COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION (CIFO)

REPORT ON PROJECT

“Using Genetic Algorithms to solve a Sudoku puzzle”

Group 3

avarela1963/CIFO_PROJECT_GROUP_3: This is a Repo for CIFO project delivery (github.com)

students:

2014553 António Varela

20191217 Pedro Paris

20230978 Bernardo Pinto Leite

20211313 Beatriz Teixeira

Introduction

Brief Introduction to Sudoku Puzzles

Sudoku is a popular number placement puzzle involving a 9x9 grid divided into nine 3x3 subgrids. The goal is to fill the grid with digits from 1 to 9, ensuring each row, column, and subgrid contains all digits exactly once. Originating in the late 18th century, Sudoku gained popularity in the 1980s through Japanese publisher Nikoli and is now a global phenomenon.

Objective

The objective is to develop a genetic algorithm (GA) to efficiently solve Sudoku puzzles. The GA will evolve a population of candidate solutions to find a valid Sudoku grid, optimizing the arrangement of digits.

Genetic Algorithms

Overview

Genetic algorithms (GAs) are inspired by natural selection and genetics. Developed by John Holland in the 1960s and 1970s, GAs evolve solutions over successive generations. They are useful for large search spaces where traditional optimization techniques may be inefficient.

Key Principles

- **Population**: A set of potential solutions.
- **Fitness Function**: Evaluates how close a solution is to the optimal.
- **Selection**: Chooses the fittest individuals for the next generation.
- **Crossover**: Combines genetic material from two parents.
- **Mutation**: Introduces random changes to maintain diversity.
- **Generation**: The process of creating a new population.
- **Termination**: Stops when a stopping criterion is met.

Methodology

Initial Population Generation

The initial population consists of potential Sudoku solutions, represented as individuals. Each individual is a complete Sudoku grid with shuffled numbers, maintaining the constraints of the initial puzzle.

For this project, the starting point for feeding the algorithm was a puzzle provided by the website https://sudoku.com/pt like the one in the figure.

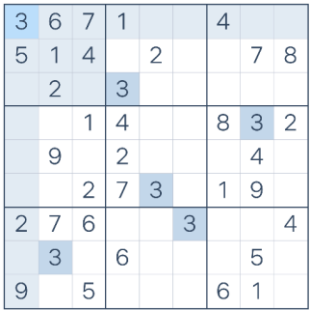

Figure: Example of sudoku puzzle use in this project

Fitness Function

The fitness function evaluates a Sudoku board based on row, column, and subgrid uniqueness, penalizing violations if there are repetition of numbers. This differentiation helps evolve more valid Sudoku boards.

Selection Process

The selection process uses weighted random selection to choose individuals for the mating pool, simulating a roulette wheel where fitter individuals have a higher chance of being selected.

Crossover and Mutation

- **Crossover**: Combines rows from two parents to create offspring.
- **Mutation**: Introduces random changes to maintain genetic diversity, ensuring occasional new solutions.

Parameters

- **Population Size**: 1000, 5000, 10000
- **Mutation Rate**: 1%
- **Crossover Rate**: 95%
- **Number of Generations**: 50

Implementation Steps

The algorithm is implemented as follows:

1. Generate initial population.
2. Evaluate fitness.
3. Select mating pool.
4. Apply crossover and mutation.
5. Evaluate new population.
6. Select best individuals for the next generation.
7. Repeat for 50 generations or until a solution is found.

