Multiple-type, two-dimensional finite bin packing problem

This is a mini-project for topic 3 in Fundamentals of Optimization course of SOICT – HUST

Meet our Team

Chu Minh Ha
20210293

Phan Dinh Nhat
20210654

Nguyen Huu Duan
20214951

Do Quang Minh
20210579

TABLE OF CONTENTS

01

Introduction

Problem

Package data

Quantity	N packages
Size	w_i x h_i

Truck data

Quantity	K trucks
Capacity	W_k x H_k
Cost	C_k

There are more trucks than needed to transport all

Target

- Packages that are placed in the **same container** must **not overlap**
- Loads all the packages into those given trucks such that **the total cost of trucks used is minimal**

Throughout our mini-project, some concepts are also being used instead of trucks (bins, cars) and packages (items)

Exact solutions

CP and MIP models

02

CP MODEL

Denotation

- ***N_items*** is the number of items given

  Item i has size of $w_i \times h_i$ with width w_i and height h_i
- ***N_bins*** is the number of bins given

  Bin j has size of $W_j \times H_j$ with width W_j , height H_j and cost C_j

CP MODEL

Variable

* $X_{ij} = 1$: item i packed in bin j

: bin j has been used

* $R_i = 1$: item i rotated 90

CP MODEL

l_i, r_i, b_i, t_i : left, right, bottom and top coordinates of item i

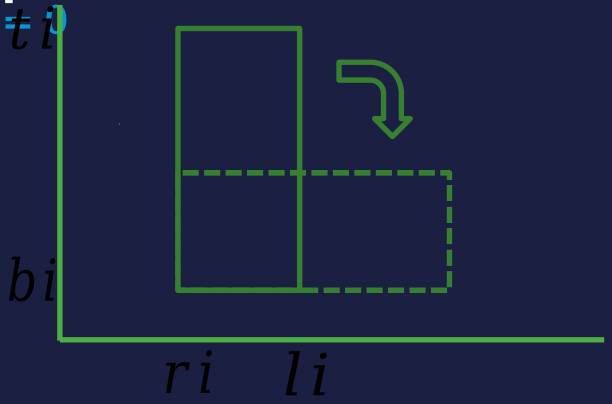
*** First way to approach:**

– if item i not rotated: $R_i = 0$

$$\rightarrow \begin{cases} r_i = l_i + w_i \\ t_i = b_i + h_i \end{cases}$$

– if item i rotated: $R_i = 1$

$$\rightarrow \begin{cases} r_i = l_i + h_i \\ t_i = b_i + w_i \end{cases}$$



Variable

Item's Coordinate

Variable

Item's Coordinate

CP MODEL

l_i, r_i, b_i, t_i : left, right, bottom and top coordinates of item i

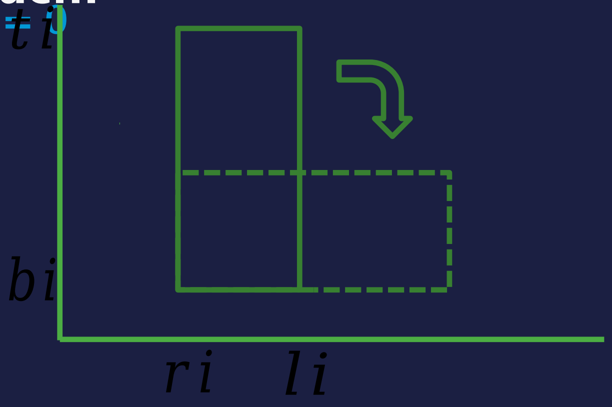
*** Another way to approach:**

- if item i not rotated: $R_i = 0$

$$\rightarrow \begin{cases} wi = w_i \\ hi = h_i \end{cases}$$

- if item i rotated: $R_i = 1$

$$\rightarrow \begin{cases} wi = h_i \\ hi = w_i \end{cases}$$



CP MODEL

- Each **item** must be packed in **exactly 1 bin**:

- Items **can not overlap** each other:
  - if
    or or or

- Items **cannot exceed** the bin:
  - if

Constraint

CP MODEL

Objective Function

Minimize

MIP MODEL

Denotation

- ***M*** is Constant value

- ***N_items*** is the number of items given

  Item i has size of w_i x h_i with width w_i and height h_i

- ***N_bins*** is the number of bins given

  Bin j has size of W_j x H_j with width W_j , height H_j and cost C_j

Variable

MIP MODEL

* $X_{ij} = 1$: item i packed in bin j

: bin j has been used

To MIP:

* $R_i = 1$: item i rotated 90

MIP MODEL

Variable

Item's Coordinate

l_i, r_i, b_i, t_i : left, right, bottom and top coordinates of item i

– if item i not rotated: $R_i = 0$

$$\rightarrow \begin{cases} ri = l_i + w_i \\ ti = b_i + h_i \end{cases}$$

– if item i rotated: $R_i = 1$

$$\rightarrow \begin{cases} ri = l_i + h_i \\ ti = b_i + w_i \end{cases}$$

To MIP:

MIP MODEL

- Each **item** must be packed in **exactly 1 bin**:

- Items **cannot exceed** the bin:
  - if

Constraint

MIP MODEL

- Items **can not overlap** each other:
  - if
    or or or

To MIP:

Constraint

Objective Function

Minimize

Heuristic

03

Overview

- **Heuristic algorithms**: Combine two algorithms, including **Guillotine** and **Maximal Rectangles**.

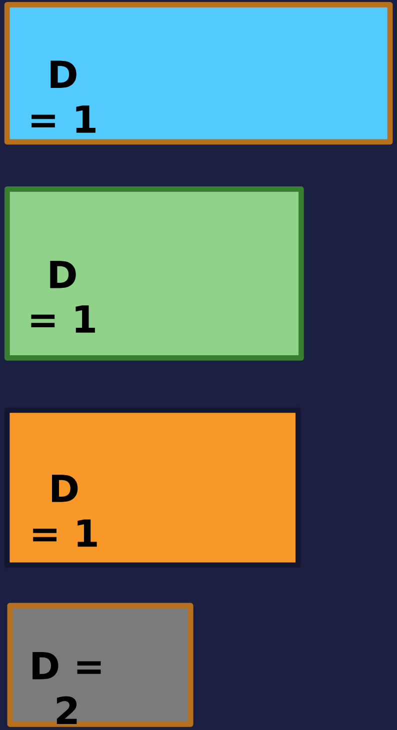
- Concept of **free rectangles**: A list of free rectangles represents the **free space of the bin**. In the Guillotine algorithm, these rectangles are pairwise disjoint.

