HERIOT WATT UNIVERSITY

Biologically inspired computation

Course Work 1

Issa Haddad

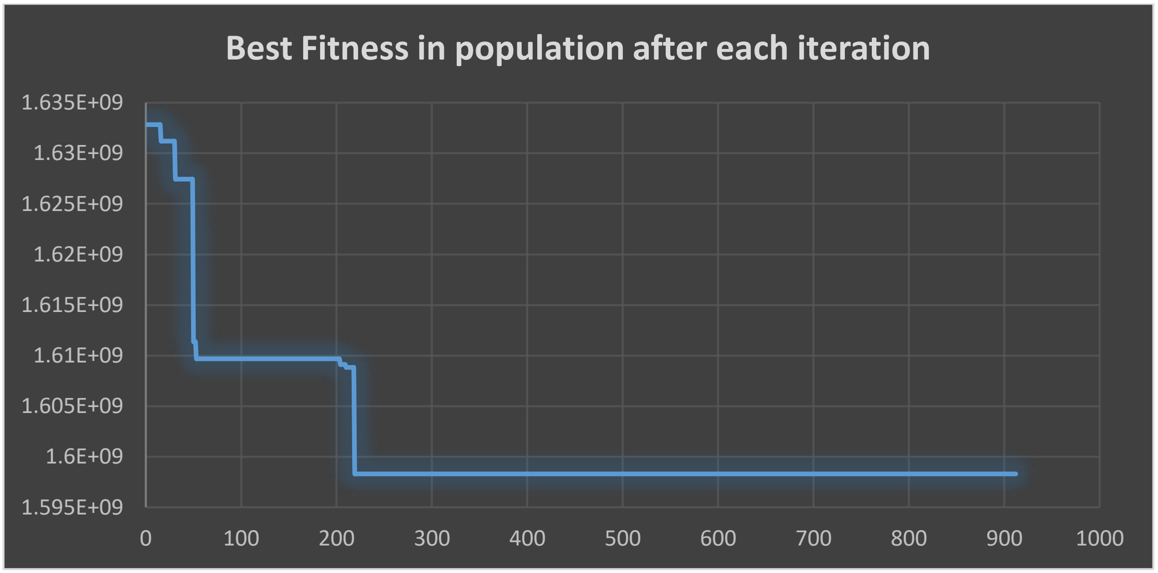
H00278537

Oct 31, 2017

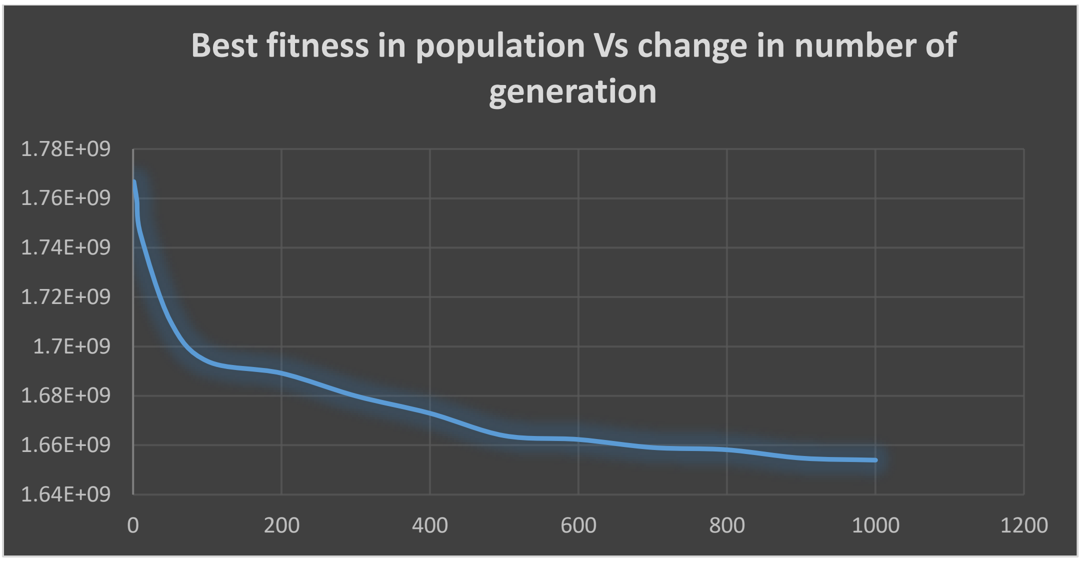
<u>Section 1.A – Evolutionary Algorithm for Traveling Salesman Problem:</u>

Before performing our evolutionary algorithm, I get content of cities (latitude, longitude) directly from the site. That is because any updates on the content will directly be reflected on my code. In the process I remove redundant cities and make sure they are all unique. I start by creating a 2D array for my population of size 500 where each element consists of an array of size Cities.size() and includes unique random elements from 1 to Cities.size(). I also create another array of size Cities.size() that holds the Fitness of each population respectively calculated using the great circle distance. I use tournament selection of size ((population/100) +1). I chose the individual with the best fitness from my tournament selection. I start by mutating this individual using “K inversion mutation”. The value of “K” and “Start” is randomized in each mutation and my code also handles situations where the value “K” crosses the size limit of the individual and continues from zero index. I then compare the fitness of the mutated child and the parent. If the parent has better fitness than the child, I increment a “No Improvement” Counter, otherwise I replace the child with the worst individual in my population and update the fitness array respectively. I finally output, keeping track of the best fitness in my population. I repeat the process until either the number of max generations reaches 1000 or the counter for “No improvement” reaches 30.

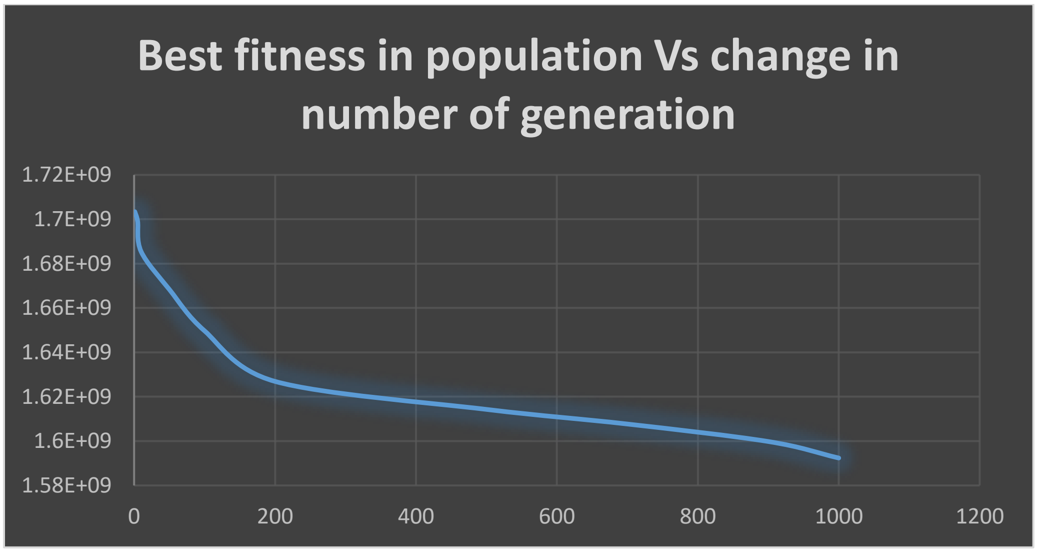
Note: I print the code on an external text file when you run the code, because Eclipse IDE does not give me enough space to view all the output.



This graph shows “Best Fitness” in population after each iteration of running the program once. This program was running on the condition (1000 generations || 30 max no improvement counter) & 500 population.



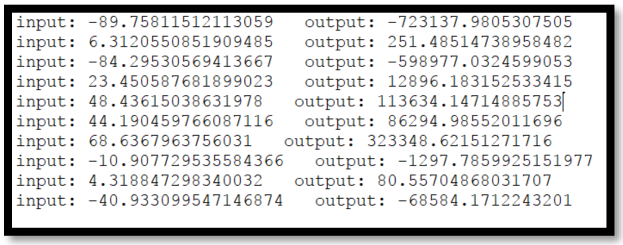
This graph shows “Best Fitness” after generation completion for 100 population and tournament size = (popsize/100)+1. This data is collected from running the program many times, with each time changing the max number of generations gradually from 1 to 1000. The fitness represents the average fitness of running the program more than once on a specific max number of generation. Max number of generations used (1,5, 10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000)



Performing the same exercise as the graph before, but increasing the number of population form 100 to 1000 and increasing the tournament size from ((popsize/100)+1) to ((popsize/100)+20). We can see that the initial “Best Fitness” from population improved from 1.78E9 to 1.70E9. we can also notice that the final “Best Fitness” from population also improved from 1.65E9 to 1.59E9.

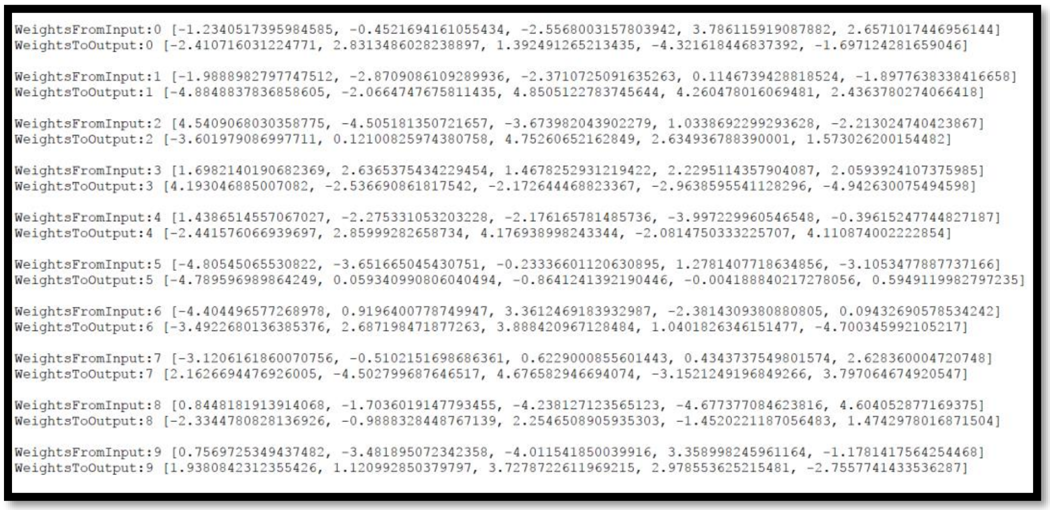
Section 1.B – Multilayer Perceptron:

The function I implemented in my Java code was the Cubic function. I have also use the Sigmoid function to be my activation function. Because of the Cubic function, my topology consists of 1 input, 1 hidden layer with 5 nodes, and 1 output. I start by initializing 10 random inputs for my Cubic function, these inputs are all unique and can take a value from -100 to +100. I respectively calculate the actual output of these inputs and store them in an Array-List of Array-List. I also initialize 10 sets of different random weights between [-5, +5]. Each set consists of 2 arrays, a 1x5 array of weights from input to hidden layer and another 5x1 array of weights from hidden layer to output layer. I perform forward propagation 10 times for each of the 10 point inputs that I have randomized. Every time using a different set of randomized weights initialized earlier. In each attempt for each point, I calculate error. For the total 10 point inputs with each set of weights, I calculate the Mean squared error and store them in an array. I then find which set of weights gave me the best or smallest Mean squared error. I repeat the procedure 1 last time for just the best set of weights and for the 10 point inputs again. The results from the code I had no intelligence, thus the output rarely provides us with meaningful insights.



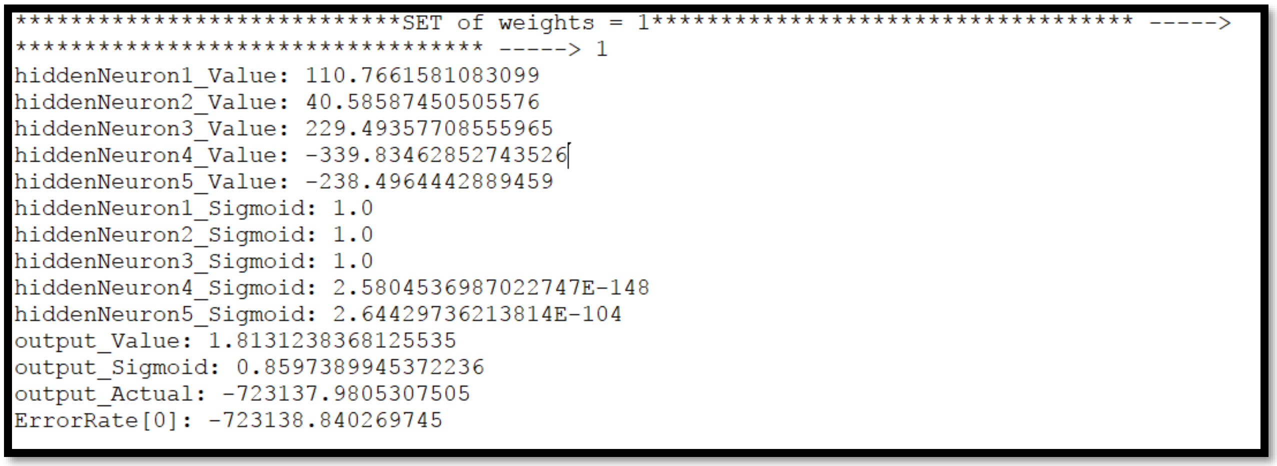
```
input: -89.75811512113059   output: -723137.9805307505
input: 6.3120550851909485   output: 251.48514738958482
input: -84.29530569413667   output: -598977.0324599053
input: 23.450587681899023   output: 12896.183152533415
input: 48.43615038631978   output: 113634.14714885753
input: 44.190459766087116   output: 86294.98552011696
input: 68.6367963756031   output: 323348.62151271716
input: -10.907729535584366   output: -1297.7859925151977
input: 4.318847298340032   output: 80.55704868031707
input: -40.933099547146874   output: -68584.1712243201
```

My initial random inputs, and their corresponding actual outputs.

