Course Name: F21BC Biologically Inspired Computation

Coursework 1 – Black-Box Optimization

Date: Friday 11 November 2016

Student`s Full Name: George Goniotakis

HWU Person ID: H00259066

Student`s Full Name: Murat Gunana

HWU Person ID: H00260187

GitHub URL: https://github.com/gg29/F21BC-Assignment-1

Contents

1. Part I - Multilayer Perceptron with Genetic Algorithm

1.1 Introduction

The purpose of this paper is to design a multilayer perceptron architecture alongside a genetic algorithm in order to find a potential solution for a certain problem within any problem space. We will use acronyms MLP for multilayer perceptron and GA for genetic algorithm throughout the paper. We particularly designed MLP to take design inputs from prompt and followed the same fashion for GA by integrating COCO blackbox optimizer. Moreover, we chose the MLP weights as genotype for the GA in order to improve the population based on fitness values that converge to a solution within the MLP. In further sections, we will go through both MLP and GA architectures and the hybrid concept how these two architectures interact with each other. Also, we will go through some findings and have a discussion on test results we carried out for the hybrid architecture code we have developed.

1.2 MLP Topology

We particularly chose a complete feedforward topology that there aren't any neurons ruled out and we restricted the MLP to a single output neuron. The MLP topology also takes hidden layers parameters to set up the architecture in accordance with input numbers of the MLP. We then formalized this concept to set up the architecture based on the parameters from the prompt. Here below is an example how it regulates the topology.

- x = number of inputs for the MLP.
- m = number of hidden layers.
- y = number of layers in the output layer which usually restricted to 1.

Then we can construct a genotype or chromosome length for our GA or in other words weights vector for the MLP as follow;

Genotype length = (m*x+y)*(x+1).

We also can formulize the desired output range as:

- i = the range of input values.

Desired output = 2 ^pow(2^pow(i)).

Let’s say we have some parameters with values x = 3; m = 2; y = 1. Then, the genotype or weights vector length will be (2*3+1)*(3+1) = 28. Which means the range of genotype will be 2^pow(28) in binary or possible alphabet {0,1} representations. Please note that Matlab only supports up to 2^pow(53). So it will give a runtime error when this amount exceeds.

In the code, the MLP takes the parameters above and calculates the output in the output layer and sets up a potential solution matrix that each element of MLP contains output for each input set. Please note that we used sigmoid function 1/(1+exp(-x)) as activation function to threshold each neuron output in the hidden layers and as well as for the output neuron. See the figure below for the graphical representation of the MLP topology in the example mentioned above. A common genotype consists of bias and weights such as in the following illustration. a) Bias are represented with b such as b(1,1) b) Weights are represented with W such as W(1,1).

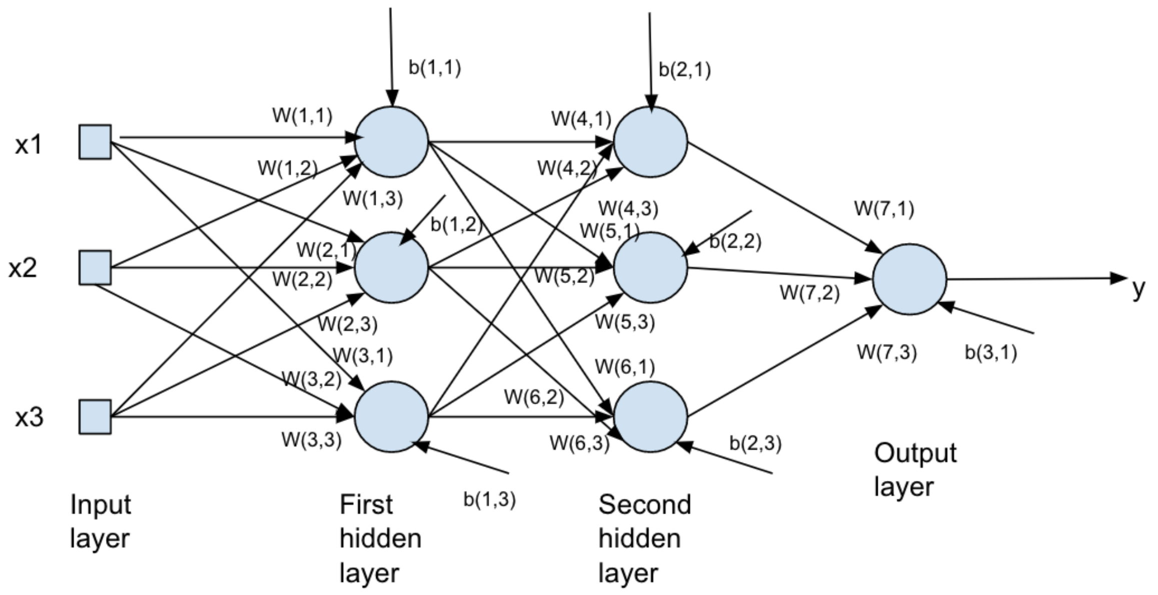

Image 1

1.3 GA Architecture

In our hybrid design, the GA carries out the tasks to evolve MLP by creating a population that each individual in the population represents a potential solution to the problem. The genotype or in other words the sequence of weights and bias in the above figure corresponds to an individual in the population. As we mentioned above the genotype alphabet consists of {0,1} and its range varies based on the input values and as well as the number of hidden layers. As in the example above the length of the genotype is 28 for instance. In the code, we used nonlinear function f(x) = x^2 to carry out some tests with different parameters.

In order to optimize the continuous function above, we created a set of different dimensions of populations with the help of COCO blackbox optimizer. We then iterated through a number of predefined generations to maximize the population based on the fitness function. The fitness function corresponds to fitness value of an individual in the population. In this case, we tried to improve the population to maximize the problem function. We picked up the best individual from the population and evolved the MLP network with all possible input values. As soon as the error rate or the number of generation has been reached a satisfactory level we then terminated the process. We followed the below steps to progress the population to find a potential solution.

1. Create an initial random population.
2. Evaluate fitness for each individual in the population.
3. Create intermediate population by using reproduction mechanism.
4. Generate new offsprings by applying genetic operators (crossover, mutation) to the intermediate population.
5. Select the best fitted individual from the population and if it is a solution to the problem then stop, otherwise go to the second step and carry out the operation.

By means of creating a random population, the GA starts with a potential solution of individuals in the problem space. In this particular problem, the fitness value of each individual is the square root of itself divided by sum of the square root of all the individuals in the population. It then applies roulette rule selection which makes a selection from the population based on the probability of each individual which is the fitness value calculated in the second step. It spins the fortune wheel exactly as many as population size. Most likely, the individuals with a low chance will be eliminated as a result of this. The GA then selects individuals randomly as pairs to mate with each other. It then performs the crossover operation by selecting a crossover point randomly which we restricted between 3rd element from the beginning to the 2nd element from the end of the genotype. It then performs the mutation by randomly making a selection and inserts the son and daughter offsprings into the population by replacing with their parents.

