NOVA IMS Information Management School

Computational Intelligence for Optimization

Project Report – Sudoku Solver

Group: Optimus Prime

Authors:

Guilherme Valler Moreira (20230538)	20230538@novaims.unl.pt
Michelle Buston Vega (20230590)	20230590@novaims.unl.pt
Raquel Rocha (20230188)	20230188@novaims.unl.pt

GitHub Repository

Contents

1 Objective

This project aims to explore Genetic Algorithms (GAs) and demonstrate how they can be applied in the design and development of a solution for a Sudoku puzzle.

This process included defining a fitness function that would allow us to find the final solution for each Sudoku puzzle and developing several Genetic Algorithm techniques, using different selection, crossover, and mutation operators. These different techniques are then combined and compared, with the final aim of determining the most effective methods and combinations.

2 Methodology

2.1 Conceptualization and Problem Definition

Prior to the development of this project, the lecturer provided a library, which was the baseline for the code used. However, significant changes were made, as the algorithms provided had to be adjusted to the problem, just like the Individual and Population classes had to be designed specifically for a Sudoku puzzle.

The most important components included in the library of this project are:

- **Classes**: In our implementation, we use the class "Individual" to represent a Sudoku board with a specific solution. This class organizes the board into rows, columns, and blocks, storing fixed positions. Initialization by blocks was tested, but unsuccessful, therefore the used initialization fills the Sudoku grid with random values in the empty positions of each row, ensuring the validity of the board for each row (no duplicates).

  The Population class contains the management and evolution of a generation. The selection of parents, crossover, and mutation operations are all within the Population class. Through the *evolve* function, the class iteratively improves the population over multiple generations, adjusting mutation and crossover probabilities based on the performance of the current population. Elitism can be used to preserve the best-performing individuals and includes checks for population improvement to ensure continuous progress toward finding an optimal solution.
- **Fitness Function**: We designed a fitness function to evaluate how closely a given solution meets the Sudoku constraints. This function calculates the number of duplicate values in columns and blocks, as there is no need to check for duplicates in rows due to our initialization ensuring valid rows.
- **Constants**: To test the performance of the algorithms, we defined three specific Sudoku puzzles with different difficulty levels: easy (81% of people can solve), medium (57% can solve) and hard (29% can solve). While we experimented with all three levels, this report will primarily focus on the easy Sudoku puzzle. The detailed configurations and results for the easy Sudoku puzzle that was used can be found in the appendix.
- **Genetic Algorithms**: These algorithms, including selection, crossover, and mutation, are implemented to evolve the population of solutions toward an optimal solution.

2.2 Implemented Methods and Operators

In this section, the Genetic Algorithms used and developed are described. Below, is a table with all the operators and methods implemented.

Methods and Operators	**Implemented Variations**
Selection Methods	1. Roulette Wheel Selection (FPS)
	2. Boltzmann Selection
	3. Tournament Selection
Crossover Operators	1. Partially Mapped Crossover (PMX) (Row)
	2. Single Point Crossover (Row)
	3. Single Point Crossover (Block)
	4. Uniform Crossover (Row)
Mutation Operators	1. Inversion Mutation
	2. Random Mutation
	3. Swap Mutation
Replacement Approach	1. Standard Replacement (Baseline)
	2. Elitism Replacement

Table 1: Genetic Algorithms Methods and Operators

In our selection methods, we went beyond the class-taught Tournament and Roulette Wheel Selection (FPS), and also included Boltzmann Selection, using a temperature setting of fifty. Moreover, we used a tournament size of ten for Tournament Selection. For crossover operations, we've included Uniform Crossover. With these additions, we wanted to diversify and improve the range of our Genetic Algorithms.

The Single Point Crossover method was, at first, tested on a block level, however, it resulted in problems with duplicates when running the evolve function, which we were not able to solve. The approach was then changed, successfully, to rows. From that point onwards, all implementations were done at a row level.

All combinations of these methods were tested with and without Elitism, leading to fifty-four different Elitism, Selection, Crossover and Mutation combinations. The *evolve* function was run ten times for each combination, with a population size of two hundred and a cap of one thousand generations.

