Genetic algorithms

For this assessment, you are asked to optimise two functions (problems) using genetic algorithms. The assessment will be marked by machine. It is therefore very important that you follow the instructions given to you very carefully. Failure to do so will result in substantial loss of marks, even zero.

To help you complete the assessment you are given 2 pieces of software: (1) A shell program code called `Example.java` (2) A compiled class file `Assess.class` that implements the problems. It contains 2 methods, namely `double Assess.getTest1(double[])` and `double[] Assess.getTest2(boolean[])`. These functions accept candidate solutions for problems 1 and 2 respectively and return the fitness of the proposed solutions. The example code in `Example.java` demonstrates how to access them. Look at the code carefully to understand how it works. Additionally, it also contains a function `checkIn(...)`. You will need to call this function at the end of your assignment in order to indicate the solutions to the two problems given to you. See below for more details.

Problem 1

The first problem is a black box function that takes 20 real-valued numbers as input. The function has a unique minimum fitness of 0 for a specific input. Your task is to write a genetic algorithm to find this minimum value. You can safely assume that the minimum can be found for input values between -5.0 and +5.0 for all variables/dials. So, you do not need to search smaller or larger values than that.

Problem 2

The second problem is a version of the well-known knapsack problem. The idea is that you want to pack a bag. You have 100 items that you would like to take with you. Each item has a utility value and a weight (assumed to be expressed in kg). You want to maximise your utility (which is the sum of the utilities of all the items you take with you) but you are not allowed to pack more than 500 kgs of weight altogether. This means that you need to choose the items that maximise your total utility while staying within the weight limit.

The knapsack problem is already implemented for you, so you do not need to worry about the list of items and their utilities of weight. The function

```
double[] Assess.getTest2(boolean[])
```

accepts a boolean array of length 100 as a candidate solution. The item you want to take with you corresponds to the index value and the boolean at the relevant index encodes whether or not you take the item with you. So, for example, if the array is given by:

```
true, true, false, true, false, false, ....
```

Then this means that you take with you items 0,1,3,...

The function `getTest2(boolean[])` returns an array of doubles. The first entry (index 0) is the weight of your combination and the second (index 1) is the sum of the utilities. If unsure, check the example code where it is shown on how to use the function. Before starting to implement your GA, play a bit with the example code to make sure you understand how fitness is returned.

Your task is to find the combination of items to pack that maximises utility while staying within the weight constraints.

Submission

Your task is to write two genetic algorithms that solve the above described problems. Once you found a solution for the two problems you need to check them in, using the method

```
void Assess.checkIn(...)
```

This function takes your name, your login, the best solution for problem 1 and the best solution for problem 2. Please see the example code in order to understand how to do this. Once you have checked in, you will get the message on the command line that you are ready to submit. Before submission, make sure that your code always outputs the check in message correctly at the end of the execution.

The `checkIn` function will be used for marking. If you do not check in your results, then your solutions will not be assessed (i.e. you get no marks for it), so ensure that you check in your results.

When marking, the same problem will be used to assess your code, but it will be numerically different. You must therefore ensure not to `checkIn` a fixed solution, but the one that is generated by your GA at run-time.

You need to place your solutions in: `/proj/co528c/ga/xyz` (for Canterbury students) and `/proj/co528m/ga/xyz` (for Medway students) on raptor where xyz is to be replaced by your own personal login. Permissions have been set so that only xyz can access files in the directory xyz. You will lose write permission at 23:55 on the day of the deadline. In order to submit just make sure that the directory contains the code and **nothing else**.

For your submission, you must use the program `Example.java`. You can extend it and add classes as you like. It is important, however, that the program compiles on raptor like:

```
javac -cp . *.java
```

and that it can be executed from the command line on raptor using

```
java Example
```

Before submitting, make sure that this works on raptor. If you code cannot compile and your program called using these above command, then you will get 0 marks.

In addition to the .java file, I also give you a class file (`Assess.class`). You need to download this into the same directory as the .java, otherwise your program will not work.

Please ensure that your program is located in the top level of your folder, i.e. in `/proj/co528c/ga/xyz` or `/proj/co528m/ga/xyz` and not in a subfolder. This means that compilation and execution must work when in the directory `/proj/co528c/ga/xyz or /proj/co528m/ga/xyz`. Failure to follow this instruction will incur a penalty in marks.

Marking criteria

Your final submission will be assessed by compiling and running it on raptor using the above two commands. It is your responsibility to ensure before submission that they work and produce the desired result. **If this does not work, you will be awarded 0 marks.**

Non-compiling/non-running programs will get 0 marks.

Rules on use of APIs: `Example.java` includes, at the moment

`java.lang.Math`

and

`java.util.*`

These allow you to use mathematical functions in java and ArrayLists. You are not allowed to use any APIs beyond that. In particular, do not use any graphics or GUI. This is not necessary and not allowed.

- You will get up to 20 marks for a program that compiles, runs, displays functionality and completes within 60 seconds on raptor.
- You will get up to 30 marks for the first problem. The number of marks will depend on the quality of the solution. The closer the solution fitness to 0 the higher the mark. Note that you are asked for the perfect solution. Do not assume that a fitness of 0.00000000000001 gets full marks, but attempt to find the 0 fitness solution. That said, you will get partial marks for good solutions.
- You will get up to 30 marks for the second problem. The higher the utility (while staying within the weight constraints) the more marks. If your solution is above the weight limit (even by a little bit), it will not get any marks.
- Your program must completed within 60 seconds, for both tasks. You will get up to 20 marks for execution speed – therefore it is to your benefit to optimise the code to run as fast as possible while still meeting the quality of solutions. The time will be measured on raptor. Note that if you have a very slow or very fast machine at home, execution times can be substantially different. So, make sure that you check the run-time on raptor.
- If your code takes longer than 60 seconds, the execution will be terminated and your marks will be capped at 20.

Remarks on the class file

Please note that the Assess.class file is compiled on raptor using the Java version installed there. It may not work on other machines that have a different version of Java installed. It is therefore recommend that you work on raptor.

Deadline

The deadline for submission is available on the student data system. Check there!

Section 2.2.1.1 of Annex 9 of the Credit Framework states that, “Academic staff may not accept coursework submitted after the applicable deadline except in concessionary circumstances”.

A Frequently Asked Questions document on Plagiarism and Collaboration is available at: `www.cs.kent.ac.uk/teaching/student/assessment/plagiarism.local`. The work you submit must be your own, except where its original author is clearly referenced. Checks will be run on submitted work in an effort to identify possible plagiarism, and take disciplinary action against anyone found to have committed plagiarism. When you use other peoples’ material, you must clearly indicate the source of the material.

Acknowledgement: Assessment adapted by Dr Palani Ramaswamy from original version by Dr Dominique Chu