1.4 Experiments – Results

We checked the performance of our final algorithm by altering the setup of it and by recording our results. We did these tests in the final version of our algorithm that includes Mutation, Crossover and Roulette Selection. These results can be found in the table below.

Experiment Title	Population Size	Dimensions	Generations	Average Number of Mutations	Average Ftarget – Fbest [2D]	Average Ftarget – Fbest [3D]	Average Ftarget – Fbest [5D]	Average Ftarget – Fbest [10D]	Average Ftarget – Fbest [20D]	Average Ftarget – Fbest [40D]
Exp1	2	[2,3,5,10,20,40]	100	6	4.022975e+25	2.895437e+25	1.721687e+25	2.636629e+24	9.231185e+23	2.184152e+23
Exp2	10	[2,3,5,10,20,40]	100	7	5.669412e+24	1.270949e+24	2.234374e+24	2.639286e+24	4.244742e+23	8.287277e+22
Exp3	20	[2,3,5,10,20,40]	100	7	2.050450e+24	1.156344e+24	6.827671e+23	1.038014e+24	3.668660e+23	5.305779e+22
Exp4	50	[2,3,5,10,20,40]	100	5	4.653339e+23	1.872420e+24	5.413972e+23	3.362812e+23	2.354478e+23	9.702151e+22
Exp5	100	[2,3,5,10,20,40]	100	8	6.410629e+23	5.615320e+23	2.977137e+23	5.387886e+23	2.090487e+23	1.349379e+23
Exp6	500	[2,3,5,10,20,40]	100	4	5.669034e+22	2.561136e+22	3.306000e+22	2.419816e+22	2.607722e+22	2.166609e+22
Exp7	2	[2,3,5,10,20,40]	200	6	4.332920e+25	2.305366e+25	4.108967e+24	1.075882e+24	9.077888e+23	3.651918e+23
Exp8	10	[2,3,5,10,20,40]	200	8	2.684744e+24	2.116454e+24	1.173143e+24	4.223688e+23	4.146437e+23	7.763569e+22
Exp9	20	[2,3,5,10,20,40]	200	4	1.150583e+24	1.041222e+24	8.261620e+23	2.683400e+23	1.306547e+23	3.957582e+22
Exp10	50	[2,3,5,10,20,40]	200	7	2.377714e+23	2.178817e+23	1.900783e+23	2.443349e+23	1.786647e+23	9.840499e+22
Exp11	100	[2,3,5,10,20,40]	200	10	1.921117e+23	2.092345e+23	2.460417e+23	5.093545e+22	1.959517e+23	9.830723e+22
Exp12	500	[2,3,5,10,20,40]	200	7	3.351860e+22	1.111211e+23	1.648076e+22	5.454047e+22	5.774682e+22	8.938907e+21

Table 1

The plots of our results are located in the figures folder in GitHub.

1.5 Discussions and Conclusion

In this particular problem which is improving the nonlinear function $f(x)=x^2$ imposes some caveats to overcome. For every potential solution in the population, we tried to evolve the MLP by passing a desired output which in this problem will be the maximum value that any individual can reach and then MLP usually evolves and returns a satisfactory solution for any given individual. For instance, consider an MLP topology with two inputs, one hidden layer which contains two neurons and an output neuron in the output layer. The genotype limit or weight vector length will be 2^pow(9) = 511 max. Let's say we passed an individual with a value of 499 (111110011) and the desired output is 1111. In order to get the MLP to converge any solution. When we pass these parameters to the MLP it gives a satisfactory output based on sigmoid function P(x) = {x>0 is -1; x<=0 is -1}. However, it should not give a solution for these parameters provided.

Let us give a counterexample from backpropagation algorithm to to clarify it. The backpropagation algorithm calculates the weights and bias with the matrix [-9.5226 -5.7882 -7.0287;-2.8552 -7.0431 -6.9364;-4.4007 -9.3660 9.8156] for the same input, desired output, and MLP topology. As we can see, there are some negative values in the vector. Assuming we are improving our GA population and it is hard to evolve the MLP with {0,1} which are +1, -1 corresponds to weight vector values above. We can use different alphabet for the genes in our genotype for instance integer values instead. However, in this case, what is the fitness function going to be to evaluate each individual in the population in order to converge such a weight vector above for instance?

However, when the genotype length is increased, it seems the MLP does not converge to any potential individual imposed to it. This arises the discussion that it depends on the problem to apply hybrid concept we experimented in this particular problem. Perhaps, it will be best to add more genetic operators to GA that minimizes or prevents to fall into local minima or plateau. As we experienced in this problem, the weights consist of -1, +1 causes the algorithm to stuck in local minima as we need to change this values to a certain range such as -4.6677 or +3.5677 with the help of more genetic operator (Montana and Davis, 2004).

2. Part II - Genetic Algorithm

2.1 Introduction

Our goal is to create a GA (Genetic Algorithm) which is going to be used for solving complex problems. The GA is going to help us find a set of good individuals amongst many pairs of individuals in our population. Evaluating the pairs by calculating their fitness and by using other techniques like crossover, mutation and different types of selection we will improve the performance of our GA. By changing the parameters of our GA (population size, the number of generations, etc.) we are going to test it in different scenarios, and this will help us understand better how we can improve it thoroughly.

2.2.1 First Version of our GA (Setup + Results)

In our first version of our GA, we will create a basic Evolutionary Algorithm (EA). Our GA is going to be based upon this so we will implement it without COCO and we will do many tests. In this way, we will be sure that our algorithm is working properly before we move it to COCO.

Starting of, we will create a random individuals generator. After twenty tries, this generator is going to pick random numbers that are going to be used as our individuals. In the beginning, we chose to work with two dimensions. This means that every member of our population will be a tuple of two values i.e. x1,x2. After that, we did an implementation of the hill-climbing algorithm in our EA. We also added a hill-climbing rating variable that is basically the distance that is going to be changed in each iteration. This will help us explore the search space with different speed if it is needed.

Thereinafter, we chose the sphere function as our fitness function which we are going to minimize. The global minimum of this function is 0, so this means that our individuals should be equal to zero in order to achieve the best fitness.

Also, we created a mutation method that we set to have a very low chance to happen (0.001). The mutation happens to a random bit for all instances. We did that because mutation can make things better i.e. if we are stuck in a plateau while we are exploring the search space, but it can also get us away from a good point if it happens at an inappropriate time.

After checking that everything is working like it was supposed to, we started altering the configuration of our EA (population size and number of generations), and we collected the results of all these different tests.

