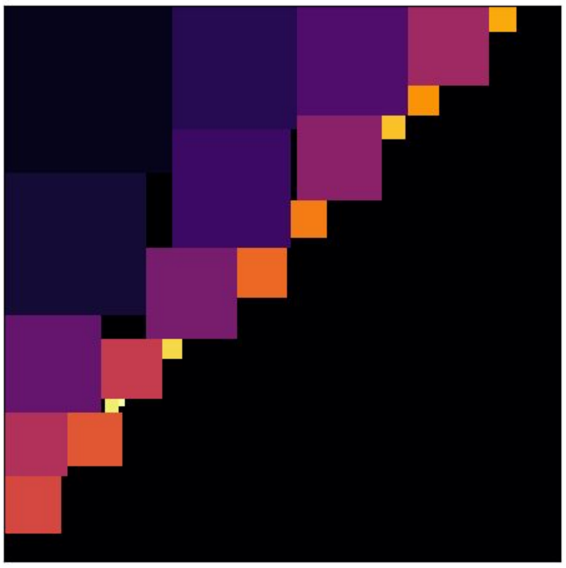
Strategies for solving the Perfect Square Placement Problem

I. Greedy approach

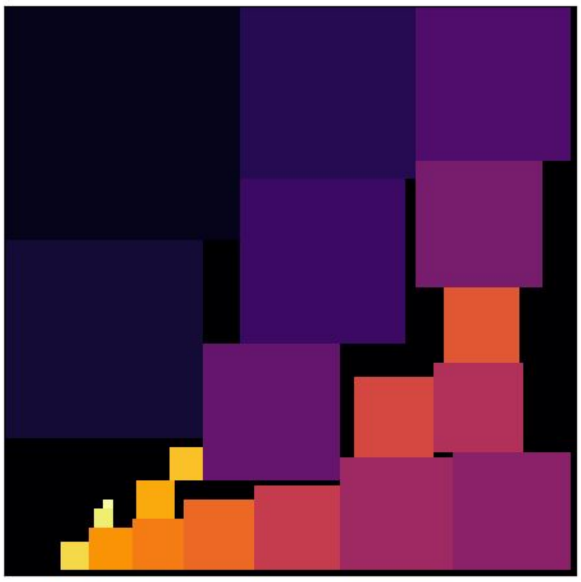
In order to optimize our previous implementation of the greedy method, we analysed the final placement of the squares.



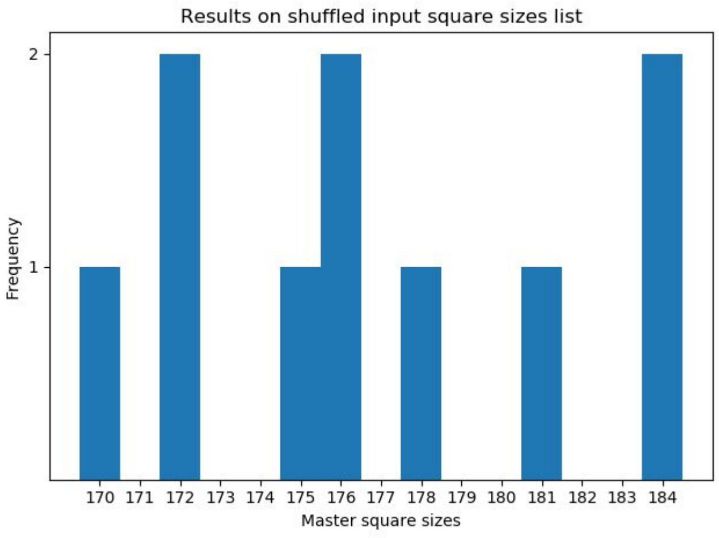
It can be observed that our way of traversing the grid on the diagonal leaves at the end of the placement phase half of the master square empty. Considering that, to further minimise the master square size we ran the greedy algorithm multiple times:

- The first run establishes a starting point for the grid size. The solution for this iteration will consist of all squares being placed in the upper side of the diagonal of the master square.

- Using the size given by the previous iteration as a baseline for the master square size, we run the greedy algorithm on decreasingly smaller sizes and attempt to fit the squares into it. For each iteration is used a decrease by one master square size, until it reaches a size in which the greedy algorithm fails to place all squares. At this point, the iterations stop and the smallest master size in which all squares are placed is considered to be the optimal solution of the objective function.



An observation worth mentioning is that the optimal value depends on the order of the squares. We ran an experiment to illustrate that. We ran 10 times the greedy algorithm that we described above and shuffled the order of the squares before each run.

