

A Comparative Analysis of Approximation Algorithms for the One-Dimensional Bin Packing Problem

From Greedy Heuristics to Discrepancy Theory

Team Alan Turing
Aman Jayesh, Mukund Hebbar,
Pranav Swarup Kumar,
Shreyas Ramasubramanian, Shreyash Chandak

December 1, 2025

Table of Contents

- 1 Introduction & Complexity
- 2 Foundational Heuristics
- 3 APTAS Framework
- 4 Karmarkar-Karp (1982)
- 5 Modern Breakthroughs (2013-2015)
- 6 Exact Algorithms: Martello-Toth Procedure
- 7 Alternative Exact Approaches: Dyckhoff's Model II
- 8 Metaheuristics: GGA
- 9 Conclusion & Hypothesis

Problem Definition & Hardness

The 1D Bin Packing Problem (1DBPP)

Given items $I = \{s_1, \dots, s_n\}$ with $s_i \in (0, 1]$, pack them into minimum m unit-capacity bins.

The Inapproximability Barrier:

- By reduction from PARTITION, distinguishing between $OPT = 2$ and $OPT = 3$ is NP-Hard.
- **Theorem:** No polynomial-time algorithm can achieve an approximation ratio better than $3/2$ unless P=NP.
- **Implication:** We must settle for:
 - ① Asymptotic Guarantees ($OPT \rightarrow \infty$).
 - ② Schemes dependent on ϵ (APTAS).
 - ③ Additive Error terms (e.g., $OPT + C$).

Online Heuristic 1: First-Fit (FF)

Mechanics:

- Process items in order of arrival s_1, s_2, \dots, s_n .
- Place item s_i in the **first** bin B_j ($j = 1 \dots k$) where it fits.
- If it fits nowhere, open a new bin B_{k+1} .

Complexity:

- Naive: $O(n^2)$.
- Optimized: $O(n \log n)$ using Segment Trees to query the first valid bin.

Theoretical Bound (Theorem 1)

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

Note: FF tends to leave larger, contiguous gaps in earlier bins, which helps accommodate future items.

Online Heuristic 2: Best-Fit (BF)

Mechanics:

- Place item s_i in the bin with the **minimum residual capacity** sufficient to hold it.
- Goal: “Tightest fit” to minimize immediate waste.

The “Sand” Problem (Fragmentation):

- BF creates bins that are nearly full but contain tiny, unusable gaps (“sand”).
- These splinters are often too small for any future item, rendering that capacity wasted.

Theoretical Bound (Theorem 2)

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

Despite the strategy difference, it shares the same worst-case ratio as FF.

Offline Heuristics: FFD & BFD

The Power of Pre-processing:

- **Rule:** Sort items such that $s_1 \geq s_2 \geq \dots \geq s_n$.
- **Intuition:** Placing “big rocks” first ensures difficult items are packed when bins have maximum capacity. Small items (“sand”) fill the remaining gaps later.

Algorithms:

- **First-Fit Decreasing (FFD):** Sort, then run First-Fit.
- **Best-Fit Decreasing (BFD):** Sort, then run Best-Fit.

Johnson's Theorem & Dósa's Tight Bound (2007)

Sorting improves the approximation ratio significantly:

$$FFD(L) \leq \frac{11}{9} OPT(L) + \frac{6}{9} \approx 1.22 \cdot OPT(L)$$

Qualitative Comparison: FF vs. BF

Although they have the same worst-case bound (1.7), their average-case behavior differs due to **Space Fragmentation**.

First-Fit Strategy	Best-Fit Strategy
Oblivious to “tightness”.	Minimizes local residual space.
Leaves large, contiguous gaps in early bins.	Creates many tiny, unusable gaps (“sand”).
Statistically better for accommodating future medium-sized items.	Risk of “suffocating” on future items slightly larger than the gaps.

Advanced Online: Harmonic-k Algorithm

Concept: Unlike FF/BF, Harmonic- k classifies items by size intervals to limit fragmentation.