Below you can see a table with these different setups and results:

Number of Binary Digits	Population Size	Dimensions	Generations	Mutations	Result after X generations	Average Ftarget - Fbest
5	1	[2]	100	0	19	0
5	3	[2]	100	0	25	0
5	5	[2]	100	0	21	0
5	10	[2]	100	0	9	0
5	20	[2]	100	0	7	0
5	50	[2]	100	0	3	0
5	100	[2]	100	0	2	0
10	1	[2]	100	0	-	5.182570e+05
10	3	[2]	100	1	-	5.373700e+04
10	5	[2]	100	0	-	1.428850e+05
10	10	[2]	100	0	47	0
10	20	[2]	100	1	49	0
10	50	[2]	100	1	85	0
10	100	[2]	100	0	57	0

Table 2

As we can observe, our algorithm succeeded many times in finding the best fitness for the different sets of individuals and especially in a short number of generations.

2.2.2 Second Version of our GA (Setup + Results)

In our second version, our aim was to move our algorithm inside COCO. We kept the number of dimensions to two. We added a method that computes the average error of all the instances in a generation so that we know if the algorithm`s overall performance is improving. Using the same tactic, altering the number of individuals and generations, we did some more tests using the COCO Framework.

The results of these tests can be found in the table below.

Experiment Title	Population Size	Dimensions	Generations	Average Number of Mutations	Average Ftarget – Fbest [2D]
Exp1	2	[2]	100	1	8.423062e+00
Exp2	4	[2]	100	1	8.600449e+00
Exp3	6	[2]	100	1	9.536022e+00
Exp4	10	[2]	100	1	2.317314e+01
Exp5	20	[2]	100	0	2.595138e+01

Exp6	50	[2]	100	0	1.056632e+01
Exp7	100	[2]	100	0	1.898573e+01
Exp8	500	[2]	100	1	8.689195e+00
Exp9	1000	[2]	100	1	8.395115e+00
Exp10	2	[2]	200	1	9.867649e+00
Exp11	4	[2]	200	1	8.423062e+00
Exp12	6	[2]	200	1	9.460182e+00
Exp13	10	[2]	200	0	8.441622e+00
Exp14	20	[2]	200	0	8.812395e+00
Exp15	50	[2]	200	0	9.781249e+00
Exp16	100	[2]	200	0	1.337709e+01
Exp17	500	[2]	200	0	9.378795e+00
Exp18	1000	[2]	200	0	8.353195e+00
Exp19	2	[2]	500	2	9.290582e+00
Exp20	4	[2]	500	1	8.535595e+00
Exp21	6	[2]	500	2	8.423062e+00
Exp22	10	[2]	500	1	8.423062e+00
Exp23	20	[2]	500	0	8.382209e+00
Exp24	50	[2]	500	0	8.440022e+00
Exp25	100	[2]	500	0	8.494422e+00
Exp26	500	[2]	500	0	9.047702e+00
Exp27	1000	[2]	500	0	8.427435e+00

Table 3

In this experiment, we can see that our average error is mainly not too good. The algorithm did a good job only in one or two test cases. Below we can see, the plot of our best result (Exp6).

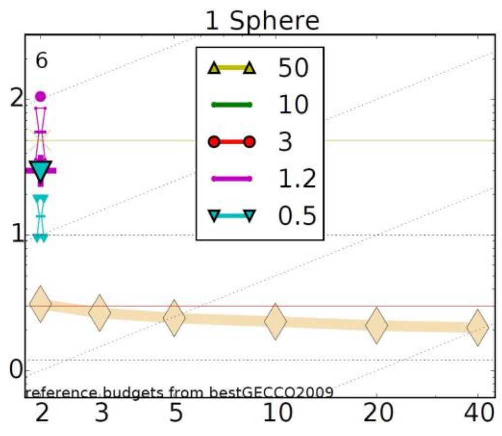

Image 2

2.2.3 Third Version of our GA (Setup + Results)

In our third version, we changed our algorithm so that it supports multiple dimensions. After this, we changed our dimensions from two to [2,3,5,10,20,40] in our exampleexperiment.m file. Following the same plan, we performed experiments again but this time we recorded the overall error of each different dimension.

The results of these experiments can be found below.

Experiment Title	Population Size	Dimensions	Generations	Average Number of Mutations	Average Ftarget – Fbest [2D]	Average Ftarget – Fbest [3D]	Average Ftarget – Fbest [5D]	Average Ftarget – Fbest [10D]	Average Ftarget – Fbest [20D]	Average Ftarget – Fbest [40D]
Exp1	2	[2,3,5,10,20,40]	100	1	9.101249e+00	1.611314e+01	3.073660e+01	5.582965e+01	1.020043e+02	1.968714e+02
Exp2	4	[2,3,5,10,20,40]	100	0	9.146475e+00	1.472095e+01	3.124549e+01	6.153280e+01	1.110079e+02	2.228515e+02
Exp3	6	[2,3,5,10,20,40]	100	0	1.082488e+01	2.259306e+01	4.767867e+01	1.758240e+02	3.179630e+02	8.734442e+02
Exp4	10	[2,3,5,10,20,40]	100	0	9.715009e+00	5.271277e+01	1.743229e+02	4.910308e+02	1.206126e+03	2.609085e+03
Exp5	20	[2,3,5,10,20,40]	100	0	1.509794e+01	6.810065e+01	2.410414e+02	9.455399e+02	2.332687e+03	5.131220e+03
Exp6	50	[2,3,5,10,20,40]	100	0	1.747736e+01	6.260552e+01	2.565180e+02	1.121288e+03	2.985150e+03	6.667382e+03
Exp7	100	[2,3,5,10,20,40]	100	0	1.393869e+01	5.184897e+01	2.575175e+02	1.067436e+03	2.907076e+03	7.144487e+03
Exp8	500	[2,3,5,10,20,40]	100	0	8.896129e+00	2.700314e+01	1.428863e+02	7.836297e+02	2.658085e+03	7.115618e+03
Exp9	1000	[2,3,5,10,20,40]	100	0	8.492822e+00	1.925913e+01	1.056671e+02	6.937748e+02	2.530544e+03	6.613433e+03
Exp10	2	[2,3,5,10,20,40]	200	1	1.116429e+01	1.711371e+01	3.015477e+01	5.385571e+01	1.029916e+02	2.031634e+02
Exp11	4	[2,3,5,10,20,40]	200	1	8.604075e+00	1.365706e+01	2.819451e+01	5.066657e+01	1.015553e+02	1.983953e+02
Exp12	6	[2,3,5,10,20,40]	200	0	8.366955e+00	1.303368e+01	2.609749e+01	5.310266e+01	1.005906e+02	1.972080e+02
Exp13	10	[2,3,5,10,20,40]	200	0	8.533995e+00	1.566892e+01	3.235232e+01	8.586707e+01	2.257533e+02	5.015119e+02
Exp14	20	[2,3,5,10,20,40]	200	0	1.006669e+01	1.965750e+01	1.092655e+02	3.545393e+02	1.073323e+03	2.754184e+03
Exp15	50	[2,3,5,10,20,40]	200	0	8.767595e+00	5.120535e+01	1.898514e+02	7.156499e+02	2.271573e+03	5.361049e+03
Exp16	100	[2,3,5,10,20,40]	200	0	9.744022e+00	3.846404e+01	2.013877e+02	8.999116e+02	2.795391e+03	6.383036e+03
Exp17	500	[2,3,5,10,20,40]	200	0	9.093142e+00	2.939324e+01	1.225864e+02	7.838009e+02	2.643421e+03	7.312393e+03
Exp18	1000	[2,3,5,10,20,40]	200	0	8.491435e+00	1.829019e+01	9.819448e+01	6.548545e+02	2.375579e+03	6.689543e+03
Exp19	2	[2,3,5,10,20,40]	500	4	9.443435e+00	1.519280e+01	3.078980e+01	5.556290e+01	1.056067e+02	2.036917e+02
Exp20	4	[2,3,5,10,20,40]	500	2	9.604182e+00	1.344704e+01	2.569038e+01	5.570134e+01	1.008450e+02	1.985399e+02
Exp21	6	[2,3,5,10,20,40]	500	2	8.420289e+00	1.473942e+01	2.591456e+01	5.363234e+01	9.816273e+01	1.986380e+02

