A Comparative Analysis of Complexity and Approximation Schemes for the One-Dimensional Bin Packing Problem

Shreyas Ramasubramanian Shreyash Chandak
Pranav Swarup Kumar Aman Jayesh Mukund Hebbar

Project Repository: `https://github.com/Pranav-Swarup/aadcourseproject`

Abstract

The **One-Dimensional Bin Packing Problem (1DBPP)** is a foundational challenge in combinatorial optimization, recognized as **NP-hard**. This report systematically analyzes the problem's complexity, approximation limitations, and state-of-the-art solution methods. We begin by establishing the theoretical baseline with a review of classic simple heuristics, including **First-Fit (FF)**, **Best-Fit (BF)**, and their decreasing counterparts, noting the tightest known worst-case approximation ratios, such as 11/9 for the offline **First-Fit Decreasing (FFD)** strategy. Given the 3/2 constant-factor lower bound for approximation, achieving better performance requires more sophisticated techniques. We then detail the construction and analysis of an **Asymptotic Polynomial-Time Approximation Scheme (APTAS)**, which achieves a near-optimal solution guarantee of $(1+\varepsilon)\cdot\mathrm{OPT}+O(1)$. Finally, we discuss recent advancements in additive approximation, tracing the development to the modern $\mathrm{OPT}+\mathbf{O}(\log \mathrm{OPT})$ bounds achieved by approaches from Rothvoß and Hoberg & Rothvoß, which leverage Configuration Linear Programs and represent the current theoretical limit of bin packing research.

Contents

1 Introduction to the Problem

The One-Dimensional Bin Packing Problem (1DBPP) is a classical optimization problem, formally defined as follows: Given a list of items $L = \{l_1, l_2, \ldots, l_n\}$ where each item l_i has a size $s(l_i) \in (0, 1]$, and a collection of identical bins, each with a capacity of 1, the objective is to partition L into the minimum number of subsets $B_1, B_2, \ldots, B_m$ such that the sum of the sizes of items in each subset B_j does not exceed 1. The minimal number of bins required for a list L is denoted by $\mathrm{OPT}(L)$.

The 1D BPP is a fundamental abstraction that captures the core difficulty of efficiently packing limited resources. Its real-world significance extends across many industries, because any scenario involving the allocation of items into fixed-capacity containers can often be mapped directly to a Bin Packing instance.

- **Manufacturing and Industrial Engineering:** The classical *Cutting Stock Problem* is a direct analogue: raw materials such as metal rods, wooden boards, paper rolls, or fabric sheets must be cut into required lengths or patterns. Efficient packing reduces scrap, decreases production cost, improves sustainability by minimizing waste, and directly impacts profit margins in large-scale operations (e.g., steel mills or textile plants).
- **Logistics and Transportation:** Packing heterogeneous items into shipping containers, pallets, or trucks can be modelled as Bin Packing. Better packing means fewer vehicles, reduced fuel consumption, improved route efficiency, and major cost savings in large distribution networks (e.g., Amazon, FedEx, international freight). Even airline baggage loading and ferry car-deck planning rely on variants of the problem.
- **Cloud Computing and Data Centers:** Virtual machine placement, CPU-time scheduling, and allocating workloads to servers all reduce to packing tasks with resource requirements (CPU, memory, bandwidth) into machines of fixed capacity. Efficient algorithms lower energy consumption, reduce the number of active servers, and improve service quality in systems like AWS, Google Cloud, or Kubernetes clusters.
- **Computer Systems and Memory Allocation:** File systems and operating systems often store files or memory blocks in fixed-sized units. Mapping variable-sized data blocks to fixed-size pages, or scheduling tasks onto limited execution slots, is naturally captured by Bin Packing. Better algorithms reduce fragmentation and improve throughput.
- **Telecommunications and Networking:** Data packets of varying sizes must be aggregated into frames or transmission blocks with capacity constraints. Efficient packing increases bandwidth utilization, reduces latency, and improves overall network throughput in systems such as 5G uplink scheduling or IP packet batching.
- **Finance and Portfolio Optimization:** When investments must be divided across fixed-capacity financial "buckets" such as risk categories or regulatory capital limits, assigning assets to these constrained containers can be viewed as a Bin Packing variant. This perspective helps reduce over-allocation and improves compliance with market or regulatory constraints.
- **Robotics and Automation:** Automated warehouses (e.g., Kiva robots in Amazon fulfillment centers) must pack items into bins on robots or shelves efficiently. Accurate packing minimizes robot travel time, increases throughput, and improves inventory density.

Beyond these specific examples, the 1DBPP serves as a canonical NP-hard optimization problem whose techniques, such as approximation schemes, rounding methods, and combinatorial heuristics, form the algorithmic foundation for many modern resource-allocation systems.

The problem's theoretical significance stems from its inherent computational difficulty. Bin packing was among the first problems proved to be **NP-hard** by reduction from the PARTITION problem, meaning that finding an exact, optimal solution is intractable for large instances unless P=NP. This complexity necessitates the use of approximation algorithms, which are polynomial-time methods that guarantee a solution close to the optimal one.

The field of bin packing approximation is broadly categorized by the nature of the approximation guarantee:

1. **Constant-Factor Approximation:** Algorithms, often simple greedy heuristics like First-Fit, that guarantee the solution $\mathrm{ALG}(L)$ is bounded by a constant factor of the optimum, i.e., $\mathrm{ALG}(L) \leq c \cdot \mathrm{OPT}(L)$, where c is typically less than 2.

2. **Asymptotic Polynomial-Time Approximation Schemes (APTAS):** Algorithms that achieve a factor of $(1+\varepsilon)$, where $\varepsilon > 0$ can be arbitrarily small, but at the cost of an additive term: $\mathrm{ALG}(L) \leq (1+\varepsilon) \cdot \mathrm{OPT}(L) + O(1)$. The running time depends polynomially on the input size n but exponentially on $1/\varepsilon$.

3. **Additive Approximation Schemes:** Recent, highly sophisticated methods that aim to minimize the additive error term, achieving results like $\mathrm{ALG}(L) \leq \mathrm{OPT}(L) + O(\log \mathrm{OPT})$.

This report will follow the historical and logical progression of bin packing research. We will first examine the constant-factor bounds for classical heuristics. Subsequently, we detail the structural insights required to develop an APTAS, demonstrating how item rounding and dynamic programming overcome the inherent difficulty of the problem. Finally, we review the advanced techniques that have led to the current state-of-the-art results in additive approximation.

2 Proof of NP-hardness for the 1D Bin Packing Problem

2.1 Optimization Problem Definition

The **one-dimensional Bin Packing Problem (BPP)** can be stated as follows.

Instance. A finite multiset of items

$$S = \{s_1, s_2, \ldots, s_n\}, \quad s_i \in (0, 1].$$

Each s_i denotes the size of item i, and each bin has unit capacity.

Objective. Partition S into the minimum number of disjoint subsets (bins)

$$B_1, B_2, \ldots, B_m$$

such that, for every bin B_j,

$$\sum_{s_i \in B_j} s_i \leq 1.$$

The goal is to minimize m. We denote the optimal number of bins by

$$OPT(S) = \min\{\, m \mid \exists\, B_1, \ldots, B_m \text{ satisfying the above constraint} \,\}.$$

2.2 Decision Problem Definition

To analyze computational complexity, the optimization problem is converted into a decision problem, which asks a yes/no question instead of minimizing a quantity.

Formally, the decision version of the 1-D Bin Packing Problem is defined as follows.

Instance. A multiset of items

$$S = \{s_1, s_2, \ldots, s_n\}, \quad s_i \in (0, 1],$$

and an integer $k > 0$.

Question. Does there exist a feasible packing of the items into at most k bins such that

$$\forall j \in \{1, \ldots, k\}, \quad \sum_{s_i \in B_j} s_i \leq 1,$$

and every item appears in exactly one bin?

We denote this decision problem as BIN-PACK-DEC. Formally:

$$\text{BIN-PACK-DEC}(S, k) = \begin{cases} 1, & \text{if } \exists \{B_1, \ldots, B_k\} \text{ with } \sum_{s_i \in B_j} s_i \leq 1, \\ 0, & \text{otherwise.} \end{cases}$$

Relationship between Optimization and Decision Versions. The optimization problem and its decision version are *polynomially equivalent* in the sense that

$$OPT(S) = \min\{\, k \mid \text{BIN-PACK-DEC}(S, k) = \text{“YES”} \,\}.$$

Hence, if the decision problem could be solved in polynomial time, the optimal packing number could be obtained by binary search over $k \in \{1, \ldots, n\}$, implying a polynomial-time algorithm for the optimization form. Therefore, the decision version is NP-complete if and only if the optimization version is NP-hard.

2.3 Reduction from Subset-Sum to Partition to Bin-Pack-Dec

We establish the computational intractability of the one-dimensional Bin Packing problem by a chain of polynomial-time reductions. We first reduce the classical Subset-Sum decision problem to Partition, and then reduce Partition to the decision version of Bin Packing (Bin-Pack-Dec). By transitivity of polynomial reductions and standard NP-completeness results, this shows that Bin-Pack-Dec is NP-complete and the optimization version of Bin Packing is NP-hard.

The full reduction is illustrated in an animation made specifically for this report by our team.[1]

2.3.1 Reduction Subset-Sum $\leq_p$ Partition.

Let $(a_1, \ldots, a_n; t)$ be an instance of Subset-Sum, and denote $A := \sum_{i=1}^{n} a_i$.

Preprocessing. If $t > A$ then the instance is trivially a NO instance; return any fixed NO instance of Partition. Otherwise, if $t > A/2$ replace t by $A - t$. This substitution is valid since a subset sums to t iff its complement sums to $A - t$. After this step we have $0 \leq t \leq A/2$.

Construction. Define an instance of Partition by appending a single integer

$$b \;:=\; A - 2t \;\in\; \mathbb{Z}_{\geq 0}$$

to the original multiset. That is, form

$$S' \;=\; \{a_1, \ldots, a_n,\, b\}.$$

The total sum of S' is

$$A' \;=\; \sum_{i=1}^{n} a_i + b \;=\; A + (A - 2t) \;=\; 2(A - t),$$

hence A' is even and $A'/2 = A - t$.

Correctness.

($\Rightarrow$) If there exists $I \subseteq \{1, \ldots, n\}$ with $\sum_{i \in I} a_i = t$, then $I' := I \cup \{b\}$ satisfies

$$\sum_{i \in I'} s_i = t + (A - 2t) = A - t = A'/2,$$

so S' admits a partition into two equal-sum subsets.

($\Leftarrow$) Conversely, suppose S' admits a partition into two subsets of sum $A'/2 = A - t$. The element b must lie in one of the two parts; removing b from that part yields a subset of $\{a_1, \ldots, a_n\}$ whose sum is

$$(A - t) - b = (A - t) - (A - 2t) = t,$$

hence the original Subset-Sum instance is a YES instance.

[1] https://www.youtube.com/watch?v=xNtxqb9TEok

The mapping adds one integer and performs only arithmetic operations on the input integers; thus it runs in polynomial time and preserves YES/NO answers. Therefore Subset-Sum $\leq_p$ Partition.

2.3.2 Reduction Partition $\leq_p$ Bin-Pack-Dec.

Let $(a_1, \ldots, a_n)$ be an instance of Partition and set $A := \sum_{i=1}^{n} a_i$.

Preprocessing. If there exists i with $a_i > A/2$ then the PARTITION instance is immediately NO; return a fixed NO instance of Bin-Pack-Dec. Otherwise all $a_i \leq A/2$.

Construction. Create a Bin Packing instance by scaling:

$$s_i := \frac{2a_i}{A} \in (0, 1], \qquad i = 1, \ldots, n,$$

and set $k := 2$. Note that

$$\sum_{i=1}^{n} s_i = \frac{2}{A} \sum_{i=1}^{n} a_i = 2.$$

Correctness.

($\Rightarrow$) If the a_i admit a partition I with $\sum_{i \in I} a_i = A/2$, then the corresponding items satisfy

$$\sum_{i \in I} s_i = \frac{2}{A} \sum_{i \in I} a_i = 1,$$

so I and its complement form two bins of capacity 1 and the Bin Packing decision instance is YES.

