Palletizing with adjacency constrain

The objective is to implement an optimization model for the palletizing problem extended by **constraints**. Palletization refers to the process of arranging and organizing products or goods on a pallet in an efficient and effective manner, the main objective of palletization is to maximize space utilization.

The extended version has constraints in the form of two lists:

1. a list of pairs of element which have to be adjacent in the arrangement
2. a list of pairs of element which cannot be adjacent in the arrangement

The ***benchmarks/data*** directory contains the files with the data for the problem. The data files have the following format:

```
3   5
1   4
2   6
4   7
10  5
3   6

0   1
5   4
3   2

1   2
```

The files must be divided into three sections, each section is separated by a blank line **\n**:

- The first section contains the dimensions of each element. The first number is the width and the second is the height.
- The second section contains the index of pairs of elements which have to be adjacent.
- The third section contains the index pairs of elements which cannot be adjacent.

Thus, the dimensions of elements will be:

- **[(3, 5), (1, 4), (2, 6), (4, 7), (10, 5), (3, 6)]**

the adjacency elements will be:

- **[(0, 1), (5, 4), (3, 2)]**

and the non-adjacency elements will be:

- **[(1, 2)]**

Note: the adjacency and non-adjacency elements are indexed from 0, where 0 is the first element in the list and so on.

Solution

It is possible to solve this problem in two ways:

1. The basic one, without constraints
2. The extended one, with constraints

Solve the problem without constraints

To solve this problem without constraints, it's only necessary to have the **two** possible positions of each shape (horizontal and vertical) and then add that constraint to the model as in the following way:

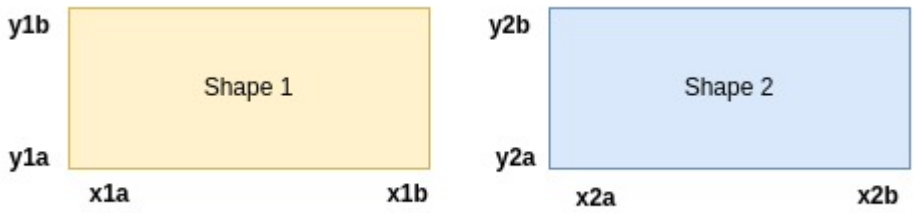
```
model.AddNoOverlap2D(horizontal, vertical)
```

Solve the problem with constraints

To solve this problem with constraints, several constraints were added to the model.

Given two shapes with the initial and final coordinates like:

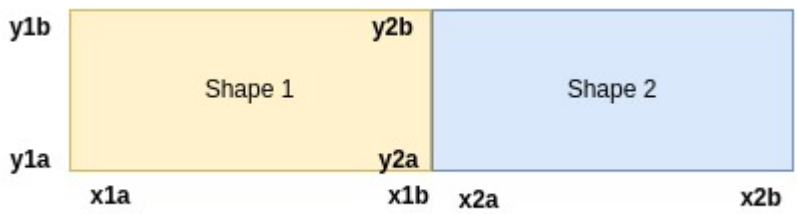
- shape1: [(x1a, y1a), (x1b, y1b)]
- shape2: [(x2a, y2a), (x2b, y2b)]



Adjacent shapes

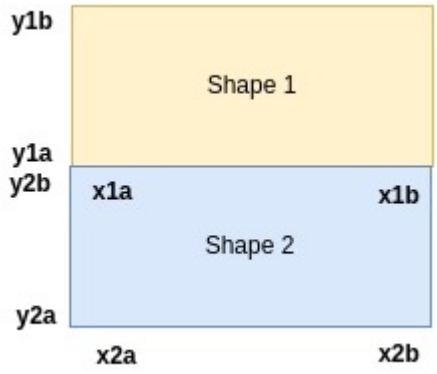
To these shapes be adjacent, one of the following conditions must be met:

- x1b == x2a

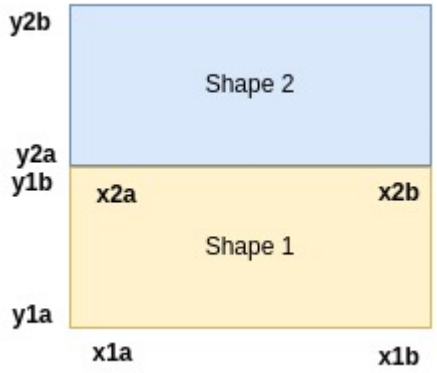


- x1a == x2b

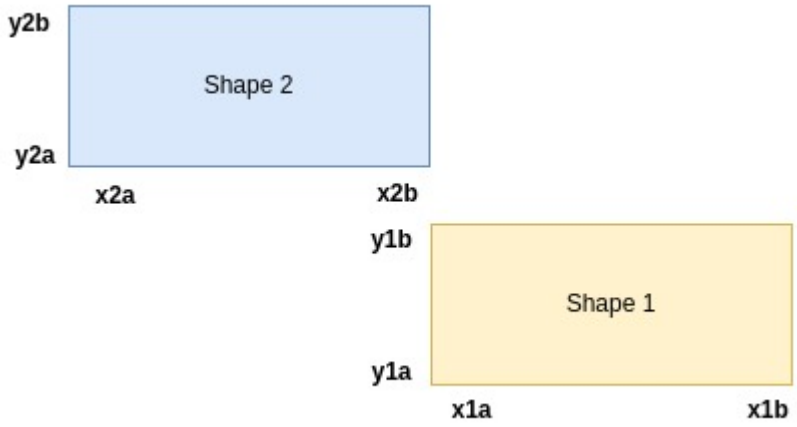
- y2b == y1a



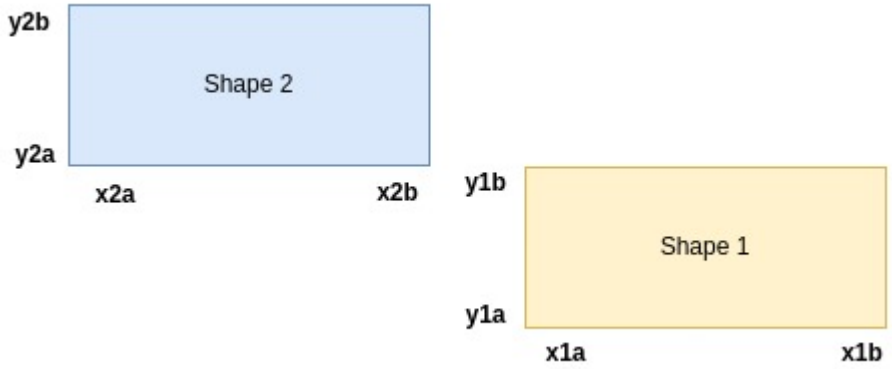
- y1b == y2a



Despite the above-mentioned conditions, they are insufficient because circumstances such as those shown below may arise.



In this example **x2b==x1a** (second condition), however the shapes are not adjacent



Regarding the picture above, **y1b==y2a** (fourth condition) and again the shapes are not adjacent.

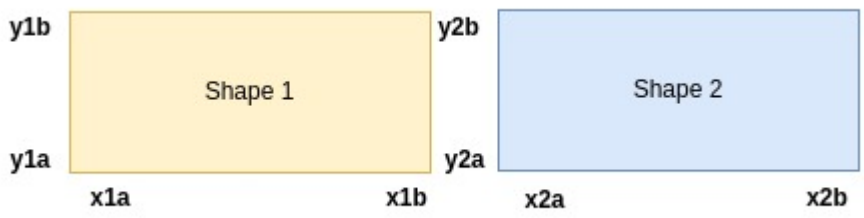
Thus, stricter conditions need to be added to the model:

- **x1b == x2a ^ y2b > y1a ^ y2a < y1b**
- **x1a == x2b ^ y2b > y1a ^ y2a < y1b**
- **y2b == y1a ^ x2b > x1a ^ x2a < x1b**
- **y1b == y2a ^ x2b > x1a ^ x2a < x1b**

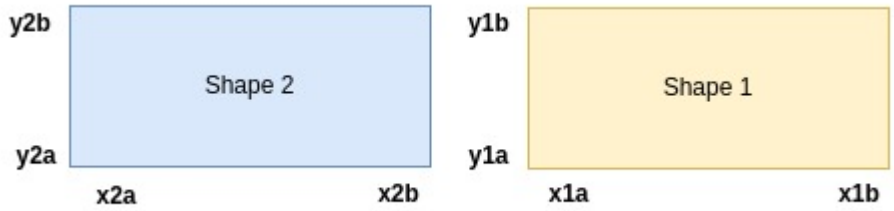
Non Adjacent shapes

To these shapes be non adjacent, one of the following conditions must be met:

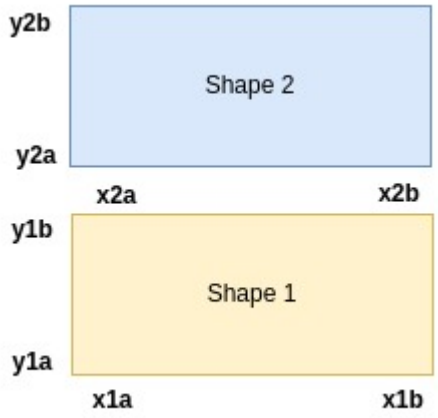
- **x1b < x2a**



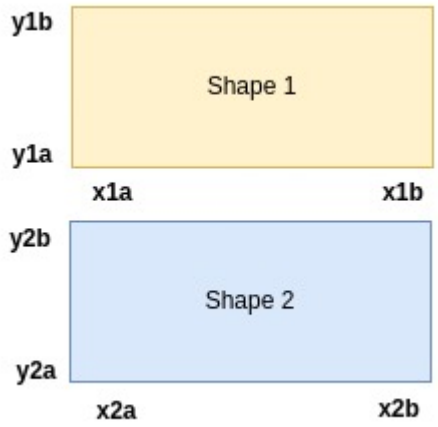
- **x2b < x1a**



- **y1b < y2a**



- **y2b < y1a**



In this case the conditions are so strict that it is not necessary to add more conditions to the model.

