AI Second Assignement

Davide Casnici

November 2022

1 Code Structure

1.1 Initialise population

Calling this function, a gray encoded new population is returned. I first sample from a uniform distribution values which lie in the current DeJong Function instance's range, then I gray encode them.

1.2 Compute fitness

To compute the fitness of the population, I first evaluate the latter with the respective method adding an infinitesimal constant. Furthermore I invert the sign of function (so the minimum become the maximum) and add to it 1.1 times the maximum of the non inverted function (so the current minimum). I in this way I have a function which maximum correspond to the minimum of the DeJong Function (this will be useful to compute the roulette probabilities). At last I divide the values by the new maximum so I have all the values bounded in the range (0,1] (which adds more numerical stability).

1.3 Rank selection

I have implemented the formula of the rank selection:

$$P_i = \frac{1}{N}(P_{worst} + (P_{best} - P_{worst})(\frac{I-1}{N-1}))$$

considering the correct values for P_{best} and P_{worst}, respectively the maximum fitness of the population and the minimum (I also normalize probabilities).

1.4 Roulette selection

Thank to the fact the maximum of the fitness corresponds to the minimum of the DeJong Function, i can simply compute the roulette selection probabilities

dividing each fitness by the sum of the fitnesses.

$$P_i = \frac{Fitness_i}{\sum_{j=1}^{n} Fitness_j}$$

1.5 Selection

In this function I check which selection rule must be applied, calling then the correspondent function. After retrieve the probabilities, I sample ten times two parents (without replacement for the partner).

1.6 Crossover

In the crossover function, I iterate through the pairs of parents. For every pair I iterate through each dimension of the parents, sampling a random number to perform the single point crossover. After have split the the dimensions in two part, I swap them between the parents of the pair in order to create the new children according to the slides.

1.7 Mutation

The mutation function iterates through each single bit of each single dimension of each child. At every time a number is sampled according to a random uniform distribution. If the latter is lower than the mutation rate (1%) then the bit is swapped.

1.8 Elitism selection

If this function is called, it merges the two populations (the old and the new one), returning only the twenty best individuals among them (according to their fitnesses).

1.9 Reconstruction

This function simply returns the new population if the elitism is set to false, otherwise it call the previous function and returns the best twenty individuals.

1.10 Update statistics

Since it was optional to use this function, I have preferred to update the statistic in the main loop.

1.11 Solve

The function iterate a loop for 600 iterations, calling sequentially functions as a pipeline and uploading the statistics (after creating and evaluating an initial population).

2 Results and considerations

2.1 Results

Looking at the plots (included in the Colab notebook) we can notice how in all the cases the genetic algorithm has converged to a good solution, sometimes even reaching the global optimum (we can notice this looking at the dataframe for the DeJong(1)). In some iterations the algorithm converges in some tens of iterations, leading to step-function-like plot, in other iterations the convergence is really slow, requiring hundreds of steps. The DeJong(4) instance is the one with the highest standard deviation among all of them and looking at its plot it makes sense due to its shape. We can also notice how for each experiment using the elitism produce a really low standard deviation and a better convergence of the genetic algorithm. Looking at the dataframes we can notice that we achieved a best value really close to the optimal value for almost all the trials, except for the ones of the DeJong(3) instance. (All my best fitnesses are equal to one because 1 because I rescale them by dividing by the maximum as explained in the [1.2], so of course the highest fitness achieved would be 1).

2.2 Final considerations

Looking at the averaged performances over the tree seeds, it is easy to notice how, on average, the rank selection is the worst method, especially if used without elitism, which improves the performance of the algorithm with both the selection rules. The best results are always obtained using the roulette selection and the elitism, this is due to the properties of elitism, which assure us that at least the best individual of the last generation is carried to the next generation (except in the case in the next generation all the individual have better values to the best of the old generation); so with elitism the best minimum value of the objective function should never decrease, and that is indeed the case we have. We can also notice how the roulette selection is better than the rank one, and this is due to the fact that the roulette is more proportional to the fitness of each individual, while the rank selection is only proportional to its rank, regardless of the fitness. The algorithms with the elitism have really few standard deviation because they take the best twenty individuals, so it is not so likely to have individuals with bad performance in the consecutive populations.