Sorting input

Bins

Sort by **"density"** *(cost/area)*, in an **ascending order**.

Ties broken with the **descending order** of the **longer side**, followed by the descending order of the **shorter side**.



Size: 20 x 9
Area: 180
C_k: 180

Size: 15 x 12
Area: 180
C_k: 180

Size: 15 x 10
Area: 150
C_k: 150

Size: 10 x 8
Area: 80
C_k: 160

Sorting input

Size: 9 x 6

Size: 9 x 3

Size: 8 x 5

Size: 8 x 3

Items

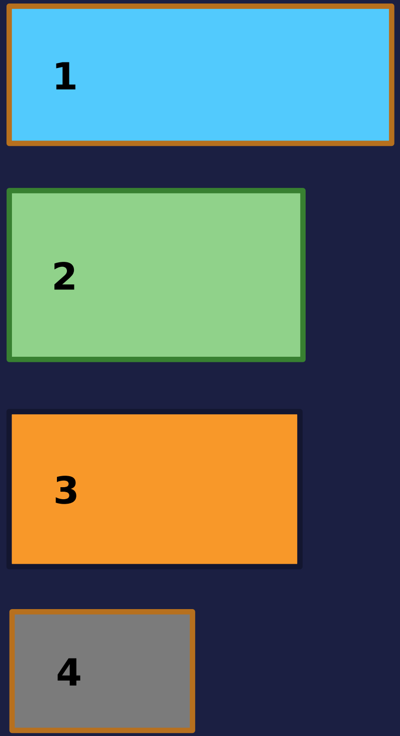
Sort by **longer side** in **descending order**.

Ties broken with the **descending order** of the **shorter side**.

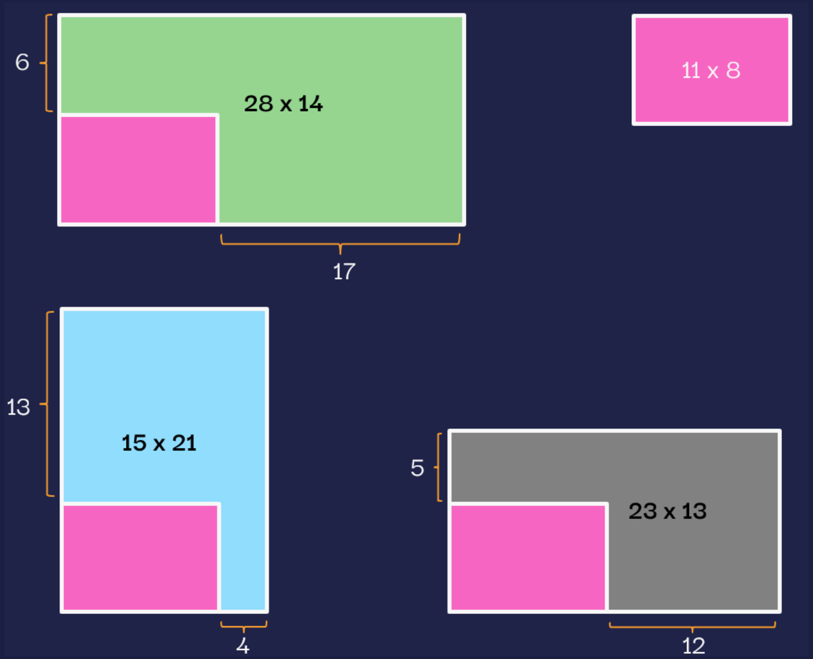
Destination for the item

Destination Bin

Bin First Fit rule – pack the item into the bin with the lowest index (after the process of sorting bin); in other words, pack in the **first bin** that the **item fits**.



Destination for the item



Destination Free Rectangle of the bin

Best Short Side Fit rule – choose a free rectangle where the shorter remainder side after insertion is minimized; in other words, **minimize** the length of the **shorter leftover side**.

Ties broken with **best longer side** (longer leftover side is minimized).

Packing process

- **Heuristic algorithms**: Combine two algorithms, including **Guillotine** and **Maximal Rectangles**.

Guillotine

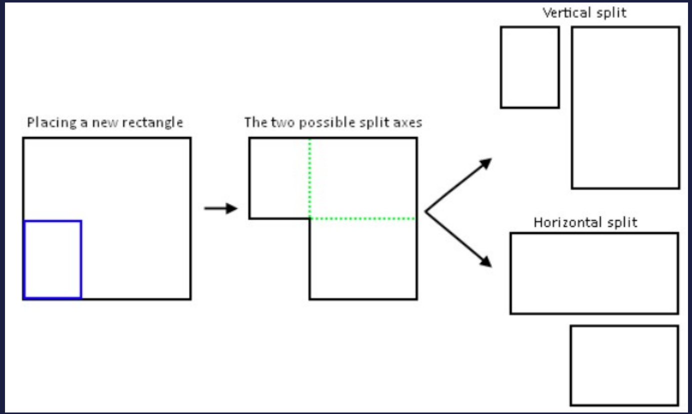
Algorithm 1: The Guillotine algorithm

```
Initialize:
Set F = {(W, H)};
Pack:
foreach Item i = (w, h) in the list of inserted items of the bin do
    Decide the free rectangle F_j ∈ F to pack the item into;
    Decide the orientation for the item and place it at the bottom-left of F_j;
    Use the guillotine split scheme to subdivide F_j into two new free rectangles F_j1 and F_j2;
    Set F ← F ∪ {F_j1, F_j2}\{F_j};
    foreach Ordered pair of free rectangles F_j1, F_j2 in F do
        if F_j1 and F_j2 can be merge together then
            F_merge = Merge F_j1 and F_j2;
            Set F ← F ∪ {F_merge}\{F_j1, F_j2};
        end
    end
end
```

Pseudocode for Guillotine algorithm

Guillotine

Splitting rule: **Best Short Side rule** - split by horizontal axis if the free rectangle's width is less than its height; otherwise, split by vertical axis.



The guillotine split placement process. After placing a rectangle, there are two ways to store the remaining free area.

