EARIN Lab 3 Variant 4 Report

Dawid Wypych

Aayush Gupta

Implementation of Solution

In this lab, a generational genetic algorithm was implemented and used to solve the Booth function. First, the population (whose size was specified by the user) is generated using random distribution within a search space (also specified by the user). The key feature of a generational genetic algorithm is that after each generation, every individual is replaced by the current generation's offspring, which are generally biased towards more fit individuals. In this case, the most fit individuals are those whose x and y values best minimize the Booth function. The population size remains the same between generations.

With the exception of the last offspring when the population size is odd, every set of parents produces two offspring. Each parent is chosen by hosting a tournament of tournament_size randomly selected individuals. The individual that is the most fit out of those in the tournament becomes one of the parents. Two parents are needed to create offspring, and they must not be the same individual (although they can have the same values if two individuals are identical). While the same individual cannot be both parents, they can be a parent to more than two offspring.

Once two parents are found, they produce offspring. When offspring are produced, there is a chance that they will be copies of their parents (equaling 1 – crossover_rate). Otherwise, the resulting offspring's values are averaged between the two parents. In order to not produce twins every time (which would reduce genetic diversity), the offspring get a random amount of the x and y values from each parent (a weighted average is used), with the sum of each parent's contributions to each offspring equaling 50 percent.

After offspring are produced, there is a chance (equaling mutation_rate) that one or both of their values will mutate. In this case a mutation is represented by having the x and/or y value being redistributed randomly within the search space originally specified. Another parameter, the mutation strength, affects how much the mutation is attenuated or multiplied. If the mutation strength is between 0 and 1, the mutated offspring's value becomes a weighted average of its original value and the new random value. If the mutation strength is greater than 1, the offspring's original value contributes nothing and the random value is multiplied.

The algorithm is run until some number of generations has passed (specified by the user). The values of the most fit individual after the last generation are the solution to the function.

When choosing the parameters, the user can choose more than one value for each parameter. The algorithm is then run for every possible combination of parameters inputted.

Due to some inconsistencies in the results when the algorithm was run repeatedly using the same set of parameters, the user has an option to run the algorithm multiple times using

the same set of parameters. The results are then averaged among all trials for each set of parameters. This mostly prevents lucky and unlucky runs from skewing the results.

Results

The algorithm was run 50 times for each set of parameters tested, which took a total of about 15 hours. This ensured that the analysis performed would be reliable. The parameters tested for are as follows (all possible combinations of these parameters were tested):

Generations = 200, 400

Population size = 100, 200

Crossover rate = 0.5, 0.8, 0.95

Mutation rate = 0, 0.15, 0.4

Mutation strength = 0.1, 0.3, 1

Tournament size = 2, 8, 30

Search space = -50 to 50

See the attached Excel sheet for the raw data.

Analysis of Results

The most obvious relationships were between the population size along with the number of generations and the accuracy of the solution along with optimization time. Increasing the population size and the number of generations generally produced better results at the cost of longer optimization time. Generally, increasing the population was more reliable than increasing the number of generations due to premature convergence, where at some point (in cases where the mutation rate or mutation strength were zero), the genetic diversity would become reduced to zero. This mainly occurred when the mutation rate was set to zero. Alternatively, if the mutation rate and mutation strength were too high, the best solution would get better with more generations until a certain point, after which it would oscillate. The relationship between the population size and the optimization time was found to be linear. The same was true with the number of generations.

The ideal crossover rate was generally high. Whereas a low (but not zero) mutation rate and mutation strength produced the most reliable results. However, whether a good solution was produced was found to be most affected by the tournament size. A small tournament size (2) was almost always found to produce a bad solution, with one notable exception. A medium tournament size (8) usually produced the best results. Whereas a large tournament size (30) never produced the best results, but generally performed much better than a small tournament size.

The one notable exception where a small tournament size produced the best solution were in cases where the crossover rate, mutation rate, and mutation strength were all low. While the solution at these parameters was not the best overall, it was still one best. Generally, the higher the crossover rate, mutation rate, and mutation strength, the better it is to have a higher tournament rate.

Increasing the tournament size also increased the computation time. However, the relationship was not found to be linear. For example, increasing the tournament size from 2 to 30 only tripled the optimization time.

One reason for why a small tournament size is best when crossover rate, mutation rate, and mutation strength are all low might be that the algorithm functions best when there is some degree of randomness to keep the genetic diversity at a reasonable level. A small tournament size is much more likely to select parents that aren't the most fit and, thus, is more chaotic. On the other hand, if randomness is too high, it becomes much more difficult to get a consistent solution.