Near-Optimal Sparsifiers for Stochastic Knapsack and Assignment Problems

Shaddin Dughmi
University of Southern California, Los Angeles, CA, USA

Yusuf Hakan Kalayci
University of Southern California, Los Angeles, CA, USA

Xinyu Liu
University of Southern California, Los Angeles, CA, USA

Abstract

When uncertainty meets costly information gathering, a fundamental question emerges: which data points should we probe to unlock near-optimal solutions? Sparsification of stochastic packing problems addresses this trade-off. The existing notions of sparsification measure the level of sparsity, called degree, as the ratio of queried items to the optimal solution size. While effective for matching and matroid-type problems with uniform structures, this cardinality-based approach fails for knapsack-type constraints where feasible sets exhibit dramatic structural variation. We introduce a polyhedral sparsification framework that measures the degree as the smallest scalar needed to embed the query set within a scaled feasibility polytope, naturally capturing redundancy without relying on cardinality.

Our main contribution establishes that knapsack, multiple knapsack, and generalized assignment problems admit $(1-\epsilon)$-approximate sparsifiers with degree polynomial in $1/p$ and $1/\epsilon$ – where p denotes the independent activation probability of each element – remarkably independent of problem dimensions. The key insight involves grouping items with similar weights and deploying a charging argument: when our query set misses an optimal item, we either substitute it directly with a queried item from the same group or leverage that group's excess contribution to compensate for the loss. This reveals an intriguing complexity-theoretic separation – while the multiple knapsack problem lacks an FPTAS and generalized assignment is APX-hard, their sparsification counterparts admit efficient $(1-\epsilon)$-approximation algorithms that identify polynomial degree query sets. Finally, we raise an open question: can such sparsification extend to general integer linear programs with degree independent of problem dimensions?

2012 ACM Subject Classification Theory of computation → Packing and covering problems; Theory of computation → Stochastic approximation

Keywords and phrases Packing Problems, Assignment Problems, Stochastic Selection, Sparsification

Digital Object Identifier 10.4230/LIPIcs.ITCS.2026.51

Related Version *Full Version*: `https://arxiv.org/abs/2512.01240` [14]

Funding This paper is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-24-1-0261. Any opinions, findings, and conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the United States Air Force.

1 Introduction

The sparsification of stochastic packing problems has emerged as a fundamental paradigm for designing algorithms that achieve near-optimal solutions with limited access to uncertain data. This approach proves particularly valuable in settings where probing or querying elements incurs costs or faces constraints. Recent developments in sparsification of stochastic packing problems have revealed elegant techniques for selecting the most pertinent information to probe.

Our goal in this paper is to expand the sparsification toolbox beyond well-studied matching settings [1, 3, 4, 5, 6, 7, 8, 12, 13, 23] and matroid optimization problems [15, 20, 23] to design sparsifiers for knapsack-type constraints.

We adopt the general framework of Dughmi et al. [15] for a packing problem instance $\langle E, \mathcal{F}, f \rangle$, where E is a ground set of elements, $\mathcal{F} \subseteq 2^E$ is a downward-closed family of feasible sets, and $f : 2^E \to \mathbb{R}_{\geq 0}$ is an objective function. In our stochastic setting, each element $e \in E$ becomes *active* independently with probability $p \in (0, 1]$, resulting in a random active set $R \subseteq E$. A *sparsification algorithm* selects a query set $Q \subseteq E$ prior to observing R, aiming to maximize the solution value within the revealed subset $Q \cap R$.

For assignment problems like MKP and GAP, solutions are typically defined as sets of item-knapsack pairs (assignments) rather than simple subsets of items. To align with our sparsification framework, we project these problems onto the ground set of items E. We say a subset of items $S \subseteq E$ is *feasible* ($S \in \mathcal{F}$) if there exists a valid assignment of items in S to knapsacks that respects all capacity constraints. Accordingly, we extend the objective function to item sets by defining $f(S)$ as the maximum value achievable by any feasible assignment of the items in S.

To rigorously evaluate sparsification algorithms, we must balance approximation quality against the "size" of the query set. While approximation is defined as usual, quantifying the size of Q for knapsack-type problems requires nuance. Traditional cardinality-based measures – which normalize $|Q|$ by the rank of $\mathcal{F}$ – fail when feasible sets vary dramatically in size, as in capacity-constrained problems. To address this, we introduce the *polyhedral sparsification degree*. Let $\mathcal{P}_{\mathcal{F}} = \text{conv}(\{\mathbf{1}_S : S \in \mathcal{F}\})$ be the polytope of feasible solutions. We define the degree of a query set Q as the minimum scalar $d \geq 1$ such that the normalized indicator vector $\frac{1}{d}\mathbf{1}_Q$ lies within $\mathcal{P}_{\mathcal{F}}$. This definition naturally generalizes existing notions[1] by capturing the intuitive notion of redundancy: a query set has degree d if it can be decomposed into d (fractionally) feasible solutions.

▶ **Definition 1** (Sparsifier). *An algorithm $\mathcal{A}$ is an* α-approximate sparsifier with degree d *if, for every problem instance, it returns a (possibly randomized) query set $Q \subseteq E$ that satisfies two conditions:*

1. Polyhedral Feasibility: *the indicator vector of the query set lies within the polytope scaled by d, in other words $\frac{1}{d} \cdot \mathbf{1}_Q \in \mathcal{P}_{\mathcal{F}}$ (holding for every realization of Q);*
2. Approximation Guarantee: *the expected value of the optimal solution within the queried active elements approximates the full-information optimum, such that*

$$\mathbb{E}_{R,Q}[\max\{f(S) : S \in \mathcal{F} \cap 2^{Q \cap R}\}] \geq \alpha \cdot \mathbb{E}_R[\text{OPT}].$$

1.1 Our contributions

Our main result establishes that knapsack-type problems admit effective sparsification despite their computational hardness. We design deterministic and non-adaptive algorithms that produce $(1 - \epsilon)$-approximate sparsifiers with polyhedral degree polynomial in $1/p$ and $1/\epsilon$, remarkably independent of the number of items or constraints. Unlike some prior work [20, 23], we require our degree to be independent of the total number of variables and constraints.[2]

[1] Sparsification in stochastic integer packing was introduced by Blum et al. [8] using vertex degree, and later generalized by Maehara and Yamaguchi [20, 23] to cardinality-based measures.

[2] To be clear, our definition of degree implicitly allows the number of queries to scale with the cardinality of a typical solution. Our pursuit of constant degree (independent of the number of items or constraints) is akin to requiring "redundancy" on the order of a constant number of solutions, as a hedge against uncertainty.

▶ **Informal Theorem** (Main Result). *For parameters $\epsilon \in (0, 1/6)$ and $p \in (0, 1]$, there exist efficient algorithms that produce a $(1 - O(\epsilon))$-approximate sparsifier for the stochastic Knapsack, Multiple Knapsack, and Generalized Assignment Problems with sparsification degree poly$(1/\epsilon, 1/p)$.*

For the single knapsack problem, our approach employs a bucketization strategy combined with a charging argument. Items are partitioned into buckets based on value, and each bucket is filled to approximately poly$(1/p, 1/\epsilon)$ times the knapsack capacity. This redundancy allows for a substitution mechanism: if an optimal item is not queried, it can likely be replaced by a queried item from the same bucket. Feasibility is maintained by prioritizing items with smaller weights, with a refined density-based strategy for the smallest value bucket.

Extending this to the Generalized Assignment Problem (GAP) presents a fundamental obstacle: item characteristics are knapsack-dependent, breaking the direct substitutability essential to the single knapsack case. An item ideal for one knapsack may be highly inefficient for another. We overcome this by increasing redundancy – filling buckets to poly$(1/p, 1/\epsilon) \cdot (1/\epsilon)$ capacity – and developing a sophisticated charging argument. When an optimal item cannot be directly substituted (often because potential substitutes are assigned elsewhere), we fractionally distribute its lost value across multiple queried elements. This ensures that we recover nearly the full optimal value without overburdening any specific element in the analysis.

1.2 Implications and experiments

We turn to the deterministic setting (i.e., $p = 1$) to reflect on complexity-theoretic implications and practical impact. Our results reveal an intriguing separation: while GAP is APX-hard and MKP lacks an FPTAS, their stochastic counterparts admit efficient $(1 - \epsilon)$-approximate sparsifiers with polynomial degree. From the Exponential Time Hypothesis [18], one would informally expect that the "hard" instances are already sparse, whereas "easy" instances may be more dense and amenable to sparsification. It is on those dense non–worst-case instances where our sparsifiers shrink the search space. This can serve as a useful preprocessing step for heuristic algorithms, guiding them to pertinent variables, even in the deterministic setting with $p = 1$.

Empirically, we validate the utility of our approach on synthetic datasets, using practical choices of hyperparameters $(\alpha, \tau, \epsilon, K)$ that are less conservative than the theoretical settings needed to ensure worst-case, high-probability guarantees. These practical choices result in sparsifiers with significantly smaller degree than the theoretical degree guaranteed by our theorem, and on instances where the total number of items exceeds the optimal solution size by a factor of four, our sparsification algorithm reduces runtime by 4× while preserving 99% of the solution quality. Furthermore, under fixed time budgets, branch-and-bound algorithms running on our sparsified instances outperform those running on full datasets by a factor of 5 in objective value. We present a summary of these results in Figure 1 and refer the reader to the full version of this paper for a detailed discussion [14].

