Approximation Algorithms for the Bin Packing Problem

Sr. No.	Name of Group Member	Student ID
1	Smit Fefar	202201253
2	Dishant Patel	202201260
3	Mihirkumar Patel	202201506
4	Zeel Danani	202201507
5	Dishank Thakkar	202201518

Instructor: Prof. Rachit Chhaya

Course: IT584 - Approximation Algorithms

Semester: Winter 2025

April 20, 2025

Contents

1 Abstract

This project explores and compares several approximation algorithms for the Bin Packing Problem, including Next Fit, First Fit, First Fit Decreasing, Harmonic-based methods, and a linear programming-based approach that combines the Gilmore-Gomory LP formulation with the Karmarkar-Karp rounding algorithm.

We evaluate all algorithms based on their average waste per bin, average bin utilization, and total number of bins used, using a Monte Carlo simulation over multiple input sizes.

Our analysis reveals how each algorithm scales with input size and highlights trade-offs between solution quality and computational efficiency.

The results demonstrate that as the number of items increases, most algorithms achieve better bin utilization and reduced waste, providing valuable insights into their practical performance.

2 Problem Statement and Mathematical Formulation

Given a collection of items, each with a specific size, and an unlimited supply of identical bins with fixed capacity of 1, the objective is to determine the minimum number of bins required to pack all the items without exceeding the capacity of any bin.

Objective: Minimize the number of bins

- Let the items be $u_1, u_2, \ldots, u_n$.
- Each item u_i has a size $s(u_i) \in (0, 1]$.
- Assign the items to bins $B_1, B_2, \ldots, B_m$ such that for every bin B_j:

$$\sum_{u_i \in B_j} s(u_i) \leq 1$$

- Minimize the number of bins used (m)

3 Proof of NP-Completeness

Reduction from Partition Problem:

The Partition Problem is defined as follows:

Given n positive integers $b_1, b_2, \ldots, b_n$ such that the total sum $B = \sum_{i=1}^{n} b_i$ is even, determine whether the indices $\{1, 2, \ldots, n\}$ can be partitioned into two disjoint sets S and T such that:

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i$$

It is a well-known fact that the Partition Problem is NP-complete.

We reduce the Partition Problem to the Bin Packing Problem in polynomial time:

Let us define:

$$u_i = \frac{2b_i}{B}$$

Then, If $b_i < \frac{B}{2}$, then $u_i < 1$:

$$\sum_{i=1}^{n} u_i = \sum_{i=1}^{n} \frac{2b_i}{B} = \frac{2\sum_{i=1}^{n} b_i}{B} = \frac{2B}{B} = 2$$

So, if we can pack all u_i into two bins of capacity 1 each, the Partition Problem is also solved. Thus, we have:

$$\text{Partition Problem} \leq_p \text{Bin Packing Problem}$$

If the Bin Packing Problem could be solved in polynomial time, so could the Partition Problem, which contradicts the NP-completeness of the Partition Problem. Hence, the Bin Packing Problem is NP-complete.

$$\text{Partition Problem is NP-complete} \Rightarrow \text{Bin Packing Problem is NP-complete}$$

Hence proved: Bin Packing is an NP-complete problem.

4 Approximation Algorithms

4.1 Next Fit (NF)

The Next Fit algorithm is a simple online algorithm for the bin packing problem. It maintains a single "current" bin and places each item into this bin if it fits. If the item does not fit, it closes the current bin and opens a new one.

4.1.1 Description

- Online algorithm - makes decisions about item placement without knowing sizes of future items
- Only considers the current open bin when placing an item
- If an item doesn't fit in the current bin, the bin is closed and a new one is opened
- Simple to implement with low computational overhead

4.1.2 Algorithm

Algorithm 1 Next Fit Bin Packing

$B \leftarrow \emptyset$ ▷ Set of bins
$current\_bin \leftarrow \emptyset$
$current\_capacity \leftarrow 0$
for each item u_i in input **do**
    if $current\_capacity + s(u_i) \leq 1$ **then**
        Add u_i to $current\_bin$
        $current\_capacity \leftarrow current\_capacity + s(u_i)$
    else
        Add $current\_bin$ to B
        $current\_bin \leftarrow \{u_i\}$ ▷ Open new bin
        $current\_capacity \leftarrow s(u_i)$
if $current\_bin$ is not empty **then**
    Add $current\_bin$ to B
return B, $|B|$

4.1.3 Example

Consider 6 items with sizes: [0.5, 0.6, 0.7, 0.5, 0.4, 0.3]

- **Bin 1:**
  - Add 0.5 (occupancy = 0.5)
  - 0.6 doesn't fit ($0.5 + 0.6 = 1.1 > 1$) → close Bin 1
- **Bin 2:**
  - Add 0.6 (occupancy = 0.6)
  - 0.7 doesn't fit ($0.6 + 0.7 = 1.3 > 1$) → close Bin 2
- **Bin 3:**
  - Add 0.7 (occupancy = 0.7)
  - 0.5 doesn't fit ($0.7 + 0.5 = 1.2 > 1$) → close Bin 3
- **Bin 4:**
  - Add 0.5 (occupancy = 0.5)
  - Add 0.4 (occupancy = 0.9)
  - 0.3 doesn't fit ($0.9 + 0.3 = 1.2 > 1$) → close Bin 4
- **Bin 5:**
  - Add 0.3 (occupancy = 0.3)
  - No more items → packing complete
- **Total bins used:** 5

The **optimal packing would use 3 bins:** [0.7, 0.3], [0.5, 0.5] and [0.6, 0.4]. demonstrating that Next Fit can use more bins than necessary.

4.1.4 Complexity

- Time complexity: $O(n)$ where n is the number of items
- Space complexity: $O(1)$ (no extra space needed beyond the input)

4.1.5 Approximation Ratio and Proof

The Next Fit algorithm has an approximation ratio of 2, meaning it never uses more than twice the optimal number of bins.

Proof:
Let OPT be the optimal number of bins and NF be the number of bins used by Next Fit. We need to show that $NF \leq 2 \cdot OPT$.

Key Observations:

1. For any two consecutive bins B_j and B_{j+1} in Next Fit's packing:

$$C(B_j) + C(B_{j+1}) > 1$$

where $C(B_j)$ denotes the sum of item sizes in bin B_j. This holds because if $C(B_j)+s(u_i) \leq 1$ for the next item u_i, Next Fit would have placed it in B_j.

2. The optimal number of bins satisfies:

$$OPT \geq \left\lceil \sum_{i=1}^{n} s(u_i) \right\rceil$$

since each bin has capacity 1.

Case 1: NF is even

- Pair the bins as $(B_1, B_2), (B_3, B_4), \ldots, (B_{NF-1}, B_{NF})$
- From Observation 1, each pair satisfies $C(B_j) + C(B_{j+1}) > 1$
- Summing over all $\frac{NF}{2}$ pairs:

$$\sum_{j=1}^{NF} C(B_j) > \frac{NF}{2}$$

- But $\sum_{j=1}^{NF} C(B_j) = \sum_{i=1}^{n} s(u_i) \leq OPT$ (from Observation 2)
- Therefore:

$$OPT > \frac{NF}{2} \implies NF < 2 \cdot OPT$$

Case 2: NF is odd

- Pair the first $NF - 1$ bins as in Case 1, leaving B_{NF} unpaired
- For the $\frac{NF-1}{2}$ pairs:

$$\sum_{j=1}^{NF-1} C(B_j) > \frac{NF-1}{2}$$

- Adding the last bin:

$$\sum_{j=1}^{NF} C(B_j) > \frac{NF-1}{2} + C(B_{NF})$$

- Since $C(B_{NF}) > 0$ (Positive quantity in RHS) and $\sum_{j=1}^{NF} C(B_j) \leq OPT$:

$$OPT > \frac{NF-1}{2} \implies NF < 2 \cdot OPT + 1$$

- This implies $NF \leq 2 \cdot OPT$

4.2 First Fit (FF)

First Fit is an online algorithm for the bin packing problem that places each item into the first bin that can accommodate it. If no existing bin has enough remaining capacity, it opens a new bin.

4.2.1 Description

- Online algorithm - processes items in the order they arrive
- For each item, checks all existing bins in order and places it in the first bin with sufficient space
- If no bin can accommodate the item, opens a new bin
- More efficient than Next Fit but requires more computation

4.2.2 Algorithm

Algorithm 2 First Fit Bin Packing

```
B ← ∅                                                    ▷ Set of bins
for each item a_i in input do
    item_placed ← false
    for each bin B_j in B do
        if remaining capacity of B_j ≥ s(a_i) then
            Add a_i to B_j
            item_placed ← true
            break
    if not item_placed then
        Create new bin with a_i
        Add new bin to B
return B, |B|
```

4.2.3 Example

Consider items with sizes: $[0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8]$ and bin capacity = 1

- Item 1 (0.2): New bin $B_1 = [0.2]$
- Item 2 (0.5): Fits in B_1: $[0.2, 0.5]$
- Item 3 (0.4): Doesn't fit in B_1 $(0.2 + 0.5 + 0.4 = 1.1 > 1)$, new bin $B_2 = [0.4]$
- Item 4 (0.7): Doesn't fit in B_1 and B_2, new bin $B_3 = [0.7]$
- Item 5 (0.1): Fits in B_1: $[0.2, 0.5, 0.1]$
- Item 6 (0.3): Doesn't fit in B_1, fits in B_2: $[0.4, 0.3]$
- Item 7 (0.8): Doesn't fit in any existing bins, new bin $B_4 = [0.8]$

Final packing uses 4 bins:

- $B_1 = [0.2, 0.5, 0.1]$ (sum=0.8)
- $B_2 = [0.4, 0.3]$ (sum=0.7)
- $B_3 = [0.7]$ (sum=0.7)
- $B_4 = [0.8]$ (sum=0.8)

Optimal packing uses 3 bins:

- $B_1 = [0.5, 0.4, 0.1]$ (sum=1)
- $B_2 = [0.7, 0.3]$ (sum=1)
- $B_3 = [0.8, 0.2]$ (sum=1)

4.2.4 Complexity

- Time complexity: $O(n^2)$ - For each item, may check all existing bins
- Space complexity: $O(n)$ - Need to store all items in bins

4.2.5 Approximation Ratio and Proof

Define the weight function $w(s)$ for items of size s as:

$$w(s) = \frac{6}{5}s + \text{penalty}(s) = \begin{cases} \frac{6}{5}s & s \leq \frac{1}{6} \\ \frac{18s-1}{10} & \frac{1}{6} < s \leq \frac{1}{3} \\ \frac{12s+1}{10} & \frac{1}{3} < s \leq \frac{1}{2} \\ \frac{6s+2}{5} & s > \frac{1}{2} \end{cases}$$

where the penalty terms are:

$$\text{penalty}(s) = \begin{cases} 0 & s \leq \frac{1}{6} \\ \frac{3s}{5} - 0.1 & \frac{1}{6} < s \leq \frac{1}{3} \\ 0.1 & \frac{1}{3} < s \leq \frac{1}{2} \\ 0.4 & s > \frac{1}{2} \end{cases}$$

Two key lemmas:

(i) If A is a set of items with $s(A) \equiv \sum_{a \in A} s(a) \leq 1$, then

$$w(A) \equiv \sum_{a \in A} w(a) \leq \frac{17}{10},$$

which implies $\text{OPT}(L) \leq \frac{17}{10} \cdot W(L)$.

(ii) FF packs ≥ 1 weight into each bin. $W(L) > \text{FF}(L) - 1$, Detailed proof of lemma 2 is quite extensive, for that reason we have not included proof of lemma 2.

From these lemmas, we can conclude that the asymptotic ratio satisfies:

$$\frac{\text{OPT}(L)}{\text{FF}(L)} \leq 1.7$$

This follows because:

- Lemma (i) shows $\text{OPT}(L) \leq \frac{17}{10} W(L)$
- Lemma (ii) shows $W(L) > \text{FF}(L) - 1$
- Combining these gives $\text{OPT}(L) \leq \frac{17}{10}(\text{FF}(L) - 1)$
- For large instances where $\text{FF}(L) \gg 1$, the -1 becomes negligible

Thus, proving that in a single bin the maximum weight that can be packed is ≤ 1.7 demonstrates that the asymptotic ratio is indeed 1.7.

