CSC 4510
Advanced Topics in Artificial Intelligence (Evolutionary Computation)
Florida Southern College

Assignment 3: Sudoku
Due: Thursday, March 21, 2019

In this assignment, you will be implementing a genetic algorithm to solve Sudoku puzzles, similar to the following paper:

T. Mantere and J. Koljonen, "Solving, Rating and Generating Sudoku Puzzles with GA", in IEEE Congress on Evolutionary Computation (CEC 2007), September 25-28, 2007, pp. 1382-1389.

Starter code has been provided for you (sudoku.py). You must modify the code so that it runs without error and can correctly solve any Sudoku puzzle (hopefully). Here are some additional details about algorithm specifics:

Representation
An initial (partially-filled) Sudoku puzzle is provided as a string in a text file (see the provided file called puzzles.txt). For example,

802003510060091003701000894608004021000258060920310400000402780005089000200007100

is the genotype that encodes the following puzzle (phenotype):

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

Notice how '0' in the initial data file means the cell is blank. In other words, the goal of the genetic algorithm is to fill in all the 0s (blank cells). During initialization, you should ensure that each individual solution is a genotype of 81 integers, where each set of 9 is a permutation of the numbers 1-9. Also, make sure that the given (fixed) numbers in the puzzle stay fixed!

Fitness
Your fitness function is comprised of the sum of two parts:

1. The number of missing digits in each column.
2. The number of missing digits in each 3x3 grid.

Notice that the number of missing digits in each row is guaranteed to be zero because of how we designed the genotype. The best possible fitness is 0, which indicates that the puzzle has been solved.

Selection
You may use any selection mechanism we discussed in class to select pairs of parents. Survivors should be selected using the fitness-based model with elitism = 1. Also, as in the paper, you should increase the fitness of the best solution in the population by 1 for each generation that it remains the top solution.

Crossover
Perform uniform crossover between rows of pairs of parents. This is similar to the paper's implementation except we are swapping rows instead of 3x3 grids. Make sure to set p_c, the probability of crossover.

Mutation
Perform swap mutation within each row of a candidate solution with probability p_m. If a swap occurs, make sure that it is valid (i.e. it does not affect the fixed numbers!). This is slightly different from the implementation in the paper.

Other notes
You are free to vary the other relevant parameters in the genetic algorithm, including the number of candidate solutions in a population, the crossover and mutation rates, number of generations until termination, etc. Make sure to specify these parameters in your final solution so that it is clearly visible.