxCover and MaxCut (we directly report the performance of GCOMB-P on these instances as reported in Ireland and Montana (2022b)).

We remark that, in contrast to our method, the learning-based approaches frequently require domain knowledge and extensive feature engineering. For instance, LeNSE uses eigenvector centrality as a node feature, which is a costly calculation for large graphs and necessitates extensive hyperparameter tuning for each dataset and problem. Similarly, COMBHelper employs problem-specific boosting, requiring domain knowledge. Nevertheless, even the best-performing machine learning approaches are less optimal than our method and require significantly higher computational overhead and running time. While LeNSE shows the lowest combined GPU and CPU memory usage among the ML approaches, QuickPrune only utilizes CPU resources, which is less than the CPU usage of LeNSE alone. Please refer to Appendix G for a detailed analysis of memory usage and runtime.

Image Retrieval System Results. Next, we present our results for the image retrieval system under size constraint. We randomly sampled five images from the query dataset and averaged the performance over ten such queries. Fig. H.2, provides an overview of the pipeline and results for the retrieval system. We compare our algorithm with SS and Random (which selects a random set of elements from the candidate dataset, with a size equal to that of QuickPrune). Notice that on the CIFAR100 dataset, SS fails to complete pruning within the three-hour cut-off time. We observe that QuickPrune outperforms Random but fails to outperform SS. Although SS performs better than our approach on the FOOD101 dataset, it returns a ground set that is 14 times larger than ours.

3.2 Set 2: Performance on Knapsack Constraint

In this subsection, we compare our algorithm with the baselines for general knapsack constraints.

Classical Problems on Graph. We set the cost of each node following Yaroslavtsev et al. (2020) (more details in Appendix C). In Table 5, we observe that QuickPrune achieves the highest combined metric on 45% of all instances tested. Specifically, for MaxCover and MaxCut, QuickPrune often outperforms the simple baseline Top-k by 20% in retaining the objective value, except for Deezer-MaxCover and Skitter-MaxCut. The Top-k approach does not account for the relationships between vertices, whereas QuickPrune considers the problem more holistically, ensuring that the selected vertices collectively form a well-balanced subgraph. While Top-k outperforms QuickPrune for IM in most cases, its poor performance across MaxCut and MaxCover shows that Top-k is not a robust approach across applications and datasets. On the other hand, GnnPruner performs quite well on some datasets but unexpectedly poorly on others.

Video Retrieval System Results. We set the cost of each video proportional to its length. For the video retrieval system, GnnPruner is not a suitable approach, as it cannot take a set of items as input. Therefore, we compare our approach with Random and Top-k approach. From Figure 5(e), we observe that QuickPrune outperforms Random by 80% while performing better than the Top-k approach.

3.3 Set 3: Multi-Budget and Single-Budget Comparison

Finally, we compare the performance of QuickPrune and QuickPrune-Single (which prunes for the maximum budget) for a range of budget. Following the multi-budget analysis in Ireland and Montana (2022b), we vary the budget b from 10 to 100. We remark that for size constraint, no significant improvement is observed for QuickPrune over QuickPrune-Single in the size-constrained experiments – we speculate that this may be at least partially explained by the efficacy of the standard greedy algorithm for the size constraint, and a greedy solution of smaller size is automatically contained in greedy solutions of larger sizes. However, for general knapsack constraint, we observe that QuickPrune typically improves by 3%-10% over QuickPrune-Single for IM (Figure 3), while only increasing the size of the ground set less than 1% percent. For MaxCover, we observe qualitatively similar improvements on some datasets; however, for MaxCut, QuickPrune and QuickPrune-Single perform similarly. We provide results for each application and dataset in Appendix H.4.

4 CONCLUDING REMARKS AND FUTURE DIRECTIONS

In this work, we introduce QuickPrune, the first theoretically principled approach to pruning large ground sets for constrained discrete maximization problems. Moreover, the theoretical guarantees extend to a range of budgets so that the pruned ground set is applicable to more than a single optimization problem. In addition to the theoretical properties, our algorithm exhibits excellent empirical performance, and outperforms pre-existing heuristics for the pruning problem. Future directions include improving the theoretical properties, as our ratio α, while constant, is rather small. Also, upper bounds on the pruning problem would shed light on the complexity of the problem. Currently, we are unaware of existing hardness results that would imply that even a ratio of $\alpha = 1$ is impossible. Finally, extending the framework to handle more than one objective function, as well as deletions to the ground set, would be an interesting avenue for future work.

References

Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in Data Subset Selection and Active Learning. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1954–1963. PMLR, June 2015.

Sun Hye Kim and Fani Boukouvala. Machine learning-based surrogate modeling for data-driven optimization: A comparison of subset selection for regression techniques. *Optimization Letters*, 14(4):989–1010, June 2020. ISSN 1862-4480. doi: 10.1007/s11590-019-01428-7.

Anay Mehrotra and Nisheeth K. Vishnoi. Maximizing Submodular Functions for Recommendation in the Presence of Biases. In *Proceedings of the ACM Web Conference 2023*, WWW '23, pages 3625–3636, New York, NY, USA, April 2023. Association for Computing Machinery. ISBN 978-1-4503-9416-1. doi: 10.1145/3543507.3583195.

Hyeyoung Ko, Suyeon Lee, Yoonseo Park, and Anna Choi. A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields. *Electronics*, 11(1):141, January 2022. ISSN 2079-9292. doi: 10.3390/electronics11010141.

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*, KDD '03, pages 137–146, New York, NY, USA, August 2003a. Association for Computing Machinery. ISBN 978-1-58113-737-8. doi: 10.1145/956750.956769.

Tianyi Zhou, Hua Ouyang, Jeff Bilmes, Yi Chang, and Carlos Guestrin. Scaling Submodular Maximization via Pruned Submodularity Graphs. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 316–324. PMLR, April 2017a.

Sahil Manchanda, AKASH MITTAL, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. GCOMB: Learning Budget-constrained Combinatorial Algorithms over Billion-sized Graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 20000–20011. Curran Associates, Inc., 2020a.

David Ireland and Giovanni Montana. LeNSE: Learning To Navigate Subgraph Embeddings for Large-Scale Combinatorial Optimisation, May 2022a.

Hao Tian, Sourav Medya, and Wei Ye. COMBHelper: A Neural Approach to Reduce Search Space for Graph Combinatorial Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18): 20812–20820, March 2024a. ISSN 2374-3468. doi: 10.1609/aaai.v38i18.30070.