Results

The solutions are stores in the ***benchmarks/*** directory and inside the folder with the same name as the file used to run the program. Inside this folder, two directories are created, one for the solution without constraints and the other for the solution with constraints. Each directory contains two files:

- The first one contains the position of each element in the arrangement, the area of the arrangement, the max values of **X** and **Y** coordinates, the **area** and the **occupation rate**.
- The second one is an image of the arrangement. In this example the file02.txt contains the following data:

```
10  10
5   4
5   8
9   8
2   1
11  9
17  20
14  9
4   9

0   6
5   4

2   3
7   8
1   2
6   7
3   7
1   8
```

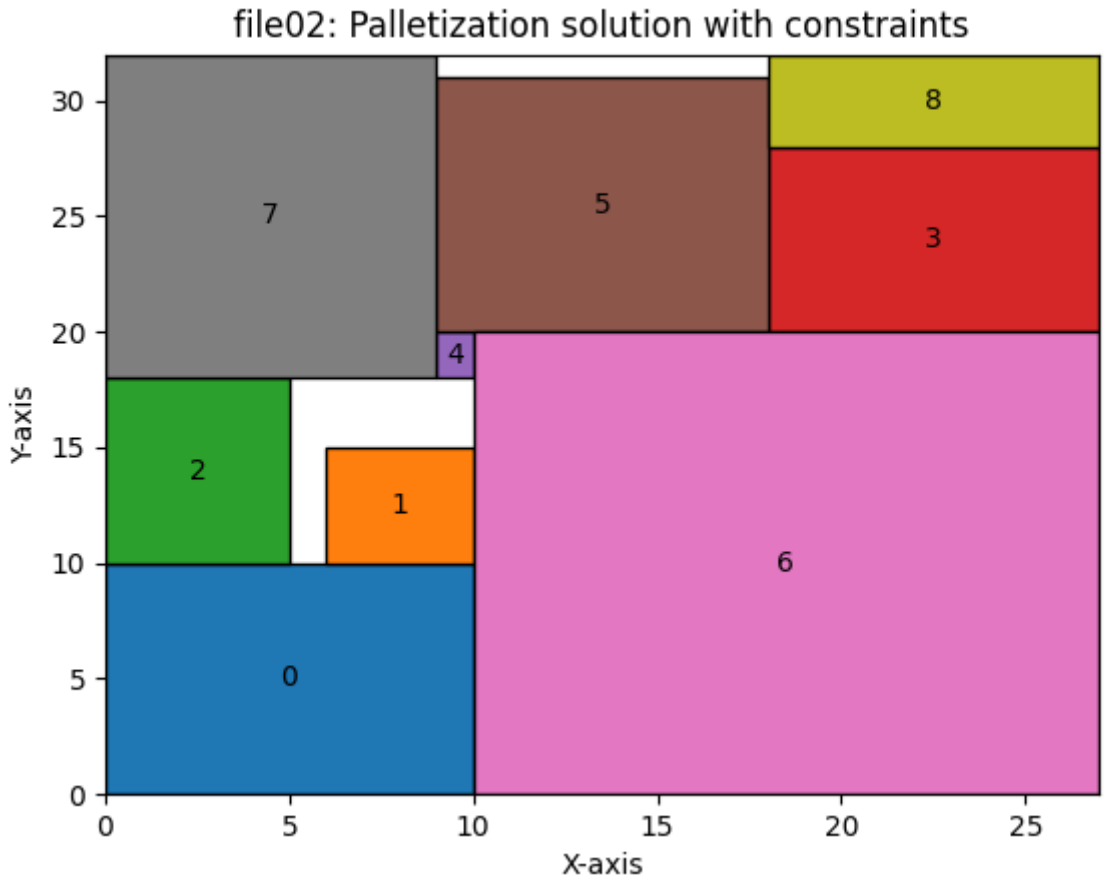
So, it means 9 objects are presented in this arranjement. The object **number 0** and the object **number 6** must be adjacent in the solution, as well as the objects **number 5** and **number 4**. At the same time the objects presented in the next list of two pairs must not be adjacent:

Non-adjacent pairs:

(2, 3)
(7, 8)
(1, 2)
(6, 7)
(3, 7)
(1, 8)

