Name: Ivan Segade Carou

T Number: T00219357

# Assessment 1 – Genetic Algorithm

## Problem chosen

The problem chosen was Magic Squares. This consists in a square where the sum of each row, column and diagonal have the same result. In my approach the range of numbers are from 1 up to the number of cells in the square without repeating any value, i.e. a 3X3 square has the numbers from 1 to 9, a 4X4 square from 1 to 16 and so on.

The code generates squares with the dimensions assigned by the user in the corresponding variable in the code. The allocation of the numbers in the cells is done randomly respecting the range of possible values. The code performs the proper calculations to figure out the result of the sum of each row, column and diagonal and compares to the expected constant. Then gets the difference of each one and sum them up and provides the cost.

## Suitable of the Genetic Algorithm

In this exercise the Genetic Algorithm will try to find the Magic Square from all the squares. First it generates the population and then mutates and makes the cross over function in order to create the generations. Then it choses the square that contains the lower value. The goal is to find the square with the value 0, which means that is a Magic Square.

The Genetic Algorithm speeds out the process of figuring out what if the right Magic Square.

generated or the square that is closer to the solution. It avoids the process of making all the calculations.

## Justification

The genes of the chromosome contains the value for each cell of the square. The values are generated randomly from the value 1 up to the total number of cell in the square. The values cannot be repeated. i.e. in a 3X3 square, the values goes from 1 up to 9, in a 4X4 square, the values goes from 1 up to 16 and so on.

The cost function first calculates the expected constant, this is the value that each row, column and diagonal must give. Then it calculates the sum of each row, column and diagonal and subs tract it to the expected constant. Finally, it sums all the previous sums up and that is the cost of that square.

The mutate function gets two parameters, the mutate rate and mutate range. The first one determines the likelihood that the mutation. The second one determines the number of swaps that the mutation performs, the more number of swaps, the bigger the mutation is.

The crossover function, first generates the split value randomly. This value determines the position of the gene that divides one parent from the other. The explore\_rate variable determines the number of positions that the split variable cannot take from the first parent. The reason of this is that the split value can have a value that is the range from 1 up to the length of the chromosome. As this value is random, it can be the full length of the chromosome. As a result, the child does not take over any gene from the second parent.

The explore\_rate must have a value bigger than 1. The reason of this is if it is 1 there would be only one gene left that the child would takes over from the second parent. As the values of genes are always different, it is obvious that the left value will be exactly the same. The bigger the explore\_rate is, the more genes the child takes over from the second parent.

## Code running

A 3X3 square.

