



**YEDİTEPE UNIVERSITY**

Faculty of Engineering  
Department of Computer Engineering

# **Term Project Report**

CSE480/591: Optimization with Metaheuristics

**Student Name:** Barış Can Sertkaya / Erhan Önalı

**Student ID:** 20210702022 / 20210702136

**Date:** 30.10.2025

**Project Title:** 1D Bin Packing Problem

**Instructor:** Asst. Prof. Dr. Gizem Sng Terci

Fall 2025 Semester

## 1. Selected Problem and Motivation

This project focuses on the One-Dimensional Bin Packing Problem, abbreviated as 1D-BPP, a well-known NP-hard combinatorial optimization problem. It models many real-world tasks such as minimizing storage space, container loading, or memory-allocation optimization. The objective is to assign a set of items with given sizes into the fewest possible fixed-capacity bins without exceeding any capacity limit.

The 1D Bin Packing Problem is chosen because it has a clear mathematical structure yet allows creative metaheuristic exploration. It is also a classic optimization problem that is easy to visualize and test, which makes it ideal for experimenting with different algorithms and comparing their performance.

From a student perspective, it offers a good balance between implementation difficulty and conceptual clarity. It is complex enough to be interesting, but still manageable to code, analyze, and present meaningful results within the project timeline.

## 2. Formal Problem Definition

In the 1D Bin Packing Problem, we decide which bin each item will be placed into. Each item must be assigned to exactly one bin, and no bin capacity can be exceeded.

### 2.1 Decision Variables

Let  $x_{ij} = 1$  if item  $i$  is placed in bin  $j$ , and 0 otherwise. Here:

- $i = 1, 2, \dots, n$  represents the items,
- $j = 1, 2, \dots, m$  represents the bins;  $m$  is an upper bound on the number of bins.

Let  $y_j = 1$  if bin  $j$  is used, and 0 otherwise.

### 2.2 Objective Function

The main objective is to minimize the total number of bins used:

$$\min \sum_{j=1}^m y_j$$

In the metaheuristic implementation, a commonly used fitness function is:

$$f = B + \alpha \times \text{capacity violation}$$

where  $B$  is the number of bins used and  $\alpha$  is a penalty coefficient discouraging infeasible solutions. In our implementation, the decoder always constructs **feasible** packings (no capacity violation), so the violation term is always zero. Therefore, the effective

objective is minimizing  $B$ , with unused capacity used only as a tie-breaker as described in Section 5.

## 2.3 Constraints

1. **Assignment constraint:** Each item must be placed in exactly one bin.

$$\sum_{j=1}^m x_{ij} = 1 \quad \forall i$$

2. **Capacity constraint:** The total size of items in a bin cannot exceed its capacity  $C$ .

$$\sum_{i=1}^n s_i x_{ij} \leq C \quad \forall j$$

3. **Bin usage constraint:** If a bin is not used, no items can be assigned to it.

$$x_{ij} \leq y_j \quad \forall i, j$$

## 3. Dataset or Benchmark Instances

**Benchmark Instances:** The Falkenauer dataset (Falkenauer, 1996) from the OR-Library (Beasley, 1990) is a standard benchmark for 1D bin packing.

**Instances used in Phase 3 experiments:**

- **TP2 example instance:** 7 items, capacity 60, defined in Section 4 and used for correctness verification.
- **OR-Library / BinPack benchmark files:** each file contains 20 instances. We used `binpack2.txt` with instances named `u250_*`, `binpack4.txt` with `u1000_*`, `binpack7.txt` with `t249_*`, and `binpack8.txt` with `t501_*`. Some instances use decimal sizes such as 36.6; these values are scaled to integers for computation without changing feasibility or the objective.

**Synthetic Instances:** We generated additional synthetic instances using uniformly distributed item sizes:

$$s_i \sim U[10, 100]$$

with bin capacity:

$$C = 150$$

In Phase 3 experiments, we used three synthetic instances with  $n$  values 60, 120, and 200. While synthetic instance difficulty can be controlled by a “tightness” factor (as discussed in earlier phases), in this phase we kept the distribution and capacity fixed and varied  $n$ , using a fixed seed for reproducibility.