1.3 Future directions

Finally, our work motivates a broader question regarding integer linear programs (ILPs). Current sparsifiers for general ILPs [23] depend on problem dimensions or column sparsity. This leaves open a fundamental challenge:

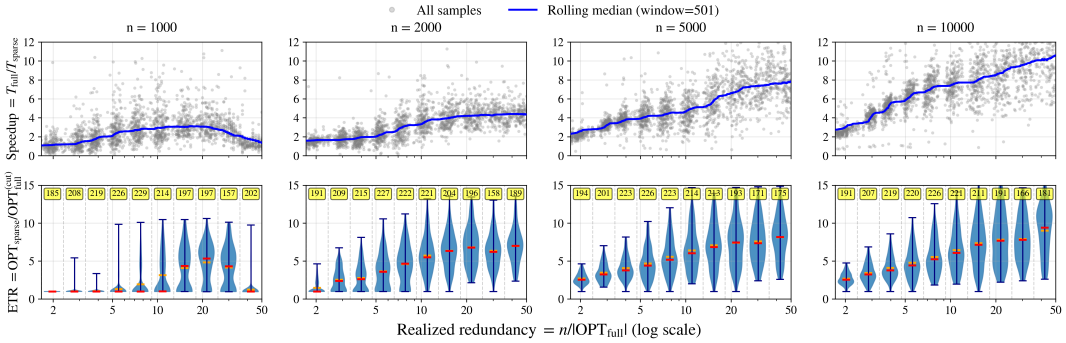


Figure 1 Performance comparison of three Gurobi-based GAP solving strategies: **(A)** full ILP; **(B)** sparsifier followed by ILP on the reduced instance; and **(C)** full ILP with early stopping matched to the runtime of (B). Experiments cover $n \in \{1000, 2000, 5000, 10000\}$ and $m = 2$. We report two metrics against *realized redundancy* (ratio of total items to optimal solution size): **(i) Speed-up Ratio:** Runtime of (A) divided by (B), where (B) maintains a ≥ 0.99-approximation; and **(ii) Efficiency Ratio:** Objective value of (B) divided by (C). The **top row** presents a scatter plot of speed-up ratios for individual instances overlaid with a rolling median. The **bottom row** illustrates the distribution of efficiency ratios within redundancy buckets, annotated with bucket medians and means as well as sample counts.

▶ **Open Question.** *Can we design $(1-\epsilon)$-approximate sparsifiers for general integer linear programs where the sparsification degree scales polynomially with $1/p$, $1/\epsilon$, and intrinsic structural parameters, but remains independent of the total number of variables and constraints?*

Resolving this would significantly advance the understanding of information requirements in stochastic optimization.

1.4 Related Works

Sparsification for stochastic packing was pioneered by Blum et al. [8] for matching. This work initiated a broad sequence of improvements in approximation bounds and query complexity for stochastic matching [1, 2, 3, 6, 7, 13, 15] and stochastic vertex cover [4, 12]. Maehara and Yamaguchi generalized this framework to stochastic packing integer programs [23] and submodular maximization [20], providing non-adaptive sparsifiers whose degree, however, depended on the universe size. Subsequently, Dughmi et al. [15] achieved dimension-independent sparsifiers for matroids.

In the deterministic setting, the Knapsack problem admits fully polynomial-time approximation schemes (FPTAS), with recent algorithms achieving near-linear time [11, 17, 21]. The Multiple Knapsack Problem (MKP) is strictly harder; Chekuri and Khanna [10] demonstrated that it admits a PTAS but not an FPTAS. The Generalized Assignment Problem (GAP) is APX-hard [9], with the state-of-the-art approximation ratio of $1 - 1/e + \varepsilon$ established by Feige and Vondrák [16]. For a detailed survey of related works see full version of the paper [14].

2 Stochastic Assignment Problems

In this section, we formally define the deterministic assignment problems addressed in this work – Knapsack, Multiple Knapsack, and Generalized Assignment – and their stochastic counterparts. Throughout our discussion, we use the terms “knapsack” and “bin” interchangeably.

2.1 Problem Definitions

We begin with the classical single-bin setting and progressively generalize to multiple heterogeneous constraints.

▶ **Problem 1** (Knapsack Problem (KP01)). *Consider a single knapsack with capacity C and a collection of n items denoted by E. Each item $i \in E$ is characterized by a value v_i and a weight w_i. The objective is to select a subset of items that maximizes total value while respecting the capacity constraint:*

$$\max \sum_{i \in S} v_i \quad \textit{subject to} \quad \sum_{i \in S} w_i \leq C, \quad S \subseteq [n].$$

▶ **Problem 2** (Multiple Knapsack Problem (MKP)). *Consider m knapsacks where knapsack $j \in [m]$ has capacity C_j, and n items E where each item $i \in E$ has value v_i and weight w_i. The objective is to assign items to the knapsacks to maximize total value while ensuring no knapsack exceeds its capacity. Formally, we seek disjoint subsets $S_j \subseteq E$ for $j \in [m]$ such that:*

$$\max \sum_{j=1}^{m} \sum_{i \in S_j} v_i \quad \textit{subject to} \quad \sum_{i \in S_j} w_i \leq C_j \quad \textit{for all } j \in [m].$$

▶ **Problem 3** (Generalized Assignment Problem (GAP)). *Consider m knapsacks where knapsack $j \in [m]$ has capacity C_j, and n items E. In contrast to the previous problems, each item $i \in E$ exhibits knapsack-dependent characteristics: when assigned to knapsack j, item i contributes value v_{ij} and consumes weight w_{ij}. The objective is to find disjoint subsets $S_j \subseteq E$ for $j \in [m]$ that maximize total value subject to capacity constraints:*

$$\max \sum_{j=1}^{m} \sum_{i \in S_j} v_{ij} \quad \textit{subject to} \quad \sum_{i \in S_j} w_{ij} \leq C_j \quad \textit{for all } j \in [m].$$

2.1.1 Stochastic Variants

For any deterministic instance Π defined above, its stochastic variant $\langle \Pi, p \rangle$ is characterized by a parameter $p \in (0, 1]$. A random subset $R \subseteq E$, termed the *active set*, is generated by sampling each element $e \in E$ independently with probability p. The goal is to select a feasible solution using only the items present in the realization R to maximize the objective value.

2.1.2 Assumptions

Without loss of generality, we assume that every individual item is feasible. That is, for every item $i \in E$, there exists at least one knapsack j such that the item's weight fits within the capacity ($w_{ij} \leq C_j$).

2.2 Notation and Feasibility

We distinguish between items and assignments using non-bold and bold notation, respectively.

- **Assignments (S):** We denote a specific assignment as $\mathbf{S} \subseteq E \times [m]$, where pairs $(i, j) \in \mathbf{S}$ indicate that item i is assigned to knapsack j. For a solution to be valid, each item must appear in at most one pair. We define the total value $v(\mathbf{S}) = \sum_{(i,j) \in \mathbf{S}} v_{ij}$ and the weight consumed in knapsack j as $w_j(\mathbf{S}) = \sum_{(i,j) \in \mathbf{S}} w_{ij}$.

- **Item Sets (S):** When $S \subseteq E$ denotes a subset of the ground set, it refers purely to the items themselves. We abuse notation slightly in the context of GAP: for a subset of items S implicitly assigned to a specific knapsack j, we write $w_j(S) = \sum_{i \in S} w_{ij}$.

A crucial distinction in our framework is the definition of feasibility for item sets. While the optimization problems maximize over assignments $\mathbf{S}$, our sparsification framework operates on the ground set E. We say that an item set $S \subseteq E$ is *feasible* if there exists a valid assignment $\mathbf{S}$ such that the set of assigned items is exactly S (i.e., $S = \{i \mid \exists j, (i,j) \in \mathbf{S}\}$). Consequently, the feasibility family $\mathcal{F}$ used to define the polytope $\mathcal{P}_{\mathcal{F}}$ consists of all such feasible item sets. Therefore, the condition for sparsification degree, $\mathbf{1}_Q \in d \cdot \mathcal{P}_{\mathcal{F}}$, is a constraint on the query set Q in the item space. The specific assignment $\mathbf{S}$ is determined only after the intersection of the query set and active set, $Q \cap R$, is revealed.

2.3 Hardness and Approximability

The computational complexity of these problems forms a natural hierarchy. The classical knapsack problem, which was proven NP-hard by Karp [19], admits a fully polynomial-time approximation scheme (FPTAS) [11, 17, 21]. However, the Multiple Knapsack Problem (MKP), even when restricted to just two knapsacks, does not admit an FPTAS [10]. The Generalized Assignment Problem (GAP) is APX-hard; Chakrabarty and Goel [9] demonstrated that it cannot be approximated better than a factor of $10/11$ unless P = NP. The current best polynomial-time approximation algorithm for GAP achieves a ratio of $1 - 1/e + \varepsilon$ for a small constant $\varepsilon > 0$ [16].

3 Warm-up: Knapsack Sparsification

We begin with a sparsification algorithm for the classical knapsack problem. This foundational case introduces key techniques that will be extended to more complex scenarios throughout the paper.

The algorithm employs a bucket-based strategy that partitions elements by value into geometrically increasing ranges. The fundamental principle ensures that the queried subset of items in each bucket is sufficiently large to either contain all items within the bucket or independently fill the knapsack constraint with high probability. In the former case, all elements remain available in the query set, ensuring that no item from the optimal solution is missed. In the latter scenario, the objective is to guarantee that whenever an item from the optimal solution is unavailable in the query set, a suitable substitute can be found.

To achieve this, the algorithm applies distinct selection criteria: for low-value elements (bucket B_0), it prioritizes items with high value-to-weight density, while for high-value buckets (B_k with $k \geq 1$), it selects the lightest elements to maximize the probability of accommodating valuable items within capacity constraints.

Before we proceed with proving that Algorithm 1 is a good sparsifier, we first establish a key concentration result for our sparsifier analysis. The proof follows from standard concentration inequalities and is given in the full version [14].

▶ **Lemma 2** (Activation Weight Concentration). *Let $S \subseteq E$ be a set of elements, each with weight w_i, such that*

$$\sum_{i \in S} w_i \geq \frac{\tau(\epsilon)}{p} \cdot C,$$

where $\tau(\epsilon) := 1 + \ln(1/\epsilon) + \sqrt{\ln^2(1/\epsilon) + 2\ln(1/\epsilon)}$. Then, if each item is active independently with probability p, the total weight of active items in S is at least C with probability at least $1 - \epsilon$.