A Practical Online Framework for Extracting Running Video Summaries under a Fixed Memory Budget. In Carlotta Demeniconi and Ian Davidson, editors, *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, Philadelphia, PA, January 2021. Society for Industrial and Applied Mathematics. ISBN 978-1-61197-670-0. doi: 10.1137/1.9781611976700.

Uriel Feige and Jan Vondrak. The Submodular Welfare Problem with Demand Queries. *THEORY OF COMPUTING*.

Uriel Feige and Joe Kilian. Zero Knowledge and the Chromatic Number. In *Conference on Computational Complexity*, pages 278–287, 1996.

Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing Non-Monotone Submodular Functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011a. ISSN 01386557. doi: 10.1137/090750688.

Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing Non-monotone Submodular Functions. *SIAM Journal on Computing*, 40(4):1133–1153, January 2011b. ISSN 0097-5397, 1095-7111. doi: 10.1137/090779346.

Uriel Feige and Rani Izsak. Welfare Maximization and the Supermodular Degree. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 247–256, 2013. ISBN 978-1-4503-1859-4. doi: 10.1145/2422436.2422466.

Shayan Oveis Gharan and Jan Vondrák. Submodular Maximization by Simulated Annealing. pages 1098–1116, 2010. doi: 10.1137/1.9781611973082.83.

Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011a. doi: 10.1137/1.9781611973082.83.

Shayan Oveis Gharan and Jan Vondrák. Submodular Maximization by Simulated Annealing. In *Symposium on Discrete Algorithms (SODA)*, pages 1098–1116, 2011b. ISBN 978-0-89871-993-2. doi: 10.1137/1.9781611973082.83.

Shayan Oveis Gharan and Jan Vondrák. Submodular Maximization by Simulated Annealing. In *Proceedings of the 2011 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 1098–1116. Society for Industrial and Applied Mathematics, January 2011c. ISBN 978-0-89871-993-2. doi: 10.1137/1.9781611973082.83.

Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *International Workshop on Internet and Network Economics (WINE)*, 2010.

S Khot. Improved Inapproximability Results for Max-Clique, Chromatic Number and Approximate Graph Coloring. *42Nd Annual Symposium on Foundations of Computer Science, Proceedings*, pages 600–609, 2001. ISSN 0272-5428. doi: 10.1109/SFCS.2001.959936.

Suraj Kothawade, Vishal Kaushal, Ganesh Ramakrishnan, Jeff Bilmes, and Rishabh Iyer. PRISM: A Rich Class of Parameterized Submodular Information Measures for Guided Subset Selection, March 2022.

Jon Lee, Vahab Mirrokni, Viswanath Nagarjan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints, February 2009.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, December 1978a. ISSN 1436-4646. doi: 10.1007/BF01588971.

Gilad Ben Uziahu and Uriel Feige. On Fair Allocation of Indivisible Goods to Submodular Agents, March 2023.

Jan Vondrák. Symmetry and Approximability of Submodular Maximization Problems. *SIAM Journal on Computing*, 42(1):265–304, January 2013. ISSN 0097-5397. doi: 10.1137/110832318.

Ethan R. Elenberg, Rajiv Khanna, Alexandros G. Dimakis, and Sahand Negahban. Restricted strong convexity implies weak submodularity. *Annals of Statistics*, 46(6B):3539–3568, 2018. ISSN 00905364. doi: 10.1214/17-AOS1679.

Krishnateja Killamsetty, Alexandre V. Evfimievski, Tejaswini Pedapati, Kiran Kate, Lucian Popa, and Rishabh Iyer. MILO: Model-Agnostic Subset Selection Framework for Efficient Model Training and Tuning, June 2023.

Vladimir Braverman, Vincent Cohen-Addad, H.-C. Shaofeng Jiang, Robert Krauthgamer, Chris Schwiegelshohn, Mads Bech Toftrup, and Xuan Wu. The Power of Uniform Sampling for Coresets. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 462–473, October 2022. doi: 10.1109/FOCS54457.2022.00051.

Dan Feldman. Introduction to Core-sets: An Updated Survey, November 2020.

Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, pages 100–108, New York, NY, USA, June 2014. Association for Computing Machinery. ISBN 978-1-4503-2375-8. doi: 10.1145/2594538.2594560.

Dmitry Kogan and Robert Krauthgamer. Sketching Cuts in Graphs and Hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS '15, pages 367–376, New York, NY, USA, January 2015. Association for Computing Machinery. ISBN 978-1-4503-3333-7. doi: 10.1145/2688073.2688093.

Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. Graph Summarization Methods and Applications: A Survey. *ACM Computing Surveys*, 51(3): 1–34, May 2019. ISSN 0360-0300, 1557-7341. doi: 10.1145/3186727.

Vahab Mirrokni and Morteza Zadimoghaddam. Randomized Composable Core-sets for Distributed Submodular Maximization. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 153–162, New York, NY, USA, June 2015. Association for Computing Machinery. ISBN 978-1-4503-3536-2. doi: 10.1145/2746539.2746624.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for Data-efficient Training of Machine Learning Models. In *Proceedings of the 37th International Conference on Machine Learning*, pages 6950–6960. PMLR, November 2020.

Murad Tukan, Alaa Maalouf, and Dan Feldman. Coresets for Near-Convex Functions. In *Advances in Neural Information Processing Systems*, volume 33, pages 997–1009. Curran Associates, Inc., 2020.

Murad Tukan, Loay Mualem, and Alaa Maalouf. Pruning Neural Networks via Coresets and Convex Geometry: Towards No Assumptions. *Advances in Neural Information Processing Systems*, 35:38003–38019, December 2022.

Yu Yang, Hao Kang, and Baharan Mirzasoleiman. Towards Sustainable Learning: Coresets for Data-efficient Deep Learning. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Guangyi Zhang, Nikolaj Tatti, and Aristides Gionis. Coresets remembered and items forgotten: Submodular maximization with deletions. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 676–685, November 2022. doi: 10.1109/ICDM54844.2022.00078.

Juho Lauri and Sourav Dutta. Fine-Grained Search Space Classification for Hard Enumeration Variants of Subset Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):2314–2321, July 2019. ISSN 2374-3468. doi: 10.1609/aaai.v33i01.33012314.