Proof for lemma 1:

Proof. We know that $\sum_{f \in B_i} s_f \leq 1$ for all bins. The total weight consists of:

$$\sum W(s_f) = \underbrace{\sum (\text{base weight})}_{\leq 1.2} + \underbrace{\sum (\text{penalty})}_{\leq 0.5}$$

Base Weight Calculation:

$$\sum \frac{6}{5} s_f \leq \frac{6}{5} \times 1 = 1.2 \quad (\text{since } s_f \leq 1)$$

Penalty Term Analysis: We show $\sum \text{penalty} \leq 0.5$ through case analysis.

Case 1: Bin contains a large object ($s_f > \frac{1}{2}$)

- Penalty for large object: 0.4
- Only one large object fits per bin

Subcase 1.1: With medium object ($\frac{1}{3} < s_f \leq \frac{1}{2}$)

- Maximum one medium object can fit because:

$$\frac{1}{2} + 2 \times \frac{1}{3} > 1$$

- No small objects can fit since:

$$\frac{1}{2} + \frac{1}{3} + \frac{1}{6} > 1$$

- Penalty: 0.4 (large) + 0.1 (medium) = 0.5

Subcase 1.2: With small objects only ($s_f \leq \frac{1}{6}$)

- Maximum two small objects fit because:

$$\frac{1}{2} + 3 \times \frac{1}{6} > 1$$

- Their combined penalty:

$$\frac{3}{5} s_1 - 0.1 + \frac{3}{5} s_2 - 0.1 = \frac{3}{5}(s_1 + s_2) - 0.2 < 0.1 \quad (\text{since } s_1 + s_2 \leq \frac{1}{2})$$

- Total penalty: 0.4 (large) + 0.1 (small) < 0.5

Case 2: No large objects

- Maximum 5 items (medium + small) per bin
- Each contributes penalty ≤ 0.1
- Total penalty $\leq 5 \times 0.1 = 0.5$

In all cases, the total penalty never exceeds 0.5. Combined with the base weight of 1.2, the total weight per bin ≤ 1.7.

Hence Proved. □

4.3 First Fit Decreasing (FFD) Algorithm

4.3.1 High-Level Idea

First Fit Decreasing is an **offline** algorithm for bin packing that:

- Sorts all items in **non-increasing order** of size
- Applies the First Fit algorithm to the sorted list

4.3.2 Algorithm

Algorithm 3 First Fit Decreasing (FFD)

Sort items $u_1, u_2, \ldots, u_n$ such that $s(u_1) \geq s(u_2) \geq \cdots \geq s(u_n)$
Initialize empty set of bins $B = \emptyset$
for each item u_i in sorted order **do**
    for each bin b_j in B **do**
        if u_i fits in b_j **then**
            Place u_i in b_j
            break
    if u_i was not placed **then**
        Create new bin with u_i
        Add to B
return B, $|B|$

4.3.3 Example

Items: $[0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8]$
Bin Capacity: 1.0
Step 1: Sort items in decreasing order
Sorted list: $[0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1]$
Step 2: Pack items using First-Fit strategy

- **Item 1 (0.8):** New bin $B_1 = [0.8]$
- **Item 2 (0.7):**
  - Check B_1: $0.8 + 0.7 = 1.5 > 1 \rightarrow$ Doesn't fit.
  - New bin $B_2 = [0.7]$
- **Item 3 (0.5):**
  - Check B_1: $0.8 + 0.5 = 1.3 > 1 \rightarrow$ Doesn't fit.
  - Check B_2: $0.7 + 0.5 = 1.2 > 1 \rightarrow$ Doesn't fit.
  - New bin $B_3 = [0.5]$
- **Item 4 (0.4):**
  - Check B_1: $0.8 + 0.4 = 1.2 > 1 \rightarrow$ Doesn't fit.
  - Check B_2: $0.7 + 0.4 = 1.1 > 1 \rightarrow$ Doesn't fit.
  - Check B_3: $0.5 + 0.4 = 0.9 \leq 1 \rightarrow$ Fits.
  - Update $B_3 = [0.5, 0.4]$
- **Item 5 (0.3):**
  - Check B_1: $0.8 + 0.3 = 1.1 > 1 \rightarrow$ Doesn't fit.
  - Check B_2: $0.7 + 0.3 = 1.0 \leq 1 \rightarrow$ Fits.
  - Update $B_2 = [0.7, 0.3]$
- **Item 6 (0.2):**
  - Check B_1: $0.8 + 0.2 = 1.0 \leq 1 \rightarrow$ Fits.

- Update $B_1 = [0.8, 0.2]$

- **Item 7 (0.1):**
  - Check B_1: $0.8 + 0.2 + 0.1 = 1.1 > 1 \rightarrow$ Doesn't fit.
  - Check B_2: $0.7 + 0.3 + 0.1 = 1.1 > 1 \rightarrow$ Doesn't fit.
  - Check B_3: $0.5 + 0.4 + 0.1 = 1.0 \leq 1 \rightarrow$ Fits.
  - Update $B_3 = [0.5, 0.4, 0.1]$

Final Packing (3 bins, Which is optimal also)

- $B_1 = [0.8, 0.2]$ (sum = 1.0)
- $B_2 = [0.7, 0.3]$ (sum = 1.0)
- $B_3 = [0.5, 0.4, 0.1]$ (sum = 1.0)

4.3.4 Complexity

- **Time Complexity**: $O(n \log n + n^2)$
  - $O(n \log n)$ for sorting
  - $O(n^2)$ for First Fit (each item may check all existing bins)
- **Space Complexity**: $O(n)$ to store items and bins

4.3.5 Approximation Ratio Proof

FFD achieves an approximation ratio of $\frac{11}{9}$ in the worst case. Here we prove a slightly looser bound of $\frac{3}{2}$ for clarity.

Proof. **Notation**:

- h = number of bins used by FFD
- OPT = optimal number of bins
- Partition items into 4 sets:

$$\begin{cases} A = \{a_i \mid a_i > \frac{2}{3}\} \\ B = \{a_i \mid \frac{1}{2} < a_i \leq \frac{2}{3}\} \\ C = \{a_i \mid \frac{1}{3} < a_i \leq \frac{1}{2}\} \\ D = \{a_i \mid a_i \leq \frac{1}{3}\} \end{cases}$$

Key Observations:

1. Each bin contains at most:
   - 1 item from A or B
   - 2 items from C
   - Any number from D
2. If a bin contains only D-items, it must be the last bin processed (due to sorting)
3. All non-last bins are $\geq \frac{2}{3}$ full (otherwise FFD would have placed more D-items in them)

Case Analysis:

- **Case 1**: Exactly one bin (B_h) contains only D-items

$$\frac{2}{3}(h-1) + c(B_h) \leq \sum a_i \leq \text{OPT}$$

where $c(B_h)$ = sum of items in last bin. This implies:

$$h \leq \frac{3}{2}\text{OPT} + 1$$

- **Case 2**: Multiple D-only bins
  - Same logic applies since all but last D-bin must be $\geq \frac{2}{3}$ full
- **Case 3**: No D-only bins
  - Maximum 2 items per bin
  - FFD matches optimal packing

Thus in all cases, $h \leq \frac{3}{2}\text{OPT} + 1$. □

4.4 Harmonic Algorithm

A major problem with online problems is that we do not know the future products and their sizes.There is a lot of uncertainty. To reduce it, the Harmonic algorithm provides a solution: categorize the items into groups based on their sizes, and assign each group a specific bin to store all items whose sizes fall within a specific range.

4.4.1 Description

- As this is an online algorithm, we do not know the sizes of future items. And to efficiently compute the number of bins, in the harmonic algorithm we divide the items into range of sizes in a harmonic way.
- We specify harmonic partitioning:

$$(0,1] = \bigcup_{k=1}^{M} I_k \quad \text{where} \quad I_k = \begin{cases} \left(\frac{1}{k+1}, \frac{1}{k}\right] & \text{for } 1 \leq k \leq M-1 \\ \left(0, \frac{1}{M}\right] & \text{for } k = M \end{cases}$$

- I_k represents the interval of sizes and k represents the max # of items possible in I_k interval.
- The value of M is choosen by the us and for the various value of M we get the different values of α
- We assign an active bin for all the intervals and during the online queries of items, whichever interval is fitted for the item if it is possible to put it into active bin for that interval then put it and continue the algorithm otherwise close this bin and increase the counter and assign a new empty active bin to that particular interval and put that item into the new bin.
- To store the number of bins used, we use the variables m_k, which indicates the number of closed bins for the k^{th} interval.

4.4.2 Algorithm

Algorithm 4 Harmonic Bin Packing

Initialize $m_k \leftarrow 0$ for all $k \in \{1, \ldots, M\}$ ▷ Bin counters
for each incoming item a_i **do**
    Find k such that $a_i \in I_k$ via binary search
    if $k \leq M - 1$ **then**
        if active I_k-bin has k items **then**
            $m_k \leftarrow m_k + 1$ ▷ Close full bin
            Open new active bin for I_k
        Place a_i in active I_k-bin
    else ▷ Handle I_M items
        if a_i fits in any active I_M-bin **then**
            Place a_i in first fitting bin
        else
            $m_M \leftarrow m_M + 1$
            Open new active bin for I_M
            Place a_i in new bin
return $\sum_{k=1}^{M} m_k$ ▷ Total bins used

- Here for I_1 to I_{M-1}, we check whether the active bins contain k items or not, basically we fixed that it contains k items at most for k=1 to M-1 inteval bins.
- And for k = M, we do it with a greedy approach and if it is possible to place the item into the I_1-bin whether it contains more than M items, we place the item into it. And if it violates the conditon of max-capacity of bin then we close the current bin and assign a new empty active bin for I_M interval.

4.4.3 Example

We are given the following items: {0.7, 0.3, 0.11, 0.4, 0.02, 0.24, 0.12, 0.23, 0.05}. And $M = 4$ is given. So the Harmonic algorithm divides items into 4 groups based on size:

- **Group** I_1: $(\frac{1}{2}, 1] \rightarrow$ max 1 item per bin
- **Group** I_2: $(\frac{1}{3}, \frac{1}{2}] \rightarrow$ max 2 items per bin
- **Group** I_3: $(\frac{1}{4}, \frac{1}{3}] \rightarrow$ max 3 items per bin
- **Group** I_4: $(0, \frac{1}{4}] \rightarrow$ packed greedily

Now we insert the items one by one and track the bin status:

1. Item 0.7 belongs to Z_1. A new bin is opened and closed after placing it. And $m_1 = 1$.
2. Item 0.3 belongs to Z_3. A new bin is started for Z_3 and the item is placed.
3. Item 0.11 belongs to Z_4. A new Z_4 bin is opened and the item is placed.
4. Item 0.4 belongs to Z_2. A new Z_2 bin is opened and the item is placed.
5. Item 0.02 belongs to Z_4. It is added to the existing Z_4 bin.
6. Item 0.24 belongs to Z_4. It is added to the existing Z_4 bin.

7. Item 0.12 belongs to Z_4. It is added to the same Z_4 bin.
8. Item 0.23 belongs to Z_4. The current Z_4 bin is full, so a new one is opened. And $m_4 = 1$.
9. Item 0.05 belongs to Z_4. It is added to the second Z_4 bin.

Final bin usage:

- Z_1 uses 1 bin: [0.7]
- Z_2 uses 1 bin: [0.4]
- Z_3 uses 1 bin: [0.3]
- Z_4 uses 2 bins: [0.11, 0.02, 0.24, 0.12], [0.23, 0.05]

So, total bins used by the Harmonic Algorithm = **5**

Optimal bin packing (offline):

- Bin 1: [0.7, 0.3] (sum = 1)
- Bin 2: [0.4, 0.24, 0.23, 0.12] (sum = 0.99)
- Bin 3: [0.11, 0.02, 0.05] (sum = 0.18)

Total bins used in the optimal solution = **3**

Approximation Factor:

$$\alpha = \frac{5}{3} = 1.67$$

4.4.4 Complexity

- Time complexity : $\mathcal{O}(n \log M + M)$
- As the initial for loop assigning zero to m_k variables takes $\mathcal{O}(M)$ time complexity. And the second loop take N iterations and in each iterations we do the $BinarySearch$ on the M active bins so total time complexity will be $\mathcal{O}(n \log M + M)$.
- Space Complexity : $\mathcal{O}(M)$

4.4.5 Approximation Ratio and Proof

Worst case analysis gives $\alpha = 1.6910...$

M	Worst-case performance bound for Harmonic$_M$
3	1.75
4	1.7222
5	1.7083
6	1.7
7	1.6944
8	1.6938
9	1.6934
10	1.6931
11	1.6928
12	1.6926
42	1.6910
⋮	⋮
∞	1.6910

Table 1: Worst-case performance bounds for Harmonic$_M$ for different values of M [6]

The detailed proof of the approximation factor for the Harmonic algorithm is quite extensive and may be difficult to grasp on a first reading. For this reason, we have chosen not to include it in this report. Reference we have taken for the the proof: [6]

4.5 LP based Approximation Algorithm

While heuristic algorithms like First Fit, Next Fit, and Harmonic are fast and easy to implement, but some of them do not always give solutions close to the optimal. To improve this, LP-based approximation algorithms use linear programming to first get a fractional solution, and then apply smart rounding techniques to get a final packing. These methods are more structured and provide better guarantees.

