Report of Project Sudoku Solver – Genetic Algorithm
Computational intelligence for Optimization
May 28th of 2021

John William Villalobos Ruiz - m20200540
https://github.com/johnruiz24/CIFO

Objective

This study aims to evaluate the influences and the performance of different approaches, algorithms, and genetic operators to solve the resolution of any sudoku puzzle with Genetic Algorithm and documenting the results obtained. The two objectives of this study were to explore and test the usefulness and efficiency of genetic algorithms to address this constraint satisfaction problem through the utilization of a wide set of techniques and the second objective is to explore how problems that are considered difficult for a human solver are also difficult for the genetic algorithm validated through the testing of multiples level of difficulties.

Overall Description of the Evaluation of Genetic Algorithms

- Having set the boundaries of the population and determined the group of candidate solutions, the fitness score is measured and the best fitness for generation 0 is selected and printed out
- Selection Method
    - The best solutions from the previous phase are selected and pushed to the next generation, going through the selection process in which a method between Tournament, Fitness Proportionate Selection or Rank is chosen.
    - The decision relies on a random number between 1 and 3 having the same probability of occurrence then the flow is redirected to one of these methods
    - Regarding the tournament an adjustment was made to randomly pick 2 candidate solutions and obtain a random uniform number between 0 and 1 this is compared with a variable called selection rate in which if the random number is greater than the selection rate the fittest candidate solution is picked otherwise pick the weakest, this is because due to the absence of maximization or minimization implementation, the decision to whether or not must always be the highest fitness relies on a random probability, the selection method is invoked twice per iteration (1000 iterations per generation) and pushed towards the crossover phase
- Crossover
    - Previously a decision on the crossover technique is determined by a random distribution number between 1 and 3 with different probabilities of occurrences comprised by 0.2 arithmetic crossover, 0.35 cycle crossover and 0.45 PMX crossover
    - The decision on the range relates to how well they contribute to find the optimal solution
    - The implementation of such methods as well as the tournaments uses the library from class with smooth modifications in some cases

    - A new method for the cycle crossover could be found on the cycle as it finds to be more suitable for the sudoku however the method from class was chosen to be delivered
- Mutation
    - Keeping the same pattern as done with the previous phases a random uniform distribution comes up with a number to be compared with the mutation rate and decide on the occurrence of the event.
    - The offspring retrieved from the crossover phase is being pushed towards the mutation stage in which a swapping mutation point takes place
    - This mutation rate helps to prevent an over excess in the number of mutations per generation this is based on the "Evolution Strategy 1/5 Success Rule" Reichenberg rule that measure the strength of the mutation in which if the number of mutations keeps climbing higher the phi operator will be updated by dividing the phi/number of mutations
        - if phi operator exceeds 0.2 it will impact negatively on "sigma" dividing by (e.g.: 0.98), such operator can be understood as the standard deviation of a gaussian distribution with no sample, therefore having a small standard deviation will lead to a lower mutation rate increasing the likelihood of no mutation occurrence for a certain period of time.
        - Conversely if phi operator decreases it will sigma positively on the standard deviation increasing its value multiplying by a factor (e.g.: 0.98) increasing the likelihood of mutation occurrence
        - This is an interesting way of being in control of the healthy mutation rate
        - Of course, I got inspired on this by the job done by somebody else, so I decided to test and see how this impact on the resolution of the puzzle
        - What I found was that having a small standard deviation makes the algorithm to struggle to find a solution even with puzzles of simple levels of difficulty
- Re-seeding
    - Occurs every time the algorithm reaches an stalling phase in which the last 2 top fittest solutions are the same
    - The algorithm wait for an interval defined on the setup to re-seed the population and with the purpose of finding an optimal solution
    - This has proven to be a game changer solving puzzles of different complexities by having this approach, an image is being found on the effect of the resolution of puzzles of different level of difficulties following this line of reasoning.