## 4. Example Scenario / Problem Instance

To illustrate the structure of the One-Dimensional Bin Packing Problem, 1D-BPP, and provide a concrete test case for the metaheuristic algorithms developed in this project, we construct a small but representative problem instance. The instance includes seven items with heterogeneous sizes and a fixed bin capacity, reflecting typical bin packing characteristics while remaining simple enough for step-by-step algorithmic analysis.

Table 1: Example instance for the 1D Bin Packing Problem

| Item                | Size      |
|---------------------|-----------|
| 1                   | 22        |
| 2                   | 17        |
| 3                   | 45        |
| 4                   | 12        |
| 5                   | 38        |
| 6                   | 27        |
| 7                   | 19        |
| <b>Bin Capacity</b> | <b>60</b> |

This instance contains items whose sizes vary between small values such as 12 and relatively large values such as 45, creating natural packing difficulty and multiple feasible arrangement combinations. The bin capacity is set to 60, enabling meaningful exploration-exploitation behavior in local search metaheuristics. Although the instance is small-scale, it captures the essential constraints of 1D-BPP and is used in Phase 3 to verify that the implementation is correct.

## 5. Metaheuristic Algorithm Design and Implementation: Tabu Search

### 5.1 Solution Representation

We represent a candidate solution as a **permutation** of item indices. A permutation is decoded into an actual packing using a constructive heuristic:

- **Decoder: Best-Fit packing.** Best Fit Decreasing-like behavior depends on the permutation order. Items are inserted one-by-one into the feasible bin that leaves the least remaining space; if no bin can fit the item, a new bin is opened.
- **Post-processing: Bin reduction.** After decoding, a greedy relocation procedure tries to empty low-load bins by moving their items into other bins using a largest-first relocation rule. If all items of a bin can be relocated, that bin is removed.

## 5.2 Objective Function

The primary objective is to **minimize the number of bins used**. For tie-breaking between solutions with the same number of bins, we minimize the total unused capacity:

$$f_\pi = \langle B_\pi, U_\pi \rangle$$

where  $B_\pi$  is the number of bins and  $U_\pi = \sum_j [C - \text{load}_j]$  is total unused capacity for the decoded packing. Solutions are compared lexicographically.

## 5.3 Initialization

The initial permutation is generated by sorting items in decreasing size, using deterministic tie-breaking, to obtain a strong baseline packing.

## 5.4 Neighborhood Structure

We use a sampled neighborhood consisting of two move types applied to the permutation:

- **Swap:** swap two positions in the permutation.
- **Insert:** remove the item at position  $i$  and insert it at position  $j$ .

For each iteration, a fixed number of random neighbor moves are sampled; see Section 6.1.2.

## 5.5 Tabu Memory and Aspiration

Each accepted move is stored in a short-term tabu list, FIFO order, with tenure  $\tau$ . A tabu move may still be accepted if it satisfies the **aspiration criterion**, meaning it improves the global best objective.

## 5.6 Stopping Criteria and Diversification

The algorithm stops when one of the following is reached:

- maximum iteration limit,
- time limit per run,
- reaching the lower bound  $\lceil \sum_i s_i / C \rceil$  on the number of bins.

If no improvement is observed for a fixed stagnation window, we diversify by shuffling the current permutation and clearing the tabu list.