One well-known LP-based algorithm is the Karmarkar-Karp algorithm, which gives a solution within an additive $\mathcal{O}(\log^2 \mathrm{OPT})$ bins from the optimal with the Gilmore-Gomory LP Formulation as its foundation.

4.5.1 Gilmore-Gomory LP Formulation

Background: The LP-based formulation proposed by Gilmore and Gomory (1961, 1963) was originally designed for the cutting stock problem, which is closely related to bin packing. This formulation laid the foundation for LP relaxations in bin packing problems.

High-Level Idea: Instead of assigning items to bins directly, this approach considers all possible combinations of items that can fit into a single bin and assigns weights to these combinations (called patterns).

Notations:

- n: Number of items.
- s_i: Size of the i-th item, for $i = 1, 2, \ldots, n$.
- P: The set of all valid patterns, where each pattern is a set of items whose total size does not exceed 1, i.e., $\sum_{i \in p} s_i \leq 1$.
- Each pattern $p \in P$ corresponds to a column in the LP.

LP Formulation:

Variables: Let x_p denote the number of times pattern $p \in P$ is used. Note that x_p can be fractional in the LP relaxation.

Objective Function: Minimize the total number of bins used:

$$\min \sum_{p \in P} x_p$$

Constraints:

- Each item must be packed in at least one bin:

$$\sum_{p \in P} a_{ip} \cdot x_p \geq 1 \quad \text{for } i = 1, 2, \ldots, n$$

where,

$$a_{ip} = \begin{cases} 1, & \text{if item } i \text{ is in pattern } p \\ 0, & \text{otherwise} \end{cases}$$

- Non-negativity constraint:

$$x_p \geq 0 \quad \text{for all } p \in P$$

Challenge: The number of patterns $|P|$ is exponential in n.

Solution: Column Generation

- Start with a small subset of patterns.
- Solve the Restricted Master Problem (RMP) with these patterns.
- Use the dual prices to identify a new pattern by solving a knapsack problem with negative reduced cost.
- Add that pattern and repeat.

Result: We get a fractional solution (LP relaxation), which is a lower bound on the optimal number of bins (OPT).

4.5.2 Column Generation

- Here we initialize the LP relaxation with a subset of patterns $P' \subseteq P$ (Restricted Master Problem - RMP).

$$\begin{aligned} \text{minimize} \quad & \sum_{p \in P'} x_p \\ \text{subject to} \quad & \sum_{p \in P'} a_{ip} x_p \geq 1, \quad \forall i = 1, 2, \ldots, n \\ & x_p \geq 0 \end{aligned}$$

- Solve the above LP problem and let y_i be the dual variables associated with the ith constraint.
- **Pricing:**
  - Find a column $p \notin P$ that could reduce the objective function value.
  - Find the column with the most negative reduced cost by solving an associated knapsack problem in the dual variables.
  - If the problem finds such a column (pattern), then add the corresponding column to the RMP.
- Iterate until no column with negative reduced cost is found. (Optimal solution)

4.5.3 Karmarkar-Karp Approximation Algorithm

Goal: Use LP relaxation + rounding to build a near-optimal integral solution for the bin packing problem.

Key Ideas:

- Solve the Gilmore-Gomory LP to get OPT (fractional bins).
- Apply a rounding scheme to get an integer number of bins.

Algorithm:

Algorithm 5 Karmarkar-Karp Approximation Algorithm

Item Grouping Divide items into groups based on size ranges. Within each group, round all item sizes to the largest size in that group.

Solve LP on Rounded Items

- Use Gilmore-Gomory LP on simplified sizes (reduced item types).

Rounding to Integer

- Use rounding techniques to convert the fractional solution to an integer one.

4.5.4 Approximation factor and Proof

Let's take, SOL = number of bins used by KK approximation algorithm.

Claim: The solution we get from the Karmarkar-Karp Approximation Algorithm satisfies:

$$\text{SOL} \leq \text{OPT} + \mathcal{O}(\log^2 \text{OPT})$$

Proof: KK uses a *Linear Programming (LP) relaxation + rounding technique* to get an integral solution.

Steps:

1. Group items by size (into size classes).
2. Round item sizes within each group to the largest value of that group.
3. Solve LP for the rounded instance (much fewer item types).
4. Round LP solution back to integer bins using a technique that adds a small overhead.

Claim 1: After performing the grouping, only log(OPT) size classes can be non-empty.

Proof:

- We know that we have n items of size $s_i \in (0, 1]$.
- OPT is the minimum number of bins needed to pack all items.
- Each bin is of capacity 1, and the total volume

$$V = \sum_{i=1}^{n} s_i \leq \text{OPT}$$

- Now, according to the KK approximation algorithm, we divide items into groups by size ranges.

- We do the partition into geometric intervals as follows:

$$\left(\frac{1}{2}, 1\right], \left(\frac{1}{4}, \frac{1}{2}\right], \left(\frac{1}{8}, \frac{1}{4}\right], \ldots, \left(\frac{1}{2^k}, \frac{1}{2^{k-1}}\right], \ldots$$

  The k^{th} size class is:

$$\left(\frac{1}{2^k}, \frac{1}{2^{k-1}}\right]$$

- We do the grouping to round each item up to the **maximum size** in its class (i.e., to $\frac{1}{2^{k-1}}$ if item is in class k).

  It simplifies the LP with fewer variables because of the fewer item types.

- We know that each item s in class k satisfies

$$s \in \left(\frac{1}{2^k}, \frac{1}{2^{k-1}}\right] \Rightarrow s > \frac{1}{2^k} \Rightarrow \text{Every item in class } k \text{ has size at least } \frac{1}{2^k}.$$

- Let's assume that the k^{th} class has n_k items.

$$\text{Total volume of class } k \geq \frac{n_k}{2^k}$$

- Assume there are m non-empty size classes, and the classes that are non-empty are $k_1, k_2, \ldots, k_m$.

  Total volume from all these non-empty classes is, $\sum_{i=1}^{m} \frac{n_{k_i}}{2^{k_i}} \leq$ Total volume of all items $\leq$ OPT

- Let's take the worst case in which each size class has just 1 item.

$$\Rightarrow \text{Total volume from } m \text{ non-empty classes } \geq \sum_{i=1}^{m} \frac{1}{2^{k_i}}$$

- We want this to be $\leq$ OPT.

- Let's take a simpler **upper** bound.

  Assume that the non-empty classes are the first m classes.

$$k = 1, 2, \ldots, m$$

$$\Rightarrow \text{Total volume} = \sum_{k=1}^{m} \frac{1}{2^k} = 1 - \frac{1}{2^m}$$

- Now if we take $m = \log(\text{OPT})$ then the smallest volume from these classes is:

$$\sum_{k=1}^{\log(\text{OPT})} \frac{1}{2^k} = 1 - \frac{1}{\text{OPT}}$$

$$\Rightarrow \text{Total volume} \geq 1 - \frac{1}{\text{OPT}}$$

- Therefore to keep total volume $\leq$ OPT, we cannot have more than $\mathcal{O}(\log \text{OPT})$ non-empty classes.

Claim 2: Rounding of the item sizes into representative values (highest value) during KK introduces at most an additive overhead of $O(\log \mathrm{OPT})$ bins.

Proof: We want to show that the total extra volume increased due to rounding to the highest value can be packed using $\mathcal{O}(\log \mathrm{OPT})$ extra bins.

- We know that the class k contains items with size $\in \left(\frac{1}{2^k}, \frac{1}{2^{k-1}}\right]$
- Maximum size possible in class $k = \frac{1}{2^k}$
- So, each item in class k will be rounded up to $\frac{1}{2^{k-1}}$
- For each item in class k,
$$\frac{1}{2^k} < \text{size} \leq \frac{1}{2^{k-1}}$$
So, maximum increase in size per item is
$$\Delta_k = \frac{1}{2^{k-1}} - \text{size} \leq \frac{1}{2^{k-1}} - \frac{1}{2^k} = \frac{1}{2^k}$$
- Then total added volume due to rounding for class k is at most
$$n_k \cdot \frac{1}{2^k}$$
- Total added volume over all classes:
$$\sum_{k=1}^{m} n_k \cdot \frac{1}{2^k}$$
- Since each bin can hold 1 unit of volume, so number of extra bins needed to pack this extra volume is at most:
$$\text{Extra bins} \leq \sum_{k=1}^{m} \frac{n_k}{2^k}$$
- We know that $\sum_{k=1}^{m} n_k = n$
$$\sum_{k=1}^{m} \frac{n_k}{2^k} \leq n \sum_{k=1}^{m} \frac{1}{2^k}$$
$$\leq n \left(1 - \frac{1}{2^m}\right) < n$$
- We now observe that we have only $m = O(\log \mathrm{OPT})$ non-empty classes, and in each class, total added volume is bounded. So, total overhead from all m classes is at most m bins, because in each class, all rounded-up items can be packed into ≤ 1 bin more than before.
- So, we need only $O(\log(\mathrm{OPT}))$ extra bins.

Conclusion:

Claim 1: Number of non-empty size-classes $= O(\log \mathrm{OPT})$

- Items are grouped into size classes based on their sizes (i.e., $\left(\frac{1}{2^k}, \frac{1}{2^{k-1}}\right]$)

- The total volume of items in each class is at least $\frac{1}{2^k}$ and since all items fit in OPT bins, the number of such non-empty size classes is at most $O(\log \text{OPT})$ because we can't have too many size classes with meaningful total volume, else we would exceed the total OPT volume budget.

Claim 2: Rounding overhead = $O(\log \text{OPT})$

- In each size class, all item sizes are rounded up to the largest possible value in that class to simplify the LP formulation.
- This rounding may increase the total volume slightly, but since we only round within $O(\log \text{OPT})$ size classes, the total number of extra bins required is at most $O(\log \text{OPT})$ because each class contributes at most 1 extra bin due to rounding. So, total overhead is additive and logarithmic.

Hence, after grouping and rounding, we can write that the solution we get from KK approximation algorithm is:

$$\text{SOL} \leq \text{OPT} + O(\log^2(\text{OPT}))$$

4.5.5 Example

We are given the following items: {0.55, 0.53, 0.45, 0.43, 0.31, 0.29, 0.21, 0.19, 0.11, 0.09} and Bin capacity = 1

- **Step 1:** Group Items into Geometric Size Classes

Class Index k	**Range**	**Items in Class**	**Rounded Size $\rightarrow \frac{1}{2^{k-1}}$**
1	$(\frac{1}{2}, 1] = (0.5, 1]$	0.55, 0.53	1
2	$(\frac{1}{4}, \frac{1}{2}] = (0.25, 0.5]$	0.45, 0.43, 0.31, 0.29	0.5
3	$(\frac{1}{8}, \frac{1}{4}] = (0.125, 0.25]$	0.21, 0.19	0.25
4	$(\frac{1}{16}, \frac{1}{8}] = (0.0625, 0.125]$	0.11, 0.09	0.125

Table 2: Grouping items into geometric size classes.