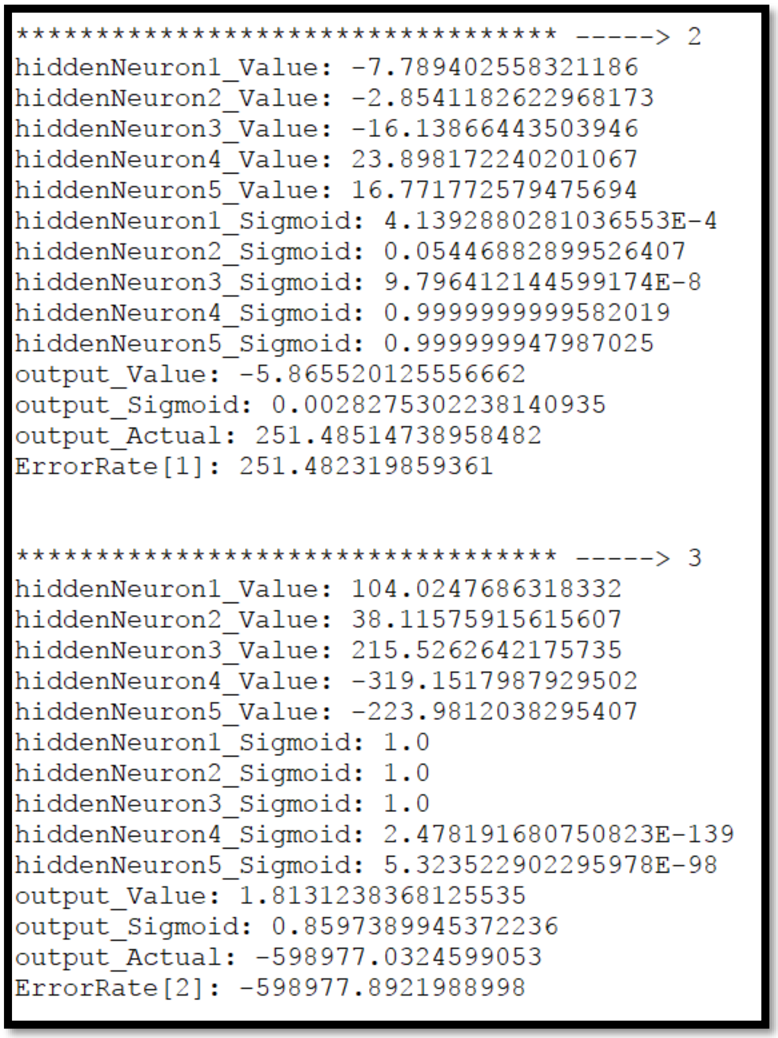


```
WeightsFromInput:0 [-1.2340517395984585, -0.4521694161055434, -2.5568003157803942, 3.786115919087882, 2.6571017446956144]
WeightsToOutput:0 [-2.410716031224771, 2.8313486028238897, 1.392491265213435, -4.321618446837392, -1.697124281659046]

WeightsFromInput:1 [-1.9888982797747512, -2.8709086109289936, -2.3710725091635263, 0.1146739428818524, -1.8977638338416658]
WeightsToOutput:1 [-4.8848837836858605, -2.0664747675811435, 4.8505122783745644, 4.260478016069481, 2.4363780274066418]

WeightsFromInput:2 [4.5409068030358775, -4.505181350721657, -3.673982043902279, 1.0338692299293628, -2.213024740423867]
WeightsToOutput:2 [-3.601979086997711, 0.12100825974380758, 4.75260652162849, 2.634936788390001, 1.573026200154482]

WeightsFromInput:3 [1.6982140190682369, 2.6365375434229454, 1.4678252931219422, 2.2295114357904087, 2.0593924107375985]
WeightsToOutput:3 [4.193046885007082, -2.536690861817542, -2.172644468823367, -2.9638595541128296, -4.942630075494598]

WeightsFromInput:4 [1.4386514557067027, -2.275331053203228, -2.176165781485736, -3.997229960546548, -0.39615247744827187]
WeightsToOutput:4 [-2.441576066939697, 2.85999282658734, 4.176938998243344, -2.0814750333225707, 4.110874002222854]

WeightsFromInput:5 [-4.80545065530822, -3.651665045430751, -0.23336601120630895, 1.2781407718634856, -3.1053477887737166]
WeightsToOutput:5 [-4.789596989864249, 0.059340990806040494, -0.8641241392190446, -0.004188840217278056, 0.5949119982797235]

WeightsFromInput:6 [-4.404496577268978, 0.9196400778749947, 3.3612469183932987, -2.3814309380880805, 0.09432690578534242]
WeightsToOutput:6 [-3.4922680136385376, 2.687198471877263, 3.888420967128484, 1.0401826346151477, -4.700345992105217]

WeightsFromInput:7 [-3.1206161860070756, -0.5102151698686361, 0.6229000855601443, 0.4343737549801574, 2.628360004720748]
WeightsToOutput:7 [2.1626694476926005, -4.502799687646517, 4.676582946694074, -3.1521249196849266, 3.797064674920547]

WeightsFromInput:8 [0.8448181913914068, -1.7036019147793455, -4.238127123565123, -4.677377084623816, 4.604052877169375]
WeightsToOutput:8 [-2.3344780828136926, -0.9888328448767139, 2.2546508905935303, -1.4520221187056483, 1.4742978016871504]

WeightsFromInput:9 [0.7569725349437482, -3.481895072342358, -4.011541850039916, 3.358998245961164, -1.1781417564254468]
WeightsToOutput:9 [1.9380842312355426, 1.120992850379797, 3.7278722611969215, 2.978553625215481, -2.7557741433536287]
```

My randomized set of weights. I use each set on the random points calculated above.