($\Leftarrow$) Conversely, if the s_i can be packed into two unit bins, each bin must have total size exactly 1 (since the grand total is 2). Scaling back by $A/2$ yields a partition of the a_i into two subsets of sum $A/2$.

The scaling mapping is computable in polynomial time (rational arithmetic) and preserves YES/NO answers, hence Partition $\leq_p$ Bin-Pack-Dec.

2.3.3 Transitivity and conclusion.

Polynomial-time reducibility is transitive. Combining the two reductions above we obtain

$$\text{Subset-Sum} \leq_p \text{Partition} \leq_p \text{Bin-Pack-Dec},$$

whence Subset-Sum $\leq_p$ Bin-Pack-Dec. Since Subset-Sum is NP-complete (see [1]), Bin-Pack-Dec is NP-hard. As Bin-Pack-Dec is in NP (a packing into k bins is polynomially verifiable), it follows that Bin-Pack-Dec is NP-complete (see [3]). Consequently the optimization form of one-dimensional Bin Packing is NP-hard.

2.4 Remarks on edge cases and polynomiality

The reductions above include simple preprocessing to handle degenerate inputs. In the reduction SUBSET-SUM $\leq_p$ PARTITION we replace the target t by $A - t$ when $t > A/2$; this step is valid because a subset sums to t iff its complement sums to $A - t$. If $t > A$ the SUBSET-SUM instance is trivially NO. The appended integer $b = A - 2t$ is non-negative by construction; if $b = 0$ the appended zero does not alter partitionability. In the reduction PARTITION $\leq_p$ BIN-PACK-DEC we first check whether any $a_i > A/2$: if so, PARTITION is immediately NO and we may output a fixed NO instance of BIN-PACK-DEC. Otherwise every scaled size $s_i = 2a_i/A$ satisfies $0 < s_i \leq 1$, so the constructed Bin Packing instance is valid. All arithmetic performed (sums, subtraction, a single division by A) is polynomial-time with respect to the binary encoding of the input integers; the bit-length of intermediate integers remains polynomially bounded. Thus both mappings are polynomial-time reductions in the standard Turing model.

3 Classification of Heuristic Algorithms

As established in the project's complexity analysis, this problem is NP-hard, necessitating the use of approximation algorithms.

We categorize the foundational heuristics based on information availability:

- **Online Algorithms (FF, BF):** The algorithm must pack item s_i immediately without knowledge of subsequent items $s_{i+1}, \ldots, s_n$.
- **Offline Algorithms (FFD, BFD):** The algorithm possesses global knowledge of the input set I and may sort or manipulate the sequence prior to packing.

3.1 First-Fit (FF)

3.1.1 Mechanics and Complexity

The First-Fit algorithm processes items in the order they arrive. For each item s_i, it scans the existing bins $B_1, B_2, \ldots, B_k$ sequentially and places the item in the first bin B_j such that the residual capacity $r(B_j) \geq s_i$. If no such bin exists, a new bin B_{k+1} is opened.

While a naive implementation requires $O(n^2)$ time (scanning all previous bins for every item), the algorithm can be optimized to $O(n \log n)$ using a segment tree or a similar data structure to query the first valid bin efficiently.

3.1.2 Theoretical Bounds

The performance of First-Fit is bounded by an asymptotic approximation ratio of 1.7.

Theorem 1 (Approximation Ratio of FF) *For any list of items L, the number of bins used by First-Fit, $FF(L)$, satisfies:*

$$FF(L) \leq 1.7 \cdot OPT(L) + 2$$

where $OPT(L)$ is the optimal number of bins.

3.2 Best-Fit (BF)

3.2.1 Mechanics and Complexity

The Best-Fit algorithm attempts to minimize immediate space wastage. For an item s_i, it searches for a bin B_j that minimizes the residual capacity $r(B_j) - s_i$, subject to $r(B_j) \geq s_i$. Ties are typically broken by choosing the lowest index.

Like FF, a naive implementation is $O(n^2)$. However, by maintaining the bins in a balanced Binary Search Tree (BST) ordered by remaining capacity, the best-fitting bin can be identified and updated in $O(\log n)$ time, resulting in an overall complexity of $O(n \log n)$.

3.2.2 Theoretical Bounds

Despite the strategic difference, Best-Fit shares the same worst-case asymptotic ratio as First-Fit.

Theorem 2 (Approximation Ratio of BF) *For any list of items L:*

$$BF(L) \leq 1.7 \cdot OPT(L) + C$$

Although the worst-case bounds are identical, empirical average-case performance often differs due to the fragmentation effects discussed in Section 3.4.

3.3 Decreasing Variants (FFD and BFD)

The offline heuristics differ from their online counterparts solely by the inclusion of a pre-processing step: the items are sorted in non-increasing order of size such that $s_1 \geq s_2 \geq \cdots \geq s_n$.

This strategy, analogous to placing "big rocks" into a jar before "sand," ensures that large items—which are hardest to pack—are processed when the maximum number of bins have maximum capacity available.

3.3.1 Theoretical Bounds

Sorting the items significantly improves the approximation guarantee.

Theorem 3 (Johnson's Theorem, 1973) *For the First-Fit Decreasing (FFD) algorithm:*

$$FFD(L) \leq \frac{11}{9}OPT(L) + 4$$

Theorem 4 (Tight Bound, Dósa 2007) *The bound was later tightened to:*

$$FFD(L) \leq \frac{11}{9}OPT(L) + \frac{6}{9}$$

This implies an asymptotic approximation ratio of approximately 1.22, a substantial improvement over the 1.7 ratio of the online variants. Best-Fit Decreasing (BFD) achieves the same asymptotic bound of 11/9.

3.4 Qualitative Comparison: The Fragmentation Trade-off

While FF and BF share identical worst-case asymptotic bounds (1.7), their internal packing dynamics differ fundamentally regarding space fragmentation.

3.4.1 The Best-Fit "Sand" Problem

Best-Fit is designed to minimize the residual space in the chosen bin. While locally optimal, this strategy often creates bins that are nearly full but contain tiny, unusable gaps (often referred to as "sand" or "splinters").

- **Consequence:** These small gaps are often too small to accommodate even the smallest future items, effectively rendering that capacity wasted.

3.4.2 The First-Fit "Gap" Advantage

First-Fit is oblivious to the "tightness" of the fit; it simply selects the first valid option.

- **Consequence:** This often leaves larger, contiguous chunks of free space in the earlier bins. These larger gaps are statistically more likely to accommodate future items than the fragmented slivers created by Best-Fit.

3.5 Heuristics Conclusion

The foundational heuristics present a clear hierarchy of efficiency versus complexity. The move from Online (FF/BF) to Offline (FFD/BFD) yields a quantifiable improvement in the approximation ratio (from 1.7 to ≈ 1.22) at the cost of the $O(n \log n)$ sorting requirement. These algorithms serve as the baseline against which advanced structured heuristics, such as Harmonic-k, and meta-heuristics, such as the Grouping Genetic Algorithm, must be measured.

4 Polynomial-Time Approximation Schemes

4.1 Introduction to Approximation Schemes

The analysis of classical greedy algorithms demonstrates an inherent limitation: the optimal solution $\text{OPT}(L)$ can be arbitrarily large, yet the ratio $\text{ALG}(L)/\text{OPT}(L)$ is bounded by a constant strictly greater than 1 (e.g., 1.222 or 1.7). Achieving a performance guarantee arbitrarily close to optimal, specifically a ratio of $(1+\varepsilon)$ for any $\varepsilon > 0$, requires a fundamentally different algorithmic approach.

A Polynomial-Time Approximation Scheme (PTAS) achieves an approximation ratio of $1 + \varepsilon$ for any fixed $\varepsilon > 0$. However, due to the volume-based lower bound $\sum s_i \leq \text{OPT}(L)$, the solution $\text{ALG}(L)$ can still be $1+\varepsilon$ times larger than $\text{OPT}(L)$, even if $\text{OPT}(L)$ is small. For instance, if $\text{OPT}(L) = 2$ and $\varepsilon = 0.1$, $\text{ALG}(L)$ might be $\lfloor 2(1.1) \rfloor = 2$.

The Bin Packing Problem admits an **Asymptotic Polynomial-Time Approximation Scheme (APTAS)**, which relaxes the requirement by allowing a small additive error term:

$$\text{APTAS}(L) \leq (1 + \varepsilon) \cdot \text{OPT}(L) + O(1)$$

where the constant $O(1)$ depends only on $1/\varepsilon$ and not on the input size n or $\text{OPT}(L)$. The $O(1)$ term ensures that for large instances (where $\text{OPT}(L) \to \infty$), the performance

guarantee approaches $1 + \varepsilon$. This is the best we can hope for in polynomial time, as no PTAS is known for 1DBPP.

The key to developing an APTAS lies in the observation that items of very small size contribute significantly to the total number of items (n) but contribute very little to the total volume (and thus OPT(L)). The APTAS approach systematically handles the complexity introduced by large items using exact methods, while managing the small items with efficient, bounded-error heuristics. This is achieved through the technique of rounding and configuration enumeration.

A brief pointer: classical greedy and decreasing heuristics (FF/BF/FFD/BFD) are summarized earlier; we continue with the fundamental 3/2 hardness baseline that motivates asymptotic schemes.

4.2 The 3/2 Inapproximability baseline (Partition → Bin Packing)

Before discussing PTAS and APTAS algorithms for Bin Packing, it is crucial to understand the *baseline limitation* on what any polynomial-time approximation algorithm can achieve. A simple but powerful idea: even extremely restricted instances of Bin Packing are already expressive enough to encode the PARTITION problem. This yields the fundamental barrier that no polynomial-time algorithm can guarantee an approximation ratio smaller than 3/2 unless P = NP. [14]

4.3 Why Partition appears inside Bin Packing

Consider the decision version of PARTITION: given positive integers $a_1, \ldots, a_n$, determine whether they can be split into two subsets with equal sum. The intuitive connection to Bin Packing is that Partition asks:

Can we pack everything into *two* perfectly filled bins?

If the total sum is A, then this is the same as asking whether we can fill two bins of capacity $A/2$ exactly. By scaling these numbers so that $A/2$ becomes bin capacity 1, the Partition instance becomes a Bin Packing instance whose optimal solution is either

$$\text{Opt} = 2 \quad \text{(YES instance)} \qquad \text{or} \qquad \text{Opt} \geq 3 \quad \text{(NO instance)}.$$

Thus, distinguishing between Opt = 2 and Opt ≥ 3 would solve Partition.

4.4 Why a $(3/2 - \delta)$-approximation would decide Partition

Suppose there exists a Bin Packing algorithm A running in polynomial time and achieving approximation factor $\rho < 3/2$. We show this implies a polynomial-time solution for PARTITION, which is believed unlikely.

Recall the crucial gap from the reduction:

$$\text{Opt} = 2 \quad \text{or} \quad \text{Opt} \geq 3.$$

These are the only two possibilities for the special instances derived from Partition.

Now run the hypothetical ρ-approximation algorithm A on this instance. We obtain:

$$A(I) \leq \rho \cdot \text{Opt}(I).$$

Case 1: Partition instance is a YES instance. Then Opt $= 2$, so the algorithm must return

$$A(I) \leq \rho \cdot 2 < 3.$$

Since $A(I)$ is an integer, we must have $A(I) = 2$.

Case 2: Partition instance is a NO instance. Then Opt ≥ 3, so

$$A(I) \geq 3.$$

Thus,

$$A(I) = \begin{cases} 2, & \text{if Partition is YES,} \\ \geq 3, & \text{if Partition is NO.} \end{cases}$$

This perfectly distinguishes YES from NO, solving Partition in polynomial time. Therefore, such an approximation ratio $\rho < 3/2$ cannot exist unless P = NP.

4.5 Why the number 3/2 is the magic threshold

The number 3/2 arises precisely because:

$$\frac{3}{2} \cdot 2 = 3.$$

This is the point where the "YES-branch" of the approximation is allowed to touch the "NO-branch" without giving away the answer to Partition.

If the approximation factor were any smaller than 3/2, even by an arbitrarily tiny amount:

$$\rho = \frac{3}{2} - \delta,$$

then:

$$\rho \cdot 2 = 3 - 2\delta < 3,$$

forcing the algorithm to return 2 bins on YES instances and ≥ 3 bins on NO instances, solving Partition.