Guillotine

Algorithm 1: The Guillotine algorithm

Initialize:

Set $F = \{(W, H)\}$;

Pack:

foreach *Item* $i = (w, h)$ *in the list of inserted items of the bin* **do**

    Decide the free rectangle $F_j \in F$ to pack the item into;

    Decide the orientation for the item and place it at the bottom-left of F_j;

    Use the guillotine split scheme to subdivide F_j into two new free rectangles F_{j_1} and F_{j_2};

    Set $F \leftarrow F \cup \{F_{j_1}, F_{j_2}\} \backslash \{F_j\}$;

    foreach *Ordered pair of free rectangles* F_{j_1}, F_{j_2} *in* F **do**

        if F_{j_1} *and* F_{j_2} *can be merge together* **then**

            F_{merge} = Merge F_{j_1} and F_{j_2};

            Set $F \leftarrow F \cup \{F_{merge}\} \backslash \{F_{j_1}, F_{j_2}\}$;

        end

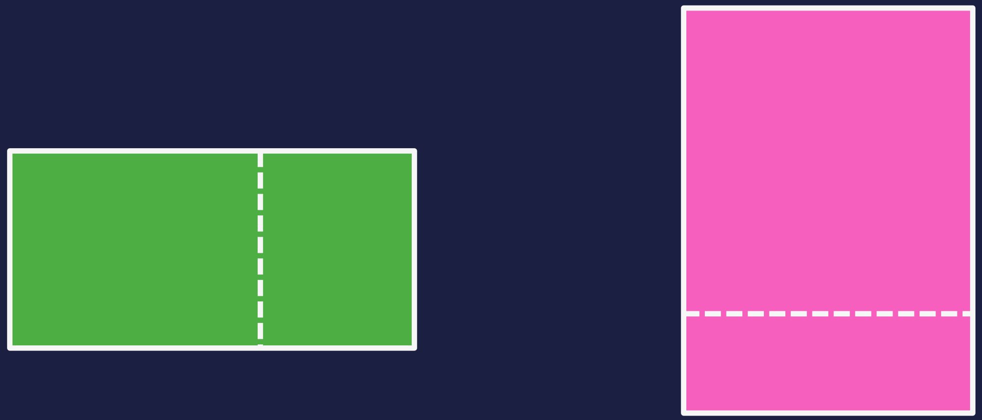
    end

end

Pseudocode for Guillotine algorithm

Guillotine

Rectangles Merging: if there exists a **pair of neighboring rectangles** and such that **U** can be exactly represented by a **single bigger rectangle**, merge these two into one.



Examples of rectangles merging

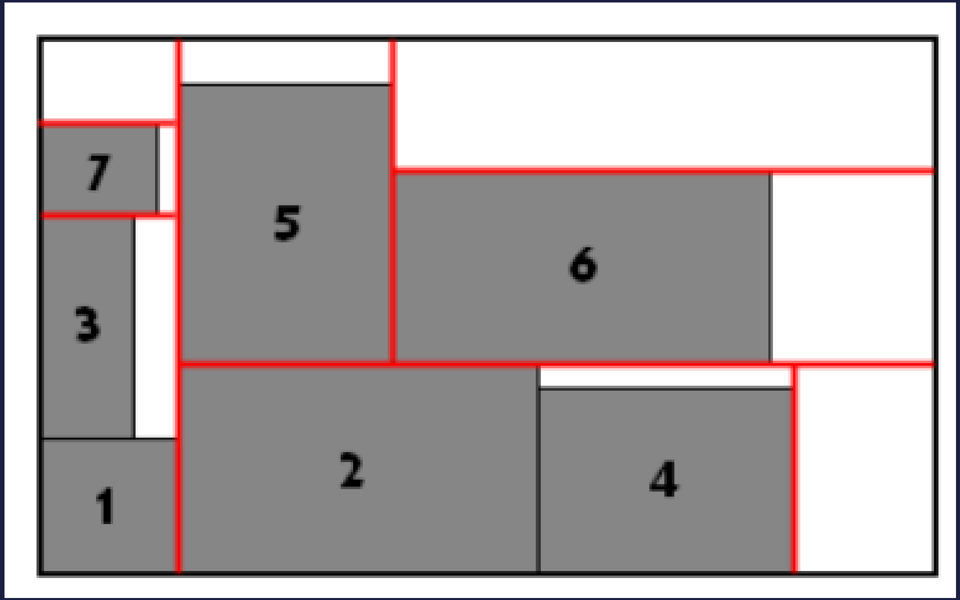
Guillotine

Algorithm 1: The Guillotine algorithm

```
Initialize:
Set F = {(W, H)};
Pack:
foreach Item i = (w, h) in the list of inserted items of the bin do
    Decide the free rectangle F_j ∈ F to pack the item into;
    Decide the orientation for the item and place it at the bottom-left of F_j;
    Use the guillotine split scheme to subdivide F_j into two new free rectangles F_j1 and F_j2;
    Set F ← F ∪ {F_j1, F_j2}\{F_j};
    foreach Ordered pair of free rectangles F_j1, F_j2 in F do
        if F_j1 and F_j2 can be merge together then
            F_merge = Merge F_j1 and F_j2;
            Set F ← F ∪ {F_merge}\{F_j1, F_j2};
        end
    end
end
```

Pseudocode for Guillotine algorithm

Guillotine



A sample packing produced by a Guillotine algorithm. The red lines denote the split choices.

Maximal Rectangles

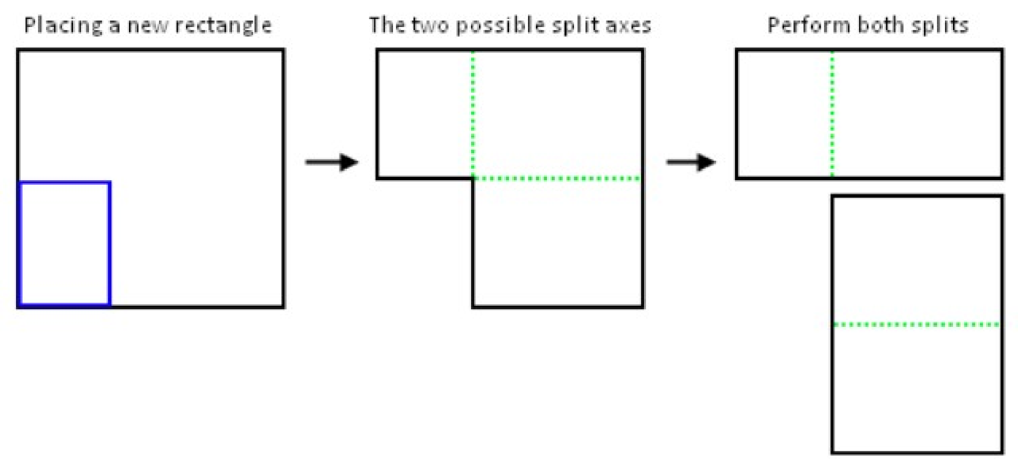
Algorithm 2: The Maximal Rectangles algorithm

