Laboratory 11 – Week 18

## A Simple Genetic Algorithm Applied to the Scales Problem

## 11.1 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 Laboratory Sheet 9 - Hill Climbing - the Scales Problem. You should work on laboratory 9 and complete it before you start this one (sheet 11).

This laboratory involves the use and application of a simple Genetic Algorithm (GA) to solve the Scales Problem. You will be running a number of experiments and plotting convergence graphs.

## 11.2 Preliminaries

Familiarise yourself with the lecture entitled “12.1 An Introduction to Genetic Algorithms”. Make sure you understand how Genetic Algorithms work.

Because Genetic Algorithms are stochastic (involve random number generators) we will often find that even with two identical runs (same parameters) a different number of individual are created in each generation. This is vastly exacerbated if the parameters are highly different. Thus we often run a Genetic Algorithm for a fixed number of fitness function calls (where most of the computational complexity is often accounted for), so that we can cross compare experiments and other methods. If we fix the number of fitness function calls (*fc*), then the Genetic Algorithm will run for the number of generations that equals or just exceeds *fc* fitness function calls.

## 11.3 Exercise 1: The Simple Genetic Algorithm

In Appendix A there are four classes that are to be extracted:

|  |  |
| --- | --- |
| **Class** | **Function** |
| Lab11 | The main class for running the experiments. |
| SimpleGeneticAlgorithm | The Genetic Algorithm Class. You will need to understand, use and modify this class. |
| ScalesChrome | The Chromosome class for the Scales problem. You will need to understand and use this class. |
| CompareScalesChrome | This is used by the sort method to sort the population of ScalesChromosomes. You need to use this class. |

The SimpleGeneticAlgorithm class performs the Genetic Algorithm. It creates the initial random population containing a number of ScalesChrome objects, it iterates for the specified number of generations or fitness function calls, and performs the genetic operators of **Crossover**, **Mutation** and **Survival**. The best individual from the final population is then returned as the solution to the specified size of Scales problem.

Create a project in Eclipse called Lab11 and copy the classes into the project. Examine the main method of class Lab11.

The constructor for the SimpleGeneticAlgorithm class has the following format:

**public** SimpleGeneticAlgorithm(**int** ps,**int** gs,**int** nb,**double** mr,**double** cr)

The parameters are defined as follows:

|  |  |
| --- | --- |
| **Parameter** | **Function** |
| ps | The population size for the genetic algorithm. |
| gs | The number of generations to run for. |
| nb | The number of weights (genes) we are solving the Scales problem for. |
| mr | The Mutation rate. |
| cr | The Crossover rate |

We are going to run the Scales problem for 1000 weights (as in the example code). We are also going to run the Genetic Algorithm for 10,000 fitness function calls rather than the specified number of generations. This is so that we can compare the results for a number of different experiments. If we change any of the parameters, then we will create differing number of individuals between experiments, and thus it may appear that one run was better than another, but it may have been the case that the best result evaluated twice as many fitness calls that the other. Thus we have two versions of the RunSGA method. One version runs the GA for the specified number of generations, and the other runs it for a number of fitness function calls.

Read through the comments and look at the structure of the classes and then run the program. What do you think of the quality of the results when compared to your Hill Climbing results?

## 11.4 Exercise 2: Optimising the Simple Genetic Algorithm

You should have found that the each run of the GA produces absolutely useless (very poor) results. This is because the parameters are not set correctly. Look at the lecture notes and choose more sensible values for the population size, the crossover rate and the mutation rate. You should be aiming to be able to run the GA for 10 times and get a final solution of 1 each time!!! Turn on the reporting flag (change the second parameter in the RunSGA call to true), run the GA with your optimal parameters once, and then plot the average fitness and best fitness against generation number as in the diagram below.

## 11.5 Implementing Uniform Crossover

The last part of this worksheet is to implement Uniform Crossover and evaluate it.

Locate the Uniform Crossover method in the SimpleGeneticAlgorithm class:

**private** **void** DoUniformCrossOver(**int** p1id,**int** p2id)

You will need to complete this method. To implement Uniform Crossover you decide (50/50 chance) which child get each gene of parent one. The other child gets the corresponding gene of parent two.

For example if child one was to get gene seven of parent one then child two will get gene seven of parent two. The resulting two children should be the same size (number of genes) as their parents.

You will also need to amend method CrossOver as described in the code in order to run Uniform Crossover. Test your code on a much smaller set of weights to see if it works. Use the print and println methods within the Scales Chromosome if needed.

Once you are sure that you have Uniform Crossover is working correctly, test it against One Point Crossover (using the full 1000 weights) and draw graphs showing their performance. The exact set of experiments you conduct is up to you.

## 11.6 Appendix A

The following class contains the code you will need for this exercise sheet:

|  |
| --- |
|  |

**NOTE: You will also need your copy of CS2004.java since these classes use random numbers.**