Results

Results 1: variation of population and variation of number of iterations

Selection – fitness proportion

Cross over – row combination

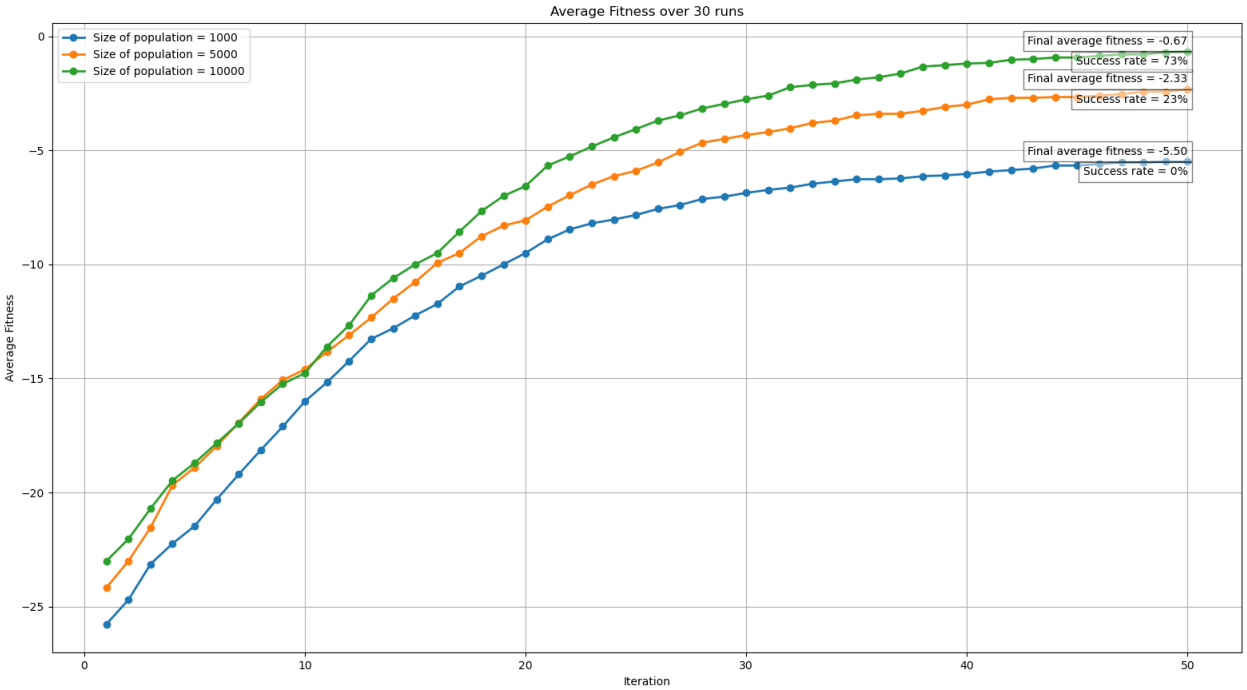

Figure: Performance variation of the algorithm with different population sizes

The plot shows that increasing the population size leads to better average fitness and higher success rates, with the best performance observed at a population size of 10,000 (final average fitness = -0.67, success rate = 73%). Smaller populations of 1,000 and 5,000 result in lower average fitness and success rates, demonstrating the importance of larger populations for effective optimization.