Initialize:
Set $F = \{(W, H)\}$;
Pack:
foreach *Item* $i = (w, h)$ *in the list of inserted items of the bin* **do**
    Decide the free rectangle $F_j \in F$ to pack the item into;
    Decide the orientation for the item and place it at the bottom-left of F_j;
    Use the max_rec split scheme to subdivide F_j into two new free rectangles F_{j_1} and F_{j_2};
    Set $F \leftarrow F \cup \{F_{j_1}, F_{j_2}\} \backslash \{F_j\}$;
    foreach *Free Rectangles* F_j *in* F **do**
        Compute $F_j \backslash i$ and subdivided the result into at most four new free rectangles $F_{j_1}, ..., F_{j_4}$;
        Set $F \leftarrow F \cup \{F_{j_1}, ..., F_{j_4}\} \backslash \{F_j\}$;
    end
    foreach *Ordered pair of free rectangles* F_{j_1}, F_{j_2} *in* F **do**
        if F_{j_1} *contains* F_{j_2} **then**
            Set $F \leftarrow F \backslash \{F_{j_2}\}$;
        end
    end
end

Pseudocode for Maximal Rectangles algorithm

Maximal Rectangles

Splitting rule: Pick both split axes at the same time to ensure that the largest possible rectangular areas are present in the list of free rectangles.



The rectangle placement rule for the MAXRECTS data structure. Both the rectangles on the right are stored in F.

Maximal Rectangles

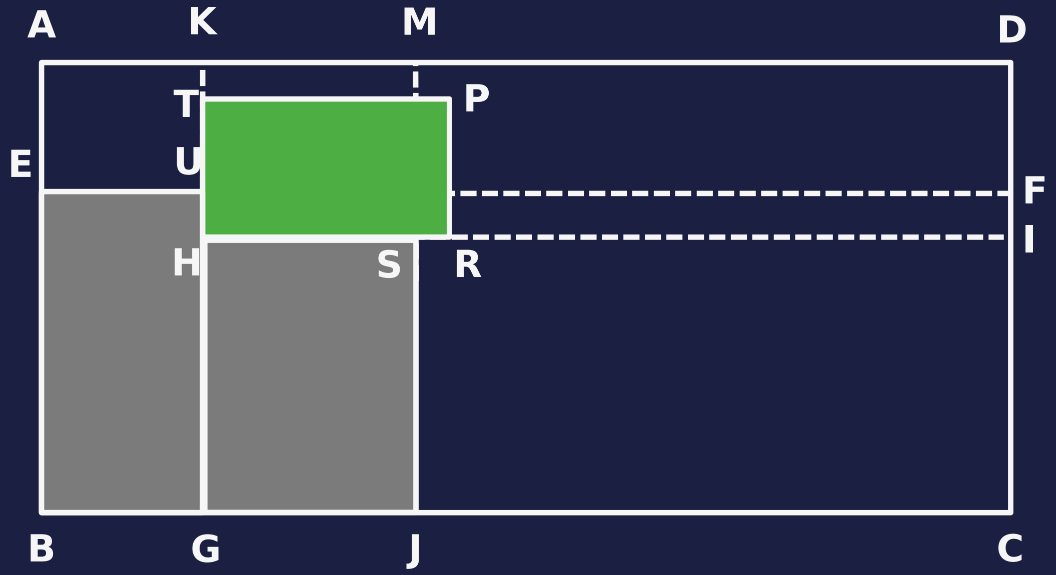
Algorithm 2: The Maximal Rectangles algorithm

Initialize:
Set $F = \{(W, H)\}$;
Pack:
foreach *Item* $i = (w, h)$ *in the list of inserted items of the bin* **do**
  Decide the free rectangle $F_j \in F$ to pack the item into;
  Decide the orientation for the item and place it at the bottom-left of F_j;
  Use the max_rec split scheme to subdivide F_j into two new free rectangles F_{j_1} and F_{j_2};
  Set $F \leftarrow F \cup \{F_{j_1}, F_{j_2}\} \backslash \{F_j\}$;
  foreach *Free Rectangles* F_j *in* F **do**
    Compute $F_j \backslash i$ and subdivided the result into at most four new free rectangles $F_{j_1}, ..., F_{j_4}$;
    Set $F \leftarrow F \cup \{F_{j_1}, ..., F_{j_4}\} \backslash \{F_j\}$;
  end
  foreach *Ordered pair of free rectangles* F_{j_1}, F_{j_2} *in* F **do**
    if F_{j_1} *contains* F_{j_2} **then**
      Set $F \leftarrow F \backslash \{F_{j_2}\}$;
    end
  end
end

Pseudocode for Maximal Rectangles algorithm

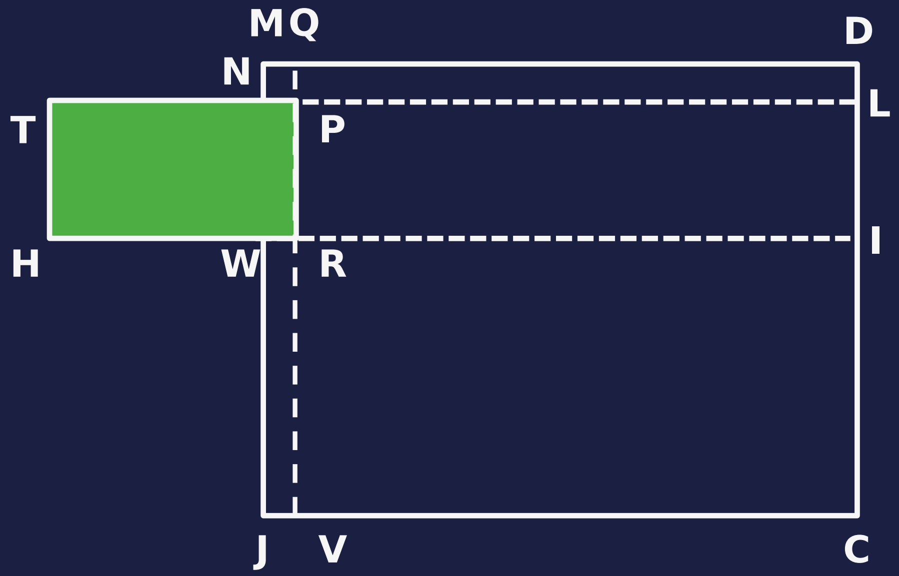
Maximal Rectangles

Any **free rectangle** that **intersects** the area occupied by the **newly inserted item** is **split** such to remove the intersection.



Maximal Rectangles

Any **free rectangle** that **intersects** the area occupied by the **newly inserted item** is **split** such to remove the intersection.