**Algorithm 1** Tabu Search for 1D Bin Packing, permutation representation

---

```

1: Input: capacity  $C$ , item sizes  $s_1..s_n$ , parameters  $I_{\max}, K, \tau, S, T$ 
2:  $\pi \leftarrow$  initial permutation, decreasing sizes
3:  $x \leftarrow$  DECODE  $\pi$  using Best-Fit
4:  $x \leftarrow$  REDUCEBINS  $x$ 
5:  $B^* \leftarrow$  BINS  $x$ ;  $U^* \leftarrow$  UNUSED  $x$ ;  $\pi^* \leftarrow \pi$ 
6: TabuList  $\leftarrow \emptyset$ ;  $it^* \leftarrow 0$ 
7: for  $it = 1$  to  $I_{\max}$  do
8:   if time  $\geq T$  then
9:     break
10:  end if
11:  if  $it - it^* \geq S$  then
12:     $\pi \leftarrow$  SHUFFLE  $\pi^*$ ; TabuList  $\leftarrow \emptyset$ 
13:  end if
14:   $\pi_{\text{cand}} \leftarrow \emptyset$ ;  $m_{\text{cand}} \leftarrow \emptyset$ ;  $x_{\text{cand}} \leftarrow \emptyset$ 
15:   $B_{\text{cand}} \leftarrow +\infty$ ;  $U_{\text{cand}} \leftarrow +\infty$ 
16:  for  $k = 1$  to  $K$  do ▷ sample  $K$  moves
17:     $m \leftarrow$  random move, Swap or Insert
18:     $\pi' \leftarrow$  apply move  $m$  to  $\pi$ 
19:     $x' \leftarrow$  DECODE  $\pi'$ 
20:     $x' \leftarrow$  REDUCEBINS  $x'$ 
21:     $B' \leftarrow$  BINS  $x'$ ;  $U' \leftarrow$  UNUSED  $x'$ 
22:    if  $m$  is tabu and not improving global best then
23:      continue
24:    end if
25:    if  $B' < B_{\text{cand}}$  or  $B' = B_{\text{cand}}$  and  $U' < U_{\text{cand}}$  then
26:       $\pi_{\text{cand}} \leftarrow \pi'$ ;  $m_{\text{cand}} \leftarrow m$ ;  $x_{\text{cand}} \leftarrow x'$ 
27:       $B_{\text{cand}} \leftarrow B'$ ;  $U_{\text{cand}} \leftarrow U'$ 
28:    end if
29:  end for
30:  if  $\pi_{\text{cand}}$  is empty then
31:    continue
32:  end if
33:   $\pi \leftarrow \pi_{\text{cand}}$ ;  $m \leftarrow m_{\text{cand}}$ ;  $x \leftarrow x_{\text{cand}}$ 
34:   $B \leftarrow B_{\text{cand}}$ ;  $U \leftarrow U_{\text{cand}}$ 
35:  add  $m$  to TabuList, tenure  $\tau$ 
36:  if  $B < B^*$  or  $B = B^*$  and  $U < U^*$  then
37:     $B^* \leftarrow B$ ;  $U^* \leftarrow U$ ;  $\pi^* \leftarrow \pi$ ;  $it^* \leftarrow it$ 
38:    if  $B^*$  equals lower bound then
39:      break
40:    end if
41:  end if
42: end for
43: Output: best solution  $\pi^*$  and decoded packing  $x^*$ 

```

---

## 5.7 Pseudo Code, Algorithm Implementation Only

## 6. Experimental Study

### 6.1 Experimental Setup

#### 6.1.1 System Information

All experiments were executed on:

- OS: macOS 26.1, Build 25B78
- CPU: Apple M4, arm64
- RAM: 16 GB
- Programming language: Rust, rustc 1.91.1

#### 6.1.2 Algorithm Parameters

Unless stated otherwise, the following Tabu Search parameters were used for each run:

- Max iterations: 5000
- Time limit: 10 seconds per run
- Neighborhood samples per iteration: 200
- Tabu tenure: 25
- Stagnation limit, diversification: 600 iterations
- Runs per instance: 5, seeds 0–4

### 6.2 Experimental Results

Tables below report the required statistics for each instance: mean objective value as mean number of bins, best objective value as minimum bins, standard deviation, mean computation time, and best computation time over 5 runs. For each instance, we additionally report:

- **Exact:** a reference number of bins (OPT if available; otherwise a lower bound).
- **Gap% runs:** the percentage gap for each run (5 values), computed relative to this reference.

### 6.3 Distance to Exact Solution

In addition to the performance statistics, we report the distance of each heuristic run to an exact reference solution.

- **TP2 example instance:** the exact minimum number of bins is computed using an exact enumeration with pruning (branch-and-bound). This is feasible only because the instance is very small ( $n = 7$ ).
- **BinPack benchmark instances:** the third value in the instance header provides the known optimal number of bins. We use this as the exact reference (OPT).
- **Synthetic instances:** computing OPT is not tractable at these sizes. Instead, we report the capacity lower bound  $\lceil \sum_i s_i / C \rceil$  as the reference in the **Exact** column, and compute Gap% runs relative to this bound.