Mechanism

- ➊ Divide the interval $(0, 1]$ into k sub-intervals:

$$I_j = \left(\frac{1}{j+1}, \frac{1}{j} \right] \text{ for } j = 1 \dots k-1, \text{ and } I_k = \left(0, \frac{1}{k} \right]$$

- ➋ **Dedicated Bins:** Items of type I_j are **only** packed into bins dedicated to type j .
- ➌ **Packing Rule:** A bin for type j can hold exactly j items (since $s_i > \frac{1}{j+1}$ implies $j+1$ items would exceed capacity, but $s_i \leq \frac{1}{j}$ allows j items).

Performance:

- As $k \rightarrow \infty$, the asymptotic approximation ratio improves.
- **Limit:** $\Pi_\infty \approx 1.6910$ (Better than FF/BF's 1.7).
- **Trade-off:** Reduces flexibility but guarantees bounded space wastage per bin type.

Phase 2: APTAS (Fernandez de la Vega & Lueker)

Goal: Achieve $(1 + \epsilon)OPT + O(1)$ for any fixed $\epsilon > 0$.

The 4-Step Framework

- ① **Eliminate Small Items:** Temporarily discard items $< \epsilon/2$.
- ② **Linear Grouping:** Round sizes to reduce instance complexity.
- ③ **Exact Solution:** Solve the restricted instance using DP/IP.
- ④ **Reinsertion:** Add small items back into gaps.

APTAS Step 2: Linear Grouping (The Core Mechanic)

To solve the problem exactly, we must reduce the number of distinct item sizes to a constant K .

Procedure:

- ① Sort large items in descending order.
- ② Partition them into $1/\epsilon^2$ groups of size k (where $k \approx \epsilon \cdot OPT$).
- ③ **Rounding:** In each group, round all items *up* to the size of the largest item in that group.

Proof Sketch (Grouping Error)

Let L be the original list and L' be the rounded list. Because we round up, L' dominates L . However, the rounded items of group i are equal to the smallest items of group $i - 1$. Thus, we can essentially “shift” the packing.

$$OPT(L') \leq OPT(L) + k \text{ (items)} \approx (1 + \epsilon)OPT$$

APTAS Steps 3 & 4: Solving and Unrounding

Step 3: Solve Bounded Instance

- With constant distinct sizes, the number of valid bin *configurations* is constant.
- We solve this using Integer Programming (IP) in constant dimensions ($O(1)$ variables), which takes constant time for fixed ϵ .

Step 4: Unrounding & Reinsertion

- Replace rounded items with original sizes (valid because $\text{size}(\text{orig}) \leq \text{size}(\text{rounded})$).
- Grease the small items ($< \epsilon/2$) into remaining gaps using First-Fit.
- If a small item doesn't fit, open a new bin. Since items are small, these extra bins are densely packed, incurring negligible cost.

Result: $A_\epsilon(I) \leq (1 + \epsilon)OPT + 1$.

The Additive Breakthrough: Karmarkar-Karp (KK82)

For 30 years, this was the theoretical benchmark.

The Guarantee

$$A(I) \leq OPT(I) + O(\log^2 OPT(I))$$

This effectively eliminates the multiplicative error ($\epsilon \cdot OPT$) seen in APTAS.

Key Innovations:

- ① **Gilmore-Gomory LP:** Formulating the problem based on Patterns.
- ② **Ellipsoid Method + Knapsack Oracle:** Solving exponential constraints efficiently.
- ③ **Iterative Rounding:** A loop that solves a sequence of diminishing residual problems.

Mechanism 1: Solving the Exponential LP

The Configuration LP has a variable x_p for every valid pattern p (exponential number of variables).

The Dual Solution:

- We consider the *Dual LP*, which has polynomial variables but exponential constraints.
- To solve this using the **Ellipsoid Method**, we need a **Separation Oracle**.
- The Oracle must find if any constraint is violated, which corresponds to finding the pattern with maximum “profit” (dual values).
- **The “Aha!” Moment:** Finding the max profit pattern is exactly the **Knapsack Problem**.
- Since Knapsack has an FPTAS, we can solve the LP to near-optimality in polynomial time.