Algorithm 1 Bucket-Based Sparsifier for Knapsack.

Input: accuracy parameter $\epsilon \in (0, 1/3)$.
Set concentration factor $\tau(\epsilon) = 1 + \ln\left(\frac{1}{\epsilon}\right) + \sqrt{\ln^2\left(\frac{1}{\epsilon}\right) + 2\ln\left(\frac{1}{\epsilon}\right)}$ as defined in Lemma 2.
Estimate $\mathbb{E}_R[\mathrm{OPT}]$ via sampling to obtain $(1 \pm \epsilon)$-approximation M.
Initialize query set $Q \leftarrow \emptyset$.
Set number of buckets $K := \lceil \frac{1}{\epsilon} \log(\frac{1}{\epsilon p}) \rceil$.
Partition elements into buckets:

$$B_k = \begin{cases} \{i \in E : v_i \le \epsilon M\} & \text{if } k = 0 \\ \{i \in E : v_i \in (\epsilon(1+\epsilon)^{k-1} M, \epsilon(1+\epsilon)^k M]\} & \text{if } 1 \le k \le K \end{cases}$$

Sort B_0 by decreasing value density v_i/w_i.
Define $\overline{B}_0$ as the shortest prefix of B_0 with total weight $\sum_{i \in \overline{B}_0} w_i \ge \frac{\tau(\epsilon)}{p} C$, or set $\overline{B}_0 = B_0$ if no such prefix exists.
Add to query set: $Q \leftarrow Q \cup \overline{B}_0$.
for $k = 1$ to K **do**
    Sort B_k in ascending order of weight w_i.
    Let $\overline{B}_k$ be the minimal prefix of B_k such that $\sum_{i \in \overline{B}_k} w_i \ge \frac{\tau(\epsilon)}{p} \cdot C$, or let $\overline{B}_k := B_k$ if no such prefix exists.
    Update $Q \leftarrow Q \cup \overline{B}_k$.
end for
Output: Return the query set Q.

▶ **Theorem 3** (Knapsack Sparsifier Performance). *For parameters $\epsilon \in (0, 1/3)$ and $p \in (0, 1]$, Algorithm 1 produces a $(1 - 4\epsilon)$-approximate sparsifier for the stochastic knapsack problem with sparsification degree*

$$O\left(\frac{\log(1/\epsilon) \cdot \log(1/(\epsilon p))}{\epsilon p}\right).$$

Proof. Let M denote our $(1 \pm \epsilon)$-approximation to $\mathbb{E}_R[\mathrm{OPT}]$, satisfying $(1-\epsilon) \cdot \mathbb{E}_R[\mathrm{OPT}] \le M \le (1+\epsilon) \cdot \mathbb{E}_R[\mathrm{OPT}]$ with probability at least $1 - \epsilon$. Let $\mathcal{E}_{\text{est}}$ be the event that this inequality holds and so M is estimated within $(1 \pm \epsilon)$ approximation. Condition on the event $\mathcal{E}_{\text{est}}$ holds.

Sparsification Degree Analysis

We first bound the size of the query set Q. The algorithm selects at most $K + 1$ buckets. For each bucket $\overline{B}_k$, the total weight is bounded by $\sum_{i \in \overline{B}_k} w_i \le C \cdot \frac{\tau(\epsilon)}{p} + \max_i w_i \le C\left(\frac{\tau(\epsilon)}{p} + 1\right)$. Summing over all $K = O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon p}))$ buckets, the total weight of the query set satisfies:

$$w(Q) = \sum_{k=0}^{K} w(\overline{B}_k) \le O\left(\frac{\tau(\epsilon)}{p} \cdot K\right) \cdot C.$$

To translate this weight bound into our polyhedral sparsification degree, we consider the linear programming relaxation of the knapsack polytope, $\mathcal{P}_{LP} = \{x \in [0,1]^E \mid \sum x_i w_i \le C\}$. Our calculation shows that $\mathbf{1}_Q \in d' \cdot \mathcal{P}_{LP}$ for $d' = O(\frac{w(Q)}{C})$. However, the sparsification degree requires embedding into the convex hull of integer solutions, $\mathcal{P}_{\mathcal{F}}$. We know that

the integrality gap of the knapsack relaxation is bounded by 2 (assuming singletons are feasible) [22]. Therefore, $\mathcal{P}_{LP} \subseteq 2 \cdot \mathcal{P}_{\mathcal{F}}$, implying that the sparsification degree is at most $2d'$, which remains $O\left(\frac{\log(1/\epsilon)\cdot\log(1/(\epsilon p))}{\epsilon p}\right)$.

Approximation Analysis

Consider any realization $R \subseteq E$ and let $S^* \subseteq R$ denote an optimal solution with value $\mathrm{OPT}(R) = \sum_{i\in S^*} v_i$ and weight $w(S^*) \le C$.

We partition the optimal solution as $S^* = S_0 \cup S_1 \cup \cdots \cup S_K$ where $S_k = S^* \cap B_k$, and define $S^{\text{low}} = S_0$ (low-value items) and $S^{\text{high}} = \bigcup_{k=1}^{K} S_k$ (high-value items). Completeness of this partition follows from the range of buckets. Since each item i is active with probability p, $\mathbb{E}[\mathrm{OPT}] \ge pv_i$, implying $v_i \le \mathbb{E}[\mathrm{OPT}]/p$. Our largest bucket boundary is at least $M/p = \mathbb{E}[\mathrm{OPT}]/p$, ensuring all items are covered.

High-Value Item Recovery. For each bucket $k \ge 1$, let $Q_k = \overline{B}_k \cap R$ represent the active queried items. By Lemma 2, we have $w(Q_k) \ge C$ with the probability of at least $1-\epsilon$ when $w(\overline{B_k})$ is sufficiently large (equivalently when $\overline{B}_k \ne B_k$). Define the event $\mathcal{E}_k$ as $\overline{B}_k = B_k$ or $w(Q_k) \ge C$.

Conditioning on this event $\mathcal{E}_k$, since $\overline{B}_k$ contains the lightest items in B_k, we can establish a matching $\phi_k : S_k \to Q_k$ such that each item $i \in S_k$ maps to some $\phi_k(i) \in Q_k$ with $w_i \ge w_{\phi_k(i)}$ and $v_i \le (1+\epsilon)\cdot v_{\phi_k(i)}$ (due to items being in the same value bucket). Notice that such a matching trivially exists when $\overline{B}_k = B_k$. This matching implies:

$$\mathbb{E}_R\left[\max_{\substack{T_k\subseteq Q_k\\ w(T_k)\le w(S_k)}} v(T_k) \mid \mathcal{E}_k\right] \ge \frac{1}{1+\epsilon}\cdot \mathbb{E}_R[v(S_k)].$$

Since $\Pr[\mathcal{E}_k] \ge (1-\epsilon)$, we obtain:

$$\mathbb{E}_R\left[\max_{\substack{T_k\subseteq Q_k\\ w(T_k)\le w(S_k)}} v(T_k)\right] \ge (1-\epsilon)\cdot\frac{1}{1+\epsilon}\cdot\mathbb{E}_R[v(S_k)] \ge (1-2\epsilon)\cdot\mathbb{E}_R[v(S_k)], \tag{1}$$

where the final inequality uses $1/(1+\epsilon) \ge 1-\epsilon$ for small ϵ.

Low-Value Item Recovery. For bucket B_0, we apply greedy selection on $\overline{B}_0 \cap R$ by value density up to a total capacity of $w(S_0) := \sum_{i\in S_0} w_i$. Let T_0 denote this greedy solution.

Each item in B_0 has value at most ϵM. When $\mathcal{E}_{\text{est}}$ holds, the maximum value in B_0 is at most $\epsilon(1+\epsilon)\mathbb{E}_R[\mathrm{OPT}] \le 2\epsilon\mathbb{E}_R[\mathrm{OPT}]$. By fractional knapsack analysis, the greedy algorithm achieves value within $2\epsilon\mathbb{E}_R[\mathrm{OPT}]$ of the fractional optimum:

$$\mathbb{E}_R[v(T_0) \mid \mathcal{E}_{\text{est}}] \ge \mathbb{E}_R[v(S_0) \mid \mathcal{E}_{\text{est}}] - 2\epsilon\cdot\mathbb{E}_R[\mathrm{OPT}].$$

Since $\Pr[\mathcal{E}_{\text{est}}] \ge 1-\epsilon$:

$$\mathbb{E}_R[v(T_0)] \ge (1-\epsilon)\cdot\mathbb{E}_R[v(S_0)] - 2\epsilon\cdot\mathbb{E}_R[\mathrm{OPT}].$$

Final Approximation Bound. Combining our bounds for high-value and low-value items, let T_k denote the maximum-value feasible subset of Q_k with $w(T_k) \le w(S_k)$ for each $k \ge 1$, and define $T = \bigcup_{k=0}^{K} T_k$. Then:

$$\begin{aligned}\mathbb{E}_R[v(T)] &= \sum_{k=0}^{K} \mathbb{E}_R[v(T_k)] \\ &\geq (1-\epsilon)\cdot \mathbb{E}_R[v(S_0)] - 2\epsilon \cdot \mathbb{E}_R[\mathrm{OPT}] + (1-2\epsilon)\cdot \sum_{k=1}^{K} \mathbb{E}_R[v(S_k)] \\ &= (1-2\epsilon)\cdot \mathbb{E}_R[v(S^*)] - 2\epsilon \cdot \mathbb{E}_R[\mathrm{OPT}] \\ &\geq (1-4\epsilon)\cdot \mathbb{E}_R[\mathrm{OPT}].\end{aligned}$$

Finally, since $w(T_k) \leq w(S_k)$ for each k, we have $w(T) = \sum_{k=0}^{K} w(T_k) \leq \sum_{k=0}^{K} w(S_k) = w(S^*) \leq C$, so T is feasible. ◀

4 Sparsifier for the General Assignment Problem

We now extend our sparsification framework to the Generalized Assignment Problem (GAP), which encompasses the Multiple Knapsack Problem as a special case.

