**Genetic Algorithm: Sudoku Implementation**

**Problem Statement**

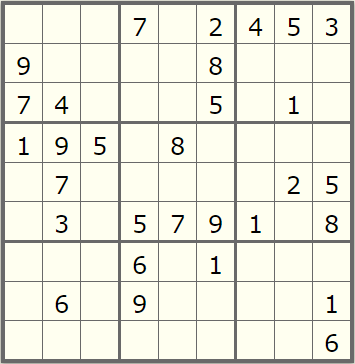
Given a broken 9×9 sudoku with at least a feasible solution and a scoring criterion, the problem is to implement the sudoku and try to find out the solution with the highest score. A completed 9×9 sudoku satisfies that each column, each row, and each of the nine 3 × 3 sub grids that compose the grid contains all the digits from 1 to 9 once.

**Solution Design**

*Genetic code*

Cut the broken sudoku into nine 3 × 3 sub grids. Each sub grid is a gene unit which is consist of the missing digits in the blanks from left to right, from top to bottom. A genetic code is formed by the nine gene units from left to right, from top to bottom. A genetic code ensures no duplicated digits in a gene unit.

e.g.



Sample

A possible genetic code corresponding to the sudoku above is shown as followed:

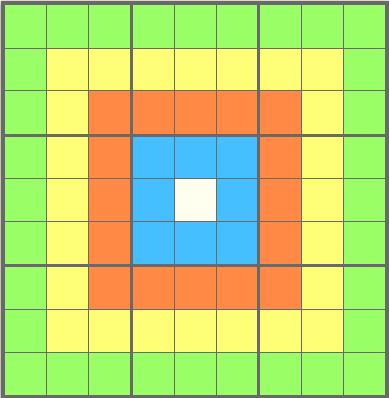
1 2 3 5 6 8 | 1 3 4 6 9 | 2 6 7 8 9 | 2 4 6 8 | 1 2 3 4 6 | 3 4 6 7 9 | 1 2 3 4 5 7 8 9 | 2 3 4 5 7 8 | 2 3 4 5 7 8 9

*Phenotype (Gene Expression)*

The phenotype of a sudoku is filling in the blanks within the sudoku using genetic code in the same order that it was retrieved.

*Fitness Function*

1. Fitness: A perfect fitness score for a completed sudoku is 144. Once there is a repetition in a row or in a column, the score will count down. A row or a column has repetitions 8 times at most. The genetic has ensured there is no repetition in a 3 × 3 sub grid, therefore, the total score is 8 × 9 rows + 8 × 9 columns = 144.
2. Score: For a completed sudoku which has a perfect score 144, the project used a scoring criterion to find out a higher-score sudoku.



Each color has its own score (Green = 3; Yellow = 5; Red = 10; Blue = 20; White = 30).

For each grid, the score is color times the digit filled in the grid.

If the sudoku is completed with no errors, its score is the sum of the grids minus 2700. If the sudoku is incomplete or has errors, its score will be 0.

*Crossover*

Two parents randomly choose some bits to exchange their gene points, which will produce two children. Only exchanging gene points will probably result in duplicated digits in the same gene unit, therefore, it needs a function to correct the error genes.

e.g. (take the first unit in the sudoku above as an example)

Parent 1: 1 2 3 5 6 8

Parent 2: 6 5 2 3 8 1

Swap in position 2, 3, 5 (marked in red)

Child 1: 1 5 2 5 8 8

Child 2: 6 2 3 3 6 1

Correct the duplicated gene point

Child 1: 1 5 2 5 8 8 (TODO: correct the unchanged duplicate digit, purple 8)

In the duplicate fragment, red 8 in child 1 corresponds to 6 in child 2. Therefore, we replace purple 8 with 6 in child 1.

🡪 1 5 2 5 8 6 (TODO: correct purple 5)

In the duplicate fragment, red 5 in child 1 corresponds to 2 in child 2. Therefore, we replace purple 5 with 2. However, digit 2 is still duplicated in child 1.

🡪 1 5 2 2 8 6 (TODO: correct purple 2)

In the duplicate fragment, red 2 in child 1 corresponds to 3 in child 2. Therefore, we replace purple 2 with 3.

🡪 1 5 2 3 8 6

Using the same theory, the genetic code of child 2 is shown below.

Child 2: 6 2 3 3 6 1 🡪 8 2 3 5 6 1

*Mutation*

A gene code mutates by the means of swapping two digits in the same gene unit.

