

Polynomial-Time Approximation Schemes for the 1-D Bin Packing Problem

Pranav Swarup Kumar

November 20, 2025

Contents

1	Introduction (1.1)	2
2	Proof of NP-hardness (1.2)	2
3	Approximation Algorithms for Bin Packing	6
3.1	Approximation Ratios and Basic Guarantees	6
4	The $3/2$ Inapproximability baseline (Partition \rightarrow Bin Packing)	7
4.1	Why Partition appears inside Bin Packing	7
4.2	Why a $(3/2 - \delta)$ -approximation would decide Partition	7
4.3	Why the number $3/2$ is the magic threshold	8
4.4	Interpretation	9
5	Special Cases Admitting Polynomial-Time Solutions	9
6	APTAS for Bin Packing	10
6.1	Intuitive Motivation	10
7	The Four-Step APTAS Framework	10
7.1	Step 1: Remove the Smaller Items	11
7.2	Step 2: Linear Grouping and Rounding	11
7.3	Step 3: Solve the Bounded-Type Instance Exactly	12
7.4	Step 4: Unround and Reinsert Small Items	13
8	Approximation Schemes Beyond APTAS	13
8.1	PTAS (Polynomial-Time Approximation Scheme)	13
8.2	Why No FPTAS Exists	14
8.3	AFPTAS (Asymptotic FPTAS)	14
8.4	Hierarchy of Schemes at a Glance	14
9	Worked Example: APTAS on a Medium-Sized Instance	15
10	Comparative Discussion	16

1 Introduction (1.1)

The bin packing problem tries to minimize the number of unit-capacity bins required to pack n items of sizes $s_1, s_2, \dots, s_n \in (0, 1]$.

In this section, 1-D bin packing is proved to be NP-Hard by reduction from the Partition Problem. *Approximation algorithms* which are extensively used to produce near-optimal solutions, are defined and discussed.

2 Proof of NP-hardness (1.2)

Optimization Problem Definition

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

Instance. A finite multiset of items

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

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)

$$B_1, B_2, \dots, B_m$$

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

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

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

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

and an integer $k > 0$.

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

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

and every item appears in exactly one bin?

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

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

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

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

Hence, if the decision problem could be solved in polynomial time, the optimal packing number could be obtained by binary search over $k \in \{1, \dots, n\}$, 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.

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

¹<https://www.youtube.com/watch?v=xNtxqb9TEok>

Reduction SUBSET-SUM \leq_p PARTITION.

Let $(a_1, \dots, a_n; t)$ 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

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

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 $I \subseteq \{1, \dots, n\}$ 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 $\{a_1, \dots, a_n\}$ whose sum is

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

hence the original SUBSET-SUM instance is a YES instance.

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.

Reduction $\text{PARTITION} \leq_p \text{BIN-PACK-DEC}$.

Let (a_1, \dots, 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:

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

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 $\text{PARTITION} \leq_p \text{BIN-PACK-DEC}$.

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 $\text{SUBSET-SUM} \leq_p \text{BIN-PACK-DEC}$. 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.

Remarks on edge cases and polynomiality

The reductions above include simple preprocessing to handle degenerate inputs. In the reduction $\text{SUBSET-SUM} \leq_p \text{PARTITION}$ 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 $\text{PARTITION} \leq_p \text{BIN-PACK-DEC}$ 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 Approximation Algorithms for Bin Packing

Having established the NP-hardness of the Bin Packing problem, we turn to the study of efficient approximation algorithms. Since finding an optimal packing is computationally intractable unless $P = NP$, the natural question becomes:

How close can we get to the optimal solution in polynomial time?

This section surveys classical constant-factor heuristics, modern polynomial-time approximation schemes, and asymptotic variants tailored to the unique structure of Bin Packing.

3.1 Approximation Ratios and Basic Guarantees

An algorithm A for a minimization problem is said to be a ρ -approximation if for every instance I ,

$$\frac{A(I)}{\text{OPT}(I)} \leq \rho.$$

For Bin Packing, the classical heuristics are extremely simple yet surprisingly effective in practice.