4.1 Key Challenges and Algorithmic Innovations

Extending our knapsack sparsifier to the GAP presents fundamental challenges that require a complete algorithmic redesign. The core difficulty stems from knapsack-dependent item characteristics, which destroy the substitutability properties essential to our knapsack analysis.

In the knapsack problem, items have fixed values v_i and weights w_i, enabling a global bucketing scheme where items with similar characteristics substitute seamlessly. GAP breaks this structure: items exhibit knapsack-specific parameters (v_{ij}, w_{ij}), so an item valuable for one knapsack may be worthless for another. This forces us to maintain separate buckets $B_{j,k}$ for each knapsack-bucket pair, immediately complicating the design.

The main challenge arises from cross-knapsack substitutability issue. Two items may belong to the same bucket for knapsack j due to similar values v_{ij}, yet reside in different buckets for knapsack j' due to vastly different values $v_{ij'}$. When our sparsifier fills bucket $B_{j,k}$ based on suitability for knapsack j, these items fail as substitutes if the optimal solution assigns corresponding items to different knapsacks. This dependency fundamentally breaks the matching argument underlying our knapsack analysis.

We address this through enhanced redundancy combined with a more involved charging argument. Our GAP sparsifier fills each bucket to $\mathrm{poly}(1/\epsilon)$ times the knapsack capacity, creating substantial redundancy in the query set. When an optimal item i^* assigned to knapsack j^* is missing from the query set, we first seek a direct substitute among queried items with similar weight and value characteristics for knapsack j^*. When direct substitution fails – typically because suitable substitutes are assigned to different knapsacks in the optimal solution – we leverage the redundancy to fractionally charge value $v_{i^*j^*}$ across $\mathrm{poly}(1/\epsilon)$ other queried items. This ensures no queried item receives excessive charge (at most $1 + \mathrm{poly}(\epsilon)$ times its own value) while recovering nearly the optimal value.

A secondary challenge is ensuring the completeness of the bucket structure. In the knapsack setting, $\mathbb{E}_R[\mathrm{OPT}]/p$ provides a natural upper bound for feasible item values. In GAP, however, an item i may be active with probability p, yet appear in a specific knapsack j's optimal assignment with significantly lower probability, making localized value bounds difficult to establish.

To address this, we introduce a *super bucket* for each knapsack with no upper bound on its value range. While items in this bucket may have arbitrarily large values and lack mutual substitutability, we prove a surprising property: even if the reconstruction algorithm makes *no attempt* to substitute for missed items in the super bucket, the aggregate loss remains globally bounded.

4.2 Algorithm Design

Our GAP sparsifier employs a bucket-based approach that incorporates substantial redundancy to handle cross-knapsack dependencies. The complete procedure is presented in Algorithm 2.

Algorithm 2 Bucket-Based Sparsifier for GAP.

Oracle access: For each knapsack $j \in [m]$, assume oracle access to $\mathbb{E}_R[\mathrm{OPT}_j]$, and denote it by M_j.
Input: accuracy parameter $\epsilon \in (0, 1/3)$.
Set concentration factor $\tau(\epsilon) = 1 + \ln\left(\frac{1}{\epsilon}\right) + \sqrt{\ln^2\left(\frac{1}{\epsilon}\right) + 2\ln\left(\frac{1}{\epsilon}\right)}$ as defined in Lemma 2.
Initialize query set $Q \leftarrow \emptyset$.
Define the number of buckets $K := \left\lceil \frac{2}{\epsilon^2} \log\left(\frac{1}{\epsilon^3}\right) \right\rceil$.
Bucket Formation:
for each knapsack $j \in [m]$ **do**
  Set buckets:

$$B_{j,k} := \begin{cases} \{(i,j) \mid i \in E,\ v_{ij} \le \epsilon^2 M_j\}, & k = 0, \\ \{(i,j) \mid i \in E,\ v_{ij} \in (\epsilon^2(1+\epsilon^2)^{k-1} M_j,\ \epsilon^2(1+\epsilon^2)^k M_j]\}, & 1 \le k \le K, \\ \{(i,j) \mid i \in E,\ v_{ij} > \epsilon^2(1+\epsilon^2)^K M_j \ \ge\ M_j/\epsilon\}, & k = K+1. \end{cases}$$

end for
Preprocessing: For each (i,j), define $\beta(i,j)$ as the bucket index such that $(i,j) \in B_{j,\beta(i,j)}$.
Iterative Assignment:
Define the number of rounds $\alpha := \frac{1}{\epsilon}$.
for round $t = 1$ to $\alpha - 1$ **do**
  for each knapsack $j \in [m]$, each bucket $k \in \{0, 1, \ldots, K+1\}$ **do**
    Initialize remaining capacity: $b(j,k) \leftarrow \frac{\tau(\epsilon^2)}{p} \cdot C_j$.
  end for
  while there exists (i,j) with $i \notin Q$, $\beta(i,j) \ge 1$, $w_{ij} \le C_j$, and $b(j, \beta(i,j)) > 0$ **do**
    Select (i,j) with minimal w_{ij} among valid pairs.
    Update $Q \leftarrow Q \cup \{i\}$ and remaining capacity $b(j, \beta(i,j)) \leftarrow b(j, \beta(i,j)) - w_{ij}$.
  end while
  while there exists (i,j) with $i \notin Q$, $\beta(i,j) = 0$, $w_{ij} \le C_j$, and $b(j,0) > 0$ **do**
    Select (i,j) maximizing v_{ij}/w_{ij} among valid pairs.
    Update $Q \leftarrow Q \cup \{i\}$ and remaining capacity $b(j,0) \leftarrow b(j,0) - w_{ij}$.
  end while
end for
Output: Return the constructed query set Q.

Beyond the algorithmic complexities discussed above, our GAP sparsifier requires access to knapsack-level value estimates. This presents an additional challenge compared to the knapsack setting, where estimating the global expectation $\mathbb{E}_R[\mathrm{OPT}]$ suffices for bucket boundary determination. In GAP, assignment decisions are interdependent across knapsacks, creating a more complex estimation problem.

Formally, for each realization R of the active set, let $\mathcal{OPT}(R)$ denote the set of all optimal feasible GAP assignments on R. We fix once and for all an arbitrary but deterministic tie-breaking rule that selects a canonical optimal assignment $\mathbf{OPT}(R) \in \mathcal{OPT}(R)$. We then define $\mathrm{OPT}(R)$ as the total value of assignment $\mathbf{OPT}(R)$ and $\mathrm{OPT}_j(R)$ as the total value of items assigned to knapsack j in $\mathbf{OPT}(R)$, so that

$$\mathrm{OPT}(R) = \sum_{j=1}^{m} \mathrm{OPT}_j(R).$$

Although the individual quantities $\mathrm{OPT}_j(R)$ may vary with the choice of tie-breaking rule, the equality above implies that

$$\sum_{j=1}^{m} \mathbb{E}_R[\mathrm{OPT}_j(R)] \;=\; \mathbb{E}_R[\mathrm{OPT}(R)],$$

and thus the aggregate contribution across knapsacks is invariant. The relative contribution of each knapsack can still vary substantially across different realizations and approximate solutions, which makes estimating the individual knapsack contributions $\mathbb{E}_R[\mathrm{OPT}_j]$ from global information alone highly challenging.

To address this issue, we assume oracle access to the expected knapsack-level optima $\mathbb{E}_R[\mathrm{OPT}_j]$ for all $j \in [m]$. Under this assumption, we establish the following performance guarantee:

▶ **Theorem 4** (GAP Sparsifier). *For parameters $\epsilon \in (0, 1/6)$ and $p \in (0, 1]$, assume oracle access to the expected knapsack optima $\mathbb{E}_R[\mathrm{OPT}_j]$ for each knapsack $j \in [m]$. Then Algorithm 2 produces a $(1-6\epsilon)$-approximate sparsifier for the stochastic GAP problem with sparsification degree*

$$O\left(\frac{\log^2(1/\epsilon)}{\epsilon^3 p}\right).$$

Since the Multiple Knapsack Problem is a special case of GAP, we obtain the following immediate corollary:

▶ **Corollary 5** (Multiple Knapsack Sparsifier). *Under the same oracle assumption, Algorithm 2 produces a $(1-6\epsilon)$-approximate sparsifier for the stochastic Multiple Knapsack problem, with sparsification degree*

$$O\left(\frac{\log^2(1/\epsilon)}{\epsilon^3 p}\right).$$

4.3 Relaxing Oracle Assumptions

In practice, obtaining precise offline computations of each $\mathbb{E}_R[\mathrm{OPT}_j]$ may be infeasible. We therefore analyze the robustness of our algorithm under weaker information settings.

4.3.1 Approximate Oracle Access

Given a β-approximation to the total stochastic optimum OPT, we can estimate each $\mathbb{E}_R[\mathrm{OPT}_j]$ via expected marginal contributions of knapsacks in the approximate solution. Using these estimates, Algorithm 2 achieves a $\beta \cdot (1 - 6\epsilon)$-approximation with the same sparsification degree bound. The analysis remains identical to Theorem 4, using the β-approximate assignment as the benchmark for the charging argument.

4.3.2 Global Oracle Access

A plausible scenario is having access to the global expectation $\mathbb{E}_R[\mathrm{OPT}]$ (or a constant factor estimate thereof) without granular knapsack-level details. In this case, we can uniformly distribute the expectation by setting $M_j = \mathbb{E}_R[\mathrm{OPT}]/m$ for all j. This forces the algorithm to cover a wider range of potential values per knapsack, introducing a logarithmic dependence on m in the sparsification degree.