Thus 3/2 is the *minimal* ratio consistent with respecting NP-hard structure. Everything better than this would collapse P and NP.

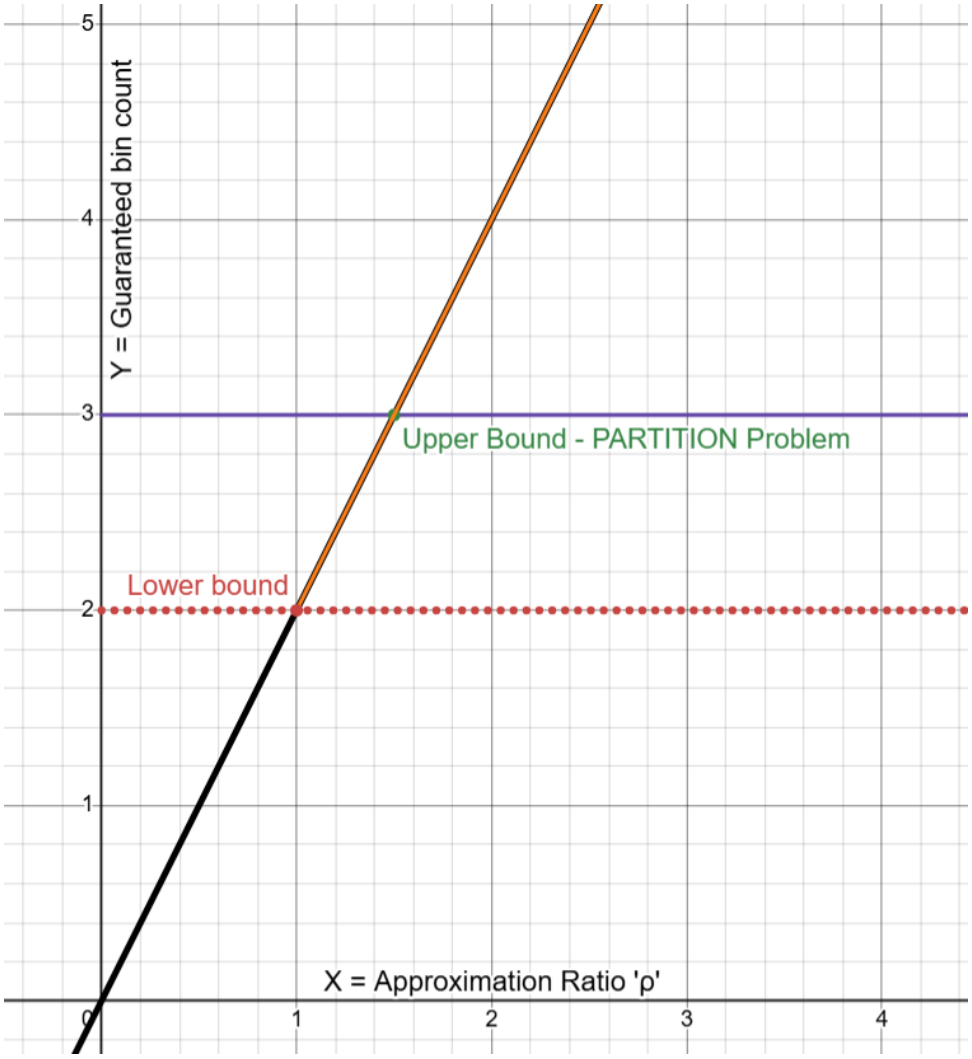

Figure 1: Approximation Ratio Bounds

4.6 Interpretation

This reduction sets a "hard floor" in the landscape of approximation algorithms for Bin Packing. Any algorithm with a better-than-3/2 guarantee would violate standard complexity assumptions. Therefore, the only hope for improving performance is by relaxing the problem variant (asymptotic vs. classical guarantee), or allowing $(1 + \varepsilon)$ approximation at the cost of time exponential in $1/\varepsilon$ which is exactly the domain of PTAS and APTAS algorithms that we study in the following sections.

4.7 Special Cases Admitting Polynomial-Time Solutions

Before presenting the APTAS itself, we highlight that certain restricted versions of Bin Packing are solvable in polynomial time. These special cases help reveal the structure that an APTAS will exploit. In particular, if we assume that

(a) there are only *constantly many distinct item sizes*, and

(b) only a *constant number of items* can fit in any bin,

then the number of feasible bin configurations (or *types*) becomes bounded by a constant. In this setting, one may enumerate all bin types, count how many bins of each type are needed to cover the items, and solve the resulting instance optimally in polynomial time.

This observation is the conceptual base for the APTAS: if we can *transform* a general instance into one that behaves like this restricted case (without losing much optimality), then we can solve the transformed instance exactly and transfer the solution back to the original with only a small loss.

5 APTAS for Bin Packing

5.1 Intuitive Motivation

The baseline hardness barrier (Section 4.2) shows that no polynomial-time algorithm can approximate Bin Packing within a factor smaller than $3/2$ unless P = NP. Thus, to obtain near-optimal packings with guarantees arbitrarily close to 1, we must accept algorithms whose running time may depend super-polynomially on $1/\varepsilon$ for accuracy parameter $\varepsilon > 0$. The goal is to carefully structure such algorithms so they remain polynomial in n for any fixed ε.

Bin Packing has enough internal structure that we can 'adjust' an arbitrary instance into one resembling the polynomial-time special case: only a constant number of significant item sizes remain, and every bin contains only a small, bounded number of large items. This transformation, performed with controlled rounding and grouping, leads to an asymptotically optimal solution.

Definition 1 (Asymptotic PTAS (APTAS)) *For every fixed $\varepsilon > 0$, an* asymptotic polynomial-time approximation scheme *for Bin Packing is a family of algorithms $\{A_\varepsilon\}_{\varepsilon>0}$ satisfying:*

$$A_\varepsilon(I) \;\le\; (1+\varepsilon)\,\mathrm{OPT}(I) \;+\; O(1),$$

where each A_ε runs in time polynomial in the number of items n (but not necessarily in $1/\varepsilon$). The additive $O(1)$ term is independent of n and is negligible for large instances.

5.2 The Four-Step APTAS Framework

We present APTAS via four conceptual steps. Below we expand these steps with intuitive explanations, the precise reasoning behind them, and how each contributes to the global strategy. [14]

5.2.1 Step 1: Remove the Smaller Items

Motivation. Small items (those of size $< \varepsilon/2$) are the primary source of irregular leftover gaps inside bins. Large items, in contrast, largely determine the bin structure. By temporarily discarding small items, we extract the "skeleton" of the instance and obtain a much more structured problem.

Procedure. Partition the items into:

$$L = \{\, i : s_i \geq \varepsilon/2 \,\}, \qquad S = \{\, i : s_i < \varepsilon/2 \,\}.$$

Solve the problem for the large items only. Later, reinsert small items greedily into the bins created for L.

Impact. Analytically, either the small items fit into the existing bins (ideal case), or almost all bins are sufficiently full that only a constant number of extra bins are needed. This is key to achieving the asymptotic $(1 + \varepsilon)$ guarantee.

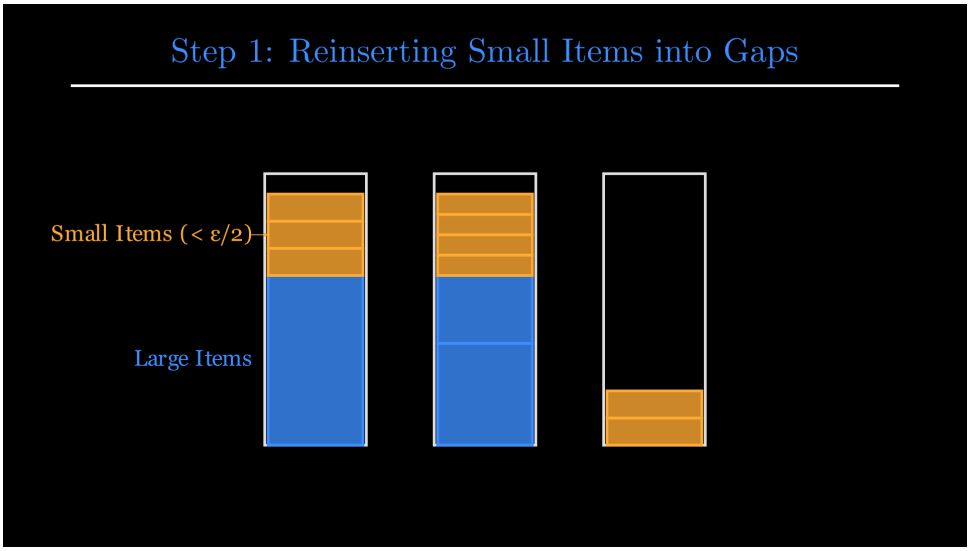

Figure 2: Step 1 APTAS for Bin-Packing

5.2.2 Step 2: Linear Grouping and Rounding

Motivation. Even after isolating the large items, there may still be many distinct sizes. We want to *compress* these sizes into only $O(1/\varepsilon^2)$ distinct classes, allowing us to use the polynomial-time special-case algorithm.

Procedure. Sort L in non-increasing order and divide it into groups of size k, where k is chosen so that $k \leq \varepsilon \cdot \mathrm{OPT}$. Within each group, *round up* each item to the largest item in that group, forming a rounded multiset L'. This reduces the number of distinct sizes dramatically.

Impact. Rounding up only increases bin usage by at most k bins, which is carefully tuned to be at most $\varepsilon \cdot \mathrm{OPT}$. We see this with stacked bars, showing each group collapsing upward to a uniform height.

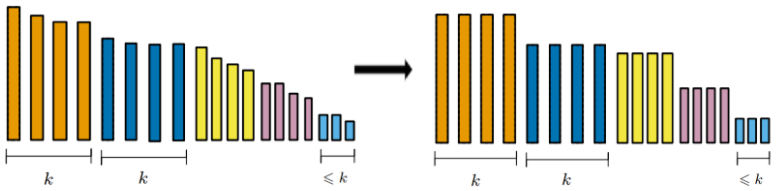

Figure 3: Step 2 APTAS for Bin-Packing

5.2.3 Step 3: Solve the Bounded-Type Instance Exactly

Motivation. Once rounding is complete, the instance has:

$$c_s = O(1/\varepsilon^2) \quad \text{distinct sizes}, \qquad c_b = O(1/\varepsilon) \quad \text{items per bin}.$$

The number of possible bin types is now constant:

$$(c_b + 1)^{c_s} = O_\varepsilon(1).$$

This reduces the problem to choosing nonnegative integers $x_1, \ldots, x_T$ giving how many bins of each type to use.

Procedure. Enumerate all feasible bin configurations (each is a small integer vector indicating how many items of each size the bin contains). Then solve the exact counting problem:

$$\sum_{t=1}^{T} x_t \cdot (\text{type } t \text{ vector}) = (\text{item multiplicity vector}).$$

A dynamic program or bounded-integer linear program (size depending only on ε) finds an *optimal* solution.

Impact. This solves the rounded instance *exactly*, something that would be impossible in general.

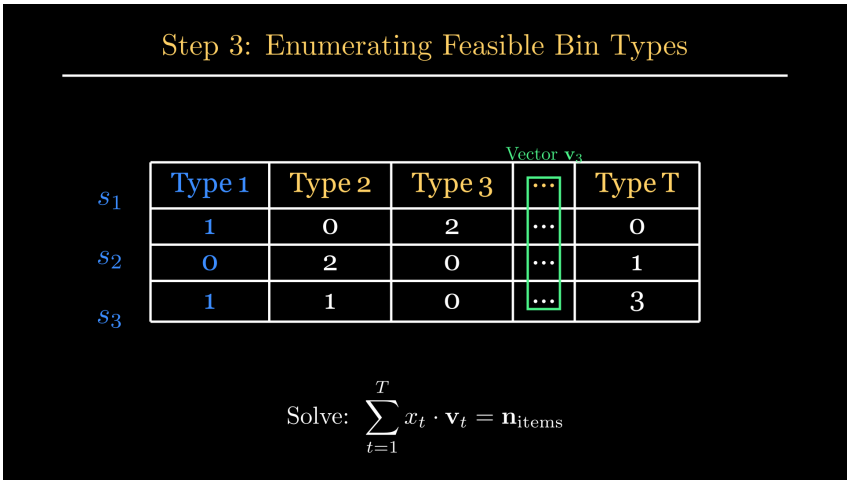

Figure 4: Step 3 APTAS for Bin-Packing

5.2.4 Step 4: Unround and Reinsert Small Items

Motivation. All rounding has made items artificially *larger*, so the exact solution covers a superset of the true instance. We need to map this solution back down to the original items while incurring only a small additive loss.

