Solving Sudoku puzzles with a Genetic Algorithm

Nova Information Management School – 2020/2021

Computational Intelligence for Optimization: Project Report

Members of the ‘Sudoku GAmers’ group: Ana Rita Domingos (R20170777), Helena Chhotobhai (R20170779), Rodrigo Mourão Nunes (R20170872), Rui Monteiro (R20170796)

1. Introduction

Sudoku is a puzzle game created in the United States during the ‘70s, originally called Number Place, the name the Americans still call nowadays, and was later introduced to Japan in 1984 by the company Nikoli, where it remained unpopular until 2004 when the magazine The Times published their first edition of a Sudoku (The Guardian, 2005).

The game consists of fulfilling the blank spots with digits from 1 to 9 so that there are no duplicate numbers in any row, column or 3x3 box. There are 6 670 903 752 021 072 936 960 possible Sudokus to be solved. Figure 1 shows two puzzles: the initial grid (on the left) and its final solution (right).

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

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

Figure 1. Example of Sudoku puzzles.

In this project, we attempted to create a Genetic Algorithm (GA) with an initial population, fitness function, selection methods and genetic operators, that could evolve over several generations in order to solve a Sudoku puzzle.

With the help of https://sudoku.com/, it was possible to extract unsolved puzzles to use on the algorithm, with various difficulties: “easy”, “medium”, “hard” and “expert”.

The project can be accessed here, on our GitHub repository, which contains the Python files with all the relevant code.

In this report, the Methodology will be presented in Section 2, the Results in Section 3, the Discussion in section 4 and the Conclusions in Section 5.

2. Methodology

This section will present our GA implementation, starting by the encoding.

- **Encoding:** In order to implement this algorithm, first it was necessary to provide an unsolved puzzle, in the form of a list of 81 numbers, where the blanks would be converted into zeros and the first row would start in index 0 and go until index 8, while the other rows would always start in indexes multiple of 9.
- **Initial Population:** The initial population was created with the help of the *grid_initial* list, which stores the indexes where it was possible for the GA to make modifications. Each one of the *n*

individuals would be created based on the indexes that were left to fill, ensuring there would not be any duplicates on the puzzles' rows (thus, all rows start as permutations of 1 to 9).

- **Fitness Function:** To evaluate the individuals at each generation, two fitness functions were created. The first one was simpler and was only based on the sum of duplicates in rows, columns and boxes. The second function was more robust and had into account not only the number of duplicates in each row, column and box, but also the absolute difference between 45 and the sum of the elements on each row/column/box, since that must be 45 in order for the Sudoku to be solved. Also, it takes into account the absolute difference between the factorial of 9 and the product of each row/column/box must be 0. This function is a weighted sum, where weights are defined according to the importance of each section of the fitness function in the solving process of a Sudoku puzzle (we gave much more weight on duplicates than in the other parts of the sum). Both fitness functions turn this into a minimization problem. Also, the global optimum on both should be equal to 0.
- **Selection:** Three selection methods were applied: Fitness Proportionate selection (FPS), Tournament selection (with size equal to 10 by default) and Rank selection, all implemented in the context of a minimization problem.
- **Crossover:** Three types of crossover operators were implemented. The first one was the Uniform crossover where each gene (number) is selected randomly from one of the corresponding genes of the parent chromosomes and are swapped (or not) according to a certain probability (Syswerda, 1989). Then, the Single-Point crossover (one crossover point is chosen) and the Two-Point crossover (two crossover points are chosen) were implemented. For that, the crossover points can only appear between rows, so a 9x9 grid will have 8 possible crossover points.
- **Mutation:** This genetic operator was a crucial part of the Sudoku solver, so we implemented six different types of mutation: Swap, Multiple Swap, Random, Intelligent Random, Inversion and Partial Inversion. It is important to point out fixed and predetermined numbers never got swapped out of position or changed in any way. Thus, the mutations never generated invalid or wrong puzzles. None of these operators mutate rows and columns at the same time (they will either mutate rows or columns on the selected individual).

Swap mutation was implemented by choosing a random row or column of the individual, and then swapping the two random numbers in the mutation points.

Multiple Swap mutation is similar to the previous one, but with one difference: it has a very high chance of choosing more than one row or column (the probability was set at 0.9 by default). It may even choose all rows or all columns and apply a Swap mutation to each one of them.

