Intro to AI
Assignment 2

Nikita Zagainov
DSAI-1

November 20, 2024

1 Implementation Details

The algorithm for solving Sudoku is implemented in `Python` and `C++` programming languages. The `Python` implementation is used for experiments and evaluation on test cases, while the `C++` implementation is used as submission on CodeForces.

For better availability of implementation details, I provide the source code on github: Link

1.1 Individual

As an individual for GA, I use matrices of size 9×9 to represent a Sudoku board with numbers from 1 to 9. each individual has a constraint to have specific numbers on specific cells, which were given in the initial sudoku board. This constrain is kept through the whole evolution process.

1.2 Fitness Function

The fitness function is used to evaluate individuals is the calculates the number of distinct numbers in each row, column, and 3×3 subgrid, then subtracts this number from 9 and sums all results up. The sudoku puzzle is solved when the fitness function returns 0.

1.3 Selection Process

During one generation of the algorithm, all individuals in population are evaluated on fitness function, then the best individuals are selected for the next generation and crossover process. Additionally, stagnation is checked to prevent the algorithm from running forever. If the best fitness is not improved for a certain number of generations, the algorithm resets the population entirely

1.4 Crossover

The crossover process is implemented as a simple random selection of rows between two parents and then combining them into a child.

1.5 Mutation

The mutation process is implemented as a random switch of two numbers in a randomly chosen row. Mutation happens with a certain probability with every individual in the end of the generation.

1.6 Hyperparameters

The set of hyperparameters was chosen using `BOHB` algorithm. The optimal set of hyperparameters:

- Population size: 1400
- Max stagnation: 50
- Mutation chance: 0.95
- Elitism count: 700

This set of hyperparameters was used for evaluation on `Python` implementation for tests. The final implementation in `C++` used increased sizes of population and elitism count to speed up the process.

2 Test Cases

We evaluated the genetic algorithm on various Sudoku test cases with different numbers of givens and complexity levels. The test cases were divided into easy, medium, hard levels. The definitions of these levels were taken from SudokuWiki.

The following numbers of givens were used:

- Easy: 42 givens

- Medium: 32 givens
- Hard: 28 givens

3 Evaluation

The evaluation was conducted on 30 different sudoku inputs (all of which can be found in the `puzzles` folder of GitHub repository).

Some examples of them:

3.1 Easy

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

3.2 Medium

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

3.3 Hard

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

3.4 Fitness Plot

The plot of convergence of the algorithm is shown in Figure 1. The plot shows average best fitness value over 10 different Sudoku puzzles for each difficulty level (30 tests in total).

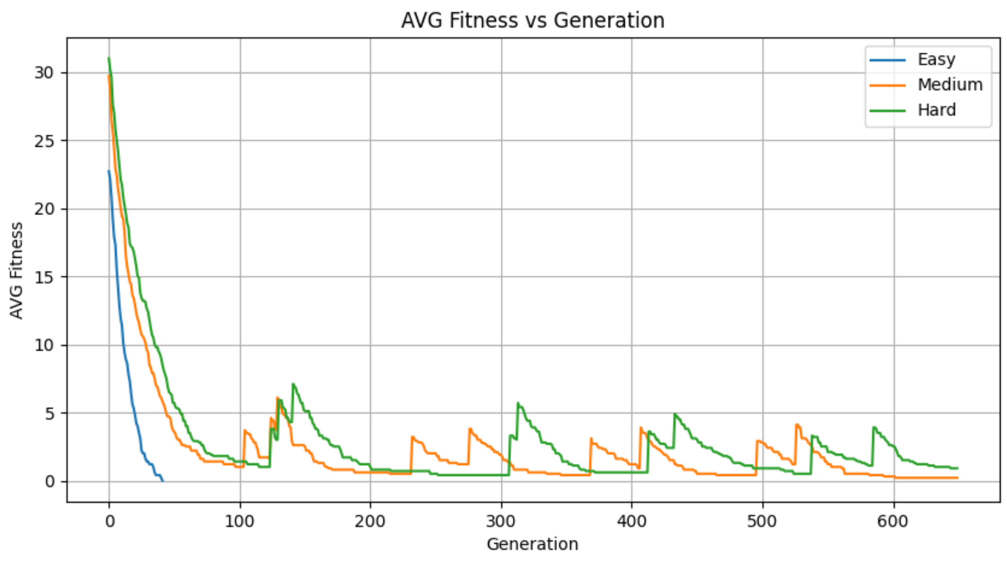

Figure 1: Average fitness value evaluated over 10 different Sudoku puzzles