New rectangles:

MDLN; QDCV; WICJ

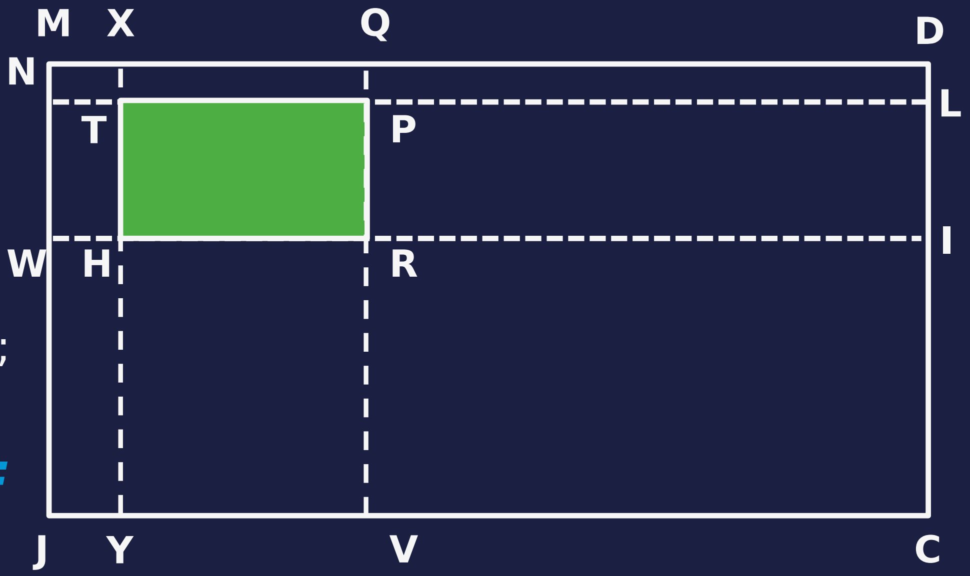
Maximal Rectangles

Any **free rectangle** that **intersects** the area occupied by the **newly inserted item** is **split** such to remove the intersection.

New rectangles:

MDLN; QDCV; WICJ

Another one: MXYJ



Maximal Rectangles

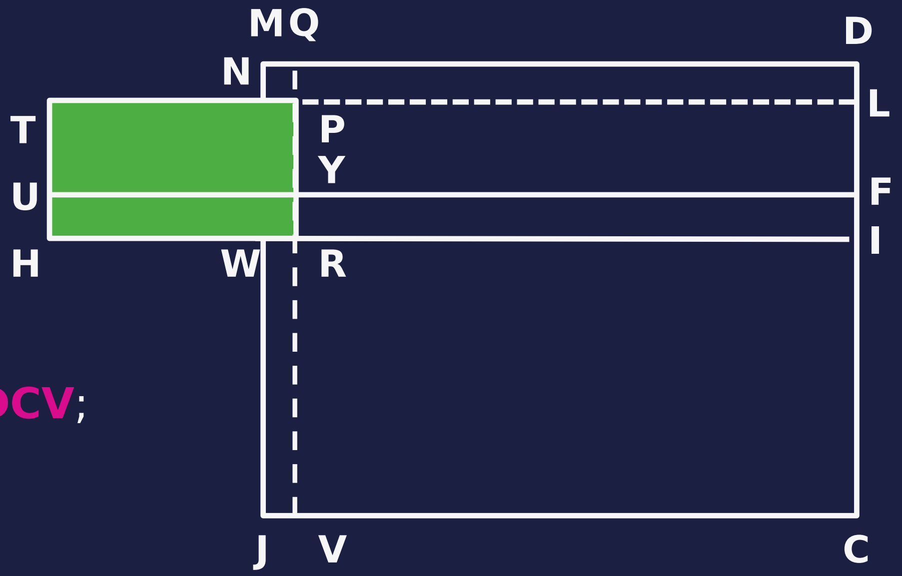
Algorithm 2: The Maximal Rectangles algorithm

Initialize:
Set $F = \{(W, H)\}$;
Pack:
foreach *Item* $i = (w, h)$ *in the list of inserted items of the bin* **do**
    Decide the free rectangle $F_j \in F$ to pack the item into;
    Decide the orientation for the item and place it at the bottom-left of F_j;
    Use the max_rec split scheme to subdivide F_j into two new free rectangles F_{j_1} and F_{j_2};
    Set $F \leftarrow F \cup \{F_{j_1}, F_{j_2}\} \backslash \{F_j\}$;
    foreach *Free Rectangles* F_j *in* F **do**
        Compute $F_j \backslash i$ and subdivided the result into at most four new free rectangles $F_{j_1}, ..., F_{j_4}$;
        Set $F \leftarrow F \cup \{F_{j_1}, ..., F_{j_4}\} \backslash \{F_j\}$;
    end
    foreach *Ordered pair of free rectangles* F_{j_1}, F_{j_2} *in* F **do**
        if F_{j_1} *contains* F_{j_2} **then**
            Set $F \leftarrow F \backslash \{F_{j_2}\}$;
        end
    end
end

Pseudocode for Maximal Rectangles algorithm

Maximal Rectangles

Delete every **free rectangle** which is **fully overlapped by others** in the list.

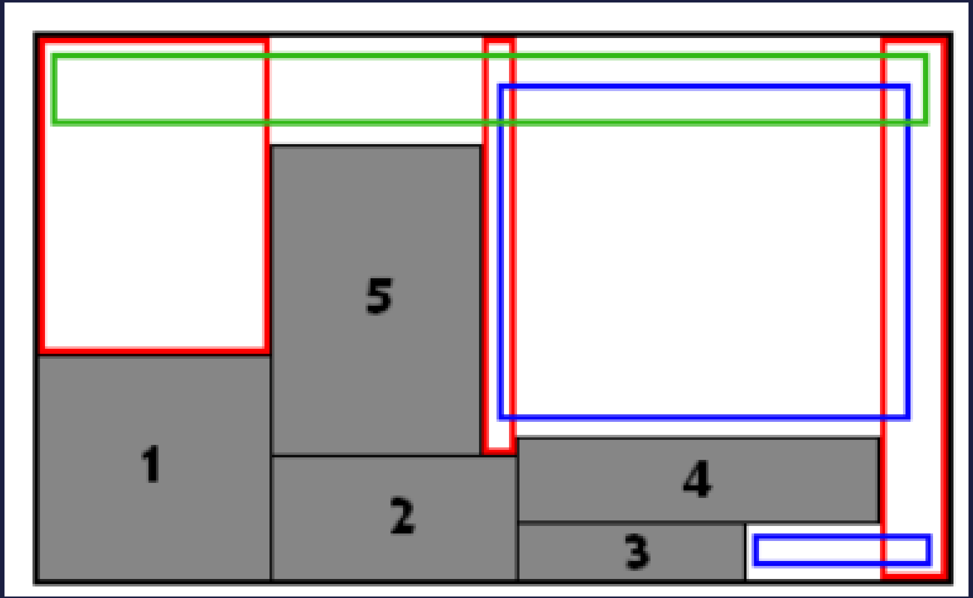


New rectangles:

MDJC {MDLN; **QDCV**; WICJ}

UFIH {**YFIR**}

Maximal Rectangles



A sample packing produced by the MAXRECTS algorithm. The maximal rectangles of F are shown in colors.