Juho Lauri, Sourav Dutta, Marco Grassia, and Deepak Ajwani. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *Journal of Heuristics*, 29(2):313–347, June 2023. ISSN 1572-9397. doi: 10.1007/s10732-023-09512-z.

Yuan Sun, Andreas Ernst, Xiaodong Li, and Jake Weiner. Generalization of machine learning for problem reduction: A case study on travelling salesman problems. *OR Spectrum*, 43(3):607–633, September 2021a. ISSN 1436-6304. doi: 10.1007/s00291-020-00604-x.

Yuan Sun, Xiaodong Li, and Andreas Ernst. Using Statistical Measures and Machine Learning for Graph Reduction to Solve Maximum Weight Clique Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1746–1760, May 2021b. ISSN 1939-3539. doi: 10.1109/TPAMI.2019.2954827.

Jiwei Zhang and Deepak Ajwani. Learning to Prune Instances of Steiner Tree Problem in Graphs, October 2022.

Andrew An Bian, Joachim M. Buhmann, Andreas Krause, and Sebastian Tschiatschek. Guarantees for Greedy Maximization of Non-submodular Functions with Applications. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Alan Kuhnle, J. David Smith, Victoria G. Crawford, and My T. Thai. Fast Maximization of Non-Submodular, Monotonic Functions on the Integer Lattice. In *International Conference on Machine Learning (ICML)*, 2018.

David Ireland and Giovanni Montana. Lense: Learning to navigate subgraph embeddings for large-scale combinatorial optimisation. In *International conference on machine learning*, pages 9622–9638. PMLR, 2022b.

Tianyi Zhou, Hua Ouyang, Jeff Bilmes, Yi Chang, and Carlos Guestrin. Scaling submodular maximization via pruned submodularity graphs. In *Artificial Intelligence and Statistics*, pages 316–324. PMLR, 2017b.

Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33:20000–20011, 2020b.

Hao Tian, Sourav Medya, and Wei Ye. Combhelper: A neural approach to reduce search space for graph combinatorial problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20812–20820, 2024b.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1–20, 2016.

Makerere AI Lab. Bean disease dataset, January 2020. URL `https://github.com/AI-Lab-Makerere/ibean/`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012. URL `https://arxiv.org/abs/1212.0402`.

Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1539–1554, 2015.

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.

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, 2003b.

Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208, 2009.

Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*, pages 722–754. PMLR, 2021.

Huy Nguyen and Rong Zheng. On budgeted influence maximization in social networks. *IEEE Journal on Selected Areas in Communications*, 31(6):1084–1094, 2013.

George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978b.

Samir Khuller, Anna Moss, and Joseph Seffi Naor. The budgeted maximum coverage problem. *Information processing letters*, 70(1):39–45, 1999.

Canh V Pham, Tan D Tran, Dung TK Ha, and My T Thai. Linear query approximation algorithms for non-monotone submodular maximization under knapsack constraint. *arXiv preprint arXiv:2305.10292*, 2023.

Checklist

1. For all models and algorithms presented, check if you include:

   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]

   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]

   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:

   (a) Statements of the full set of assumptions of all theoretical results. [Yes]

   (b) Complete proofs of all theoretical results. [Yes]

   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:

   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   All parameter settings and implementation choices for the evaluated algorithms are reported. Further, the source code and documentation is provided, with instructions for how to reproduce the results.

   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]

   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]

   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

   (a) Citations of the creator If your work uses existing assets. [Yes]

   (b) The license information of the assets, if applicable. [Yes]

   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]

   (d) Information about consent from data providers/curators. [Not Applicable]

   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

   (a) The full text of instructions given to participants and screenshots. [Not Applicable]

   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A Proofs for Section 2

Proof of Prop. 1. Let $m^* = \left(1 + \frac{\kappa}{\delta c_{min}}\right)\log(n/\varepsilon) + 1$.

First, we show that at any time during the execution of Alg. 1, $|A| \leq m^* + |A_s|$. Fix a value B of A_s, and let $A_i, 1 \leq i \leq m$ assume all distinct values of A while A_s has value B, in the order in which they were assigned. Thus, $A_1 = A_s$, $A_2 = A_s + e_1, \ldots, A_m = A_s + e_1 + \ldots + e_{m-1}$. Then, let $(y_i = f(A_i))_{i=1}^m$. By Line 7, $y_i \geq \left(1 + \frac{\delta c_{min}}{\kappa}\right) y_{i-1}$, for each $i \in [m]$. Let $\beta = \frac{\delta c_{min}}{\kappa}$; then by Fact 1, if $i \geq m^* - 1$, then $y_i \geq \frac{n}{\varepsilon} y_1$, and a deletion is triggered on Line 11, which changes the value of A_s. Hence, we have $m \leq m^*$, which shows $|A| \leq |A_s| = m^*$.

Observe that after a deletion, $|A_s| = m \leq m^*$. Since A_s has initial value $\emptyset$, it holds that $|A_s| \leq m^*$ throughout the execution of the algorithm. Therefore, $|A| \leq 2m^*$, and thus $|\mathcal{U}'| \leq 2m^* + 1$. □

Proof of Prop. 2. Let $(\dot{B}_i)_{i=1}^m$ be the sequence of sets deleted by the algorithm on Line 12, in the order in which they were deleted. Let $j(i)$ be the iteration of the **for** loop on which B_i was deleted, and let $A_{j(i)}$ be the value of the set A at the beginning of iteration $j(i)$. Let $B_0 = \hat{A}$, and let $B_i = B_{i-1} \setminus \hat{B}_i$, for $i = 1$ to m. Observe that $B_m = \dot{A}$.