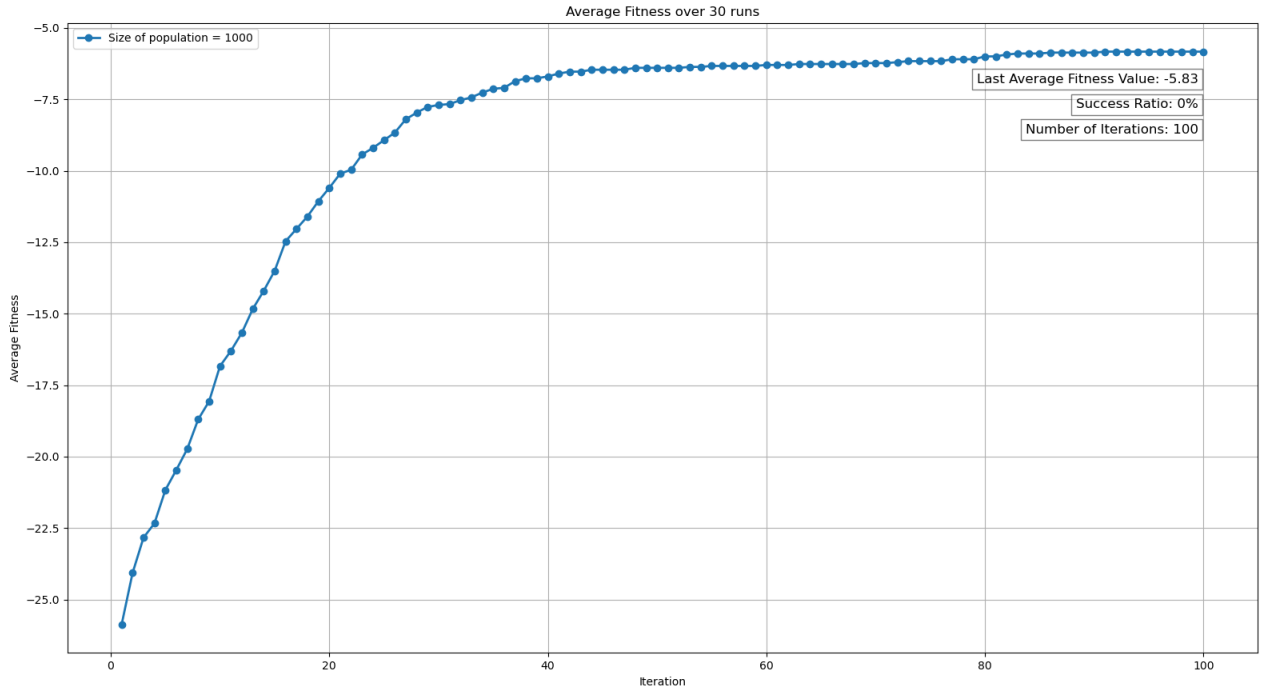

Figure: Performance Variation of the Algorithm with Increasing Number of Iterations

The plot shows that with a population size of 1,000 and 100 iterations, average fitness improves from -25 to -5.83 but plateaus after 40 iterations. Despite the optimization, the success rate remains at 0%, indicating no perfect solutions. This suggests that simply increasing iterations is insufficient for further improvement.

Results 2: Evaluation of Performance with Population Variations Using Different Crossover Functions

Here are the results for the cross_over function, where random columns from each parent are selected. The number of iterations is fixed at 50, and the experiment was conducted over 30 runs.

Selection – fitness proportion

Cross over – column combination

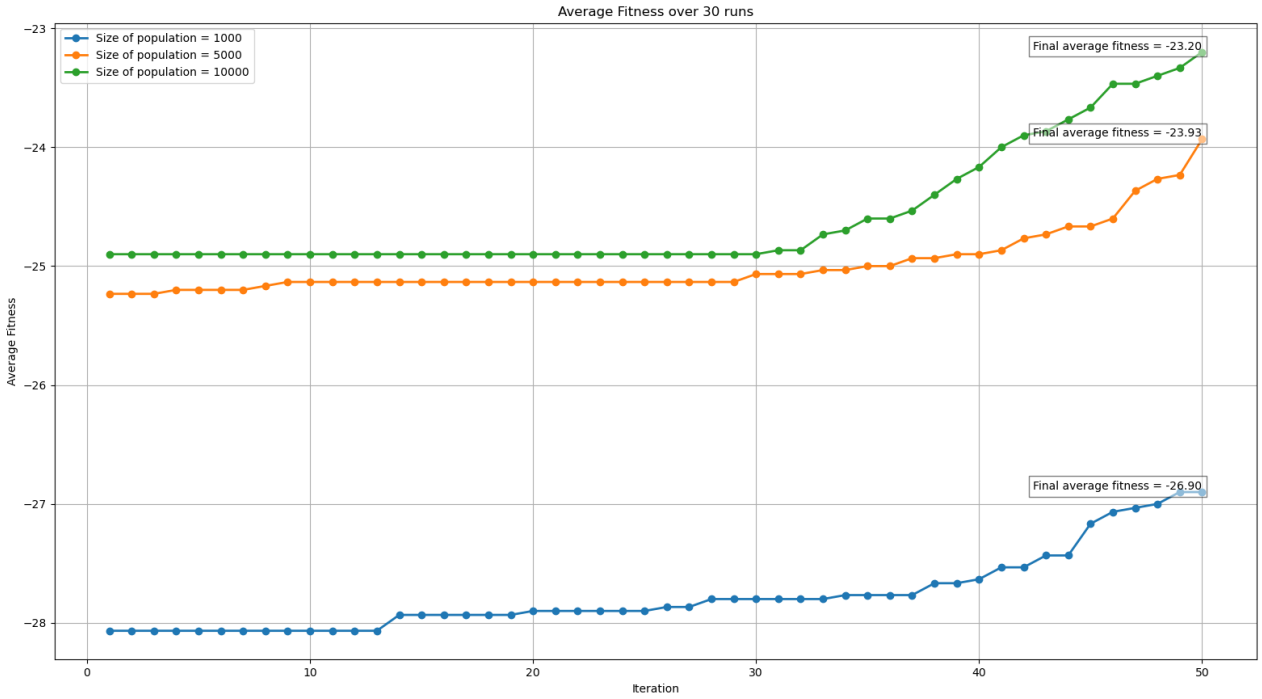

Figure: Performance Variation of the Algorithm with Different Crossover Functions