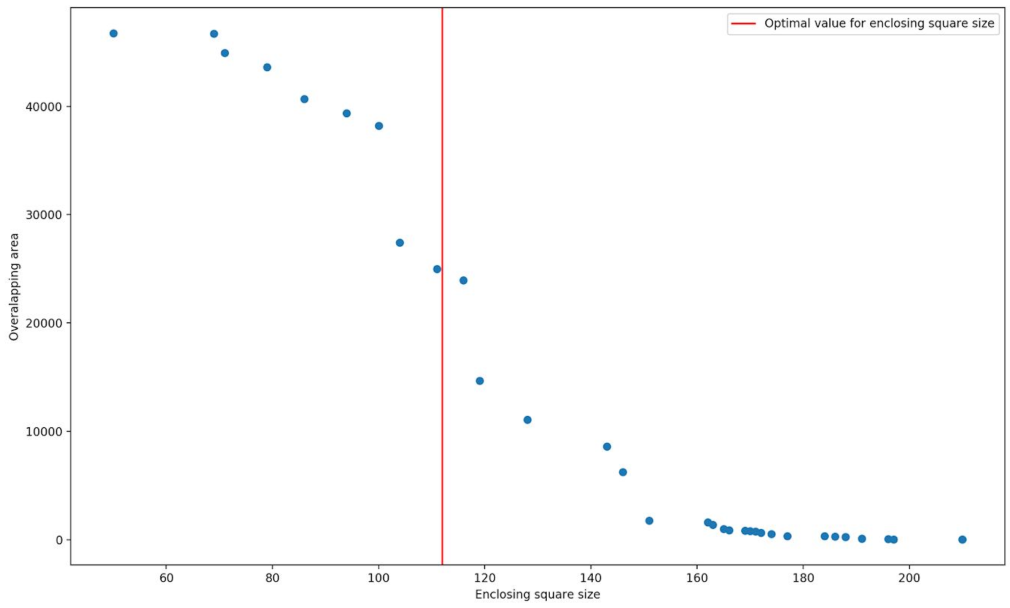


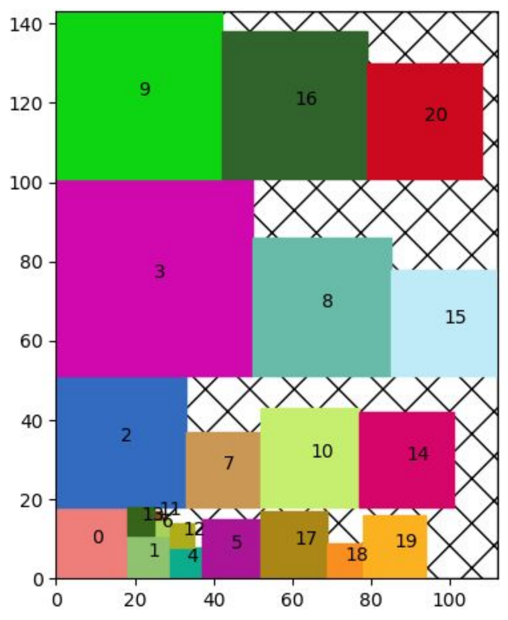
The objective function value varies a lot for the same input square sizes.

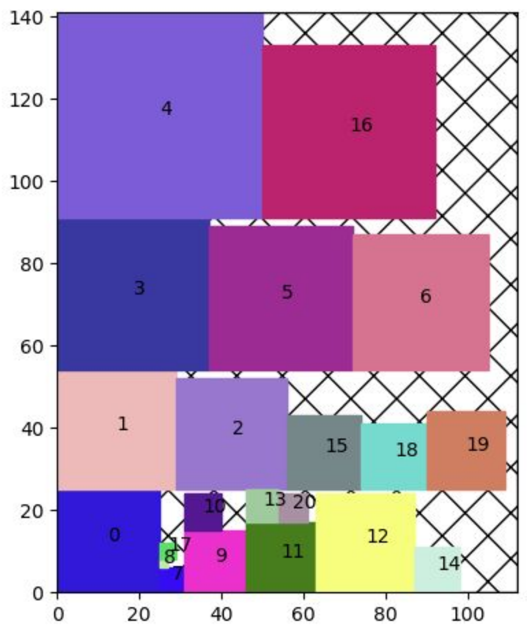
When running on the same instances, but with the input square sizes in an ascending order the algorithm obtained the master square size 195, while the experiment of descending order, so placing the bigger squares first, reached 167.
To conclude this experiment, the greedy approach finds a solution for the problem that is not optimal in most cases and is sensitive to the order of the squares, the descending order yields better results.

II. Genetic algorithm approach

The second approach using a genetic algorithm was mostly inspired by A new metaheuristic genetic-based placement algorithm for 2D strip packing. The idea is to use the genetic algorithm for just one objective, in our case minimize the size of the master square and use a deterministic, close to the optimal algorithm for square placing. We were inspired by the results we discovered for the greedy approach: the order in which a deterministic approach gets the squares influences a lot the final solution.
Basically, we reduced our problem to a strip packing problem (using as few strips of known width to pack a set of squares) which can be solved using already implemented heuristics like priority heuristic.
In this case, the individual will be a representation of a permutation of square sizes. The fitness function is the height (number of stripes * stripe width) obtained by using the permutation of squares as input to a deterministic algorithm. The mutation and crossover operators are the classic ones for permutation representations. We experimented with ordered and partially-mapped crossover. For the mutation, we tried shuffle indexes mutation.







Statistical test to showcase the improvement of the greedy optimised approach

```
Welch Two Sample t-test

data:  greedy_basic and greedy_optimised
t = 2.616, df = 16.872, p-value = 0.01815
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  8.300937 77.699063
sample estimates:
mean of x mean of y
    192.1     149.1
```

P-value < 0,05 => reject H0