YEDITEPE UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

Term Project Report

CSE480/591: Optimization with Metaheuristics

Student Name: Barış Can Sertkaya / Erhan Önaldı

Student ID: 20210702022 / 20210702136

Date: 30.10.2025

Project Title: 1D Bin Packing Problem

Instructor: Asst. Prof. Dr. Gizem Süngü 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:

- $i = 1, 2, \ldots, n$ represents the items,
- $j = 1, 2, \ldots, m$ 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, the fitness function is defined as:

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

where B is the number of bins used and α is a penalty coefficient discouraging infeasible solutions.

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. The random generator seed was fixed to ensure 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
Bin Capacity	**60**

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_π 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 τ. 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.

5.7 Pseudo Code, Algorithm Implementation Only

6. Experimental Study

6.1 Experimental Setup

6.1.1 System Information

All experiments were executed on:

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

Input: capacity C, item sizes $s_1..s_n$, parameters $I_{\max}, K, \tau, S, T$
$\pi \leftarrow$ initial permutation, decreasing sizes
$x \leftarrow$ Decode π using Best-Fit
$x \leftarrow$ ReduceBins x
$B^\star \leftarrow$ Bins x; $U^\star \leftarrow$ Unused x; $\pi^\star \leftarrow \pi$
TabuList $\leftarrow \emptyset$; $it^\star \leftarrow 0$
for $it = 1$ to $I_{\max}$ **do**
    if time $\geq T$ **then**
        break
    end if
    if $it - it^\star \geq S$ **then**
        $\pi \leftarrow$ Shuffle $\pi^\star$; TabuList $\leftarrow \emptyset$
    end if
    $\pi_{\text{cand}} \leftarrow \emptyset$; $m_{\text{cand}} \leftarrow \emptyset$; $x_{\text{cand}} \leftarrow \emptyset$
    $B_{\text{cand}} \leftarrow +\infty$; $U_{\text{cand}} \leftarrow +\infty$
    for $k = 1$ to K **do** ▷ sample K moves
        $m \leftarrow$ random move, Swap or Insert
        $\pi' \leftarrow$ apply move m to π
        $x' \leftarrow$ Decode π'
        $x' \leftarrow$ ReduceBins x'
        $B' \leftarrow$ Bins x'; $U' \leftarrow$ Unused x'
        if m is tabu **and** not improving global best **then**
            continue
        end if
        if $B' < B_{\text{cand}}$ **or** $B' = B_{\text{cand}}$ **and** $U' < U_{\text{cand}}$ **then**
            $\pi_{\text{cand}} \leftarrow \pi'$; $m_{\text{cand}} \leftarrow m$; $x_{\text{cand}} \leftarrow x'$
            $B_{\text{cand}} \leftarrow B'$; $U_{\text{cand}} \leftarrow U'$
        end if
    end for
    if π_{cand} is empty **then**
        continue
    end if
    $\pi \leftarrow \pi_{\text{cand}}$; $m \leftarrow m_{\text{cand}}$; $x \leftarrow x_{\text{cand}}$
    $B \leftarrow B_{\text{cand}}$; $U \leftarrow U_{\text{cand}}$
    add m to TabuList, tenure τ
    if $B < B^\star$ **or** $B = B^\star$ **and** $U < U^\star$ **then**
        $B^\star \leftarrow B$; $U^\star \leftarrow U$; $\pi^\star \leftarrow \pi$; $it^\star \leftarrow it$
        if $B^\star$ equals lower bound **then**
            break
        end if
    end if
end for
Output: best solution $\pi^\star$ and decoded packing $x^\star$

- 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.

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.

- For the TP2 example instance, the exact minimum number of bins is computed using an exact branch-and-bound procedure, which is feasible due to the small problem size.
- For the BinPack benchmark instances, the third value in the instance header provides the known optimal number of bins. We use this as the exact reference.

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 Obj	Best Obj	Std. Dev.	Mean Time [s]	Best Time [s]	Gap% run
TP2_example.bpp	4	4.00	4	0.00	0.4526	0.4497	0.00,0.00

Table 3: Results on synthetic instances, 5 runs each

Instance	Exact	Mean Obj	Best Obj	Std. Dev.	Mean Time [s]	Best Time [s]	Gap% runs
synthetic-60	-	24.00	24	0.00	4.3736	4.3393	-
synthetic-120	-	48.00	48	0.00	9.8773	9.7572	-
synthetic-200	-	78.00	78	0.00	10.0023	10.0005	-

