Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Temas Selectos de Optimización

Bidimensional Bin Packing Problem

Teacher: Dr. MARÍA ANGÉLICA SALAZAR AGUILAR

Group: 006

Oscar Garza Hinojosa	1997010	ITS
Fernando Yahir Garcia Davila	1995329	ITS
Derek Alejandro Sauceda Morales	1999672	ITS
Guillermo Vladimir Flores Báez	2127967	ITS

Period January - June 2025

Introduction

For this report we need to understand what is a Combinatorial Optimization Problem (COP) & a Constructive Heuristic (CH), a COP involve finding the best solution from a finite set of possibilities, characterized by discrete feasible solutions, a well-defined objective function, and often a very large configuration space subject to various constraints. A CH is a problem-solving technique used in combinatorial optimization to build a feasible solution step by step. It starts with an empty or partial solution and iteratively adds components until a complete or good solution is obtained.

In this report we will take a dive into a COP called Bidimensional Bin Packing Problem, with which we will do some studies and experiments involving this problem.

The Two-Dimensional Bin Packing Problem (2D-BPP) is a classic combinatorial optimization problem that involves arranging a set of rectangular items within a finite number of fixed size rectangular bins. The objective is to minimize the number of bins used, ensuring that items do not overlap and are aligned with the bin edges.

The 2D-BPP has numerous real world applications across various industries:

- **Manufacturing and Cutting Industries**: Optimizing the cutting of raw materials like wood, metal, or fabric to minimize waste.
- **Logistics and Shipping**: Efficiently loading containers, pallets, or trucks to maximize space utilization and reduce transportation costs.
- **Textile and Garment Industry**: Arranging patterns on fabric to minimize material usage during clothing production.
- **Glass and Sheet Metal Cutting**: Determining optimal layouts for cutting glass panes or metal sheets to fulfill specific size requirements with minimal waste.
- **Furniture Manufacturing**: Planning the layout of furniture components on raw material sheets to optimize usage and reduce costs.
- **Printing Industry**: Organizing various print jobs on large sheets to maximize paper usage and minimize waste.
- **VLSI Design**: In electronic design automation, arranging circuit components on silicon wafers efficiently.
- **Cloud Computing and Data Centers**: Allocating virtual machines to physical servers in a way that optimizes resource utilization.

General Problem Description

There are an unlimited number of identical rectangular bins, each with a fixed width W and height H. There are N rectangular items. Item i has a width wi and a height hi for i = 1,..., N. The problem consists in assigning each item to a bin and a position (xi, yi) within that bin such that:

1. No two items overlap. For any two distinct items i and j assigned to the same bin, their rectangular regions must not intersect.
2. Each item is completely contained within the bin it is assigned to. That is, for each item i placed at (xi, yi) in a bin, it must satisfy $0 \le xi \le W - wi$ and $0 \le yi \le H - hi$.

3. The objective is to minimize the total number of bins used to pack all N items.

Sets and Indices:

- Let N = {1, 2,..., n}: set of items.
- Let B={1, 2,..., n}: set of potential bins (at most one item per bin ⇒ upper bound).
- W, H: width and height of each bin.
- For each item i ∈ N:
  - wi: width of item i.
  - hi: height of item i.

Decision Variables:

$$x_i \in [0, W]: x - coordinate\ of\ the\ bottom - left\ corner\ of\ item\ i.$$
$$y_i \in [0, H]: y - coordinate\ of\ the\ bottom - left\ corner\ of\ item\ i.$$
$$r_i \in \{0, 1\}: 1\ if\ item\ i\ is\ rotated\ 90°, 0\ otherwise.$$
$$z_{ik} \in \{0, 1\}: 1\ if\ item\ i\ is\ placed\ in\ bin\ k, 0\ otherwise.$$
$$o_{ij}^{k} \in \{0, 1\}: 1\ if\ item\ i\ is\ to\ the\ left\ of\ item\ j\ in\ bin\ k.$$
$$u_k \in \{0, 1\}: 1\ if\ bin\ k\ is\ used, 0\ otherwise.$$

