Genetic Algorithm
An Evolutionary Optimizer

Nisarg Dave, Data Scientist, Michigan Technological University

***Abstract*—This paper demonstrates the GA process for optimization of benchmark functions such as DeJong, Ackley, and Restringin. The paper elaborates the process for developing genetic algorithm to find minima for the given functions. The steps for the genetic algorithm are elaborated in detail. The paper includes complete demonstration of GA including encoding, initialization, crossover and mutation with selection criterias.**

***Index Terms*—Genetic algorithm, Evolutionary computation, Genetic programming, Evolutionary optimization.**

I. Introduction

THIS paper describes the overall process of genetic algorithm for the given problem. Building the optimizer for the given criterion functions using the GA process is elaborated. Furthermore the selection of parameter choices are demonstarted using the entire GA steps. The importance of elitism and parameter choice is discussed from various aspects.

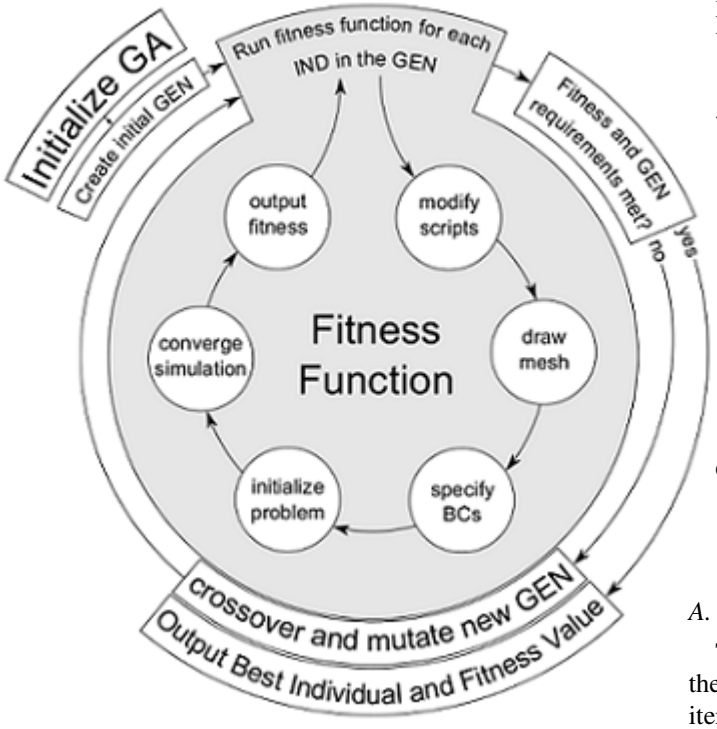


Fig. 1: Genetic Algorithm process in a Nutshell

Let's start with the defination, Optimizing the

$$f(\overrightarrow{x}), x \in R^d \ \overrightarrow{x} \in [-x/2, x/2]$$

I'm gonna develop the function named myGA,

$$function[fxbest, xbest] = myGA(func, xmin, xmax, tolerance, popsize, generationsNum, crossProb, mutateProb)$$

Where we need to take some function as a input with the specific range in which we want the minima. Other parameters we can set if we want to customize results using the optimal parameters. I made other parameters optional, if you want to set it to specific value then you can otherwise it will automatically consider the default value. The optional parameters are tolerance, popsize, generationsNum, crossProb and mutateProb.

II. Genetic Algorithm process

Data: Function with the range [xmin,xmax]
Result: Optimized fxbest and xbest value for given function using specified parameters
Initialization;
Randomly initialize the population using the range with tolerance parameter Decode and calculate the function values and transformed values;
while *generationNum is less than n* **do**
  Cycle crossover;
  random mutate;
  select best chromosomes;
  if *Elitism* **then**
    filter best n members and using elitism keep them;
    return elite;
  else
    repeat the process;
  end
end

Algorithm 1: myGA algorithm

A. Initialization

The initialization method i used was Random. I did select the points using gaussian normal distribution for starting iteration.

```
function [ initPop , chromosomeEachSizes ,
chromosomeSize ] = init( popSize , tolerance ,
xmin , xmax)
chromosomeEachSizes =
ceil( log2( (xmax-xmin)./tolerance ) );
```

```
chromosomeSize = sum(chromosomeEachSizes);
initPop = randi([0,1], [popSize,
chromosomeSize]);
end
```

B. Encoding

I made encoding simple for simplicity purpose. I used 2 bits 0 and 1 for encoding the randomly generated population from Gaussian normal distribution. I'm assigning the 0 and 1 bits for making clearer representation. Correction isn't needed after the crossover because encoding is pretty straightforward in this problem.