Procedure. Replace each rounded-up large item in L' by the corresponding original item in L (which only makes bins easier to fill). Then insert small items S greedily. If any bins overflow, create new bins, but the earlier analysis guarantees that at most a constant number of such bins are needed.

Impact. The total number of extra bins introduced by both rounding and small-item reinsertion is at most $1+\varepsilon\,\mathrm{OPT}$, completing the APTAS guarantee. We show this visually as a "compression–release" diagram: bins expand during rounding and then shrink back during unrounding and reinsertion.

Figure 5: Step 4 APTAS for Bin-Packing

6 Approximation Schemes Beyond APTAS

Having fully developed the APTAS machinery in the previous section, we now briefly place it within the larger hierarchy of approximation schemes for Bin Packing. The following notions capture different trade-offs between accuracy and running time, and help situate why the APTAS (rather than a PTAS or FPTAS) is the "correct" achievable target for this problem.

6.1 PTAS (Polynomial-Time Approximation Scheme)

A **PTAS** provides, for every fixed $\varepsilon > 0$, an algorithm A_ε satisfying

$$A_\varepsilon(I) \leq (1+\varepsilon)\,\mathrm{OPT}(I),$$

with running time polynomial in n but potentially *super-polynomial in* $1/\varepsilon$.

For general NP-hard optimization problems this is a strong guarantee: one may get arbitrarily close to optimality, but high precision may be computationally expensive. In Bin Packing specifically, a PTAS would be an exact generalization of the APTAS guarantee, except without the additive $O(1)$ slack.

However, as seen from the baseline hardness results (Section 4.2), removing that additive slack altogether is impossible under standard complexity assumptions.

6.2 Why No FPTAS Exists

A **Fully Polynomial-Time Approximation Scheme (FPTAS)** would require running time polynomial both in n and in $1/\varepsilon$. For Bin Packing this is provably impossible unless P = NP:

Theorem 5 (Garey–Johnson [3]) *Bin Packing does not admit an FPTAS unless* P = NP.

Conceptually, an FPTAS would be "too efficient"—fast enough to resolve the YES/NO gap induced by reductions from PARTITION. Therefore, even though we can approximate arbitrarily well, we *cannot* do so while keeping running time polynomial in $1/\varepsilon$.

6.3 AFPTAS (Asymptotic FPTAS)

An **Asymptotic FPTAS (AFPTAS)** strikes the middle ground. It achieves

$$A_\varepsilon(I) \leq (1+\varepsilon)\,\mathrm{OPT}(I) + O(1),$$

just like an APTAS, but with running time

$$\mathrm{poly}(n, 1/\varepsilon).$$

Thus the AFPTAS improves the computational dependence on ε, while still allowing the additive $O(1)$ term that is unavoidable in the Bin Packing setting. This class of schemes is algorithmically deeper, typically requiring linear grouping, LP-based configuration techniques, or entropy rounding ideas; see Jansen and Klein [8] for a notable construction.

6.4 Hierarchy of Schemes at a Glance

To summarize the relationships:

$$\text{FPTAS} \subsetneq \text{PTAS} \subsetneq \text{APTAS} \quad \text{and} \quad \text{FPTAS} \subsetneq \text{AFPTAS} \approx \text{APTAS}.$$

- A PTAS gives exact $(1+\varepsilon)$ but may take time $n^{O(1/\varepsilon)}$ or worse.
- An FPTAS is impossible for Bin Packing.
- An APTAS gives the best achievable guarantee: $(1+\varepsilon)\mathrm{OPT} + O(1)$.
- An AFPTAS matches this guarantee while being polynomial in $1/\varepsilon$.

This hierarchy reinforces why the APTAS was historically such a milestone for Bin Packing, and why its AFPTAS refinements continue to be an active area of research.

7 Worked Example: APTAS on a Medium-Sized Instance

To illustrate how the APTAS operates in practice, consider the following Bin Packing instance of eight items (sizes in $(0, 1]$):

$$I = \{0.52,\ 0.49,\ 0.48,\ 0.47,\ 0.31,\ 0.30,\ 0.29,\ 0.18\}.$$

7.1 Step 1: Classify Small and Large Items

Fix $\varepsilon = 0.25$ for demonstration. Large items are those $\geq \varepsilon/2 = 0.125$; all items in I qualify, so for this instance $S = \emptyset$ and $L = I$.

Although this is a degenerate case for small-item removal, it makes the remaining steps clearer: all structure comes from grouping and rounding.

7.2 Step 2: Linear Grouping and Rounding

Sort L in decreasing order (already sorted above). Set group size

$$k = \lceil \varepsilon \cdot |L| \rceil = \lceil 0.25 \cdot 8 \rceil = 2.$$

Thus we divide the sequence into groups:

$$G_1 = \{0.52, 0.49\}, \quad G_2 = \{0.48, 0.47\}, \quad G_3 = \{0.31, 0.30\}, \quad G_4 = \{0.29, 0.18\}.$$

We round each group *up* to the largest element in that group:

$$G_1 \to 0.52, \quad G_2 \to 0.48, \quad G_3 \to 0.31, \quad G_4 \to 0.29.$$

This yields the rounded instance

$$L' = \{0.52, 0.52,\ 0.48, 0.48,\ 0.31, 0.31,\ 0.29, 0.29\},$$

with only *four distinct sizes*.

Grouping ensures that the rounding overhead is bounded by at most one rounded item per group, hence by at most $k = 2$ bins overall.

7.3 Step 3: Enumerating Feasible Bin Types

Since all rounded items are at least 0.29, a bin can contain at most three items. The (relevant) possible bin types using the four rounded sizes include, for example:

$$\begin{array}{rll} 0.52 + 0.48 & (= 1.00) & \text{(two-item type)} \\ 0.52 + 0.31 & (= 0.83) & \\ 0.52 + 0.29 & (= 0.81) & \\ 0.48 + 0.31 & (= 0.79) & \\ 0.48 + 0.29 & (= 0.77) & \\ 0.31 + 0.31 + 0.29 & (= 0.91) & \\ 0.31 + 0.29 + 0.29 & (= 0.89) & \end{array}$$

(There are only a few combinations to check because there are only four size classes and at most three items per bin.)

Solving the exact covering problem for L' (via a small ILP or DP) yields an optimal rounded packing:

$$\begin{aligned}&\text{Bin A:} \quad 0.52 + 0.48 = 1.00\\ &\text{Bin B:} \quad 0.52 + 0.48 = 1.00\\ &\text{Bin C:} \quad 0.31 + 0.31 + 0.29 = 0.91\\ &\text{Bin D:} \quad 0.29\end{aligned}$$

Thus $\mathrm{OPT}(L') = 4$ (note: the previous version mistakenly omitted one rounded item and reported 3).

7.4 Step 4: Unrounding and Final Packing

We now replace rounded items with their original items. Because rounding increased sizes, un-rounding can only make packing easier. Map each rounded item back to one original item from the same group:

$$\begin{aligned}&\text{The two rounded 0.52s} \mapsto \{0.52,\, 0.49\}\\ &\text{The two rounded 0.48s} \mapsto \{0.48,\, 0.47\}\\ &\text{The two rounded 0.31s} \mapsto \{0.31,\, 0.30\}\\ &\text{The two rounded 0.29s} \mapsto \{0.29,\, 0.18\}\end{aligned}$$

Applying these replacements to the rounded packing above (one feasible choice of assignment) gives:

$$\begin{aligned}&\text{Bin 1:} \quad 0.52 + 0.48 = 1.00\\ &\text{Bin 2:} \quad 0.49 + 0.47 = 0.96\\ &\text{Bin 3:} \quad 0.31 + 0.30 + 0.29 = 0.90\\ &\text{Bin 4:} \quad 0.18\end{aligned}$$

All bins respect capacity, so the APTAS produces:

$$A_\varepsilon(I) = 4, \qquad \text{and in fact } \mathrm{OPT}(I) = 4.$$

This satisfies the guarantee

$$A_\varepsilon(I) \le (1+\varepsilon)\,\mathrm{OPT}(I) + O(1)$$

with $\varepsilon = 0.25$. (In this instance the algorithm attains the optimum.)

8 Karmarkar-Karp Algorithm Introduction: The 30-Year Benchmark

The one-dimensional Bin Packing Problem (1DBPP) asks for the minimum number of unit-capacity bins to pack n items, each with a size $s_i \in (0, 1]$. While simple offline heuristics like First-Fit Decreasing (FFD) are fast ($O(n \log n)$) and provide good multiplicative guarantees (e.g., $A(I) \le \frac{11}{9}OPT(I) + 4$), the frontier of theoretical research focuses on *additive* guarantees.

For over three decades, the seminal 1982 algorithm by Narendra Karmarkar and Richard M. Karp (KK82) stood as the undisputed benchmark in this area. It was the

first algorithm to break from multiplicative errors and provide a polynomial-time solution with a provably small *additive* error, a monumental breakthrough at the time.

The KK82 algorithm achieves a final packing $A(I)$ using at most:

$$A(I) \leq OPT(I) + O(\log^2 OPT(I)) \tag{1}$$

bins. This $O(\log^2 OPT)$ term—an error that scales polylogarithmically with the *optimal solution size* rather than linearly—established the additive integrality gap for the standard LP relaxation. This result became the "benchmark to beat" until the work of Rothvoß (2013) and Hoberg & Rothvoß (2015), which refined the bound to $O(\log OPT)$.

This analysis deconstructs the core mechanisms of the KK82 algorithm to explain *how* this $O(\log^2 OPT)$ guarantee is achieved.

The LP relaxation is formulated as follows:

$$\begin{array}{lll} \text{minimize} & \sum_{p \in \mathcal{P}} x_p & \text{(Minimize total bins)} \\ \text{subject to} & \sum_{p \in \mathcal{P}} t_{pi} x_p \geq b_i & \text{(for each item type } i = 1..m\text{)} \\ & x_p \geq 0 & \text{(for all patterns } p \in \mathcal{P}\text{)} \end{array}$$

8.1 The Computational Challenge

The LP has two main computational hurdles:

1. **Exponential Variables:** The number of patterns $N = |\mathcal{P}|$ can be exponential in the number of items n.
2. **Many Constraints:** The number of constraints m (distinct item types) can be as large as n.

8.2 The Core Problem

The LP yields an optimal *fractional* solution, OPT_f (also denoted $lin(I)$). The entire challenge is to "round" this fractional solution $\mathbf{x}$ into an integer packing $A(I)$ while introducing only a tiny additive error. The $O(\log^2 OPT)$ term is, precisely, the "price of integrality" paid during this rounding process.

8.3 The KK Mechanism (Part 1): Taming the Constraints via Grouping

The first challenge, having $m = n$ constraints, is handled by a pre-processing step that reduces the number of distinct item types. This is a "main innovation" of the KK82 paper.

8.3.1 Elimination of Small Items

First, a threshold g (e.g., $g = 1/n$) is chosen, and all "small" items $s_i \leq g$ are removed and set aside. This is critical because it ensures the remaining items are "large," meaning any bin can contain at most $1/g$ (e.g., n) of them. These small items are added back at the very end, fitting into existing gaps.

8.3.2 Geometric Grouping

The remaining n' items are "grouped" to reduce the number of distinct types m from $O(n')$ to $O(\text{poly}(\log n'))$. This works in two phases:

1. **Logarithmic Partition:** Items are partitioned into groups I_r based on their size, where $I_r = \{\text{items } i \mid s_i \in (1/2^{r+1}, 1/2^r]\}$.
2. **Linear Grouping:** Within *each* group I_r, items are sorted and further partitioned into k buckets (where k is a parameter). The sizes of all items in a bucket are then rounded *up* to the size of the largest item in that bucket.

