COS 314 Assignment 2 Report By: Jared Gratz 16054972

Representation:

The representation used within the program is a 2-dimentional list of lists. The 2D list of lists is basically a list of rows from the puzzle. This seemed easier to code as rows could be read as lines straight from the file containing the unfinished puzzle. Compared to the papers provided with the assignment, there was no need to have additional computations to convert the 3x3 sub-grids into their own “rows”. Rows still represented genes from how the papers had done so, while the 2D list is still representative of a chromosome.

This change from the papers from 3x3 sub-grid to rows for the genes merely means condition 3) needs to swap with condition 1), *from Timo Mantere & Janne Koljonen’s Solving and Analyzing Sudokus with Cultural Algorithms paper at 2008 IEEE Congress on Evolutionary Computation (CEC 2008), page 4054.*

Initial population generation:

The initial population is generated by firstly duplicating the given unfinished puzzle by the constant variable in the program that specifies the population size. The function will then fill all the empty spaces, 0’s, with a shuffled set of values from 1 to 9. The function will go row by row, in each individual in the population, with a new set generated for each row. An additional check is done at each row such that a given value in the unfinished puzzle is not overwritten. These randomly shuffled sets create different solutions to the given puzzle, whether they are correct or not.

The generated set along with the additional check help ensure that conditions 1) and 4) remain fulfilled. Leaving conditions 2) and 3) to be optimized, *from Timo Mantere & Janne Koljonen’s Solving and Analyzing Sudokus with Cultural Algorithms paper at 2008 IEEE Congress on Evolutionary Computation (CEC 2008), page 4054.*

Fitness evaluation:

The fitness score for each individual in the population is initially set to zero at the start. The function will then increment the score by one for every missing digit in each column and 3x3 sub-grid. This means an individual that has a lower score has a stronger solution. The aim is to minimize the fitness score.

This scoring function should be optimizing conditions 2) and 3). Unfortunately, the “aging” penalty presented in the paper is not implemented and may have affected results.

Selection method:

A tournament selection is implemented because a tournament can help maintain diversity among the population, while also providing an easy way to adjust for selection scrutiny. Selection scrutiny can be adjusted by changing the tournament match size, additional details in next section. As you may know a parent is selected from the strongest individual in a match. While a match has randomly selected individuals from the population. The number of individuals selected is based on the tournament match size.

A small caveat to the tournament selection method described above is that an individual could be selected to become a parent more than once. However, the crossover operator described in the next section should minimize the possible lack of diversity that may arise as a result.

Genetic operators and parameters:

Other than the selection operator described above, there is the Crossover operator that is used to create the offspring. The crossover operator is implemented similarly to the uniform crossover method. The crossover operator will create two offspring from two parents. The offspring are created by swapping the genes of the parents. Genes are represented by the puzzle rows to maintain conditions 1) and 4). However, there is only a chance of a swap occurring between parents’ genes. This method should maintain diversity within the population.

Finally, there is the Mutation operator which is applied to the offspring. The Mutation operator will cause swaps to occur within the genes of the offspring. A chance is given to each position in the gene to swap its digit with another random position’s digit in the same gene. An additional check is in place to check that a given digit, from the given unfinished puzzle, is not swapped. Thus, maintaining conditions 1) and 4).

Four constant variables are set at the beginning of the sudoku.py file, just under the import statements. These are the main parameters of the algorithm, with the addition of the number of generations given in the command line.

The first parameter is POPULATION\_SIZE. It specifies the size of the population and should always be set to an even number such that when divided by two the quotient is also divisible by 2. This ensures there are no lone parents and will always produce two offspring.

The second parameter is TOURNAMENT\_MATCH\_SIZE. It specifies how big the tournament size will be in the selection operator. A lower value will result in more diversity among the parents with less selection scrutiny. A higher value will increase selection scrutiny and elitism among the parents, but also increasing the chance of duplicate parents. The value must not be less than one, because a parent needs to be chosen. It is unnecessary to make the value larger than POPULATION\_SIZE, as it would have the same result as if the two values were equal.

The third parameter is CROSSOVER\_CHANCE. It specifies the chance that the genes in the two parents will swap. A value of five should provide the most diversity. A value more or less than five will result in offspring with less genes from the other parent.

The fourth and final parameter is MUTATION\_CHANCE. It specifies the chance of each position in the gene to swap its digit with another random position’s gene. An additional check prevents given digit positions from swapping. A value of zero is guarantees a swap will occur. While a value of ten will prevent any mutation.

The parameters have been set to the following: POPULATION\_SIZE = 100, TOURNAMENT\_MATCH\_SIZE = 10, CROSSOVER\_CHANCE = 2 and MUTATION\_CHANCE = 9. For a generation value of 2000 passed in the command line, these values produced generally the better solutions.