Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

Temas Selectos de Optimización

Bidimensional Bin Packing Problem Report

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 experiments involving this problem.

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:

$$x_i + {w_i}^{r_i} \ \leq \ x_i + M(1 - {o_{ij}}^{k}) \ \ (i\ is\ left\ of\ j)$$

$$x_j + {w_j}^{r_j} \ \leq \ x_i + M{o_{ij}}^{k} \ \ (j\ is\ left\ of\ i)$$

$$y_i + {h_i}^{r_i} \ \leq \ y_j + M(1 - {o_{ij}}^{k}) \ \ (i\ is\ below\ j)$$

$$y_j + {h_j}^{r_j} \ \leq \ y_i + M{o_{ji}}^{k} \ \ (j\ is\ below\ i)$$

Where:

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

Linking item-bin assignments to bin usage:

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

Decision Variables must be discrete:
$r_i,\ z_{ik},\ o_{ij}^{k},\ u_k\ \in \{0, 1\}$

Constructive Heuristic

The pseudocode used solves the Bidimensional Bin Packing Problem by packing all objects into a fixed size bin. It reads object dimensions and IDs from a txt file from a particular address, calculates the total area of objects, and estimates the minimum number of bins required theoretically. The list of objects are sorted by decreasing area, and the algorithm attempts to place them in existing bins, checking both their original and rotated orientations. If an object doesn't fit, a new bin is created. Finally, the program outputs the number of bins used, details about each object's placement, and provides a visual representation of the packing arrangement.

Initialize Program

- Set up global bin dimensions.

Read Input Data from File

- Open the file.
- Read the number of objects.
- Read the bin/container width and height.
- For each object entry:
    - Read its ID, width, and height.
    - Calculate its area.
    - Store object data in a list.

Verify Area Requirements

- Calculate the total area of all objects.
- Calculate the area of one bin.

- Estimate the minimum number of bins theoretically required by dividing total area by bin area.

Sort Objects

- Sort the list of objects in descending order based on their area.

Attempt to Place Objects into Bins

- For each object in the sorted list:
  - Try placing it in existing bins:
    - Check each position within the bin:
      - Try in original orientation.
      - Try in rotated orientation if applicable.
      - Check for overlaps with already placed objects.
      - If a valid position is found, place the object.
  - If it cannot fit in any existing bin:
    - Create a new bin.
    - Try placing it in the new bin.
    - If it still does not fit, report an error or warning.

Display Results

- Print how many bins were used.
- For each bin:
  - List objects, their positions, sizes, and orientation.

Visualize the Bins

- For each bin:
    - Draw a rectangle representing the bin.
    - Draw and label each placed object within it using color and text.

End Program

Experimental Results

Computer Features

- **Model**: Asus Tuf Gaming A15
- **Processor**: AMD Ryzen 5 4600H Series
- **iGPU**: AMD Radeon(TM) Graphics
- **GPU**: NVIDIA GeForce GTX 1660 Ti
- **RAM**: 8GB
- **ROM**: 500GB
- **OS**: Windows 11 Home 24H2 x64 Bits

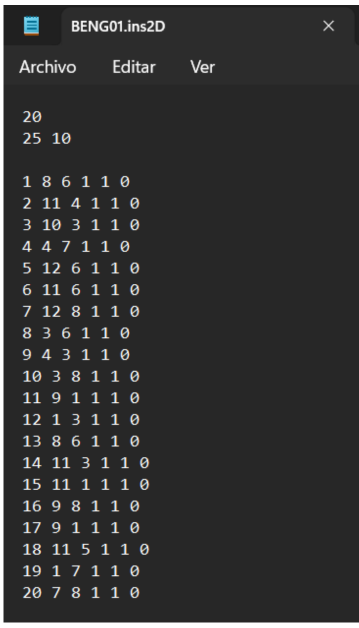
Instances Description

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:

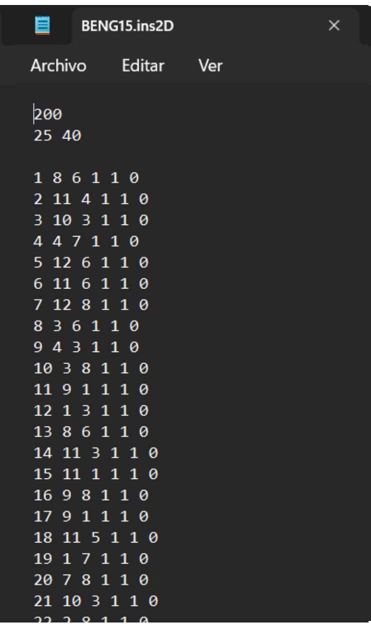
- m: number of items
- W: width of the bin
- H: height of the bin
- ID: item ID
- w_i: width of item i
- h_i: height of item i
- d_i: demand of item i (minimum number of copies to be packed)
- b_i: maximum number of copies of item i
- p_i: profit of item i

In this case for our type of problem we only take into account 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.



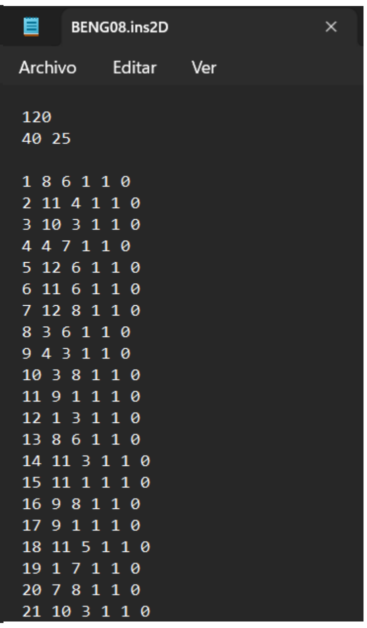
```
20
25 10

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
```

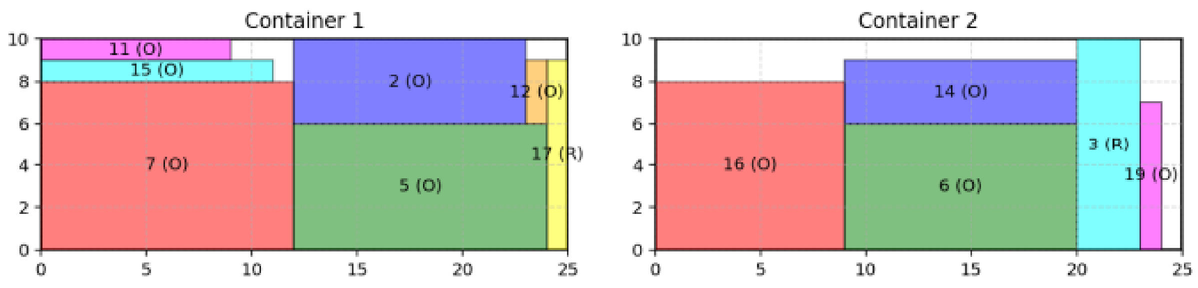
Results Table

Instance Name	Objective_Value	Computation Time(s)	Gap(%) with respect to the best known solution
BENG01.ins2D	4	0.5521	33.33%
BENG02.ins2D	7	0.8552	16.67%
BENG03.ins2D	9	1.0811	0%
BENG04.ins2D	11	1.4102	0%
BENG05.ins2D	14	1.8615	0%
BENG06.ins2D	2	0.4200	0%
BENG07.ins2D	3	0.95003	0%
BENG08.ins2D	5	1.6299	0%
BENG09.ins2D	6	2.4277	0%
BENG10.ins2D	7	3.4253	0%
BENG11.ins2D	2	0.5048	0%
BENG12.ins2D	3	0.9730	0%
BENG13.ins2D	5	1.7862	0%

BENG14.ins2D	6	2.6430	0%
BENG15.ins2D	7	3.8879	0%

Analysis

What we can say from the experimental results is that it seems when the instance is smaller the Gap% increases so we´re going to focus more on that ones, 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:



But this needs to be studied more deeply because it varies depending on the bin´s area, quantity of objects and each area of them.

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.

We think the program performance can be improved by using a better computer, or recreate the algorithm in another coding language or maybe by not using the graphical solution.

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 best fit algorithm, i think for our proposed algorithm there is work to do to maybe make it more efficient and faster in the future.

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