This grouping process creates a new, simpler instance K with a manageable (polynomial) number of item types m. The "cost" of this rounding-up adds a small, bounded error in each iteration, which contributes to the final $O(\log^2 OPT)$ term.

8.4 The KK Mechanism (Part 2): Solving the Exponential LP

After grouping, the LP has m (polynomial) constraints, but still N (exponential) variables. KK82 solves this using a now-classic approach based on duality.

8.4.1 The Dual LP

The algorithm considers the *dual* of the Configuration LP. By duality principles, variables and constraints are swapped. The dual has m variables (polynomial) but N constraints (exponential).

8.4.2 The Ellipsoid Method & The Separation Oracle

An LP with a polynomial number of variables and an exponential number of constraints can be solved in polynomial time using the **Ellipsoid Method**, provided one can supply a **Separation Oracle**. The oracle is given a potential solution $\mathbf{y}$ (a vector of m dual variables) and must either certify that $\mathbf{y}$ is feasible (satisfies all N constraints) or return a constraint p that is violated.

8.4.3 The "Aha!" moment: The Oracle is the Knapsack Problem

The dual constraints are of the form $A_p^T \mathbf{y} \leq 1$ for each pattern p. The oracle's job is to find a pattern p that *maximizes* $A_p^T \mathbf{y}$.

If we interpret the dual variables y_i as the **"profit"** or "value" of packing an item of type i, and the item size s_i as its **"weight"**, then $\max_p(A_p^T \mathbf{y})$ is the task of finding the **maximum-profit** set of items that can fit into a single bin (i.e., total weight ≤ 1). This is the exact definition of the **0/1 Knapsack Problem**.

8.4.4 The Full Stack

The Knapsack Problem is NP-hard. However, it admits a Fully Polynomial-Time Approximation Scheme (FPTAS). The Ellipsoid Method is robust enough to work with an *approximate* separation oracle. Therefore, the LP is solved by "stacking" these algorithms: the Ellipsoid Method is called, and at each of its steps, it calls the Knapsack FPTAS to act as its oracle.

8.5 The KK Mechanism (Part 3): Iterative Rounding & The $O(\log^2 OPT)$ Guarantee

Solving the LP gives a fractional solution $\mathbf{x}$. The final step is to round this $\mathbf{x}$ to an integer packing. KK82 does this using an iterative process that runs in a loop for $t = O(\log n)$ iterations.

8.5.1 The Iterative Loop

In each iteration i:

1. **Start with Residual Instance:** Begin with the set of items I_i remaining from the last iteration.
2. **Group:** Apply the Geometric Grouping to I_i, creating instance J_i and a "discard" instance J_i'. The items in J_i' are packed into new bins, creating the first source of additive error, Y_i.
3. **Solve LP:** Solve the Configuration LP for instance J_i to get a fractional solution $\mathbf{x}$.
4. **Partial Rounding:** "Buy" the integer part of the solution. For every pattern p where $x_p \geq 1$, pack $\lfloor x_p \rfloor$ bins with that pattern. These items are permanently removed.
5. **Define New Residual:** The *new* residual instance I_{i+1} consists of all items that were part of the fractional remainders, $\mathbf{x} - \lfloor \mathbf{x} \rfloor$.
6. **Repeat:** The key insight is that the *total size* of the residual problem decreases geometrically at each step, guaranteeing the loop terminates in $O(\log n)$ iterations.

8.6 Source of the $O(\log^2 OPT)$ Guarantee

The final solution cost is the sum of the fractional optimum ($lin(I) \leq OPT(I)$) plus all the bins added during the iterative process.

- **Number of Iterations:** $t = O(\log n)$.
- **Error per Iteration:** In each iteration i, the algorithm adds Y_i bins from packing the "grouped-out" items. This error is bounded by $Y_i \leq O(\log(1/g))$, which is $O(\log n)$.
- **Total Additive Error:**

Since n is bounded by $OPT(I)$ (as $s_i > g$), $\log n$ is $O(\log OPT(I))$. This gives the final $OPT(I) + O(\log^2 OPT(I))$ guarantee.

8.7 The KK82 Legacy

The Karmarkar-Karp algorithm provides a canonical example of the trade-off between complexity and solution quality.

- **FFD (Phase 1):** $O(n \log n)$ time, but a *multiplicative* error ($\frac{11}{9}OPT$).
- **KK82 (Phase 2):** A high-order polynomial time (e.g., $O(n^8)$) but achieves a "near-perfect" *additive* error ($OPT + O(\log^2 OPT)$).

The Karmarkar-Karp algorithm is not a single function but a complex, multi-stage "stack" of advanced techniques. Its $O(\log^2 n)$ bound on the integrality gap, derived from its specific iterative rounding technique, created a theoretical barrier that stood for 30 years until the recent introduction of discrepancy theory methods by Rothvoß.

9 Advanced Bin Packing Algorithms: Rothvoß (2013) to Hoberg & Rothvoß (2015)

9.1 Introduction: The Quest for Additive Guarantees

The one-dimensional Bin Packing Problem (1DBPP) asks for the minimum number of unit-capacity bins required to pack a given set of items with sizes $s_1, \ldots, s_n \in (0, 1]$. Its NP-hardness necessitates approximation algorithms. While simple heuristics like First-Fit Decreasing (FFD) offer good *multiplicative* guarantees ($\approx 1.222 \cdot OPT + \text{const}$), a significant line of theoretical research seeks *additive* guarantees of the form $OPT + C$, where C is a small, slowly growing function, ideally independent of the number of items n.

For over 30 years, the landmark result was the **Karmarkar-Karp (KK82) algorithm**, achieving $OPT_f + O(\log^2 OPT_f)$ bins, where OPT_f is the optimal value of the fractional relaxation. This review focuses on the two papers that finally improved this bound.

9.2 Preliminaries: The Gilmore-Gomory LP Relaxation

Both KK82 and the subsequent breakthroughs leverage the **Gilmore-Gomory Linear Program (LP) relaxation**.

- **Structure:** The LP operates over *patterns*. A pattern p is a multiset of items that fits into a single bin, represented as a vector $p \in \mathbb{Z}^n_{\geq 0}$ where p_i is the count of item i, such that $\sum_{i=1}^n p_i s_i \leq 1$.
- Let $\mathcal{P}$ be the set of all possible valid patterns. The LP formulation is:

$$\min \sum_{p \in \mathcal{P}} x_p \quad \text{subject to} \quad \sum_{p \in \mathcal{P}} p_i x_p \geq b_i \quad \forall i = 1, \ldots, n, \quad x_p \geq 0 \quad \forall p \in \mathcal{P}$$

  Here, b_i is the required number of items of type i, and x_p is a variable representing how many times pattern p is used.

- **Challenge:** The number of possible patterns $|\mathcal{P}|$ can be exponential in n. However, the LP has only n constraints. Efficient algorithms (like Ellipsoid method variants or Column Generation, referenced in KK82 and Rothvoß's papers) can find an approximate solution x with cost $\leq OPT_f + \delta$ in polynomial time (polynomial in n, $\sum b_i$, and $1/\delta$). Often, one works with a basic feasible solution, which has at most n non-zero x_p values.
- **The Core Problem:** The LP yields a fractional optimum OPT_f. The challenge is to round the fractional vector x (with potentially millions of components x_p) into an integer vector y (representing an actual packing) such that the cost $\sum y_p$ is very close to $\sum x_p$, specifically $\sum y_p \leq \sum x_p + C$.

9.3 The 2013 Breakthrough: Rothvoß

Rothvoß's 2013 paper, "Approximating Bin Packing within $O(\log OPT \cdot \log \log OPT)$ bins," was the first improvement over KK82. It introduced discrepancy theory as a new rounding tool.

9.4 The New Tool: Discrepancy Theory via Lovett-Meka (LM12)

Instead of KK82's rounding, Rothvoß employed the **Constructive Partial Coloring Lemma** (Lovett & Meka, 2012).

- **Intuition:** Discrepancy theory aims to find a coloring (e.g., ± 1) for elements of a ground set such that for any given subset, the sum of colors is small (the set is "balanced"). The LM12 algorithm provides a constructive, randomized way to achieve this, generalized to rounding fractional vectors.
- **The LM12 Guarantee (Lemma 1 in Rothvoß 2013):** Given a starting fractional point $x \in [0,1]^m$ and constraints defined by vectors $v_1, \ldots, v_n$, the algorithm finds a new point $y \in [0,1]^m$ such that:
  1. **Near-Integrality:** At least half of the coordinates y_j are very close to 0 or 1 ($y_j \in [0, \delta] \cup [1-\delta, 1]$).
  2. **Low Error (Discrepancy):** The change in each constraint value is bounded: $|v_i y - v_i x| \leq \lambda_i ||v_i||_2$.

The Entropy Condition: The correctness of the rounding depends on choosing error parameters λ_i that satisfy the "entropy condition" derived from Lovett-Meka. This condition ensures that the randomized walk (Brownian motion) does not hit the error boundaries too frequently before the variables become integral. Mathematically, for a system with m variables, the parameters must satisfy:

$$\sum_{i=1}^{n} \exp(-\lambda_i^2/16) \leq \frac{m}{16} \tag{2}$$

Rothvoß (2013) applies this lemma iteratively. In each of the $O(\log m)$ iterations, half of the remaining fractional variables are rounded to 0 or 1, while the cumulative error on each constraint i is carefully controlled by the $\lambda_i \|A_i\|_2$ term.
Here is the formal statement of the lemma:

Lemma 1 (Constructive Partial Coloring, Lovett-Meka 2012) *Let $A \in \mathbb{R}^{n \times m}$ be a matrix, $x \in [0,1]^m$ be a fractional solution, and $\lambda_1, \ldots, \lambda_n > 0$ be parameters satisfying $\sum_{i=1}^{n} e^{-\lambda_i^2/16} \leq \frac{m}{16}$.*

There exists a polynomial-time randomized algorithm that finds a point $y \in [0,1]^m$ such that:

1. ***(Low Discrepancy):** The error on each constraint is bounded by the L_2-norm of the corresponding row:*

$$|(Ay)_i - (Ax)_i| \leq \lambda_i \|A_i\|_2 \quad \forall i = 1, \ldots, n$$

2. ***(Near-Integrality):** For a given $\delta > 0$, at least half of the coordinates are nearly integral. For $m' \geq (1-\delta)m$ coordinates $j \in [m]$, we have:*

$$y_j \in [0, \delta] \cup [1-\delta, 1]$$

9.5 The Core Challenge: "Spiky" Patterns and the L_2-Norm

The effectiveness of LM12 hinges on the error bound $|v_i y - v_i x| \leq \lambda_i ||v_i||_2$. Rothvoß's insight was that this bound is problematic if $||v_i||_2$ is large.

- In the bin packing context, the vectors v_i correspond to (sums of) rows of the pattern matrix A. The i-th row A_i lists how many times item i appears in each pattern.
- A large $||A_i||_2 = \sqrt{\sum_p (A_{ip})^2}$ norm occurs if there are patterns p with large entries A_{ip} — i.e., patterns containing many copies of item i.
- This is the **"spiky pattern"** problem: a pattern p heavily utilizing a single item type i (large A_{ip}) causes a large L_2-norm for row i, leading to a potentially large rounding error for that item's constraint. This was particularly problematic for very small items where A_{ip} could be large.

9.6 The Novel Technique: "Gluing"

To mitigate the large L_2-norms caused by spiky patterns, Rothvoß introduced **"gluing"** as a pre-processing step (detailed in **Section 5.2 of the 2013 paper**) before rounding.

- **Trigger:** Gluing is applied when a pattern p (with fractional value $x_p = r/q$) uses many copies (p_i) of a *small* item i, such that the total size $p_i s_i$ exceeds a threshold (e.g., related to $1/\text{polylog}(n)$). Specifically, if $p_i \geq w \cdot q$ for carefully chosen w, q.
- **Mechanism (Figure 1b):** It takes $w \cdot q$ copies of item i within pattern p and conceptually "glues" them into q copies of a *new, artificial, larger item* i' with size $s_{i'} = w \cdot s_i$. The pattern p is modified to use q copies of i' instead. The fractional value x_p remains r/q.
- **Purpose:** This transformation replaces a large entry A_{ip} (for the small item i) with smaller entries (q) for the new, larger item i'. This "smooths" the matrix rows associated with small items, reducing their L_2-norms and making the LM12 rounding effective.

