HERIOT WATT UNIVERSITY

Biologically Inspired Computation

Coursework 1

Black-Box Optimization

Ali Habib and Emma Aitken

H00156787 and H00150638

Contents

1. Introduction

The link to the GitHub with our code is:
https://github.com/habiba29/BioInspired/tree/master/bbobpproc/bbob.v15.03/python

The name of the file that contains the code is “bbo.py”.

The file that contains the results is ppdata -> results.

Our Multi-Layer Perceptron is built with three layers. L0 is our input layer, L1 the hidden layer, and L2 the output layer. We decided upon the use of three layer as it allowed us to have multiple sets of weights to be optimized by the Genetic Algorithm without the MLP becoming overly complex. The connection, that represents the synapse between neurons on layer 0 and 1 of the MLP, is syn0 and syn1 represents the synaptic connection between layer 1 and 2.

The genetic algorithm that evolves the weights on the synaptic connections uses several technics to progress them. Mutation, crossover and tournament selection are the technics used, the reasoning behind these choices will be explained in section 2. The GA measures the fitness of each of the set of weights that link L0 to L2. The lower the fitness is the better.

2. Setup and Settings

The MLP design is a 3-layered designed neural network, where the input is random, as this allows you to train the weights such that the output of the NN is as close to the expected output as possible. The input is an array scaled to the size of the training instance and dim. Dim allows it to be scaled which is provided by COCO. The expected output is found by passing the random input to the coco function so it gives us a target value. The weight of the first synapses is random as it is unknown due to the nature of the hidden layer. The second synapses are set as the difference of the length of the chromosome and the amount of neuron, this is to give a workable size of weights. The second synapses are the weights that are changing as these are the weights from the hidden layer to the output which is what we are trying to correct to population (chromosome/weights). The size of the chromosome scaled to the dim provided by COCO multiplied by a constant of 40 neurons, scale the chromosome to the size of the problem.

The input is passed to the activation function, which is a sigmoid function, giving us L0(the first layer) this allows us to use the first synapses and L0 to produce the hidden layer L1, using the second synapses and L1 is then gives us our actual output, which is later compared to the expected output given to us using out input within the COCO function.

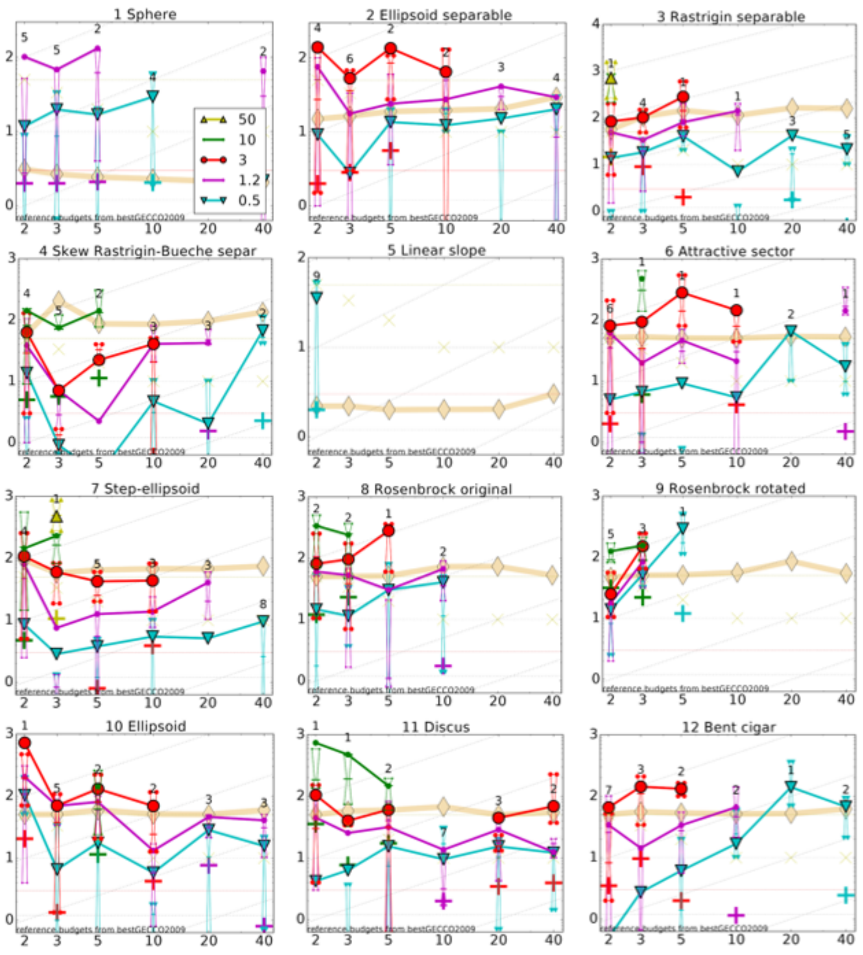
To train the population we get the initial error which is to use the initial population as the chromosome being inputted to the Neural network along with the input. We then get the error(fitness) of the output from the Neural Network and what COCO is expecting from our output. Using this error we try and find the error which is closes to 0 (Meaning its very close to what's expected) . We then store the best error as the first in the population and mutate the rest of the population. We then run the mutated population through the NN again and check the error again. This provides us with the error after mutation which allows us to see how close the expected output and our actual input are, allowing us to see whether the population (weights) are optimal.