To ensure that the algorithm didn't stagnate, we implemented a *check_improvement* function, that evaluated the performance of the population, and adjusted the mutation and crossover probabilities if the population did not show any improvement after ten generations, probably reaching a local minima. This way, it was possible to explore more solution spaces when the algorithm was not providing the expected results. If there is no improvement after thirty generations, the method resets the population entirely.

Regarding Elitism, the selected range was set to ten, meaning that the ten fittest individuals from each generation were preserved without any changes and directly passed on to the next generation.

3 Results and Discussion

3.1 Elitism vs. No Elitism

As previously mentioned, we tested all combinations both with and without Elitism to assess its impact on the results. At first glimpse, it was evident that the results without Elitism were inferior to those with it, as seen in the two tables below. It's also relevant to mention that the results below refer to the Easy Sudoku puzzle.

No combination without Elitism solved all puzzles, and the average number of generations required to reach a solution was nearly double compared to when using Elitism. With Elitism, the best configuration achieved an impressive average of 268.8 generations to solve ten puzzles, a great improvement compared to the best combination without Elitism, which solved only six puzzles with an average of five hundred and seventy-eight generations.

All the top ten performing configurations with Elitism solved at least six puzzles, with the best two configurations solving all of them. On the other hand, without Elitism, this was not possible, with only one combination managing to solve more than five puzzles.

After analyzing these findings, the group decided to exclude configurations without Elitism from further consideration.

Selection	**Crossover**	**Mutation**	**Average Generations**	**Solved Puzzles**
Boltzmann Selection	Uniform XO	Random Mutation	578.1	6
Fitness Proportionate Selection	Uniform XO	Random Mutation	672	5
Boltzmann Selection	Uniform XO	Swap Mutation	526.1	5
Fitness Proportionate Selection	Uniform XO	Inversion Mutation	713.9	4
Fitness Proportionate Selection	Partially Mapped XO	Inversion Mutation	630.1	4
Boltzmann Selection	Partially Mapped XO	Swap Mutation	616.8	4
Fitness Proportionate Selection	Partially Mapped XO	Random Mutation	788.2	3
Fitness Proportionate Selection	Uniform XO	Swap Mutation	810.7	2
Tournament Selection	Uniform XO	Random Mutation	956.2	1

Table 2: Top Ten Genetic Algorithm Configurations without Elitism

Selection	**Crossover**	**Mutation**	**Average Generations**	**Solved Puzzles**
Boltzmann Selection	Uniform XO	Random Mutation	268.8	10
Fitness Proportionate Selection	Partially Mapped XO	Swap Mutation	311.2	10
Boltzmann Selection	Partially Mapped XO	Inversion Mutation	378.0	9
Boltzmann Selection	Uniform XO	Swap Mutation	408.1	9
Fitness Proportionate Selection	Partially Mapped XO	Inversion Mutation	424.8	9
Boltzmann Selection	Partially Mapped XO	Swap Mutation	349.2	8
Boltzmann Selection	Partially Mapped XO	Random Mutation	499.0	7
Tournament Selection	Partially Mapped XO	Swap Mutation	576.2	6
Fitness Proportionate Selection	Uniform XO	Swap Mutation	616.5	6
Fitness Proportionate Selection	Partially Mapped XO	Random Mutation	744.9	6

Table 3: Top Ten Genetic Algorithm Configurations with Elitism

3.2 GA Operators

The performance of the genetic algorithm configurations was evaluated using different selection, crossover, and mutation methods. Table 3 (above) summarizes the top ten results, including the average number of generations required to solve the puzzles and the number of solved puzzles. The complete table with all configurations can be found in table 4 in the appendix.