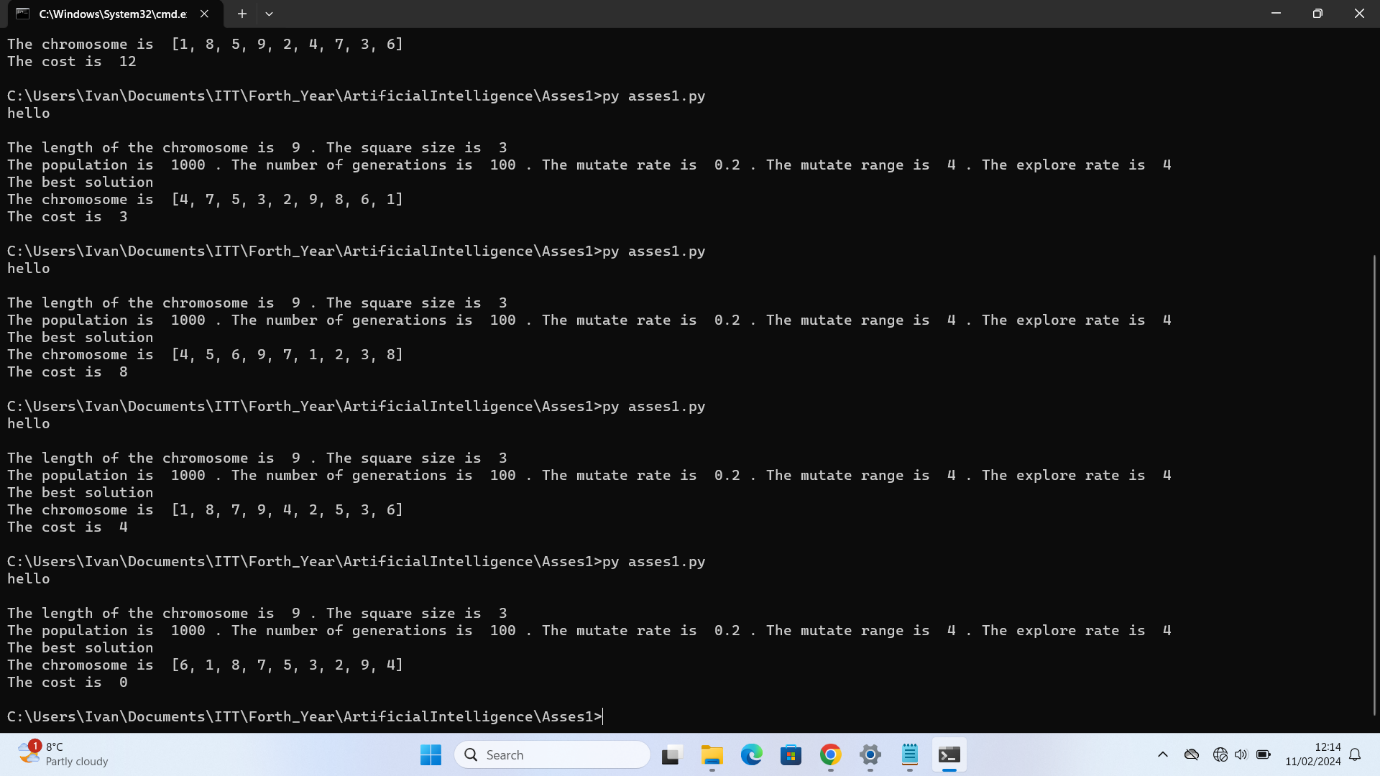


Figure 1 3X3 square

A 5X5 square.

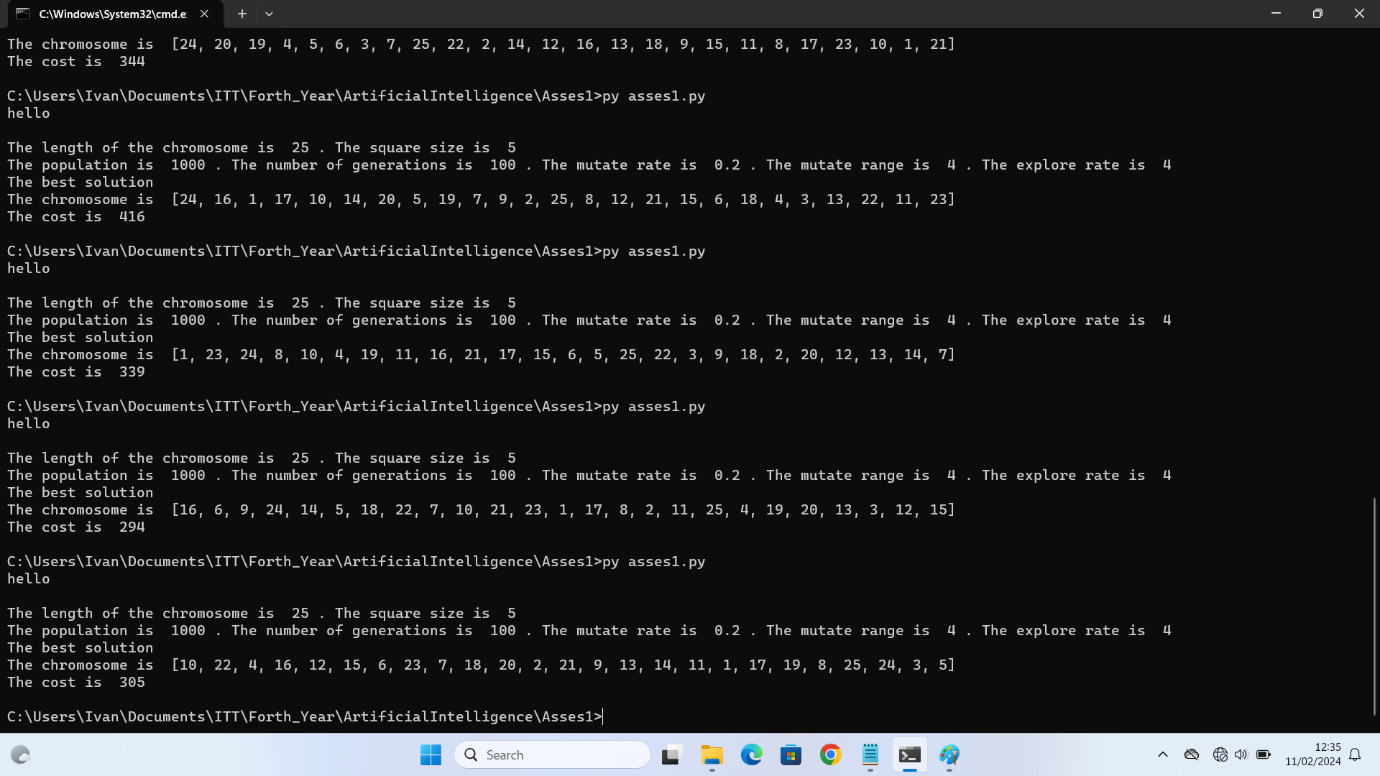


Figure 2 5X5 square

A 5X5 square with a higher number of generations and mutations.

