INNOPOLIS UNIVERSITY

Introduction to Artificial Intelligence F24

Assignment 2 Report

Timofey Ivlev
B23-ISE-03
t.ivlev@innopolis.university

07.12.2024

Late date is due to sickness. I have DoE approved legal excuse.

Report of Assignment 2: Sudoku Solver

Program structure

First thing to say is that I used C# language (dot net v 6.0) for my solver. The evolutionary algorithm is implemented as a method in class Sudoku, which is main class of two classes of the program. Another class is SudokuGrid, which represents a 9x9 grid, including table of values ”values”, boolean table ”isGiven” which shows whether each value is given from problem statement.

Evolutionary algorithm flow

Before staring the actual algorithm, we read inputed initial grid, and count all available given numbers (which later used in fitness function). Then, we make an infinite loop, which restarts the Evolutionary Algorithm method, until we found a solution. Each restart we add mutation level but not bigger than 30% for each generation. (this rule is avoided when we have a long stagnation in fitness score of generation). The Evolutionary Alogtihm has two main stages, second stage contains several substeps.

- Creating initial population
  - We use priority queue for storing population (so that every time we fill it with new members, the 10 best grids with lowest ranks can be speedily piped)
  - We evaluate the fitness score of each member of population
- Generations loop
  - Before going into loop we initiate outer constants (generations counter, diversity selection boolean, guaranteed mutation boolean, size of diversity selection (which was set as 2 via experimenting)) and there are also lists for keeping history of minimum fitness function values of minimum element of population. (used to detect stagnation)
  - In loop, we create array ”temp” containing number (selection size, is set as 9 via experimenting) of best grids. Then array is filled from population array, which is cleared to be ready for new generation.

- Now, we need to perform number of checks of the received best population
  - If for 10 generations in a row there were no difference in min fitness score, we turn on diversity selection, now for each to best selection we add two random grid from population. This enhances variability of fitness score on crossover
  - Check for found solution. If best selection contains a grid with 0 fitness score we print this grid and finish the algorithm
- Calculate the variance of best population, and add to history of variance list. If for 10 generations with turned on diversity selection, the variance of fitness function of best population is still the same, we turn on enhanced mutation (now mutation chance is 100% (instead of 30%) for each child, and number of mutations is from 3 to 9 (instead of 1 to 3)
- Add minimum rank of best population grid, to track convergence
- Perform crossover for each member of best population, and calculate it rank (fitness function values). We get a lot children (best population squared minus repeated grids) and add these children to population array (which is priority queue, so we do not need to sort it)
- Increase the counter of generations and repeat the loop until the solution is found (fitness function converges)

Fitness Function

Fitness function calculates fitness score by adding squares of row and columns violations, and penalty values, which are calculated from counting duplicates within given values in rows and columns of initial grid. Penalty has a arbitrary big value 1000.

Mutation method

Mutation is performed via swapping arbitrary non given values in arbitrary 3x3 sub grids. There could be from 1 to 3 mutations if stagnation is not big, else mutation number is from 3 to 9.

Crossover

Crossover is performed via swapping arbitrary 3x3 subgrids from both parents.

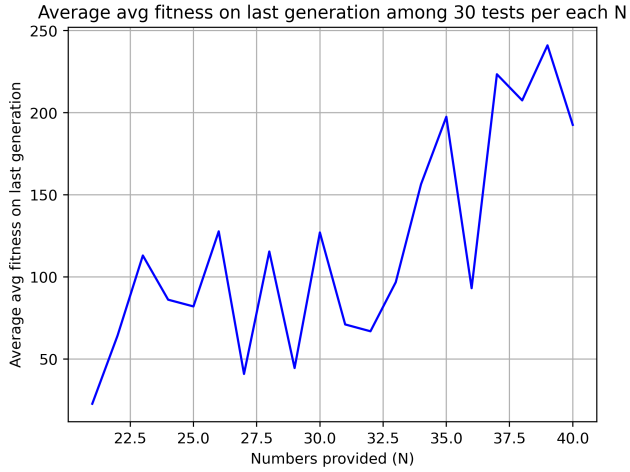
Specifics of variation parameters

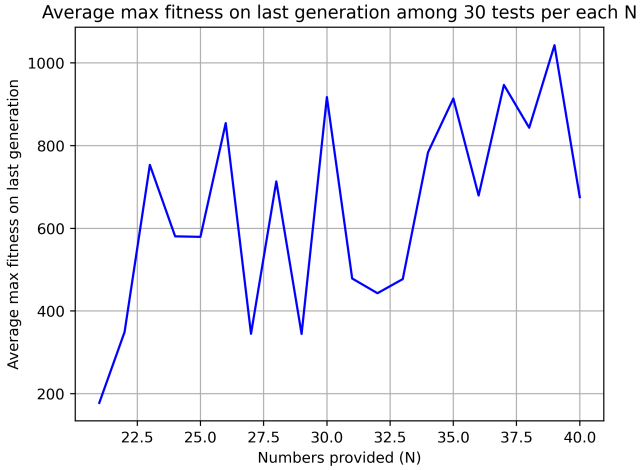
All of these parameters were found by experimenting.

- Critical number of repeated minimum fitness score ”FitnessCriticalNumber” is set as 240.
- ”InitialPopulationSize” sets the number of grids generated at the first interation of the algorithm
- ”SelectionSize” sets a number of best species taken from all population
- ”MaxPopulationSize” sets the maximum number of grids for picking diversity selection random numbers. Calculated as square of selection size - minus one selection size (in program selection size is 9, and max population size 9x9-9=72)
- ”MutataionBase” is number which is increased when there are many restrats (increasing muation chance from 20% to 30%)

Statistics plots

These statistics were generated with a help of python program created by my colleague student Gleb Popov Gleb's telegram.





Statistics data

The full resulted data with each test time shown can be found here. Here is the average values for each difficulty:
EASY
average time 0.5248787878787878
maximum fitness 745.6181818181818
average fitness 152.16060606060606

MEDIUM
average time 1.22225
maximum fitness 564.275
average fitness 82.2

HARD
average time 1.6935999999999998
maximum fitness 487.96666666666664
average fitness 73.68666666666667