Laboratory 9 – Week 10

## Hill Climbing - the Scales Problem

## 9.1 Introduction

**Firstly, this worksheet is one of the worksheets from which your laboratory worksheets portfolio of work will be assessed.**

It is **VERY** important that you have completed Worksheet 8 - Scales and Fitness Functions. You should work on laboratory 8 and complete it before you start this one (sheet 9).

This laboratory involves developing, running and experimenting with the Hill Climbing algorithm for solving the *Scales* problem.

## 9.2 Preliminaries

Familiarise yourself with the lecture entitled “9.1 Heuristic Search, Hill Climbing and Simulated Annealing”. Pay particular attention to the section on the Hill Climbing algorithm (the Random Mutation version - RMHC).

Create a project and associated class called Lab9. Within this laboratory you will be adding to the ScalesSolution class from laboratory 8. The aims of this laboratory are as follows:

1. Creating a small change method within the ScalesSolution class.
2. Creating a Hill Climbing method within the Lab9 class.
3. Reading the “1000 Primes.txt” file in.
4. Running a number of experiments.

## 9.3 Exercise 1: Small Change Method

Add the ScalesSolution class and the CS2004 class from laboratory 8 into your project. Within ScalesSolution create a method called SmallChange as follows:

**public** **void** SmallChange()

{

}

This method will change a random element of the scasol field/attribute. As in the lecture notes, we will generate a random integer (say *p*) between 0 and *n*-1 (n = scasol.length()). We will then change position *p* of scasol, i.e. it is a '1' we make it a '0' and if it is a '0' we set it to '1'.

However this might be a little less straight forward than expected since in Java, strings are considered almost constant, i.e. there is no method to change part of a string.

There are numerous ways in which we can do this. One option is as follows:

1. Create a random integer p that ranges between 0 and n-1.
2. Create an empty new string say x.
3. Copy from elements 0 to p-1 from scasol to x.
4. Copy the changed version of position p of string scasol to x.
5. Copy from p+1 to n-1 of scasol to x.
6. Set scasol to be x.

Alternatively the method getChars within the String class might be of use.

Test you code on a set of logical examples, e.g. a string equal to “11111” and “00000”, and then some random strings. For example:

**public** **static** **void** main(String args[])

{

ScalesSolution s = **new** ScalesSolution("11111");

s.println();

s.SmallChange();

s.println();

}

You *might* get the following output:

11111

11011

## 9.4 Exercise 2: Hill Climbing Method

Before we start on the Hill Climbing method, we need to create a way of copying an instance of the ScalesSolution class. The simplest way to do this is to add the following method to ScalesSolution:

**public** String GetSol()

{

**return**(scasol);

}

This method simply returns the string representation that the instance of the class contains. Try the following code:

ScalesSolution s1 = **new** ScalesSolution(10);

s1.println();

ScalesSolution s2 = **new** ScalesSolution(s1.GetSol());

s2.println();

You should get the same string being displayed twice. This is how we are going to copy a solution.

We are now going to write the code of the Random Mutation Hill Climbing Algorithm (RMHC) within the Lab9 class. Create a method as follows:

**public** **static** ScalesSolution RMHC(ArrayList<Double> weights,**int** n,**int** iter)

{

ScalesSolution sol = **new** ScalesSolution(n);

**return**(sol);

}

This method takes in an ArrayList of weights and a parameter n of what sized solution we are searching for and will return a ScalesSolution representation of the best solution (as found by RMHC) for solving the *Scales* problem applied to the first n weights of the ArrayList weights. The hill climbing algorithm will run for iter iterations.

We now need to implement the RMHC as in the lecture notes:

1. We need to add a For loop that iterates for the specified number of iterations.
2. We need to create an initial random solution of size n.
3. We need to evaluate the fitness of our current solution within the loop.
4. We need to copy the current solution (say oldsol).
5. We make a small change to the current solution and evaluate the fitness to another variable.
6. If the new fitness is worse than the old, we copy oldsol back to being our current solution.
7. After the For loop has completed we return the current solution.

It is often worth displaying within the loop, the current iteration and current fitness. This allows us to verify that the fitness is decreasing or remaining the same. If it goes up then we have made a mistake in our code.

Verify that the algorithm works on some small problems, e.g. the weights 1, 2, 3, 4 and 10 as in the lecture.

## 9.5 Exercise 3: Reading in Data and Running Some Experiments

Once you have verified that the RMHC algorithm works read in the “1000 Primes.txt” file as we did in laboratory 8 and then run the algorithm for 1,000 iterations. Once this is working, run the algorithm ten times and record the best fitness for each run. Do you get the same average as in the lecture notes? Time how long each run takes for 1,000 iterations.

Now run the algorithm for as many times as possible within 5 minutes, running each repeat for 10,000 iterations. Does a run for 10,000 iterations take ten times longer than a run for 1,000? What average result do you get? How does it compare with the results in the lecture notes?

Finally do you think using a String for the representation was a good idea? If not, what would have been better?