The plot shows that larger population sizes (5,000 and 10,000) result in better average fitness values compared to a population of 1,000. However, improvements are minimal and plateau early. The population of 10,000 achieves the best final average fitness at -23.20, indicating that a larger population is beneficial but with diminishing returns.

Results 3: : Evaluation of Performance with Population Variations Using Different selection Functions

Selection - tournament

Cross over – line combination

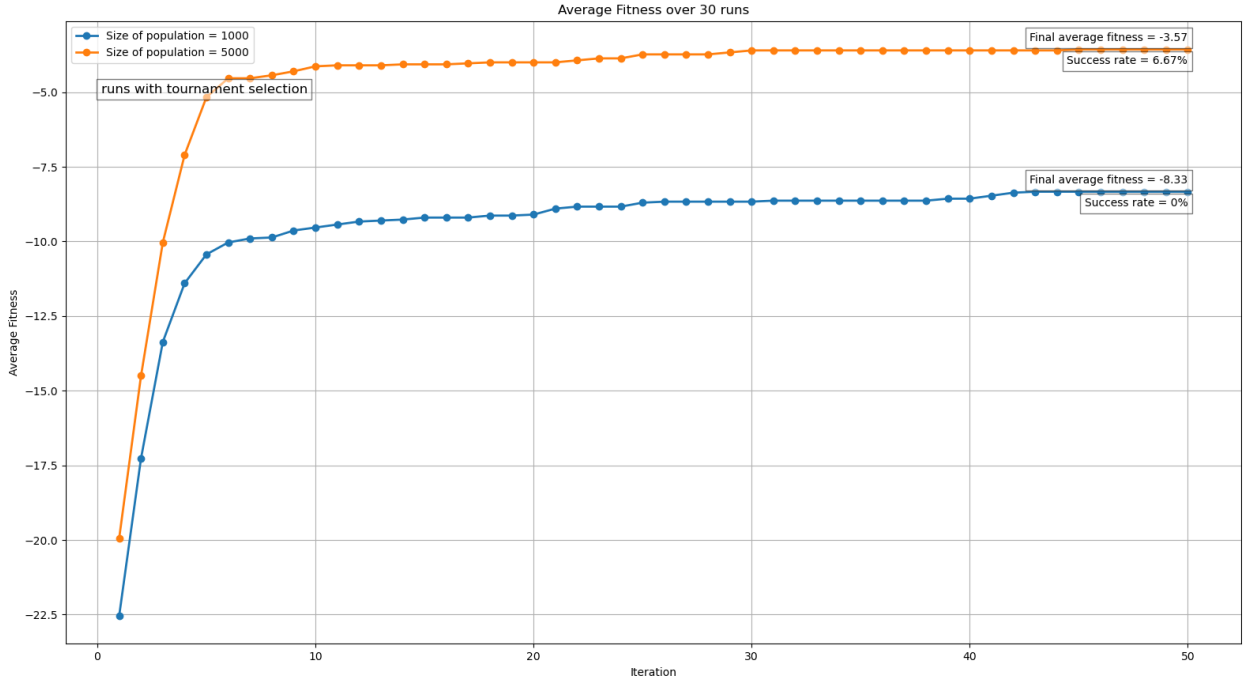

Figure: Algorithm Performance Variation with Different Selection Functions

The plot shows that using tournament selection with a population of 5,000 significantly improves average fitness, reaching -3.57 with a 6.67% success rate, compared to a population of 1,000, which only achieves -8.33 with a 0% success rate. Larger populations combined with tournament selection enhance performance and success rates, indicating the importance of both population size and selection method. The algorithm was so slow that no runs were made with a population of 10,000 individuals.