Random mutation was an operator that we invented for the Sudoku solver. It randomly chooses which columns or rows to mutate (according to a probability that was set at 0.5 by default). Then, for each column or row previously selected, it chooses two random genes (numbers) to mutate. For each one of these numbers, the operator mutates it to a random value between 1 and 9.

Intelligent Random mutation was also invented by us, and it is very similar to the previous one. The only difference is that this implementation never inserts random numbers that were already among the predetermined numbers of the selected column/row (e.g., if row number 4 is selected and has numbers {1, 3, 4, 9} in the initial grid, then none of those numbers will be inserted during mutation).

Inversion mutation was implemented in the following way: it has a chance of choosing more than one row or column (the probability was set at 0.5) to mutate, then it inverts the selected rows or columns (excluding the numbers that are predetermined in the initial grid).

Finally, Partial Inversion mutation was also created by us and functions similarly to the previous one, but with a difference: instead of inverting the whole column or row, it only inverts a part of it, which is randomly selected.

- **Statistical Analysis:** Two functions were created in order to perform statistical tests. The first one is characterized by receiving two arguments: the CSV name that has the data, and the plot_name. It imports the data, makes the necessary changes and creates a plot with three lines: Best fitness, Worst fitness and Average fitness for each Generation. The second one has the objective of comparing several GAs, receiving three arguments: a list with the names of the CSV files that have the data, a list with the legend and the plot_name. Then it imports the CSV files one by one, makes the necessary changes and concatenates all dataframes. With this new dataframe, it plots the Average best fitness for each Generation with a 95% confidence interval. The number of lines in the plot will be equal to the number of files provided to the function.
- **Pseudo-Code:** The pseudo-code for the GA applied to this Sudoku problem is:

```
1. Generate an initial population P (calculating the fitness for each individual)
2. Repeat until (number of generations)
    2.1. Create an empty population P'
    2.2. Select the best individual (if elitism is True)
    2.3. Repeat until (number of generations)
        2.3.1. Select two individuals from P, using a selection method
        2.3.2 Apply the crossover operator according to the crossover rate to individuals selected at 2.3.1
        2.3.3. Apply the mutation operator according to the mutation rate to individuals selected at 2.3.2
        2.3.4. Insert the individuals obtained at 2.3.3 in population P'
        2.3.5. Substitute the best individual selected from 2.2. with the best individual in P' (calculating its fitness)
        2.3.6. Print the Generation, the fitness of the best and worse individuals
    2.4. Replace population P with P'
3. Return the best individual in P
```

3. Results

To better understand which would be the best GA for our problem, we tested several selection methods, genetic operators, probabilities, among others. Also, to ensure our choices were statistically valid, we generated CSV files with data about many GA configurations, in order to compare them. The files can be accessed here. All the algorithms were run 30 times, in order to ensure we had a large enough sample. Data about fitness values and individuals' representations were saved into all files.

Moreover, the "easy" and "expert" puzzles were used to test the GAs with the following common parameters: the populations' size equaled 1 000 with 75 generations, crossover probability of 0.7, mutation probability of 0.9 and *True* elitism (this last parameter had little impact in the GA's solutions).

We used very high probabilities for crossover and mutation because these genetic operators revealed themselves as essential for the proper evolution of our GA. Also, low tournament sizes in Tournament selection led to better fitness values during the evolution process.

- **Comparison of GAs**

After analyzing the plots in Appendix A, that take into consideration the average best fitness comparing FPS, Ranking and Tournament selection, for the Uniform crossover and Swap mutation, for the "easy" and "expert" puzzles, it can be concluded that the best selection is the Tournament. It can be seen that in the "easy" puzzle, FPS is slightly worse, but in the "expert" puzzle this difference is not very noticeable. Moreover, Tournament selection starts to decrease more, compared to FPS and Rank selection, after generation 15.

Moreover, it can be seen that in contrary to what has been said before, in the last two combinations, Single-Point crossover and Swap mutation and Two-Point crossover and Swap mutation, Tournament selection starts to get lower values for the average best fitness in the beginning. Also, it is clear that FPS selection has the worst average best fitness, Rank selection has the second lowest and Tournament selection is the one that has the best, having fitness values close to 10 after 75 generations.

Therefore, it can be concluded that the best parameters are: Tournament selection and Two-Point crossover. The explanation behind this conclusion is that FPS and Rank selection are based on probabilities, and many times the worst fit individuals never got selected, therefore decreasing the diversity in populations.