Mechanism 2: Iterative Rounding Procedure

We cannot simply round x_p values because standard rounding introduces linear error. KK82 uses a refined loop:

- 1: **while** Instance is large **do**
- 2: 1. **Solve LP** to get fractional solution x .
- 3: 2. **Integral Packing:** Take $\lfloor x_p \rfloor$ bins of each pattern.
- 4: 3. **Residual:** Collect the fractional remainders $(x_p - \lfloor x_p \rfloor)$ as a new instance I_{res} .
- 5: 4. **Grouping:** Apply *Geometric Grouping* to I_{res} to reduce item types.
- 6: 5. **Repeat** with the reduced residual instance.
- 7: **end while**

Proof Sketch: The $O(\log^2 OPT)$ Bound

Why $\log^2 OPT$?

- ① **Geometric Decay:** The size of the residual instance I_{res} drops by a constant factor (e.g., half) in each iteration.
- ② **Loop Count:** Therefore, the loop runs $T = O(\log n)$ times.
- ③ **Error Accumulation:**
 - In each iteration t , the Geometric Grouping step introduces a small additive error E_t .
 - This error is bounded: $E_t \approx O(\log(\text{item size constraint}))$.
 - Summing errors over $O(\log n)$ iterations:

$$\sum_{t=1}^{\log n} O(\log n) \approx O(\log^2 n) \approx O(\log^2 OPT)$$

Breaking the Barrier: Rothvoß (2013)

The Problem with KK82: “Spiky” patterns (bins with many copies of one small item) caused the $O(\log^2 OPT)$ error during the rounding phase.

The 2013 Solution:

- **Discrepancy Theory:** Uses the Lovett-Meka Constructive Partial Coloring Lemma.
- **Gluing:** Conceptually “glues” small items together to smooth out spiky patterns.
- **Result:** Improved bound to $OPT + O(\log OPT \cdot \log \log OPT)$.

Hoberg & Rothvoß (2015): The Tight Bound

To reach the optimal additive gap, they fundamentally restructured the packing process.

The 2-Stage Packing Mechanism:

- ① **Stage 1 (Items → Containers):** Pack items into “Containers” (multisets fitting in a bin).
- ② **Stage 2 (Containers → Bins):** Pack Containers into Bins.

Why this works?

Containers inherently limit the number of copies of any object in a pattern (structural smoothness). Applying Lovett-Meka rounding to Containers incurs only $O(1)$ error per iteration.

Final Result: $OPT + O(\log OPT)$.

Exact Algorithms: Martello-Toth Procedure (MTP)

Heuristics give approximations, but MTP is an exact Branch-and-Bound algorithm to find the minimal m .

Key Components:

- ① **Lower Bounds (L_1, L_2)**: Essential for pruning the search tree.
- ② **Reduction Procedures**: Using *Dominance Criteria* to fix bins early.
- ③ **Branch-and-Bound (DFS)**: Systematically exploring bin compositions.

Mathematical Lower Bounds: L_1 and L_2

The L_1 Bound (Volume Bound)

$$L_1 = \lceil \frac{\sum s_i}{C} \rceil$$

- Continuous relaxation (fluid model).
- Worst-case performance ratio: 1/2.

The L_2 Bound (Martello-Toth)

- Analyzes unavoidable "wasted" space.
- Classifies items into Large (N_1), Medium (N_2), and Small (N_3) based on a parameter K .
- **Formula:**

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

- R_{N_2} is the residual space in bins with Medium items.

Reduction & Dominance

Before branching, we reduce the problem size by fixing "obvious" bins.

Dominance Criterion

A feasible set F_1 dominates F_2 if the optimal solution with a bin $B = F_1$ is never worse than with $B = F_2$.

Reduction Algorithm (Greedy Matching):

- Identify the largest remaining item i .
- Check if it forms a "perfect" bin with another item j (e.g., $s_i + s_j = C$).
- If so, fix bin $\{i,j\}$ and remove items from the pool.

The Branch-and-Bound Algorithm

Procedure:

- ① **Initialize:** Calculate global Lower Bound ($LB = L_2$) and Heuristic Upper Bound (UB). If $LB = UB$, stop.
- ② **Branching:** Build a solution bin by bin (Depth-First Search). Try to fill the current bin with the largest fitting item.
- ③ **Pruning (Critical):** At any node:
 - Let m_{curr} be fixed bins.
 - Calculate $L_2(I_{rem})$ for remaining items.
 - If $m_{curr} + L_2(I_{rem}) \geq UB$, **Prune!** (Backtrack).
- ④ **Update:** If a better solution is found, update UB .