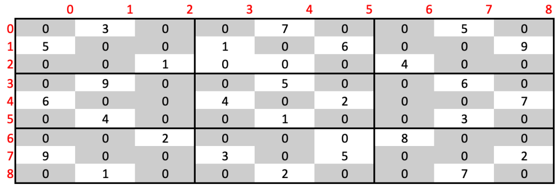
Approaches and Genetic Operators Implemented

Technique Operator	**Variations Implemented**
Crossover Operator	Arithmetic Crossover PMX Crossover Cycle Crossover
Mutation Operator	Swapping Mutation

Parent Selection Approach	Tournament Selection Roulette Wheel Ranking Selection *Adjustment on Tournament Selection
Survivor Selection Approach	Elitism
Fitness Function	

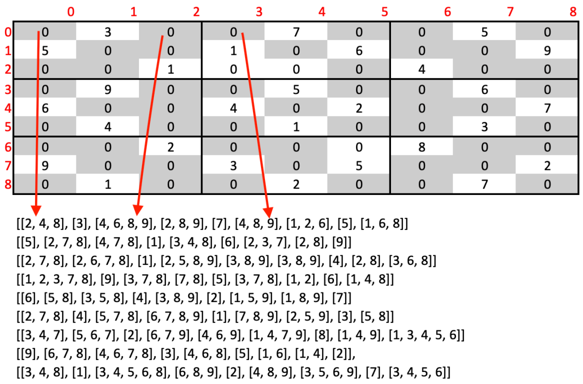
Questions

1. Why did you choose the representation you did?
   The representation was conceived by thinking about all possible set of numbers for those given in blank that could lead towards the finding of an optimal solution whether by trying permutation numbers from 1-9 even if those are invalids values or putting a key attention on valid numbers, as this matrix optimization problem has a large search space many thousands of combinations therefore this work relies on valid numbers to build up the initial set of individuals assembling the candidate solutions.
   The next steps were taken to produce the representation
   - Analyzing the set of permutations for any given blank space across the matrix, black spaces retrieved as part of the web scrapping were replaced by 0, the local finding might break no rule from the game comprised into 3 main criteria
     - A row, column or block containing all numbers between 0 and N starting on 9 and jumping in increments of 3
     - No duplicated elements can be anyhow found across any of the 3 objects listed above
     - For instance, let's assume we might solve the following matrix having 0 as the not given number



	0	1	2	3	4	5	6	7	8
0	0	3	0	0	7	0	0	5	0
1	5	0	0	1	0	6	0	0	9
2	0	0	1	0	0	0	4	0	0
3	0	9	0	0	5	0	0	6	0
4	6	0	0	4	0	2	0	0	7
5	0	4	0	0	1	0	0	3	0
6	0	0	2	0	0	0	8	0	0
7	9	0	0	3	0	5	0	0	2
8	0	1	0	0	2	0	0	7	0

     - The algorithm rises 3 question to solve applying a permutation of numbers from 1 to 9 (9 X 9) matrix
       - Does the number 1 exist across the row or column 0?
         - Column: No
         - Row: No
       - Does the number 1 exist inside the block [0,1,2:0,1,2]
         - Block: Yes
       - Concluding that as the number 1 already exists in its given block, breaks the rule of the game therefore is not being appended into the set of legal numbers
     - The algorithm keeps iterating with numbers from 2 to 9 and come up with the following set of combination [2,4,8]
     - Having completed the scanning process, we come up with the following set of numbers