Exp22	10	[2,3,5,10,20,40]	500	1	8.382209e+00	1.308792e+01	2.478910e+01	4.920051e+01	9.589800e+01	1.966136e+02
Exp23	20	[2,3,5,10,20,40]	500	1	8.353195e+00	1.273388e+01	2.342577e+01	4.986087e+01	1.023696e+02	2.182778e+02
Exp24	50	[2,3,5,10,20,40]	500	0	8.423062e+00	1.440990e+01	5.810518e+01	2.829328e+02	1.019134e+03	2.508198e+03
Exp25	100	[2,3,5,10,20,40]	500	0	8.391382e+00	1.962176e+01	1.009654e+02	6.077202e+02	1.881900e+03	4.981140e+03
Exp26	500	[2,3,5,10,20,40]	500	0	8.594582e+00	2.828138e+01	1.428072e+02	7.161345e+02	2.631924e+03	7.114896e+03
Exp27	1000	[2,3,5,10,20,40]	500	0	8.411009e+00	2.012664e+01	1.071037e+02	6.564782e+02	2.383236e+03	6.582237e+03

Table 4

Using the table, we can see that we have many variations on the error rate of our algorithm. Our best error rates of each dimension are these on the green color. Their plots can be found below.

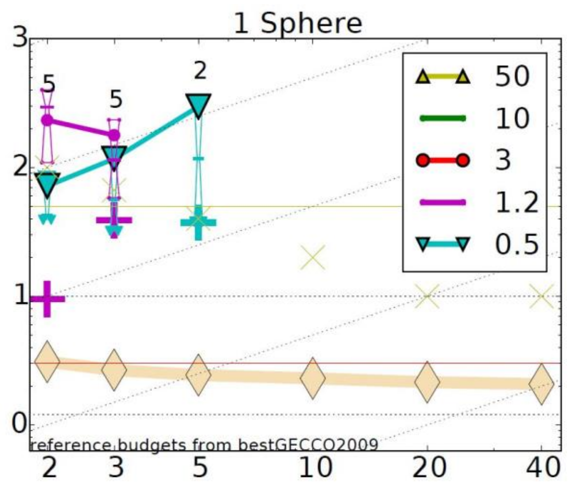

Image 3

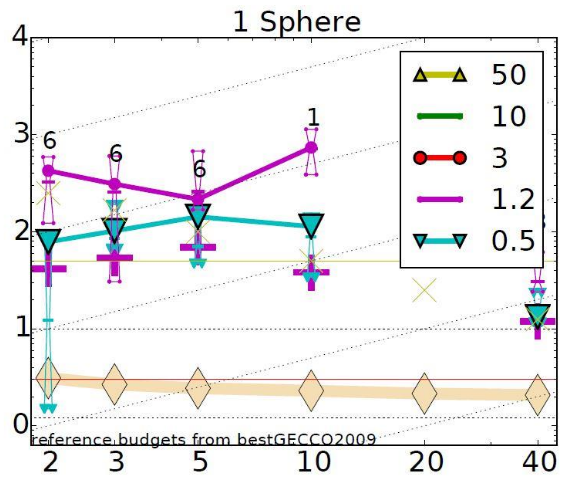

Image 4

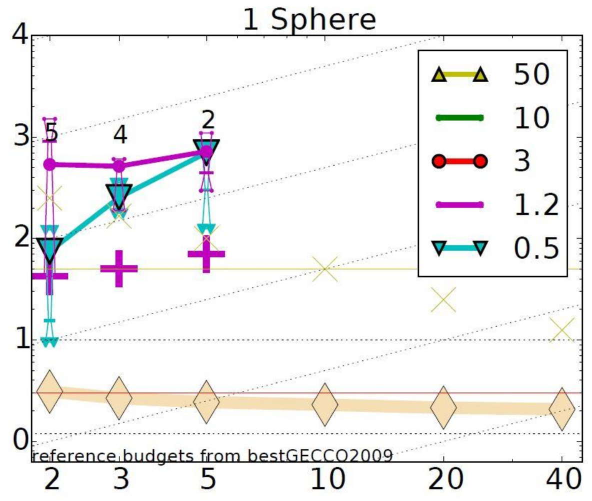

Image 5

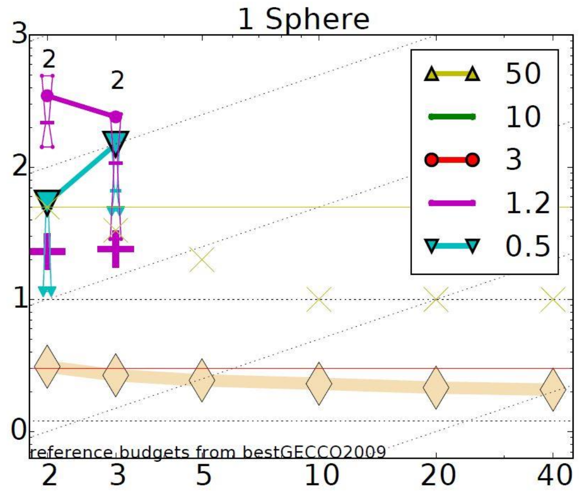

Image 6

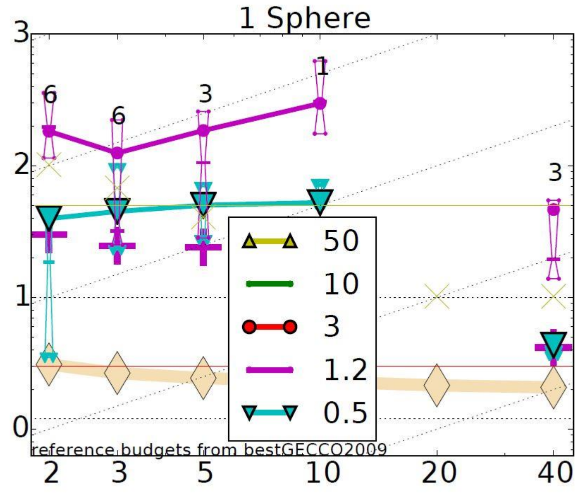

Image 7

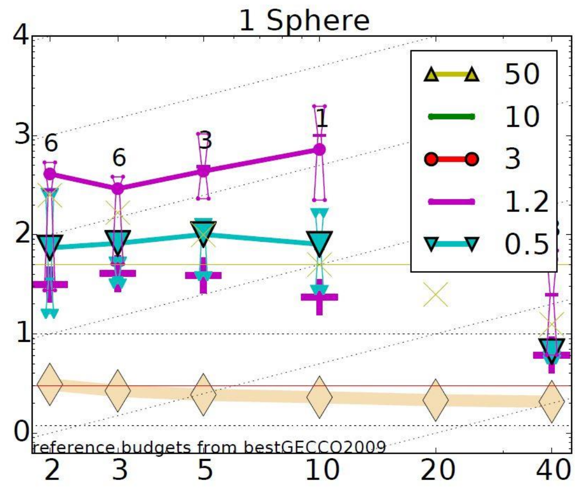

Image 8

2.2.4 Fourth Version of our GA (Setup + Results)

In our fourth version, we created a method that does single-point crossover picking a random point for each instance. We also added a crossover rate variable in order to control the crossover probability of happening. We did various experiments by modifying the setup of our algorithm each time, and we collected the results.

Experiment Title	Population Size	Dimensions	Generations	Average Number of Mutations	Average Ftarget – Fbest [2D]	Average Ftarget – Fbest [3D]	Average Ftarget – Fbest [5D]	Average Ftarget – Fbest [10D]	Average Ftarget – Fbest [20D]	Average Ftarget – Fbest [40D]
Exp1	2	[2,3,5,10,20,40]	100	0	8.793942e+00	1.556156e+01	3.032489e+01	5.699420e+01	1.020044e+02	1.983180e+02

Exp2	4	[2,3,5,10,20,40]	100	0	8.504235e+00	1.314267e+01	2.916203e+01	6.128318e+01	1.115251e+02	2.259169e+02
Exp3	6	[2,3,5,10,20,40]	100	1	8.986689e+00	1.410443e+01	3.884482e+01	1.656452e+02	3.686131e+02	8.093868e+02
Exp4	10	[2,3,5,10,20,40]	100	0	1.436792e+01	3.591779e+01	1.178276e+02	4.654304e+02	1.185090e+03	2.628713e+03
Exp5	20	[2,3,5,10,20,40]	100	0	1.747757e+01	6.770164e+01	1.859123e+02	8.593877e+02	2.034790e+03	5.060590e+03
Exp6	50	[2,3,5,10,20,40]	100	0	2.216525e+01	7.138226e+01	2.674500e+02	1.055789e+03	2.713298e+03	6.846166e+03