Introduction to Dyckhoff's One-Cut Model (1981)

Motivation

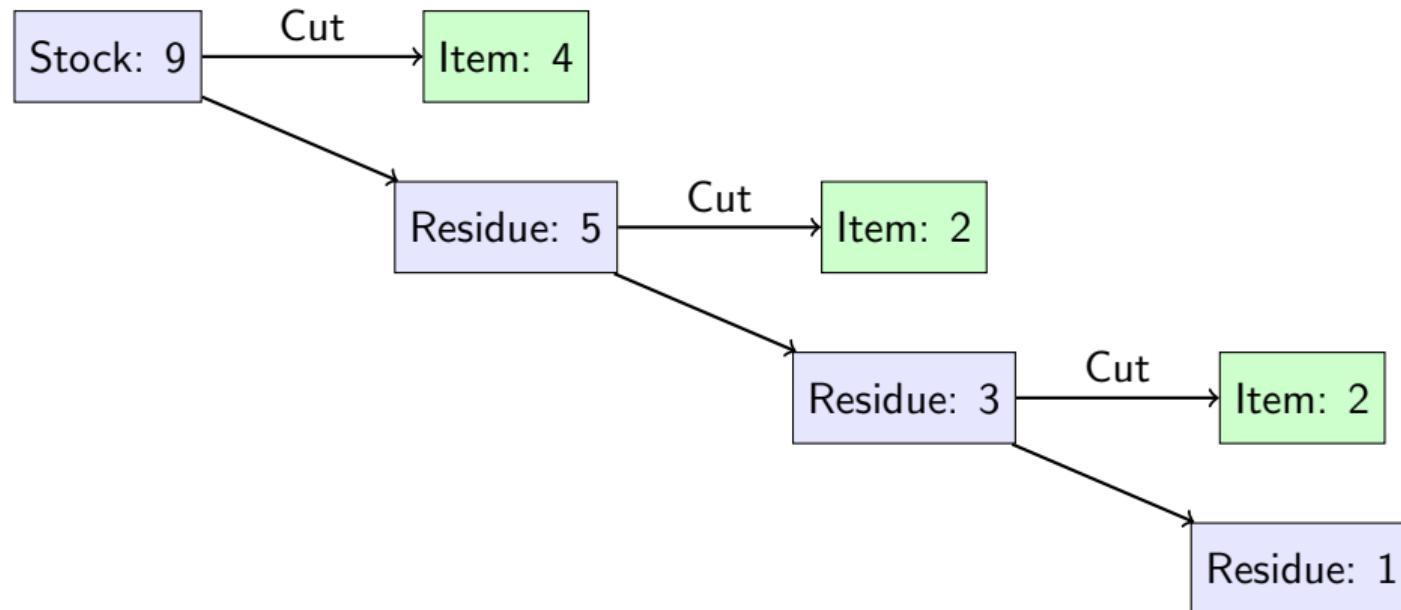
The classic Gilmore-Gomory LP relies on **Column Generation** because it has an exponential number of cutting pattern variables. **Dyckhoff's Model II** solves the problem using **standard Simplex** by ensuring a polynomial number of variables.

The Core Idea: Recursive Cuts

- Instead of defining a whole pattern at once (e.g., $\{4, 2, 2, 1\}$), we define "**One-Cut**" operations.
- A piece of length k is cut into:
 - ① An order item of length l .
 - ② A residual piece of length $k - l$.
- Complex patterns are built sequentially through recursion.

Visualizing the One-Cut Principle

Consider cutting a Stock length of **9** into items **4, 2, 2, 1**.



Each step produces one item and one smaller residual piece for further cutting.

Mathematical Formulation: Variables

Let S be stock lengths, D be demand lengths, and R be valid residual lengths.

Decision Variables:

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

Interpretation:

- $y_{k,l}$ represents the number of pieces of length k that are cut to produce **one** item of order length l .
- This operation implicitly produces a residue of size $k - l$.

Mathematical Formulation: Constraints

The model relies on **Flow Conservation** for every length I ($\text{Input} \geq \text{Output}$).

