Jonathen Sundy

Sudoku Solver using Genetic Algorithms

Representation used:

Each chromosome in the population is represented as a Chromosome object. In the object, there is a $genes\ ArrayList < int[] >$ where each entry in the ArrayList is an integer array of size 9 that represents a single sub-grid. Hence, there are 9 of these entries that represent all the sub-grids going from right to left, and top to bottom of a given sudoku.

Initial Population Generation:

The population is represented as an $ArrayList < Chromosome >$ where the size is initialized with a population size parameter defined by the user. Hence, each chromosome is generated by randomizing each sub-grid by ensuring that each 3x3 sub-grid contains each integer from 1 to 9 and that the given numbers stay in their original positions.

Fitness Evaluation:

Sudoku solver must obey four different conditions:

1. Each row has to contain each integer from 1 to 9,
2. each column has to contain each integer from 1 to 9,
3. each 3×3 sub-grid must contain each integer from 1 to 9,
4. the given numbers must stay in the original positions.

Conditions 3) and 4) are intrinsically satisfied.

Hence, since there are 9 rows and 9 columns, we have 18 equality constraints that the solution must fulfil.

The fitness function is derived from the set theory. It requires that each vector, row, x_5, and column, x_6, is considered as a set that must be equal to the set A, which contains integers from 1 to 9:

$$A = \{1,2,3,4,5,6,7,8,9\}$$
$$g_{i3}(x) = |A - x_i|$$
$$g_{j3}(x) = |A - x_j|$$

The function above calculates the number of missing digits in each row (x_5) set and column (x_6) set, where $|\cdot|$ denotes for the cardinality of a set.

- In the optimal situation, all digits appear in every row and column sets, and fitness function value becomes zero.
- Else, if some digit is missing from one row or column a penalty value 1 is added, if two digits are missing, penalty value two is added, and so on.

After point c_3, an extra penalty of two is added, if a same digit as some given appears in the same vector, that is added at is two.

Hence, our fitness function is one that is used to minimize the fitness of each chromosome in the population. When the fitness value of the chromosome in position 0 in the ordered list is 0, the loop terminates and displays the solution.

Selection:

A mating pool is used to create the possible parents for the next generation. The mating pool size is defined by the user. Hence, for each chromosome to be added to the mating pool, a binary tournament is held where two chromosomes are chosen in the population at random. The chromosome with the lowest fitness is added to the mating pool. Hence, this adds stronger chromosomes to be chosen for the next generation.

When generating a new generation, the chromosomes in the population are ordered from best to least fitness such that chromosomes are chosen by starting from the end of the population. This is done such that the chromosomes closer to the beginning of the list have a higher chance of selecting a parent from the mating pool to carry over to the next generation. The number of offspring to generate by crossover and mutation are calculated using their respective probabilities specified by the user.

Hence, for all chromosomes that are not allowed to be generated by crossover are generated by randomly choosing a parent in the mating pool to be carried over to the next generation,

It could be the case that the mating pool is filled with the same strong parent, hence, multiple copies of the same parent could possibly be copied into the next generation. This possibly converges the solution to a local optimum if the parent is a victim to the horizon effect.

Genetic Operators Used:

Uniform Crossover is only applied between sub-blocks. The crossover point cannot be between two sub-grids. The algorithm ensures that two parents that are chosen from the mating pool at random are two different parents. This constraint ensures that the offspring generated from crossover create a new diverse chromosome that has traits of either parent.

After such constraint is satisfied, each sub-grid in the newly defined child is created by cloning the corresponding sub-grid from one or the other parent. This is chosen according to a random generated binary crossover mask of the same length of the number of sub-grids in a chromosome (length 9). Where there is a 1 in the crossover mask, the sub-grid is copied from the first parent, and where there is a 0 in the mask the sub-grid is copied from the second parent. A new crossover mask is randomly generated for each pair of parents. Offspring, therefore, contain a mixture of genes from each parent. The number of effective crossing point is not fixed but will average L/2 (where L is the number of sub-grids in a given chromosome).

Swap Mutation is applied only inside sub-blocks. A sequence of 1 to 5 (length drawn randomly) inside a sub-grid was applied. The values of two positions are exchanged. A constraint is applied such that each position is not the same and that either of them are not a given position in a given sub-grid.

Once these positions are determined, another heuristic is applied to ensure that the mutation can occur. This heuristic is such that every time a mutation is applied it affects one or two columns, and one or two rows, at a total 3 or 4 row and column vectors. In an optimal swap situation, the two digits that are swapped should appear in these vectors zero times before the swap (or 2 times in the case of 3 vectors). When the puzzle is solved, any randomly selected two digits should appear in any randomly selected two rows and two columns total of 4 times. Hence, we give the system some 'slack' otherwise the condition is too tight such that when a swap mutation is attempted, it is allowed only if the digits to be swapped appear in the destination rows and columns 5 times.