Objective Function:

$$Min \sum_{k \in B}^{n} u_k$$

Constraints:

Each item must be placed in exactly one bin:

$$\sum_{k \in B}^{n} z_{ik} = 1 \ \forall i \in N$$

Item placement within bin boundaries:

$$x_i + (1 - r_i) \cdot w_i + r_i \cdot h_i \ \leq \ W + (1 - \sum_{k} z_{ik}) \cdot M \ \forall i \ \in \ N$$

$$y_i + (1 - r_i) \cdot h_i + r_i \cdot w_i \ \leq \ H + (1 - \sum_{k} z_{ik}) \cdot M \ \forall i \ \in \ N$$

Non-overlapping constraints:

$$x_i + w_i^{r_i} \ \leq \ x_i + M(1 - o_{ij}{}^{k}) \ \ (i\ is\ left\ of\ j)$$
$$x_j + w_j^{r_j} \ \leq \ x_i + Mo_{ij}{}^{k} \ \ (j\ is\ left\ of\ i)$$
$$y_i + h_i^{r_i} \ \leq \ y_j + M(1 - o_{ij}{}^{k}) \ \ (i\ is\ below\ j)$$
$$y_j + h_j^{r_j} \ \leq \ y_i + Mo_{ji}{}^{k} \ \ (j\ is\ below\ i)$$

Where:

$$w_i^{r_i} = (1 - r_i) \cdot w_i + r_i \cdot h_i$$
$$h_i^{ri} = (1 - r_i) \cdot h_i + r_i \cdot w_i$$

Linking item-bin assignments to bin usage:

$$z_{ik} \ \leq \ u_k \ \ \forall i \ \in \ N, \forall k \ \in \ B$$

Decision Variables must be discreet:

$r_i, z_{ik}, o_{ij}^k, u_k \in \{0, 1\}$

The foundational mathematical model for the Two Dimensional Bin Packing Problem (2D-BPP), particularly the Mixed Integer Linear Programming (MILP) formulation, has been significantly influenced by the work of several researchers. Notably:

- **F. R. K. Chung, M. R. Garey, and D. S. Johnson**: In their seminal 1982 paper, "On Packing Two-Dimensional Bins," these authors laid the groundwork for formalizing the 2D-BPP, introducing key concepts that have shaped subsequent research in the field. epubs.siam.org
- **David Pisinger and Mikkel Sigurd**: Their 2007 study, "Using Decomposition Techniques and Constraint Programming for Solving the Two-Dimensional Bin-Packing Problem," advanced the modeling of 2D-BPP by integrating decomposition methods and constraint programming, enhancing the solvability of complex instances. SpringerLink+2ResearchGate+2pubsonline.informs.org+2

Constructive Heuristic

Constructive Description

The general process implements a constructive heuristic based on a 2D adaptation of the First-Fit Decreasing (FFD) strategy. It attempts to pack all rectangles into the fewest number of bins possible by following these general steps:

1. **Sorting**:
   All rectangles are sorted in descending order of area to prioritize larger objects first.
2. **Placement in Existing Bins**:
   Each rectangle is tested for placement in the first bin where it fits, trying all valid positions and orientations.
3. **Opening New Bins**:
   If the rectangle doesn't fit in any current bin, a new bin is created, and placement is attempted there.
4. **Candidate Positions**:
   Each bin maintains a list of "candidate positions" (initially just (0, 0)) representing potential bottom-left corners where a rectangle could be placed. When a rectangle is successfully placed, new candidate positions are generated to the right and top of the placed rectangle.
5. **Rotation Option**:
   Each rectangle is tested in both non-rotated and rotated (90°) orientation (if it is not square) to maximize fitting chances.