Exp7	100	[2,3,5,10,20,40]	100	0	1.611512e+01	5.303491e+01	2.270724e+02	1.060219e+03	2.892349e+03	7.364302e+03
Exp8	500	[2,3,5,10,20,40]	100	0	8.634155e+00	2.918109e+01	1.161862e+02	7.961867e+02	2.604825e+03	7.115053e+03
Exp9	1000	[2,3,5,10,20,40]	100	0	8.420289e+00	2.099030e+01	1.010086e+02	6.040916e+02	2.421928e+03	6.738374e+03
Exp10	2	[2,3,5,10,20,40]	200	2	8.832342e+00	1.478310e+01	2.954190e+01	5.734840e+01	1.053511e+02	2.007454e+02
Exp11	4	[2,3,5,10,20,40]	200	1	8.384982e+00	1.390912e+01	2.461260e+01	5.497615e+01	1.018594e+02	1.973243e+02
Exp12	6	[2,3,5,10,20,40]	200	1	8.414315e+00	1.336515e+01	2.655916e+01	5.179414e+01	9.767624e+01	1.955794e+02
Exp13	10	[2,3,5,10,20,40]	200	1	8.422742e+00	1.324262e+01	2.719131e+01	8.288449e+01	2.795796e+02	4.359646e+02
Exp14	20	[2,3,5,10,20,40]	200	0	9.645462e+00	1.942206e+01	7.012807e+01	4.360873e+02	1.183579e+03	2.512465e+03
Exp15	50	[2,3,5,10,20,40]	200	0	1.226509e+01	4.070486e+01	1.659433e+02	7.549355e+02	2.560839e+03	5.386619e+03
Exp16	100	[2,3,5,10,20,40]	200	0	1.392162e+01	3.986756e+01	1.843133e+02	1.061608e+03	2.788164e+03	6.646713e+03
Exp17	500	[2,3,5,10,20,40]	200	0	8.501249e+00	2.538514e+01	1.272295e+02	6.958420e+02	2.676327e+03	6.911986e+03
Exp18	1000	[2,3,5,10,20,40]	200	0	8.422742e+00	2.091846e+01	1.113606e+02	7.090033e+02	2.476362e+03	6.496043e+03
Exp19	2	[2,3,5,10,20,40]	500	4	9.005675e+00	1.507167e+01	2.992018e+01	5.609148e+01	1.055496e+02	2.028790e+02
Exp20	4	[2,3,5,10,20,40]	500	3	8.366955e+00	1.319794e+01	2.627324e+01	5.639993e+01	1.037990e+02	1.991262e+02
Exp21	6	[2,3,5,10,20,40]	500	2	8.423062e+00	1.396371e+01	2.408813e+01	5.050460e+01	9.915294e+01	1.987756e+02
Exp22	10	[2,3,5,10,20,40]	500	1	8.366955e+00	1.298376e+01	2.479773e+01	4.750572e+01	9.650192e+01	1.960393e+02
Exp23	20	[2,3,5,10,20,40]	500	0	8.353195e+00	1.286881e+01	2.202815e+01	4.889356e+01	1.035799e+02	2.125538e+02
Exp24	50	[2,3,5,10,20,40]	500	0	8.353195e+00	1.311575e+01	4.388418e+01	2.625299e+02	9.498295e+02	2.431559e+03
Exp25	100	[2,3,5,10,20,40]	500	0	8.509995e+00	1.894007e+01	7.681466e+01	4.750293e+02	2.008194e+03	4.779669e+03
Exp26	500	[2,3,5,10,20,40]	500	0	8.400235e+00	2.143814e+01	1.164151e+02	6.984189e+02	2.700604e+03	7.073398e+03
Exp27	1000	[2,3,5,10,20,40]	500	0	8.353195e+00	2.062892e+01	9.300490e+01	6.530043e+02	2.440017e+03	6.511297e+03

Table 5

We can see that we have two improvement in the error rate of the dimensions = 10,40 (1.955794e+02 instead of 1.966136e+02 and 1.055789 instead of 1.067436) respectively. The figures of our best results are shown below.