Greedy Heuristics

- **First-Fit (FF)**: scan bins in order and place the item in the first bin that fits.
- **Best-Fit (BF)**: place an item in the bin that leaves the *least* remaining space after insertion.
- **First-Fit Decreasing (FFD)** and **Best-Fit Decreasing (BFD)**: sort items by non-increasing size and run FF or BF.

These heuristics satisfy the following classical bounds due to Johnson, Demers, Ullman, Garey, and Graham [4, 5]:

$$\text{FFD}(I), \text{BFD}(I) \leq \frac{11}{9} \text{OPT}(I) + 1.$$

Although these algorithms can be implemented in $O(n \log n)$ time, they are fundamentally limited in their worst-case guarantees, motivating the development of PTAS and APTAS schemes.

4 The $3/2$ Inapproximability baseline (Partition \rightarrow 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]

4.1 Why Partition appears inside Bin Packing

Consider the decision version of PARTITION: given positive integers a_1, \dots, 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

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

Thus, distinguishing between $\text{Opt} = 2$ and $\text{Opt} \geq 3$ would solve Partition.

4.2 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 $\text{Opt} = 2$, 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 $\text{Opt} \geq 3$, 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 $P = NP$.

4.3 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.

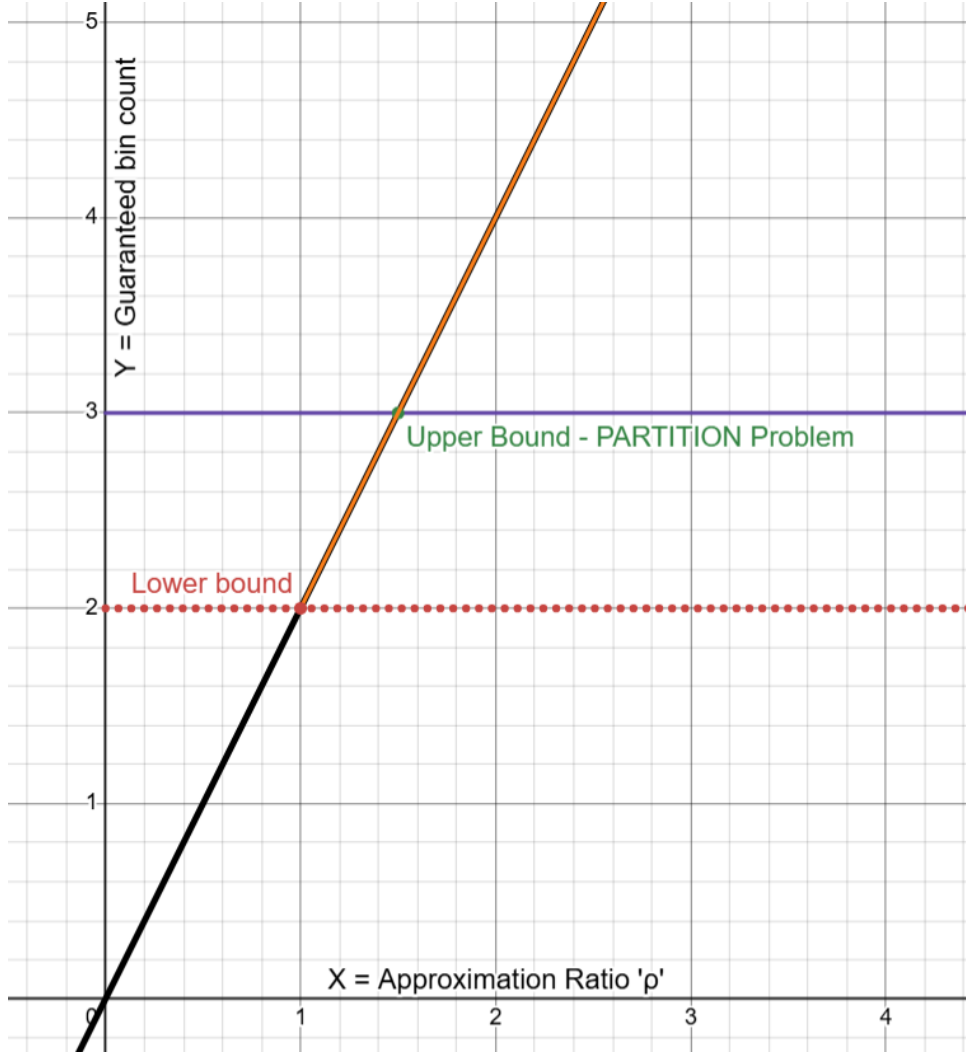


Figure 1: Approximation Ratio Bounds

4.4 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.