Flow Balance Constraint

For every length $I \in D \cup R$:

$$\underbrace{\sum_{k \in A_I} y_{k,I}}_{\text{Created by cuts}} + \underbrace{\sum_{k \in B_I} y_{k+I,k}}_{\text{Created as Residue}} \geq \underbrace{\sum_{k \in C_I} y_{I,k}}_{\text{Used for smaller cuts}} + \underbrace{N_I}_{\text{Final Demand}}$$

- **LHS:** Total availability of length I (produced directly + leftover).
- **RHS:** Total consumption of length I (cut into smaller items + shipped as demand).

Comparison: Model I vs. Model II

Feature	Model I (Gilmore-Gomory)	Model II (Dyckhoff)
Variables	Exponential (Patterns)	Polynomial (One-Cuts)
Constraints	$ D $ (Small)	$ D + R $ (Larger)
Solver	Column Generation	Standard Simplex
Structure	Static Patterns	Dynamic Flow

Conclusion: Model II is superior when the number of distinct lengths is moderate, as it avoids complex pricing subproblems.

Why Metaheuristics? The Need for GGA

Limitation of Classical Heuristics:

- Greedy (FFD) gets stuck in local optima.
- Theoretical schemes (KK82) are computationally impractical (N^{high}).

The Genetic Approach:

- **Standard GA Failure:** Encoding solution as a list of items ($Item_i \rightarrow Bin_j$) creates a massive search space with redundancy (ordering doesn't matter).
- **The Solution (Falkenauer): Grouping Genetic Algorithm (GGA).**
- **Core Idea:** The fundamental unit of evolution is the **Group (Bin)**, not the item.
- We evolve a population of *packings*, preserving well-packed bins across generations.

The Cost Function: Navigating the Landscape

Problem: Minimizing the number of bins N creates a “flat” fitness landscape. A solution with $N = 10$ (mostly full) looks the same as $N = 10$ (barely full). The algorithm has no gradient to follow.

Falkenauer’s Objective Function: Maximize the average “fullness” with non-linear scaling ($k > 1$):

$$f_{BPP} = \frac{\sum_{i=1}^N (F_i/C)^k}{N}$$

Where F_i is the sum of items in bin i , C is capacity, and $k = 2$.

Why $k = 2$?

It rewards “extremist” solutions. Two half-full bins score lower than one full bin + one empty bin. This drives the algorithm to completely fill bins and empty others.

The Crossover Operator (Group-Based)

Standard crossover destroys valid bin structures. GGA uses a specialized 4-step process:

- ① **Selection:** Choose two parents. Select a set of “best bins” from Parent A to inject.
- ② **Injection:** Insert these bins into Parent B.
- ③ **Elimination:** Some items now appear twice (once in the injected bins, once in Parent B's original bins). Remove the **original bins** in Parent B that contain these duplicates.
- ④ **Re-insertion:** The items from the removed bins that were *not* duplicates are now “loose”. Re-insert them using a heuristic (FFD or Dominance).

Result: Offspring inherits high-quality compact bins from A, while adapting the rest.

Mutation & Local Optimization

Mutation Strategy:

- Randomly select a small number of bins.
- **Dissolve** them completely.
- Treat their items as “loose” and re-insert them.
- Purpose: Prevents premature convergence by forcing the algorithm to break up suboptimal local packings.

The Role of Heuristics (Hybridization):

- GGA relies heavily on a local heuristic for the “Re-insertion” phase.
- **Dominance Criterion (Martello & Toth):** Prefer packings that dominate others (leave strictly less space or accommodate larger items).
- **Practical Implementation:** Often replaced by First-Fit Decreasing (FFD) or Best-Fit Decreasing (BFD) for speed during re-insertion.

Summary of Algorithms Hierarchy

Algorithm	Type	Guarantee	Speed
First-Fit	Online	$1.7 \cdot OPT$	Fast
Harmonic- k	Online	$\approx 1.69 \cdot OPT$	Fast
FFD	Offline	$1.22 \cdot OPT$	Fast
Martello-Toth	Exact	Optimal	Exponential
Dyckhoff One-Cut	Exact LP	Optimal	Pseudo-Polynomial
APTAS	Scheme	$(1 + \epsilon)OPT + 1$	Slow
Karmarkar-Karp	Theoretical	$OPT + O(\log^2 OPT)$	Very Slow
Hoberg-Rothvoß	State-of-Art	$OPT + O(\log OPT)$	Theoretical