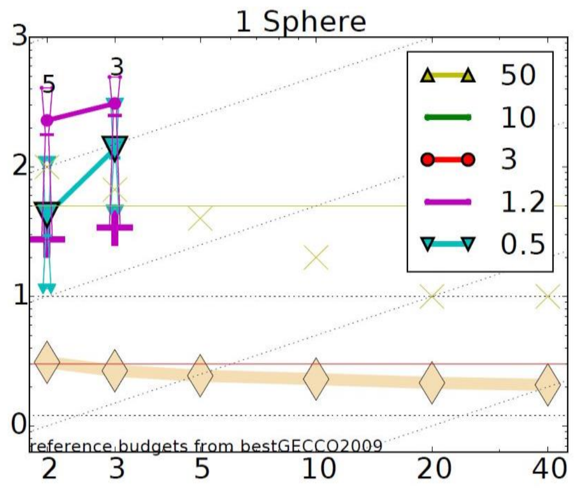

Image 9

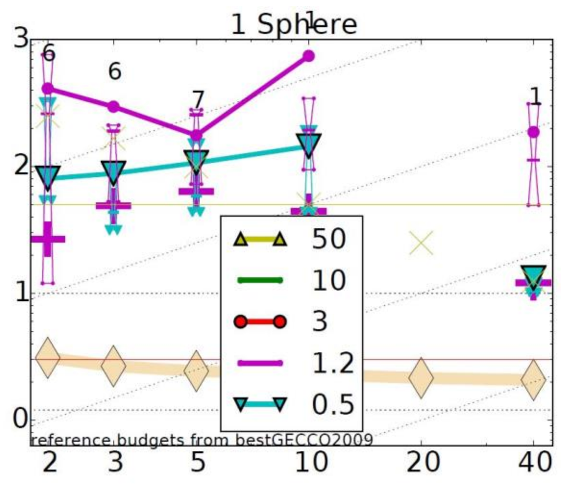

Image 10

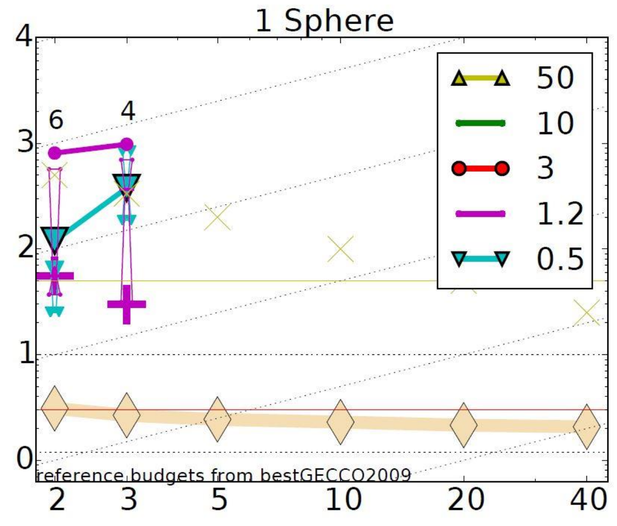

Image 11

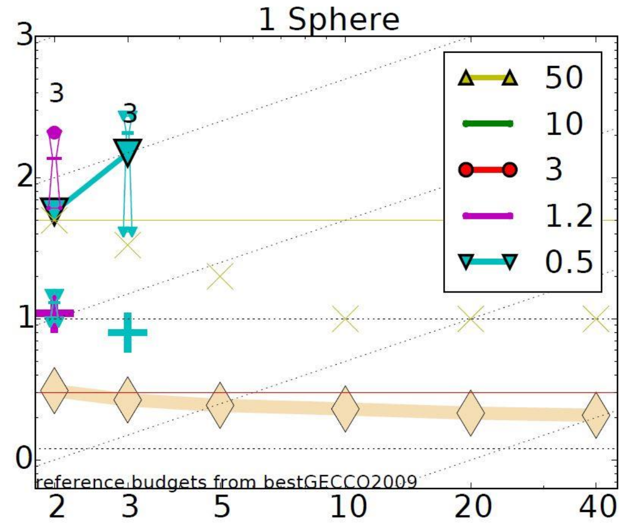

Image 12

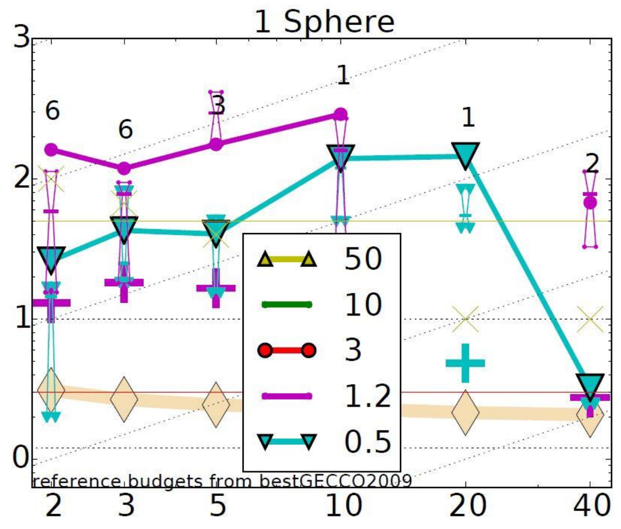

Image 13

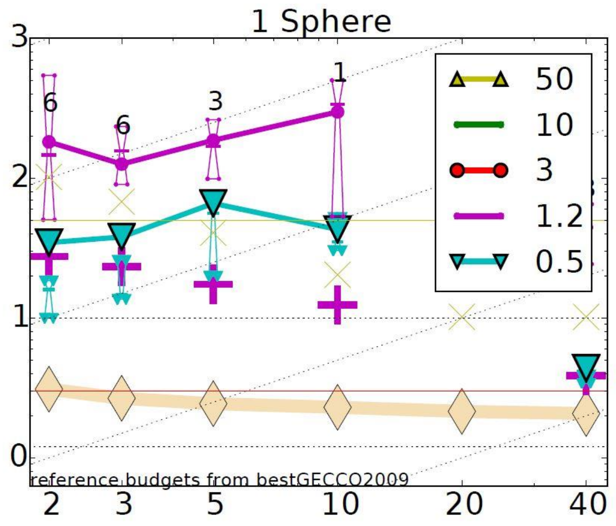

Image 14

2.2.5 Fifth Version of our GA (Setup + Results)

In this final version of our algorithm, we included a roulette selection method. This method calculates the fitness of each individual and the overall fitness of each generation. After that, it assigns weights to each of the individuals and it creates a new matrix by picking new individuals with probability = weight. With this way, if an individual has a better fitness than another it has a bigger chance to be picked in the new population.

Also this time, we performed twenty-seven different experiments with different setup each time. The results of these experiments are below.