For each run, the percentage gap is computed as:

$$\text{gap}(\%) = \frac{B_{\text{found}} - B_{\text{exact}}}{B_{\text{exact}}} \times 100$$

Table 2: Results on TP2 example instance, 5 runs

| Instance        | Exact | Mean<br>Obj | Best<br>Obj | Std.<br>Dev. | Mean<br>Time [s] | Best<br>Time [s] | Gap% runs                    |
|-----------------|-------|-------------|-------------|--------------|------------------|------------------|------------------------------|
| TP2_example.bpp | 4     | 4.00        | 4           | 0.00         | 0.4526           | 0.4497           | 0.00,0.00,0.00,0.00,<br>0.00 |

Table 3: Results on synthetic instances, 5 runs each

| Instance      | Exact | Mean<br>Obj | Best<br>Obj | Std.<br>Dev. | Mean<br>Time [s] | Best<br>Time [s] | Gap% runs                    |
|---------------|-------|-------------|-------------|--------------|------------------|------------------|------------------------------|
| synthetic-60  | 24    | 24.00       | 24          | 0.00         | 4.2377           | 4.1850           | 0.00,0.00,0.00,0.00,<br>0.00 |
| synthetic-120 | 47    | 48.00       | 48          | 0.00         | 9.6286           | 9.4925           | 2.13,2.13,2.13,2.13,<br>2.13 |
| synthetic-200 | 78    | 78.00       | 78          | 0.00         | 10.0027          | 10.0001          | 0.00,0.00,0.00,0.00,<br>0.00 |

Table 4: Results on `binpack2.txt`, 20 instances, 5 runs each

| Instance                      | Exact | Mean Obj | Best Obj | Std. Dev. | Mean Time [s] | Best Time [s] | Gap% runs                |
|-------------------------------|-------|----------|----------|-----------|---------------|---------------|--------------------------|
| <code>binpack2_u250_00</code> | 99    | 100.00   | 100      | 0.00      | 10.0026       | 10.0003       | 1.01,1.01,1.01,1.01,1.01 |
| <code>binpack2_u250_01</code> | 100   | 101.00   | 101      | 0.00      | 10.0037       | 10.0003       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack2_u250_02</code> | 102   | 103.00   | 103      | 0.00      | 10.0029       | 10.0015       | 0.98,0.98,0.98,0.98,0.98 |
| <code>binpack2_u250_03</code> | 100   | 101.00   | 101      | 0.00      | 10.0025       | 10.0007       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack2_u250_04</code> | 101   | 102.00   | 102      | 0.00      | 10.0029       | 10.0015       | 0.99,0.99,0.99,0.99,0.99 |
| <code>binpack2_u250_05</code> | 101   | 103.00   | 103      | 0.00      | 10.0025       | 10.0005       | 1.98,1.98,1.98,1.98,1.98 |
| <code>binpack2_u250_06</code> | 102   | 102.80   | 102      | 0.40      | 8.0205        | 0.0861        | 0.98,0.98,0.98,0.98,0.00 |
| <code>binpack2_u250_07</code> | 104   | 104.40   | 104      | 0.49      | 10.0032       | 10.0004       | 0.96,0.00,0.00,0.96,0.00 |
| <code>binpack2_u250_08</code> | 105   | 106.20   | 106      | 0.40      | 10.0031       | 10.0013       | 0.95,1.90,0.95,0.95,0.95 |
| <code>binpack2_u250_09</code> | 101   | 102.00   | 102      | 0.00      | 10.0020       | 10.0001       | 0.99,0.99,0.99,0.99,0.99 |
| <code>binpack2_u250_10</code> | 105   | 106.00   | 106      | 0.00      | 10.0026       | 10.0006       | 0.95,0.95,0.95,0.95,0.95 |
| <code>binpack2_u250_11</code> | 101   | 102.40   | 102      | 0.49      | 10.0018       | 10.0001       | 0.99,0.99,1.98,1.98,0.99 |
| <code>binpack2_u250_12</code> | 106   | 107.00   | 107      | 0.00      | 10.0048       | 10.0039       | 0.94,0.94,0.94,0.94,0.94 |
| <code>binpack2_u250_13</code> | 103   | 104.00   | 104      | 0.00      | 10.0036       | 10.0025       | 0.97,0.97,0.97,0.97,0.97 |
| <code>binpack2_u250_14</code> | 100   | 101.00   | 101      | 0.00      | 10.0024       | 10.0009       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack2_u250_15</code> | 105   | 107.00   | 107      | 0.00      | 10.0013       | 10.0001       | 1.90,1.90,1.90,1.90,1.90 |
| <code>binpack2_u250_16</code> | 97    | 98.00    | 98       | 0.00      | 10.0026       | 10.0018       | 1.03,1.03,1.03,1.03,1.03 |
| <code>binpack2_u250_17</code> | 100   | 101.00   | 101      | 0.00      | 10.0022       | 10.0009       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack2_u250_18</code> | 100   | 101.60   | 101      | 0.49      | 10.0025       | 10.0003       | 1.00,2.00,2.00,2.00,1.00 |
| <code>binpack2_u250_19</code> | 102   | 103.00   | 103      | 0.00      | 10.0015       | 10.0002       | 0.98,0.98,0.98,0.98,0.98 |