- **Step 2:** Round Each Item to Class Max Size
  Rounded Items: {1, 1, 0.5, 0.5, 0.5, 0.5, 0.25, 0.25, 0.125, 0.125}
- **Step 3:** Solve LP for the Simplified Item Sizes
  Optimal Bin Packing for Simplified Item Sizes

  1. Bin 1: 1
  2. Bin 2: 1
  3. Bin 3: 0.5 + 0.5 = 1
  4. Bin 4: 0.5 + 0.5 = 1
  5. Bin 5: 0.25 + 0.25 + 0.125 + 0.125 = 0.75

  Total bins used = 5

Key Observations:

- Rounding increased total volume slightly but helped reduce item types.
- Overhead due to rounding was minimal (only 1–2 extra items spread).
- Number of non-empty size classes = 4 $\rightarrow$ within $\mathcal{O}(\log \mathrm{OPT})$.
- Number of extra bins needed to pack simplified items $\rightarrow$ within $\mathcal{O}(\log \mathrm{OPT})$.

5 Limits on Approximation for Bin Packing

Unless $\mathbf{P} = \mathbf{NP}$, there exists no polynomial-time approximation algorithm for the bin packing problem with an approximation factor strictly better than $\frac{3}{2}$.

Proof. We prove this by reduction from the **NP**-complete PARTITION problem:
Reduction Construction:

1. Given an instance of PARTITION with set $S = \{b_1, b_2, \ldots, b_n\}$ where $\sum_{i=1}^{n} b_i = B$ (even), create a bin packing instance:
$$a_i = \frac{2b_i}{B} \quad \text{for each } i$$
2. Each $a_i \in (0, 1]$, and $\sum_{i=1}^{n} a_i = 2$ since:
$$\sum_{i=1}^{n} a_i = \frac{2}{B} \sum_{i=1}^{n} b_i = \frac{2B}{B} = 2$$

Key Observations:

- If S is partitionable, then $\exists T_1, T_2$ such that:
$$\sum_{b_i \in T_1} b_i = \sum_{b_j \in T_2} b_j = \frac{B}{2}$$
This corresponds to a bin packing with:
$$\sum_{a_i \in T_1} a_i = \sum_{a_j \in T_2} a_j = 1$$
Thus, OPT = 2 (exactly two bins).
- If S is not partitionable, any valid packing requires at least 3 bins, so OPT ≥ 3.

Contradiction Argument: Suppose there exists a polynomial-time approximation algorithm $\mathcal{A}$ with factor $\alpha < \frac{3}{2}$.

- For partitionable instances:
$$\mathcal{A}(I) \leq \alpha \cdot \mathrm{OPT} < \frac{3}{2} \times 2 = 3$$
Since the number of bins must be integer, $\mathcal{A}(I) \leq 2$.
- For non-partitionable instances:
$$\mathcal{A}(I) > \mathrm{OPT}$$
$$\mathcal{A}(I) \geq 3$$

Thus, $\mathcal{A}$ can decide the PARTITION problem in polynomial time by:

1. Converting the input to a bin packing instance
2. Running $\mathcal{A}$
3. Answering "YES" if $\mathcal{A}(I) \leq 2$, "NO" otherwise

This would imply $\mathbf{P} = \mathbf{NP}$, contradicting the widely believed conjecture that $\mathbf{P} \neq \mathbf{NP}$. □

The $\frac{3}{2}$ approximation barrier is tight:

- First Fit Decreasing achieves $\frac{11}{9} \approx 1.222$ in the worst case
- No polynomial-time algorithm can guarantee better than 1.5 unless $\mathbf{P} = \mathbf{NP}$

6 Comparison of Algorithms

6.1 Approximation Ratios

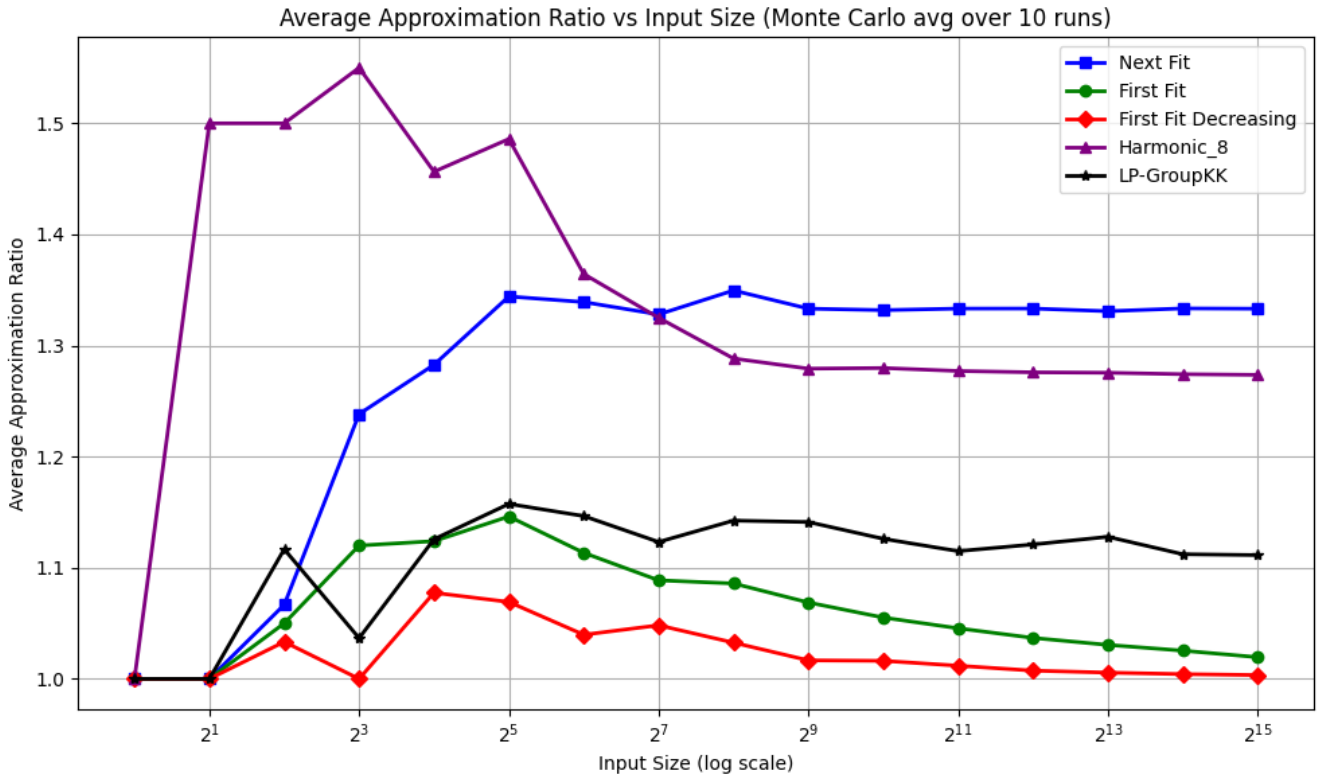

Figure 1: Comparison of Average Approximation Ratios of Bin Packing Algorithms Across Varying Input Sizes (Log Scale)

- **First Fit Decreasing (FFD)** consistently achieves the lowest approximation ratio, approaching 1 as input size increases, making it the most efficient algorithm overall.
- **Next Fit (NF)** performs the worst among all, stabilizing around a ratio of 1.33 for large inputs, indicating high inefficiency.
- **Harmonic_8** and **LP-GroupKK** show moderate performance; Harmonic_8 has high initial ratios that plateau near 1.27, while LP-GroupKK remains fairly stable around 1.11.

6.2 Average Bin Utilization

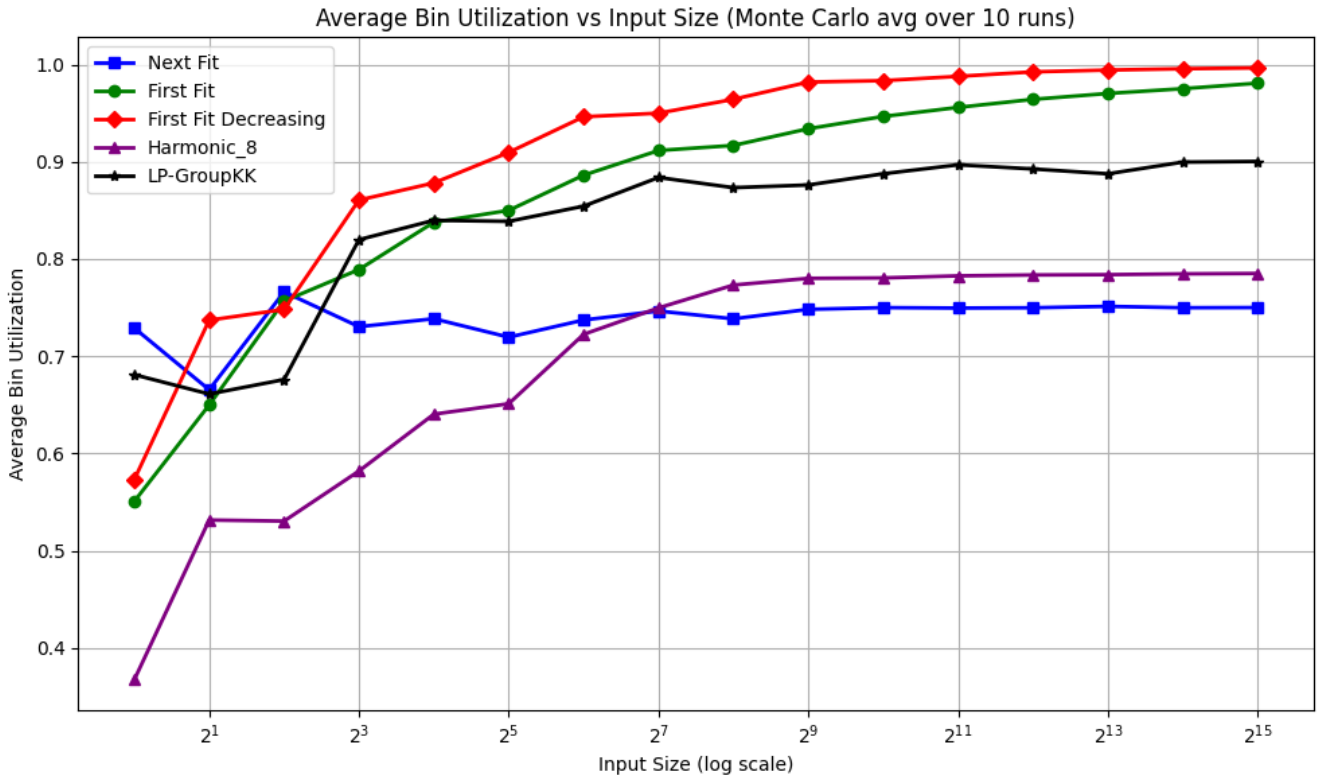

Figure 2: Average Bin Utilization vs Input Size (log scale) for various bin packing algorithms based on Monte Carlo averages over 10 runs. Higher bin utilization implies more efficient space usage.

- **First Fit Decreasing (FFD)** achieves the highest average bin utilization, approaching 1 as input size increases, indicating optimal packing behavior.
- **First Fit (FF)** also performs well with utilization consistently improving with input size, though it stays slightly below FFD.
- **LP-GroupKK** maintains high bin utilization, but lags behind FF and FFD at larger input sizes.
- **Next Fit (NF)** shows relatively low and stagnant bin utilization, demonstrating inefficiency especially for larger inputs.
- **Harmonic_8** has the lowest bin utilization across all input sizes, making it less suitable for space-efficient packing.