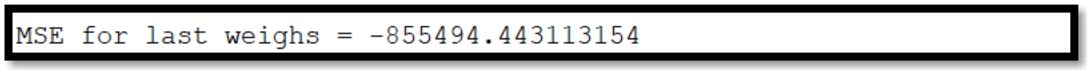


```
****************************SET of weights = 1*********************************** ----->
********************************** -----> 1
hiddenNeuron1_Value: 110.7661581083099
hiddenNeuron2_Value: 40.58587450505576
hiddenNeuron3_Value: 229.49357708555965
hiddenNeuron4_Value: -339.83462852743526
hiddenNeuron5_Value: -238.4964442889459
hiddenNeuron1_Sigmoid: 1.0
hiddenNeuron2_Sigmoid: 1.0
hiddenNeuron3_Sigmoid: 1.0
hiddenNeuron4_Sigmoid: 2.5804536987022747E-148
hiddenNeuron5_Sigmoid: 2.64429736213814E-104
output_Value: 1.8131238368125535
output_Sigmoid: 0.8597389945372236
output_Actual: -723137.9805307505
ErrorRate[0]: -723138.840269745
```



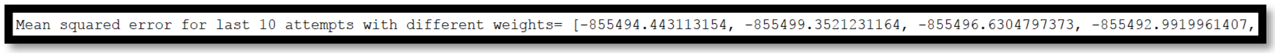
```
********************************** -----> 2
hiddenNeuron1_Value: -7.789402558321186
hiddenNeuron2_Value: -2.8541182622968173
hiddenNeuron3_Value: -16.13866443503946
hiddenNeuron4_Value: 23.898172240201067
hiddenNeuron5_Value: 16.771772579475694
hiddenNeuron1_Sigmoid: 4.1392880281036553E-4
hiddenNeuron2_Sigmoid: 0.05446882899526407
hiddenNeuron3_Sigmoid: 9.796412144599174E-8
hiddenNeuron4_Sigmoid: 0.9999999999582019
hiddenNeuron5_Sigmoid: 0.999999947987025
output_Value: -5.865520125556662
output_Sigmoid: 0.0028275302238140935
output_Actual: 251.48514738958482
ErrorRate[1]: 251.482319859361


********************************** -----> 3
hiddenNeuron1_Value: 104.0247686318332
hiddenNeuron2_Value: 38.11575915615607
hiddenNeuron3_Value: 215.5262642175735
hiddenNeuron4_Value: -319.1517987929502
hiddenNeuron5_Value: -223.9812038295407
hiddenNeuron1_Sigmoid: 1.0
hiddenNeuron2_Sigmoid: 1.0
hiddenNeuron3_Sigmoid: 1.0
hiddenNeuron4_Sigmoid: 2.478191680750823E-139
hiddenNeuron5_Sigmoid: 5.323522902295978E-98
output_Value: 1.8131238368125535
output_Sigmoid: 0.8597389945372236
output_Actual: -598977.0324599053
ErrorRate[2]: -598977.8921988998
```

The above 2 figures are a snapshot from the code I ran, it shows me calculating the output for the first 3 input points on the first set of weights. I continue doing the same for all 10 points and repeat the process for all weights.

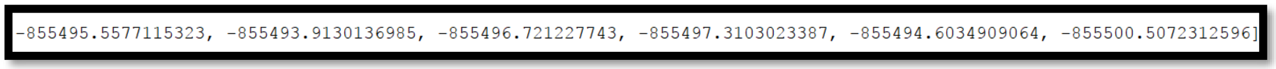


```
MSE for last weighs = -855494.443113154
```

After each set of weights on each of the 10 points, I calculate the MSE and store it in a variable.

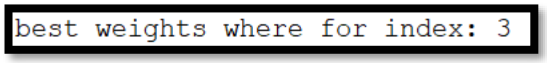


```
Mean squared error for last 10 attempts with different weights= [-855494.443113154, -855499.3521231164, -855496.6304797373, -855492.9919961407,
```



```
-855495.5577115323, -855493.9130136985, -855496.721227743, -855497.3103023387, -855494.6034909064, -855500.5072312596]
```

I output the MSE for each set of weights.



```
best weights where for index: 3
```

I determine which set had the least (closest to zero) MSE, and I output its index. I repeat the process for the 10 points again for this particular set of weights.

Section 2 – Research:

In our fast progressing world today of computer science and applied mathematics, many complex problems arise in which scientists could not find a direct solution using exhaustive search methods and applications. One type of these problems include combinatorial optimization problems. These problems involve grouping, ordering, and assignment, and usually revolve around finding shortest/cheapest route, scheduling, and structure prediction. Perhaps the most famous and the most widely studied of these combinatorial optimization problems is the Travelling Salesman Problem implemented in Part 1A of our coursework. Because our extremely huge solution space does not allow us to apply exhaustive search methods, we tend to resort to genetic algorithms to best explore our search space in hopes of finding the best or the fittest solution. The TSP problem is of type NP-hard problems, which means that the solution could perhaps be of the polynomial order.

In the essence of trying to solve the Traveling Salesman Problem, one must compromise between applying pressure and finding a fast solution or applying diversity and finding a better solution. The disadvantages of endorsing pressure over diversity could result in your algorithm getting stuck in a local optimum, while the disadvantages of endorsing diversity over pressure could result in your algorithm running for an extremely long period of time. Performance of our genetic algorithm highly depends on the selections criteria, mutation method, and the type of update used (Hussain, et al., 2017). Various methods and approaches have been proposed over the years to solve the Traveling Salesman Problem. Some of these methods include Branch and bound, cutting planes, 2-opt, particle swarm, ant colony, simulated annealing, tabu search, neural networks, bee colony, cuckoo search, and finally the genetic algorithm which we have used in part 1-A of our coursework. The Ant Colony Optimization (ACO) and the Particle Swarm Optimization (PSO) are two approaches that fall under a bigger umbrella concept algorithm called the Swarm intelligence.

The Swarm intelligence algorithm evolved from the studying the behavior of bird-flocking, ant colonies, fish colonies, etc. The algorithm involves a population of agents, also known as colony, that interact directly with each other, and indirectly through the environment (Tao YQ, Cui DW, Miao XL, & Chen H, 2007). In the (ACO) approach, the colony represents artificial ants that explore optimal solutions of a certain parameter space represented as graph. These ants record the quality and the position of the solution they pass through, enabling other ants or agents in later iterations to find better solutions. The (PSO) mimics the behavior of bird-flocking and fish-schooling. It starts by generating an initial population of particles, where each holds one solution to a problem along with three other parameters; position, velocity, and fitness. Every particle alters its position at each iteration based on the velocity parameter which is computed from the best among all particles in population and its former best position. The (PSO) approach seem to be the more popular approach over the previous (ACO), for the simple fact that it is very easy to implement and has a high performance (MAH, Imran, & Akter, 2016).

One interesting algorithm used to solve the Traveling Salesman Problem is the simulated annealing. This algorithm is also a heuristic algorithm because it does not find an exact solution, but rather finds a near-optimal solution that is very close to the global optimum and in a reasonable amount of time. Simulated annealing is an enhanced version of the hill-climbing approach which is a part the greedy search approach. Because hill-climbing always starts at a random point and only moves towards a point that has a better fitness, it usually falls into the trap of getting stuck in the local optimum. The simulated annealing approach occasionally takes a point with a lower fitness, to explore additional hills, thus getting a more diverse solution. It starts by picking a random tour, then it adapts a small permutation to obtain a new tour. If the new tour has better fitness, then accept the new tour. If the new tour has a worse fitness, then maybe accept it, but with a certain probability. This probability keeps decreasing after every iteration until a final solution has been elected (Bookstaber, 2014).

As mentioned before, combinatorial optimization problems are applicable to many of the real world problems and does not just revolve around finding the shortest route or path. Another example of a famous combinatorial optimization problem includes the Job-shop scheduling problem (JSP). This problem revolves around allocating a number jobs to a specific number machines to minimize the length of the schedule, or the time from when the jobs start till the time when the jobs finish. Each job is consisted of multiple tasks; each task can only be accomplished only in a particular machine. Three constraints criteria must also be met on this problem. First, the tasks are sequential, so task 2 cannot start before task 1 for job1. Second, at any particular time, a machine can only be working on one given task. Three, a task cannot be interrupted once started (Bhatt & Chauhan, 2015). This Problem has been extensively reseach upon since the 1960’s.

Because this problem is both computationally challenging (NP-hard), and involves sequential tasks, a genetic algorithm has to be used with order based operator and mutation. Because of the many constraints this problem holds, a good evolutionary algorithm to use would be the tabu search. Tabu search also starts with an initial random solution and then check its neighborhood for the next best solution. This also has a tendency to fall into the trap of getting stuck in a local minimum, however tabu search is a bit more lenient. It is a bit more lenient in the sense that it can accept a worse solution only if there exists no better solution in the neighborhood. Another catch for tabu search, is the fact that it does not traverse previously traversed solution. It has memory that keeps track of all the visited solutions, so that it does not visit them again. This makes the search more computationally expensive, but usually ends up finding better solutions than other algorithms (Dell’Amico & Trubian, 1993).

Reference:

Hussain et al. (2017). Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator. *Computational Intelligence and Neuroscience, Volume 2017* (2017), Article ID 7430125, 7 pages

Tao YQ., Cui DW., Miao XL., and Chen H. (2007) An Improved Swarm Intelligence Algorithm for Solving TSP Problem. In: Huang DS., Heutte L., Loog M. (eds) *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence. ICIC 2007*. *Lecture Notes in Computer Science, vol 4682*. Springer, Berlin, Heidelberg

MAH, A., Hossain, S.I., and Akter, S. (2016). A Comparative Study of Prominent Particle Swarm Optimization Based Methods to Solve Traveling Salesman Problem. *International Journal of Swarm Intelligence and Evolutionary Computation.*

Bookstaber, D. (2014). Simulated Annealing for Traveling Salesman Problem. Retrieved December 1, 2014, from http://www.eit.lth.se/fileadmin/eit/courses/ets061/Material2014/SATSP.pdf

Bhatt, N., and Chauhan, N.R. (2015) Genetic algorithm applications on Job Shop Scheduling Problem: A review. *Soft Computing Techniques and Implementations (ICSCTI), 2015 International Conference on*

Dell’Amico, M., and Trubian, M. (1993) Applying Tabu Search to the Job-Shop Scheduling Problem. *Annals of Operations Research 41(3):231-252 · September 1993*

Annex:

Part 1-A – Evolutionary Algorithm

Main class

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.Scanner;
import java.util.*;

public class main {

	public static void main(String[] args) throws Exception {


			// specify site to get content
		URL site = new
URL("http://www.math.uwaterloo.ca/tsp/world/lu980.tsp");
		BufferedReader in = new BufferedReader(
		new InputStreamReader(site.openStream()));
		FileWriter fwriter0 =new FileWriter("output.txt",true);
			BufferedWriter outputFile0 = new BufferedWriter(fwriter0);
		String inputLine;

		// Array-list of Array-list that will contain all cities with
Latitude & Longitude
		ArrayList<ArrayList<Double>> Citylist = new
ArrayList<ArrayList<Double>>();

		//read file
		while ((inputLine = in.readLine()) != null)

			//read lines that start with a number
			if (Character.isDigit(inputLine.charAt(0)))
			{

			// split lines to number - latitude - longitude (by space)
			String[] splited = inputLine.split("\\s+");
			double d0 = Double.parseDouble(splited[0]);
			double d1 = Double.parseDouble(splited[1]);
			double d2 = Double.parseDouble(splited[2]);

			// put content Latitude and longitude to buffer Array-list
			ArrayList<Double> city = new ArrayList<Double>();
			city.add(d1); city.add(d2);

			//Eliminate redundant Cities
			//check of main Array-list of Array-list contains buffer
Array-list and add if not
			if (!Citylist.contains(city))
			{
```

```
                            Citylist.add(city);
                      }
                      }
                in.close();

                // call Genetic Algorithm class
                genetic_algorithim ga= new
genetic_algorithim(Citylist,outputFile0);
                outputFile0.close();
          }

}
```

Genetic_algorithim class

```
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

public class genetic_algorithim {

      // population initialization
      int[][] population;

      // population size initialization
      int popsize = 500;

      // fitness of population initialization
      int [] Popfitness;

      // Max number of generations
      int Maxgen =1000;

      //tournament selection selection size
      int tourSize = (int) Math.ceil(popsize/100)+1;

      public genetic_algorithim(ArrayList<ArrayList<Double>> Cities,
BufferedWriter outputFile1) throws IOException
      {
            PrintWriter writer = new PrintWriter("bestfitness.txt", "UTF-8");
            // printing of all Cities and their corresponding latitude and
longitude
            System.out.println("My Cities");
            for (int i=0; i<Cities.size();i++)
        {
                  outputFile1.newLine();
                  outputFile1.write(i+1 + ": Latitude --> " +
Cities.get(i).get(0) + " Longitude --> " + Cities.get(i).get(1));
                  outputFile1.newLine();
            System.out.println(i+1 + ": Latitude --> " + Cities.get(i).get(0)
+ " Longitude --> " + Cities.get(i).get(1));
        }

            // setting population of 5 solutions with each containing
cities.size() elements
            population = new int[popsize][Cities.size()];

            // setting fitness of population array to 5 (matches number of
solutions
            Popfitness = new int[popsize];
            outputFile1.newLine();
            outputFile1.newLine();
            System.out.println();
            System.out.println();

            //creating a population
            outputFile1.newLine();
            outputFile1.write("My Population");
            outputFile1.newLine();
            System.out.println("My Population");
            for (int i=0; i<popsize; i++)
            {
                  //each solution in population will have Cities.size() with
distinct random elements from 1 to excluding Cities.size()
                  population[i] = new Random().ints(1,
Cities.size()+1).distinct().limit(Cities.size()).toArray();
                  // Call individual Class
                  outputFile1.newLine();
                  outputFile1.write("Population:" + i + "--->");
                  outputFile1.newLine();
                  System.out.println("Population:" + i + "--->");
                  Individual idv = new Individual(Cities, population[i],
outputFile1);
                  //populate population fitness array
                  Popfitness[i] =idv.GetFitness(Cities, population[i]);
            }

            outputFile1.newLine();
            outputFile1.newLine();
            outputFile1.write("Fitness Array is given:");
            outputFile1.newLine();
            outputFile1.newLine();
            outputFile1.write(Arrays.toString(Popfitness));
            outputFile1.newLine();
            outputFile1.newLine();
            outputFile1.newLine();
            outputFile1.newLine();
            System.out.println();
            System.out.println("Fitness Array is given:");
            System.out.println(Arrays.toString(Popfitness));
            System.out.println();
            System.out.println();
            System.out.println();