Results 4: Performance Evaluation of Algorithm with Varying Initial Sudoku Puzzle Difficulties Using the best Previous Configuration

Easy difficulty sudoku

After selecting the crossover method of alternating line interposition from the parent puzzles to generate the offspring puzzles, and choosing the fitness proportion selection method for the algorithm, we are now evaluating its performance with puzzles of varying difficulty. We will start with an easy puzzle from https://sudoku.com/pt/facil/, which has 38 filled cells out of the 81 total spaces in the grid (47%).

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

Figure: The easy Sudoku puzzle used as input data for the experiments.

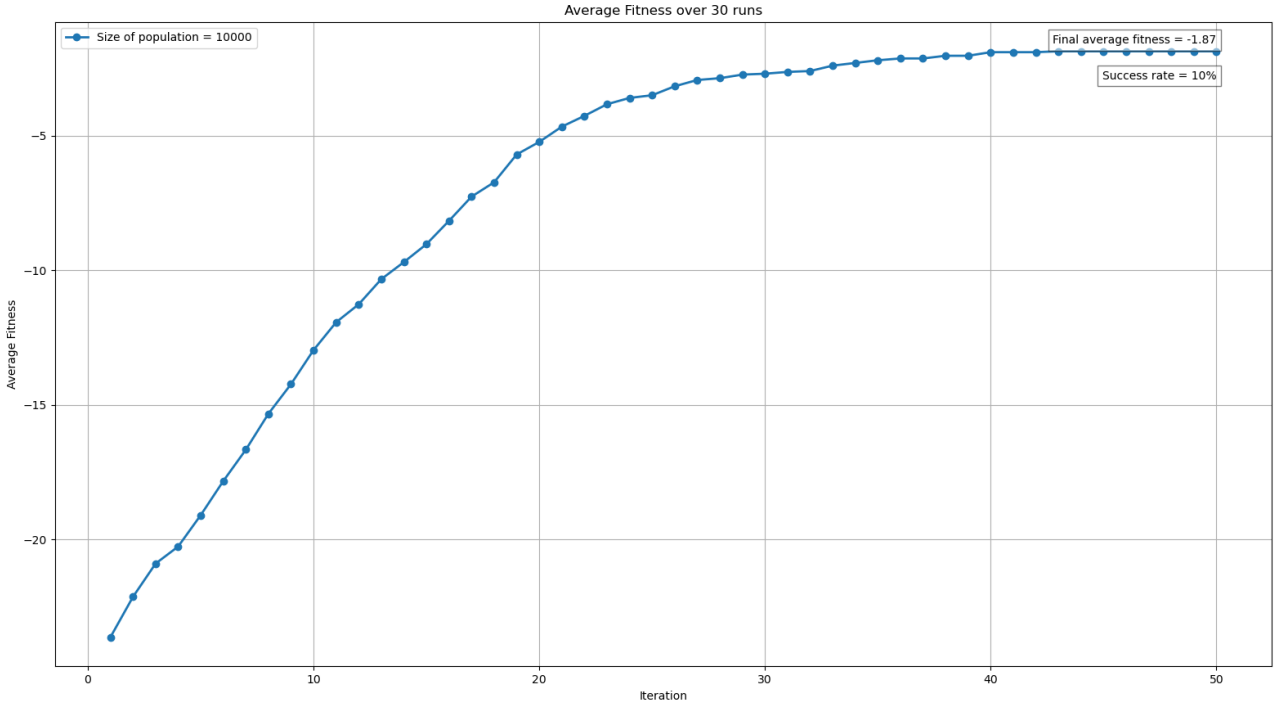

Figure: Results Obtained for the easy difficulty sudoku puzzle

Having conducted two sets of runs with the algorithm configured with fitness proportion selection and row combination crossover, using a population of 10,000 individuals, we observed slightly different results in each group. The success rate in one group was 24%, while in the other it was only 10%. However, the average fitness value achieved was the same in both groups.

Average difficulty sudoku

Having obtained a medium-difficulty Sudoku puzzle from the website https://sudoku.com/pt/medio/, we proceeded with new runs of the algorithm using the same configurations as before.

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

Figure: The medium difficulty Sudoku puzzle used as input data for the experiments.