Intuitively, the contribution of any specific knapsack j lies somewhere between a uniform share ($\approx \mathbb{E}_R[\mathrm{OPT}]/m$) and the total value ($\approx \mathbb{E}_R[\mathrm{OPT}]$). To ensure we capture the relevant items regardless of how the optimal solution distributes value, we set the minimum value threshold M_j based on the uniform lower bound $\mathbb{E}_R[\mathrm{OPT}]/m$. Consequently, the bucket hierarchy for *every* knapsack must span the expansive range from this uniform average up to the global maximum. This widens the value range by a factor of m, which, due to the geometric progression of bucket boundaries, incurs a logarithmic penalty in the number of buckets.

▶ **Corollary 6.** *Assume oracle access to the expected global optimum $\mathbb{E}_R[\mathrm{OPT}]$. Then the modified version of Algorithm 2 with $M_j = \mathbb{E}_R[\mathrm{OPT}]/m$ and $K := \left\lceil \frac{2}{\epsilon^2} \log\left(\frac{m}{\epsilon^3}\right) \right\rceil$ is a $(1-6\epsilon)$-sparsifier with degree $O\left(\frac{\log^2(1/\epsilon)\log(m)}{\epsilon^3 p}\right)$.*

5 Analysis of GAP Sparsifier

Before beginning the analysis, we briefly recall the relevant notation. Non-bold symbols (e.g., S) denote sets of items, whereas bold symbols (e.g., $\mathbf{S}$) denote assignment solutions, represented as sets of item–knapsack pairs. For each realization R of the active set, we previously introduced a canonical optimal assignment $\mathbf{OPT}(R)$ via a deterministic tie-breaking rule. This choice is used solely for analysis; the sparsification algorithm (Algorithm 2) never requires access to $\mathbf{OPT}(R)$ for any individual R. For simplicity of notation, throughout this section we omit the dependence on R and write $\mathbf{OPT}$.

With this notation in place, our analysis centers on a reconstruction algorithm that conceptually simulates $\mathbf{OPT}$ while constructing a feasible assignment $\mathbf{ALG}$ using only queried active items. The reconstruction processes buckets (j, k) sequentially, maintaining a partial assignment $\overline{\mathbf{OPT}}$ that incrementally grows toward $\mathbf{OPT}$.

The reconstruction algorithm employs three specialized subroutines. FillLargeBucket, for $1 \le k \le K$, replaces missed high-value items with lighter alternatives. FillSmallBucket, for $k = 0$, uses density-based substitution for low-value items. FillAllSuperBuckets, for $k = K + 1$, performs no substitutions and simply inserts each queried item using its original assignment while discarding all missed items.

Before providing definitions of these subroutines, we first state some desired properties of these subroutines and hence our ReconstructionProcedure. Based on these properties, we first prove the main result of this section in Section 5.1. Later, Section 6 will be devoted to formal definitions of subroutines, FillLargeBucket, FillSmallBucket, and FillAllSuperBuckets, as well as proofs of their desired properties.

Algorithm 3 ReconstructionProcedure.

$\overline{\mathbf{OPT}} \leftarrow \emptyset$, $\mathbf{ALG} \leftarrow \emptyset$
for $j = 1$ to m **do**
  $(\overline{\mathbf{OPT}}, \mathbf{ALG}) \leftarrow$ FillSmallBucket$(\overline{\mathbf{OPT}}, \mathbf{ALG}, j, 0)$
  for $k = 1$ to $\left\lceil \frac{2}{\epsilon^2} \log \frac{1}{\epsilon^3} \right\rceil$ **do**
    $(\overline{\mathbf{OPT}}, \mathbf{ALG}) \leftarrow$ FillLargeBucket$(\overline{\mathbf{OPT}}, \mathbf{ALG}, j, k)$
  end for
end for
$(\overline{\mathbf{OPT}}, \mathbf{ALG}) \leftarrow$ FillAllSuperBuckets$(\overline{\mathbf{OPT}}, \mathbf{ALG})$

To formally track the capacity usage and value gain during reconstruction, we define the total weight and value contributed to each knapsack by the subroutine calls in ReconstructionProcedure. Consider an assignment $\mathbf{S} \in \{\mathbf{ALG}, \overline{\mathbf{OPT}}\}$. For a single call to either FillSmallBucket or FillLargeBucket, we define:

- $\Delta w_j(\mathbf{S})$: for each knapsack $j \in [m]$, the increase in the total weight assigned to j in $\mathbf{S}$, namely the sum of w_{ij} over all item–knapsack pairs (i, j) that are added to $\mathbf{S}$ during this call;
- $\Delta v(\mathbf{S})$: the increase in the total value of $\mathbf{S}$ across all knapsacks, namely the sum of v_{ij} over all pairs (i, j) that are added to $\mathbf{S}$ during this call.

Feasibility

The ReconstructionProcedure maintains feasibility through three key properties: (1) each call to the procedure adds only queried active items to $\mathbf{ALG}$, ensuring that $\mathbf{ALG} \subseteq Q \cap R$ at all times; (2) no item is ever assigned more than once within either solution: once an item appears in $\mathbf{ALG}$ (respectively, $\overline{\mathbf{OPT}}$), it is never reassigned within that same solution; and (3) for every knapsack j, the total weight assigned in $\mathbf{ALG}$ never exceeds that in $\overline{\mathbf{OPT}}$, ensuring that capacity constraints remain satisfied throughout. Lemmas 7, 8, and 9 formally establish these properties for large, small, and super buckets, respectively.

▶ **Lemma 7** (Feasibility in Large Buckets). *When FillLargeBucket is called for knapsack j and bucket k, the procedure assigns only queried active items to $\mathbf{ALG}$, and no item already appearing in $\mathbf{ALG}$ (respectively, $\overline{\mathbf{OPT}}$) is ever reassigned within that same solution. Moreover, for every knapsack $j' \in [m]$,*

$$\Delta w_{j'}(\mathbf{ALG}) \leq \Delta w_{j'}(\overline{\mathbf{OPT}}).$$

▶ **Lemma 8** (Feasibility in Small Buckets). *When FillSmallBucket is called for knapsack j and bucket k, the procedure assigns only queried active items to $\mathbf{ALG}$, and no item already appearing in $\mathbf{ALG}$ (respectively, $\overline{\mathbf{OPT}}$) is ever reassigned within that same solution. Moreover, for every knapsack $j' \in [m]$,*

$$\Delta w_{j'}(\mathbf{ALG}) \leq \Delta w_{j'}(\overline{\mathbf{OPT}}).$$

▶ **Lemma 9** (Feasibility in Super Buckets). *When FillAllSuperBuckets is called, the procedure assigns only queried active items to $\mathbf{ALG}$, and no item already appearing in $\mathbf{ALG}$ (respectively, $\overline{\mathbf{OPT}}$) is ever reassigned within that same solution. Moreover, for every knapsack $j' \in [m]$,*

$$\Delta w_{j'}(\mathbf{ALG}) \leq \Delta w_{j'}(\overline{\mathbf{OPT}}).$$

Approximation

The ReconstructionProcedure adds new item–knapsack assignments to both $\overline{\mathbf{OPT}}$ and **ALG** in such a way that the total value of the new assignments added to **ALG** closely tracks that of the new assignments added to $\overline{\mathbf{OPT}}$. This ensures that, throughout the reconstruction process, the value of **ALG** remains a good approximation to the value of $\overline{\mathbf{OPT}}$. The following three lemmas prove this property for large-value, small-value, and super-value buckets, respectively.

▶ **Lemma 10** (Approximation Guarantee in Large Buckets). *When FillLargeBucket is called for knapsack j and bucket k, the following inequality holds:*

$$\mathbb{E}_R[\Delta v(\mathbf{ALG})] \geq (1-2\epsilon)\,\Delta v(\overline{\mathbf{OPT}}).$$

▶ **Lemma 11** (Approximation Guarantee in Small Buckets). *When FillSmallBucket is called for knapsack j and bucket k, the inequality*

$$\mathbb{E}_R[\Delta v(\mathbf{ALG})] \geq (1-2\epsilon)\,\Delta v(\overline{\mathbf{OPT}}) - \epsilon^2 M_j$$

holds, where $M_j = \mathbb{E}_R[\mathrm{OPT}_j]$ denotes the expected contribution of knapsack j to the optimal assignment.

▶ **Lemma 12** (Approximation Guarantee in Super Buckets). *Assume $0 < \epsilon \leq 1/2$. When FillAllSuperBuckets is called, the following inequality holds:*

$$\mathbb{E}_R\left[\Delta v(\overline{\mathbf{OPT}})\right] - \mathbb{E}_R\left[\Delta v(\mathbf{ALG})\right] \leq 3\epsilon \cdot \mathbb{E}_R\left[v(\mathbf{OPT})\right].$$

Correct Benchmark

Finally, to demonstrate that the final output **ALG** is a good approximation of **OPT**, we ensure that ReconstructionProcedure terminates with $\overline{\mathbf{OPT}} = \mathbf{OPT}$. The following lemma establishes this property:

▶ **Lemma 13** (Completeness). *Upon completion of ReconstructionProcedure (Algorithm 3), we have $\overline{\mathbf{OPT}} = \mathbf{OPT}$.*

Now, we are ready to prove the main result of this section using these properties. Formal definitions of subroutines FillSmallBucket, FillLargeBucket, and FillAllSuperBuckets, as well as proofs of Lemmas 7-13 will be provided in Section 6.

5.1 Proof of Theorem 4

We now prove the main result of this section using the lemmas stated above.

Proof of Theorem 4. Fix any realization $R \subseteq E$.