	0	1	2	3	4	5	6	7	8
0	0	3	0	0	7	0	0	5	0
1	5	0	0	1	0	6	0	0	9
2	0	0	1	0	0	0	4	0	0
3	0	9	0	0	5	0	0	6	0
4	6	0	0	4	0	2	0	0	7
5	0	4	0	0	1	0	0	3	0
6	0	0	2	0	0	0	8	0	0
7	9	0	0	3	0	5	0	0	2
8	0	1	0	0	2	0	0	7	0

```
[[2, 4, 8], [3], [4, 6, 8, 9], [2, 8, 9], [7], [4, 8, 9], [1, 2, 6], [5], [1, 6, 8]]
[[5], [2, 7, 8], [4, 7, 8], [1], [3, 4, 8], [6], [2, 3, 7], [2, 8], [9]]
[[2, 7, 8], [2, 6, 7, 8], [1], [2, 5, 8, 9], [3, 8, 9], [3, 8, 9], [4], [2, 8], [3, 6, 8]]
[[1, 2, 3, 7, 8], [9], [3, 7, 8], [7, 8], [5], [3, 7, 8], [1, 2], [6], [1, 4, 8]]
[[6], [5, 8], [3, 5, 8], [4], [3, 8, 9], [2], [1, 5, 9], [1, 8, 9], [7]]
[[2, 7, 8], [4], [5, 7, 8], [6, 7, 8, 9], [1], [7, 8, 9], [2, 5, 9], [3], [5, 8]]
[[3, 4, 7], [5, 6, 7], [2], [6, 7, 9], [4, 6, 9], [1, 4, 7, 9], [8], [1, 4, 9], [1, 3, 4, 5, 6]]
[[9], [6, 7, 8], [4, 6, 7, 8], [3], [4, 6, 8], [5], [1, 6], [1, 4], [2]],
[[3, 4, 8], [1], [3, 4, 5, 6, 8], [6, 8, 9], [2], [4, 8, 9], [3, 5, 6, 9], [7], [3, 4, 5, 6]]
```

- The list of eligible values has to be translated into a set of candidate solutions, the algorithm starts by randomly picking a value for the given cells until it reaches a matrix of 81 numbers called this as individual, the iteration continues up to the maximum number of candidate solutions comprised by the analysis.

2. How did you design the fitness function?

   The fitness was built comprising into the 3 main criteria any sudoku puzzle has which are:

   - A row, column or block containing all numbers between 0 and N starting on 9 and jumping in increments of 3
   - No duplicated elements can be anyhow found across any of the 3 objects listed above

   On top of this we can then define the fitness function as:

   - Count the unique numbers in any given row, column or region divided by the total of non-unique or duplicated numbers
     - The quotient number plays the role of a penalizer for those for those set of elements that do not pose a unicity negatively impacting the matrix solution for those set of elements that do not pose a unicity, negatively impacting the matrix solution
     - In order to manages 2 different types of penalization the formula increases by 1 to the division quotient except for those that have no duplication
   - Which are then sum up and divided by the size of the puzzle
   - Finally, the current fitness is obtained by the multiplication of the scores obtained throughout the rows, columns, and blocks
     - The maximum value across each of these objects is 1 pointing that no duplication was found by the algorithm

$$\underset{\textbf{Max Value = 1}}{\overset{\textbf{Row Fitness}}{\frac{\sum \frac{\text{COUNT (Unique Numbers)}}{\text{Number of Duplicated + 1}}}{\text{Size of the Matrix}}}} \;\mathbf{x}\; \underset{\textbf{Max Value = 1}}{\overset{\textbf{Column Fitness}}{\frac{\sum \frac{\text{COUNT (Unique Numbers)}}{\text{Number of Duplicated + 1}}}{\text{Size of the Matrix}}}} \;\mathbf{x}\; \underset{\textbf{Max Value = 1}}{\overset{\textbf{Block Fitness}}{\frac{\sum \frac{\text{COUNT (Unique Numbers)}}{\text{Number of Duplicated + 1}}}{\text{Size of the Matrix}}}}$$

- Alternatively, a light penalizer has been developed to remove the increment from the quotient meaning that the count of unique numbers will be divided by the number of duplicated numbers
- Likewise, the code handles an exception for those with no duplication keeping the number without any division

        - This small increase +1 (was established because of those with no duplication could potentially lead to a division by zero)

3. Did you try using different fitness functions to see the impact on your GA?
   One of advantages of this matrix constraint optimization problem is the flexibility to promptly rehearse the fitness function, the fitness function described above could easily be fitted into the mean average of the scores throughout the row, column, and block this was the first approach taken to tackle down the sudoku problem however this slight variation decreases the penalty on duplicated solutions making them appear with a higher value next to those that are really close to the optimal solution.
   For instance, let's assume that the intermediate scores are: row, column, and block = 0.9, computing the fitness with the mean average leads to 0.9 conversely with the multiplication of such scores the number falls down to 0.73 this approach solves the easy and very easy solutions but struggled to solve the moderate and difficult levels because it reached a plateau around a higher score disregarding those candidates with a lower score that could potentially lead to find the optimal solution
   This was part of the implementation and produced higher scores since the first generation around 0.7 and 0.8 but proven to be not efficient for more complex matrix on the contrary the implemented fitness starts with small scores around 0.2 and the value begins to rise until it reaches a certain stability which in some cases leads to find the optimal solution.