6.3 Average Waste per Bin

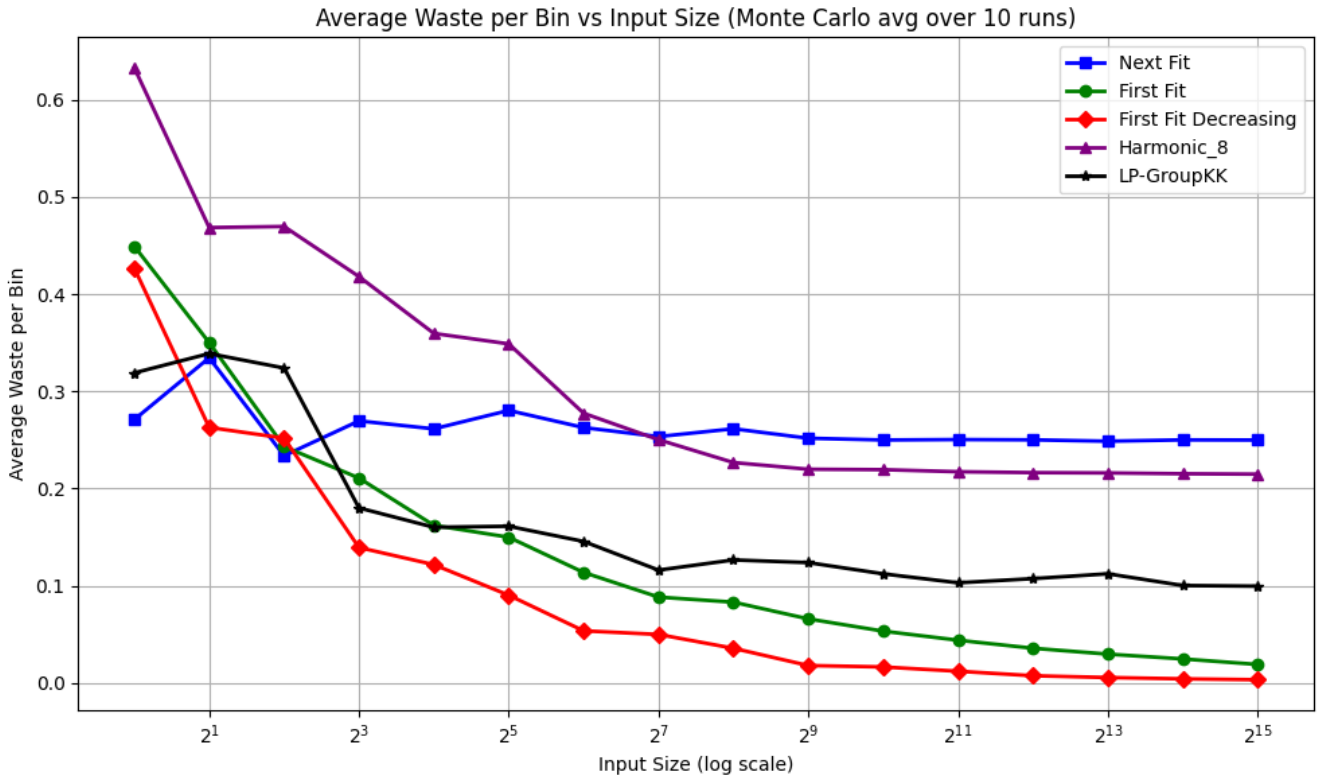

Figure 3: Average Waste per Bin vs Input Size (log scale) for different bin packing algorithms based on Monte Carlo averages over 10 runs. Lower waste indicates better bin space efficiency.

- **First Fit Decreasing (FFD)** achieves the lowest average waste across all input sizes, with waste approaching zero for large inputs, indicating excellent space efficiency.
- **First Fit (FF)** also performs well, with steadily decreasing waste as input size increases, though it remains slightly higher than FFD.
- **LP-GroupKK** shows moderately low waste, outperforming Next Fit and Harmonic_8, but not as efficient as FF or FFD.
- **Next Fit (NF)** maintains a relatively constant waste, unable to adapt well with larger inputs, and is outperformed by FF, FFD, and LP-GroupKK.
- **Harmonic_8** exhibits the highest average waste per bin for all input sizes, making it the least efficient among the five in terms of bin utilization.

6.4 Average Number of Bins

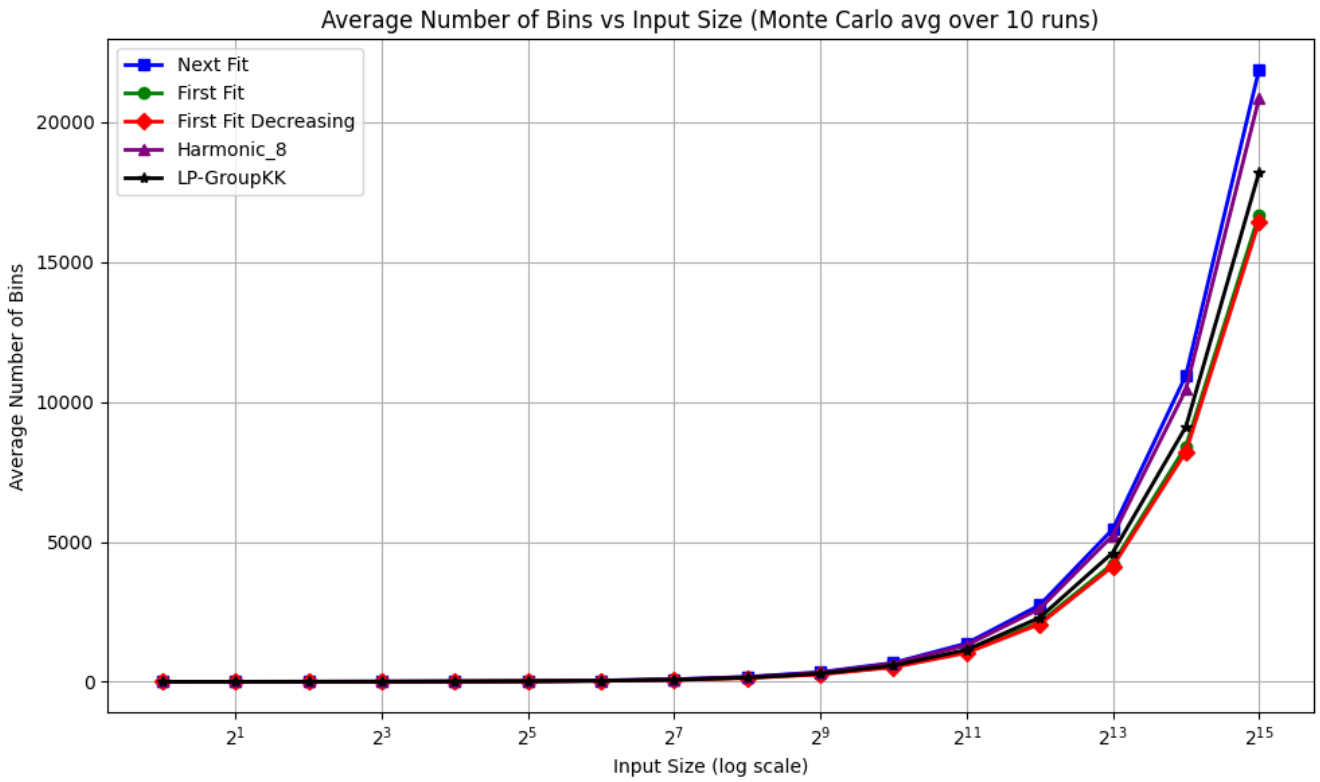

Figure 4: Average Number of Bins vs Input Size (log scale) for different bin packing algorithms, averaged over 10 Monte Carlo simulations. Lower values indicate better packing efficiency.

- **First Fit Decreasing (FFD)** consistently uses the fewest number of bins, showing its superior efficiency in minimizing the number of bins required.
- **First Fit (FF)** closely follows FFD, performing well across all input sizes with only slightly more bins required.
- **LP-GroupKK** shows moderate performance, staying between FF and Harmonic_8, and improving with larger inputs.
- **Harmonic_8** requires more bins than FFD and FF, especially as input size grows, indicating weaker packing density.
- **Next Fit (NF)** performs the worst in terms of bin usage, needing the highest number of bins consistently across all input sizes.

6.5 Bin Fill Distribution

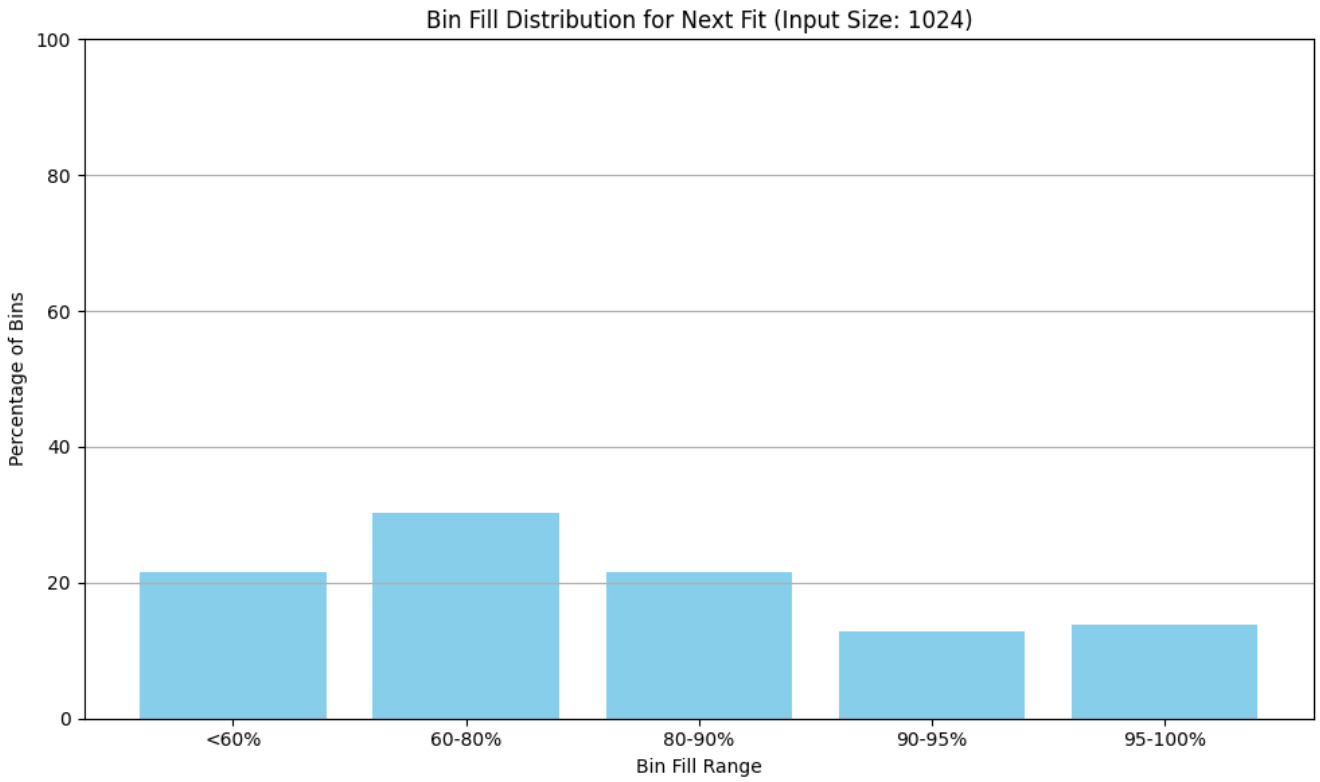

Figure 5: Bin fill distribution for the **Next Fit** algorithm with an input size of 1024.