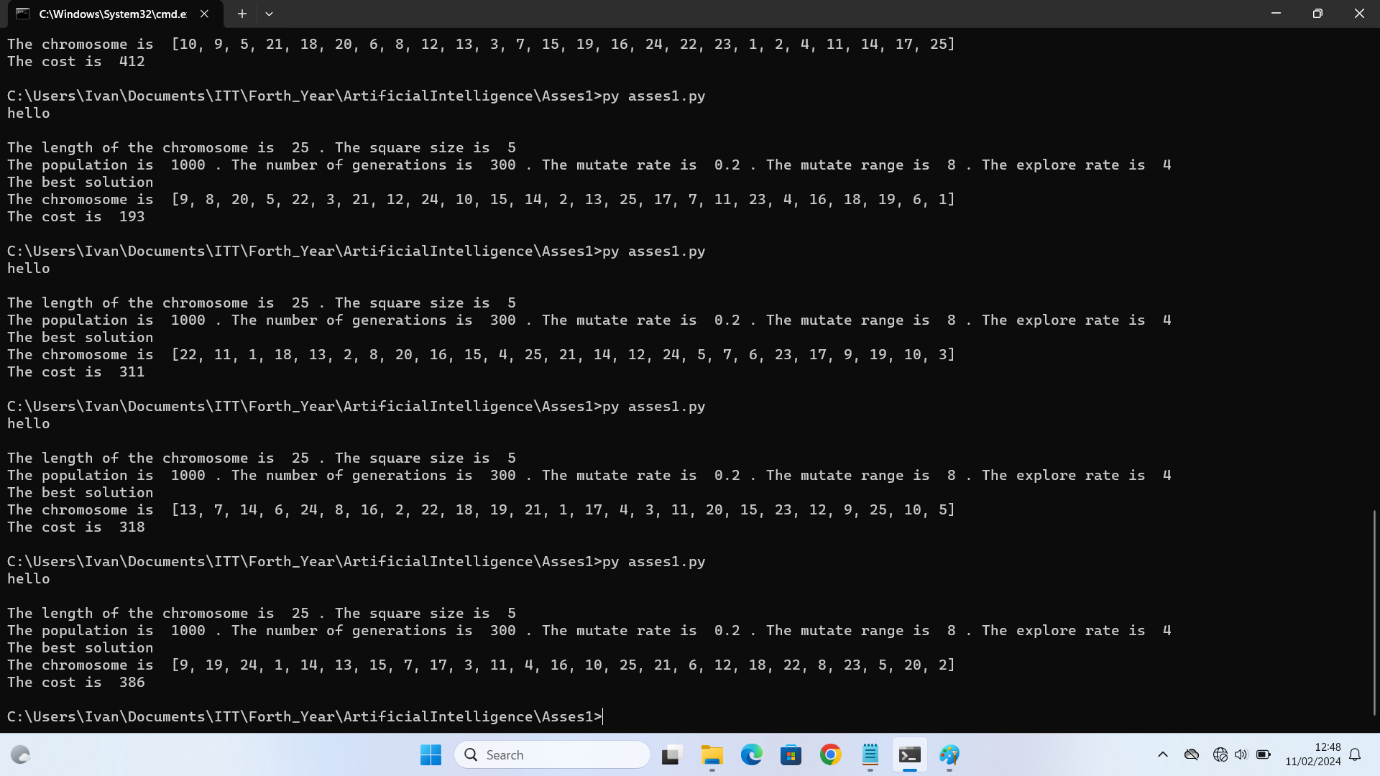


Figure 3 5X5 square with different parameters

## Conclusion

The Genetic Algorithm may be a good solution for finding a magic square. However, as the exercise is designed now, it is hard that the algorithm finds the best solution. With the 3X3 square the algorithm was able to find it, as shown in the screenshot. However, when the square is bigger, the algorithm is not effective. As shown in the screenshots, after several attempts, the algorithm was not able to create the best solution, even though the number of generations, and mutations were increased in a second attempt. The results received do not differ from the previous attempt. As a conclusion, a bigger mutation does not guarantee a better effectiveness of the algorithm.

The reason of the low effectiveness of this approach may be in the way that the numbers are allocated. They are allocated randomly. When the situation of the numbers must have a certain level of balance across the square. Therefore, when the numbers are generated, it could be some code that checks if there is that kind of level of numbers across the square.