- **Limitation:** As noted by Hoberg & Rothvoß (2015), this gluing procedure in the 2013 paper was complex and primarily effective only for items below a size threshold of $1/\text{polylog}(n)$.

9.7 The Full Algorithm (Section 6) and Result

The Rothvoß (2013) algorithm iterates $O(\log n)$ times. Each iteration involves the following steps, justified by the source's corresponding proofs:

1. **Discretize:** Utilizing **Lemma 10 of Rothvoß (2013)**, the algorithm ensures x_p values are multiples of some $1/q = 1/\text{polylog}(n)$.
2. **Group & Glue:** By applying the "gluing" technique established in **Lemma 8 (Rothvoß, 2013)**, the instance is made "well-spread" for small items (i.e., ensuring A_{ip} is small relative to the total count $A_i x$ for small i).
3. **Round:** As detailed in **Theorem 9 (Rothvoß, 2013)**, the algorithm applies the LM12 procedure to the modified matrix/solution, rounding half the remaining fractional variables with controlled error. The error analysis in Theorem 9 carefully bounds the discrepancy based on the properties achieved by gluing.

The complexity arose because the "Group & Glue" step introduced an error term related to the parameters used (specifically, involving $\log(1/\delta)$ where δ related to $1/\text{polylog}(n)$), contributing an $O(\log\log OPT)$ factor to the error accumulated in each of the $O(\log OPT)$ iterations. The final guarantee: $OPT_f + O(\log OPT \cdot \log\log OPT)$. The runtime is polynomial, dominated by solving the LP and the LM12 calls.

10 The 2015 Refinement: Hoberg & Rothvoß

The 2015 Hoberg & Rothvoß paper, "A Logarithmic Additive Integrality Gap for Bin Packing," achieved the tighter $O(\log OPT)$ bound by introducing a fundamentally cleaner structure.

10.1 The Core Idea: 2-Stage Packing (Section 2)

Instead of fixing the spiky pattern problem after the fact, they reformulated the problem to avoid it structurally.

1. **Stage 1: Items $\to$ Containers.** Define a **container** C as any valid multiset of original items that fits in a bin ($C \in \mathbb{Z}^n_{\geq 0}$ with $\sum s_i C_i \leq 1$). Let $s(C)$ be its total size.
2. **Stage 2: Containers $\to$ Bins.** Define a **pattern** p now as a multiset of *containers* that fits in a bin ($p \in \mathbb{Z}^{\mathcal{C}}_{\geq 0}$ where $\mathcal{C}$ is the set of all containers, such that $\sum_{C \in \mathcal{C}} p_C s(C) \leq 1$).
3. **Intermediate Variables:** They introduce integer variables y_C representing the number of times container C is "created" or used.
4. **Packing Graphs (Figure 1):** The process is modeled with two bipartite graphs:

- $G_1(b, y)$: Matches original items (demand b_i) to available slots within the chosen containers (supply $y_C \cdot C_i$).
- $G_2(x, y)$: Matches the chosen containers (demand y_C) to slots within the final fractional patterns (supply $x_p \cdot p_C$).

5. **Deficiency:** The objective is implicitly tied to minimizing a "deficiency" metric. Formally, for a packing graph $G = (V_l \cup V_r, E)$, the deficiency is defined as the minimum weight of left-nodes that cannot be covered by any valid assignment a:

$$\text{def}(G) := \min_a \sum_{v \in V_l} s(v) \cdot (\text{mult}(v) - a(\delta(v))) \tag{3}$$

The algorithm's goal is to ensure that rounding fractional values in x increases this deficiency by only a constant amount $O(1)$.

This 2-stage packing concept is formalized into a new, structured LP relaxation. Let $\mathcal{C}$ be the set of all valid "containers" (multisets of items that fit in a bin), and let $\mathcal{P}_\mathcal{C}$ be the set of all valid "patterns" of containers (multisets of containers that fit in a bin).

The Hoberg & Rothvoß (2015) algorithm finds a fractional solution (x, y) to the following system:

The 2-Stage Packing Formulation

$$\begin{aligned}
\min \quad & \sum_{p \in \mathcal{P}_\mathcal{C}} x_p && && (4)\\
\text{s.t.} \quad & \sum_{C \in \mathcal{C}} y_C \cdot C_i \geq b_i && \forall \text{ items } i && (5)\\
& \sum_{p \in \mathcal{P}_\mathcal{C}} x_p \cdot p_C \geq y_C && \forall \text{ containers } C \in \mathcal{C} && (6)\\
& x_p \geq 0, \quad y_C \geq 0 && \forall p \in \mathcal{P}_\mathcal{C}, C \in \mathcal{C} &&
\end{aligned}$$

Here, y_C represents the (fractional) number of times container C is "created", and x_p is the number of times pattern p is used.

- Equation (5) ensures that all original items b_i are packed into containers (corresponds to graph G_1).
- Equation (6) ensures that all created containers y_C are packed into the final bins (corresponds to graph G_2).

The key insight is that the rounding algorithm is only applied to Equation (6), which packs containers. As noted, a pattern p simply cannot contain an arbitrarily large number of copies (p_C) of the same container C. This structurally avoids the "spiky" L_2-norm problem, as the matrix for (6) is inherently "smooth".

10.2 The Final Algorithm and Bound

The Hoberg & Rothvoß (2015) algorithm also iterates $O(\log n)$ times:

1. **Rebuild Containers:** Detailed in **Section 3 of Hoberg & Rothvoß (2015)**, this step replaces the complex "gluing". It involves sophisticated grouping (**Lemma 10** of the source) and reassigning/combining containers (**Lemma 13** of the source) within the existing fractional patterns x.

   The Shadow Matrix: To bound the rounding error, the authors construct a **Shadow Incidence Matrix** $\tilde{A}$. Unlike the standard matrix A, $\tilde{A}$ retains the history of grouped containers, ensuring that for any container class σ, the L_1 norm of the rows remains sufficiently large ($||\tilde{A}_C||_1 \geq \sigma^{-1/2}$). This structural property allows the Lovett-Meka algorithm to round variables with very low discrepancy, specifically bounding the L_2 norm of the error vectors.

2. **Round:** As described in **Section 4**, the algorithm applies the *same* LM12 algorithm (specifically **Claim 14**) to the fractional pattern vector x, using constraints derived from the (rebuilt) container incidence matrix A.

10.3 Why it's Cleaner and Better

The 2-stage structure provides several advantages leading to the cleaner analysis and tighter bound:

- **Inherent Smoothness:** The objects being packed into the final patterns are now "containers." Since containers must have size $s(C) \leq 1$, a pattern p simply cannot contain an arbitrarily large number of copies (p_C) of the same container C. The large entries in the matrix rows that plagued the 2013 analysis for tiny items are structurally avoided when dealing with containers.

- **Simplified Pre-processing:** The "Rebuilding Containers" step (Section 3 of the source) achieves the necessary "well-spread" properties more elegantly and robustly than the 2013 "gluing," which had limitations on item size. The 2015 paper notes their procedure works even for items/containers up to size $\Omega(1)$.

- **Error Reduction ($O(1)$ Deficiency):** Because the matrix A (representing container patterns) is inherently better behaved, the LM12 rounding step (Section 4) incurs only a constant $O(1)$ increase in total deficiency per iteration, as proven in **Lemma 17** of the source paper. The complex error term involving $\log \log OPT$ from the 2013 analysis disappears.

- **Full Spectrum Parameters:** The 2015 paper notes they use the "full spectrum" of error parameters λ_I in LM12, whereas the 2013 paper used only two types, contributing to the cleaner $O(1)$ error per iteration.

The final result: The total additive gap is $O(\log OPT)$ iterations $\times$ $O(1)$ error per iteration, yielding the tight $\mathbf{OPT_f + O(\log OPT)}$ bound.

10.4 Summarising Rothvoß & Hoberg

The progression from Rothvoß (2013) to Hoberg & Rothvoß (2015) exemplifies refinement in theoretical algorithm design. Both papers leverage the Gilmore-Gomory LP and the powerful Lovett-Meka discrepancy rounding algorithm. However, the 2013 paper required a complex, somewhat limited "gluing" mechanism to force the problem structure into shape for the rounding tool, resulting in an $O(\log OPT \cdot \log\log OPT)$ gap. The 2015 paper achieved the final $O(\log OPT)$ gap through a more fundamental insight: redefining the problem via a 2-stage packing (Items $\rightarrow$ Containers $\rightarrow$ Bins). This elegant structural change inherently created the "smoothness" needed for discrepancy rounding, leading to a cleaner algorithm, a simpler analysis, and a tighter, likely optimal, additive bound. Visualizing the contrast between "gluing" and "2-stage packing" will be key to understanding this significant theoretical advancement.

Formally, the two breakthrough results are:

Theorem 6 (Rothvoß 2013) *There exists a polynomial-time algorithm that, given a bin packing instance I, finds a packing using at most*

$$OPT_f(I) + O(\log(OPT_f(I)) \cdot \log\log(OPT_f(I)))$$

bins, where $OPT_f(I)$ is the optimal value of the Gilmore-Gomory LP relaxation.

Theorem 7 (Hoberg & Rothvoß 2015) *There exists a polynomial-time algorithm that, given a bin packing instance I, finds a packing using at most*

$$OPT_f(I) + O(\log(OPT_f(I)))$$

bins.

11 Hybrid Genetic Grouping Algorithm

11.1 Fundamental Axioms & Assumptions

This derivation rests on three core hypotheses accepted as the foundation for the algorithm's design:

1. **The Building Block Hypothesis (Holland):** A Genetic Algorithm optimizes by identifying, preserving, and recombining short, low-order, high-fitness partial solutions called "schemata" or "building blocks".

2. **The Grouping Structure Assumption:** In the Bin Packing Problem (BPP), the cost function depends solely on the composition of the groups (bins). Therefore, the only meaningful building block is a specific, well-filled group (bin), regardless of its position in the chromosome.

3. **The Dominance Conjecture:** A bin that is locally optimal (maximally filled) dominates other possible bins formed from the same items. We assume that constructing a global solution requires the accumulation of these "Dominant Bins".

11.2 Definitions

To formalize the proof, we define the following variables:

- H **(The Schema):** A *Dominant Bin*. Defined as a specific subset of items that fills a bin capacity C efficiently (representing a "group-schema of order one").
- $m(H,t)$**:** The number of individuals in the population at generation t containing bin H.
- $f(H)$**:** The fitness contribution of bin H, defined by the algorithm's cost function as $(F_H/C)^k$ where $k = 2$.
- $\bar{f}$**:** The average fitness of the entire population.
- $P_{disruption}$**:** The probability that the reproductive operators (Crossover/Mutation) destroy the composition of bin H during transmission.

11.3 The Derivation

Theorem: The HGGA satisfies the conditions required for the expected count of Dominant Bins, $E[m(H,t+1)]$, to grow exponentially over successive generations.

11.3.1 Step 1: Guaranteed Generation (The Existence Proof)

Standard GAs rely on random chance to generate good schemata. HGGA ensures $m(H,t) > 0$ through hybridization.

- **Mechanism:** The algorithm applies a local optimization heuristic (inspired by the Dominance Criterion) that iteratively swaps items to maximize bin fill before evaluation.
- **Implication:** The algorithm acts as a localized search engine that manufactures Dominant Bins (H), ensuring valid building blocks exist in the gene pool.

11.3.2 Step 2: Selection Pressure (The Growth Proof)

We examine the Selection Ratio $\frac{f(H)}{\bar{f}}$.

- **Mechanism:** The cost function raises the bin fill ratio to the power of $k = 2$.
- **Derivation:**

$$\text{If } Fill_H \approx C \implies f(H) \approx 1.0$$

$$\text{If } Fill_{avg} \ll C \implies f(avg) \ll 1.0$$

  Consequently, the ratio $\frac{f(H)}{\bar{f}}$ is significantly greater than 1.
- **Implication:** Individuals containing H are selected as parents with high probability via Tournament Selection.