Table 5: Results on `binpack4.txt`, 20 instances, 5 runs each

| Instance                       | Exact | Mean Obj | Best Obj | Std. Dev. | Mean Time [s] | Best Time [s] | Gap% runs                |
|--------------------------------|-------|----------|----------|-----------|---------------|---------------|--------------------------|
| <code>binpack4_u1000_00</code> | 399   | 403.00   | 403      | 0.00      | 10.0286       | 10.0053       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack4_u1000_01</code> | 406   | 410.60   | 410      | 0.49      | 10.0162       | 10.0028       | 1.23,0.99,1.23,0.99,1.23 |
| <code>binpack4_u1000_02</code> | 411   | 415.40   | 415      | 0.49      | 10.0290       | 10.0006       | 0.97,0.97,1.22,0.97,1.22 |
| <code>binpack4_u1000_03</code> | 411   | 416.00   | 416      | 0.00      | 10.0195       | 10.0036       | 1.22,1.22,1.22,1.22,1.22 |
| <code>binpack4_u1000_04</code> | 397   | 401.00   | 401      | 0.00      | 10.0352       | 10.0094       | 1.01,1.01,1.01,1.01,1.01 |
| <code>binpack4_u1000_05</code> | 399   | 404.00   | 404      | 0.00      | 10.0231       | 10.0035       | 1.25,1.25,1.25,1.25,1.25 |
| <code>binpack4_u1000_06</code> | 395   | 399.00   | 399      | 0.00      | 10.0299       | 10.0116       | 1.01,1.01,1.01,1.01,1.01 |
| <code>binpack4_u1000_07</code> | 404   | 408.00   | 408      | 0.00      | 10.0194       | 10.0040       | 0.99,0.99,0.99,0.99,0.99 |
| <code>binpack4_u1000_08</code> | 399   | 403.00   | 403      | 0.00      | 10.0357       | 10.0176       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack4_u1000_09</code> | 397   | 403.40   | 403      | 0.49      | 10.0341       | 10.0076       | 1.51,1.76,1.51,1.76,1.51 |
| <code>binpack4_u1000_10</code> | 400   | 404.00   | 404      | 0.00      | 10.0234       | 10.0042       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack4_u1000_11</code> | 401   | 405.00   | 405      | 0.00      | 10.0315       | 10.0107       | 1.00,1.00,1.00,1.00,1.00 |
| <code>binpack4_u1000_12</code> | 393   | 397.00   | 397      | 0.00      | 10.0173       | 10.0087       | 1.02,1.02,1.02,1.02,1.02 |
| <code>binpack4_u1000_13</code> | 396   | 400.00   | 400      | 0.00      | 10.0331       | 10.0183       | 1.01,1.01,1.01,1.01,1.01 |
| <code>binpack4_u1000_14</code> | 394   | 399.00   | 399      | 0.00      | 10.0272       | 10.0160       | 1.27,1.27,1.27,1.27,1.27 |
| <code>binpack4_u1000_15</code> | 402   | 407.00   | 407      | 0.00      | 10.0264       | 10.0081       | 1.24,1.24,1.24,1.24,1.24 |
| <code>binpack4_u1000_16</code> | 404   | 407.00   | 407      | 0.00      | 10.0281       | 10.0038       | 0.74,0.74,0.74,0.74,0.74 |
| <code>binpack4_u1000_17</code> | 404   | 409.00   | 409      | 0.00      | 10.0316       | 10.0041       | 1.24,1.24,1.24,1.24,1.24 |
| <code>binpack4_u1000_18</code> | 399   | 402.00   | 402      | 0.00      | 10.0307       | 10.0167       | 0.75,0.75,0.75,0.75,0.75 |
| <code>binpack4_u1000_19</code> | 400   | 405.80   | 405      | 0.40      | 10.0279       | 10.0116       | 1.50,1.50,1.50,1.50,1.25 |

