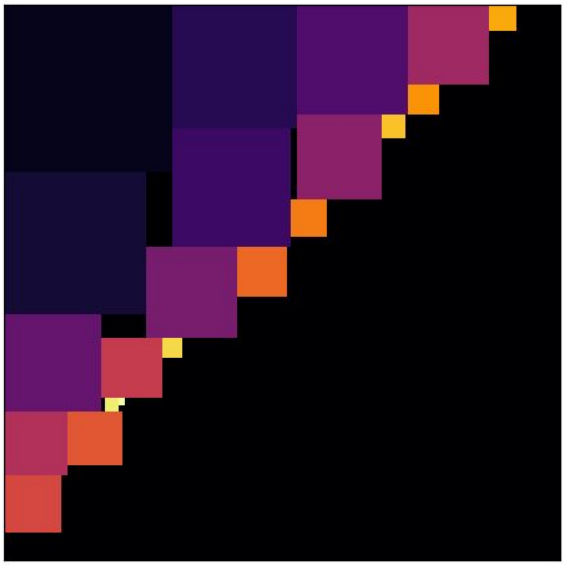
Strategies for solving the Perfect Square Placement Problem

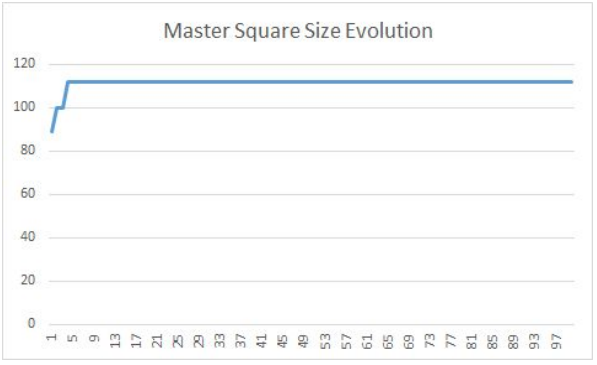
I. Greedy approach

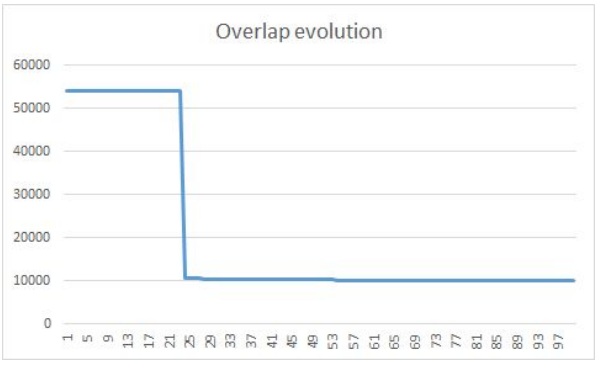
To get an upper limit for the performance obtainable for this problem we implemented a greedy approach for solving it. Our method uses a grid as the space of all possible values from solution (placements) of a square. The algorithm iterates this space for every square that needs to be placed and chooses as the solution the first placement which satisfies the constraint that squares should not overlap each other. This approach yields a solution containing the placement coordinates for all squares and the size of a bigger square, we will call it *master square*, containing all placed squares.
The objective function of our LP problem was to minimize the master square size, meaning that the placed squares should be as tight as possible. To achieve better solutions, the implementation of the greedy choice was improved by optimizing the traversal of the grid. The traversal is done along the second diagonal in order to minimize the size of the master square.



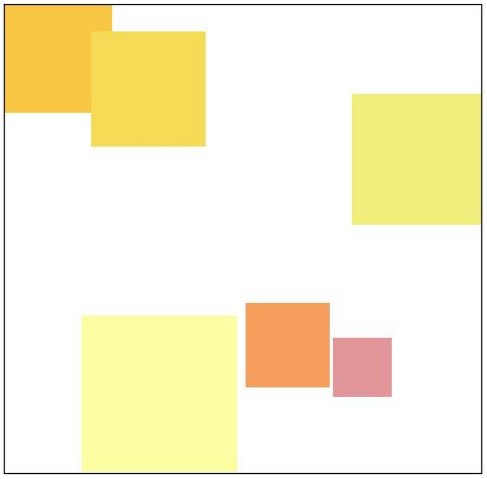
II. Genetic algorithm approach

As presented before, the search space for the perfect square placement is implicitly high-dimensional and a genetic algorithm is suitable for finding a good solution in this space due to the implicit parallelism it provides.
The design of the genetic algorithm starts with the individual representation. For the first approach regarding the genetic algorithm, the individual was an array of tuples (x, y) representing a valid configuration for the placement of the squares, meaning that we could place the squares according to an array of tuples inside a master square. The problem here was to find the right size for this master square. We used the best value obtained in the greedy approach.
Next, we chose a bi-objective fitness function. The genetic algorithm makes the selection based on two criteria: the overlap space among squares and the maximum square size the individual needs to place all the squares.





The mutation operator was implemented as changing, at random point a set of coordinates with random values. The crossover operator was to interchange two arrays of coordinates at a random cut point.
Initialization was done with arrays that place every square at the origin of the axis, having the first component of the fitness function the worst value possible and the other component, the other way around.
The selection method was the roulette and we started with a population of 100 individuals, each iteration (100 in total), “playing” the roulette another 100 times. The mutation rate was 0.1 and the crossover rate 0.6.



At the start of evolution: master square size 108, 37335 overlap

End of evolution, master square size 112, 10129 overlap