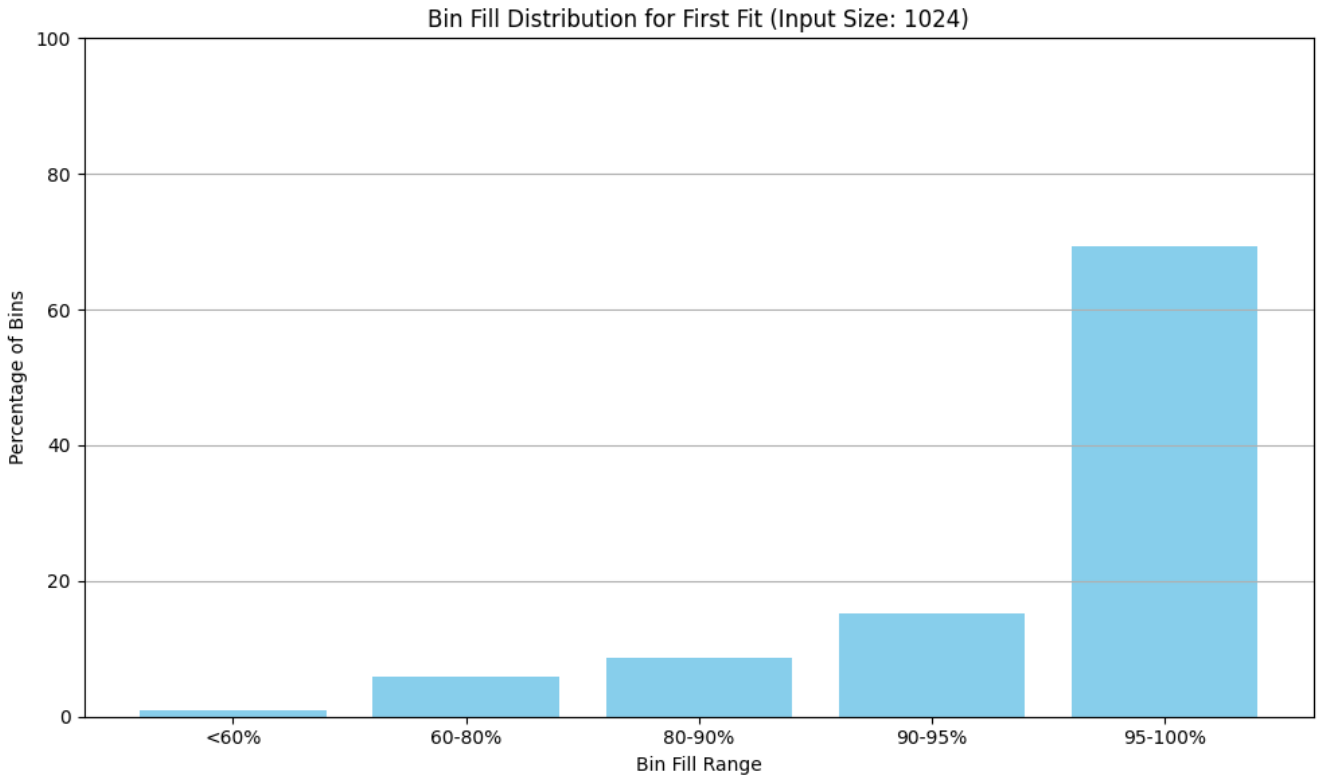

Figure 6: Bin fill distribution for the **First Fit** algorithm with an input size of 1024.

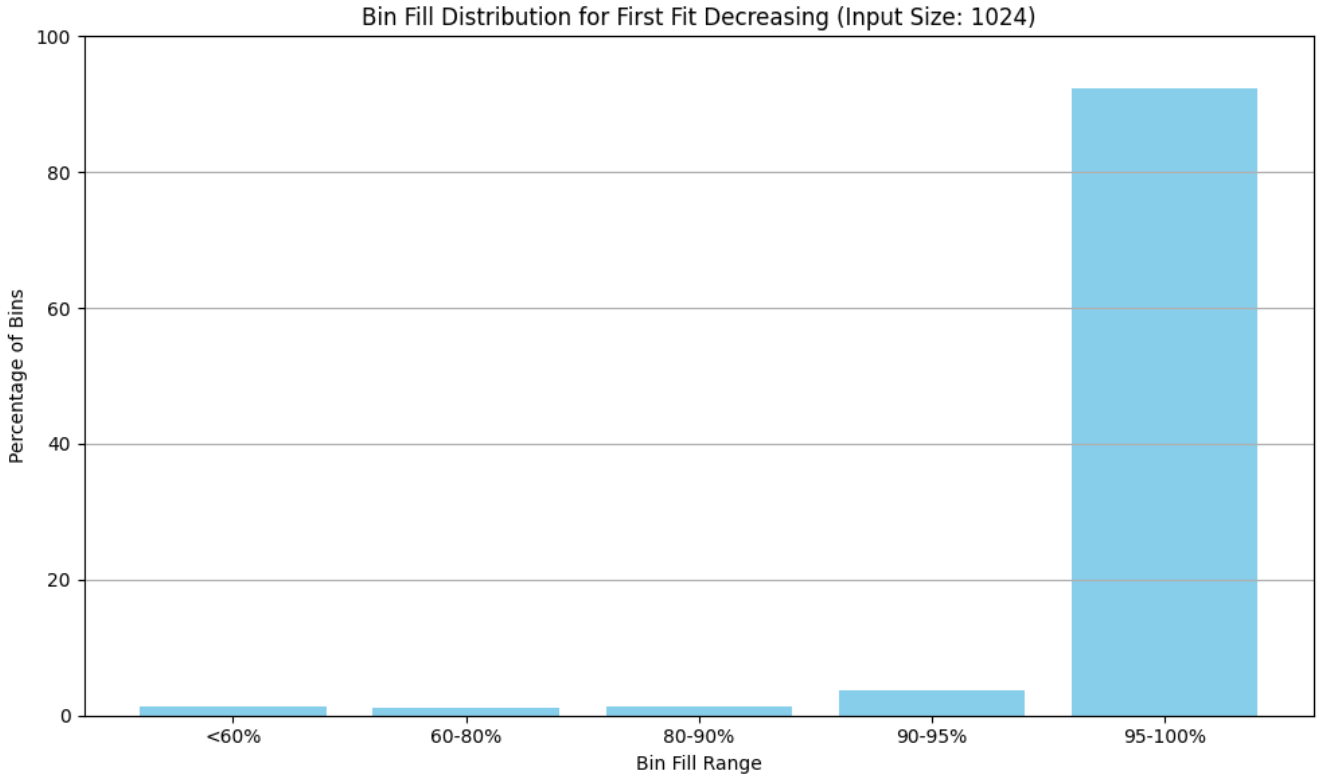

Figure 7: Bin fill distribution for the **First Fit Decreasing** algorithm with an input size of 1024.

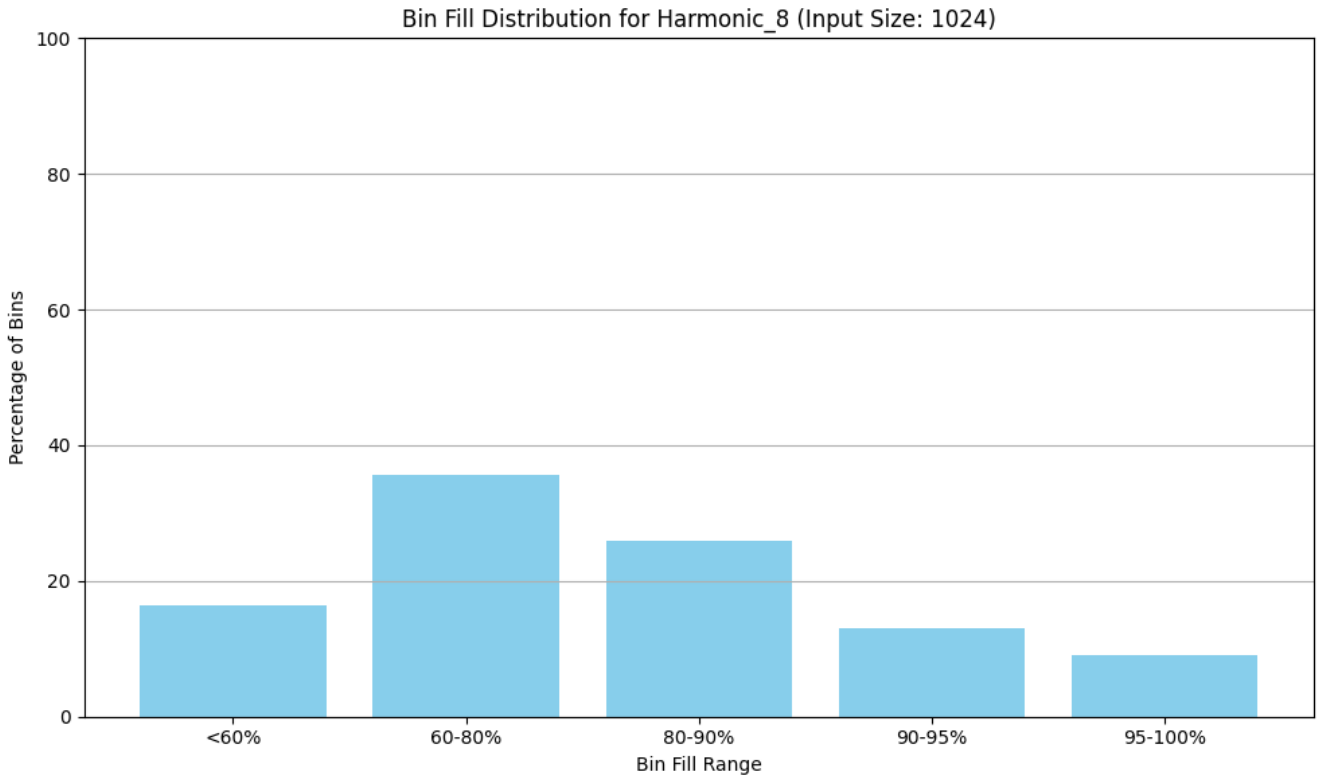

Figure 8: Bin fill distribution for the **Harmonic_8** algorithm with an input size of 1024.

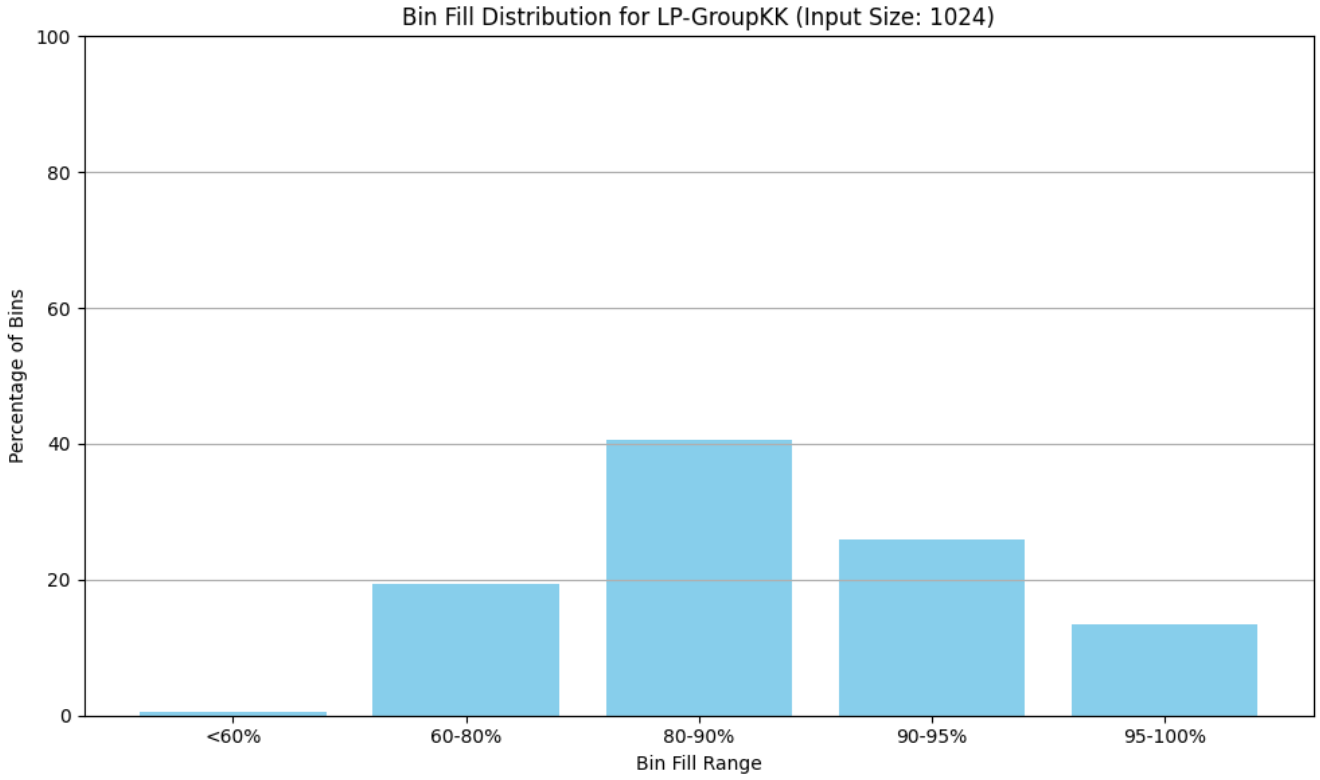

Figure 9: Bin fill distribution for the **LP-GroupKK** algorithm with an input size of 1024.