Constructive Pseudocode

1. Start by sorting the rectangles from largest area to smallest.
2. Maintain a list of bins, initially empty.
3. For each rectangle in that sorted list:
4. Try to place it in one of the bins you already have:

    a. Within each bin, look at each “candidate corner” (starting at the bottom-left, then moving upward and rightward).
    b. For each corner, check first without rotating the rectangle, then—with a 90° rotation—if it isn’t a square.
    c. If the rectangle fits entirely within the bin at that corner (doesn’t go outside the bin’s borders and doesn’t overlap any already-placed rectangle), put it there.
    d. As soon as it fits, stop looking at other corners or bins—move on to the next rectangle.
5. If the rectangle doesn’t fit in any existing bin:
    a. Create a brand-new empty bin.
    b. Repeat the same corner-by-corner, rotation-by-rotation check inside that new bin.
    c. If it still can’t fit, report an error; otherwise, record it in the new bin.
6. Whenever you successfully place a rectangle, generate two new “candidate corners”: one immediately to the right of that rectangle, and one immediately above it—so you’ll try those spots for the next rectangles.
7. Continue until every rectangle has been placed.

Evaluation Function, Selection & Insertion Strategy, and Feasibility

In this Heuristic, constructive routine there really isn’t a “score” calculated for every unplaced rectangle at each step, what serves as the *evaluation function* is simply the rectangle’s area, and the heuristic uses that to decide which item to try next, plus the bottom-left criterion to decide where to try placing it.

Concretely:

1. **Item selection (unassigned elements):**
    - All unplaced rectangles are sorted once at the very start in descending order of area.
    - That means, at each iteration, the next rectangle chosen is the one with the largest remaining area.
    - No further scoring or re-evaluation of those items happens after this initial sort.
2. **Position selection within a bin:**
    - Within each bin, the heuristic keeps a small list of candidate “corners” and orders them by lowest y-coordinate first, then lowest x-coordinate (the classic bottom-left rule).
    - It tries those candidate spots in that order, and also tries the rectangle in its original orientation first, then rotated 90° (if not square).

So, in short:

- **Evaluation of “which rectangle to place next”** = sort by area (largest first).
- **Evaluation of “where to place it”** = bottom-left candidate ordering (smallest y, then smallest x).
- **Feasibility check** = does it fit entirely within the bin bounds and not overlap any already placed rectangle?

There is no separate numerical score beyond “fits or doesn’t fit,” and no recalculation of priorities after each placement. This simple area based ordering combined with the bottom-left placement rule is what drives the constructive heuristic’s decisions.

Local Search

Description & Pseudocode

For this project we decided to implement two types of local search strategies to make the system more adaptable to the user, we have First Improvement (FI) and Best Improvement (BI) to provide an initial solution for the 2D bin packing problem. In First Improvement, the algorithm iterates through each bin and each object within it, attempting to move an object to a different bin. As soon as it finds the first valid move that results in fewer total bins (one bin becomes empty), it applies the change and returns the new solution immediately, prioritizing speed over optimality. In contrast, Best Improvement exhaustively evaluates all possible objects moving across all bin pairs. It selects the move that yields the greatest reduction in the number of bins, applying it only if it results in a better solution than the original. Both strategies work on deep copies of the solution to preserve the original state and make comparisons based on the number of bins used, which reflects the efficiency of the packing.

```
FUNCTION LocalSearch_FirstImprovement(bins, bin_width, bin_height):
    MAKE a deep copy of bins to avoid modifying the original
    FOR each bin_i in bins:
        FOR each rectangle r in bin_i:
            FOR each bin_j in bins:
                IF bin_i is the same as bin_j:
                    CONTINUE
                REMOVE r from bin_i
                IF r can be placed in bin_j:
                    PLACE r in bin_j
                    IF bin_i is now empty:
                        REMOVE bin_i from bins
                    RETURN the modified bins (improved solution found)
                ELSE:
                    RE-INSERT r back into bin_i
    RETURN the original bins (no improvement found)
```

```
FUNCTION LocalSearch_BestImprovement(bins, bin_width, bin_height):
    MAKE a deep copy of bins as best_bins
    SET best_score = number of bins
```

```
FOR each bin_i in bins:
    FOR each rectangle r in bin_i:
        FOR each bin_j in bins:
            IF bin_i is the same as bin_j:
                CONTINUE
            MAKE a deep copy of bins as candidate_bins
            REMOVE r from bin_i in candidate_bins
            IF r can be placed in bin_j in candidate_bins:
                PLACE r in bin_j
                REMOVE any empty bins from candidate_bins
                IF number of bins in candidate_bins < best_score:
                    UPDATE best_score
                    UPDATE best_bins = candidate_bins

RETURN best_bins (either improved or same as original)
```

