Report Assignment 2

Kadirov Saidaziz DSAI-02

27.11.2024

1. Algorithm Description

1.1 Representation

Each solution referred to as an **individuals**, is represented as a 9×9 matrix corresponding to the Sudoku grid. The cells with pre-filled numbers (givens) are kept constant throughout the evolutionary process, ensuring that only the empty cells are subject to variation. This representation maintains the integrity of the initial puzzle while allowing the Genetic Algorithm (GA) to explore possible configurations for the remaining cells.

1.2 Fitness Function

The **fitness function** evaluates how close an individual is to a valid Sudoku solution. It calculates the number of conflicts in rows, columns, and 3×3 subgrids:

- **Row Fitness:** For each row, the number of unique digits is counted. The fitness score increases by the difference between 9 and the count of unique digits.
- **Column Fitness:** Similarly, each column is evaluated for uniqueness, with the fitness score incremented by the number of missing unique digits.
- **Subgrid Fitness:** Each of the nine 3×3 subgrids is assessed, and the fitness score is adjusted based on the number of unique digits present.

A perfect solution has a fitness score of 0, indicating no conflicts across rows, columns, or subgrids.

1.3 Selection Mechanism

The **selection process** prioritizes individuals with lower fitness scores, promoting those closer to the optimal solution. The population is sorted based on fitness, and

the top-performing individuals are retained for the next generation. This elitism approach ensures that the best solutions are carried forward, maintaining a high-quality gen pool.

1.4 Crossover Operator

The **crossover** mechanism combines genetic material from two parent individuals to produce offspring. For each row in the Sudoku grid, the algorithm randomly selects the corresponding row from one of the parents with a 50% probability. This method preserves the structure of the parents while introducing variability, facilitating the exploration of new solution spaces.

1.5 Mutation Operator

The **mutation** operator introduces random changes to maintain genetic diversity within the population. Specifically, it swaps two non-fixed numbers within a randomly chosen row. This mutation occurs with a high probability (92%) to ensure continual variation and prevent premature convergence to suboptimal solutions.

1.6 Hyperparameters

The performance of the GA is influenced by several hyperparameters:

- **Population Size:** 2,500 individuals ensure a diverse gen pool.
- **Generations:** Up to 15,000 iterations allow enough opportunity for convergence.
- **Max Stagnation:** The algorithm restarts the population if no improvement is observed over 50 generations, preventing stagnation.
- **Mutation Chance:** A high mutation rate of 92% maintains diversity.
- **Elitism Count:** Retaining the top 600 individuals preserves the best solutions across generations.

These hyperparameters were fine-tuned using Python-based simulations to balance exploration and exploitation effectively.

2. Experimental Setup

2.1 Test Cases

The GA was evaluated on Sudoku puzzles categorized into four complexity levels: **Easy**, **Medium**, **Hard**, and **Expert**. Each level corresponds to a different number of initial givens and the techniques required for solving. Below are representative examples for each complexity level:

- **Easy**

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

- **Medium**

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

- **Hard**

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

- **Expert**

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

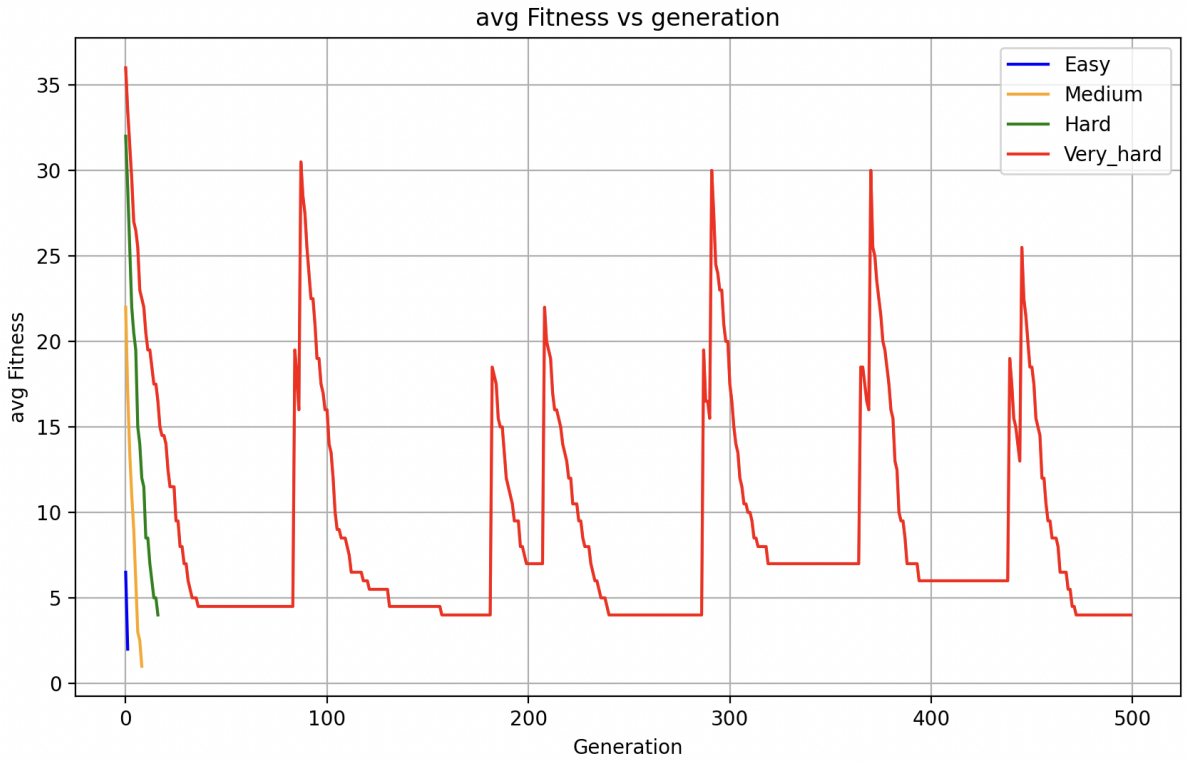
Reference: test cases are generated using the sudoku.name

3. Results

3.1 Statistical Analysis

The GA was run multiple times for each complexity level to gather statistical data on its performance. For each test case the average and maximum fitness scores of the population at the final generations were recorded. The results indicate that as the complexity increases, achieving a fitness score of 0 becomes more challenging and requires more generations.

Summary of Results



3.2 Fitness Plots

The following plots illustrate the relationship between the number of givens and the fitness scores achieved by the Genetic Algorithm in the final generations. The X-axis represents the number of givens, while the Y-axis shows the average and maximum fitness scores.

Note: The plots were generated using Python and demonstrate the GA's performance across different Sudoku complexities.

4. Conclusion

The implemented Genetic Algorithm effectively solves Sudoku puzzles by evolving candidate solutions through selection, crossover, and mutation. The algorithm demonstrates robust performance, particularly on puzzles with a higher number of givens. As complexity increases, the GA requires more generations to achieve optimal solutions, highlighting the trade-off between puzzle difficulty and computational effort. Future improvements could involve optimizing hyperparameters further or incorporating advanced genetic operators to enhance convergence rates.

all sources and full code presented in my Github: https://github.com/inopolis/intro-to-AI-assignments