            // initialize counter that keeps record of number of generations
without improvement
```

```
            int NoImprovCounter=0;


            //Initialize array with tournament size

            int [] tourSel = new int [tourSize];

            int count = 1;
            for (int i =0; i<Maxgen && NoImprovCounter <30 ; i++ )
            {
                  //Randomize the indices from which we will select our
solutions from the population (selection)
                  tourSel = new Random().ints(0,
popsize).distinct().limit(tourSize).toArray();
                  //Print selected solutions and their corresponding fitness
                  outputFile1.newLine();
                  outputFile1.write(count + ": selected solutions and their
corresponding fitness");
                  outputFile1.newLine();
                  System.out.println(count + ": selected solutions and their
corresponding fitness");
                  for (int g=0; g<tourSize; g++)
                  {
                        Individual idv = new Individual(Cities,
population[tourSel[g]], outputFile1);
                  }
                  //compare fitness of both selections, get most fit
                  int FittestParent = 0;
                  for (int k =0; k<tourSize; k++)
                  {
                        if
(Popfitness[tourSel[k]]<Popfitness[tourSel[FittestParent]])
                        {
                              FittestParent = k;
                        }
                  }
                  outputFile1.newLine();
                  outputFile1.write("Best solution from this selection is:
");

                  outputFile1.newLine();
                  System.out.println("Best solution from this selection is:
");

                  Individual idv2 = new Individual(Cities,
population[tourSel[FittestParent]], outputFile1);

                  outputFile1.newLine();
                  outputFile1.newLine();
                  outputFile1.newLine();
                  outputFile1.newLine();
                  System.out.println();
                  System.out.println();
                  System.out.println();

                  //mutate Fittest and check for improvement
                  int [] MutatedChild = new int[Cities.size()];
                  MutatedChild = idv2.GetMutation(Cities,
population[tourSel[FittestParent]], outputFile1);
```

```
                    int ChildFitness = idv2.GetFitness(Cities, MutatedChild);

                    //Check If parnet fitness is greater than child fitness
                    int minPopFitness = GetWorstFitnessFromPop ();
                    if(Popfitness[tourSel[FittestParent]]<ChildFitness)
                    {
                          // increment no improvment counter
                          NoImprovCounter++;
                    }
                    else
                    {
                          NoImprovCounter=0;
                          //update worst in population with child
                          population[minPopFitness] = MutatedChild;
                          //update weakest population fitness with child
fitness
                          Popfitness[minPopFitness] = ChildFitness;
                    }


                    outputFile1.newLine();
                    outputFile1.write("no improvment counter = " +
NoImprovCounter);
                    outputFile1.newLine();
                    outputFile1.newLine();

      outputFile1.write("*****************************************************
**");
                    outputFile1.newLine();
                    outputFile1.newLine();
                    outputFile1.write("best Fitness in Pop: " +
Popfitness[GetBestFitnessFromPop ()]);
                    outputFile1.newLine();
                    outputFile1.newLine();

      outputFile1.write("*****************************************************
**");
                    outputFile1.newLine();
                    outputFile1.newLine();
                    outputFile1.newLine();
                    System.out.println("no improvment counter = " +
NoImprovCounter);

      System.out.println("*****************************************************
***");
                    System.out.println("best Fitness in Pop: " +
Popfitness[GetBestFitnessFromPop ()]);
                    writer.println(Popfitness[GetBestFitnessFromPop ()]);

      System.out.println("*****************************************************
***");
                    System.out.println();
                    System.out.println();

                    count++;
              }
              writer.close();
```

```
        }


        // function to get index of worst fitness in population
        public int GetWorstFitnessFromPop ()
        {
                int index = 0;
                for(int i=0; i<popsize; i++ )
                {
                        if (Popfitness[i]>Popfitness[index])
                        {
                                index = i;
                        }
                }
                return index;
        }

        public int GetBestFitnessFromPop ()
        {
                int index = 0;
                for(int i=0; i<popsize; i++ )
                {
                        if (Popfitness[i]<Popfitness[index])
                        {
                                index = i;
                        }
                }
                return index;
        }
}
```

Individual class

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;

public class Individual {

        public Individual(ArrayList<ArrayList<Double>> Cities, int[]
indiv,BufferedWriter outputFile2) throws IOException
        {

                //print Elements of solution
                outputFile2.newLine();
                outputFile2.write(Arrays.toString(indiv));
                outputFile2.newLine();
                System.out.println(Arrays.toString(indiv));

                //print fitness of solution
                outputFile2.newLine();
                outputFile2.write("Fitness: " + GetFitness(Cities,indiv));
                outputFile2.newLine();
```

```
            System.out.println("Fitness: " + GetFitness(Cities,indiv));
      }

      public void Print(ArrayList<ArrayList<Double>> Cities, int[] indiv,
BufferedWriter outputFile2) throws IOException
      {
                        //print Elements of solution
                        outputFile2.newLine();
                        outputFile2.write(Arrays.toString(indiv));
                        outputFile2.newLine();
                        System.out.println(Arrays.toString(indiv));

                        //print fitness of solution
                        outputFile2.newLine();
                        outputFile2.write("Fitness: " +
GetFitness(Cities,indiv));
                        outputFile2.newLine();
                        System.out.println("Fitness: " +
GetFitness(Cities,indiv));
      }

      public int GetFitness(ArrayList<ArrayList<Double>> Cities, int[] indiv)
      {
            int F = 0;
            for (int i = 0; i<Cities.size(); i++)
            {
            double Lat_i, Lat_j;
            double Long_i, Long_j;
            double q1, q2, q3, q4, q5;
            double distance;
            int distance_int;
            // From each individual element subtracts 1 to match index of
city
            // individual span is 1-->634 but cities are 0-->633 index
            Lat_i = (Math.PI * Cities.get(indiv[i]-1).get(0))/180;
            Long_i = (Math.PI * Cities.get(indiv[i]-1).get(1))/180;
            if (i+1 < Cities.size())
            {
            Lat_j = (Math.PI * Cities.get(indiv[i+1]-1).get(0))/180;
            Long_j = (Math.PI * Cities.get(indiv[i+1]-1).get(1))/180;
            }
            else
            {
                  Lat_j = (Math.PI * Cities.get(indiv[0]-1).get(0))/180;
                  Long_j = (Math.PI * Cities.get(indiv[0]-1).get(1))/180;
            }
            q1 = Math.cos(Lat_j) * Math.sin(Long_i - Long_j);
            q3 = Math.sin((Long_i - Long_j)/2.0);
            q4 = Math.cos((Long_i - Long_j)/2.0);
            q2 = Math.sin(Lat_i+Lat_j) * q3 * q3 - Math.sin(Lat_i - Lat_j)
*q4 *q4;
            q5 = Math.cos(Lat_i-Lat_j) * q4 * q4 - Math.cos(Lat_i + Lat_j)
*q3 *q3;
            distance = 6378388 * Math.atan2(Math.sqrt(q1 *q1 + q2 *q2), q5) +
1.0;
            distance_int = (int)distance;
            F = F + distance_int;
            }
            return Math.abs(F);
      }

      public int[] GetMutation(ArrayList<ArrayList<Double>> Cities, int[]
indiv, BufferedWriter outputFile2) throws IOException
      {
            int [] MutatedIndiv = new int[Cities.size()];
            MutatedIndiv = indiv;
            Random random = new Random();
            //k-inversion mutation
            //chose random starting position
            int Start = random.nextInt(Cities.size()-1) + 1;
            //chose random k value where min value is 3
            int k = random.nextInt(Cities.size() - 3) + 3;
            outputFile2.newLine();
            outputFile2.write("using " + k + "-inversion mutation starting at
index:" + Start );
            outputFile2.newLine();
            outputFile2.newLine();
            outputFile2.write("parent: ");
            outputFile2.newLine();
            System.out.println("using " + k + "-inversion mutation starting
at index:" + Start );
            System.out.println("parent: ");
            Print(Cities, indiv, outputFile2);
            int [] temp = new int[k];
            int zeroindex = -1;
            for(int i =0; i<k;i++)
            {
                  if (Start+i<Cities.size())
                  {
                        temp[i]=indiv[Start+i];
                  }
                  else
                  {
                        temp[i]=indiv[zeroindex+1];
                        zeroindex++;
                  }
            }
            outputFile2.newLine();
            outputFile2.write("array to invert: ");
            outputFile2.newLine();
            outputFile2.newLine();
            outputFile2.write(Arrays.toString(temp));
            outputFile2.newLine();
            System.out.println("array to invert: ");
            System.out.println(Arrays.toString(temp));
            int k_copy = k;
            int zeroindex2 = -1;
            for(int i =0; i<k; i++)
            {
                  if(Start+i<Cities.size())
                  {
                        MutatedIndiv[Start+i] = temp[k_copy-1];
                  }
                  else
```

```
                        {
                              MutatedIndiv[zeroindex2+1] = temp[k_copy-1];
                              zeroindex2++;
                        }
                        k_copy--;
                  }
                  outputFile2.newLine();
                  outputFile2.write("child: ");
                  outputFile2.newLine();
                  System.out.println("child: ");
                  Print(Cities, MutatedIndiv, outputFile2);
                  return MutatedIndiv;
            }
      }
```