Move to Generate Neighbor Solutions

In both First Improvement (FI) and Best Improvement (BI) local search strategies for the 2D Bin Packing problem, the move used to generate neighbor solutions is the same, and it consists of:

The Move (Applied by FI and BI)

"Try to move a rectangle r from one bin to a different bin."

This move includes the following steps:

1. Select a rectangle r from a bin bin_i.
2. Temporarily remove r from bin_i.
3. Try placing r into another bin bin_j, considering both normal and rotated orientations.
4. If the placement in bin_j is successful:
    - Keep r in bin_j.
    - If bin_i becomes empty after the move, delete it from the bin list.
    - A new solution (neighbor) is created with potentially fewer bins.

How FI and BI Differ Using This Move

- **First Improvement (FI)**:
    - As soon as it finds the first valid move that reduces the total number of bins, it accepts it immediately and stops searching further.
    - This leads to faster convergence, but not necessarily the best possible improvement.

- **Best Improvement (BI)**:
    - It tries all possible moves for all rectangles across all bins.
    - It evaluates each candidate move and keeps the one that leads to the best improvement (i.e., minimum number of bins used).
    - This leads to a better-quality solution, but at the cost of longer runtime.

In both cases, the neighborhood is defined by all possible reassignments of a single rectangle to a different bin, and the goal of the move is to reduce the number of bins

Exploration Type

For this project we decided to implement two types of local search strategies to make the system more adaptable to the user, we have First Improvement (FI) and Best Improvement (BI) to provide an initial solution for the 2D bin packing problem.

- **First Improvement (FI)**:
    - As soon as it finds the first valid move that reduces the total number of bins, it accepts it immediately and stops searching further.
    - This leads to faster convergence, but not necessarily the best possible improvement.
- **Best Improvement (BI)**:
    - It tries all possible moves for all rectangles across all bins.
    - It evaluates each candidate move and keeps the one that leads to the best improvement (i.e., minimum number of bins used).
    - This leads to a better-quality solution, but at the cost of longer runtime.

Stopping Criteria

The stopping criterion for both the First Improvement (FI) and Best Improvement (BI) local search strategies in this bin packing implementation is based on whether a better solution is found in a single iteration. In the case of First Improvement, the search stops as soon as it finds the first move that reduces the total number of bins. This means the process terminates after one successful improvement move. For Best Improvement, the strategy evaluates all possible moves in the current configuration and applies only to the one that offers the best improvement (i.e., greatest reduction in the number of bins); if no improvement exists, it also stops. In both strategies, if no single move can reduce the total number of bins, the local search terminates immediately, meaning the search is limited to a single iteration and does not perform deeper or repeated explorations beyond one level of neighborhood.

Experimental Results

Computer Features

- **Model**: Asus Tuf Gaming A15
- **Processor**: AMD Ryzen 5 4600H Series 3GHz

- **iGPU**: AMD Radeon (TM) Graphics
- **GPU**: NVIDIA GeForce GTX 1660 Ti
- **RAM**: 32GB
- **ROM**: 512GB
- **OS**: Windows 11 Home 24H2 x64 Bits