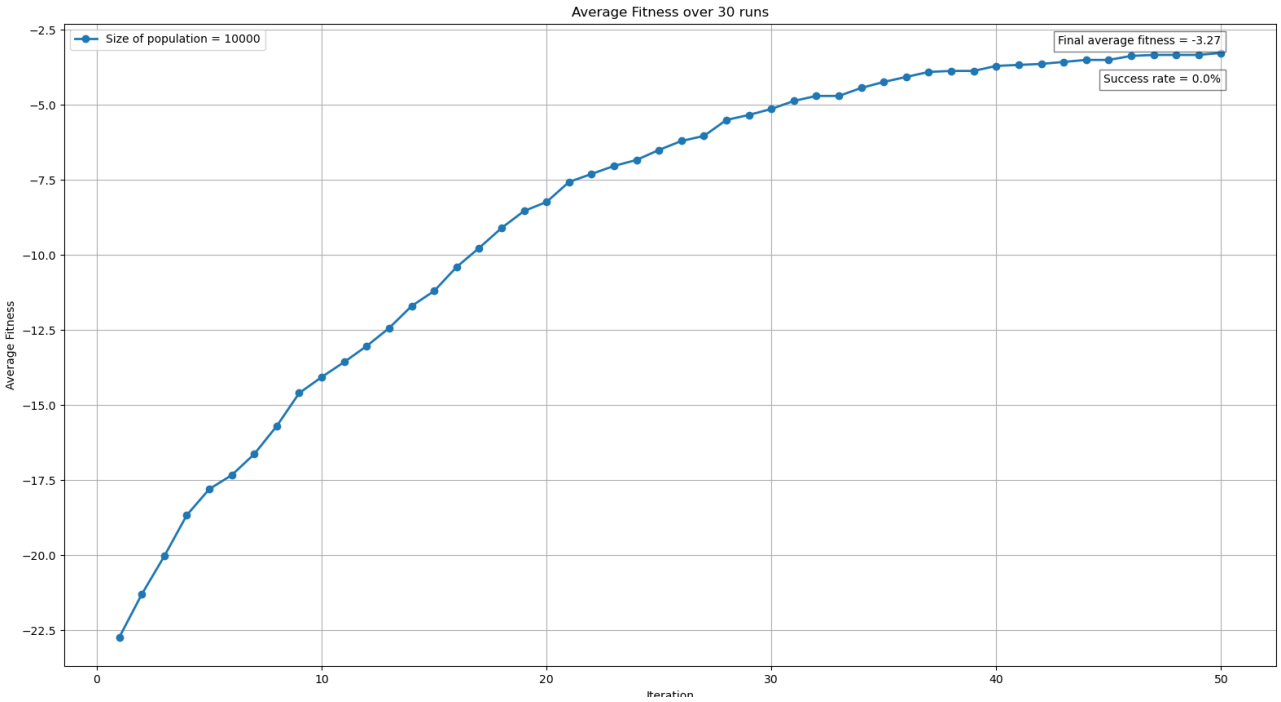

Figure: Results Obtained for the medium difficulty puzzle

The plot shows the average fitness over 30 runs for a population size of 10,000 individuals. The average fitness improves steadily, starting from around -22.5 and reaching approximately -3.27 by the final iteration, indicating effective optimization. However, the rate of improvement slows after 25 iterations, suggesting a plateau. Despite the fitness improvement, the success rate remains at 0.0%, implying no perfect solutions were found. This may point to the problem's difficulty or the need for further algorithm tuning.

Enhanced Difficulty Sudoku

Having obtained a difficult Sudoku puzzle from the website https://sudoku.com/pt/dificil/, we proceeded with new runs of the algorithm using the same configurations as before.

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

Figure: The enhanced difficulty Sudoku puzzle used as input data for the experiments.