Part 1-B – Multilayer Perceptron

Main class

```
import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.Random;


public class Main {


      public static void main(String[] args) {

            // TODO Auto-generated method stub


            Random rand = new Random();

            ArrayList<ArrayList<Double>> DataSet = new
ArrayList<ArrayList<Double>>();

            double[][] WeightsFromInput;

            double[][] WeightsToOutput;

            WeightsFromInput = new double[10][5];

            WeightsToOutput = new double[10][5];

            double [] ErrorRate = new double[10];

            double [] ErrorRate2 = new double[10];

            double [] Mean_SE = new double[10];


            // Initializing 10 random inputs for Cubic function

            double [] In = new double[10];

            In = new Random().doubles(-100,
101).distinct().limit(10).toArray();
```

```
            // creating a Dataset of 10 random inputs for Cubic function and 
their corresponding outputs

            for(int i=0;i<10; i++)

            {

                  ArrayList<Double> Set = new ArrayList<Double>();

                  double out = (In[i]*In[i]*In[i]);

                  Set.add(In[i]); Set.add(out);

            DataSet.add(Set);

            System.out.println("input: " + DataSet.get(i).get(0) + " 
output: " + DataSet.get(i).get(1));

            }

            System.out.println("\n\n");



            //initializing 10 sets of weights randomly from input to hidden 
layer

            //initializing 10 sets of weights randomly from hidden layer to 
output

            for (int i =0; i<10;i++)

            {

                  for(int j =0; j<5;j++)

                  {

                  WeightsFromInput[i][j] = -5 + (5 - (-5)) * 
rand.nextDouble();

                  WeightsToOutput[i][j] = -5 + (5 - (-5)) * 
rand.nextDouble();

                  }

                  System.out.println("WeightsFromInput:" + i + " " + 
Arrays.toString(WeightsFromInput[i]));

                  System.out.println("WeightsToOutput:"  + i + " " + 
Arrays.toString(WeightsToOutput[i]) + "\n");

            }
            //performing forward propagation for all 10 points (with
different weights for each time)

            // topology = 1 input, hidden layer= 1x5, output = 1

            System.out.println("\n");

            System.out.println("performing 10 forward propagation for all 10
points (with different weights for each time)");

            for(int j =0;j<10;j++){

                  System.out.println("****************************SET of
weights = " + (j+1) + "*********************************** -----> ");

            for(int i =0;i<10;i++)

            {

                  double hiddenNeuron1_Value;

                  double hiddenNeuron2_Value;

                  double hiddenNeuron3_Value;

                  double hiddenNeuron4_Value;

                  double hiddenNeuron5_Value;


                  double hiddenNeuron1_Sigmoid;

                  double hiddenNeuron2_Sigmoid;

                  double hiddenNeuron3_Sigmoid;

                  double hiddenNeuron4_Sigmoid;

                  double hiddenNeuron5_Sigmoid;


                  double output_Value;

                  double output_Sigmoid;


                  System.out.println("********************************** ----
-> " + (i+1));
```

```
                  hiddenNeuron1_Value = DataSet.get(i).get(0)*
WeightsFromInput[j][0];

                  System.out.println("hiddenNeuron1_Value: " +
hiddenNeuron1_Value);

                  hiddenNeuron2_Value = DataSet.get(i).get(0)*
WeightsFromInput[j][1];

                  System.out.println("hiddenNeuron2_Value: " +
hiddenNeuron2_Value);

                  hiddenNeuron3_Value = DataSet.get(i).get(0)*
WeightsFromInput[j][2];

                  System.out.println("hiddenNeuron3_Value: " +
hiddenNeuron3_Value);

                  hiddenNeuron4_Value = DataSet.get(i).get(0)*
WeightsFromInput[j][3];

                  System.out.println("hiddenNeuron4_Value: " +
hiddenNeuron4_Value);

                  hiddenNeuron5_Value = DataSet.get(i).get(0)*
WeightsFromInput[j][4];

                  System.out.println("hiddenNeuron5_Value: " +
hiddenNeuron5_Value);



                  hiddenNeuron1_Sigmoid = Sigmoid(hiddenNeuron1_Value);

                  System.out.println("hiddenNeuron1_Sigmoid: " +
hiddenNeuron1_Sigmoid);

                  hiddenNeuron2_Sigmoid = Sigmoid(hiddenNeuron2_Value);

                  System.out.println("hiddenNeuron2_Sigmoid: " +
hiddenNeuron2_Sigmoid);

                  hiddenNeuron3_Sigmoid = Sigmoid(hiddenNeuron3_Value);

                  System.out.println("hiddenNeuron3_Sigmoid: " +
hiddenNeuron3_Sigmoid);

                  hiddenNeuron4_Sigmoid = Sigmoid(hiddenNeuron4_Value);

                  System.out.println("hiddenNeuron4_Sigmoid: " +
hiddenNeuron4_Sigmoid);

                  hiddenNeuron5_Sigmoid = Sigmoid(hiddenNeuron5_Value);
                  System.out.println("hiddenNeuron5_Sigmoid: " +
hiddenNeuron5_Sigmoid);




                  output_Value = (hiddenNeuron1_Sigmoid *
WeightsToOutput[j][0]) + (hiddenNeuron2_Sigmoid * WeightsToOutput[j][1])
+(hiddenNeuron3_Sigmoid * WeightsToOutput[j][2]) +(hiddenNeuron4_Sigmoid *
WeightsToOutput[j][3]) +(hiddenNeuron5_Sigmoid * WeightsToOutput[j][4]);

                  System.out.println("output_Value: " + output_Value);

                  output_Sigmoid = Sigmoid(output_Value);

                  System.out.println("output_Sigmoid: " + output_Sigmoid);

                  System.out.println("output_Actual: " +
DataSet.get(i).get(1));


                  //calculating error from each point

                  ErrorRate[i] = DataSet.get(i).get(1) - output_Sigmoid ;

                  System.out.println("ErrorRate["+i+"]: " + ErrorRate[i]);

