**Genetic Algorithm using MapReduce**

The objective of the assignment was to parallelize and run the Genetic Algorithm in MapReduce.

**Approach: Fitness Evaluation level**

The idea is to make each Mapper evaluate the fitness for each chromosome in parallel, while a single Reducer is responsible to collect the results and to perform the evolving operations like selection, crossover and mutation, to generate the new offspring.

In the Driver class, I passed the input and made the 40 chromosomes in binary format and wrote them in a file. The file has key-value pair with key as chromosomes and constant “one” as value.

I used this file as the input to the Mapper, in which fitness level is calculated for each chromosome.

To parallelize the computation of the fitness function for each chromosome, I wanted to assign it to different mapper. To achieve that, I fed the Mapper with input having unique keys. The input of Mapper has key as the chromosome and constant “one” as value.

Mapper Input 🡪 [<chromosome>, one]

In the Mapper class, the fitness value is calculated of each chromosomes and the output is the constant “blah” as key and list of chromosome and fitness as value.

Mapper Output 🡪 [blah, (<chromosome>, <fitness>)]

This will be Reducer Input as well.

Because of the same key, the output of Mapper will be aggregated on basis of key and the value will be passed to a single reducer which will perform selection, crossover and mutation.

The output of the Reducer is the most optimized solution in binary form.

I was able to run the code in Standalone mode.

**Bug in professor code:**

There was a bug in professor Java code. *Else* condition is missing when calculating the score of chromosomes. The correct code is

// Scores this chromo

public final void scoreChromo(int target)

{

total = addUp();

if (total == target)

score = 0;

else

score = (double)1 / (target - total);

}

**How would you accomplish a similar task using ANN?**

In artificial neural network, the weights are adjusted for recognizing patterns and predicting outputs based on inputs.

To train the genetic algorithm using artificial neural network, we can optimize the fitness function by adding a layer to the program which calculates the weights.

All the weights in the network are joined to make one string. This string is then used in the GA as a member of the population. Each string represents the weights of a complete network. We can use gradient descent methods for adapting the weights of the network.

The initial set of solutions is produced by a random number generator. Each solution in the population is a string comprising n elements, where n is the number of trainable connections. Use binary encoding, where each element is 16 bits long and holds the value of a trainable connection. From the point of view of the GA, all connection weights are handled in the same way, i.e. training of feedback connections is carried out identically to training the feedforward connection.

Input Vector **🡪** ANN **🡪** Difference between Output Vector & Input Vector **🡪** Adjust Weights

**How can you leverage MapReduce to train your ANN?**

To train the data, MapReduce is used to parallelize the computation. The MapReduce job consists of the mapper and reducer functions, of which the mapper function will extract key/value pairs from input data and transfer them into a list of intermediate key/value pairs, and reducer function merges these intermediate values corresponding to the same key generated from mapper function to produce output values. In our case, the input value will be the generated chromosomes.

The intermediate key/value pair will be the weights. Reducer function will use these weights to compute the fitness values of the chromosomes. If there exits an intolerable difference between the precision of training set and expected precision, the MapReduce job will loop until an acceptable result has worked out.