3.2.1 Selection Method Analysis

- **Boltzmann Selection (bs)** performed exceptionally well, with configurations solving nine to ten puzzles within a few generations, once with an impressive average of 268.8. When combined with weaker crossover and mutation methods, it naturally performed worse.
- **Fitness Proportionate Selection (fps)** was part of the most successful configurations, solving all 10 puzzles within an average of 311.2 generations when combined with PMX and Row Swap Mutation. However, in other combinations, it did not succeed as greatly.
- **Tournament Selection (ts)** was the least effective and is part of only one of the top ten configurations. It did not show great effectiveness, with only six puzzles solved within an average of 576.2 generations in the best case. We believe that by increasing the tournament size, we could improve its performance but we couldn't test it due to time constraints.

3.2.2 Crossover Operators Analysis

- **Row Single Point Crossover (rspxo)** showed the worst results, not even being present in the top ten configurations. Overall, this method was only able to solve three puzzles, and it's present in eight of the ten worst combinations.
- **Row Uniform XO (uxo)** showed a strong performance when paired with Boltzmann Selection and

Row Random Mutation, solving all ten puzzles in 268.8 generations, on average. When paired with Swap Mutation it was also able to solve nine puzzles in four hundred and eight generations. However, when combined with other mutation and selection methods, it was not as efficient.

- **Row Partially Mapped XO (rpmxo)** appeared most frequently in the top configurations, proving that it is actually efficient. Although it showed good results overall, it did not stand out as the best crossover method.

3.2.3 Mutation Operators Analysis

- **Row Swap Mutation (rsm)** was the most successful, appearing in the majority of the top configurations, contributing to solving up to ten puzzles with certain combinations.
- **Row Random Mutation (rrm)** is a part of the best combination, solving ten puzzles within two hundred and sixty-eight generations with Boltzzman Selection and Uniform Crossover. In other cases, its performance was within average.
- **Row Inversion Mutation (rim)** showed good results when combined with the best algorithms (such as PMX and Boltzmann Selection), however, in the remaining cases, the results were average.

3.3 Extras

As an extra step, we decided to run a single combination with a cap of ten thousand generations to see how it would perform. The combination used the following parameters:

- **Population size:** 200
- **Maximum generations:** 10.000
- **Selection method:** Tournament Selection
- **Tournament size:** 10
- **Boltzmann temperature:** 50
- **Crossover probability:** 80%
- **Crossover method:** Partially Mapped Crossover (row)
- **Mutation probability:** 10%
- **Mutation method:** Random Mutation

Given these parameters, the algorithm performed much better than its one-thousand-capped counterpart. It was able to solve ten out of ten solutions, with an average generation count of 1093. Given this huge difference between the two algorithms, as the ten-thousand-capped algorithm performed much better while needing very little above the one-thousand-generation cap, we believe that, as we are capping the generation count at a relatively low number, it comes down to a bit of luck in the initialization and reaching, or not, the global minima.

3.4 Summary

The analysis of the top ten GA configurations underscores the significant impact of the choice of selection, crossover, and mutation methods on solving Sudoku puzzles. Boltzmann Selection, particularly when paired with Row Uniform Crossover and various mutation techniques, consistently showed the best performance. These findings suggest that Boltzmann Selection and Row Uniform Crossover are highly effective in maintaining genetic diversity and promoting convergence toward optimal solutions. As mentioned previously, detailed results of all configurations tried can be found in the appendix (table 4).

Although the gathered data is regarding the easy Sudoku, it's also relevant to mention that the algorithms also solved the medium puzzle, however, when it came to the hard one, it stagnated reaching a minimum fitness of two.

4 Conclusions and Future Work

Overall, the group is satisfied with the results, as we successfully solved the easy and medium Sudoku puzzles with relatively low population sizes and a limited number of generations. However, due to time constraints, we couldn't increase these values, as the algorithms would have required too long to run.

In the future, it would be worthwhile to increase the population size and the number of generations and experiment with different parameter settings. This could help us determine if our algorithms could eventually solve the hard Sudoku puzzle. Trying other operators and methods like Cycle Crossover would also be a good addition to our project, as well as different fitness functions.

In conclusion, the group considers the project's outcome successful, although there are areas for improvement. Choosing the Sudoku was a good decision, as it allowed us to achieve solid results while working on an interesting and engaging problem.