To conclude which would be the best mutation operator, we developed two more tests (one for the "easy" puzzle and another for the "expert") with Tournament selection and Two-Point crossover.

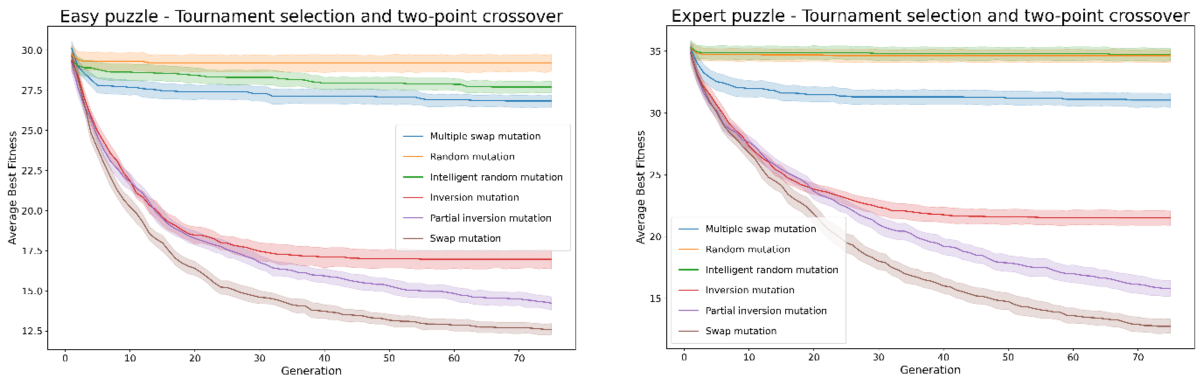


Figure 2. Comparison of GAs for the mutation operators.

The plots in Figure 2 highlight similar conclusions: Swap, Partial Inversion and Inversion mutation perform much better than Multiple Swap, Random and Intelligent Random mutation. Furthermore, Swap mutation has a statistically significant difference in average best fitness when compared to the other mutation operators. Thus, we concluded Swap mutation is the best performing one for our Sudoku problem.

- **Robust Fitness Function:** In order to get the most of the algorithm, as previously said, two different fitness functions were tested. The second and more robust one was done to penalize more the individuals with bad fitness, increasing the range between individuals with good and bad fitness values (because of the weights attributed to each component of the function). Nonetheless, this fitness function did not improve on our previous results, so we decided to remain with the first one (that only considered the sum of duplicate digits in rows/columns/boxes).

4. Discussion

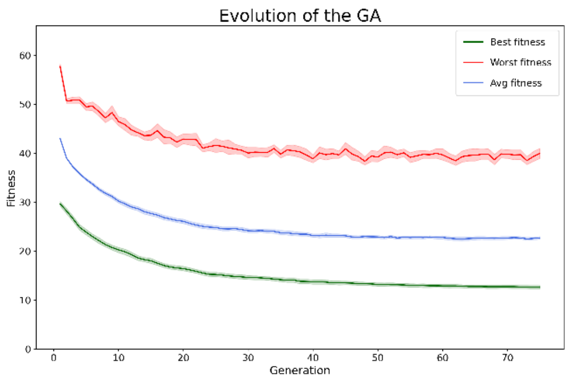


Figure 3. "Easy" puzzle.

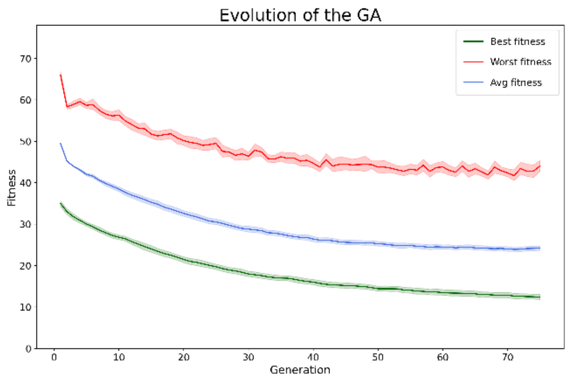


Figure 4. "Expert" puzzle.

After all the combinations of selection, crossover and mutation, it was found that the best combination was Tournament selection with Two-Point crossover and Swap mutation with a crossover probability of 0.7 and mutation probability of 0.9, for both puzzles that we tested: "easy" and "expert". As it can be seen in Figures 3 and 4, in both cases, the initial fitness of all individuals is high (between 30 and 70 duplicates in total), but as the generations pass, the average fitness of the population decreases and, in both cases, it is similar (around 25 duplicates). Also, the best fitness in both cases is very close to 10 duplicates. If the representation of the best individual is printed, it can be concluded that 10 duplicates represent around 6 numbers misplaced and the solution provided by the GA is very close to the solution of the puzzle. One output of our GA can be seen in Appendix B.