Parent: 1 2 3 5 6 8 (TODO: swap position 2, 5)

Child: 1 6 3 5 2 8

*General Mechanism*

The project firstly will create two independent colonies whose population initially are 10,000 for each, by shuffling the digits in gene units in the gene codes. The colonies isolate to each other and will evolve separately, which handled by two threads.

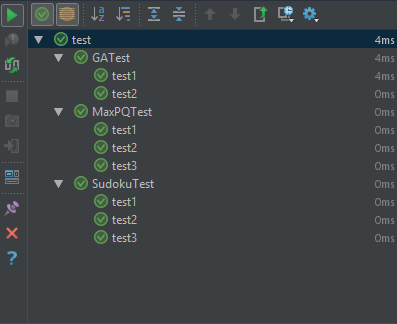
Each colony will experience crossover, mutation and selection in each generation. Assuming n is the initial population of the colony in a generation, it will have *n × ln(n)* crossovers and *n × mutationRate* mutations. The selection will select top *n × fitnessRate* (where *fitnessRate* is between *fitnessRateUpperBound=*0.08 and *fitnessRateLowerBound=*0.03) fitness sudokus. In case the population goes too large or too small, the selection adjusts the population between *populationUpperBound*=6000 and *populationLowerBound*=4000.

The evolution stops after 100 generations.

**Results**

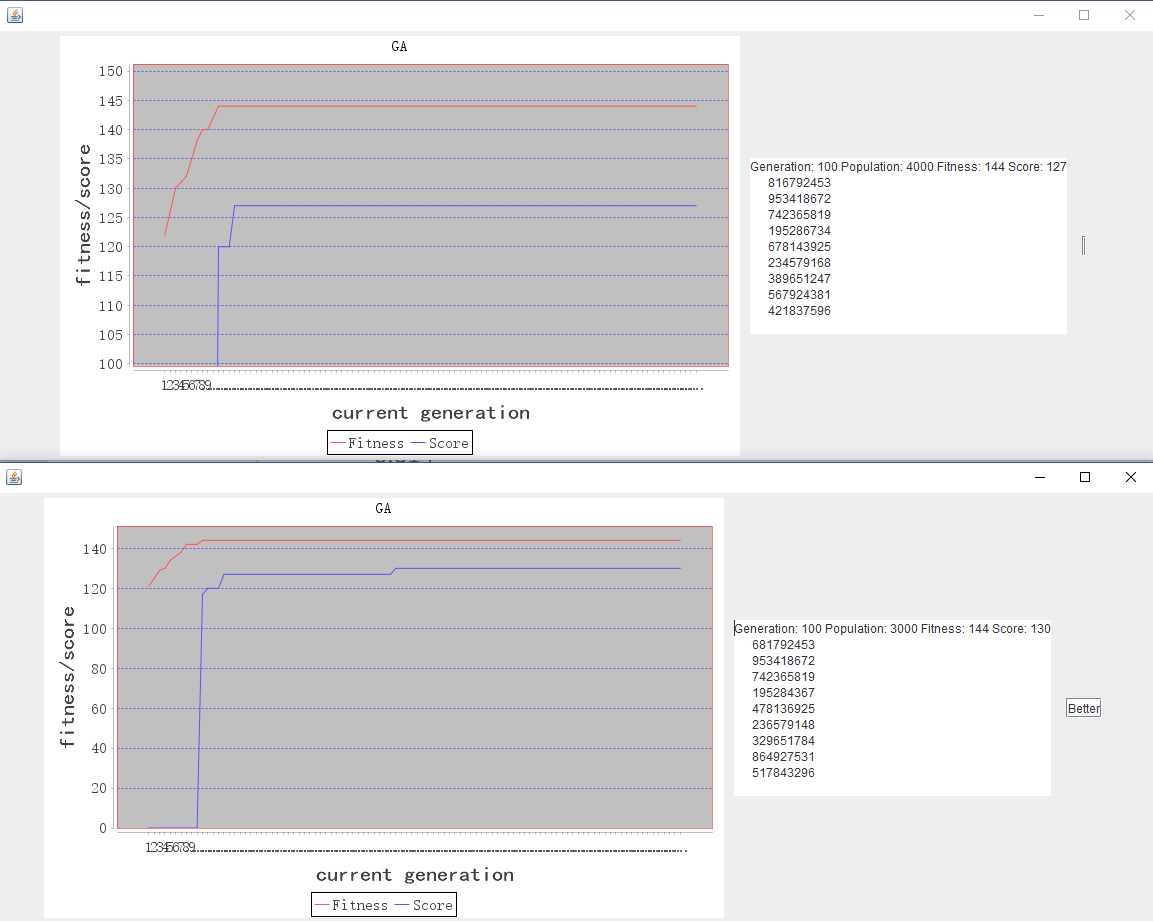
*Unit Tests*

The project passes all unit tests using the sample sudoku above.