C. Decoding and Calculating fitness

Taking input of chromesomes and their range and then transforming them into the weighted average fitness for each iteration. After this I did calculation of fitness function using eval and transformed variables.

```
varNum = length(chromosomeEachSizes);
popSize = size(pop ,1);
transformed =
(xmax-xmin) ./ (2.^chromosomeEachSizes-1);
chromosomeEachSizes = [0 cumsum(
chromosomeEachSizes)];
for i = 1:varNum

    popVar{i} = pop(:,chromosomeEachSizes(i)
    +1:chromosomeEachSizes(i + 1) );

     var{i} = sum(ones(popSize ,1)*2.^(size
     (popVar{i},2)-1:-1 :0).*popVar{i} ,2) .*
     transformed(i) + xmin(i) ;
end
varsTransformed = [var{1,:}];
for i = 1 :popSize
     fvals(i) = eval([func ,
     '(varsTransformed(i,:))']);
end
end
```

D. Crossover and Mutation

Before making alterations, I used elitism for keeping the best fit chromosomes safe for the next iteration. The elite variable will contain the best fit members. using elitism is very useful to make optimization process fast and accurate. using elitism in process made by results more accurate.
$elite = popPrev(indexMax, :)$;
For the crossover I'm ideally using 0.6 probability and it just works fine. Although, i made it optiuonal parameter so people can try their custom crossProb, if they want to see the effect of changing the parameter. Moreover, the mutation probability i kept is very very low nearly (0.001). I tried using various mutation probabilities but with even 0.1 or 0.01 the results were not that optimum. GA took more generations for getting reasonable answer. By using very low mutation probability (0.001) its giving output faster (with less number of generations) and more accuracte.

```
function [ popNew ] =
crossover( popPrev, crossProb )

parisToCross = rand(pairsNum, 1)< crossProb;
pointsToCross = parisToCross.*randi(
[1,chromosomeEachSizes],[pairsNum, 1]);

for i=1:pairsNum
     popNew([2*i-1,2*i],:)=[popSorted([2*i-1,2*i],
     1:pointsToCross(i)),
     popSorted([2*i,2*i-1],pointsToCross(i)+
     1:chromosomeEachSizes)];
end
end

function [ popNew ] =
mutate( popPrev, mutateProb )

popNew = popPrev;
pointsToMutate = find(rand
(size(popPrev))< mutateProb);
popNew(pointsToMutate) =
1 - popPrev(pointsToMutate);
end
```

E. Selection Phase

Selection is the very important stage in GA. Selecting the previous best fit that we obtained from elitism so getting those from elite variable,
$[popNewSub, elite, fxbest] = selectchromo(pop, fvals)$;

Now after getting the sorted best fit chromosomes, I need to select the best fit from the current population so for that using selectchromo function and getting the new fxbest and new best fit population.

III. Parameter selection

The Important step of any GA process is parameter selection. I tried different things for selecting optimal parameters.

1) Tolerance: The tolerance is the threshold of robustness during the calculation in fitness. The tolerance should be less so your algorithm will be robust enough to survive in all situation. The function tolerance is important in representation step too. I'm taking the default values of 1e-4.

2) Population size: The population size is also one of the prominent factor affecting GA perfromance. I do prefer to take it as a positve even integer. In GA, at least population should be 100. I did randomly initialize GA with 100 population. I did set population number from 100 to 1000.

3) Generation Number: Generation number is most important because for me it decides the termination criteria of my GA. I do take 300 as a default value. I did test the GA for different generation number value such as 50,100,200,250,300,400. Morover, as generations increases the termination will be delayed and thus the answer will be more optimized and accurate. Higher number of generations usually gives better results but no free lunch, we need to find trade off so I kept it 300. At 300 generation, trigger will stop the GA process and presents the current fxbest and xbest as final solution.

4) Crossover Probability: The crossover probability is generally between 0.5 to 0.7. I chose 0.6 and the reason for that is the trial and run. I was getting more accurate and fast answer for crossProb = 0.6. I did run my algorithm several times with different crossover values.