Analysis

04

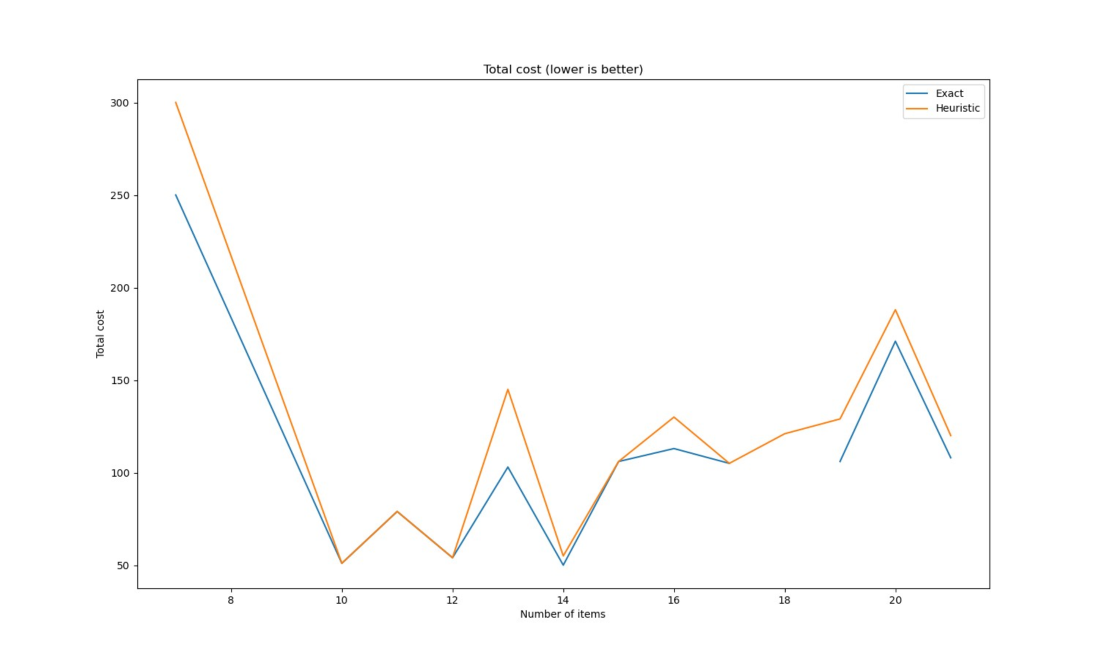
Exact solution

RESULTS

F: *Feasible Solution*
N/A: *No Solution Found*

Test	Input sizes		Exact methods						Heuristics				
			CP 1		CP 2		MIP						
	n_packs	n_bins	f	t(s)	f	t(s)	f	t(s)	f_min	f_max	f_avg	std_dev	t_avg(s)
0007.txt	7	3	250	0.027932056	250	0.029741828	250	3.176666667	300	300	300	0	0.000029500
0010.txt	10	10	51	0.049313393	51	0.082288480	51	2.642000000	51	51	51	0	0.000037000
0011.txt	11	11	79	0.053943779	79	0.194204638	79	14.00300000	79	79	79	0	0.000035500
0012.txt	12	12	54	0.057868931	54	0.088900393	54	7.898000000	54	54	54	0	0.000040500
0013.txt	13	13	103	0.109191075	103	0.248680102	F	F	145	145	145	0	0.000039000
0014.txt	14	14	50	0.218952388	50	0.156991295	50	27.73466667	55	55	55	0	0.000051000
0015.txt	15	15	106	0.513134012	106	0.859766784	F	F	106	106	106	0	0.000057500
0016.txt	16	16	113	0.905138111	113	0.434518921	F	F	130	130	130	0	0.000049000
0017.txt	17	17	105	117.0229775	105	48.88790709	F	F	105	105	105	0	0.000052500
0018.txt	18	18	F	F	F	F	F	F	121	121	121	0	0.000059000
0019.txt	19	19	106	5.214479066	106	4.515408114	F	F	129	129	129	0	0.000058500
0020.txt	20	20	171	2.519827466	171	3.259184459	F	F	188	188	188	0	0.000053500
0021.txt	21	21	108	8.500796449	108	13.22533175	F	F	120	120	120	0	0.000060000

Exact solution



Exact solution

- **CP model** gives **exact solutions** for tests with **sizes**: 7 x 3, 10 x 10, 11 x 11, 12 x 12, 13 x 13, 14 x 14, 15 x 15, 16 x 16, 17 x 17, 19 x 19, 20 x 20, 21 x 21.
- **MIP model** gives **exact solutions** for test with **sizes**: 7 x 3, 10 x 10, 11 x 11, 12 x 12, 14 x 14.

General

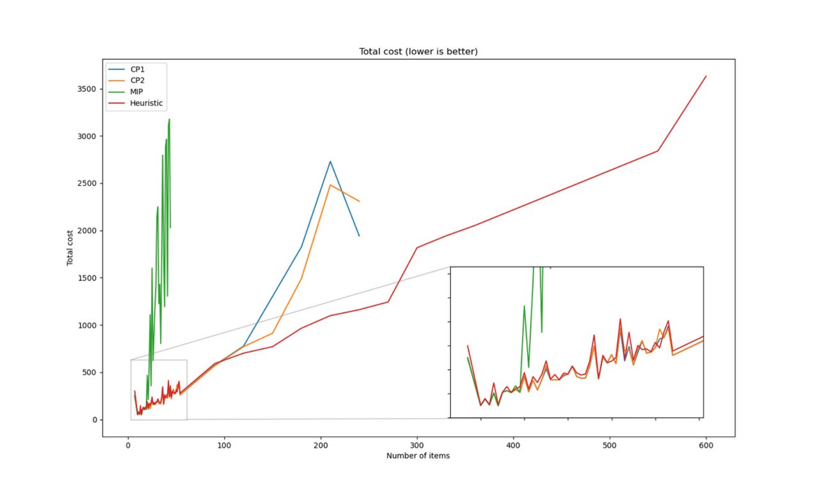
- **CP cannot** handle data sets **larger than 240 x 240**.
- **MIP cannot** handle data sets **larger than 44 x 44**.
- **Heuristic can** handle **all test cases** (the largest test size is 10,000 x 10,000).