Therefore, it can be concluded that the GA almost achieved the global optimum in the “easy” and “expert” puzzles, having only 11 duplicates, approximately. However, this could not be achieved and one possible reason is premature convergence and lack of diversity in the individuals. To overcome this limitation, some strategies were implemented.

The first one was to create populations with more than 1 000 individuals, to get more diversity and further explore the fitness landscape. The second one was the implementation of Uniform crossover, to try to avoid premature convergence. The third one was to decrease the crossover rate along the generations with an *alpha* value. This could lead to less disruption in the individuals’ genotype. However, having this decrease in the crossover rate made the results worse in comparison with a constant crossover rate. The fourth one was to create new mutations taking into consideration the context of this problem, the Sudoku solver. Therefore, the Random and Intelligent Random mutations were created and implemented to reach improved results and to see if the GA was able to get to the global optimum. However, this did not happen, since these mutations led to results that were worse than the best result we had.

Although several strategies to overcome premature convergence and lack of diversity were applied, none of them made the GA reach the global optimum, finishing always in a local optimum. Nonetheless, the solutions from our best GA configuration were very close to it.

5. Conclusions

The objective of this project was to use GAs to evolve solutions for a Sudoku problem. The implementation of the algorithm was made by creating an initial population and evolving it implementing a selection method and genetic operators, evaluating it using a fitness function. After testing and analyzing a set of combinations of the implementation of three selection methods, three crossover operators and six mutation operators and for different levels of difficulty, it can be concluded that the best combination for solving the “easy” and the “expert” Sudoku is using Tournament selection, Two-Point crossover and Swap mutation.

Moreover, it can be concluded that the GA with this combination of parameters was close to achieve the perfect solution, since it had only 11 duplicates. Our GA could solve puzzles of any degree of difficulty and get good results. However, some strategies to overcome the premature convergence and lack of diversity were applied, but none of them made the GA reach the global optimum, finishing always in a local optimum. In general GAs are algorithms that sometimes do not reach global optimum, but ours found satisfactory results.

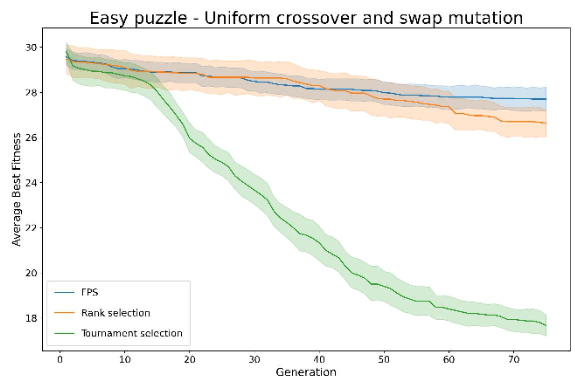
References

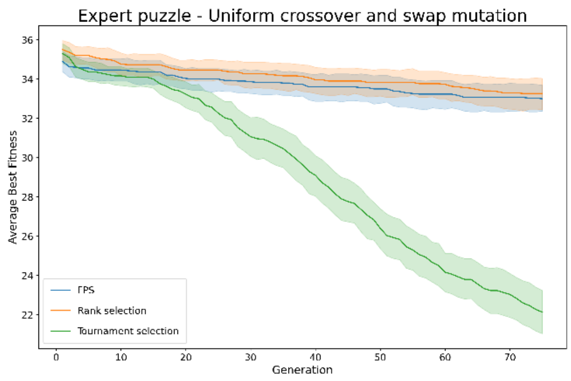
The Guardian (2005). *So you thought Sudoku came from the Land of the Rising Sun*. Available at: <https://www.theguardian.com/media/2005/may/15/pressandpublishing.usnews> [Accessed 15 May 2021].