Comparing solution with and without constraints

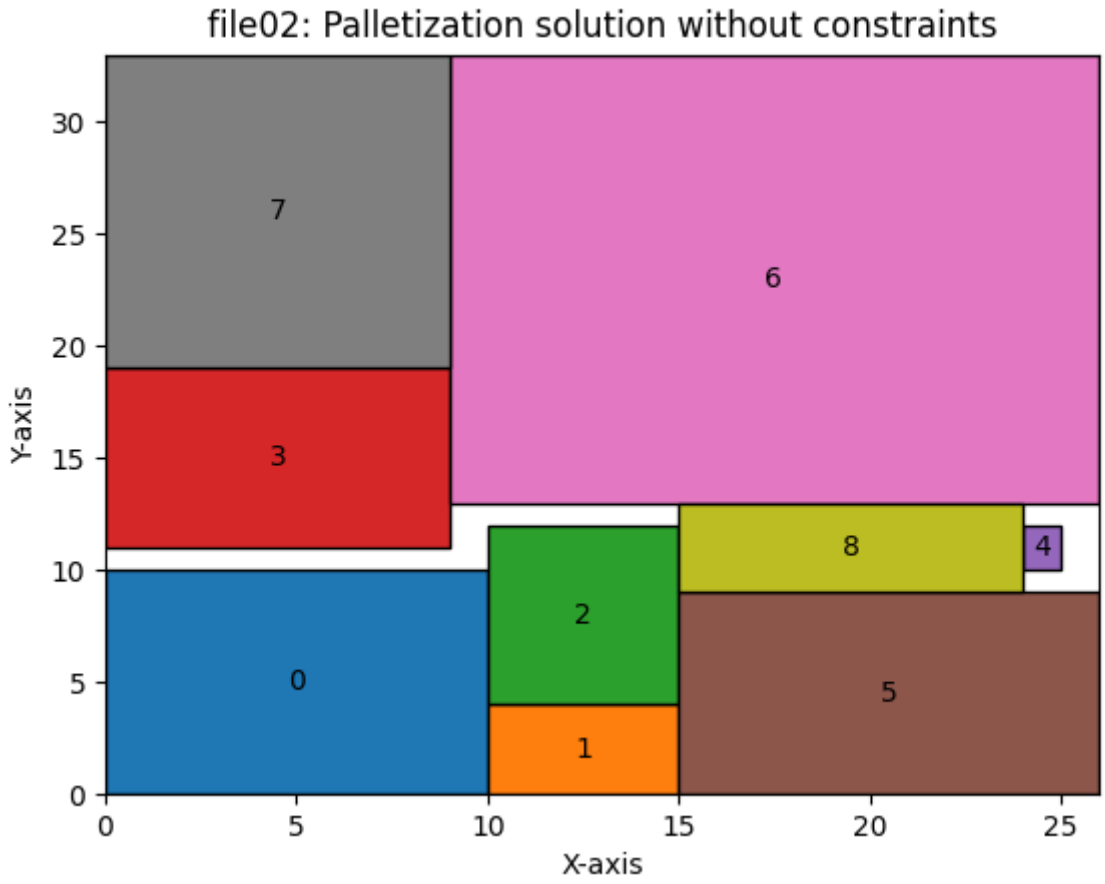
With constraints solution



As it is possible to see, the adjacent pairs are, indeed adjacents:
(0,6)
(5,4)

and all pairs present in the non-adjacency list comply with this same non-adjacency condition.

Without constraints solution



As it's possible to see, in this solution, the pairs:
(0, 6)
(5,4)

are not adjacent, unlike the previous solution where this condition had to be met.

Regarding the non adjacency contraint, the following pairs are presented in the non adjacency list, however in this solution they are adjacents.

(1, 2)
(6, 7)
(3, 7)