Table 6: Results on `binpack7.txt`, 20 instances, 5 runs each

| Instance         | Exact | Mean Obj | Best Obj | Std. Dev. | Mean Time [s] | Best Time [s] | Gap% runs                |
|------------------|-------|----------|----------|-----------|---------------|---------------|--------------------------|
| binpack7_t249_00 | 83    | 87.80    | 87       | 0.40      | 10.0024       | 10.0000       | 6.02,6.02,4.82,6.02,6.02 |
| binpack7_t249_01 | 83    | 87.40    | 87       | 0.49      | 10.0024       | 10.0010       | 4.82,6.02,4.82,6.02,4.82 |
| binpack7_t249_02 | 83    | 87.00    | 86       | 0.63      | 10.0025       | 10.0004       | 6.02,4.82,4.82,4.82,3.61 |
| binpack7_t249_03 | 83    | 87.40    | 87       | 0.49      | 10.0021       | 10.0013       | 4.82,4.82,6.02,4.82,6.02 |
| binpack7_t249_04 | 83    | 87.80    | 87       | 0.40      | 10.0032       | 10.0007       | 6.02,6.02,6.02,6.02,4.82 |
| binpack7_t249_05 | 83    | 87.40    | 87       | 0.49      | 10.0024       | 10.0008       | 4.82,4.82,6.02,4.82,6.02 |
| binpack7_t249_06 | 83    | 87.80    | 87       | 0.40      | 10.0020       | 10.0001       | 6.02,6.02,6.02,6.02,4.82 |
| binpack7_t249_07 | 83    | 88.00    | 88       | 0.00      | 10.0034       | 10.0017       | 6.02,6.02,6.02,6.02,6.02 |
| binpack7_t249_08 | 83    | 87.20    | 87       | 0.40      | 10.0022       | 10.0004       | 4.82,4.82,4.82,4.82,6.02 |
| binpack7_t249_09 | 83    | 87.60    | 87       | 0.49      | 10.0023       | 10.0006       | 6.02,6.02,4.82,6.02,4.82 |
| binpack7_t249_10 | 83    | 87.80    | 87       | 0.40      | 10.0031       | 10.0015       | 6.02,4.82,6.02,6.02,6.02 |
| binpack7_t249_11 | 83    | 87.60    | 87       | 0.49      | 10.0017       | 10.0002       | 4.82,6.02,6.02,6.02,4.82 |
| binpack7_t249_12 | 83    | 87.40    | 87       | 0.49      | 10.0026       | 10.0008       | 4.82,4.82,6.02,4.82,6.02 |
| binpack7_t249_13 | 83    | 87.80    | 87       | 0.40      | 10.0028       | 10.0009       | 6.02,6.02,4.82,6.02,6.02 |
| binpack7_t249_14 | 83    | 87.00    | 87       | 0.00      | 10.0013       | 10.0003       | 4.82,4.82,4.82,4.82,4.82 |
| binpack7_t249_15 | 83    | 87.80    | 87       | 0.40      | 10.0015       | 10.0001       | 6.02,6.02,4.82,6.02,6.02 |
| binpack7_t249_16 | 83    | 87.40    | 87       | 0.49      | 10.0025       | 10.0001       | 4.82,4.82,4.82,6.02,6.02 |
| binpack7_t249_17 | 83    | 87.20    | 87       | 0.40      | 10.0032       | 10.0022       | 4.82,4.82,4.82,6.02,4.82 |
| binpack7_t249_18 | 83    | 87.60    | 87       | 0.49      | 10.0016       | 10.0006       | 4.82,6.02,6.02,6.02,4.82 |
| binpack7_t249_19 | 83    | 87.60    | 87       | 0.49      | 10.0027       | 10.0005       | 6.02,6.02,6.02,4.82,4.82 |