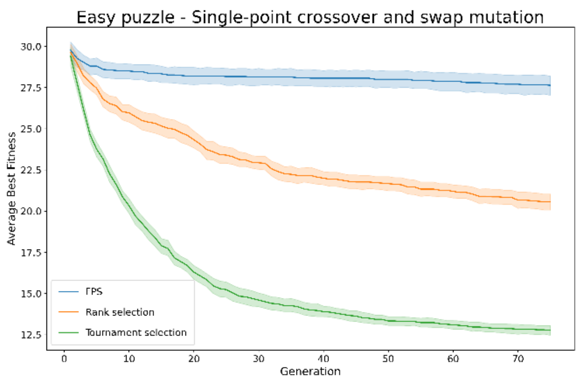
Syswerda, Gilbert. (1989). *Uniform Crossover in Genetic Algorithms*. Proc. 3rd Intl Conference on Genetic Algorithms 1989.

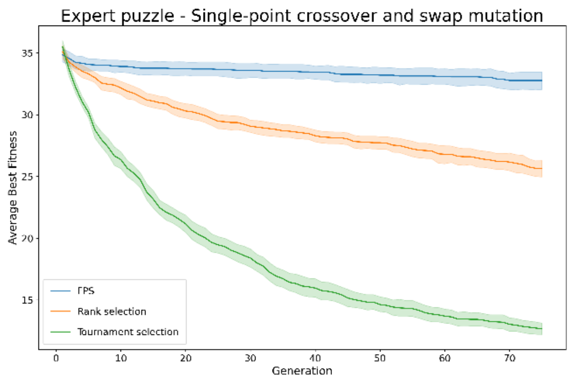
Appendices

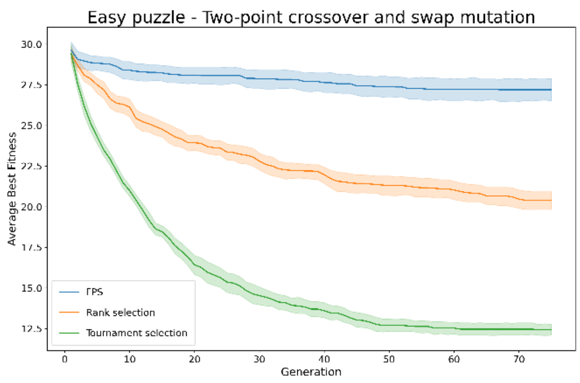
Appendix A: Comparison of GAs for the selection methods.

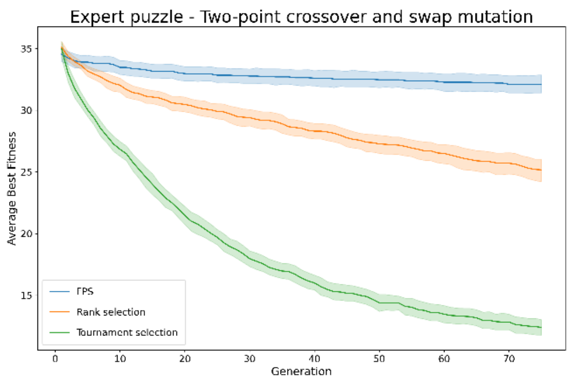




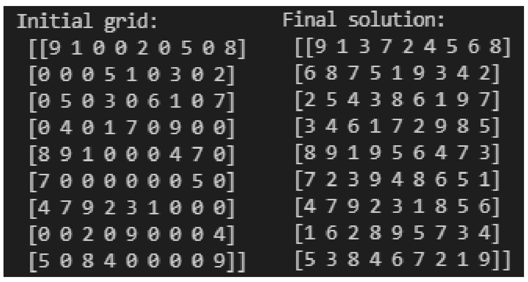








Appendix B: Example of an output on the “easy” puzzle, with fitness = 13.



```
Initial grid:             Final solution:
 [[9 1 0 0 2 0 5 0 8]      [[9 1 3 7 2 4 5 6 8]
 [0 0 0 5 1 0 3 0 2]       [6 8 7 5 1 9 3 4 2]
 [0 5 0 3 0 6 1 0 7]       [2 5 4 3 8 6 1 9 7]
 [0 4 0 1 7 0 9 0 0]       [3 4 6 1 7 2 9 8 5]
 [8 9 1 0 0 0 4 7 0]       [8 9 1 9 5 6 4 7 3]
 [7 0 0 0 0 0 0 5 0]       [7 2 3 9 4 8 6 5 1]
 [4 7 9 2 3 1 0 0 0]       [4 7 9 2 3 1 8 5 6]
 [0 0 2 0 9 0 0 0 4]       [1 6 2 8 9 5 7 3 4]
 [5 0 8 4 0 0 0 0 9]]      [5 3 8 4 6 7 2 1 9]]
```