We first bound the size of the query set by analyzing the total weight selected for each knapsack. Algorithm 2 maintains $K = \left\lceil \frac{2}{\epsilon^2} \log\left(\frac{1}{\epsilon^3}\right) \right\rceil = O\left(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon}\right)$ buckets per knapsack and iterates for $\alpha = \lceil 1/\epsilon \rceil$ rounds. In each specific round t and bucket (j,k), the algorithm selects items with a total weight of at most $(\frac{\tau(\epsilon^2)}{p} + 1)C_j$, where the $+1$ accounts for the discrete weight of the final item. Aggregating over all rounds and buckets, the total weight implicitly associated with knapsack j is bounded by:

$$w_j(Q) \leq \alpha \cdot K \cdot O\left(\frac{\tau(\epsilon^2)}{p}\right) \cdot C_j = O\left(\frac{\log^2(1/\epsilon)}{\epsilon^3 p}\right) C_j$$

This weight bound establishes that the indicator vector $\mathbf{1}_Q$ lies within the natural linear programming relaxation of the problem, scaled by a factor $d_{LP} = O\left(\frac{\log^2(1/\epsilon)}{\epsilon^3 p}\right)$. To translate this into the required polyhedral sparsification degree (which is defined with respect to $\mathcal{P}_{\mathcal{F}}$, the convex hull of *integer* feasible solutions), we leverage the fact that the integrality gap of the standard linear relaxation for GAP is at most 2 (assuming feasible singletons). This implies polytope of relaxed LP $\mathcal{P}_{LP}$ is contained by $2 \cdot \mathcal{P}_{\mathcal{F}}$. Consequently, the sparsification degree is at most $2d_{LP}$, which preserves the asymptotic bound:

$$d = O\left(\frac{\log^2(1/\epsilon)}{\epsilon^3 p}\right).$$

Feasibility

For each procedure call to FillLargeBucket, FillSmallBucket, or FillAllSuperBuckets, let $\Delta w_{j'}^{j,k}(\cdot)$ denote the weight-change function for knapsack j'. Applying Lemmas 7–9 and summing over all (j,k), we obtain

$$w_{j'}(\mathbf{ALG}) = \sum_{j,k} \Delta w_{j'}^{j,k}(\mathbf{ALG}) \;\le\; \sum_{j,k} \Delta w_{j'}^{j,k}(\overline{\mathbf{OPT}}) = w_{j'}(\mathbf{OPT}),$$

where the final equality follows from Lemma 13.

Moreover, Lemma 7 and Lemma 8 ensure that **ALG** includes each item only once and that every item inserted into **ALG** comes from the queried active set $Q \cap R$, guaranteeing that **ALG** is a valid solution after sparsification and realization. Since **OPT** is feasible, the weight domination $w_{j'}(\mathbf{ALG}) \le w_{j'}(\mathbf{OPT})$ implies that **ALG** is also feasible.

Approximation Guarantee

For each knapsack j and bucket index $k \le K$, let $\Delta v^{j,k}$ denote the value increase function when calling FillLargeBucket or FillSmallBucket. For the super bucket $k = K+1$, we define $\Delta v^{j,K+1}$ analogously as the value contribution of FillAllSuperBuckets for knapsack j. By Lemmas 10–13, we have

$$\begin{aligned}
\mathbb{E}_R[v(\mathbf{ALG})] &= \sum_j \sum_{k=0}^{K} \mathbb{E}_R[\Delta v^{j,k}(\mathbf{ALG})] + \sum_j \mathbb{E}_R[\Delta v^{j,K+1}(\mathbf{ALG})] && (2)\\
&\ge \Big((1-2\epsilon)\sum_j \sum_{k=0}^{K} \mathbb{E}_R[\Delta v^{j,k}(\overline{\mathbf{OPT}})] \;-\; \epsilon^2 \sum_j M_j\Big)\\
&\quad + \Big(\sum_j \mathbb{E}_R[\Delta v^{j,K+1}(\overline{\mathbf{OPT}})] \;-\; 3\epsilon\, \mathbb{E}_R[v(\mathbf{OPT})]\Big) && (3)\\
&\ge (1-2\epsilon)\sum_j \sum_{k=0}^{K+1} \mathbb{E}_R[\Delta v^{j,k}(\overline{\mathbf{OPT}})] \;-\; \epsilon^2 \sum_j M_j \;-\; 3\epsilon\, \mathbb{E}_R[v(\mathbf{OPT})] && (4)\\
&= (1-2\epsilon)\mathbb{E}_R[v(\mathbf{OPT})] \;-\; \epsilon^2 \mathbb{E}_R[v(\mathbf{OPT})] \;-\; 3\epsilon\, \mathbb{E}_R[v(\mathbf{OPT})] && (5)\\
&= (1-2\epsilon-\epsilon^2-3\epsilon)\mathbb{E}_R[v(\mathbf{OPT})]\\
&\ge (1-6\epsilon)\mathbb{E}_R[v(\mathbf{OPT})], && (6)
\end{aligned}$$

where equation (2) follows from linearity of expectation; equation (3) applies Lemmas 10 and 11 to derive the first parenthesized term, and applies Lemma 12 to derive the second parenthesized term; equation (4) rearranges the terms; and equation (5) uses $\mathbb{E}_R[v(\mathbf{OPT})] = \sum_j M_j$ together with Lemma 13. The final inequality holds for all $0 < \epsilon \le 1/6$. ◀

6 Reconstruction Procedures

This section details the ReconstructionProcedure, which incrementally constructs the optimal solution $\overline{\mathbf{OPT}}$ and a feasible algorithmic solution **ALG** using the queried active elements. We first establish the necessary notation and then analyze the feasibility and approximation guarantees of our reconstruction subroutines.

6.1 Notation and Preliminaries

Optimal Assignment. Let v_i^{OPT} denote the value of item i in the optimal solution: $v_i^{\mathrm{OPT}} = v_{ij}$ if $(i,j) \in \mathbf{OPT}$, and 0 otherwise. Throughout the text, we assume that optimal solution is deterministically fixed among feasible solutions satisfying optimality.

Buckets and Query Sets. We denote by $B_{j,k}$ the set of item-knapsack pairs (i,j) such that value v_{ij}, i.e., the value of item i in bucket j, falling into the k-th value scale for knapsack j. Let $\overline{B}_{j,k,t}$ be the set of active items included to query set in round t, and $\overline{B}_{j,k} = \bigcup_t \overline{B}_{j,k,t}$ be the total set of queried items for this bucket. By construction, these sets are pairwise disjoint across (j,k,t).

Partition of OPT. We partition the items in **OPT** based on whether they were successfully queried. Note that an item i assigned to j' in **OPT** might be queried via a different bucket (j,k).

- $S_{j,k}^{\mathrm{queried}} := \{i \mid i \in OPT \cap \overline{B}_{j,k}\}$: Items used by **OPT** that were successfully queried via bucket (j,k).
- $S_{j,k}^{\mathrm{missed}} := \{i \mid (i,j) \in \mathbf{OPT} \cap B_{j,k} \text{ and } i \notin Q\}$: Items assigned to knapsack j in **OPT** falling in bucket $B_{j,k}$ that were *not* queried.

This forms a partition: $OPT = \bigcup_{j,k}(S_{j,k}^{\mathrm{queried}} \cup S_{j,k}^{\mathrm{missed}})$.

6.2 Subroutine Design

We gradually build the solutions $\overline{\mathbf{OPT}}$ and **ALG**, which are initially set to $\emptyset$. The reconstruction relies on three subroutines handling High-Value ($1 \le k \le K$), Low-Value ($k = 0$), and Super-Value ($k = K+1$) regimes. In all regimes, if $S_{j,k}^{\mathrm{missed}} = \emptyset$, we simply assign $S_{j,k}^{\mathrm{queried}}$ to both $\overline{\mathbf{OPT}}$ and **ALG**. If missed items exist, we credit them to $\overline{\mathbf{OPT}}$ and attempt to find substitutes from $\overline{B}_{j,k}$ for **ALG**.

Super-Value Regime. FillAllSuperBuckets performs no substitution. It assigns $S_{j,K+1}^{\mathrm{queried}}$ to both solutions and $S_{j,K+1}^{\mathrm{missed}}$ only to $\overline{\mathbf{OPT}}$. While seemingly wasteful, the loss is globally bounded.

6.3 Analysis

Throughout the analysis for each j,k and t, we condition on the **Excess Weight Event** $\mathcal{E}_{j,k,t} := \{w_j(\overline{B}_{j,k,t}) \ge C_j\}$. Whenever $S_{j,k}^{\mathrm{missed}} \neq \emptyset$, the bucket must have exhausted its query capacity. Lemma 2 ensures that $\Pr[\mathcal{E}_{j,k,t}] \ge 1 - \epsilon^2$ for each j,k, and t separately, unless all items i with $w_{ij} < C_j$ appearing in the pairs $(i,j) \in B_{j,k}$ have already been included in the query set. Note that in this latter case, the query set contains all relevant items for this bucket, which implies that the reconstruction can assign exactly the same set of items as the stochastic optimum, making the comparison immediate.

Algorithm 4 FillLargeBucket($\overline{\mathbf{OPT}}, \mathbf{ALG}, j, k$) [Abstracted].

while $S_{j,k}^{\text{missed}} \neq \emptyset$ **do**
  Pick $i \in S_{j,k}^{\text{missed}}$.
  if $\exists i' \in \overline{B}_{j,k}$ unused by both **OPT** and **ALG** **then**
    Direct Substitution: $\overline{\mathbf{OPT}} \leftarrow \overline{\mathbf{OPT}} \cup (i,j)$; $\mathbf{ALG} \leftarrow \mathbf{ALG} \cup (i',j)$.
  else
    Redistribution:
    Let $T \subseteq [\alpha - 1]$ be a collection of indices t such that there exists $i'_t \in \overline{B}_{j,k,t}$ with $(i'_t, j'_t) \in \mathbf{OPT}$ and $i'_t \notin \mathbf{ALG}$.
    Let $t^* \in T$ be the index with $v_{i'_t}^{\text{OPT}}$ is minimum.
    Option A: Assign each i'_t to j'_t. Option B: Assign each i'_t to j'_t except $t = t^*$ and assign i'_{t^*} to j.
    Select the option maximizing **ALG** value. Update $\overline{\mathbf{OPT}}$ with $\{(i'_t, j'_t)\}_{t \in T} \cup \{(i,j)\}$.
  end if
  Remove processed items from $S_{j,k}^{\text{queried}}$ and $S_{j,k}^{\text{missed}}$.
end while
Cleanup: Add remaining $S_{j,k}^{\text{queried}}$ to both $\overline{\mathbf{OPT}}$ and **ALG**.