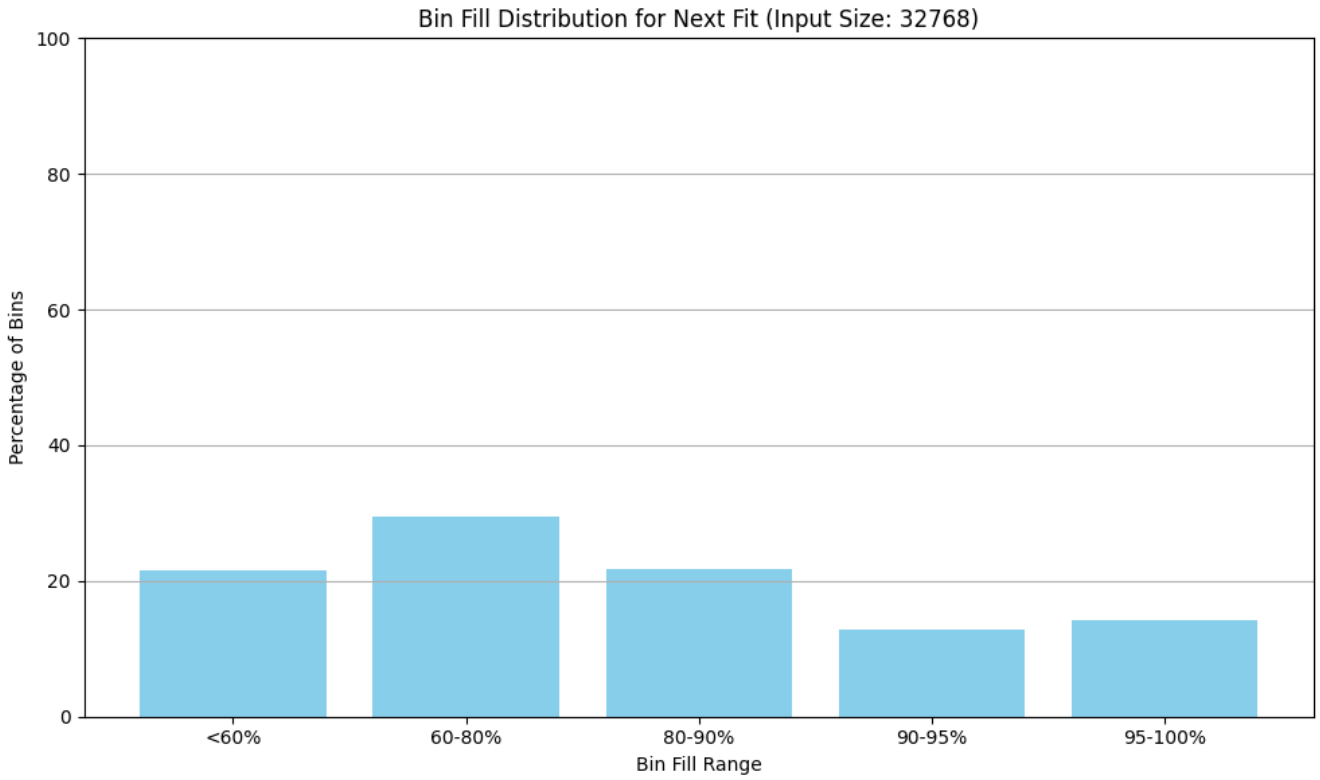

Figure 10: Bin fill distribution for the **Next Fit** algorithm with an input size of 32768.

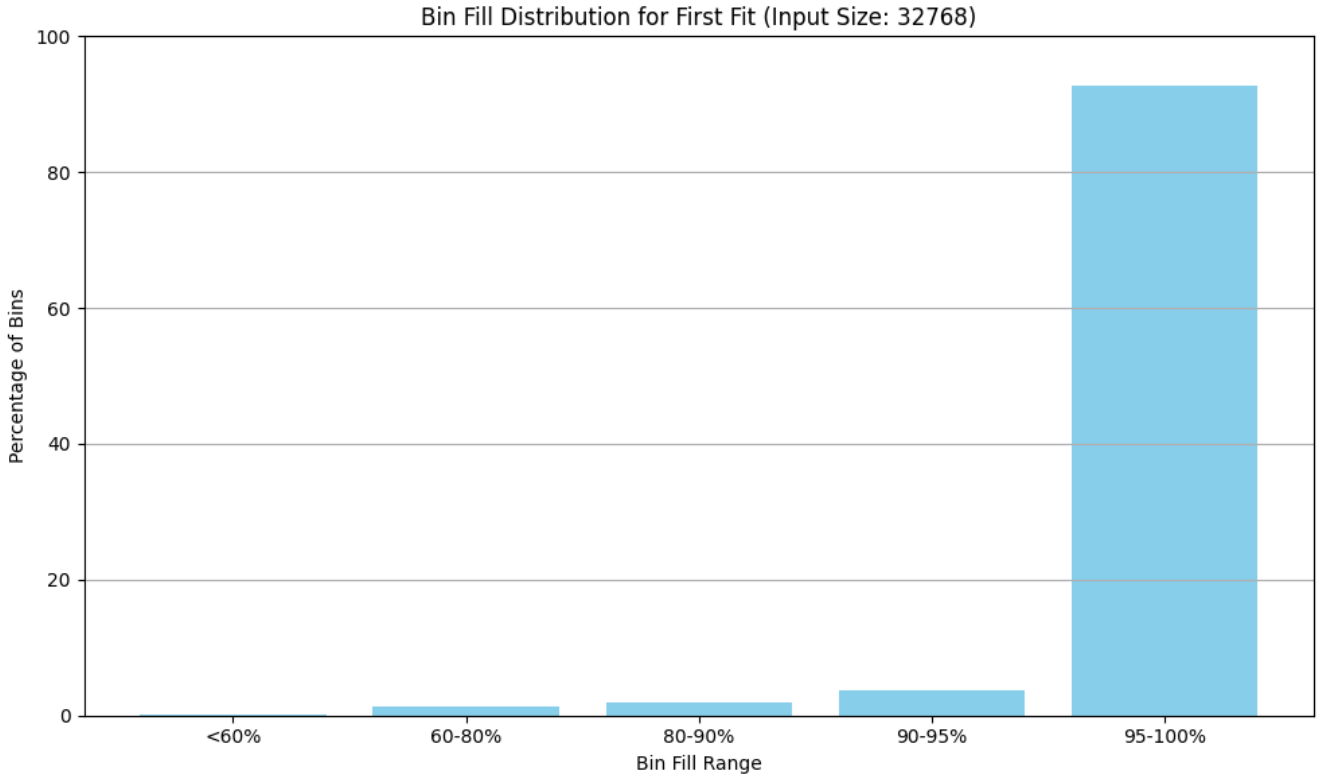

Figure 11: Bin fill distribution for the **First Fit** algorithm with an input size of 32768.

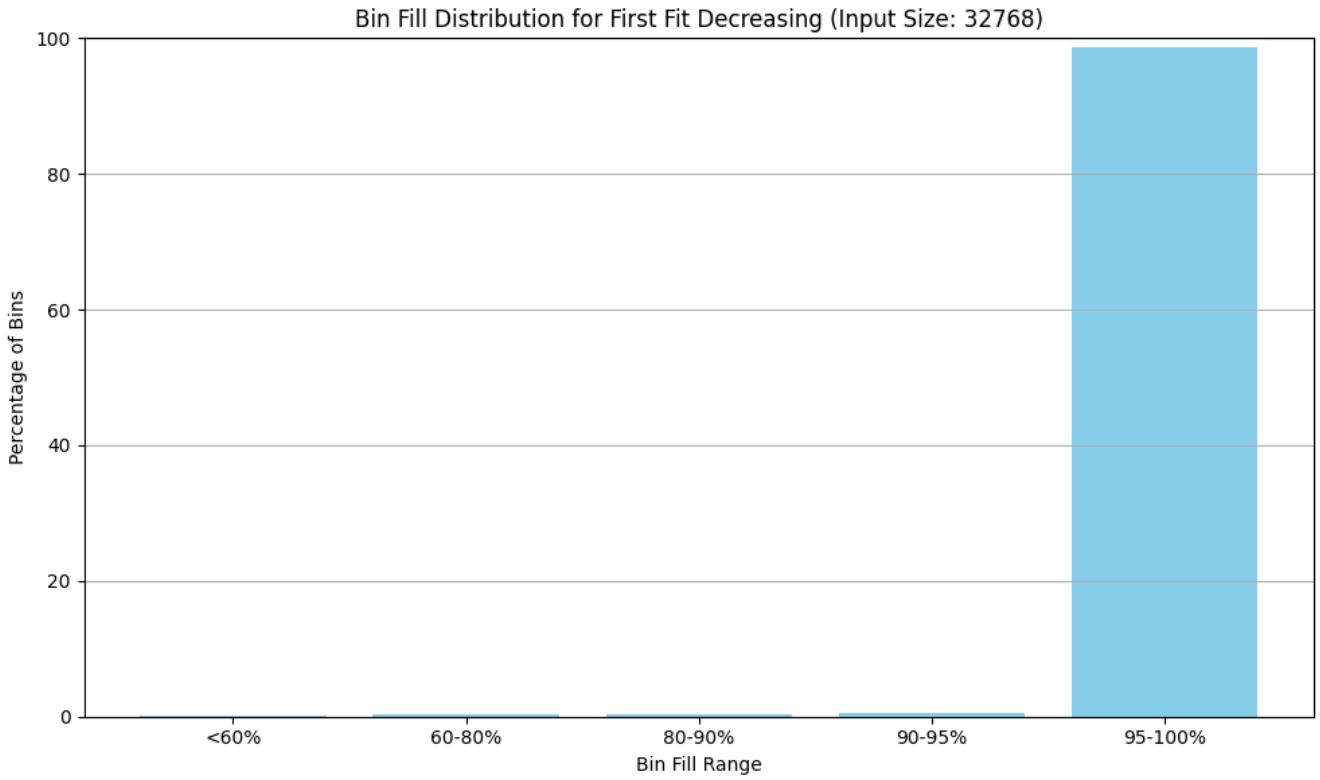

Figure 12: Bin fill distribution for the **First Fit Decreasing** algorithm with an input size of 32768.

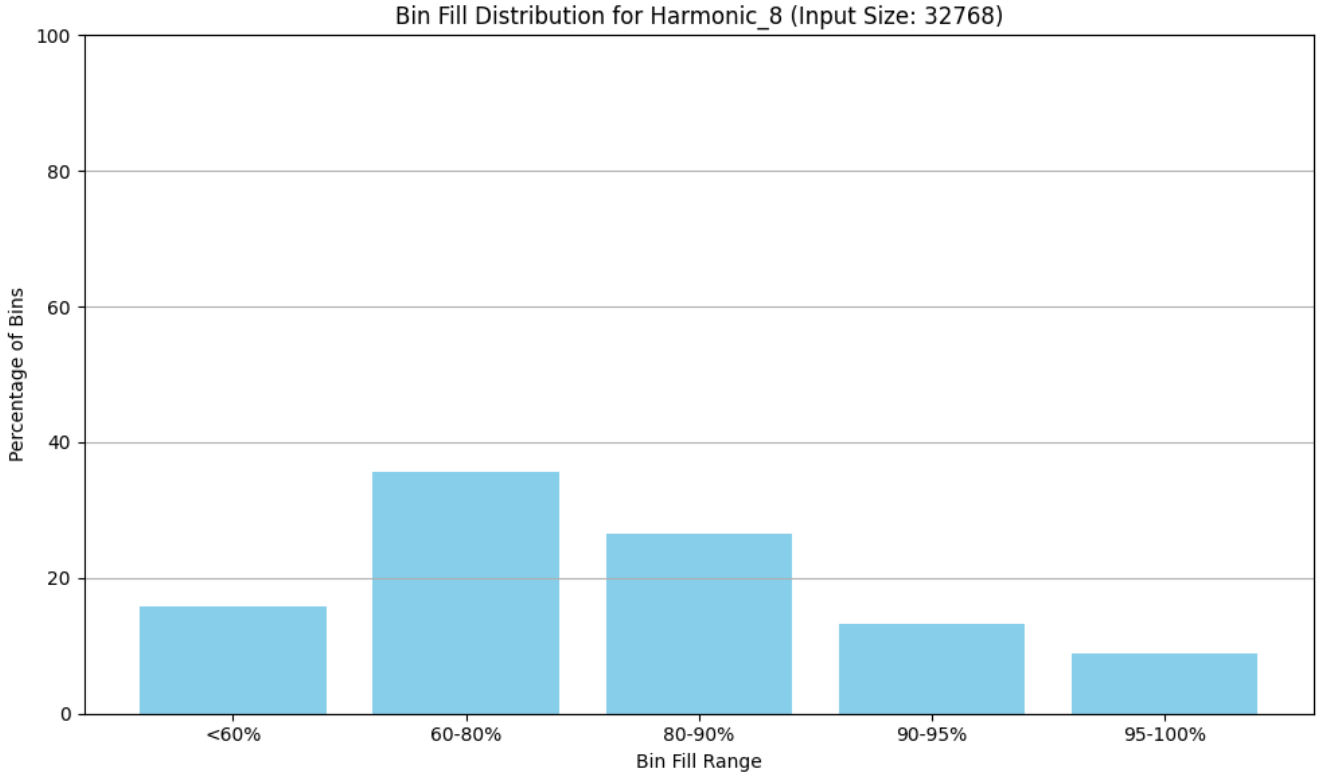

Figure 13: Bin fill distribution for the **Harmonic_8** algorithm with an input size of 32768.

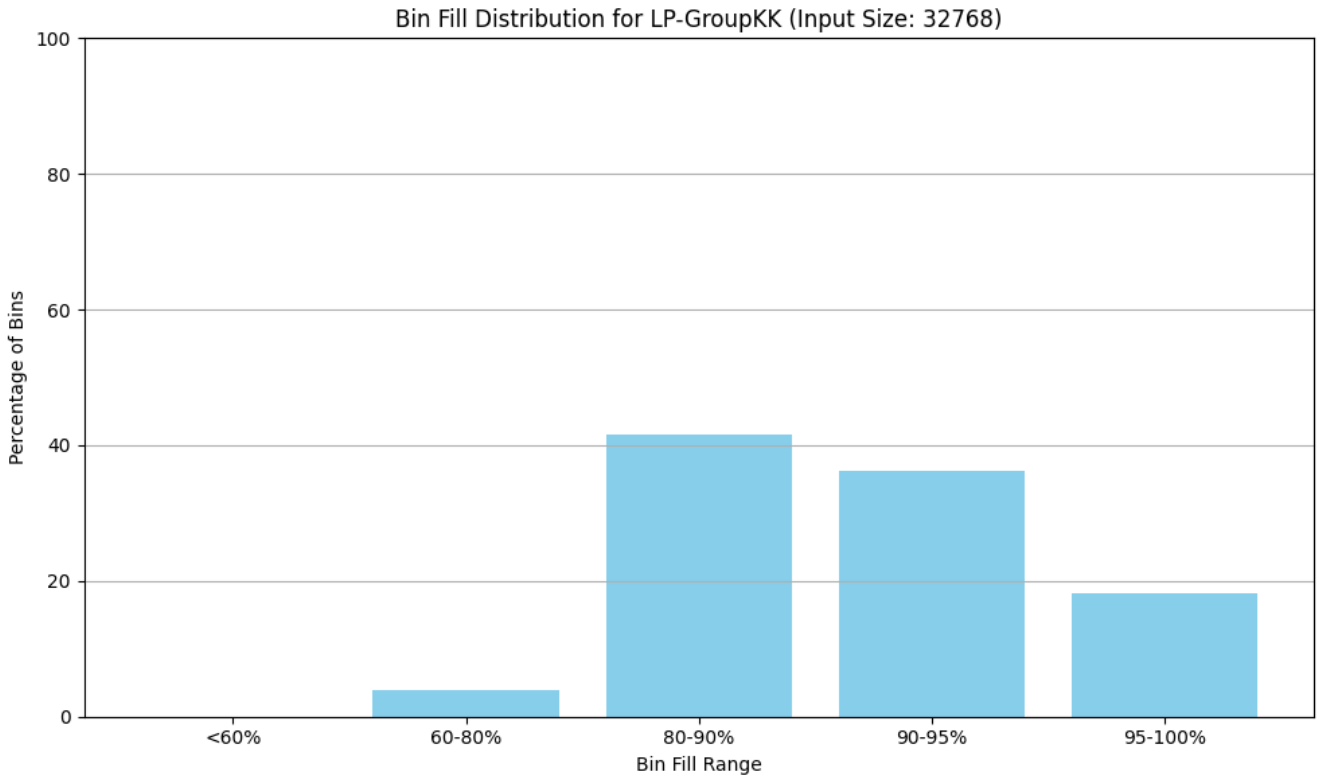

Figure 14: Bin fill distribution for the **LP-GroupKK** algorithm with an input size of 32768.

- **Next Fit (NF)**: Poor utilization with many bins $< 60\%$ full.
- **First Fit (FF)**:Moderate efficiency with bins typically $60\% - 90\%$ full.
- **First Fit Decreasing (FFD)**:High efficiency with most bins $90\% - 100\%$ full.
- **Harmonic_8**:Balanced distribution with bins clustered in $60\% - 95\%$ range.
- **LP-GroupKK**: Give packing with bins $80\% - 100\%$ full.
- As the total number of input items increases, all algorithms tend to perform better in terms of space utilization. Although the absolute number of bins used increases with the input size, each bin becomes more tightly packed, leading to a reduction in average waste per bin. This trend reflects the improved efficiency of the algorithms at scale, as larger instances allow better opportunities for optimal item placements.

6.6 Average time taken

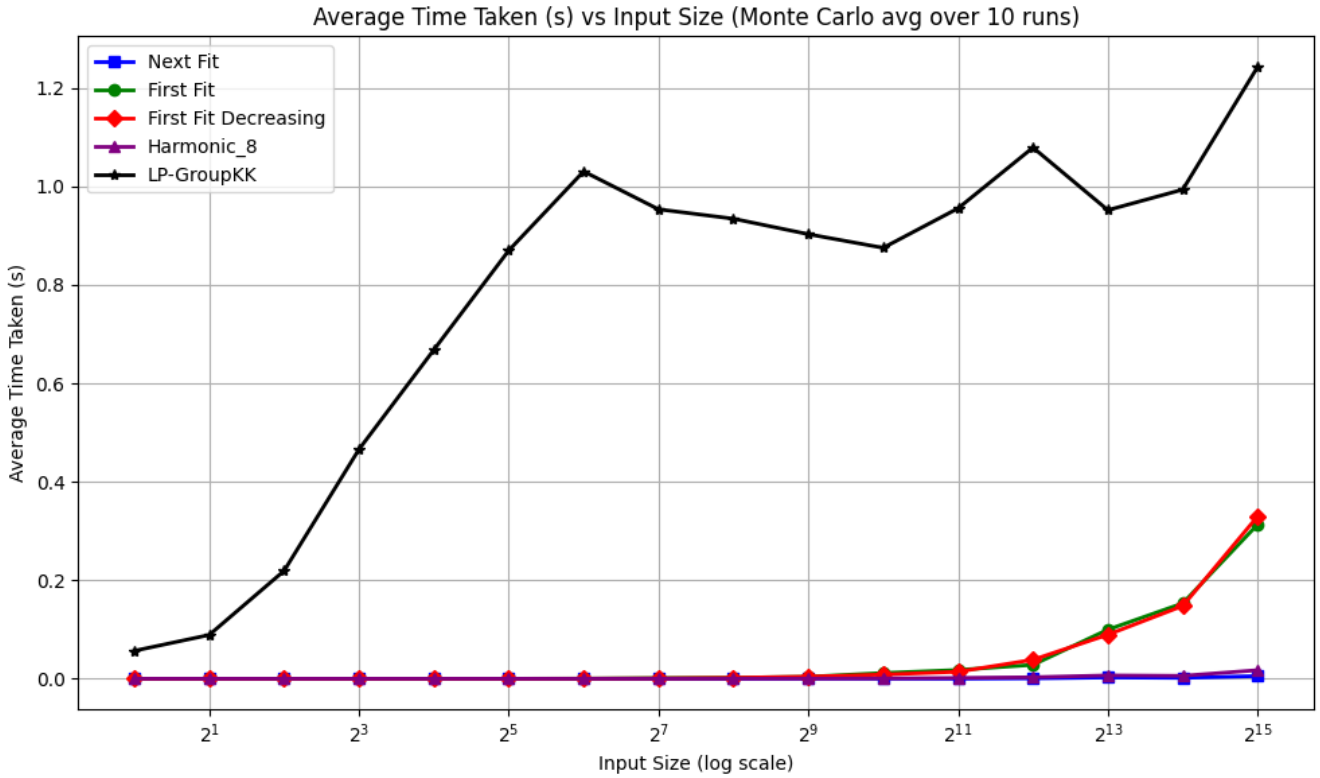

Figure 15: Comparison of average time taken (in seconds) by different bin packing algorithms as input size(log scale) increases. Results are averaged over 10 Monte Carlo runs.

- **Time Efficiency:** `Next Fit` and `First Fit` show the fastest execution times across all input sizes, while `LP-GroupKK` and `Harmonic_8` exhibit significantly slower performance as input size increases.

7 Conclusion

Our analysis of bin packing approximation algorithms yielded several key findings:

- **Best Performance:**
  - **First Fit Decreasing (FFD)** achieved optimal space utilization with the fewest bins
  - **First Fit (FF)** provided good online performance with slightly higher bin counts
- **Worst Performance:**
  - **Next Fit (NF)** showed the poorest space efficiency despite simple implementation
- **Trade-offs:**
  - **Harmonic** balanced online operation with moderate efficiency
  - **LP-based (Karmarkar-Karp)** offered theoretical guarantees at higher computational cost

Key Theoretical Insight: No polynomial-time algorithm can guarantee better than $1.5\times$ optimal packing unless $\mathbf{P} = \mathbf{NP}$.

Practical Recommendations

- Use **FFD** for offline scenarios where pre-sorting is possible
- Choose **FF** for online applications requiring fast decisions
- Avoid **NF** except in extremely resource-constrained environments

Real-World Applications of Bin Packing Algorithms

- **Container Loading:** Optimizing how goods are packed into shipping containers to minimize space and reduce costs
- **Memory Allocation:** Managing computer memory by allocating processes to memory blocks (similar to items in bins)
- **Cloud Computing:** Assigning virtual machines to physical servers to optimize resource utilization
- **Data Storage:** Organizing files on hard drives or SSDs to minimize wasted storage space
- **Stock Cutting:** Minimizing material waste when cutting raw materials (wood, metal, fabric) into smaller pieces
- **Production Scheduling:** Assigning tasks to machines or workers with limited capacity
- **Event Planning:** Arranging tables and chairs in event spaces
- **Advertising:** Packing ads into limited advertising space (TV/radio commercials, newspaper columns)
- **Healthcare:** Scheduling surgeries in operating rooms with limited time slots

Open Problem:

Does any efficient algorithm exist that packs all items in OPT + 1 bins ?

References

[1] Williams, D. P. and Shmoys, D. B. *The Design of Approximation Algorithms.* Cambridge University Press, 2010.

[2] Johnson, D. S. Near-optimal bin packing algorithms. Technical Report MAC TR-109, Project MAC, MIT, Cambridge, 1972.

[3] Baker, B. S. A new proof for the first fit decreasing algorithm. Technical Report, Bell Laboratories, Murray Hill, 1983.

[4] E.G. Coffman, M.R. Garey, and D.S. Johnson. *Approximation Algorithms for Bin Packing: A Survey.* In Approximation Algorithms for NP-Hard Problems, PWS Publishing, 1997.

[5] V. Vazirani, *Approximation Algorithms*, Springer, 2001.

[6] C. C. LEE and D. T. LEE *A Simple On-Line Bin-Packing Algorithm.* In Journal of the Association for Computing Machinery, Volume 32, Issue 3, 1st July 1985

[7] Guo, L. *Approximation Algorithms: Lecture Notes - Bin Packing.* (Lecture 13), West Virginia University.

Contributions

This project was collaboratively completed, and the following topic-wise contributions were made:

- **Problem Statement and Mathematical Formulation**: All
- **Proof of NP-Completeness:** Dishank
- **Approximation Algorithms:**
  - Next Fit (NF): Zeel
  - First Fit (FF): Dishank
  - First Fit Decreasing (FFD): Mihir
  - Harmonic Algorithm: Smit
  - LP-based Approximation (Gilmore-Gomory + Karmarkar-Karp): Dishant
- **Limits on Approximation:** Zeel, Smit
- **Comparison of Algorithms:** Mihir, Dishant
- **Report Writing:** All