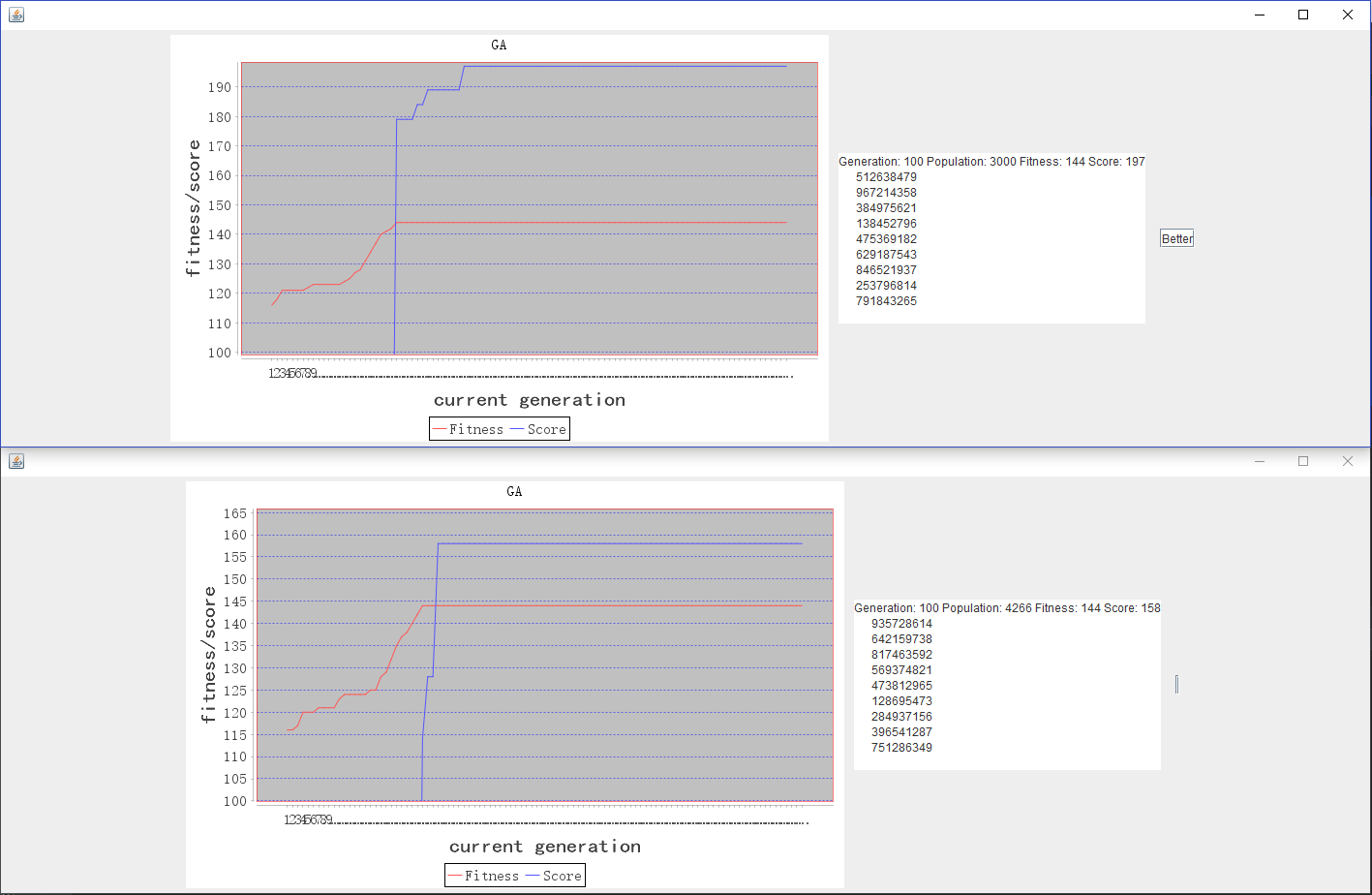


*Running Results*

Using Sample:



Using blank sudoku



Other puzzle:

