VPol_GA.py User guide

By Ryan Debolt

Sections:

Obtaining:

To obtain this code and any of the files you will need to run it, you will go to https://github.com/osu-particle-astrophysics/GenetisGeneticAlgorithms and then clone this repository onto your user space. From here copy VPol_Ga.py (as well as any other needed files) and move them into the directory that you are using it in. This ensures you will have a backup version in the repository if your modifications go wrong as well as ensuring that you will not push a bugged version of the code to the github.

Arguments:

This version of the genetic algorithm contains a total of 11 arguments that need to be passed in order to run a generation. Without passing these in or doing it improperly will result in the GA not running properly or at all. These arugments are as follows

1. **NPop:**
    a. Otherwise called population, this parameter controls the number of individuals that the GA will create a run with. It is suggested you run around 100 individuals in a generation or more if you are able as more individuals will tend to lead to better results
2. **Generation:**
    a. This argument simply reads in the current generation that you are in in this loop. This allows the GA to know where it is and behave accordingly. In particular, it knows how to behave when at the 0th or Nth generation.
3. **S_no:**
    a. This is the Survival number (Previously known as reproduction but renamed for clarity). This controls how many individuals from the previous generation will be allowed to survive unaltered into the next generation.*
4. **C_no:**
    a. This is the Crossover number. This argument controls how many individuals will be created by sexual reproduction. It is important to make sure this is an even number as well as every two parents will produce two offspring. If you attempt to give an odd number, the GA will not run.*
5. **I_no:**
    a. This is the Immigration number. This controls how many individuals will be created in the population by random generation. This simulates the immigration of individuals from other populations and serves to introduce genetic diversity.*
6. **M_rate:**
    a. This controls the mutation rate of individuals created by crossover. The number you pass in here will be divided by 100 to create the rate that is used.
7. **Sigma:**
    a. Controls the gaussian width of the mutations. This will also be divided by 100 when passed in.

8. **Elite_no:**
    a. Determines how many individuals per generation will be selected by Elite selection. Only do 1 individual at most, preferably 0. This is more here for assurance purposes than to be used for runs.**
9. **Roul_no:**
    a. Determines how many individuals per generation will be selected by Roulette selection.**
10. **Rank_no:**
    a. Determines how many individuals per generation will be selected by Rank selection.**
11. **Tour_no:**
    a. Determines how many individuals per generation will be selected by Tournament selection.**

* Arguments of S_no, C_no, and I_no, must add to the population size to work

**Arguments of selection methods must add to the population size to work

Functions:

Functions are the foundation of this genetic algorithm. To keep things simple to alter, I have written all repeated functions of the GA into easily read and managed functions. Bellow, I show the name of each function and the variables that are passed in. They are as follows:

1. **DataRead**(Fitness, I_Pop, Gen)
    a. Ran every generation except for generation 0, this function finds the generationDNA.csv and fitnessScore.csv files to read in the previous generation that our new generation will be built from.
2. **DataWrite**(NPop, F_Pop, freq_coeff, freqVector, S_no, C_no, selected, Gen, seed)
    a. Ran every generation. This function writes out the generationDNA.csv file that will be used to determine the fitness scores of individuals. It also writes out the parents.csv file that is used by us to track lineage as well as the random number seed used by that generation.
3. **Sort**(Fitness, I_Pop, P_loc)
    a. This function runs an insertion sort algorithm to put the individuals read in from the previous generation in order based on their fitness scores, this is used by Rank selection.
4. **Generate**(Sections, Parameters)
    a. This function creates a single random individual within the bounds specified by our geometry and then passes that individual to the function that called it.
5. **SizeCheck**(R, L, A, B)
    a. This function reads in the genes of one section of our design and then checks to make sure that is does not violate any constraints we have put on it. These include not exceeding the borehole size, not self-intersecting, and not exceeding the bounds of our gene limits
6. **Select**(Opp_no, Fitness, Elite_no, Roul_no, Rank_no, Tour_no, Pool)
    a. This function calculates how many individuals will need to be selected by each selection method, calls each selection method based on this result, and then returns a list of the locations of the selected individuals to the function that called it.
7. **Elite**()
    a. Returns the location of the highest ranked individual of a generation.
8. **Rank**(Fitness)

    a. This selection method selects an individual through weighted selection where the weight is determined by the rank of the individual. It then returns the location of this individual.
9. **Roulette**(Fitness)
    a. This selection method selects an individual through weighted selection where the weight is determined by the proportional fitness of the individual. It then returns the location of this individual.
10. **Tournament**(Pool, Fitness)
    a. This selection method first gathers a small subset of the population determined by the Pool variable and then selects the individual from that group with the highest fitness score. It then returns the location of this individual.
11. **Initialize**(F_Pop, Sections, Parameters)
    a. This function calls Generate NPop times to create a completely random generation that will serve as the initial generation for the loop.
12. **Survival**(S_no, I_Pop, F_Pop, Fitness, Elite_no, Roul_no, Rank_no, Tour_no, Pool)
    a. Takes selected individuals from the initial population and then places them into the final population, simulating survival.
13. **Crossover**(S_no, C_no, I_Pop, F_Pop, Fitness, Elite_no, Roul_no, Rank_no, Tour_no, Pool, M_rate, sigma)
    a. From a list of selected individuals, this function pairs up two parents whose genes are then swapped uniformly to create two children individuals that are then sent to mutation before being added to the final population.
14. **Mutation**(F_Pop, start, stop, M_rate, sigma)
    a. Takes in individuals from crossover and then mutates the genes based on the mutation rate and sigma passed in. No Mutation is guaranteed.
15. **Immigration**(S_no, C_no, I_no, F_Pop, Sections, Parameters)
    a. Immigration calls generate to create I_no random individuals that will then be added to the final population.

Main Code:

The main code calls the functions above to create the entire generation. It also provides print statements between functions to confirm jobs were completed properly. The order of this code is as follows

1. **Read in the passed arguments**
    a. If the passed-in arguments are incorrect or one or more are missing, the function will end itself and output an error statement.
2. **Establish variables and arrays**
    a. Some variables are established without the need to read them in from the call.
    b. The function will take some integer augments and convert them to the necessary format they are needed for in functions that use them.
    c. Finally, any arrays that are needed are initialized
3. **Check the generation**
    a. If the generation is the 0th, the code will call initialize to create the generation and then use DataWrite to write this information to files
    b. Otherwise, the code moves to step 4
4. **Read in data**
    a. Calls DataRead to get the needed information
5. **Call Survival**
6. **Call Crossover and Mutation**
7. **Call Immigration**
8. **Call Datawrite to write the new generation's information to files.**
9. **End**

Running:

To run this program, you must first ensure that the following files are in the same directory as the GA, or you have specified where to find them in the GA itself.

1. **fitnessScores.csv**
    a. This stores the fitness scores and uncertainties of those fitness scores in a file.
2. **generationDNA.csv**
    a. Stores the parameter values of each antenna in a file.
3. **parents.csv**
    a. Stores information about genetic operators, the seed used to create individuals, and the parents of each individual in a file.

Once you have these files in the specified directory run this command to run the GA

```
Python3 VPol_Ga.py NPop, Gen, S_no, C_no, I_no, M_rate, Sigma, Elite, Roul, Rank, Tour
```

Where each variable corresponds to the arguments specified earlier. An example of a run call might look something like this:

```
python3 VPol_GA.py 100 1 6 70 24 20 5 0 20 60 20
```

The last thing is to ensure that you have a fitness function that occurs between instances of the GA so that your fitness scores are updated between generations. Good Luck.