5) Mutation probability: The selection of mutation probability is important. Initially I kept mutation probability as 0.1. It was not that high but for that value I was getting less accurate xbest. After setting mutation probability to 0.01, I got better xbest. so I did set the mutation probability 0.001 (even lower).

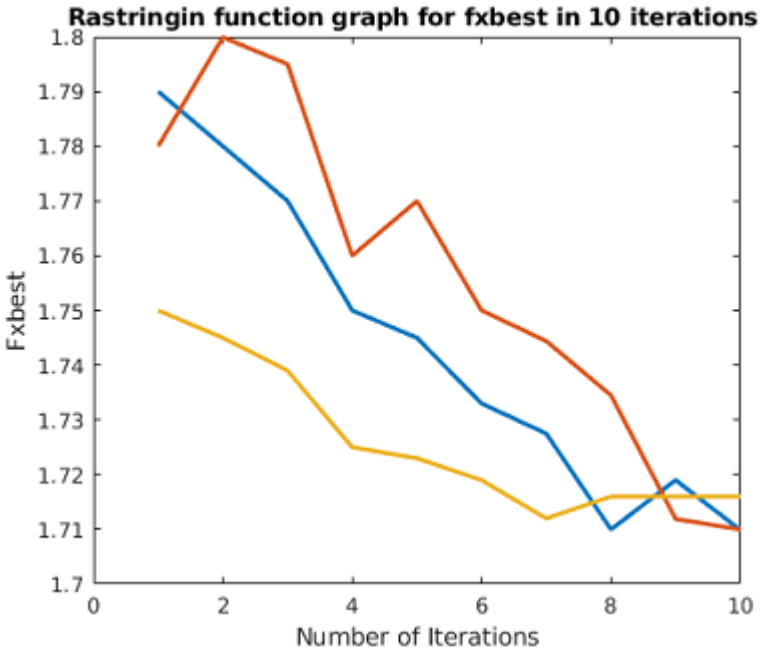
IV. Results on Benchmark functions

After building the Genetic algorithm. I did run the algorithm for the different benchmark functions such as DeJong, Ackley and Rastringin. I did note the fxbest and xbest values for the each run and performed such 10 runs for each function. I'm reporting the result of fxbest, xbest and time elapsed for each of them.

Results of De Jong's function			
Iteration	Fxbest	Xbest	TT(in sec)
1	0.000001	-0.001	43.87
2	0.000005	-0.002	45.72
3	0.000003	0.0019	44.58
4	0.000006	0.0025	46.83
5	0.000005	0.0024	43.90
6	0.000008	0.0009	44.72
7	0.000002	-0.00004	45.81
8	0.000002	0.0001	44.42
9	0.000005	0.0224	46.63
10	0.000003	0.0019	45.96

Results of Ackley's function			
Iteration	Fxbest	Xbest	TT(in sec)
1	1.7275	-0.44, -0.26, 0.89	67.39
2	1.7199	-0.12, -0.12, 0.038	65.91
3	1.7245	0.13, -0.11, -0.67	63.57
4	1.7302	0.17, 1.03, 0.82	63.43
5	1.7213	0.20, 0.06, 0.26	64.71
6	1.7217	0.24, 0.21, -0.20	66.80
7	1.7211	-0.17, -0.20, -0.16	68.13
8	1.7266	-0.23, 0.44, -0.78	65.10
9	1.721	0.18, -0.20, 0.13	67.84
10	1.7232	-0.40,-0.36,-0.003	67.94

Results of Rastringin's function			
Iteration	Fxbest	Xbest	TT(in sec)
1	0.0000027	-0.88, -0.90, -1.01	68.43
2	0.0000026	1.12, -0.11, -1.93	69.44
3	0.0000024	-0.023, -0.803, 1.91	70.22
4	0.0000025	0.065, 1.92, -1.94	70.92
5	0.0000023	-1.87, -0.89, -1.90	69.74
6	0.0000022	0.98, -2.92, -0.91	71.66
7	0.0000021	-0.82, 0.14, -1.02	68.05
8	0.000002	0.10, 0.89, 0.005	68.95
9	0.000002	-2.12, -2.168, 1.03	68.11
10	0.000002	-0.008, 1.87, -0.87	70.80



V. Conclusion

In conclusion, I can say that the GA can be good solution to any global optimization problem with some fitness function. GA can lead to drastically accurate solution to any given problem.