Genetic Algorithm: Pseudocode

The following document details a pseudocode version of the genetic algorithm. The design was inspired by a summary article on genetic algorithms by Beasley, Bull, and Martin (Beasley, Bull, and Martin, 1993). The algorithm solves a 4x4 Sudoku puzzle in which it populates a four-letter word into the grid to form a solution. The goal of this puzzle is to arrange the letters from a four-letter word into a 4x4 grid such that no letter repeats for every row, column and 2x2 sub grid (excluding sub-grids made from combining rows and columns 2 and 3 together).

Overview

Preparing the Dataset

Step One: Check Input

Input Constraints:

- There must be exactly four distinct characters provided in the partial solution.
- Each distinct character can appear more than once in the partial solution.
- The number of generations must be passed into the algorithm.

If an input passes these constraints, continue. Else stop and return error message.

Step Two: Binary Representation

Convert each unique letter into a binary representation.

For example: If the word is ‘word’

Letter	Binary
‘w’	(0,0)
‘o’	(0,1)
‘r’	(1,0)
‘d’	(1,1)

Step Three: Initial Population

Generate a set of new whole solutions with randomly assigned grid values.

Generation Constraints:

- The initial solution provided must remain intact, i.e., only empty gaps are filled.
- When filling in the solution, each letter must only appear in the solution exactly four times.

Authors: Giuseppe D’Elia and Cheylea Hopkinson

Generating Solutions

From this point forward, we now begin a loop until we have found a correct solution (a solution with a fitness score of zero) or have reached maximum generations as set in the input stage.

The loop is summarised below:

WHILE WE HAVE NOT REACHED MAXIMUM GENERATIONS AND NO SOLUTION HAS SCORE OF ZERO:

1. EVALUATE THE CURRENT FITNESS VALUE OF THE WHOLE POPULATION
2. SELECT THE BEST SOLUTIONS
3. APPLY THE CROSSOVER OPERATOR
4. CALL MUTATION OPERATOR
5. DELETE BETWEEN 5 AND 10 SOLUTIONS WITH THE LOWEST SCORE
6. ADDS 5 NEW RANDOM INDIVIDUALS TO THE POPULATION
7. REPEAT

Step One: Calculating Fitness

For each randomly generated solution, score the fitness and order the set by score ascending.

How to Score:

1. If any element is repeated in a row +1 to the row violation score for every match.
2. If any element is repeated in a column +1 to the column violation score for every match.
3. If any element is repeated in an eligible 2x2 sub-grid, +1 to the sub-grid violation score for every match.
4. If the row violation or column violation score are greater than three, proceed to 5., otherwise proceed to 6.
5. Calculate the total score as the highest value between the row and column score added to the sub-grid score. (This is to account for over counting).
6. Calculate the total score as the sum of all three scores.

Once the scoring is complete, a list of solutions and their scores is returned.

Step Two: Selection

Take a random sample from the top ten scoring solutions to use for creating a new population.

Step Three: Crossover

This function chooses a random number between 1 and 14. Loop between couples and combining them.

UNTIL THERE IS A COUPLE, REPEAT:

1. SPLIT first chromosome in the random index chosen in the previous step.
2. SPLIT the second chromosome in the random index chosen in the previous step.
3. JOIN the first part of the first chromosome with the end part of the second chromosome.
4. JOIN the first part of the second chromosome with the end part of the first chromosome.
5. ADD in the new individual to the population.

Step Four: Mutation

For each element in each selected solution:

- Randomly generate a number between 1 and 0.
- If this number is greater than 0.23, apply the mutation operator to change the value.

Constraints:

- Do not apply this step to elements that exist in the initial partial solution.
- Only generate a word that does not appear in that row and that column.

Step Five: Remove Low Performing Solutions

For every multiple of five cycles, remove between five and ten of the worst performing solutions.

Step Six: New Random Solutions

For every multiple of five cycles, generate five new random solutions

Step Seven: Repeat

If a solution with a score of zero has not yet been found, repeat the above steps with the new population generated from the above steps.

Exiting the algorithm

- If fails preparation stage, return appropriate error.
- If a correct solution is found, return all unique solutions in the population with a score of zero.
- If no solution has a score of zero, return that a solution could not be found.

Authors: Giuseppe D'Elia and Cheylea Hopkinson

Pseudocode

```
This code generates a solution for a partially solved 4x4 sudoku grid, where (i,
j) represents the location within the grid.

word = [w, o, r, d]

initial_solution = [[(0, 0), w], [(0, 1), o], [(0, 2), r], [(0, 3), d]]

binaryfunction(word):

   Convert each element of word into binary representation.

generateInitialPopulation(max_count, initial_solution):

   while count(population) < max_count

      Use the total letters that can be populated into the grid and remove what
      is already in the solution.

      Randomly add the letters to the blank spaces in the grid.

      Add solution to the population

      Return population

generations = x (where x is an integer)

iteration_no = 1 (start from 1)

while iteration_no < generations and score != 0

      geneticAlgorithm(initial_solution, population):

            scorePopulation(population):

                  For each solution in the population count the conflicts for
                  each row, column, and sub-grid.

                  Sort the score from best to worst.

                  Return population and scores.

            Selection(population):

                  Take the first ten values (best values) of the population and
                  remove the others.

            crossover(population):

                  For all values in the population

                  Take two solutions from the population and split each one at a
                  given index.
```

Authors: Giuseppe D'Elia and Cheylea Hopkinson

```
        Swap the different parts of the solutions creating two new
        solutions.

        Add this value to the new population

        return population

mutation(population):

        for each element in each solution in the population:

                if element is in initial_solution;

                        Do nothing, continue

                else:

                        Generate a number a between 1 and 0

                        if a > 0.23:

                                Change the element to another element such
                                that a word that does not appear in that
                                row and that column

                        else:

                                Do nothing, continue

eliminateLow(population):

        if iteration_no / 5 has no remainder:

                randomly remove between 5 and 10 solutions with the
                highest (worst) scores

        else:

                Do nothing, continue

addNewRandom(population)

        if iteration_no / 5 has no remainder:

                generateInitialPopulation(5, initial_solution)

                (Add five new randomly generated solutions to the
                population).

        else:

                continue
```