11.3.3 Step 3: Zero-Disruption Transmission (The Survival Proof)

We examine the survival probability $(1 - P_{disruption})$.

- **Mechanism:** The HGGA alters the encoding such that **One Gene = One Bin**. The Crossover operator (BPRX) transmits randomly selected genes (whole bins) from parent to child.
- **Derivation:** Since H is treated as an atomic unit, the crossover point cannot fall "inside" the bin definitions.

$$P_{disruption}(H) \approx 0 \implies (1 - P_{disruption}) \approx 1$$

- **Implication:** Once selected, the building block H is transmitted intact to the next generation.

11.4 Conclusion for HGGA

Combining these steps into the Schema Theorem inequality:

$$E[m(H, t+1)] \geq \underbrace{m(H,t)}_{\substack{\text{Generated by} \\ \text{Step 1}}} \cdot \underbrace{\left[\frac{f(H)}{\bar{f}}\right]}_{\substack{\text{Step 2:} \\ \gg 1}} \cdot \underbrace{(1 - P_{disruption})}_{\substack{\text{Step 3:} \\ \approx 1}}$$

Since the Growth Factor is strictly greater than 1 and the Disruption probability is negligible, the expected number of Dominant Bins grows geometrically. The population will progressively converge toward a state composed entirely of Dominant Bins—which constitutes the global optimum.

Q.E.D.

12 The Martello-Toth Procedure (MTP)

12.1 Problem Definition

Given a set of n items with integer sizes $I = \{s_1, s_2, \ldots, s_n\}$ and a fixed bin capacity C, the objective is to partition I into the minimum number of subsets (bins) $B_1, \ldots, B_m$ such that the sum of sizes in each bin does not exceed C.

We assume without loss of generality that items are sorted in non-increasing order of size:

$$s_1 \geq s_2 \geq \cdots \geq s_n \tag{7}$$

13 Mathematical Lower Bounds

The efficiency of the MTP depends on the tightness of its lower bounds. A lower bound $L(I)$ allows the algorithm to prune branches of the search tree where the current partial solution plus the lower bound of the remaining items exceeds the best known solution.

13.1 The L_1 Bound (Volume Bound)

The simplest lower bound is derived from the continuous relaxation of the problem (assuming items can be split like liquid).

$$L_1 = \left\lceil \frac{\sum_{i=1}^{n} s_i}{C} \right\rceil \tag{8}$$

It is proven that the worst-case performance ratio of L_1 is 1/2. While fast ($O(n)$), it is often loose for instances where items are large relative to C.

13.2 The L_2 Bound (Martello-Toth Bound)

The core mathematical contribution of the MTP is the L_2 bound, which improves upon L_1 by analyzing "wasted" space that is mathematically unavoidable.

13.2.1 Definitions

For any integer parameter K such that $0 \leq K \leq C/2$, we classify the items into three sets:

$$\begin{aligned} N_1(K) &= \{i \in I : s_i > C - K\} && \text{(Large items)} \\ N_2(K) &= \{i \in I : C - K \geq s_i > C/2\} && \text{(Medium items)} \\ N_3(K) &= \{i \in I : C/2 \geq s_i \geq K\} && \text{(Small items)} \end{aligned}$$

13.2.2 The $L(K)$ Function

Based on these sets, we define a function $L(K)$:

$$L(K) = |N_1| + |N_2| + \max\left(0, \left\lceil \frac{\sum_{i \in N_3} s_i - R_{N_2}}{C} \right\rceil\right) \tag{9}$$

where R_{N_2} is the total residual capacity left in the bins containing items from N_2:

$$R_{N_2} = |N_2|C - \sum_{i \in N_2} s_i$$

Theorem 8 (Correctness of $L(K)$) *For any $K \in [0, C/2]$, $L(K)$ is a valid lower bound on the optimal number of bins m.*

Proof 1 *Consider the packing requirements of the sets N_1, N_2, and N_3:*

1. ***Separation of Large Items:*** *Every item in N_1 has size $s_i > C - K$. Every item in N_2 has size $s_i > C/2$. Since $K \leq C/2$, all items in $N_1 \cup N_2$ have size strictly greater than $C/2$. Therefore, no two items from $N_1 \cup N_2$ can fit into the same bin. This implies that at least $|N_1| + |N_2|$ bins are required just to hold these items.*

2. ***Incompatibility of N_1 and N_3:*** *An item from N_3 has size $s_i \geq K$. An item from N_1 has size $s_j > C - K$. The sum $s_i + s_j > C$. Thus, no item from N_3 can be placed in a bin containing an item from N_1.*

3. ***Filling the Gaps:*** *The items in N_3 must be packed either into the remaining space of bins containing N_2 items, or into completely new bins. The total available space in the bins utilized by N_2 items is exactly $R_{N_2} = |N_2|C - \sum_{i \in N_2} s_i$.*

4. ***Calculation:*** *The total size of items in N_3 is $\sum_{i \in N_3} s_i$. The portion of this total size that cannot fit into the N_2 bins is:*

$$Excess = \max\left(0, \sum_{i \in N_3} s_i - R_{N_2}\right)$$

This excess volume must go into new bins. The minimum number of additional bins required for this excess is $\lceil Excess/C \rceil$.

5. ***Conclusion:*** *The total bins required is the sum of bins for $N_1 \cup N_2$ plus the additional bins for the overflow of N_3.*

$$m \geq (|N_1| + |N_2|) + \left\lceil \frac{\max(0, \sum_{N_3} s_i - R_{N_2})}{C} \right\rceil$$

This concludes the proof.

13.2.3 The Optimized L_2 Bound

The bound L_2 is defined as the maximum value of $L(K)$ over all feasible K:

$$L_2 = \max_{0 \leq K \leq C/2} L(K) \tag{10}$$

To compute this efficiently, it is sufficient to check K only for distinct values of $s_i \leq C/2$. If items are sorted, this can be computed in $O(n)$ time.

13.3 Reduction Procedures

Before and during the branching process, MTP applies reduction procedures to fix bins permanently, reducing the problem size. This relies on the concept of **Dominance**.

Definition 2 (Feasible Set Dominance) *A feasible set F_1 dominates a feasible set F_2 if the optimal solution obtained by setting a bin $B = F_1$ is not worse than the optimal solution obtained by setting $B = F_2$.*

13.4 Dominance Criterion

A sufficient condition for dominance is: If F_1 and F_2 are distinct feasible sets, and there exists a partition $P = \{P_1, \ldots, P_\ell\}$ of F_2 and a subset $\{i_1, \ldots, i_\ell\} \subseteq F_1$ such that:

$$s_{i_h} \geq \sum_{k \in P_h} s_k \quad \text{for } h = 1, \ldots, \ell \tag{11}$$

then F_1 dominates F_2.

13.5 Reduction Algorithm

The MTP reduction algorithm (Procedure REDUCTION) iteratively looks for dominating sets to fix into bins.

1. **Initialization:** Initialize fixed bins count $j \leftarrow 0$. Define I as the set of unpacked items.
2. **Loop Start:** Begin repetition until the condition in step 5 is met.
3. **Selection:** Let i be the largest remaining item in I.
4. **Find Optimal Set F:**
   - Find the feasible set $F \subseteq I$ containing i that packs the bin most effectively (dominates other sets containing i).
   - **Check 1 (Single Item):** If i cannot fit with any other item, set $F = \{i\}$.
   - **Check 2 (Pairs):** If i fits with i' and $\{i, i'\}$ fills the bin optimally (e.g., $s_i + s_{i'} = C$), set $F = \{i, i'\}$.
5. **Bin Fixation:** If the set F is not empty:
   - Increment bin count $j \leftarrow j + 1$.
   - Fix Bin $B_j := F$.
   - Remove items in F from I.
6. **Loop End:** Repeat from step 2 until no further reductions are possible in I.

This procedure essentially greedily matches items if they form a "perfect" or "dominant" bin, reducing the complexity for the subsequent Branch-and-Bound phase.

13.6 The Exact Branch-and-Bound Algorithm

The complete Martello-Toth Procedure combines the bounds and reductions into a Depth-First Search (DFS).

13.7 Algorithm Steps

1. **Initialization:**
   - Sort items $s_1 \geq s_2 \cdots \geq s_n$.
   - Calculate global lower bound $LB = L_2$.
   - Calculate heuristic upper bound UB (using First-Fit Decreasing or Best-Fit Decreasing).
   - If $LB == UB$, the heuristic solution is optimal. STOP.
2. **Reduction:**
   - Apply Procedure REDUCTION to fix easy bins. Update problem size and LB.

3. **Branching (Backtracking):**
   - MTP builds a solution bin by bin.
   - For the current bin, it attempts to place the largest available item.
   - It then recursively attempts to fill the remaining capacity of the current bin with the next largest fitting items.
   - Once a bin is closed, it moves to the next bin.
4. **Pruning (The Critical Step):** At any node in the search tree:
   - Let $m_{current}$ be the number of bins already fixed/filled.
   - Let I_{rem} be the set of remaining unpacked items.
   - Calculate the lower bound for the remainder: $L_2(I_{rem})$.
   - **Condition:** If $m_{current} + L_2(I_{rem}) \geq UB$, then this branch cannot lead to a better solution than what we already found. **PRUNE (Backtrack)**.
5. **Updating Best Solution:** If a valid packing is found with z bins and $z < UB$:
   - Update $UB = z$.
   - If $UB == LB$, STOP (Optimal found).

13.8 Complexity and Optimality

The MTP is an exact algorithm. While the worst-case time complexity is exponential (due to the NP-hardness of BPP), the effective use of the L_2 bound allows it to solve many instances efficiently. The L_2 bound has an asymptotic worst-case performance ratio of 2/3, meaning it is tighter than simple volume bounds.

14 Comparative Discussion

The table below summarizes several standard algorithm families for one-dimensional Bin Packing. We expand on the meaning of their approximation guarantees, additive gaps, and running times.

Table 1: Comparison of Bin Packing Algorithms

Algorithm	Approx. Guarantee	Additive Gap	Runtime
FF / BF	Constant	$O(\text{OPT})$	$O(n \log n)$
FFD / BFD	$\frac{11}{9} + \frac{6}{9}$	$+1$	$O(n \log n)$
Harmonic-K	≈ 1.691	$O(\text{OPT})$	Fast
APTAS	$(1+\varepsilon)$	$O(1)$	$n^{O(1/\varepsilon)}$
AFPTAS	$(1+\varepsilon)$	$O(1)$	$\text{Poly}(n, 1/\varepsilon)$
LP-based (Hoberg–Rothvoß)	$1+\varepsilon$	$O(\log \text{OPT})$	Slow

14.1 Discussion of Algorithm Families

FF / BF (First Fit / Best Fit). These simple greedy heuristics run in $O(n \log n)$ time and achieve constant-factor approximation guarantees. Their asymptotic ratio approaches 1.7, and the additive gap can be as large as $O(\mathrm{OPT})$ on adversarial inputs. Despite weaker bounds, they are extremely fast and broadly used in practice.[4]

FFD / BFD (First Fit Decreasing / Best Fit Decreasing). Sorting items in nonincreasing order greatly improves performance. The classical bound for FFD is

$$\mathrm{FFD}(I) \leq \frac{11}{9}\,\mathrm{OPT}(I) + \frac{6}{9},$$

yielding an asymptotic ratio of 11/9 and an additive error bounded by a universal constant. These algorithms maintain $O(n \log n)$ runtime and serve as strong practical baselines.[10]

Harmonic-K. Harmonic algorithms partition item sizes into classes and pack each class separately. The limiting asymptotic ratio of the classical harmonic family is $T_\infty \approx 1.691$. Larger K improves the guarantee but increases running time. These algorithms perform well on structured or heavy-tailed input distributions.[11]

APTAS (Asymptotic PTAS). The scheme of Fernández de la Vega and Lueker produces, for any $\varepsilon > 0$, a packing with ratio $(1 + \varepsilon)$ and additive error $O(1)$. The runtime typically has the form $n^{O(1/\varepsilon)}$, which becomes large for very small ε, limiting practical use.[6]

AFPTAS (Asymptotic Fully PTAS). AFPTAS methods refine the APTAS by ensuring runtime polynomial in both n and $1/\varepsilon$. These approaches employ grouping, rounding, and configuration-LP machinery. They remove the exponential dependence on $1/\varepsilon$, substantially increasing practical relevance.[12]

LP-based Approaches (Hoberg–Rothvoß). Advanced LP-based methods leveraging discrepancy theory guarantee packings within $\mathrm{OPT} + O(\log \mathrm{OPT})$ bins, giving near-optimal additive performance. These techniques are extremely powerful but require solving large linear programs and are slower than combinatorial heuristics.[13]

15 The One-Cut Linear Programming Approach for the Cutting Stock Problem

15.1 Problem Definition

The classical Gilmore-Gomory approach to the One-Dimensional Cutting Stock Problem relies on column generation to handle an exponential number of cutting patterns. Dyckhoff (1981) proposed an alternative formulation, "Model II", based on a recursive "One-Cut" principle. This document details the mathematical formulation of Model II, which uses a polynomial number of variables and constraints for many practical instances, allowing it to be solved using standard Linear Programming software without column generation.