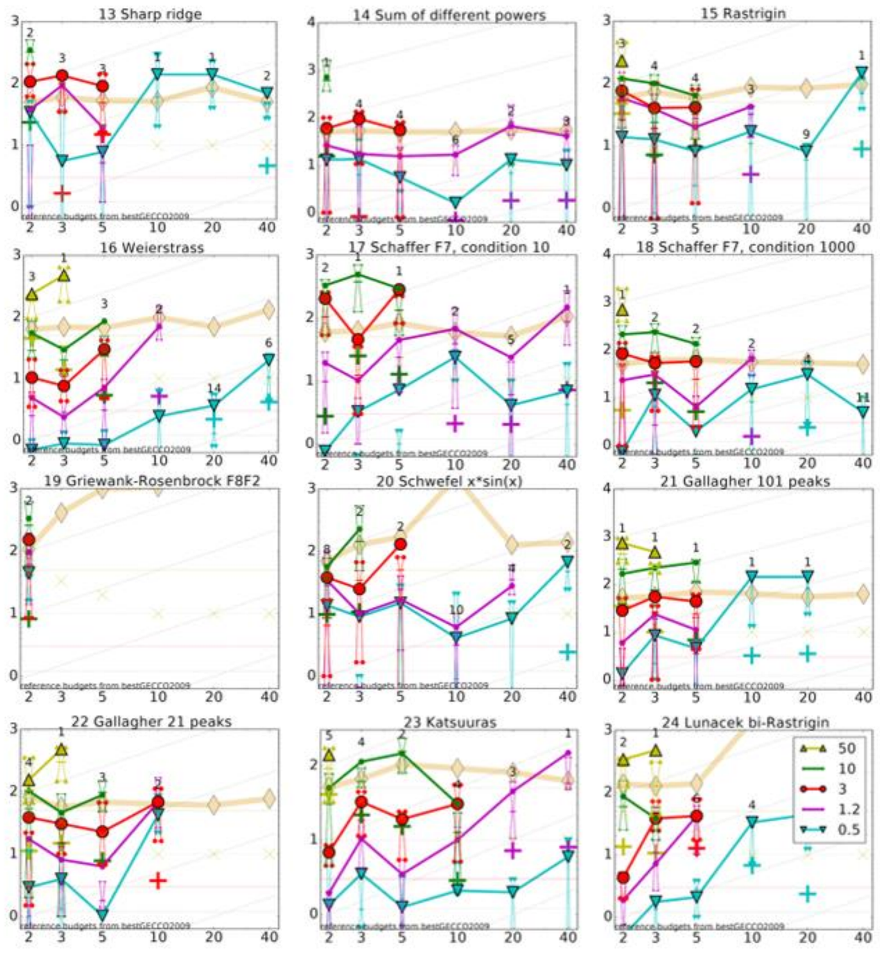
The mutation used in the GA is a single-gene random-allele mutation and the mutation rate used is 0.3. It was decided to use single-gene mutation as it allows occasionally for the same individual to appear twice in the next generation. This occurrence could indicate a strong individual and therefore increase either the likelihood or the speed at which the optimal weights are discovered. The mutation rate was chosen as when it was lower such as; 0.08, 0.1, 0.15, we felt that the generations were not changing enough to give an optimal solution and if it was increased any further – 0.7, 0.6, 0.55 – we felt that it was drifting too far from the optimal result as it was evolving rapidly.

One-point crossover is the type of crossover used rather than either two-point crossover or uniform crossover. The one-point crossover takes a random position of one of the parents and splices it. It then splices the second parent at the same position and combines the first portion of parent one with the second portion of parent two to create their child. The child is then added to the next generation. We did not feel that there was much to be gain by using two-point crossover – randomly choosing two points in the parents and combining the first and third portions of parent one and the second portion of parent two to create the child – but this method of crossover added complexity to the program.

Tournament selection, which picks two individuals of the same generation and has them "compete" against each other. The winner of the tournament, which is the individual that has the lower fitness, survives and is included in the next generation. This selection method allows for the best individuals to be carried forward into the next generation but weaker individuals can also be carried forward as they may be competing against other weak individuals. This allows for diversity in the population to be kept and not become elitist.

3. Results


1 Sphere
2 Ellipsoid separable
3 Rastrigin separable
4 Skew Rastrigin-Bueche separ
5 Linear slope
6 Attractive sector
7 Step-ellipsoid
8 Rosenbrock original
9 Rosenbrock rotated
10 Ellipsoid
11 Discus
12 Bent cigar
50
10
3
1.2
0.5
reference budgets from bestGECCO2009
2
3
5
10
20
40



4. Conclusion

To conclude the MLP designed seems to work as a NN but the training of the function is not correct as the error seems to be extremely large and never seems to get any better. This means the mutation is not being correctly Implemented and due to time constraints, the tournament and crossover were never implemented into the GA but were left in the code so that we could show we had thought about it.