                  System.out.println("\n");

            }

                  Mean_SE[j] = MeanSquaredError(ErrorRate);

                  System.out.println("MSE for last weighs = " + Mean_SE[j]);

            }

            System.out.println("\n\n");


            // getting mean squared error for first 10 points

            System.out.println("Mean squared error for last 10 attempts with
different weights= " + Arrays.toString(Mean_SE));


            //getting index of smallest error to use best weights

            int index=0;

            for(int i=0;i<10;i++)
            {

                  // performing absolute because error could be negative, we
want closest to zero

                  if (Math.abs(Mean_SE[i])< Math.abs(Mean_SE[index]))

                  {

                        index = i;

                  }

            }

            System.out.println("\n\n");

            System.out.println("best weights where for index: " + index);



                        //repeating procedure (with same best weights for
each time)

                        // topology = 1 input, hidden layer= 1x5, output = 1

            System.out.println("\n");

            System.out.println("repeating procedure (with same best weights
for each time)");

                        for(int i =0;i<10;i++)

                        {

                              double hiddenNeuron1_Value;

                              double hiddenNeuron2_Value;

                              double hiddenNeuron3_Value;

                              double hiddenNeuron4_Value;

                              double hiddenNeuron5_Value;



                              double hiddenNeuron1_Sigmoid;

                              double hiddenNeuron2_Sigmoid;

                              double hiddenNeuron3_Sigmoid;

                              double hiddenNeuron4_Sigmoid;
                              double hiddenNeuron5_Sigmoid;


                              double output_Value;

                              double output_Sigmoid;



      System.out.println("********************************** -----> " +
(i+1));


                              hiddenNeuron1_Value = DataSet.get(i).get(0)*
WeightsFromInput[index][0];

                              System.out.println("hiddenNeuron1_Value: " +
hiddenNeuron1_Value);

                              hiddenNeuron2_Value = DataSet.get(i).get(0)*
WeightsFromInput[index][1];

                              System.out.println("hiddenNeuron2_Value: " +
hiddenNeuron2_Value);

                              hiddenNeuron3_Value = DataSet.get(i).get(0)*
WeightsFromInput[index][2];

                              System.out.println("hiddenNeuron3_Value: " +
hiddenNeuron3_Value);

                              hiddenNeuron4_Value = DataSet.get(i).get(0)*
WeightsFromInput[index][3];

                              System.out.println("hiddenNeuron4_Value: " +
hiddenNeuron4_Value);

                              hiddenNeuron5_Value = DataSet.get(i).get(0)*
WeightsFromInput[index][4];

                              System.out.println("hiddenNeuron5_Value: " +
hiddenNeuron5_Value);


                              hiddenNeuron1_Sigmoid =
Sigmoid(hiddenNeuron1_Value);

                              System.out.println("hiddenNeuron1_Sigmoid: " +
hiddenNeuron1_Sigmoid);
```

```
                              hiddenNeuron2_Sigmoid =
Sigmoid(hiddenNeuron2_Value);

                              System.out.println("hiddenNeuron2_Sigmoid: " +
hiddenNeuron2_Sigmoid);

                              hiddenNeuron3_Sigmoid =
Sigmoid(hiddenNeuron3_Value);

                              System.out.println("hiddenNeuron3_Sigmoid: " +
hiddenNeuron3_Sigmoid);

                              hiddenNeuron4_Sigmoid =
Sigmoid(hiddenNeuron4_Value);

                              System.out.println("hiddenNeuron4_Sigmoid: " +
hiddenNeuron4_Sigmoid);

                              hiddenNeuron5_Sigmoid =
Sigmoid(hiddenNeuron5_Value);

                              System.out.println("hiddenNeuron5_Sigmoid: " +
hiddenNeuron5_Sigmoid);




                              output_Value = (hiddenNeuron1_Sigmoid *
WeightsToOutput[index][0]) + (hiddenNeuron2_Sigmoid *
WeightsToOutput[index][1]) +(hiddenNeuron3_Sigmoid *
WeightsToOutput[index][2]) +(hiddenNeuron4_Sigmoid *
WeightsToOutput[index][3]) +(hiddenNeuron5_Sigmoid *
WeightsToOutput[index][4]);

                              System.out.println("output_Value: " +
output_Value);

                              output_Sigmoid = Sigmoid(output_Value);

                              System.out.println("output_Sigmoid: " +
output_Sigmoid);

                              System.out.println("output_Actual: " +
DataSet.get(i).get(1));



                              //calculating error from each point

                              ErrorRate2[i] = DataSet.get(i).get(1) -
output_Sigmoid ;
```

```
                              System.out.println("ErrorRate2["+i+"]: " +
ErrorRate2[i]);

                              System.out.println("\n");

                        }


                        System.out.println("\n\n");


                        // getting mean squared error for first 10 points

                        System.out.println("Mean squared error for last 10
attempts with same weights= " + MeanSquaredError(ErrorRate2));

      }


      // returns value of Sigmoid function

      public static double Sigmoid(double V)

      {

            return (1/(1+Math.pow((Math.E), (-V))));

      }


      public static double MeanSquaredError(double [] err)

      {

            double MSE = 0;

            for (int i=0; i<10; i++)

            {

                  MSE = MSE + err[i];

            }

            return MSE;

      }


}
```