5 Division of Labor

The classes Individual and Population were developed by Guilherme.

The entire group participated in the development of the algorithms. At first, it was up to each member to try different implementations independently, so that everyone could understand how they are done and include more diversity. Ultimately, we used Guilherme's implementations of the Boltzmann and Tournament Selection, his mutation operators, and his Single Point Crossover and PMX implementations, while Raquel provided the code for Uniform crossover and FPS selection.

The three group members participated in the running and testing of the code, which provided the information used for the final results.

Finally, the report was mainly written by Michelle and Raquel.

Appendices

Selection	**Crossover**	**Mutation**	**Average Generations**	**Solved Puzzles**
Boltzmann Selection	Uniform XO	Random Mutation	268.8	10
Fitness Proportionate Selection	Partially Mapped XO	Swap Mutation	311.2	10
Boltzmann Selection	Uniform XO	Swap Mutation	408.1	9
Fitness Proportionate Selection	Partially Mapped XO	Inversion Mutation	424.8	9
Boltzmann Selection	Partially Mapped XO	Inversion Mutation	378.0	9
Boltzmann Selection	Partially Mapped XO	Swap Mutation	349.2	8
Boltzmann Selection	Partially Mapped XO	Random Mutation	499.0	7
Fitness Proportionate Selection	Partially Mapped XO	Random Mutation	744.9	6
Fitness Proportionate Selection	Uniform XO	Swap Mutation	616.5	6
Tournament Selection	Partially Mapped XO	Swap Mutation	576.2	6
Fitness Proportionate Selection	Uniform XO	Random Mutation	715.2	5
Boltzmann Selection	Uniform XO	Inversion Mutation	681.7	5
Tournament Selection	Partially Mapped XO	Random Mutation	857.1	4
Fitness Proportionate Selection	Uniform XO	Inversion Mutation	836.0	4
Tournament Selection	Partially Mapped XO	Inversion Mutation	899.0	3
Tournament Selection	Uniform XO	Random Mutation	890.9	3
Tournament Selection	Single Point XO	Swap Mutation	844.1	2
Tournament Selection	Uniform XO	Swap Mutation	992.3	1
Boltzmann Selection	Single Point XO	Swap Mutation	946.7	1
Tournament Selection	Uniform XO	Inversion Mutation	932.5	1
Boltzmann Selection	Single Point XO	Inversion Mutation	1000.0	0
Boltzmann Selection	Single Point XO	Random Mutation	1000.0	0
Fitness Proportionate Selection	Single Point XO	Inversion Mutation	1000.0	0
Fitness Proportionate Selection	Single Point XO	Random Mutation	1000.0	0
Fitness Proportionate Selection	Single Point XO	Swap Mutation	1000.0	0
Tournament Selection	Single Point XO	Inversion Mutation	1000.0	0
Tournament Selection	Single Point XO	Random Mutation	1000.0	0

Table 4: All Genetic Algorithm Configurations and Their Performance (With Elitism)