Algorithm 5 FillSmallBucket($\overline{\mathbf{OPT}}, \mathbf{ALG}, j, k$) [Abstracted].

if $S_{j,k}^{\text{missed}} = \emptyset$ **then**
  Assign $S_{j,k}^{\text{queried}}$ to both $\overline{\mathbf{OPT}}$ and **ALG**.
else
  Sort $\overline{B}_{j,k}$ by density v_i^{OPT}/w_{ij} in non-decreasing order.
  Let $S \subseteq \overline{B}_{j,k}$ be the longest prefix of items i in $\overline{B}_{j,k}$ satisfying

$$w_j(S) \leq w_j(S_{j,k}^{\text{missed}}) \quad \text{and} \quad v_{ij} > v_i^{\text{OPT}}.$$

  Assign: $\overline{\mathbf{OPT}} \leftarrow \overline{\mathbf{OPT}} \cup S_{j,k}^{\text{missed}} \cup S_{j,k}^{\text{queried}}$.
  Assign: **ALG** takes items in S (reassigned to j) and remaining queried items (original assignment).
end if

6.3.1 Substitution Availability

▶ **Lemma 14** (Substitution Properties). *Conditioned on $\mathcal{E}_{j,k,t}$, if $S_{j,k}^{\text{missed}} \neq \emptyset$:*

1. *BucketFilled: $w_j(\overline{B}_{j,k,t}) \geq C_j$.*
2. *AlternativeExists: For $k \neq 0$, queried items are lighter ($w_{i'j} \leq w_{ij}$); for $k = 0$, queried items have higher density ($v_{i'j}/w_{i'j} \geq v_{ij}/w_{ij}$) where $i' \in \overline{B}_{j,k}$ and $i \in B_{j,k} \setminus \overline{B}_{j,k}$.*
3. *SizeMatch: $|\overline{B}_{j,k,t}| \geq |S_{j,k}^{\text{missed}}|$ for $k \neq 0$.*

Proof Sketch. Follows immediately from the greedy selection in Algorithm 2 (selecting lightest/densest items first) and the capacity guarantee provided by $\mathcal{E}_{j,k,t}$. ◀

6.3.2 Feasibility

Lemma 7, Lemma 8, and Lemma 9 establish that every subroutine maintains the feasibility of the partial solution **ALG** by ensuring its weight consumption is dominated knapsack-wise by $\overline{\mathbf{OPT}}$. The following proof sketch summarizes the critical arguments. Specifically, after each subroutine call, the following conditions hold:

(1) $\mathbf{ALG} \subseteq Q \cap R$;

(2) Each item is assigned at most once;

(3) For every knapsack j', the marginal weight increase satisfies $\Delta w_{j'}(\mathbf{ALG}) \leq \Delta w_{j'}(\overline{\mathbf{OPT}})$.

Proof Sketch. Properties (1) and (2) follow immediately from the construction of the bucket sets $\overline{B}_{j,k}$, which form a disjoint partition of the queried items. We verify Property (3) by inspecting the substitution logic in each regime:

- **Large Buckets:** For missed items ($S^{\text{missed}}_{j,k}$), $\overline{\mathbf{OPT}}$ keeps its original assignment. **ALG** either keeps the same items or makes feasibility-preserving substitutions: in *Direct Substitution*, a missed item i is replaced by a lighter queried item i' ($w_{i'j} \leq w_{ij}$); in *Redistribution*, **ALG** assigns a subset of the items assigned by $\overline{\mathbf{OPT}}$, and any reassignment as substitution it makes also satisfies $w_{i'j} \leq w_{ij}$. For queried items ($S^{\text{queried}}_{j,k}$), **ALG** takes the same items or a feasible subset. Thus, in all branches, **ALG** only keeps $\overline{\mathbf{OPT}}$'s items or replaces them with lighter feasible ones.
- **Small Buckets:** The substitution prefix S is explicitly constructed such that its total weight satisfies $w_j(S) \leq w_j(S^{\text{missed}}_{j,k})$. For items not in S or S^{missed}, **ALG** exactly mirrors $\overline{\mathbf{OPT}}$.
- **Super Buckets:** **ALG** selects a strict subset of the assignments made by $\overline{\mathbf{OPT}}$, specifically including only the successfully queried items and dropping the missed ones.

Consequently, the capacity constraints are satisfied relative to $\overline{\mathbf{OPT}}$, ensuring the feasibility of **ALG**. ◀

6.3.3 Approximation Guarantees

Here, we will restate lemmas utilized in Section 5.1 and provide their proof sketches.

▶ **Lemma 10** (Approximation Guarantee in Large Buckets). *When* FillLargeBucket *is called for knapsack j and bucket k, the following inequality holds:*

$$\mathbb{E}_R[\Delta v(\mathbf{ALG})] \geq (1 - 2\epsilon)\, \Delta v(\overline{\mathbf{OPT}}).$$

Proof Sketch. We analyze the value added during the processing of a missed item i.

- **Direct Substitution:** We swap i for i'. Since they are in the same bucket, $v_{i'j} \geq (1 - \epsilon^2)v_{ij}$.
- **Redistribution:** We identify a set of indices T (size $\approx \alpha$). $\overline{\mathbf{OPT}}$ makes the assignment $\{i'_t \to j'_t\}_{t \in T}$ and $\{i \to j\}$. However, **ALG** makes the best feasible assignment of size $|T|$. By averaging, the least valuable knapsack assignment in the original bundle contributes at most $1/\alpha$ of the total value. Thus, **ALG** captures at least $(1 - 1/\alpha)$ of the value assigned to $\overline{\mathbf{OPT}}$.

Considering the failure probability of $\mathcal{E}_{j,k,t}$, we account for a factor of $(1 - \epsilon^2)^\alpha \approx 1 - \epsilon$. Setting $\alpha = 1/\epsilon$ yields the result. ◀

▶ **Lemma 11** (Approximation Guarantee in Small Buckets). *When* FillSmallBucket *is called for knapsack j and bucket k, the inequality*

$$\mathbb{E}_R[\Delta v(\mathbf{ALG})] \geq (1-2\epsilon)\,\Delta v(\overline{\mathbf{OPT}}) - \epsilon^2 M_j$$

holds, where $M_j = \mathbb{E}_R[\mathrm{OPT}_j]$ denotes the expected contribution of knapsack j to the optimal assignment.

Proof Sketch. If $S_{j,0}^{\text{missed}} \neq \emptyset$, consider the set of items in $\overline{B}_{j,0}$ that our procedure is allowed to reassign to knapsack j. We sort these candidates by ascending opt-value-to-weight ratio and take the maximal prefix S whose total weight fits in the remaining capacity of knapsack j, and such that S contributes more value after reassignment than under its original assignment (see Figure 2 for a visual illustration of this prefix-selection step). By AlternativeExists, every candidate in S has density higher than every missed item. Thus, after reassignment, $\mathbf{ALG}$ captures the highest-density-with-respect-to-j $(1-1/\alpha)$ portion of the weight that $\overline{\mathbf{OPT}}$ assigns in this call. In addition, when defining S we may have to exclude one item that would cause the capacity on j to be exceeded; this "boundary" item i^* lies in a small bucket and therefore satisfies $v_{i^*j} \leq \epsilon^2 M_j$. Combining these two effects and the failure probability of $\mathcal{E}_{j,k,t}$ yields $\Delta v(\mathbf{ALG}) \geq (1-1/\alpha)\,\Delta v(\overline{\mathbf{OPT}}) - \epsilon^2 M_j$. ◀

▶ **Lemma 12** (Approximation Guarantee in Super Buckets). *Assume $0 < \epsilon \leq 1/2$. When* FillAllSuperBuckets *is called, the following inequality holds:*

$$\mathbb{E}_R\left[\Delta v(\overline{\mathbf{OPT}})\right] - \mathbb{E}_R\left[\Delta v(\mathbf{ALG})\right] \leq 3\epsilon \cdot \mathbb{E}_R\left[v(\mathbf{OPT})\right].$$

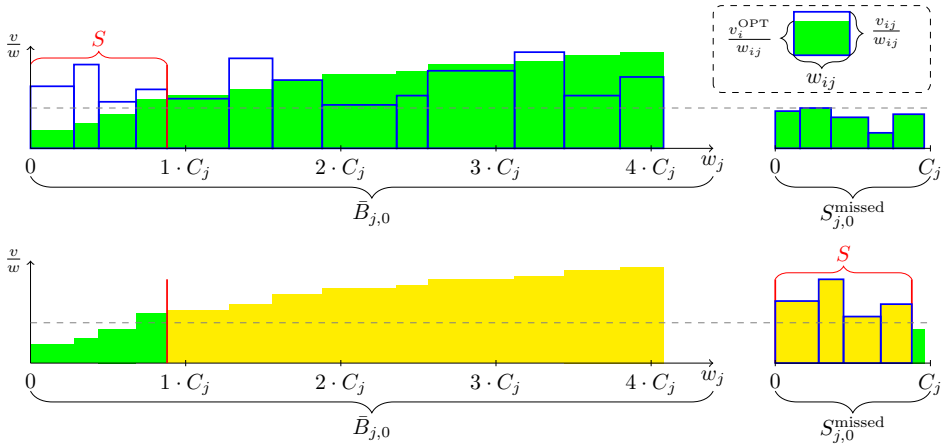
Proof Sketch. The difference comes exactly from the value of missed super items. Let J be the set of knapsacks with non-empty missed super items. For every $j \in J$, the super bucket must be full (event $\mathcal{E}_{j,K+1,1}$ holds). By construction, whenever these queried super items are assigned to knapsack j, each of them has value at least M_j/ϵ. In particular, this implies that the realized contribution of these super items, together with the realized contribution of knapsack j to OPT, already accounts for a substantial portion of the total optimum. Consequently, the expected loss incurred by those missed super items is small relative to the overall expected gain. Summing over all $j \in J$, we obtain that the total value of missed super items across J is bounded by $O(\epsilon)\,\mathbb{E}[\mathrm{OPT}]$. ◀