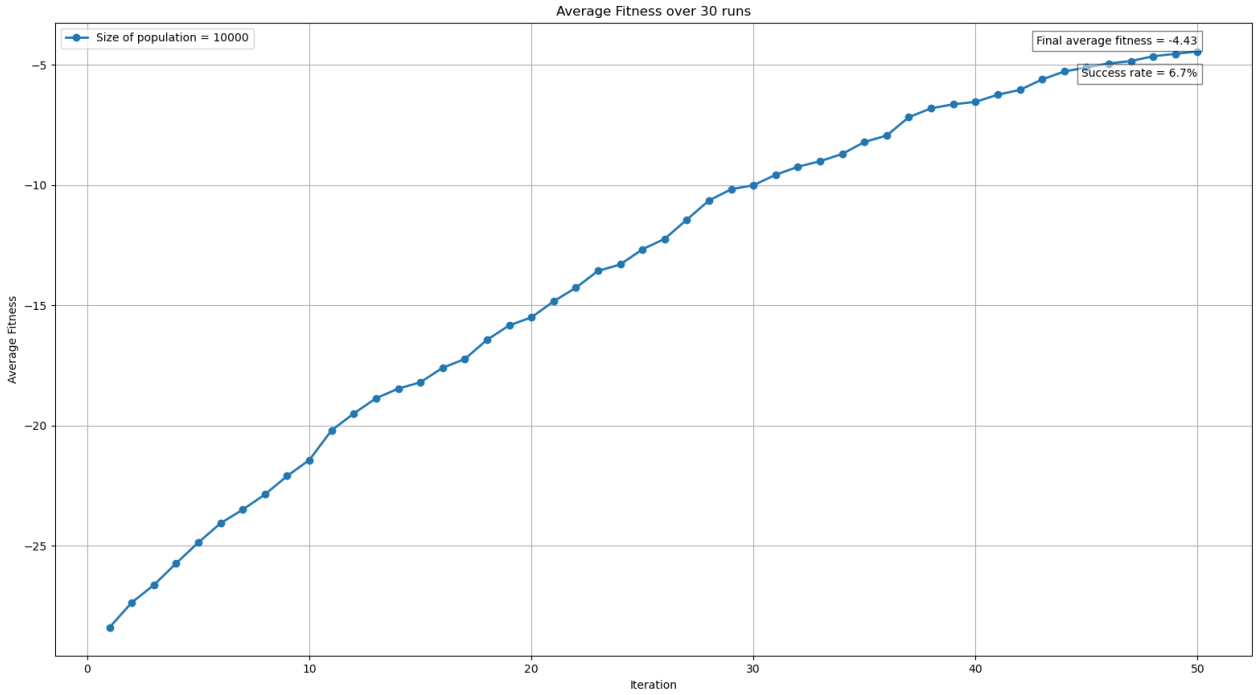

Figure: Results Achieved for the Advanced Difficulty Sudoku Puzzle

The plot shows the average fitness over 30 runs for a population of 10,000 individuals solving a difficult Sudoku puzzle. The fitness improves from around -27.5 to approximately -4.43, indicating effective optimization. The improvement rate remains steady, showing no significant plateau. The success rate is 6.7%, indicating some runs achieved perfect solutions. Overall, the algorithm shows effective optimization and a noticeable improvement in success rate.

Comparison of performance

The plot below compares the average fitness over 30 runs for Sudoku puzzles of varying difficulties (easy, medium, and difficult) with a population size of 10,000 individuals.