   Additional variations were tested such as
   - Count of the unique numbers divided by the size of the matrix
   - Perform the main average of the sum of the 3 scores or the multiplication of the scores

$$\underset{\textbf{Max Value = 1}}{\overset{\textbf{Row}}{\sum \frac{\text{COUNT (Unique Numbers)}}{\text{Size of Matrix}}}} + \underset{\textbf{Max Value = 1}}{\overset{\textbf{Column}}{\sum \frac{\text{COUNT (Unique Numbers)}}{\text{Size of Matrix}}}} + \underset{\textbf{Max Value = 1}}{\overset{\textbf{Block}}{\sum \frac{\text{COUNT (Unique Numbers)}}{\text{Size of Matrix}}}}$$

4. Which configurations work best together? How many did you try and how did you determine the best one?

   One of the findings of this work was to test the effectiveness of different genetic algorithms strategies in parallel and found that in terms of crossover keeping the same configuration for the other operators (mutation and tournament) we can rank these techniques by how they impact on the resolution time and different levels of complexity
   - Arithmetic Crossover
     - This method itself couldn't solve puzzles of the 2nd level of complexity (called Easy)
   - Cycle Cross Over
     - This method itself couldn't solve puzzles of difficult complexities
   - PMX
     - This method itself could solve puzzles of different complexities

However, keeping them together was able to solve any kind of complexity.

5. Have you implemented elitism? Does the inclusion or exclusion of elitism impact your GA?
   Elitism was critical to solve the matrix, without elitism the algorithm struggled to find a solution predominantly with no success.
   The idea behind this was straightforward, before the generation process starts sort the set of candidate solutions by its fitness and preserve the top 7% for the coming generation, meaning that the remaining candidates to be found comprises the 93% of the overall population.

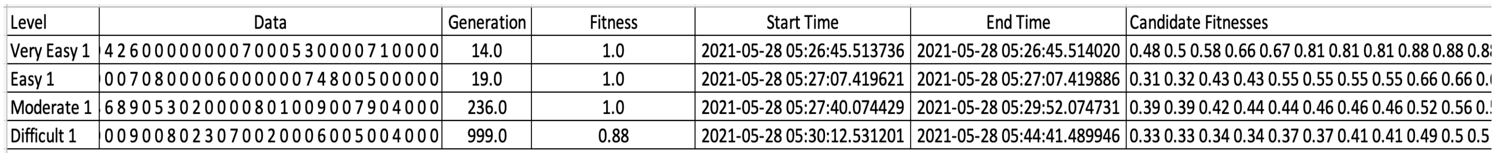
6. Do you get good results for the project you choose? What could be improved?
   I think

   For difficult puzzles in which there are fewer given numbers, the number of generations required to reach the optimum solution becomes enormous and the probability of obtaining a solution in realistic time becomes small hence increasing the number of generations for puzzles with higher complexities could be a way to tackle down in a promptly way.
   It would be also interesting applying different kind of techniques to model in a different way the population and its fitness, techniques such the ones listed below could contribute on this matter
   - The Full House
   - The Naked Singles
   - The Hidden Singles
   - The Lone Rangers
   - The Naked pairs