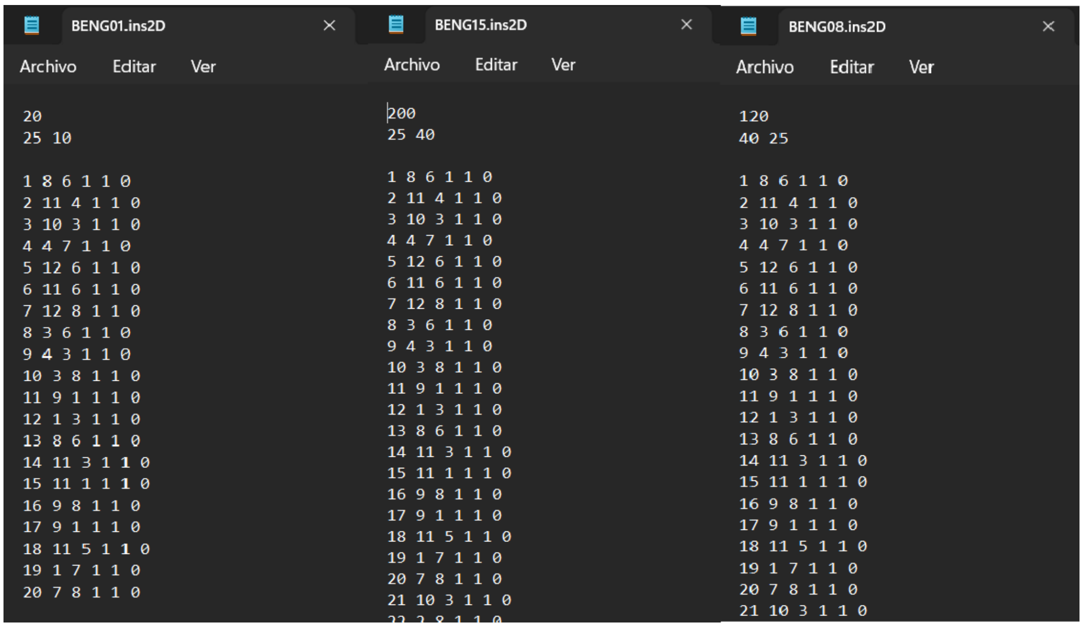
Instances Source & Description

The instances were downloaded from:
https://site.unibo.it/operations-research/en/research/2dpacklib

The instances are in the following format:
m (Number of items)
W H (Bin Measurements)
ID w_i h_i d_i b_i p_i (For each item)
where:

- m: number of items
- W: width of the bin
- H: height of the bin
- ID: item ID
- w_i: width of item i
- h_i: height of item i
- d_i: demand for item i (minimum number of copies to be packed)
- b_i: maximum number of copies of item i
- p_i: profit of item i

In this case, for our type of problem we only consider the ID, w_i, h_i columns of the objects, the other data is ignored because they are for other variants of the 2D-BPP.
Some instances have different Bin´s area and number of objects, having smaller and larger instances.



```
20
25 10

1 8 6 1 1 0
2 11 4 1 1 0
3 10 3 1 1 0
4 4 7 1 1 0
5 12 6 1 1 0
6 11 6 1 1 0
7 12 8 1 1 0
8 3 6 1 1 0
9 4 3 1 1 0
10 3 8 1 1 0
11 9 1 1 1 0
12 1 3 1 1 0
13 8 6 1 1 0
14 11 3 1 1 0
15 11 1 1 1 0
16 9 8 1 1 0
17 9 1 1 1 0
18 11 5 1 1 0
19 1 7 1 1 0
20 7 8 1 1 0
```

```
200
25 40

1 8 6 1 1 0
2 11 4 1 1 0
3 10 3 1 1 0
4 4 7 1 1 0
5 12 6 1 1 0
6 11 6 1 1 0
7 12 8 1 1 0
8 3 6 1 1 0
9 4 3 1 1 0
10 3 8 1 1 0
11 9 1 1 1 0
12 1 3 1 1 0
13 8 6 1 1 0
14 11 3 1 1 0
15 11 1 1 1 0
16 9 8 1 1 0
17 9 1 1 1 0
18 11 5 1 1 0
19 1 7 1 1 0
20 7 8 1 1 0
21 10 3 1 1 0
```

```
120
40 25

1 8 6 1 1 0
2 11 4 1 1 0
3 10 3 1 1 0
4 4 7 1 1 0
5 12 6 1 1 0
6 11 6 1 1 0
7 12 8 1 1 0
8 3 6 1 1 0
9 4 3 1 1 0
10 3 8 1 1 0
11 9 1 1 1 0
12 1 3 1 1 0
13 8 6 1 1 0
14 11 3 1 1 0
15 11 1 1 1 0
16 9 8 1 1 0
17 9 1 1 1 0
18 11 5 1 1 0
19 1 7 1 1 0
20 7 8 1 1 0
21 10 3 1 1 0
```