Table 7: Results on `binpack8.txt`, 20 instances, 5 runs each

| Instance         | Exact | Mean Obj | Best Obj | Std. Dev. | Mean Time [s] | Best Time [s] | Gap% runs                |
|------------------|-------|----------|----------|-----------|---------------|---------------|--------------------------|
| binpack8_t501_00 | 167   | 177.60   | 177      | 0.80      | 10.0051       | 10.0015       | 5.99,5.99,7.19,6.59,5.99 |
| binpack8_t501_01 | 167   | 177.40   | 177      | 0.49      | 10.0057       | 10.0010       | 5.99,5.99,6.59,5.99,6.59 |
| binpack8_t501_02 | 167   | 177.40   | 176      | 1.02      | 10.0051       | 10.0011       | 7.19,5.39,5.99,6.59,5.99 |
| binpack8_t501_03 | 167   | 177.00   | 176      | 0.63      | 10.0059       | 10.0005       | 5.99,5.99,6.59,5.39,5.99 |
| binpack8_t501_04 | 167   | 177.00   | 176      | 0.63      | 10.0047       | 10.0022       | 5.99,5.39,5.99,6.59,5.99 |
| binpack8_t501_05 | 167   | 177.20   | 176      | 0.98      | 10.0056       | 10.0007       | 5.99,7.19,5.99,5.99,5.39 |
| binpack8_t501_06 | 167   | 176.20   | 175      | 0.98      | 10.0047       | 10.0007       | 5.39,4.79,6.59,5.39,5.39 |
| binpack8_t501_07 | 167   | 176.60   | 175      | 1.02      | 10.0065       | 10.0054       | 6.59,5.99,4.79,5.39,5.99 |
| binpack8_t501_08 | 167   | 177.00   | 176      | 0.89      | 10.0056       | 10.0026       | 5.39,6.59,5.99,5.39,6.59 |
| binpack8_t501_09 | 167   | 177.00   | 177      | 0.00      | 10.0038       | 10.0019       | 5.99,5.99,5.99,5.99,5.99 |
| binpack8_t501_10 | 167   | 176.60   | 176      | 0.80      | 10.0082       | 10.0000       | 5.99,5.39,5.39,5.39,6.59 |
| binpack8_t501_11 | 167   | 176.40   | 176      | 0.49      | 10.0074       | 10.0052       | 5.39,5.39,5.39,5.99,5.99 |
| binpack8_t501_12 | 167   | 177.20   | 177      | 0.40      | 10.0059       | 10.0012       | 5.99,5.99,5.99,5.99,6.59 |
| binpack8_t501_13 | 167   | 177.00   | 176      | 0.63      | 10.0090       | 10.0077       | 5.99,5.99,5.99,5.39,6.59 |
| binpack8_t501_14 | 167   | 176.60   | 176      | 0.49      | 10.0055       | 10.0014       | 5.99,5.99,5.39,5.99,5.39 |
| binpack8_t501_15 | 167   | 177.00   | 176      | 0.63      | 10.0049       | 10.0003       | 5.99,5.99,5.99,6.59,5.39 |
| binpack8_t501_16 | 167   | 176.60   | 176      | 0.49      | 10.0068       | 10.0011       | 5.39,5.39,5.99,5.99,5.99 |
| binpack8_t501_17 | 167   | 176.00   | 175      | 0.63      | 10.0059       | 10.0040       | 5.99,5.39,5.39,4.79,5.39 |
| binpack8_t501_18 | 167   | 176.80   | 176      | 0.75      | 10.0067       | 10.0018       | 5.39,5.39,5.99,6.59,5.99 |
| binpack8_t501_19 | 167   | 178.00   | 177      | 0.63      | 10.0067       | 10.0008       | 6.59,7.19,6.59,6.59,5.99 |

## 7. Discussion and Conclusion

This phase implemented a permutation-based Tabu Search algorithm for the 1D Bin Packing Problem in Rust and validated it on the TP2 example instance. The experimental study followed the course requirement of running each instance 5 times and reporting mean/best/std objective values along with computation time.

Across the benchmark instances, results are stable across different random seeds. Many benchmark runs terminate close to the 10-second time limit, which explains why mean and best time values are often very similar across runs. Standard deviation is frequently zero when all 5 runs converge to the same number of bins within the time limit; nonzero standard deviation appears in instances where at least one run reaches a different bin count (e.g., by finding a slightly better packing).

On the BinPack instances, the reported Exact values show that the algorithm typically remains within a small percentage gap of OPT under the given runtime and parameter limits. Overall, Tabu Search provides a practical balance between solution quality and computation time for medium-to-large instances, while the exact comparison helps quantify how close the heuristic solutions are to the optimum.

## References

## References

Beasley, J. E. (1990). Or-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5–30.