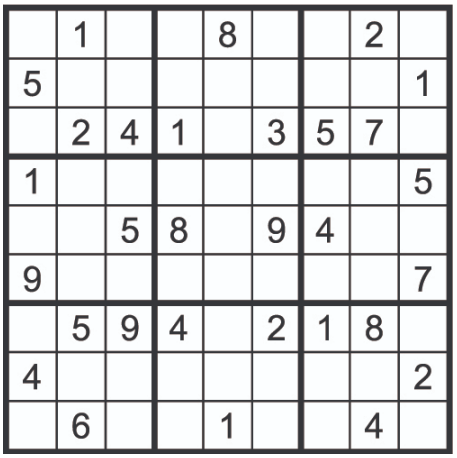
Genetic Algorithm - based Sudoku Solver



Group: CIFO Heroes

- Edgardo Juarez - m20200749
- Mohamed Elbawab - m20201102
- Salim Bouaichi - m20200547

1.0 Introduction

Sudoku is a logic-based, combinatorial number-placement puzzle. The objective of the game is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid contains all the digits from 1 to 9. As a team, we had an interest in solving sudoku puzzles. So, we thought that it's an interesting problem to be solved using genetic algorithm. The team tried a real representation that is filling a sudoku with immutable values and avoiding duplicates (fitness)

We have made a simple presentation of the problem, by providing the python code with an unsolved sudoku problem as 9x9 Matrix (represented by 9 rows with 9 values each) with immutable numbers as guideline for the code and zeros representation empty values. Then, our sudoku solver code will try to solve it. The output of the solver is the full solved matrix of the sudoku. The population is a set of possible solutions to the Sudoku puzzle is the generation needed to solve the puzzle, known as individuals which for the algorithm present a fitness and board values as the characteristics of a solution of a Sudoku board. Population can be regulated by tuning features as the number of candidates, the maximum number of generations and the patience for fitness to evolve. All Individuals are built from the Original Sudoku board which has immutable squares that governs over all of them, making it impossible to substitute, or duplicate their value on its rows, columns and blocks. This is a crucial condition for the team to build any algorithm.

The fitness scores are assigned to each of the Individual in the Population based on its closeness to the optimal (only correct) solution. This is achieved by evaluating the number of different values in each of the 9 rows, columns and blocks and then adding up their scores. This approach reflects real life, as the most duplicated number you have in any of the Sudoku evaluation sections, the furthest it is from the optimal solution.

$$\prod_{1}^{3}\sum_{1}^{9} 1/9n$$

Where n represents the number of different values of each row, column or block.

This function obeys the maximization of the problem, being 1 the optimal value of the solution. Although it could be modified to solve minimization problems the code was not designed for it.

2.0 Selection

2.1 Roulette wheel selection (Fitness proportionate selection)

It is the most frequently used selection strategy where the fitness is used to associate a probability of selection to each individual.

if fi is the fitness of individual i in the population, its probability of being selected is:

pi = fi / Σ j(fj) for j = 1 … N (N is the number of individuals in the population)

Fitness_proportionate(population)

For the total population

sum += fitness of current element

End For

For 0 to length of the set

Map the fitness of the population to a number between 0 and 1

Multiply the mapped fitness by X

For 0 to the (mapped fitness * X)

Add the current population to the mating pool

End for

End For

2.2 Tournament Selection

It is a selection strategy that selects the fittest candidates from the current generation in a genetic algorithm, and they are then passed on to the next generation for crossover.

The tournament selection method may be described in pseudo code:

- choose k (the tournament size) individuals from the population at random
- choose the best individual from the tournament with probability p
- choose the second best individual with probability p*(1-p)
- choose the third best individual with probability p*((1-p)^2)
- and so on

3.0 Mutation

Mutation Is another factor in Genetic Algorithms to maintain genetic diversity and it works pretty much as the biological mutation in the real world, as it introduces a random alteration in the individuals within the limits of reality established by the programmer. For this projects purposes 3 different mutation type were tested and compared, any of which is only executed if a randomly generated number returns a value lower than the mutation rate.

Random Resetting:

As in Bit Flip Mutation random bits flips its value from 0 to 1 and vice versa, in Random Resetting a random value is changed from one valid value in its position to another. The algorithm works by selection a random row and column and substituting the current value by any other within 0 and 9 (even the same one) confirming before that the modification won’t modify any of the immutable sudoku numbers or repeat its value on

its columns, row or block. This type of mutation can only work if duplication within rows and numbers is available.

Swap Mutation

Swap Mutation selects two random values within an Individual and exchange one for the other, this approach is particularly good for Genetic Algorithms that doesn't allow duplication. The algorithm used by the team selects a random row and two random columns within the individuals to exchange its values, only one row is selected to avoid generating duplicate values on it. It also checks that none of the modifications would change the immutable sudoku number or repeat its values on its columns, rows or blocks, in that case the algorithm will be repeated.

Inversion Mutation

Another good choice for a mutation algorithm regarding GA that doesn´t allow duplication within its genes, is Inversion Mutation. Inversion mutation, as it's name shows, converts the values from a part of the individual in order to enhance diversification. For this case it's not possible to begin thinking in mutating without considering the immutable sudoku squares and its position, as the series of values that can be selected within each row for inversion cannot contain any of these squares, so the inversion would only be done among serial spaces in the between those. The algorithm selects from safe initial points (after an immutable square or at the beginning of the row) as the first value of the series, until the next safe point which is the final of the series to be inverted. Only series with more than 1 space available are selected for inversion, once the series and values have been determined a final check is done, to validate that none of the modifications will repeat the immutable squares value on its rows, columns or blocks.