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

(a) Easy Sudoku Puzzle

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

(b) Easy Sudoku Solution

Figure 1: Sudoku Puzzle and Solution

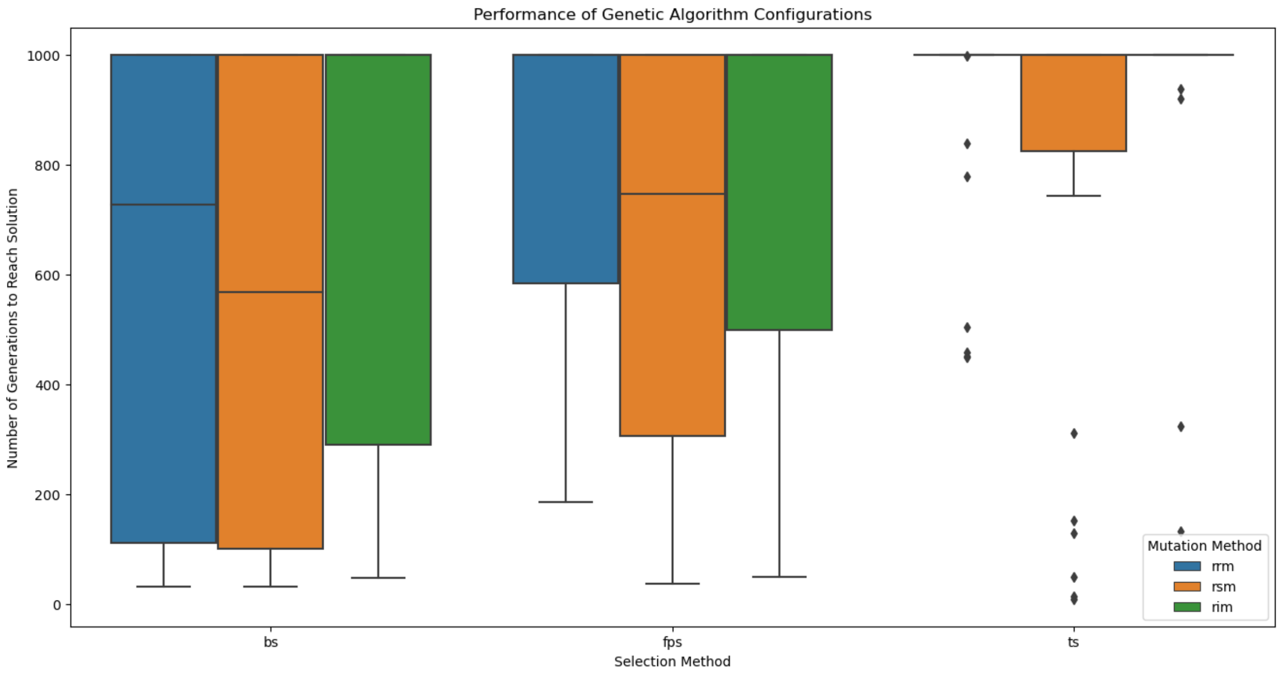

Figure 2: Performance of GA Configurations by Selection and Mutation Method (with Elitism)