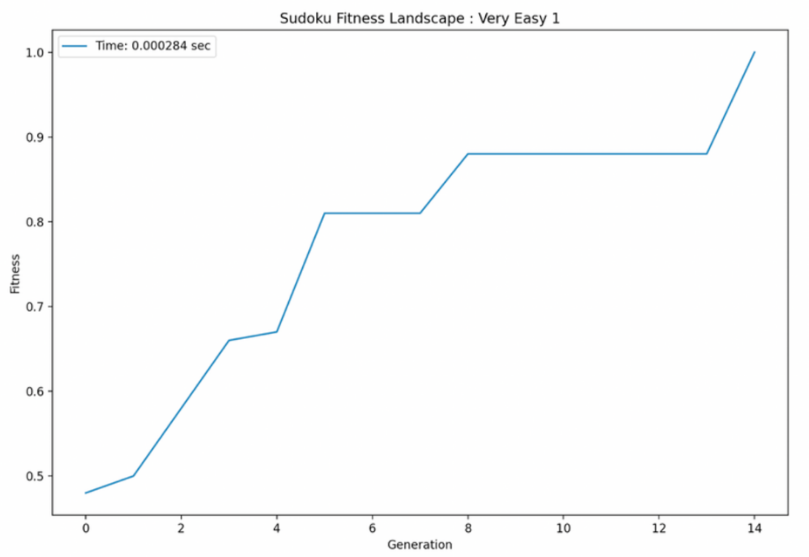
   Furthermore, having a control of the number of operations performed for every candidate solution at every iteration and generation will definitely give a broader insight in a more accurate way of tunning the algorithm to in certain types of scenarios and complexities better to choose a specifical set of combinations of genetic algorithms such as for instance:
   - Difficult: PMX and Tournament and Swapping Mutation
   - Moderate: Cross Cycle and FPS and Inverse Mutation

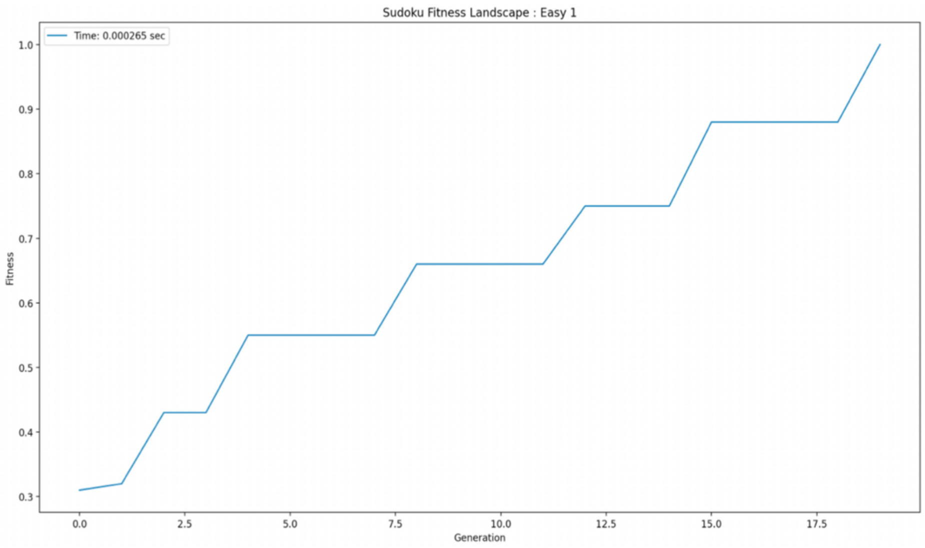
   As well as a more appropriate number of ideal candidates and generations per each level of complexity

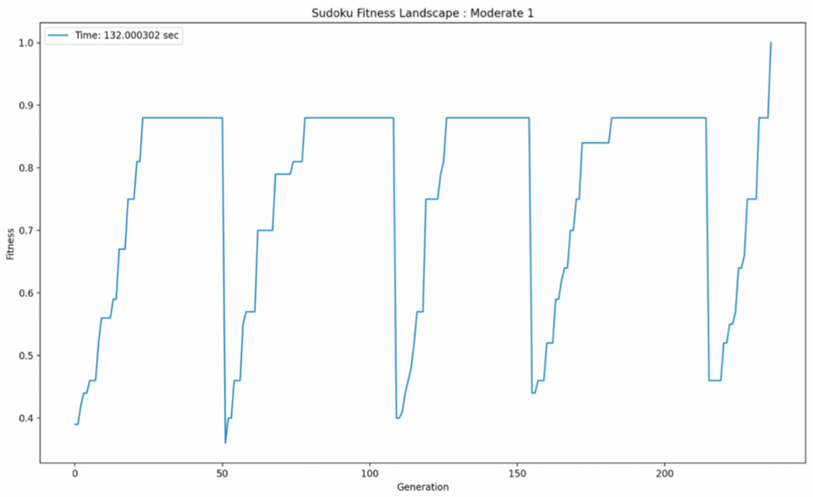
   Statistically summary retrieved by the Puzzle at the end of the processing