Results Table

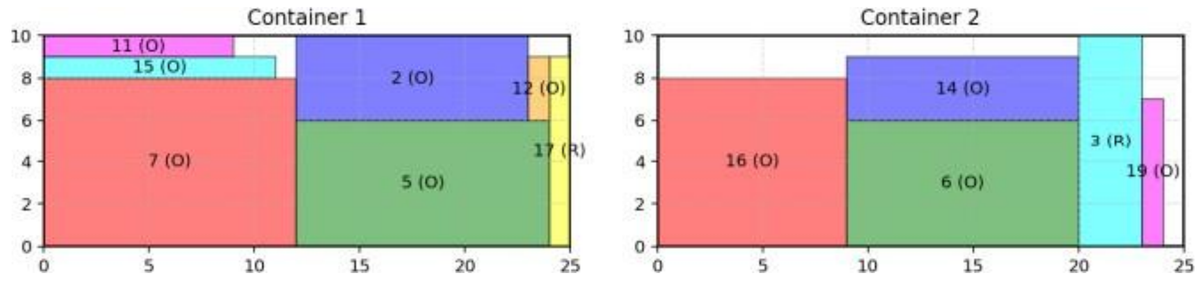
Instance Name	Objective_Value_Constructive	Objective_Value_LocalSearch(FI)	Objective_Value_LocalSearch(BI)	Constructive Computation Time(s)	Improvement %
BENG01.ins2D	4	4	4	0.0008	0%
BENG02.ins2D	7	7	7	0.0014	0%
BENG03.ins2D	10	10	10	0.0034	0%
BENG04.ins2D	12	12	12	0.0051	0%
BENG05.ins2D	14	14	14	0.0086	0%
BENG06.ins2D	2	2	2	0.0032	0%
BENG07.ins2D	3	3	3	0.0245	0%
BENG08.ins2D	47	47	47	0.3127	0%
BENG09.ins2D	60	60	60	0.3958	0%
BENG10.ins2D	163	163	163	0.3118	0%
BENG11.ins2D	188	188	188	0.4041	0%
BENG12.ins2D	154	154	154	0.6503	0%

BENG13.ins2D	160	160	160	0.6179	0%
BENG14.ins2D	33	33	32	3.6464	3.03%
BENG15.ins2D	280	280	279	41.6475	0.36%

Instance Name	Objective Value (FI)	Objective Value (BI)	Computation Time(FI) sec	Computation Time(BI) sec	Gap % Respect to Lower Bound
BENG01.ins2D	4	4	0.0004	0.0157	33.33%
BENG02.ins2D	7	7	0.0007	0.1137	16.67%
BENG03.ins2D	10	10	0.0013	0.3763	11.11%
BENG04.ins2D	12	12	0.0015	0.7757	9.09%
BENG05.ins2D	14	14	0.0018	1.3891	0%
BENG06.ins2D	2	2	0.0007	0.0203	0%
BENG07.ins2D	3	3	0.0012	0.1534	0%
BENG08.ins2D	47	47	0.0058	119.7910	0%
BENG09.ins2D	60	60	0.0086	225.2384	0%
BENG10.ins2D	163	163	0.0146	232.4329	0%
BENG11.ins2D	188	188	0.0114	246.3181	0%
BENG12.ins2D	154	154	0.0110	258.9003	0%
BENG13.ins2D	160	160	0.0119	271.7329	0%
BENG14.ins2D	33	32	0.0140	307.7170	0.0003%
BENG15.ins2D	280	279	0.0729	1678.5709	0.0007%

Analysis

What we can say from the experimental results is that it seems when the instance is smaller the Gap% increases so that's because the program sorts the objects array by decreasing area meaning the program tries to place the bigger objects first making more possible that we are wasting little gaps on the first containers leading to use an unnecessary bin later for smaller objects as we can see on the image:



Now taking about the local search strategies applied on the project we can say there is no significant improvement compared to what the constructive heuristic obtains, because for the FI strategy it only analyzes one move in each container so it doesn’t really make an impact on the solution, for the BI strategy seems that for the larger instances like the BENG14/15 where thousands of objects are studied it has more options to make more moves so that the probability of making an impact on the solution is higher, but this strategy negatively impacts on the computer performance and for instances of millions of objects it would take days or weeks depending of the instances and the computation capacity, maybe by replacing the actual heuristic for a flexible one can improve computation time and would give more margin of improvement to the local search but this needs to be studied more deeply.

By the solutions that the program finds we can tell they´re good solutions by comparing the solution found vs the lower bound calculated for each instance meaning that the solutions found are not far from the best possible solution of each instance and that the solutions are feasible, but this needs to be tested on largest instances with thousands of objects.

Also, our program skips graphical representation when more than 10 containers are used to save computation resources, because for instances where there are hundreds of containers it would take so much time to compute the problem.

Repository Link

https://github.com/Oscar135-33/2D-BPP-TSO

Conclusions

Oscar Garza Hinojosa: For this particular problem there are plenty ways to solve it by generally using heuristic methods such that it doesn't take so long to find a good or feasible solution, in our case we proposed a heuristic that is a mix of the bottom left algorithm and the first fit algorithm giving us good solutions, i think for our proposed algorithm there is work to do to maybe make more efficient and faster in the future the BI local search strategy.

Guillermo Vladimir Flores Báez: The results were favorable, although some areas for improvement in efficiency and performance were identified, especially in smaller instances. The approach has potential to be optimized and applied to more complex scenarios.

Fernando Yahir Garcia Davila: The constructive heuristic we used to solve the two-dimensional bin packing problem gave good results, especially on medium and large instances. On smaller ones, there was some inefficiency, probably because the algorithm sorts the objects by area, which can lead to wasted space early on. Even so, the results were quite close to the optimal and met all the problem constraints. I think the program's performance could be improved by using a more powerful computer, changing the programming language, or removing the graphical output.

Derek Alejandro Sauceda Morales: I gained a better understanding of how constructive heuristics solve challenging problems, such as the 2D Bin Packing, thanks to this project. I observed how small instances can be challenging and how sorting objects by area affects the results. All things considered, the solutions were nearly the best known, and I believe that better hardware or code optimization could boost performance.

References

Davies, S. (n.d.). Combinatorial Optimization. Cmu.edu. Retrieved April 9, 2025, from https://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/photoz/.g/web/glossary/comb.html

Students, P. (n.d.). 2DPackLib. Unibo.It. Retrieved April 9, 2025, from https://site.unibo.it/operations-research/en/research/2dpacklib

Survey for 2-D packing algorithms. (n.d.). Liv.ac.uk. Retrieved April 9, 2025, from https://www.csc.liv.ac.uk/~epa/surveyhtml.html

(N.d.). Sciencedirect.com. Retrieved April 9, 2025, from https://www.sciencedirect.com/topics/computer-science/heuristic-method

Cao, D., & Kotov, V. M. (2011). A best-fit heuristic algorithm for two-dimensional bin packing problem. Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology, 7, 3789–3791.

Chazelle. (1983). The bottomn-left bin-packing heuristic: An efficient implementation. IEEE Transactions on Computers. Institute of Electrical and Electronics Engineers, C–32(8), 697–707. https://doi.org/10.1109/tc.1983.1676307

Local search. (N.d.). Cmu.edu. Retrieved May 16, 2025, from 2.3 Local Search in Combinatorial Optimization

(N.d.). Sciencedirect.com. Retrieved May 16, 2025, from https://www.sciencedirect.com/science/article/abs/pii/S0377221725002693#:~:text=Local%20search%20methods%20start%20from%20a%20feasible%20solution,They%20are%20a%20common%20component%20of%20most%20metaheuristics.