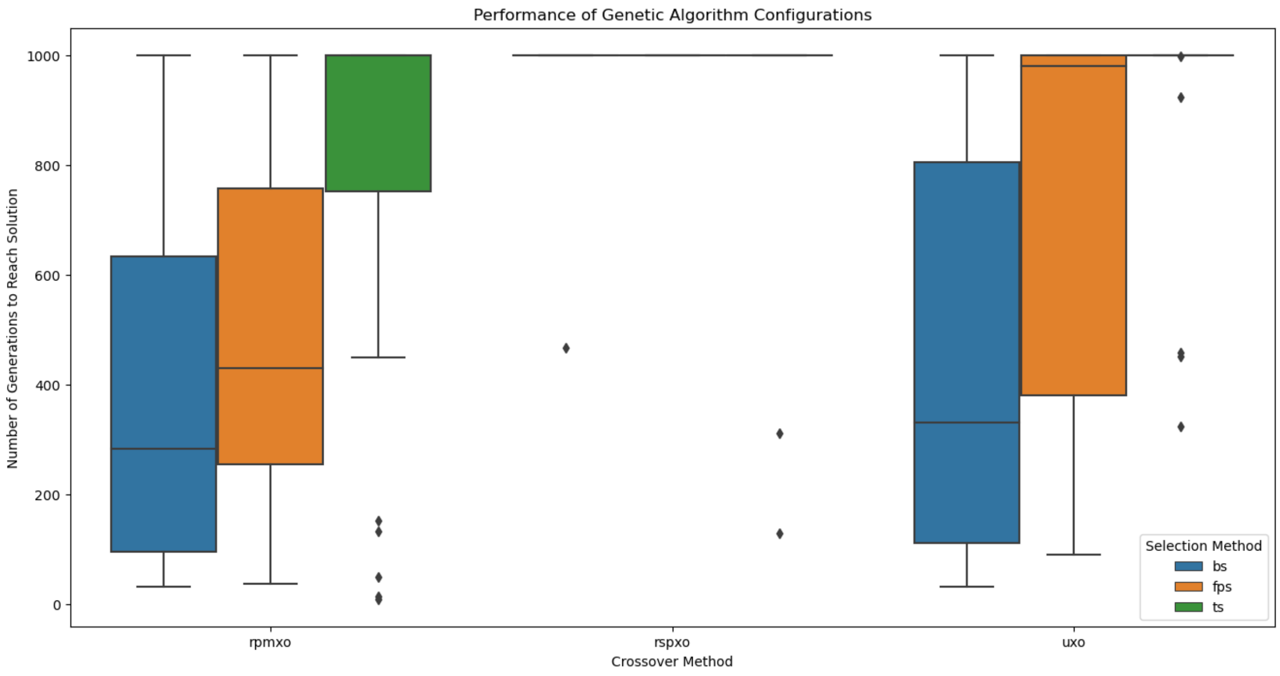

Figure 3: Performance of GA Configurations by Selection and Crossover Method (with Elitism)

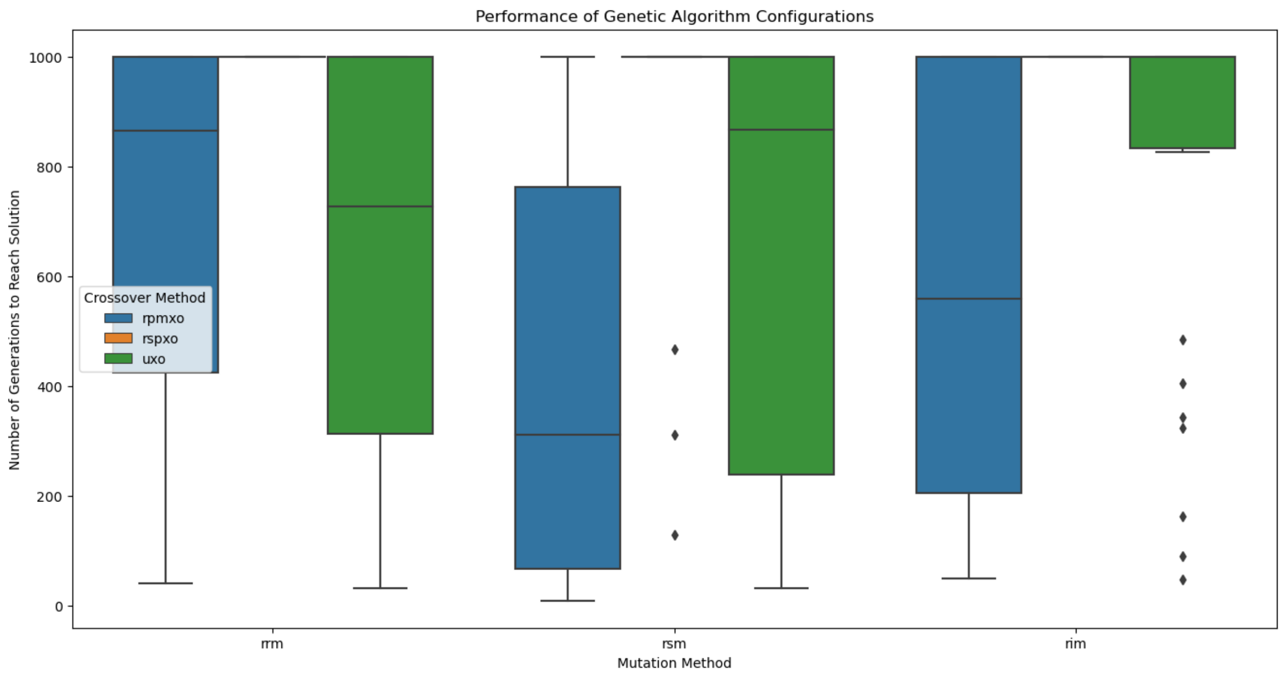

Figure 4: Performance of GA Configurations by Mutation and Crossover Method (with Elitism)