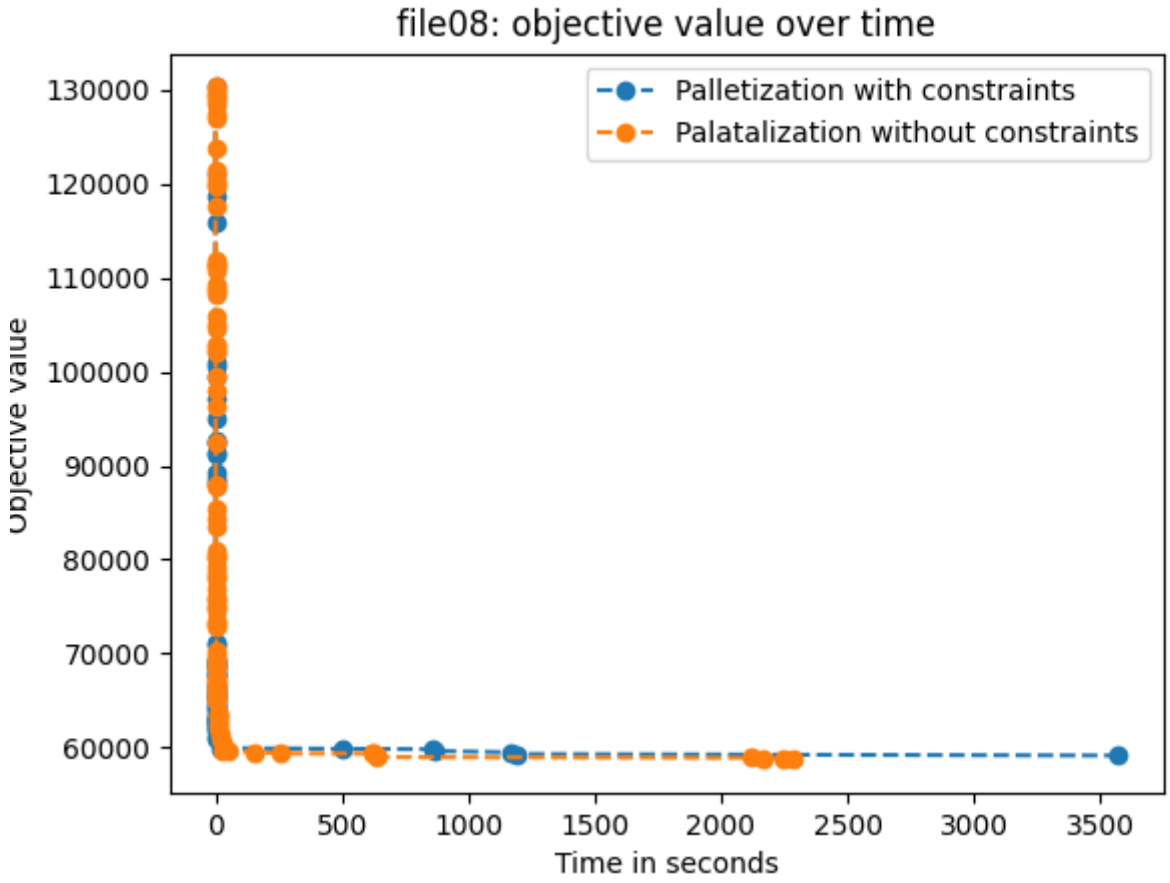
Optimization progress

Inside the directory with the same name as the file used to run the program, there is an image that shows the optimization progress over time for the solution **with** and **without** constraints.

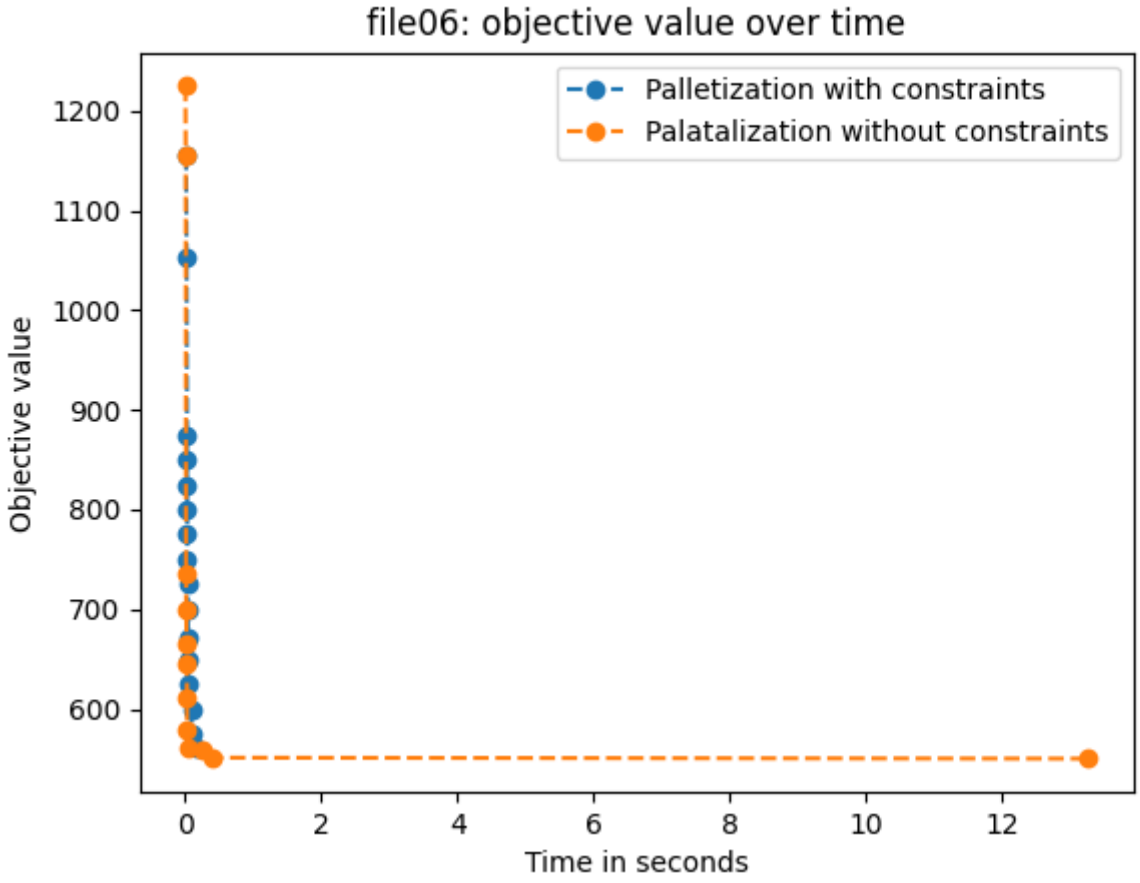
Behaviour of objective function depending of the applied constraints