▶ **Lemma 13** (Completeness). *Upon completion of* ReconstructionProcedure *(Algorithm 3), we have $\overline{\mathbf{OPT}} = \mathbf{OPT}$.*

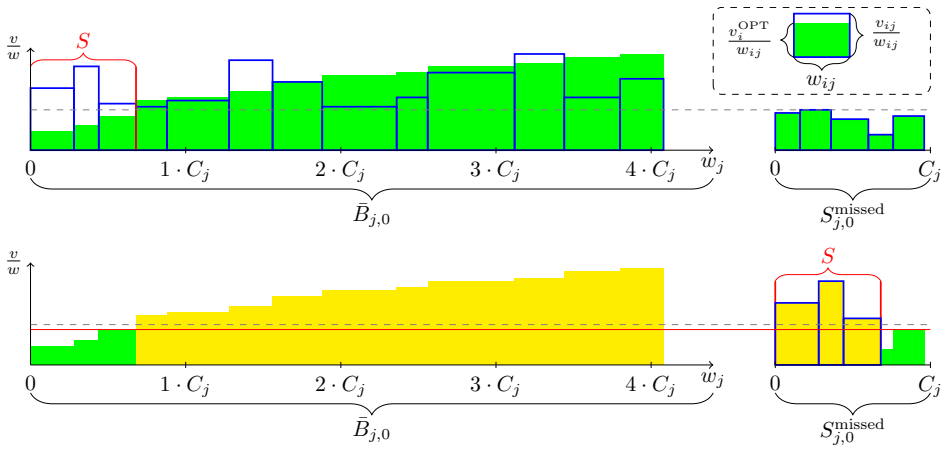
Proof. The sets $S_{j,k}^{\text{queried}}$ and $S_{j,k}^{\text{missed}}$ form a partition of $\mathbf{OPT}$. The subroutines iterate through all buckets, adding every element of this partition exactly once to $\overline{\mathbf{OPT}}$. ◀

7 Conclusion

We introduced a polyhedral framework for sparsification that extends beyond uniform structures such as matching and matroids to capacity-constrained problems including knapsack, multiple knapsack, and the generalized assignment problem. Our results demonstrate that despite the inherent hardness of these problems, one can construct $(1-\epsilon)$-approximate sparsifiers with degree polynomial in $1/p$ and $1/\epsilon$, independent of the problem size. This establishes a clean separation between optimization complexity and sparsification complexity: while exact or near-exact optimization remains intractable, identifying a small query set that preserves optimality up to $(1-\epsilon)$ is efficiently possible.



Case 1: Green rectangles represent $\overline{\mathbf{OPT}}$'s value gain, blue rectangles show item values in knapsack j, and yellow rectangles represent **ALG**'s value gain. Set $S \subseteq \overline{B}_{j,k}$ is the largest prefix with $w_j(S) \leq w_j(S^{\mathrm{missed}}_{j,k})$. **ALG** reassigns all items in S to knapsack j, nearly covering the missed set with at most $\mathrm{poly}(\epsilon)$ loss. The uncovered value (green rectangles) constitutes at most $\frac{1}{\alpha}$ of $\overline{\mathbf{OPT}}$'s value.



Case 2: In this case, the set $S \subseteq \overline{B}_{j,k}$ is the largest prefix where $v_{ij} > v_i^{\mathrm{OPT}}$ for all $i \in S$; items not in S have sufficiently high value, making substitution unnecessary. **ALG** reassigns all items in S to knapsack j. The red line separates green rectangles (below/on line) from yellow rectangles (above line), showing the uncovered portion corresponds to lowest-density items in $\overline{\mathbf{OPT}}$, constituting at most $\frac{1}{\alpha}$ of $\overline{\mathbf{OPT}}$'s value.

Figure 2 Visualization of substitution in FillSmallBucket($\overline{\mathbf{OPT}}$, ALG, j, 0). Assume $\alpha = 5$. Each subfigure shows bucket $\overline{B}_{j,0}$, missed set $S^{\mathrm{missed}}_{j,0}$, and selected subset $S \subseteq \overline{B}_{j,0}$. Items are rectangles with width w_{ij}, height v/w, and area representing value. **ALG** substitutes S for $S^{\mathrm{missed}}_{j,0}$ in knapsack j, recovering $\frac{4}{5} = 1 - \frac{1}{\alpha}$ of $\overline{\mathbf{OPT}}$'s value.

More broadly, our work highlights sparsification as a lens for rethinking stochastic combinatorial optimization. The polyhedral notion of degree captures structural redundancy without relying on cardinality, suggesting applications far beyond knapsack-type problems. A central open question remains: can we design size-independent sparsifiers for general integer linear programs, with degree depending only on $1/p$, $1/\epsilon$, and intrinsic structural parameters? Progress on this front would push the boundary of query-efficient optimization and clarify the fundamental role of sparsification in stochastic combinatorial optimization.

References

1. Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *2nd Symposium on Simplicity in Algorithms (SOSA 2019)*, volume 69 of *OASIcs*, pages 11:1–11:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.SOSA.2019.11.
2. Sepehr Assadi, Sanjeev Khanna, and Yang Li. The stochastic matching problem with (very) few queries. *ACM Transactions on Economics and Computation (TEAC)*, 7(3):16:1–16:19, 2019. doi:10.1145/3355903.
3. Amir Azarmehr, Soheil Behnezhad, Alma Ghafari, and Ronitt Rubinfeld. Stochastic matching via in-n-out local computation algorithms. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing (STOC 2025)*, pages 1055–1066. ACM, 2025. doi:10.1145/3717823.3718279.
4. Soheil Behnezhad, Avrim Blum, and Mahsa Derakhshan. Stochastic vertex cover with few queries. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 1808–1846. SIAM, 2022. doi:10.1137/1.9781611977073.73.
5. Soheil Behnezhad and Mahsa Derakhshan. Stochastic weighted matching: (1 - ε)-approximation. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1392–1403. IEEE, 2020. doi:10.1109/FOCS46700.2020.00131.
6. Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Stochastic matching with few queries: (1 - ε)-approximation. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2020)*, pages 1111–1124. ACM, 2020. doi:10.1145/3357713.3384340.
7. Soheil Behnezhad, Alireza Farhadi, MohammadTaghi Hajiaghayi, and Nima Reyhani. Stochastic matching with few queries: New algorithms and tools. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 2855–2874. SIAM, 2019. doi:10.1137/1.9781611975482.177.
8. Avrim Blum, John P. Dickerson, Nika Haghtalab, Ariel D. Procaccia, Tuomas Sandholm, and Ankit Sharma. Ignorance is almost bliss: Near-optimal stochastic matching with few queries. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation (EC 2015)*, pages 325–342. ACM, 2015. doi:10.1145/2764468.2764479.
9. Deeparnab Chakrabarty and Gagan Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and GAP. *SIAM J. Comput.*, 39(6):2189–2211, 2010. doi:10.1137/080735503.
10. Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, 2005. doi:10.1137/S0097539700382820.
11. Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. A nearly quadratic-time FPTAS for knapsack. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC 2024)*, pages 283–294. ACM, 2024. doi:10.1145/3618260.3649730.
12. Mahsa Derakhshan, Naveen Durvasula, and Nika Haghtalab. Stochastic minimum vertex cover in general graphs: A 3/2-approximation. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, pages 242–253. ACM, 2023. doi:10.1145/3564246.3585230.
13. Mahsa Derakhshan and Mohammad Saneian. Query efficient weighted stochastic matching. In *52nd International Colloquium on Automata, Languages, and Programming (ICALP 2025)*, volume 334 of *LIPIcs*, pages 67:1–67:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.67.
14. Shaddin Dughmi, Yusuf Hakan Kalayci, and Xinyu Liu. Near-optimal sparsifiers for stochastic knapsack and assignment problems. *arXiv*, abs/2512.01240, 2025. arXiv:2512.01240.
15. Shaddin Dughmi, Yusuf Hakan Kalayci, and Neel Patel. On sparsification of stochastic packing problems. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023)*, volume 261 of *LIPIcs*, pages 51:1–51:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.51.

16 Uriel Feige and Jan Vondrák. Approximation algorithms for allocation problems: Improving the factor of $1 - 1/e$. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006)*, pages 667–676. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.14.

17 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

19 Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer, Boston, MA, 1972. doi:10.1007/978-1-4684-2001-2_9.

20 Takanori Maehara and Yutaro Yamaguchi. Stochastic monotone submodular maximization with queries. *CoRR*, abs/1907.04083, 2019. arXiv:1907.04083.

21 Xiao Mao. $(1 - \varepsilon)$-approximation of knapsack in nearly quadratic time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC 2024)*, pages 295–306. ACM, 2024. doi:10.1145/3618260.3649677.

22 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

23 Yutaro Yamaguchi and Takanori Maehara. Stochastic packing integer programs with few queries. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 293–310. SIAM, 2018. doi:10.1137/1.9781611975031.21.