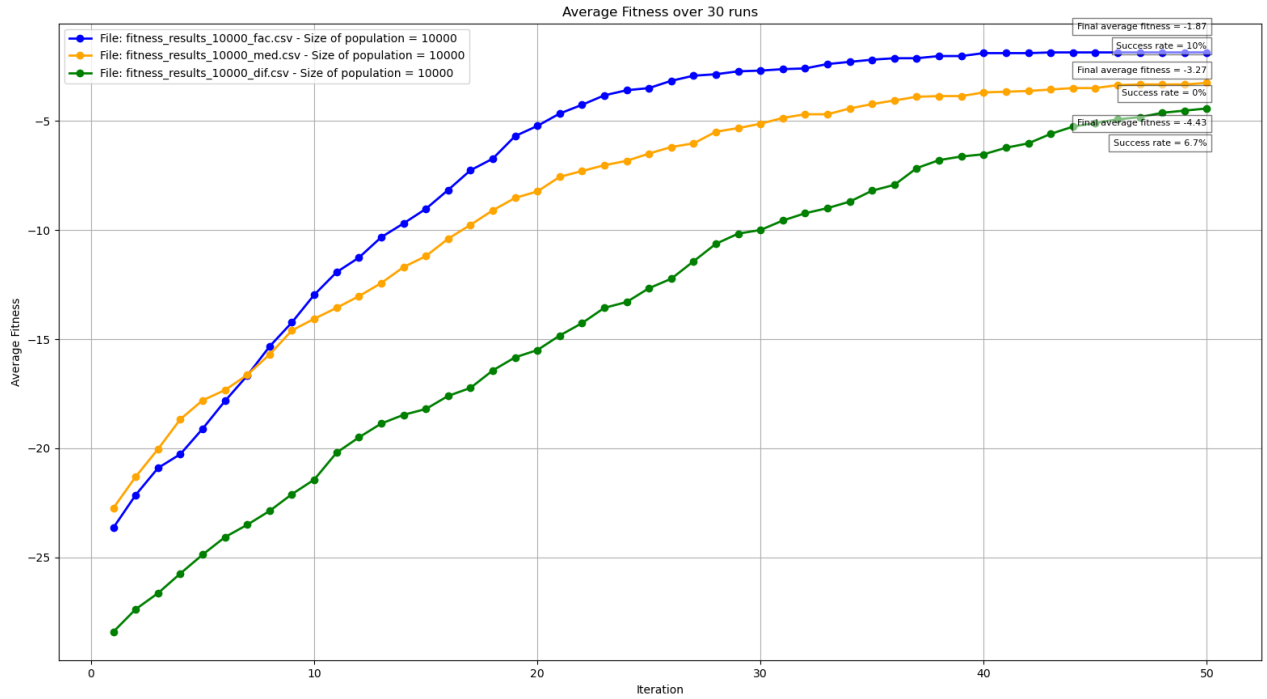

Figure: Algorithm Performance Across Puzzles of Varying Difficulty Levels

Key observations include:

Overall, the results show consistent improvement in average fitness across all difficulties over the iterations, demonstrating the algorithm's effective optimization. For the easy puzzles (blue), the fitness starts around -15 and improves to approximately -1.87, achieving a 10% success rate. Medium puzzles (orange) begin around -20 and improve to approximately -3.27, though they have a 0% success rate. Difficult puzzles (green) start near -27 and improve to approximately -4.43, with a 6.7% success rate.

Regarding convergence, easy puzzles converge faster and reach a higher average fitness, while medium and difficult puzzles show slower convergence and lower final fitness values. Success rates vary significantly: easy puzzles have the highest success rate at 10%, difficult puzzles follow at 6.7%, and medium puzzles have no successful runs, reflecting their higher complexity. Fitness trends exhibit a pronounced improvement rate in the early iterations for all difficulties, gradually plateauing as iterations progress.

The plot effectively illustrates how the algorithm's performance varies with puzzle difficulty, highlighting the increased challenges posed by more complex puzzles. As puzzle difficulty increases, the final average fitness steadily decreases.

Discussion

Elitism was used, and it is a fundamental mechanism contributing to the results obtained by the algorithm.

There was no time to experiment with varying the crossover and mutation probabilities.

The consistent improvement across iterations demonstrates effective optimization, but the slower convergence and lower success rates for harder puzzles highlight areas for algorithm refinement. The stochastic nature of the results underscores the need for multiple runs to ensure reliability.One significant limitation is the time required to run the algorithms with a population of 10,000. Although this population size was found to yield the best results, it also considerably increases the computation time.

When the algorithm reaches a plateau, adjusting the mutation or crossover probability can improve diversity and potentially enhance performance. All plots were generated using the average results of 30 runs for each set of parameters. Despite variations between plots with the same parameters, these differences confirm the stochastic nature of the process.

Conclusion

From the base configuration, we tested a different crossover method of alternating columns and a different selection method, tournament selection. We also evaluated the time required to run 30 iterations of the algorithm to obtain the average performance.

The base configuration was found to be the best in terms of execution time for 30 runs, success rate, and final fitness achieved. Therefore, it was selected for further evaluation with Sudoku puzzles of different difficulties. From this, we found that the algorithm achieves, as expected, its best results with an easy puzzle.

We also investigated the impact of increasing the number of iterations and found that increasing the population size had a more significant impact on performance than increasing the number of iterations.