Constraints directly influence the objective function by restricting the set of feasible solutions. The objective function represents the goal which in this example represents minimizing the area. When constraints are introduced, the feasible solution space is reduced, and the objective function is evaluated only for the solutions that satisfy all constraints. This can take more time, once more calculations are required.

In this example, as you can see the best solution with constraints took more time to be found than the solution without constraints. So in general, **the more constraints** the problem has, **the higher the time** necessary to solve the problem.



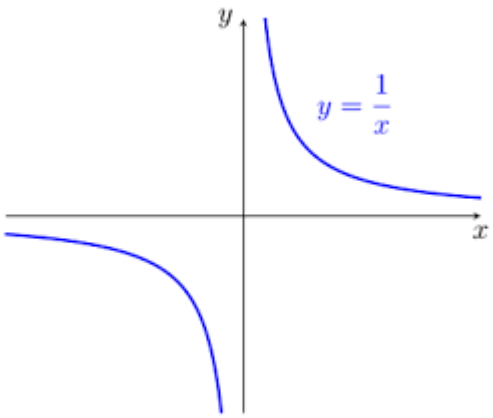
However, in the following case the solution without constraints took more time to be found and it's worse in terms of value, than the solution with constraints.



Graph analyse

As it is possible to analyse from the images, the value returned from **Objective Function** is maximum in the beggining, but it is also in the beggining that it find more number of solutions and as the time goes by the number of solutions is lower, but the solution value is higher.

This graph has the same behaviour as the positive part of inverse function, shown in the follow image.



How to run the program

It is possible to run the program with a file created by your own, you just have to put the correct path, to the file, like in the command below:

```
python3 palletizationWithConstraints.py -f benchmarks/data/file01.txt
```

If you don't want to create your own file, you can use one of the files in the ***benchmarks/data*** directory. The command above will run the program with the file ***file01.txt***.

Results with my files

	With Constraints		Without contraints	
File	**Area**	**Occupation (%)**	**Area**	**Occupation (%)**
file00.txt	532	97.6	527	98.5
file01.txt	130	97.7	130	97.7
file02.txt	864	96.6	858	97.3
file03.txt	15158	98	15000	99
file04.txt	5047	97	4998	98
file05.txt	1950	99.3	1943	99.7
file06.txt	560	97.1	550	98.9
file07.txt	4416	**97.9**	4450	**97.2**
file08.txt	59118	97.7	58680	98.5
file09.txt	110	100	110	100

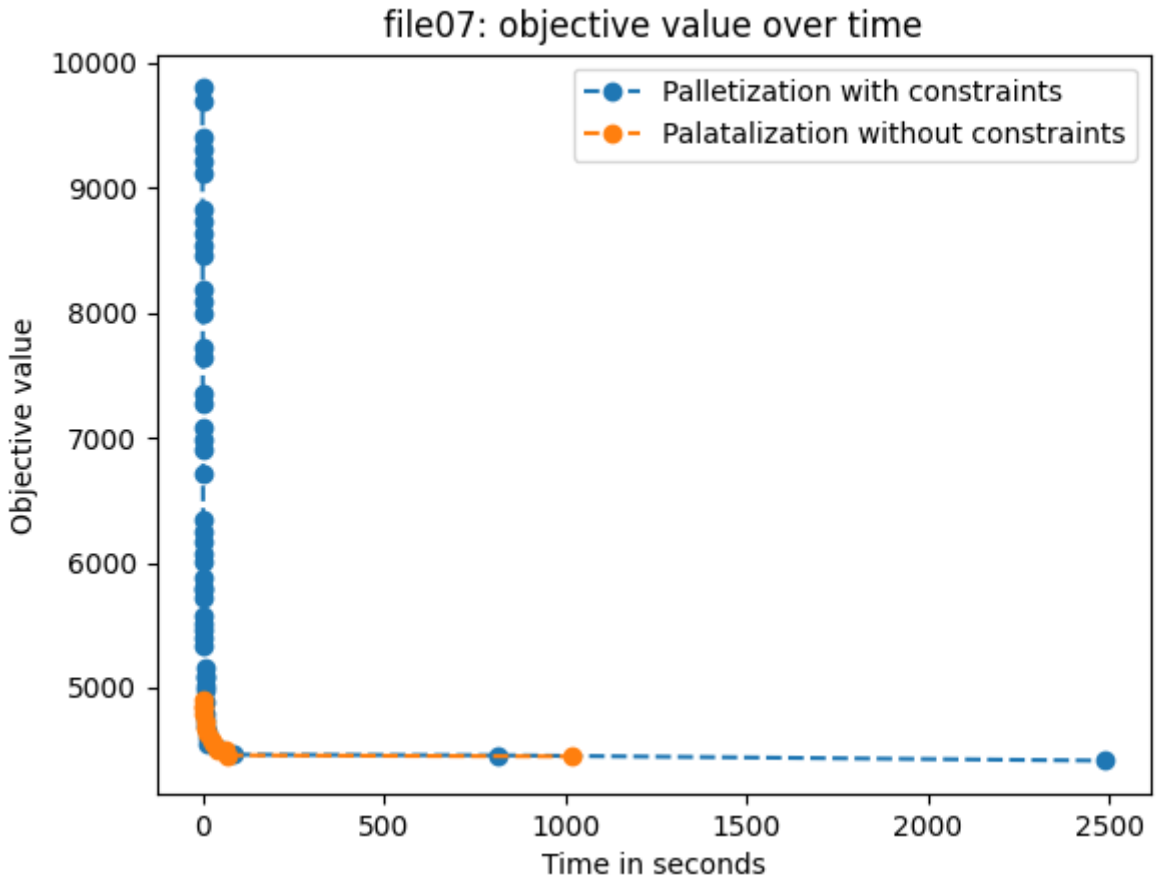
To test the developed solution, 10 files were created with different data. The results are presented in the table below.