5 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.

6 APTAS for Bin Packing

6.1 Intuitive Motivation

The baseline hardness barrier (Section 4) 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:*

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

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.

7 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]

7.1 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:

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

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.

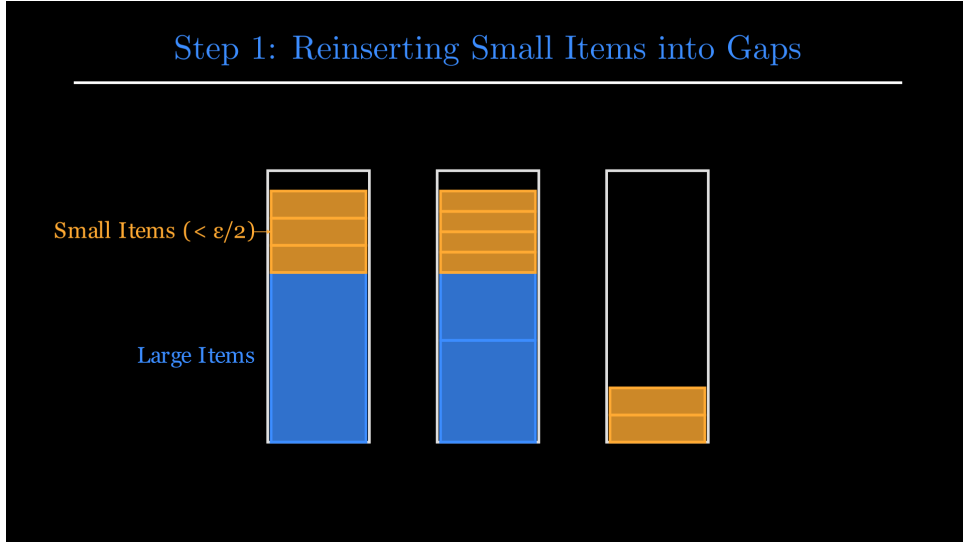


Figure 2: Step 1 APTAS for Bin-Packing

7.2 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 $k \leq \varepsilon \cdot \text{OPT}$. 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 $\varepsilon \cdot \text{OPT}$. We see this with stacked bars, showing each group collapsing upward to a uniform height.

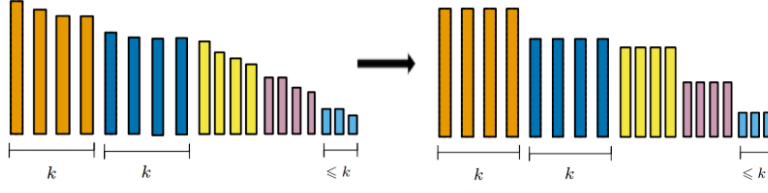


Figure 3: Step 2 APTAS for Bin-Packing

7.3 Step 3: Solve the Bounded-Type Instance Exactly

Motivation. Once rounding is complete, the instance has:

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

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, \dots, 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.

Step 3: Enumerating Feasible Bin Types

			Vector \mathbf{v}_3	
	Type 1	Type 2	Type 3	Type T
s_1	1	0	2	0
s_2	0	2	0	1
s_3	1	1	0	3

Solve: $\sum_{t=1}^T x_t \cdot \mathbf{v}_t = \mathbf{n}_{\text{items}}$

Figure 4: Step 3 APTAS for Bin-Packing

7.4 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 $1 + \varepsilon \text{OPT}$, 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.