Experiment Title	Population Size	Dimensions	Generations	Average Number of Mutations	Average Ftarget – Fbest [2D]	Average Ftarget – Fbest [3D]	Average Ftarget – Fbest [5D]	Average Ftarget – Fbest [10D]	Average Ftarget – Fbest [20D]	Average Ftarget – Fbest [40D]
Exp1	2	[2,3,5,10,20,40]	100	1	1.257154 e+01	1.927121 e+01	3.416761 e+01	6.004195 e+01	1.044549 e+02	2.091875 e+02
Exp2	4	[2,3,5,10,20,40]	100	1	1.257154 e+01	1.944188 e+01	3.424193 e+01	6.000936 e+01	1.033036 e+02	2.041230 e+02
Exp3	6	[2,3,5,10,20,40]	100	0	1.181048 e+01	1.940970 e+01	3.294538 e+01	5.897455 e+01	1.239709 e+02	2.072640 e+02
Exp4	10	[2,3,5,10,20,40]	100	0	1.167928 e+01	2.416383 e+01	3.231864 e+01	7.939538 e+01	1.941060 e+02	3.466735 e+02
Exp5	20	[2,3,5,10,20,40]	100	0	1.192024 e+01	2.315078 e+01	4.255404 e+01	2.180323 e+02	5.286532 e+02	2.270335 e+03
Exp6	50	[2,3,5,10,20,40]	100	0	1.192056 e+01	1.536776 e+01	7.741970 e+01	5.638642 e+02	1.495608 e+03	4.373448 e+03
Exp7	100	[2,3,5,10,20,40]	100	0	1.396248 e+01	2.833791 e+01	1.143642 e+02	6.662546 e+02	1.930726 e+03	5.609112 e+03
Exp8	500	[2,3,5,10,20,40]	100	0	1.135458 e+01	2.152693 e+01	6.119469 e+01	3.591206 e+02	1.758073 e+03	5.568360 e+03
Exp9	1000	[2,3,5,10,20,40]	100	0	1.083544 e+01	1.741382 e+01	4.301829 e+01	3.055773 e+02	1.506599 e+03	5.017377 e+03
Exp10	2	[2,3,5,10,20,40]	200	2	1.257154 e+01	1.946044 e+01	3.463420 e+01	6.052042 e+01	1.079638 e+02	2.094563 e+02
Exp11	4	[2,3,5,10,20,40]	200	0	1.257154 e+01	1.942737 e+01	3.362592 e+01	6.030728 e+01	1.053230 e+02	2.064848 e+02
Exp12	6	[2,3,5,10,20,40]	200	0	1.257154 e+01	1.867697 e+01	3.164715 e+01	5.792203 e+01	1.040081 e+02	2.040330 e+02
Exp13	10	[2,3,5,10,20,40]	200	1	1.257154 e+01	1.946044 e+01	3.254322 e+01	5.633686 e+01	1.063070 e+02	2.049922 e+02
Exp14	20	[2,3,5,10,20,40]	200	0	1.257154 e+01	1.772241 e+01	2.649295 e+01	5.796423 e+01	1.721038 e+02	4.407443 e+02
Exp15	50	[2,3,5,10,20,40]	200	0	1.058605 e+01	1.941970 e+01	3.779959 e+01	1.768532 e+02	8.863666 e+02	2.552023 e+03
Exp16	100	[2,3,5,10,20,40]	200	0	1.171949 e+01	1.835574 e+01	4.433890 e+01	2.338728 e+02	1.255934 e+03	4.383540 e+03
Exp17	500	[2,3,5,10,20,40]	200	0	1.051373 e+01	1.942533 e+01	5.083549 e+01	3.811698 e+02	1.761087 e+03	5.670297 e+03
Exp18	1000	[2,3,5,10,20,40]	200	0	1.116866 e+01	1.852222 e+01	5.574840 e+01	3.823471 e+02	1.546772 e+03	5.109214 e+03
Exp19	2	[2,3,5,10,20,40]	500	3	1.244216 e+01	1.913117 e+01	3.293844 e+01	5.981537 e+01	1.081847 e+02	2.100909 e+02
Exp20	4	[2,3,5,10,20,40]	500	2	1.235362 e+01	1.836239 e+01	3.309092 e+01	5.903311 e+01	1.057452 e+02	2.083460 e+02
Exp21	6	[2,3,5,10,20,40]	500	2	1.201688 e+01	1.767092 e+01	3.171629 e+01	5.773005 e+01	1.039104 e+02	2.046073 e+02
Exp22	10	[2,3,5,10,20,40]	500	1	1.178690 e+01	1.789485 e+01	3.161591 e+01	5.663920 e+01	1.012832 e+02	2.045019 e+02
Exp23	20	[2,3,5,10,20,40]	500	0	1.229261 e+01	1.700280 e+01	2.958993 e+01	5.453709 e+01	1.004120 e+02	2.022643 e+02
Exp24	50	[2,3,5,10,20,40]	500	0	1.194968 e+01	1.636980 e+01	2.786293 e+01	5.215551 e+01	1.073643 e+02	3.866870 e+02

Exp25	100	[2,3,5,10,20,40]	500	0	1.160173 e+01	1.475514 e+01	3.619796 e+01	9.064270 e+01	3.673272 e+02	1.986122 e+03
Exp26	500	[2,3,5,10,20,40]	500	0	1.116610 e+01	1.586701 e+01	5.973640 e+01	3.668476 e+02	1.952580 e+03	5.680688 e+03
Exp27	1000	[2,3,5,10,20,40]	500	0	1.111042 e+01	1.493519 e+01	4.213977 e+01	2.804164 e+02	1.602208 e+03	5.376645 e+03

Table 6

Using the roulette selection method we can observe a slight improvement in the error rate of 20 dimensions (1.004120 instead of 1.005906).

Below we can see the figures of our best results.