I'd like to notice that for file09.txt, the solution with and without constraints found was the perfect one, once the occupation percentage is 100%.

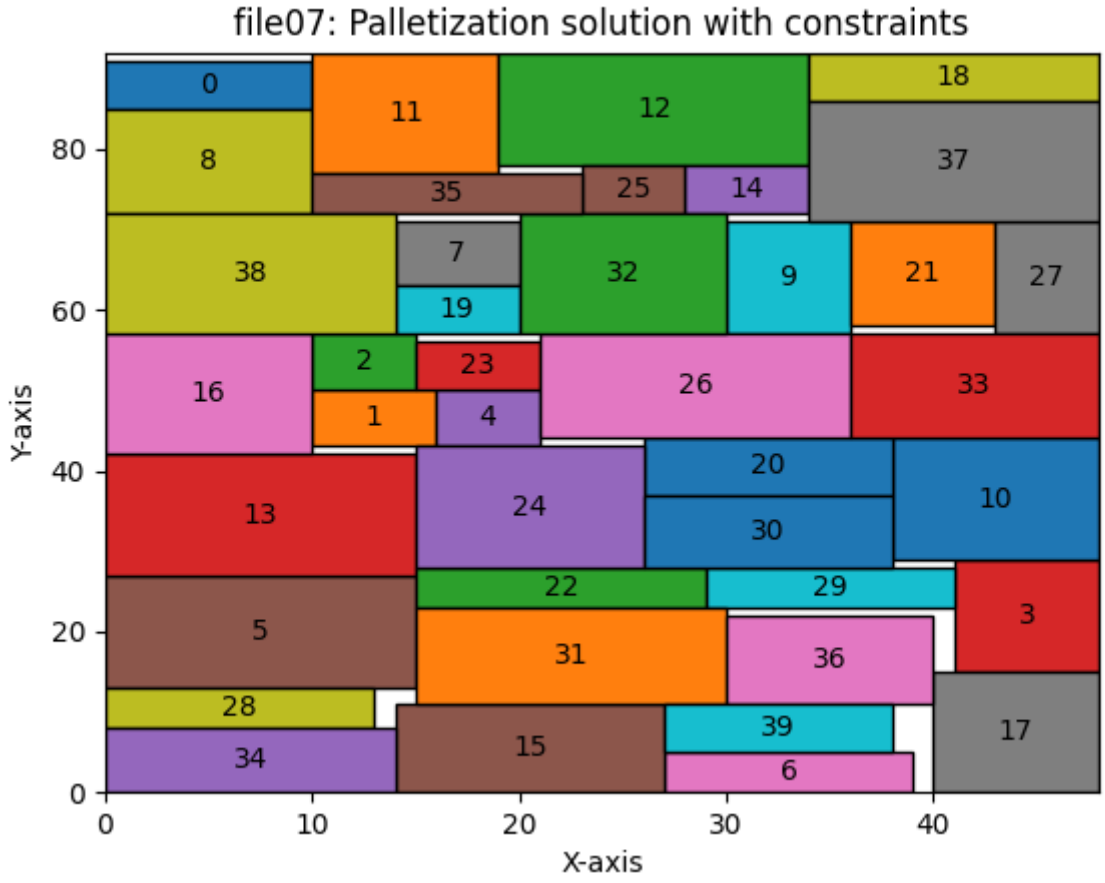
The maximum percentage difference between the solution with and without constraints is **1.8%** and it takes place to file06.txt, thus it's possible to conclude that the implementation of this algorithm produces good results.