Compare Total cost

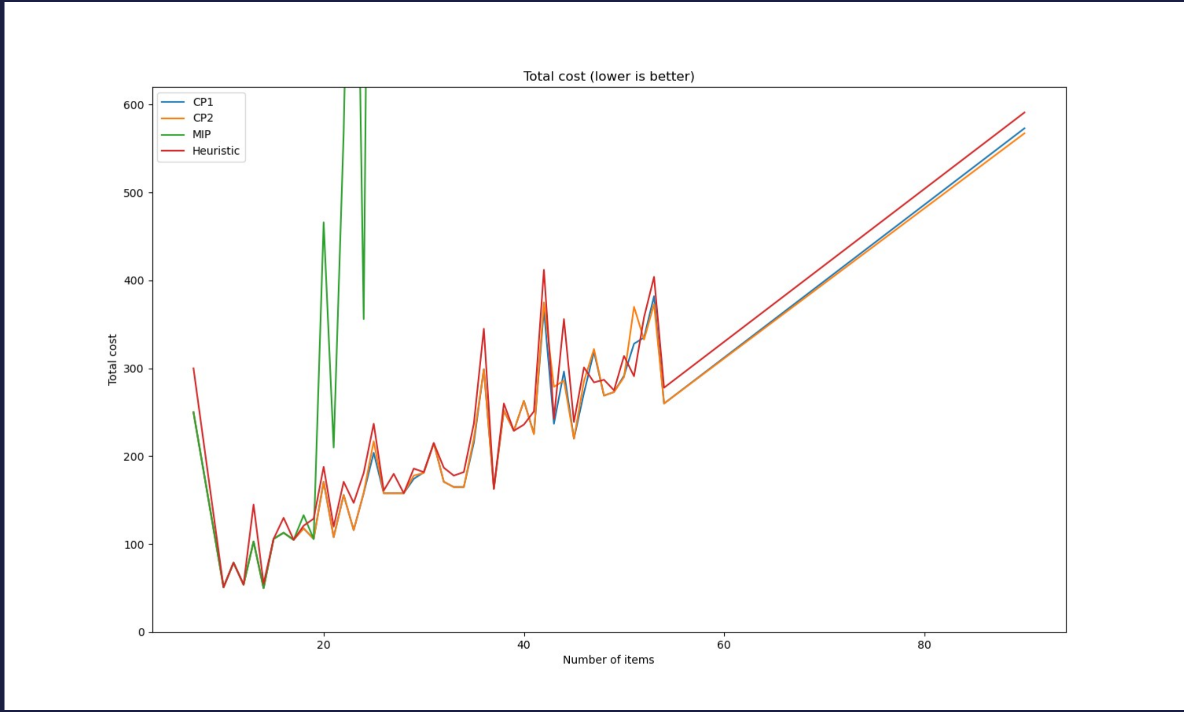
- **MIP** gives the **worst** results.

- **CP1 and CP2** give **nearly equivalent** results, but with **larger data** sets, **CP2** gives **better** results.

- **Heuristic** gives **really good** results. With tests of size **smaller than 100 x 100**, it is still **a bit inferior to CP**, but for **all other tests**, it is **significantly better**.



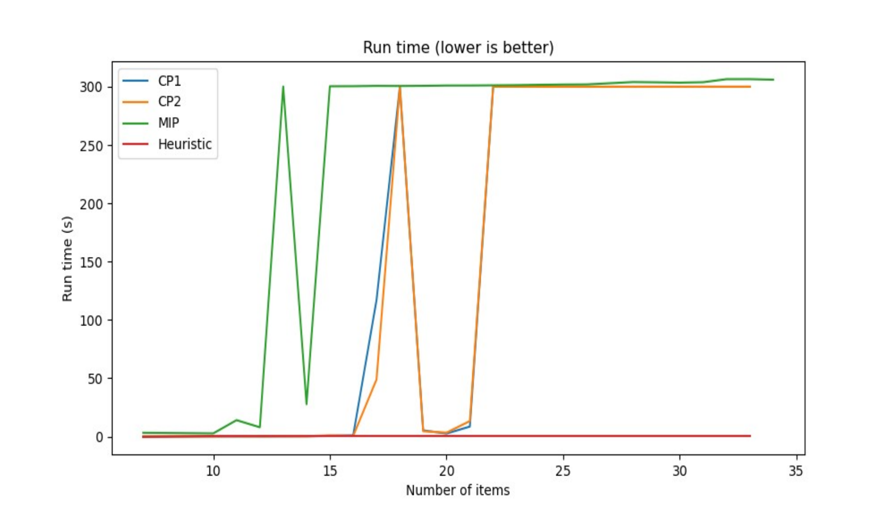
Compare Total cost



Zoomed comparing total cost graph

Compare Running Time

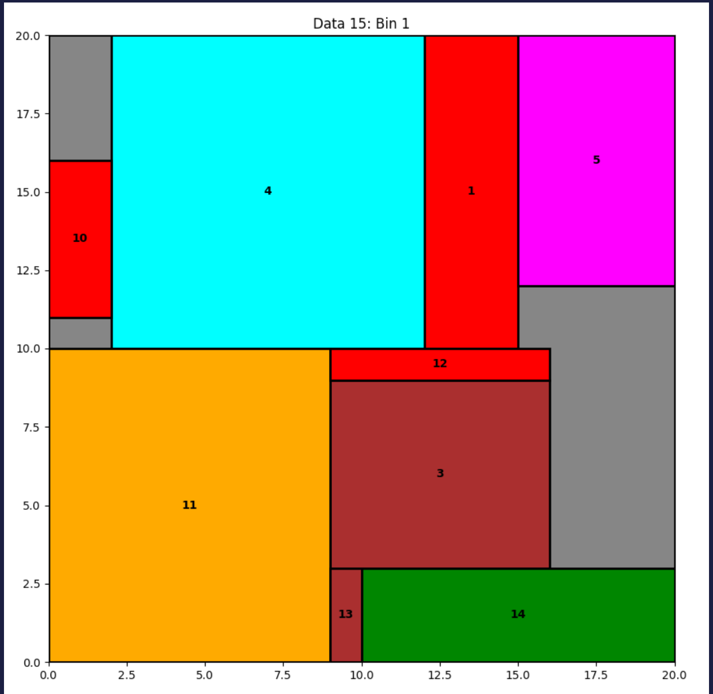
- **MIP** reaches the **time limit** of 300 seconds for all tests with **size ≥ 15 x 15**.
- **CP** reaches the **time limit** of 300 seconds for all tests with **size ≥ 22 x 22**.
- **Heuristic** has a **very short run time**, every test is under 1 second, even for the test size of 10,000 x 10,000.