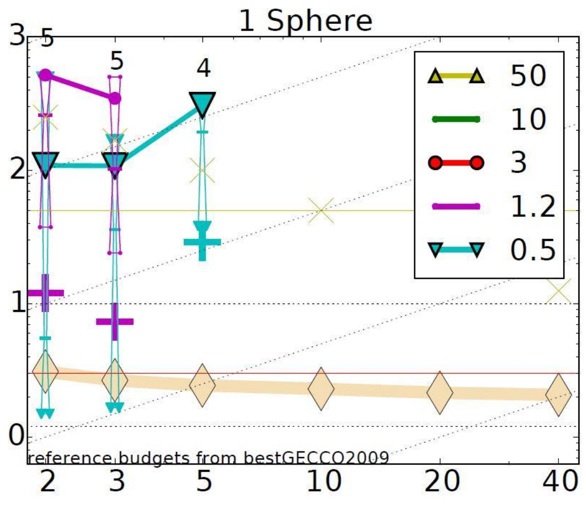

Image 15

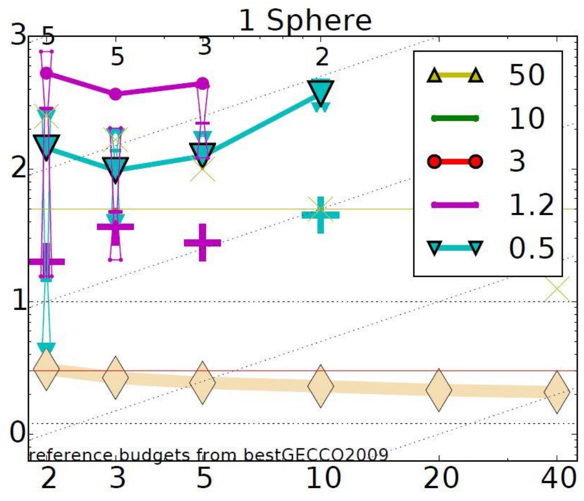

Image 16

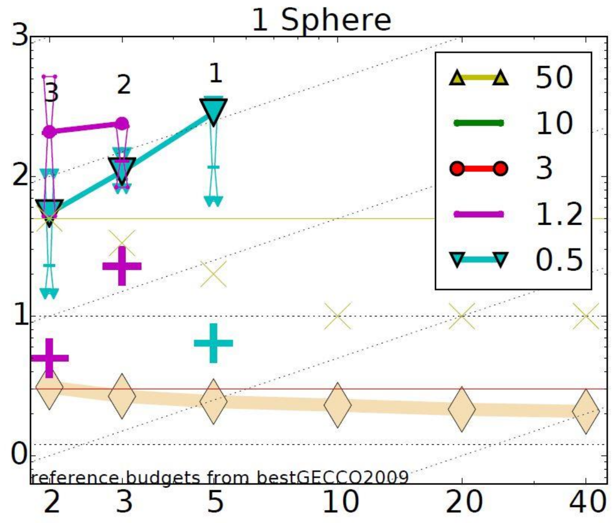

Image 17

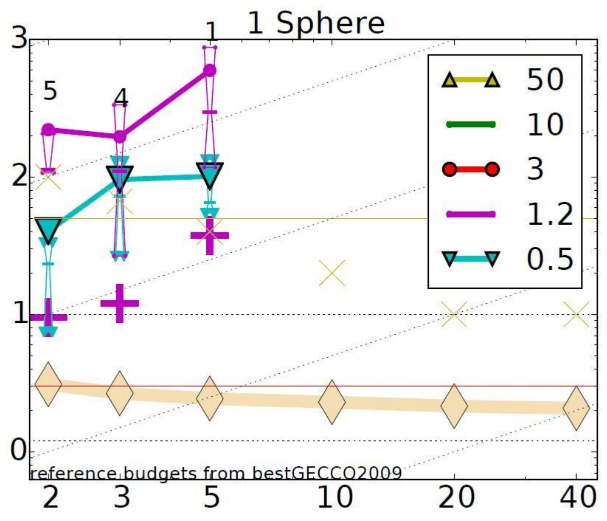

Image 18

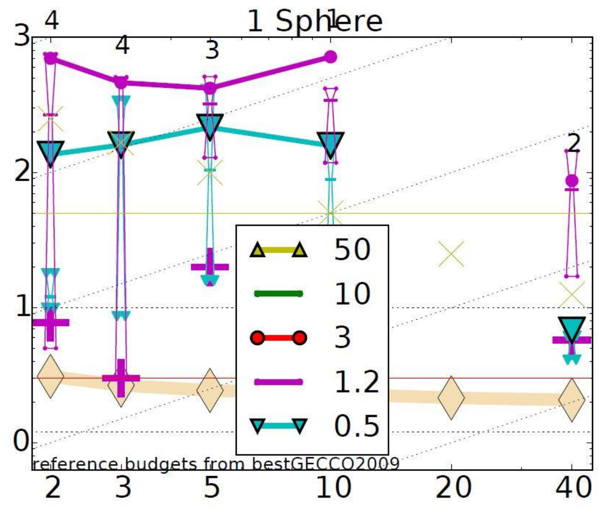

Image 19

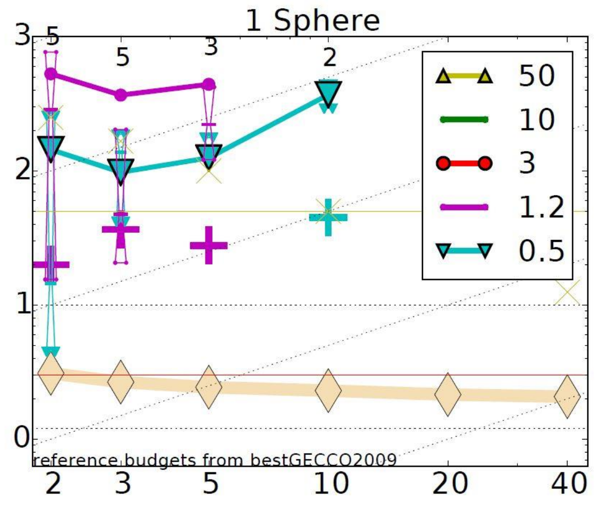

Image 20

2.3 Conclusion

We created a Genetic Algorithm that is based on hill-climbing. Changing the variable that controls the rate of climbing we can be sure that we are not moving very fast at the search space so that we do not miss any important spots. With mutation, it can overcome obstacles in the search space like plateaus, local minima, etc. Using crossover, it can generate new interesting individuals that will explore the search space faster and provide better results. With variables that control the rate of these two methods, the algorithm, can be configured easily to cover different needs i.e. wider variety of individuals, slower but steadier search, etc.

Selection using the roulette method can create a better set of individuals that will lead to better fitness. If we combine all these different methods, we are going to have better results.

Our experiments also show that sometimes if we increase a specific element as the number of generations may be time-consuming but will definitely lead in better performance. So, the algorithm`s parameters should be configured for each individual problem to guarantee performance.

References

1. Seiffert, U. (2001) *Multiple Layer Perceptron Training Using Genetic Algorithms*. Available at: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2001-460.pdf (Accessed: 13 November 2016).
2. Andersen, H.C. and Tsoi, A.C. (1993) *A Constructive Algorithm for the Training of a Multilayer Perceptron Based on the Genetic Algorithm*. Available at: http://www.complex-systems.com/pdf/07-4-1.pdf (Accessed: 13 November 2016).
3. Correa, A., González, A. and Ladino, C. (2011) *Genetic Algorithm Optimization for Selecting the Best Architecture of a Multi-Layer Perceptron Neural Network: A Credit Scoring Case*. Available at: http://support.sas.com/resources/papers/proceedings11/149-2011.pdf (Accessed: 13 November 2016).
4. Anon, (2016). [online] Available at: https://tjhsst.edu/~rlatimer/techlab08/LeswingPosterQ2-08.pdf [Accessed 13 Nov. 2016].
5. Montana, D.J. and Davis, L. (2004) *Http://www.Ijcai.Org/Proceedings/89-1/Papers/122.Pdf*. Available at: http://www.ijcai.org/Proceedings/89-1/Papers/122.pdf (Accessed: 13 November 2016).