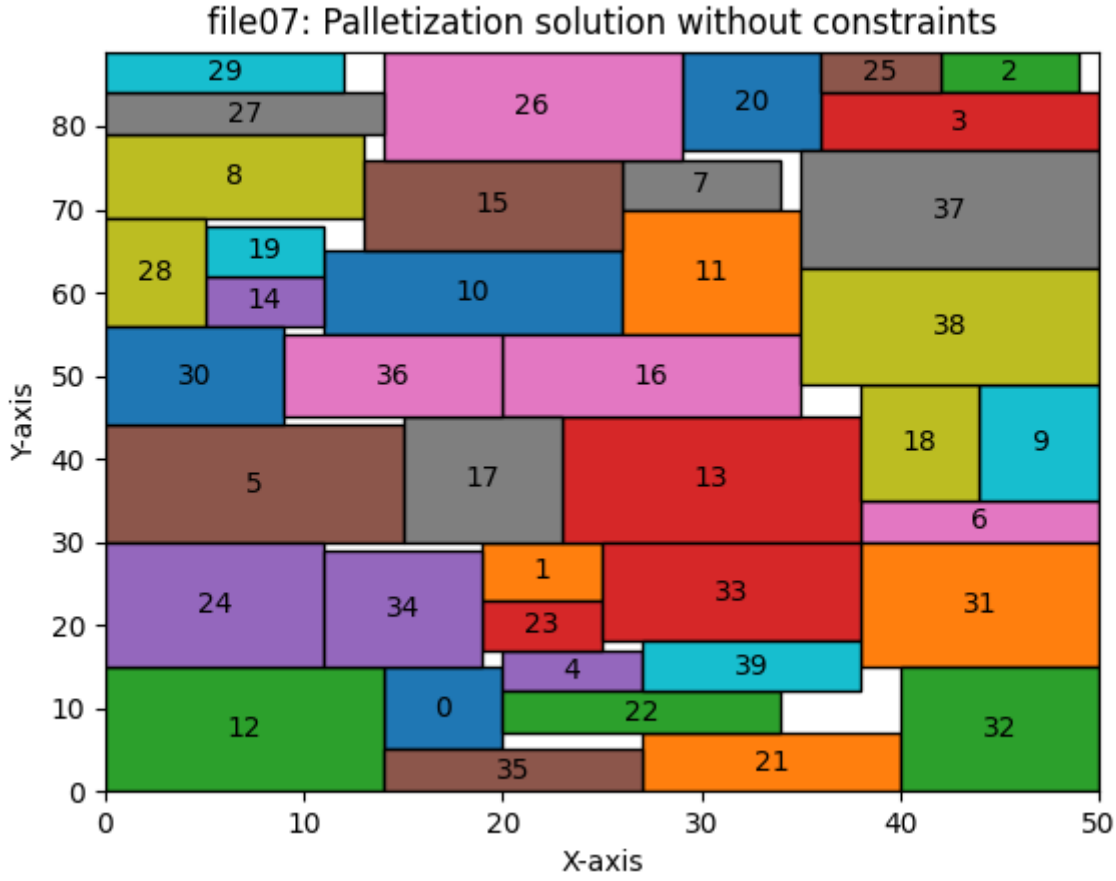
To almost all files tested, the solution without constraints is better, regarding the occupied space, than the solution with constraints, but regarding file07.txt, as you can see in the table the solution with constraints is better than the solution without constraints, what was unexpected once the outcomes of all other files

are better in the solution without constraints than in the solution with constraints.



As you can see the best solution with constraints was found near **2500 seconds**, approximately **42 minutes**, while the solution without constraints was found near **1000 seconds** (+/- 17 minutes). Once the solver time is limited to **one** hour, it suggests that it most likely did not have enough time to develop a better solution than the constraint solver.


file07: Palletization solution with constraints
Y-axis
X-axis
0
11
12
18
8
37
35
25
14
38
7
32
9
21
27
19
16
2
23
26
33
1
4
20
13
24
10
30
22
29
3
5
31
36
28
39
17
34
15
6


file07: Palletization solution without constraints
Y-axis
X-axis
29
25
2
27
26
20
3
8
15
7
37
28
19
11
14
10
38
30
36
16
18
9
5
17
13
6
24
34
1
33
31
23
4
39
12
0
22
32
35
21

Although it seems that the unconstrained solution is better, in reality it is not, since the axis boundaries are not the same in both images.

Note
More results can be found inside benchmarks directory.

Conclusions

After discussing the results, it's possible to state that the solution without constraints produces better outcomes than the results with constraints, furthermore, the time required to solve the problem without constraints is, in general, less than solving the same problem with constraints.

Author

Work done by João Farias
Politechnika Rzeszowska, June 2023.