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
binpack2_u250_00	99	100.00	100	0.00	10.0026	10.0003	1.01,1.01,1.01,1.01,1.01
binpack2_u250_01	100	101.00	101	0.00	10.0037	10.0003	1.00,1.00,1.00,1.00,1.00
binpack2_u250_02	102	103.00	103	0.00	10.0029	10.0015	0.98,0.98,0.98,0.98,0.98
binpack2_u250_03	100	101.00	101	0.00	10.0025	10.0007	1.00,1.00,1.00,1.00,1.00
binpack2_u250_04	101	102.00	102	0.00	10.0029	10.0015	0.99,0.99,0.99,0.99,0.99
binpack2_u250_05	101	103.00	103	0.00	10.0025	10.0005	1.98,1.98,1.98,1.98,1.98
binpack2_u250_06	102	102.80	102	0.40	8.0205	0.0861	0.98,0.98,0.98,0.98,0.00
binpack2_u250_07	104	104.40	104	0.49	10.0032	10.0004	0.96,0.00,0.00,0.96,0.00
binpack2_u250_08	105	106.20	106	0.40	10.0031	10.0013	0.95,1.90,0.95,0.95,0.95
binpack2_u250_09	101	102.00	102	0.00	10.0020	10.0001	0.99,0.99,0.99,0.99,0.99
binpack2_u250_10	105	106.00	106	0.00	10.0026	10.0006	0.95,0.95,0.95,0.95,0.95
binpack2_u250_11	101	102.40	102	0.49	10.0018	10.0001	0.99,0.99,1.98,1.98,0.99
binpack2_u250_12	106	107.00	107	0.00	10.0048	10.0039	0.94,0.94,0.94,0.94,0.94
binpack2_u250_13	103	104.00	104	0.00	10.0036	10.0025	0.97,0.97,0.97,0.97,0.97
binpack2_u250_14	100	101.00	101	0.00	10.0024	10.0009	1.00,1.00,1.00,1.00,1.00
binpack2_u250_15	105	107.00	107	0.00	10.0013	10.0001	1.90,1.90,1.90,1.90,1.90
binpack2_u250_16	97	98.00	98	0.00	10.0026	10.0018	1.03,1.03,1.03,1.03,1.03
binpack2_u250_17	100	101.00	101	0.00	10.0022	10.0009	1.00,1.00,1.00,1.00,1.00
binpack2_u250_18	100	101.60	101	0.49	10.0025	10.0003	1.00,2.00,2.00,2.00,1.00
binpack2_u250_19	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
binpack4_u1000_00	399	403.00	403	0.00	10.0286	10.0053	1.00,1.00,1.00,1.00,1.00
binpack4_u1000_01	406	410.60	410	0.49	10.0162	10.0028	1.23,0.99,1.23,0.99,1.23
binpack4_u1000_02	411	415.40	415	0.49	10.0290	10.0006	0.97,0.97,1.22,0.97,1.22
binpack4_u1000_03	411	416.00	416	0.00	10.0195	10.0036	1.22,1.22,1.22,1.22,1.22
binpack4_u1000_04	397	401.00	401	0.00	10.0352	10.0094	1.01,1.01,1.01,1.01,1.01
binpack4_u1000_05	399	404.00	404	0.00	10.0231	10.0035	1.25,1.25,1.25,1.25,1.25
binpack4_u1000_06	395	399.00	399	0.00	10.0299	10.0116	1.01,1.01,1.01,1.01,1.01
binpack4_u1000_07	404	408.00	408	0.00	10.0194	10.0040	0.99,0.99,0.99,0.99,0.99
binpack4_u1000_08	399	403.00	403	0.00	10.0357	10.0176	1.00,1.00,1.00,1.00,1.00
binpack4_u1000_09	397	403.40	403	0.49	10.0341	10.0076	1.51,1.76,1.51,1.76,1.51
binpack4_u1000_10	400	404.00	404	0.00	10.0234	10.0042	1.00,1.00,1.00,1.00,1.00
binpack4_u1000_11	401	405.00	405	0.00	10.0315	10.0107	1.00,1.00,1.00,1.00,1.00
binpack4_u1000_12	393	397.00	397	0.00	10.0173	10.0087	1.02,1.02,1.02,1.02,1.02
binpack4_u1000_13	396	400.00	400	0.00	10.0331	10.0183	1.01,1.01,1.01,1.01,1.01
binpack4_u1000_14	394	399.00	399	0.00	10.0272	10.0160	1.27,1.27,1.27,1.27,1.27
binpack4_u1000_15	402	407.00	407	0.00	10.0264	10.0081	1.24,1.24,1.24,1.24,1.24
binpack4_u1000_16	404	407.00	407	0.00	10.0281	10.0038	0.74,0.74,0.74,0.74,0.74
binpack4_u1000_17	404	409.00	409	0.00	10.0316	10.0041	1.24,1.24,1.24,1.24,1.24
binpack4_u1000_18	399	402.00	402	0.00	10.0307	10.0167	0.75,0.75,0.75,0.75,0.75
binpack4_u1000_19	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. In some cases, the algorithm terminates early when it reaches the theoretical lower bound on the number of bins. Overall, Tabu Search provides a practical balance between solution quality and computation time for medium-to-large instances.

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.