4.0 Crossover

As for the cross over. We have used multiple types

Cycle cross over with single point

In the code we have used a separate function for the cross over. First, we have decided the single points of the crossover by picking a random number between 0 and 1.0. This number would help in choosing the cross over rate. This function would help in choosing the randomness of the function generated.

Then we would choose the cross over point. Where we choose a number between 1 and 7. This number will be the location of the crossover of the 2 parents.

The cross over value and the parents would then be transferred to the cycle cross over. The sudoku is the perfect application for the cycle cross over where it chooses the value and then check the index on the other parent. Hence all values will be covered within these 2 parents. For the single point cross over the change for the parents would happen from the cross over point to the end of the array. Other than that, the child values are copied as per the parent values.

For the cycle crossover, we start by filling the child rows with zeros. Then we cycle until these zeros are removed. So we start be checking the 1st unused value and then we get this value from parent 1 to child 1 and parent 2 to child 2. These used values are then removed from the remaining values array that we have and that we pick the values from. We then cycle to fill all of the needed values and return the child rows.

Cycle cross over with double point

Similar to the single point crossover, the double point cross over will use a separate function for the cross over. First, we will start by picking a random number to assure the randomness of the function like the single point. The random number was increased to be 0 to 1.1 while testing to assure more cycles are done. So we adjusted the range to be from 0 to 1.1. and a while the chosen number is bigger than 1, the function will rerun until a new number is picked under 1.

Then 2 crossover point are chosen. Where we choose 2 numbers between 1 and 8. These 2 numbers will be the range of the crossover of the 2 parents. If the 1st number is higher than the second number, the numbers will be inverted so the range order would be adjusted.

The cross over range and the parents would then be transferred to the cycle cross over. For the double point cross over the change for the parents would happen from the cross over first range to the second range. Other than that, the child values are copied as per the parent values. As for the cycle crossover, it is performed as per the cycle crossover with a single point.

5.0 Results, Discussion and Conclusions

Several trials were made using the different combinations shown within the previous sections (Selection, crossover, mutation). Combined average values are shown in Table 1. Some of the highlighted values will be shown below.

For the Roulette, Double cycle crossover and swap mutation:

Easy Sudoku was solved with an average of 4.4 generations (Fig 3). A normal sudoku was solved with 204 generations. Few trials could not get an answer with the normal one.

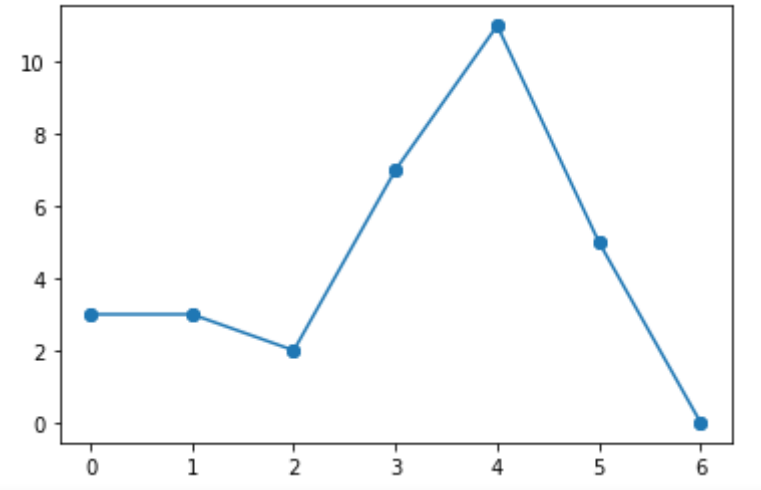


Fig 3. Generation number for each run, Easy Sudoku, Roulette, Double cycle crossover, swap mutation

Number of Runs	Selection	Crossover	Mutation	Average Generation
20	Tournament	Double-Cycle Crossover	Swap	**Easy | 3.33**
20	Roulette	Single-Cycle Crossover	Swap	**Easy | 21.33**
20	Roulette	Double-Cycle Crossover	Inversion	**Easy|337.33**
20	Tournament	Single-Cycle Crossover	Inversion	**Easy 400**
20	Tournament	Single-Cycle Crossover	Swap	**Easy | 5**
20	Tournament	Double-Cycle Crossover	Inversion	**Easy |180**
20	Roulette	Single-Cycle Crossover	Inversion	**Easy |122**
20	Roulette	Double-Cycle Crossover	Swap	**Easy | 4.4**

Table 1: Selection, crossover, mutation outputs for Easy Sudoku puzzle.

From the previous runs, we have concluded the following.

- For the selection methods: Roulette is more effective in solving the sudoku puzzles.
- For the crossover: Double crossover is more effective in solving the sudoku puzzles. It is even faster within the process.
- For the Mutation: Swap is more effective in solving the sudoku puzzles.

Room for improvement: We would suggest using more powerful tools and more sudoku examples with different levels (Medium and Hard). Also, there were GA algorithms that we didn't try due to the complexity of the problem.