Relationship to the 1D Bin Packing Problem (1DBPP). The 1DBPP is closely related to the 1DCSP: both problems involve packing items of given lengths into containers of fixed capacity, and both seek to minimize the number of containers (bins or stock lengths) used. In fact, the 1DBPP can be viewed as a special case of the 1DCSP in which each pattern contains *at most one* copy of each item size, whereas the cutting-stock formulation allows multiple copies of the same item type per pattern. Therefore, the modelling ideas behind Model II recursive subdivision, capacity-based decomposition, and pattern generation through "one cut at a time" extend naturally to Bin Packing as well. This connection is why techniques developed for the cutting-stock literature, including Gilmore–Gomory and Dyckhoff's Model II, often carry over directly to efficient formulations or relaxations of the 1DBPP.

We consider the **Standard Problem** of one-dimensional cutting stock optimization. Given:

- A set of standard stock lengths $S = \{s_1, \ldots, s_K\}$.
- A set of required order lengths (demand) $D = \{d_1, \ldots, d_I\}$.
- Demand requirements N_l for each order length $l \in D$.
- Costs c_l associated with consuming a standard length $l \in S$.

The objective is to satisfy all demands N_l while minimizing the total cost of stock used.

15.2 The One-Cut Concept

Unlike the classical approach (Model I), which defines a variable for every possible complex cutting pattern (e.g., "one bin contains 2 items of size A and 3 of size B"), Model II is based on a recursive cutting process.

Assumption 1 (The One-Cut Principle) *The cutting process is modeled as an unlimited sequence of cutting operations. In each operation, a piece of length k is divided into exactly two new pieces:*

1. *A section of an order length $l \in D$ (where $l < k$).*
2. *A residual section of length $k - l$.*

This simple structure $[k; l]$ allows complex patterns to be built successively. For example, cutting a length of 9 into $\{4, 2, 2, 1\}$ is modeled as:

$$[9; 4] \rightarrow \text{Residue } 5 \rightarrow [5; 2] \rightarrow \text{Residue } 3 \rightarrow [3; 2] \rightarrow \text{Residue } 1$$

15.3 Mathematical Formulation (Model II)

15.4 Sets and Parameters

Let R be the set of all relevant residual lengths (lengths that can be produced by cutting order lengths from stock lengths). The model considers all lengths in the set $L = S \cup D \cup R$.

15.5 Decision Variables

The fundamental decision variables represent the number of times a specific "one-cut" is performed:

$$y_{k,l} \geq 0 \quad \text{for } k \in S \cup R,\ l \in D,\ l < k \tag{12}$$

$y_{k,l}$ represents the number of pieces of length k that are cut to produce one item of order length l and a remainder of $k - l$.

15.6 Constraints

The model relies on **flow conservation (balance) constraints** for every length l that is not a standard stock length (i.e., for all $l \in (D \cup R) \setminus S$).

The logic is: *Total Input of length l $\geq$ Total Output of length l.*

$$\underbrace{\sum_{k \in A_l} y_{k,l}}_{\text{Production from larger cuts}} + \underbrace{\sum_{k \in B_l} y_{k+l,k}}_{\text{Production as residue}} \geq \underbrace{\sum_{k \in C_l} y_{l,k}}_{\text{Consumption for smaller cuts}} + \underbrace{N_l}_{\text{Final Demand}} \tag{13}$$

Where the sets are defined as:

- $A_l = \{k \in S \cup R \mid k > l\}$: Lengths k that can be cut to produce l as the primary order piece.
- $B_l = \{k \in D \mid k + l \in S \cup R\}$: Lengths $k + l$ that, when cut into order size k, leave l as the residue.
- $C_l = \{k \in D \mid k < l\}$: Order lengths k that can be cut *from* length l.

15.7 Objective Function

The objective is to minimize the net cost of standard lengths consumed.

$$\text{Minimize } Z = \sum_{l \in S} c_l \left(\sum_{k \in C_l} y_{l,k} - \sum_{k \in B_l} y_{k+l,k} \right) \tag{14}$$

This calculates the net consumption of standard length l as: (Total pieces of l cut) minus (Total pieces of l produced as residue from larger stock).

Feature	Model I (Gilmore-Gomory)	Model II (Dyckhoff)
Variables	Cutting Patterns (Exponential)	One-Cuts (Polynomial)
Constraints	$\|D\|$ (Number of order lengths)	$\|D\| + \|R\|$ (Orders + Residues)
Solution Method	Column Generation	Standard Simplex
Structure	Static Patterns	Dynamic Flow

Table 2: Comparison of Approaches

15.8 Comparison with Classical Model (Model I)

15.9 Model Size Analysis

Model II generally has more constraints than Model I but drastically fewer variables.

- **Model I:** Constraints $\approx I$. Variables $\approx$ Millions.
- **Model II:** Constraints $\approx S_{max}$ (max standard length). Variables $\approx I \cdot (K + S_{max})$.

For problems with many stock lengths or a high ratio of demand sizes to stock size ($|D|/S_{max} \approx 1$), Model II can be significantly more efficient and easier to implement.

15.10 Conclusion

Dyckhoff's Model II offers a distinct advantage for cutting stock problems where the variety of stock lengths is high or where "trim loss" has value (reusable residue). By treating the cutting process as a flow of materials through "one-cut" transformations, it avoids the complexity of generating all combinatorial patterns, providing an exact solution using standard LP solvers.

16 Limitations and Future Scope

The current study, while comprehensive within the domain of the One-Dimensional Bin Packing Problem (1DBPP), is subject to specific constraints. We identify the following limitations and propose corresponding avenues for future research.

1. **Dimensionality Constraints**
    - **Limitation:** This project is strictly bound to the One-Dimensional Bin Packing Problem (1DBPP). While this models simpler resource allocation (e.g., time scheduling or single-resource memory), it fails to capture the geometric complexities of physical logistics.
    - **Future Scope:** Extending the comparative analysis to 2D or 3D Bin Packing (e.g., packing pallets or shipping containers), where constraints like rotation, item shape, and structural stability become critical factors.
2. **Static Nature of Data (No Departures)**
    - **Limitation:** The study assumes a “static” or “insert-only” environment where items arrive and stay forever. It does not account for Dynamic Bin Packing, where items also depart (e.g., tasks finishing in a cloud server), leading to fragmentation over time.

- **Future Scope:** Analyzing Fully Dynamic Bin Packing, incorporating item departures and measuring "migration cost"—the cost of moving items between bins to defragment the space.

3. **Homogeneity of Bins**

   - **Limitation:** The current scope assumes all bins are identical with a fixed capacity. Many real-world scenarios (e.g., heterogeneous server clusters or mixed fleets of delivery trucks) involve Variable Sized Bin Packing (VSBPP).
   - **Future Scope:** Investigating heuristics for Heterogeneous Bin Packing, where the algorithm must choose between different bin sizes to minimize total cost rather than just bin count.

4. **Practicality of Asymptotic Schemes**

   - **Limitation:** While the project analyzes APTAS and Karmarkar-Karp theoretically, their practical utility is limited by high time complexity on small-to-medium datasets. As noted in the proposal, the complexity of APTAS ($n^{O(1/\epsilon)}$) is often prohibitive for practical use, and Karmarkar-Karp relies on heavy machinery (Ellipsoid method) that is computationally expensive for $N < 1000$.
   - **Future Scope:** Exploring AFPTAS (Asymptotic Fully Polynomial Time Approximation Schemes) or hybrid metaheuristics that combine the theoretical guarantees of LP-based methods with the speed of local search (like Tabu Search) to bridge the gap between theory and real-time application.

5. **Adversarial vs. Stochastic Inputs**

   - **Limitation:** The experimental evaluation focuses on standard benchmarks (Falkenauer, Scholl). While effective for comparison, these may not reflect adversarial worst-case inputs that specifically break greedy heuristics (as proven in the theoretical bounds of First-Fit) or highly specific real-world distributions (e.g., heavy-tailed internet traffic).
   - **Future Scope:** Conducting a Robustness Analysis using procedurally generated adversarial datasets designed to trigger the worst-case performance ratios (1.7 for FF) to empirically validate the lower bounds derived in the theoretical section.

6. **Single-Objective Optimization**

   - **Limitation:** The project optimizes solely for the number of bins. It does not consider secondary objectives like load balancing (spreading items evenly) or energy efficiency (minimizing active bins for power saving), which are often trade-offs in real-world engineering.
   - **Future Scope:** Multi-objective optimization analysis, comparing how algorithms like Best-Fit (which packs tightly) vs. Worst-Fit (which load balances) perform when both space efficiency and resource distribution are prioritized.

17 Bonus Disclosure

This section serves as an explicit disclosure to the evaluator, indicating specific report components that we are choosing to submit for **Bonus Credit** consideration. The following table identifies these components and their locations within the report.

Component Type	Component Description	Location in Report
1	The Karmarkar Karp Algorithm	Section 8
2	Hoberg and Rothvoß	Section 9
3	The Martello-Toth Procedure	Section 12
4	Manim Animations	Youtube Links Below
5	Live Heuristic Animations Visualiser	Link Below

Video References:
1. `https://www.youtube.com/watch?v=xNtxqb9TEok`
2. `https://www.youtube.com/watch?v=pc60EVLT4j0`
3. `https://www.youtube.com/watch?v=BHxoNvHYT70`
4. `https://www.youtube.com/watch?v=mZl0LN9C3R4`

Live Heuristic Simulator:
`https://binpacking-1d-visualiser.streamlit.app/`

References

[1] R. M. Karp, 'Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (eds.), Plenum Press, New York, 1972, pp. 85–103.

[2] N. Karmarkar and R. M. Karp, 'The differencing method of set partitioning," *Computer Science Division*, Univ. of California, Berkeley, Tech. Rep. UCB/CSD 82-113, 1982.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

[4] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing*, 3(4), 1974.

[5] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin-packing: a survey," in *Approximation Algorithms for NP-hard Problems*, PWS, 1997.

[6] W. Fernández de la Vega and G. Lueker, "Bin packing can be solved within $1+\varepsilon$ in linear time," *Combinatorica*, 1(4):349–355, 1981.

[7] C. C. Lee and D. T. Lee, "A simple on-line bin-packing algorithm," *Journal of the ACM*, 32(3), 1985.

[8] K. Jansen and K. Klein, "A robust AFPTAS for online bin packing with cardinality constraints," *SIAM Journal on Computing*, 43(4), 2013.

[9] R. Hoberg and T. Rothvoß, "A nearly linear-time algorithm for bin packing with $O(\log OPT)$ bins," *FOCS*, 2017.

[10] G. Dósa. The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is 11/9. *Algorithmica*, 42:267–284, 2007.

[11] C. C. Lee and D. T. Lee. A simple on-line bin-packing algorithm. *Journal of the ACM*, 32(3):562–572, 1985.

[12] K. Jansen and D. Kraft. A faster FPTAS for bin packing with cardinality constraints. *Theoretical Computer Science*, 505:17–25, 2013.

[13] R. Hoberg and T. Rothvoß. A logarithmic additive integrality gap for bin packing. In *Proceedings of the 28th ACM-SIAM SODA*, 2017.

[14] Raphael Clifford, Benjamin Sach, University of Bristol, Lecture Slides - Advanced Algorithms Part 4,"