Conclusion

- **MIP** is **not good** in terms of results and running time.
- **CP1** is **better than MIP** with better results and faster running time (in some early tests).
- **Heuristic** gives **the best** results in both cost and running time.

Visualizing



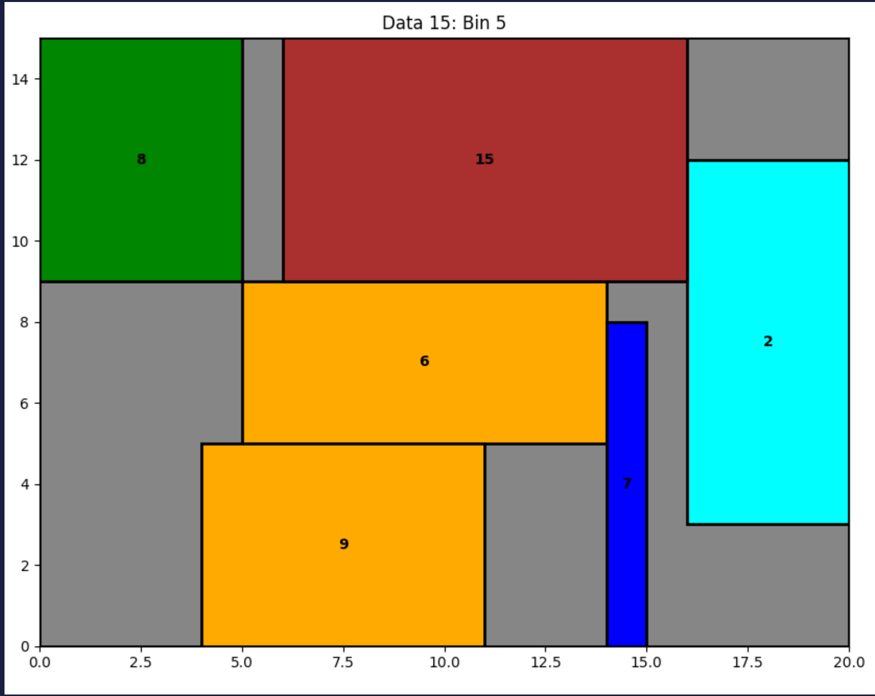
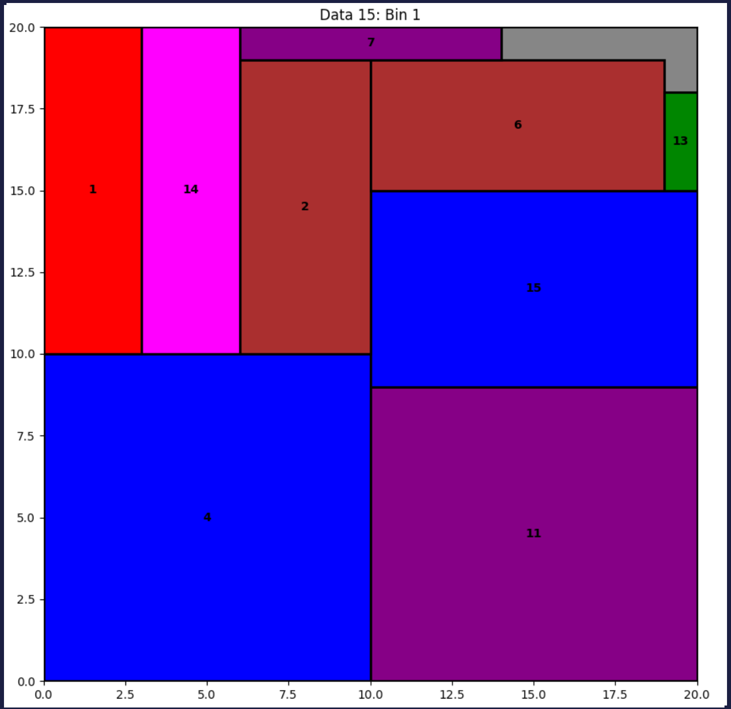


Figure generated for output of CP solver with test 0015.txt

Visualizing



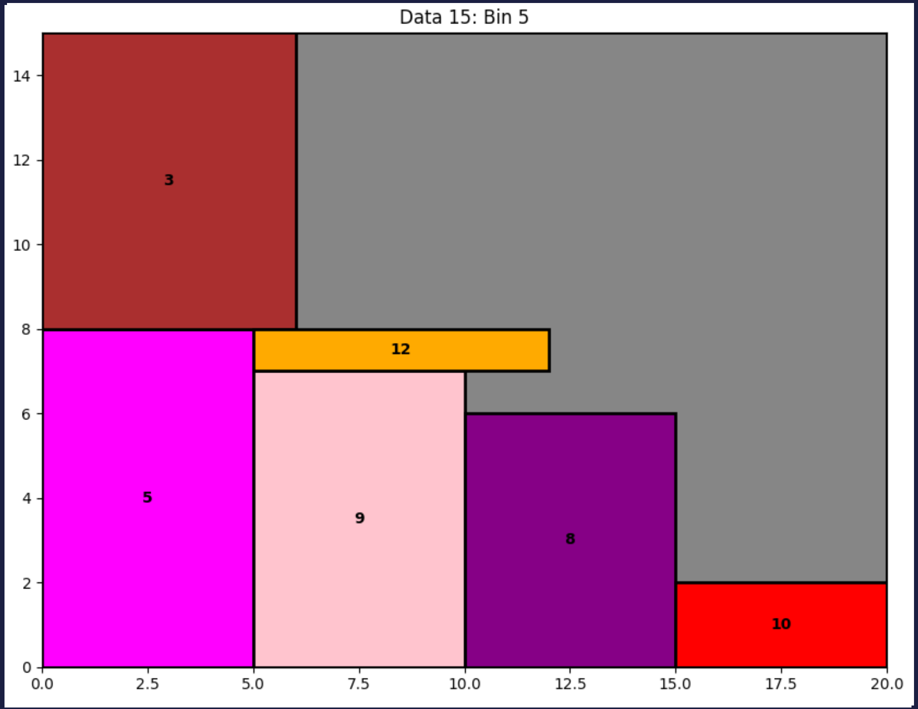


Figure generated for output of Heuristic solver with test 0015.txt

Multiple-type, two-dimensional finite bin packing problem

For more information, please visit us in:
https://github.com/phannhat17/mini-project-optimization/

CYBER SECURITY

Thanks for your attention!