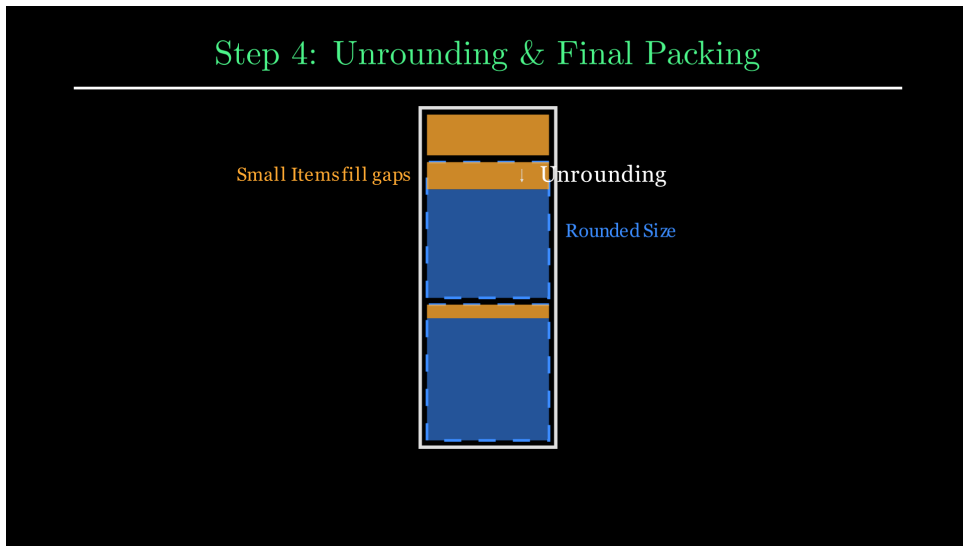


Figure 5: Step 4 APTAS for Bin-Packing

8 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.

8.1 PTAS (Polynomial-Time Approximation Scheme)

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

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

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), removing that additive slack altogether is impossible under standard complexity assumptions.

8.2 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 1 (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 $1/\varepsilon$.

8.3 AFPTAS (Asymptotic FPTAS)

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

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

just like an APTAS, but with running time

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

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.

8.4 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: $(1 + \varepsilon)\text{OPT} + O(1)$.
- 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.

9 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 $(0, 1]$):

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

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.

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:

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

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.

Step 3: Enumerating Feasible Bin Types

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

$$\begin{array}{lll} T_1 : & 0.52 + 0.48 & (= 1.00) \\ T_2 : & 0.52 + 0.31 & (= 0.83) \\ T_3 : & 0.48 + 0.31 & (= 0.79) \\ T_4 : & 0.31 + 0.29 + 0.29 & (= 0.89) \end{array}$$

These types are easily enumerated because the number of size classes is small (4) and the number of items per bin is at most 3.

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

$$\begin{array}{ll} \text{Bin A:} & 0.52 + 0.48 = 1.00 \\ \text{Bin B:} & 0.52 + 0.48 = 1.00 \\ \text{Bin C:} & 0.31 + 0.29 + 0.29 = 0.89 \end{array}$$

Thus $\text{OPT}(L') = 3$.

Step 4: Unrounding and Final Packing

We now replace rounded items with their original items. Because rounding increased sizes, unrounding can only make packing easier. The bins become:

Bin A: 0.52, 0.48, Bin B: 0.49, 0.47, Bin C: 0.31, 0.30, 0.29, 0.18.

Bin C has total load:

$$0.31 + 0.30 + 0.29 + 0.18 = 1.08.$$

This slightly exceeds capacity, so we place the last item (0.18) into a new bin. Thus we obtain:

Bin 1: $0.52 + 0.48 = 1.00$
 Bin 2: $0.49 + 0.47 = 0.96$
 Bin 3: $0.31 + 0.30 + 0.29 = 0.90$
 Bin 4: 0.18

Therefore the APTAS produces:

$$A_\varepsilon(I) = 4, \quad \text{while } \text{OPT}(I) = 3.$$

This satisfies the guarantee

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

with $\varepsilon = 0.25$.

10 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

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 $O(\text{OPT})$ 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

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

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 $\text{OPT} + O(\log \text{OPT})$ bins, giving near-optimal additive performance. These techniques are extremely powerful but require solving large linear programs and are slower than combinatorial heuristics.[13]

Glossary

- **Approximation Guarantee:** The multiplicative factor by which an algorithm may exceed OPT on large instances.
- **Additive Gap:** The fixed number of extra bins beyond OPT that may be used.
- **Runtime:** Practical computational effort; “Fast” refers to greedy or harmonic heuristics, while “Slow” refers to LP-heavy or configuration-based methods.

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,”