CSC412/512 – Intelligent Systems
HW3 – Due Wednesday, February 21, 11:59 PM

In this assignment you are to implement a genetic algorithm solution to create a full Sudoku board that satisfies the rules. (In Sudoku, each number 1-9 must appear exactly once in each row, each column and each 3x3 cell of the 9x9 board).
You can implement your code in Java, C++ or Python. You are not to use any canned search algorithms; you must implement the classes that make up the GA solver yourself. I have implemented the SudokuBuilder and SudokuBoard classes for you, but you will need to implement the rest of it yourself. My code is in Java but you can use it as a starting point to implement in another language if you wish.

Here are the classes that you need to write:
GASolution is a class that contains the "genome" – a sequence of numbers that specify a Sudoku board layout. It likely contains other data elements as well, as needed. The class has methods that support all of the things a single solution needs to do – mutation, breeding, fitness evaluation and so on.
GAPopulation is a class that contains a collection of GASolutions. The GAPopulation undergoes new generations, in which the best solutions are bred together to create children. The goal is to execute repeated generations until the best solution is a fully legal Sudoku boad – with a fitness function of PERFECTSCORE (243).

You will execute your solution by running the main() method in SudokuBuilder.

NOTE 1: I wrote all of this fairly quickly, but if you think you have found a problem in my code, CHECK WITH ME BEFORE ASSUMING IT'S WRONG AND CHANGING IT!
NOTE 2: I emphasize again, you are NOT to download someone else's code for solving this problem – whether source code or a package. You are to implement a valid genetic algorithm and experiment with them.
NOTE 3: GA solutions are great because they don't rely on partial derivatives or a priori knowledge, but they can take a long time to run. My solution often took many hours.
NOTE 4:

When you are done, put all of the code for your classes, along with the TEXT (no screenshots) from running your program a couple of times, into a single Word file and submit via Blackboard.

```
// SudokuBuilder Creed Jones CSC512 CBU December 27, 2017
//  this class uses a GA to create a fully valid sudoku board

import java.util.Random;

public class SudokuBuilder {
    private static final int NUMGENERATIONS = 100000000;
    private static final int PERFECTSCORE = 243;
    private static double mutateProb = 0.0001;
    private int popsize;
    private GAPopulation pop;
    private Random rand = new Random();
    private final Boolean doCrossover = false;

    public SudokuBuilder() {
        pop = new GAPopulation();
        pop.initialize();
    }

    public SudokuBuilder(int size) {
        pop = new GAPopulation(size);
        pop.initialize();
    }

    public int newGeneration() {
        if (doCrossover) {
            // breeding
            pop = pop.breed();
        }
        else {
            pop = pop.crossover();
        }
        // mutate
        pop.mutate(mutateProb);

        // sort and assign breeding probabilities
        return pop.sort();
    }

    private String getFitnessString() {
        return pop.getFitnessString();
    }

    private String getResultString() {
        return pop.toString();
    }

    public static void main(String[] args){
        SudokuBuilder builder = new SudokuBuilder();
        int maxfitness = 0,lastfitness = 0;
        Stopwatch swatch = new Stopwatch();
        swatch.start();
        try {
            // PrintWriter writer = new PrintWriter("learnHistory.csv", "UTF-8");
            for (int generation = 0; generation < NUMGENERATIONS; generation++) {
```

```
                lastfitness = maxfitness;
                maxfitness = builder.newGeneration();
                if (lastfitness != maxfitness) {
                    System.out.println("Generation: " + generation + ", Fitness = " +
maxfitness + "; mutateProb = " + mutateProb);
                }
                // writer.println(builder.getFitnessString());
                if (maxfitness == PERFECTSCORE) {
                    break;
                }
                if ((mutateProb < 0.01) && (generation % 1000) == 0) {
                    mutateProb *= 1.01;
                }
            }
            // writer.close();
        }
        catch (Exception e) {

        }
        swatch.stop();
        System.out.print(builder.getResultString());
        System.out.println("Process finished in " + swatch.getElapsedTimeSecs() + "
seconds.");
    }
}
```

```
// SudokuBoard Creed Jones CSC512 CBU December 27, 2017
//  this class represents a single Sudoku Board

import java.util.ArrayList;
import java.util.List;

public class SudokuBoard {
    private static final int SZ = 9;   // there are spots in the code that won't
                                       // work if SZ != 9
    private Integer cells[][];

    public SudokuBoard() {
        cells = new Integer[SZ][SZ];
    }       // initializes to all zeros

    public String toString() {
        String res = "";
        for (int yinc = 0; yinc < SZ; yinc++) {
            if ( (yinc == 3) || (yinc == 6) ) {
                res = res + "-----------------------\n\r";
            }
            for (int xinc = 0; xinc < SZ; xinc++) {
                if ( (xinc == 3) || (xinc == 6) ) {
                    res = res + " | ";
                }
                res = res + cells[yinc][xinc] + " ";
            }
            res += "\n\r";
        }
        return res;
    }

    public void fill(Integer[] data) {
        int inc = 0;
        for (int yinc = 0; yinc < SZ; yinc++) {
            for (int xinc = 0; xinc < SZ; xinc++) {
                cells[yinc][xinc] = data[inc++];
            }
        }
    }

    private int numPresent(List<Integer> arr, int size) {
        int result = 0;
        for (int incr = 1; incr <= size; incr++) {
            if (arr.contains(incr)) {
                result++;
            }
        }
        return result;
    }

    public int getFitness() {
        int fitness = 0, inc;

        List<Integer> checkArray = new ArrayList<Integer>(SZ);
```

```
    // first check the rows for each value from 1 to 9
    inc = 0;
    for (int xinc = 0; xinc < SZ; xinc++) {
        checkArray.clear();
        for (int yinc = 0; yinc < SZ; yinc++) {
            checkArray.add(cells[yinc][xinc]);
        }
        fitness += numPresent(checkArray, SZ);
    }

    // now check the columns for each value from 1 to 9
    inc = 0;
    for (int yinc = 0; yinc < SZ; yinc++) {
        checkArray.clear();
        for (int xinc = 0; xinc < SZ; xinc++) {
            checkArray.add(cells[yinc][xinc]);
        }
        fitness += numPresent(checkArray, SZ);
    }

    // finally check each 3x3 cell...
    for (int youter = 0; youter < 3; youter++) {
        for (int xouter = 0; xouter < 3; xouter++) {
            checkArray.clear();
            for (int yinner = 0; yinner < 3; yinner++) {
                for (int xinner = 0; xinner < 3; xinner++) {
                    checkArray.add(cells[youter*3 + yinner][xouter*3+xinner]);
                }
            }
            fitness += numPresent(checkArray, SZ);
        }
    }

    return fitness;
}

public static void main(String[] args) {    // to test the fitness function
    SudokuBoard sb = new SudokuBoard();
    Integer sampData[] = {  2, 9, 5, 7, 4, 3, 8, 6, 1,
                            4, 3, 1, 8, 6, 5, 9, 2, 7,
                            8, 7, 6, 1, 9, 2, 5, 4, 3,
                            3, 8, 7, 4, 5, 9, 2, 1, 6,
                            6, 1, 2, 3, 8, 7, 4, 9, 5,
                            5, 4, 9, 2, 1, 6, 7, 3, 8,
                            7, 6, 3, 5, 2, 4, 1, 8, 9,
                            9, 2, 8, 6, 7, 1, 3, 5, 4,
                            1, 5, 4, 9, 3, 8, 6, 7, 2};

    Integer sampData2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9,
                            1, 2, 3, 4, 5, 6, 7, 8, 9,
                            1, 2, 3, 4, 5, 6, 7, 8, 9,
                            1, 2, 3, 4, 5, 6, 7, 8, 9,
                            1, 2, 3, 4, 5, 6, 7, 8, 9,
                            1, 2, 3, 4, 5, 6, 7, 8, 9,
                            1, 2, 3, 4, 5, 6, 7, 8, 9,
```

```
                                1, 2, 3, 4, 5, 6, 7, 8, 9,
                                1, 2, 3, 4, 5, 6, 7, 8, 9};

        sb.fill(sampData);
        System.out.println("Fitness is: " + sb.getFitness());
        sb.fill(sampData2);
        System.out.println("Fitness is: " + sb.getFitness());
    }
}
```