We have that

$$\begin{aligned} f(B_{i-1}) - f(B_i) &\overset{(a)}{\leq} \gamma^{-1} f(\dot{B}_i) \\ &\overset{(b)}{<} \frac{\gamma^{-1}\varepsilon}{n} f(A_{j(i)}) \\ &\overset{(c)}{\leq} \frac{\gamma^{-1}\varepsilon}{n} f(\hat{A}), \end{aligned}$$

where Inequality (a) follows from γ-submodularity, Inequality (b) is from the condition to delete set $\dot{B}_i$ on Line 11, and Inequality (c) follows from monotonicity. From here,

$$\begin{aligned} f(\hat{A}) - f(\dot{A}) &= \sum_{i=1}^{m} f(B_{i-1}) - f(B_i) \\ &\leq m \cdot \frac{\gamma^{-1}\varepsilon}{n} f(\hat{A}) \leq \gamma^{-1}\varepsilon f(\hat{A}). \end{aligned}$$ □

Proof of Lemma 1. If $c(A_m) \leq \kappa$, let $A^* = A_m$ and there is nothing to show. Otherwise, let $A' = \{a_{i'}, \ldots, a_m\}$, where $i' = \max_{i \in [m]}\{i : c(\{a_i, \ldots, a_m\}) > \kappa\}$. We have

$$\begin{aligned} f(A') &\overset{(a)}{\geq} \gamma(f(A_m) - f(A_m \setminus A')) \\ &\overset{(b)}{=} \gamma \sum_{i=i'}^{m} \Delta\left(a_i | A_{i-1}\right) \\ &\overset{(c)}{\geq} \gamma^2 \sum_{i=i'}^{m} \frac{\delta c(a_i) f(A_{i-1})}{\kappa} \\ &\overset{(d)}{>} \gamma^2 \delta f(A_{i'-1}) = \gamma^2 \delta f(A \setminus A'), \end{aligned}$$

where Inequality (a) follows from γ-submodularity and nonnegativity, Equality (b) is a telescoping sum, Inequality (c) is from the assumed condition (2), and Inequality (d) is from monotonicity and the fact that $c(A') > \kappa$. Thus

$$\begin{aligned} f(A_m) &\leq \gamma^{-1}(f(A') + f(A_m \setminus A')) \\ &< \gamma^{-1}(1 + 1/(\delta\gamma^2)) f(A') = \frac{\delta\gamma^2 + 1}{\delta\gamma^3} f(A'). \end{aligned}$$

Let $A'' = A' \setminus a_{i'}$; by choice of A', it holds that $c(A'') \leq \kappa$. Finally, from γ-submodularity,

$$\begin{aligned} f(a^*) + f(A'') &\geq f(a_{i'}) + f(A'') \\ &\geq \gamma f(A') \geq \frac{\delta\gamma^4}{\delta\gamma^2 + 1} f(A_m). \end{aligned}$$

Thus, set $A^* = \arg\max\{f(a^*), f(A'')\}$. □

Proof of Prof. 3. Let $\dot{A} = \{\dot{a}_1, \ldots, \dot{a}_m\}$ be in the order in which the elements were added into set A. That is, if $j(i)$ the iteration of the **for** loop in which $\dot{a}_i$ is considered, $i < i'$ implies $j(i) < j(i')$. Let A_j be the value of the set A at the beginning of iteration j of the **for** loop. Then

$$\begin{aligned} \Delta\left(\dot{a}_i | \dot{A}_{i-1}\right) &\overset{(a)}{\geq} \gamma \Delta\left(\dot{a}_i | A_{j(i)}\right) \\ &\overset{(b)}{\geq} \gamma \frac{\delta c(\dot{a}_i) f(A_{j(i)})}{\kappa} \\ &\overset{(c)}{\geq} \gamma \frac{\delta c(\dot{a}_i) f(\dot{A}_{i-1})}{\kappa}, \end{aligned}$$

where (a) follows from γ-submodularity since $\dot{A}_{i-1} \subseteq A_{j(i)}$, (b) follows from the condition on Line 7, (c) follows from monotonicity. Also $c(\dot{a}_i) \leq \kappa$, for all i. Therefore, $\dot{A}_i$ and a^* satisfy the conditions of Lemma 1, which implies the result. □

Proof of Prop. 4. Let $O \subseteq \mathcal{U}$, such that $c(O) \leq \kappa$ and $f(O) = \text{OPT}$. For each $e \in \mathcal{U}$, let $j(e)$ be the iteration of the **for** loop in which e was considered, and let A_j be the value of the set A at the beginning of the jth iteration. Then

$$\begin{aligned} \text{OPT} - f(\hat{A}) &\overset{(a)}{\leq} f(O \cup \hat{A}) - f(\hat{A}) \\ &\overset{(b)}{\leq} \gamma^{-1} \sum_{o \in O \setminus \hat{A}} \Delta\left(o | A_{j(o)}\right) \\ &\overset{(c)}{<} \gamma^{-1} \sum_{o \in O \setminus \hat{A}} \frac{\delta c(o) f(A_{j(o)})}{\kappa} \\ &\overset{(d)}{\leq} \gamma^{-1} \sum_{o \in O \setminus \hat{A}} \frac{\delta c(o) f(\hat{A})}{\kappa} \\ &\overset{(e)}{\leq} \gamma^{-1} \delta f(\hat{A}), \end{aligned}$$

where (a) follows from monotonicity, (b) follows from γ-submodularity, (c) holds since the condition on Line 7 must have failed since $o \notin \hat{A}$, (d) follows from monotonicity, and (e) holds since $\sum_{o \in O} c(o) \leq \kappa$. □

Proof of Prop. 5. Let $X \subseteq O$ be a maximal subset with $c(X) \leq \kappa(1-\eta)$. By Assumption NHI, $X \neq \emptyset$. If $X = O$, there is nothing to show. Otherwise, let $o \in O \setminus X$. By definition, $c\,(O \setminus (X \cup \{o\}) < \kappa - \kappa(1-\eta) = \eta\kappa < (1-\eta)\kappa$, since $\eta \leq 1/2$. Therefore, $O_1 = X$, $O_2 = \{o\}$, and $O_3 = O \setminus (X \cup \{o\})$ form the requisite partition. □

B Experimental Setup

We run all our experiments on a Linux server equipped with an NVIDIA RTX A6000 GPU and an AMD EPYC 7713 CPU, using PyTorch 2.4.1 and Python 3.12.7. Code and data are available at: `https://github.com/ankurnath/QuickPrune`.

C Problem Formulation

In this section, we formally introduce the four applications mentioned in the paper. A CO problem on graph can be described with an undirected graph $G(V, E)$, where V is the set of vertices, E is the set of edges and each node $v \in V$ is associated with a cost $c(v)$. For knapsack constraint, we set the cost of each node $v \in V$ as $c(v) = \frac{\beta}{|V|}(|N(v)| - \alpha)$ where $N(v)$ is the set of neighbors of v , $\alpha = \frac{1}{20}$ and β is a normalizing factor so $c(v) \geq 1$, so that the cost of each node is roughly proportional to the value of the node following Yaroslavtsev et al. (2020).

Maximum Cover. Given a budget k , the goal of this problem is to find a subset of nodes $S \subseteq V$ that maximizes the objective function, $f(S) = |\{v | v \in S \vee \exists (u, v) \in E, u \in S\}|$ where $\sum_{v \in S} c(v) \leq k$.

Maximum Cut (MaxCut). Given a budget k, the goal of this problem is to find a subset of nodes $S \subseteq V$ that maximizes the objective function, $f(S) = |\{(u, v) \in E : v \in S, u \in V \setminus S\}|$ where $\sum_{v \in S} c(v) \leq k$.

Influence Maximization (IM). Given a budget k, probabilities $p(u, v)$ on the edges $(u, v) \in E$ and a cascade model $\mathcal{C}$, the goal of this problem is to find a subset of nodes $S \subseteq V$ that maximizes the expected spread, $f(S) = \mathbf{E}[\sigma(S)]$ where $\sigma(S)$ denotes the spread of S and $\sum_{v \in S} c(v) \leq k$. For the experiments, we consider the independent cascade model (Kempe et al., 2003b) and set $p = 0.01$ following Chen et al. (2009). Because relatively large propagation probability p the influence spread is not very sensitive to different algorithms and heuristics, because a giant connected component exists even after removing every edge with probability $1 - p$.

Information Retrieval System. Given a budget k, a set of candidate items C (images or videos), and a set of query items Q, the objective is to find a subset $S \in C$ that maximizes the graph cut function, $f(S) = \lambda \sum_{i \in Q} \sum_{j \in S} s(i, j) - \sum_{i, j \in S} s(i, j)$ where s is the similarity kernel and $\lambda \geq 2$. We set $\lambda = 10$ for our experiments. Note that the condition on λ is to ensure that f remains a monotone submodular function Iyer et al. (2021). For video recommendation, the constraint is $\sum_{v \in S} L(v) \leq k$, where $L(v)$ represents the length of a video normalized the minimum length of the video. For the image retrieval system, the constraint is simply $|S| \leq k$.

D Baselines

In this section, we provide details about each algorithm discussed in our paper.

GCOMB-P. The idea is as follows: For a graph $G_i = (V, E)$ from training distribution and a budget of b, the vertices are initially sorted into descending order based on the sum of outgoing edge-weight (or degree in an unweighted graph), and $rank(v)$ denotes the position of vertex v in this ordered list. A stochastic solver is used m times to obtain m different solution sets $\{S^{(1)}, S^{(2)}, ..., S^{(m)}\}$ for budget b. The goal here is to predict all nodes that could potentially be part of the solution set. We define $r_b^{G_i} = \max_{v \in \cup_j S^{(j)}} rank(v)$ to be the highest rank of all vertices among the vertices in each of the m solution sets. We repeat this process for each graph in the training distribution and define $r_b = \max_{G_i \in G_{train}} r_b^{G_i}$ for budget b. To generalize across budgets, we compute r_b for a series of budgets and obtain (b, r_b) pairs. To generalize across graph sizes, budgets and ranks are normalized with respect to the number of nodes in the graph. During testing, for an unseen budget b, linear interpolation is used to predict r_b.

LeNSE. Ireland and Montana (2022b) proposed extracting a small portion of the original graph that most likely contains the optimal solution, where the solution can be extracted using any existing heuristics. The idea is as follows: Given a training distribution, extract the optimal solutions for each graph. Next, generate random subgraphs containing different portions of nodes from the optimal solution, where subgraphs are sets of nodes and their 1-hop neighbors. Assign these subgraphs to different classes based on the ratio of the objective function value from the subgraph to that of the original graph. The number of classes depends on the problem and dataset and usually requires domain knowledge. In the next step, a reinforcement learning agent is trained to navigate a subgraph induced by a fixed number of random nodes and their 1-hop neighbors, updating the subgraph at each step by replacing a single vertex with its neighbors. By updating the subgraph in this manner, the agent aims to obtain a subgraph close to the optimal subgraph in the embedding space created by the encoder.

Submodular Sparsification (SS). Zhou et al. (2017b) proposed a randomized pruning method to reduce the submodular maximization problem via a novel concept called the submodularity graph. The submodularity graph is a weighted directed graph $G(V, E, w)$ defined by a normalized submodular function $f : 2^V \to \mathcal{R}_+$, where V is the set of nodes corresponding to the ground set, and each directed edge $(u, v) \in E$ has weight $w(u, v) = f(v \mid u) - f(u \mid V \setminus u)$. As computing all edge weights is quadratic in complexity, they proposed a randomized approach. The idea is as follows: Sample a subset of random nodes from the original ground set, remove these random nodes from the ground set at each step, and add them to the pruned ground set. Then, remove a subset of the top elements from the ground set that are deemed unimportant from the sample nodes in that set. After several stages of pruning, when the original ground set size falls below a certain threshold, the remaining elements are merged with the pruned ground set.

COMBHelper. Tian et al. (2024b) introduced a method to train a GNN for vertex classification in combinatorial problems, predicting nodes likely to be part of the solution. To improve scalability, they applied knowledge distillation to transfer knowledge to a smaller GNN and used problem specific weight boosting to enhance the performance.

GNNPruner. We propose a modified version of COMBHelper, in which we similarly train a vertex GNN classifier to predict solution nodes. Like COMBHelper and LeNSE, we generate training labels using solutions from a heuristic. However, unlike these methods, we use random numbers as node features for experiments under size constraints, and replace SAGEConv (Hamilton et al., 2017) with a lightweight GCN (Kipf and Welling, 2016) with fewer layers. Empirical evidence shows that this architectural shift helps capture structural relationships in the graph without relying on the handcrafted features used by COMBHelper and LeNSE. Additionally, it improves scalability by using fewer layers. We simplify prepossessing and improve prediction accuracy through more generalized learning by leveraging random features (shown in Table 1). However, in experiments involving knapsack constraints, instead of using random numbers as node features, we provide the degree, cost, and the degree-to-cost ratio of each node as input features. This adjustment appears to improve performance.

E Dataset Information

In Table 2, we list the graphs used in our empirical evaluations along with the sizes of the respective training and testing splits for the traditional CO problems on graphs. We follow Ireland and Montana (2022b) to determine what percentage of the original graph edges was used for training and testing.

Table 2: The real-world graphs used to perform our experiments.

	Train Size		Test Size	
Graph	Vertices	Edges	Vertices	Edges
Facebook	3847	26470(30%)	4002	61764
Wiki	4891	30228(30%)	6358	70534
Deezer	48870	149460(30%)	53511	348742
SlashDot	47546	140566(30%)	67640	327988
Twitter	55827	134229(10%)	80712	1208067
DBLP	63004	41994(10%)	315305	1007872
YouTube	185193	179257(06%)	1098104	2808367
Skitter	147604	110952(01%)	1694318	10984346

For the information retrieval problem, we use the Beans dataset (Lab, 2020), CIFAR100 (Krizhevsky, 2009), FOOD101 (Bossard et al., 2014), and UCF101 (Soomro et al., 2012). Since the Beans dataset contains only three classes, we undersample one class to make the problem more challenging and use that class for querying. For the FOOD101 dataset, we sample 100 images per food category for our candidate set and query from the test dataset. Similarly, for the UCF-101 dataset, we select 100 videos for each action and randomly query from the validation dataset.

F Network architectures and hyper-parameters

QuickPrune. For all graph problem experiments, we set $\delta = 0.1$, $\epsilon = 0.1$, and $\eta = 0.5$. In the knapsack-constrained MaxCover experiment, we set $\delta = 0.5$. For both image and video retrieval systems, we use $\delta = 0.05$, $\epsilon = 0.1$, and $\eta = 0.5$.

SS. For all experiments, we set $r = 8$ and $c = 8$, following Zhou et al. (2017b). We find that this configuration empirically performs well.

GNNPruner. The architecture consists of two layers, each containing a graph convolutional layer with ReLU activation and 16 hidden channels. We use the Adam optimizer with a learning rate of 0.001 and a weight decay

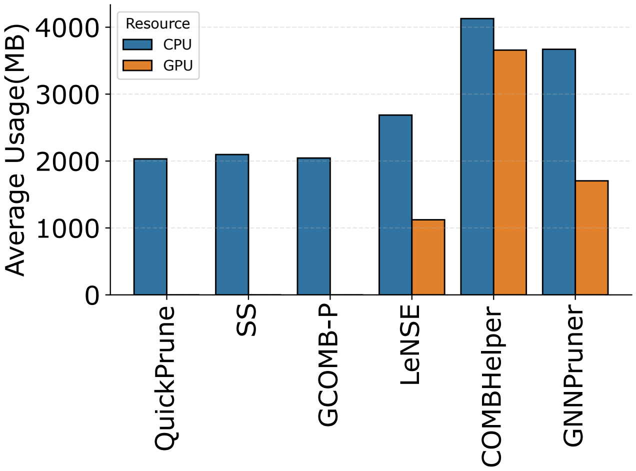


Figure 4: Average GPU and CPU memory utilization among algorithms for MaxCover on YouTube dataset.

Table 3: Comparison of the runtime (in seconds) of algorithms across different datasets for MaxCover.

Algorithm	Facebook	Wiki	Deezer	Slashdot	Twitter	DBLP	YouTube	Skitter
QuickPrune (OURS)	0.064	0.094	0.437	0.590	1.075	1.208	5.443	10.317
SS	3.178	5.804	41.995	58.281	120.555	168.011	769.871	2238.035
GNNpruner	1.073	1.272	6.741	6.654	17.222	15.157	48.702	162.288
CombHelper	1.071	1.195	5.997	5.964	15.914	14.644	47.352	NA
GCOMB-P	0.002	0.004	0.026	0.039	0.058	0.197	0.501	0.982
LeNSE	27.024	25.238	34.210	38.315	76.827	38.325	231.557	1593.406

of 5×10^{-4}, training with cross-entropy loss. At each epoch, we randomly sample vertices both from the solution and non-solution sets to prevent the loss from being dominated by the non-solution vertices.

G Efficiency and scalability analysis

From Figure 4 , we observe that most algorithms (QuickPrune, SS, GCOMB-P) only utilize CPU resources. Algorithms like LeNSE, COMBHelper, and GnnPruner use both CPU and GPU resources, with COMBHelper showing the highest combined usage, making it the most computationally demanding algorithm , followed by GNNPruner; in contrast, LeNSE exhibits relatively minimal usage. LeNSE only modifies a subgraph, so it does not need to load the entire graph into the memory.

We also report the runtime for MaxCover on each dataset in Table 3. For other constraints and applications, the results are qualitatively similar.

H Additional table and plots

In this section, you can find the tables and plots omitted in the main paper due to space constraints.

H.1 Heuristics

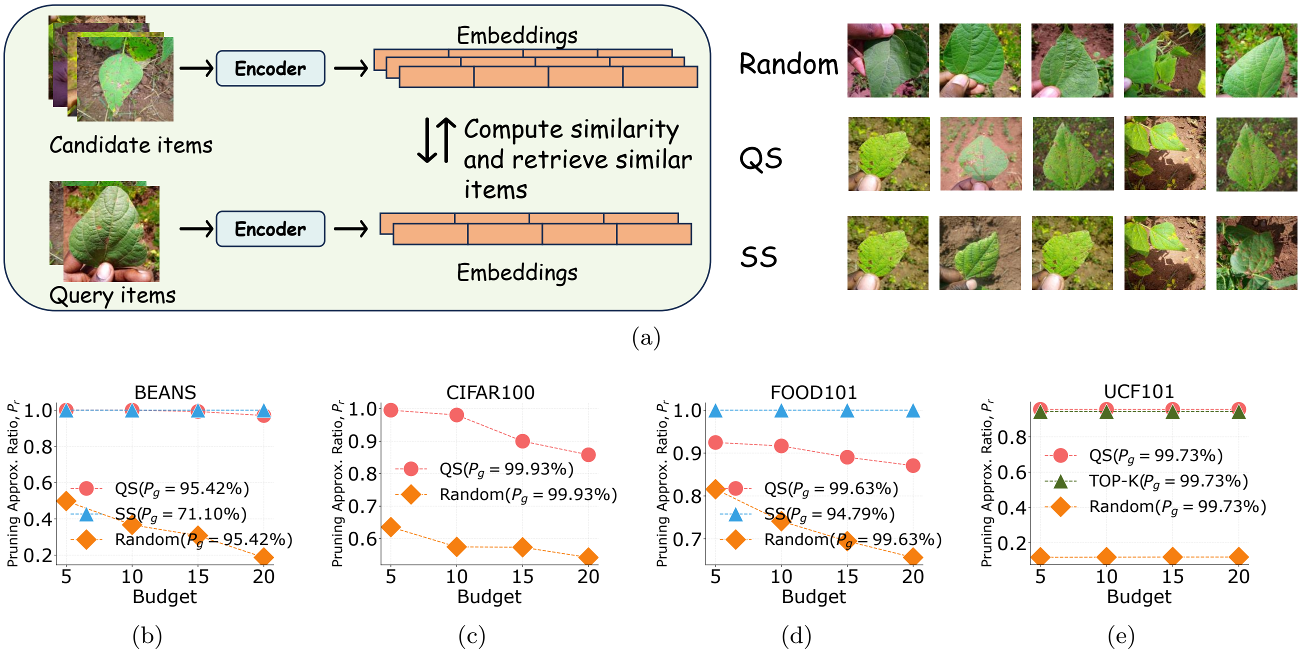
In Table 2, we summarize the algorithms used for each application under size and knapsack constraints. Note that for the knapsack-constrained experiments for IM, we use the improved greedy heuristic (referred to as Algorithm 2 in the original paper) from Nguyen and Zheng (2013).

Table 4: Summary of Heuristic Approaches

Problem	Size	Knapsack
MaxCover	Nemhauser et al. (1978b)	Khuller et al. (1999)
MaxCut	Nemhauser et al. (1978b)	Pham et al. (2023)
IM	Tang et al. (2015)	Nguyen and Zheng (2013)
Retrieval	Nemhauser et al. (1978b)	Khuller et al. (1999)

H.2 Image and Video Retrieval System.

In Figure H.2, we present the results for information retrieval system. We employ the cosine similarity metric to assess the similarity between two images. Instead of using a generalist model trained on large datasets, we use a model fine-tuned on the specific dataset. This approach allows the underlying model to better understand the input images.



(a)

(b) (c) (d) (e)

Figure 5: Retrieval system: (a) The pipeline of the retrieval system and a visual representation of images selected by various algorithms for the Beans Dataset. (b-e) Multi-budget analysis of the retrieval system for budgets ranging from 5 to 20. Note that P_g represents the percentage of the ground set that has been pruned.

H.3 Knapsack constrained experiments

In Table 5, we present the results of our experiments under knapsack constraint.

H.4 Comparison between QuickPrune and QuickPrune-Single

In this section, we present the comparison between QuickPrune and QuickPrune-Single under knapsack constraint. We run QuickPrune-Single with the maximum budget in the range (in our case, this would be 100).

Table 5: Comparison of pruning algorithms for knapsack constraint experiments (best combined metric in bold).

	QuickPrune			Top-k			GnnPruner		
Graph	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$	$P_r \uparrow$	$P_g \uparrow$	$C \uparrow$
Maximum Cover									
Facebook	0.9725	0.9871	**0.9600**	0.5505	0.9871	0.5434	0.3303	0.9936	0.3282
Wiki	0.9400	0.9851	**0.9260**	0.7550	0.9851	0.7438	1.0000	0.8696	0.8696
Deezer	0.8750	0.9983	0.8735	0.9150	0.9983	0.9134	1.0000	0.9725	**0.9725**
Slashdot	0.8950	0.9988	**0.8939**	0.5700	0.9988	0.5693	1.0000	0.8325	0.8325
Twitter	0.8700	0.9987	**0.8689**	0.5950	0.9987	0.5942	1.0000	0.4783	0.4783
DBLP	0.8450	0.9997	0.8447	0.7150	0.9997	0.7148	1.0000	0.9971	**0.9971**
YouTube	0.9150	0.9999	**0.9149**	0.5200	0.9999	0.5199	1.0000	0.3626	0.3626
Skitter	0.9050	0.9999	0.9049	0.8150	0.9999	0.8149	1.0000	0.9887	**0.9887**
Maximum Cut									
Facebook	0.9899	0.9666	0.9568	1.0000	0.9666	0.9666	0.9697	0.4984	0.4833
Wiki	0.9600	0.9928	**0.9531**	0.5100	0.9928	0.5063	1.0000	0.1836	0.1836
Deezer	0.9600	0.9985	0.9586	0.8000	0.9985	0.7988	1.0000	0.9745	**0.9745**
Slashdot	0.9700	0.9991	**0.9691**	0.6700	0.9991	0.6694	1.0000	0.4395	0.4395
Twitter	0.9500	0.9996	0.9496	0.2900	0.9996	0.2899	1.0000	0.0222	0.0222
DBLP	0.9500	0.9999	0.9499	0.4300	0.9999	0.4300	1.0000	0.9988	**0.9988**
YouTube	0.9700	0.9999	**0.9699**	0.7300	0.9999	0.7299	1.0000	0.9406	0.9406
Skitter	0.9700	0.9999	0.9699	1.0000	0.9999	**0.9999**	1.0000	0.9993	0.9993
Influence Maximization									
Facebook	0.9934	0.9148	**0.9088**	0.9934	0.9148	**0.9088**	0.1173	0.9854	0.1156
Wiki	0.9660	0.8774	0.8476	1.0000	0.8774	**0.8774**	0.8280	0.8491	0.7031
Deezer	1.0000	0.9820	**0.9820**	1.0000	0.9820	**0.9820**	0.4678	0.9719	0.4547
Slashdot	0.9132	0.9868	0.9011	1.0120	0.9868	**0.9986**	0.7385	0.8403	0.6206
Twitter	0.9121	0.9986	0.9108	0.9689	0.9986	**0.9675**	0.1399	0.9999	0.1398
DBLP	1.0000	0.9958	0.9958	1.0000	0.9958	0.9958	1.0000	0.9999	**0.9999**
YouTube	0.9171	0.9994	0.9165	0.9502	0.9994	**0.9496**	0.9814	0.3299	0.3238
Skitter	0.8291	0.9999	0.8290	0.9391	0.9999	**0.9390**	0.9999	0.1399	0.1398

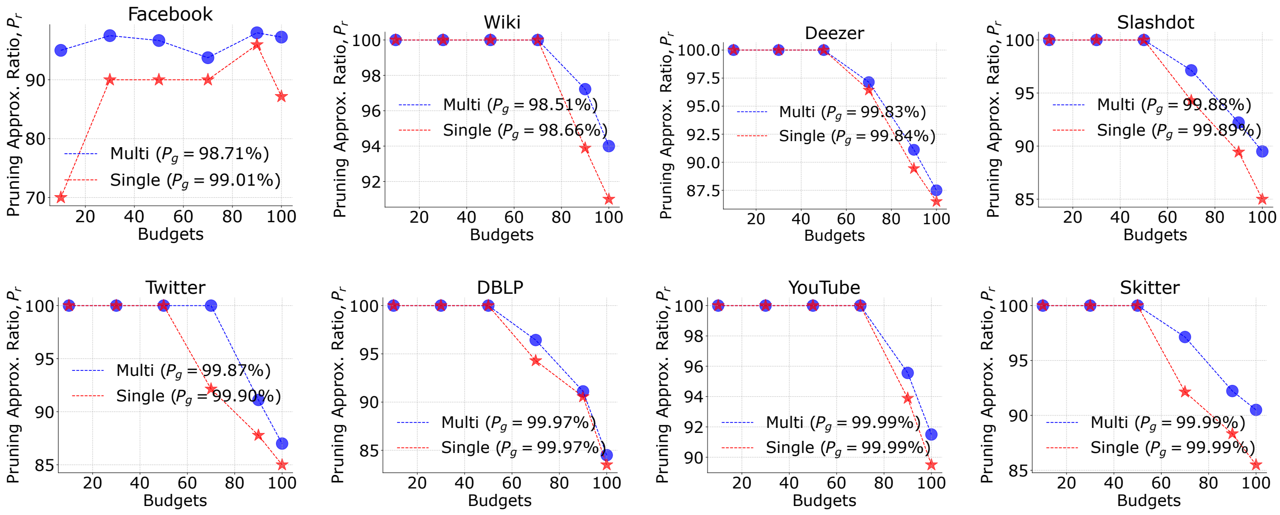


Figure 6: Multi-Budget vs Single Budget for MaxCover

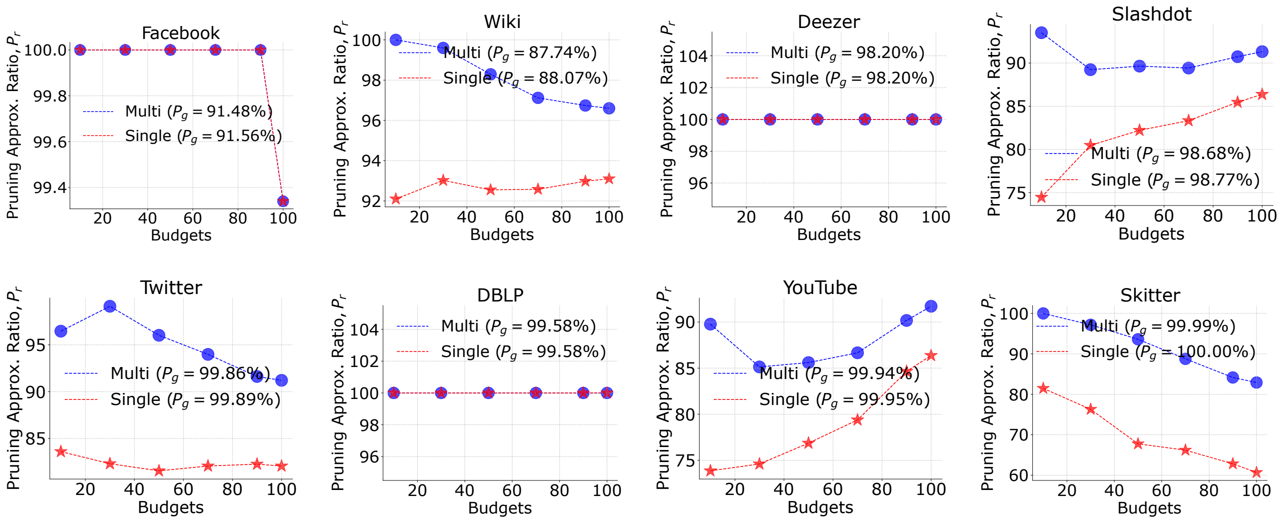


Figure 7: Multi-Budget vs Single Budget for IM

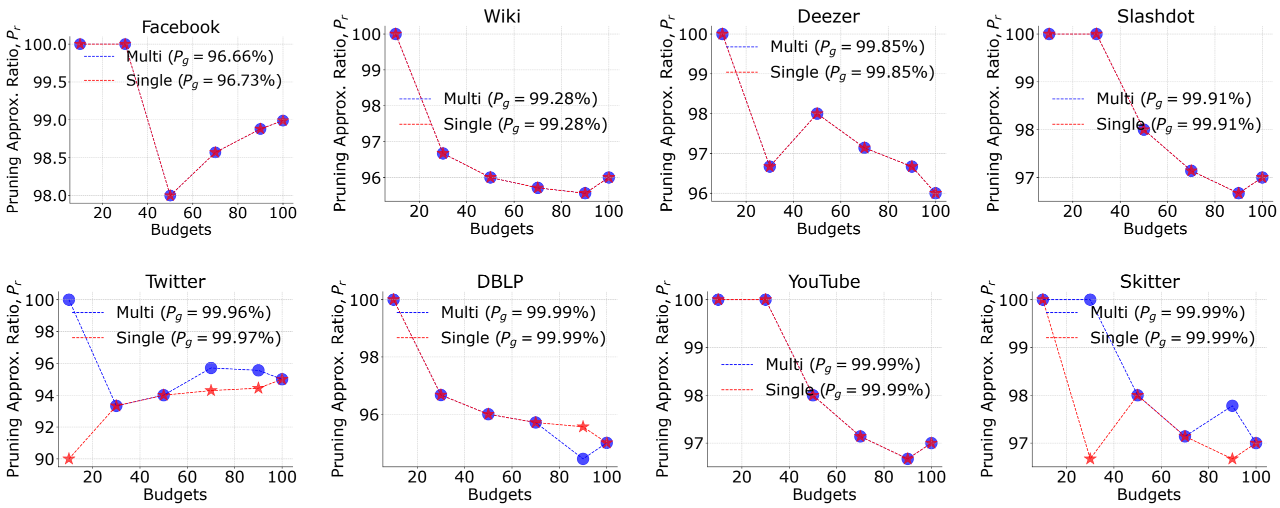


Figure 8: Multi-Budget vs Single Budget for MaxCut