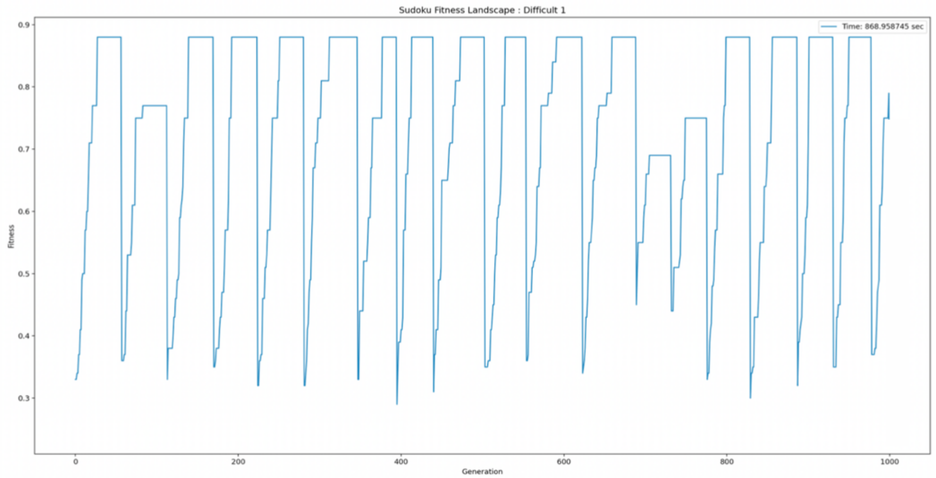


Level	Data	Generation	Fitness	Start Time	End Time	Candidate Fitnesses
Very Easy 1	4 2 6 0 0 0 0 0 0 0 0 7 0 0 0 5 3 0 0 0 0 7 1 0 0 0 0	14.0	1.0	2021-05-28 05:26:45.513736	2021-05-28 05:26:45.514020	0.48 0.5 0.58 0.66 0.67 0.81 0.81 0.81 0.88 0.88 0.8
Easy 1	0 0 7 0 8 0 0 0 0 6 0 0 0 0 0 0 7 4 8 0 0 5 0 0 0 0 0	19.0	1.0	2021-05-28 05:27:07.419621	2021-05-28 05:27:07.419886	0.31 0.32 0.43 0.43 0.55 0.55 0.55 0.55 0.66 0.66 0.
Moderate 1	6 8 9 0 5 3 0 2 0 0 0 0 8 0 1 0 0 9 0 0 7 9 0 4 0 0 0	236.0	1.0	2021-05-28 05:27:40.074429	2021-05-28 05:29:52.074731	0.39 0.39 0.42 0.44 0.44 0.46 0.46 0.46 0.52 0.56 0.
Difficult 1	0 0 9 0 0 8 0 2 3 0 7 0 0 2 0 0 0 6 0 0 5 0 0 4 0 0 0	999.0	0.88	2021-05-28 05:30:12.531201	2021-05-28 05:44:41.489946	0.33 0.33 0.34 0.34 0.37 0.37 0.41 0.41 0.49 0.5 0.5


Sudoku Fitness Landscape : Very Easy 1
Time: 0.000284 sec
Fitness
1.0
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8
10
12
14
Generation


Sudoku Fitness Landscape : Easy 1
Time: 0.000265 sec
Fitness
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Generation


Sudoku Fitness Landscape : Moderate 1
Time: 132.000302 sec
Fitness
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
Generation


Sudoku Fitness Landscape : Difficult 1
Time: 868.958745 sec
Fitness
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
200
400
600
800
1000
Generation