COMS30014 - AI

Genetic Algorithm Lab 1: The Simple GA

0. Prep

=======

Before the lab session, please download the Python GA code for this lab (ga-code-lab1.py) and get it running on your computer or a 2.11 machine.

- It is written in Python 3.
- It does not use any packages apart from "random" and "math" and "statistics" (which are standard Python packages).
- I've tried to write code that is easy to read, rather than efficient, compact or even overly "Pythonic"
- (If you want to adapt it to run faster, etc., that's fine as long as you understand it first)
- Ahead of the lab itself, if your code won't run, or something in the code doesn't make sense, raise a question on the unit's Discussion Forum.

The code implements a classic but very simple evolutionary algorithm, operating on a very simple example problem - sometimes called the "Weasel Program" (https://en.wikipedia.org/wiki/Weasel_program).

- It was first introduced by Richard Dawkins in his Blind Watchmaker book
- It's a thought experiment on how "random" evolutionary search can be surprisingly fast
- We are going to use it as a way of introducing the basic form of a genetic algorithm
- ..and as a way of thinking about slightly more complicated genetic algorithm ideas
- if you complete this lab and the next GA lab your ability to answer exam questions or complete the unit coursework will be improved

Once you have the code downloaded and running OK, feel free to read the following ahead of the lab. But please don't start the lab exercises until you're in the lab with the other students. :)

The Standard GA Set Up
====================

The code implements a very simple genetic algorithm hereafter referred to as 'the Standard GA Set Up':

```
1. *Initialise* a population of N individuals
2. *Assess* each individual's fitness
3. Repeat:
   (i) until we evolve a perfect individual, or
   (ii) until we complete G generations of evolution
4.    *Breed* N new offspring from the fitter parents
5.    *Mutate* the new offspring
6.    *Assess* each offspring's fitness
7. *Return* the best individual from the final generation
```

Each individual in the population is a fixed-length string of alphabetic characters. The fitness of an individual is equal to the number of its characters that match a target string: "methinks it is like a weasel".

To complete this lab you need to be able to run the code (or your own version of the code) and adapt it in some relatively simple ways to see what happens.

You will also benefit from being able to plug the data that your code generates into a utility that can make graphs like the one included below. I used Excel to make this graph just to show you that you don't need anything fancy. However, there are many other, better, graph plotting utilities and you are free to use whichever is your favourite.

Please read the note after the example graph. It explains the properties that a *good* graph needs to have. It's a very good idea to get into the habit of generating good graphs as good graphs will get you marks in courseworks and in your final year project, whereas bad graphs will lose you marks.

Before the lab, work out a quick and easy way to plot the output from the code into a graph. One option would be to generate the graphs in Python using a library such as matplotlib or plotly.

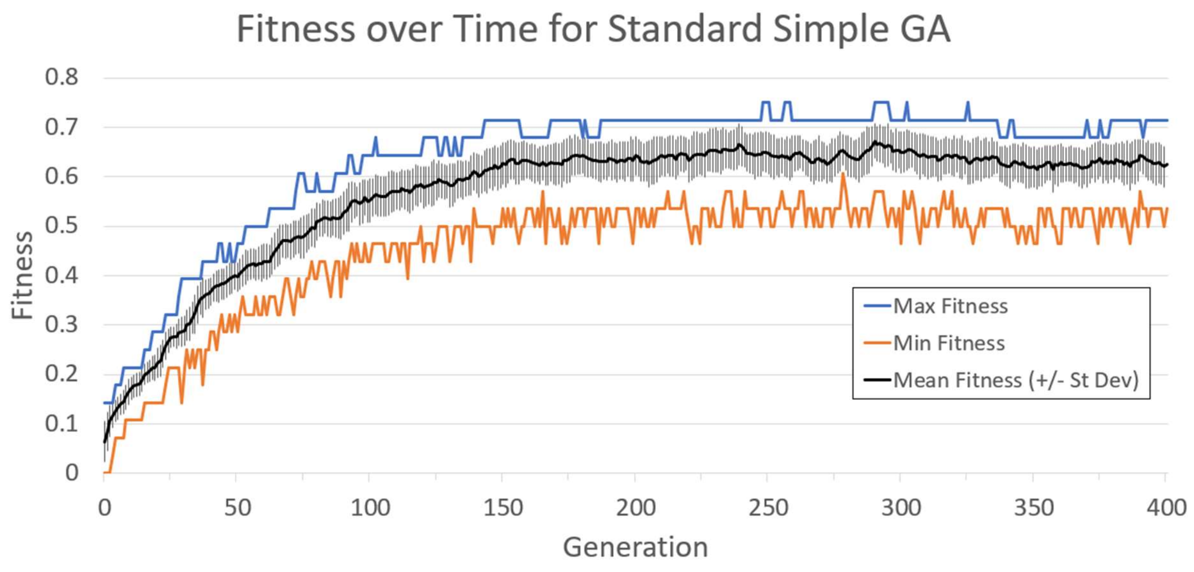

Figure 1: A plot of how the fitness of an evolving population changes during the first 400 generations of evolution for the "methinks it is like a weasel" problem. Evolutionary progress slows before stalling around generation 150 at a fitness of around 0.7. Genetic Algorithm Parameters: population size=100, mutation rate=1/28, tournament size=2, no Crossover, no Elitism, population is initialised to random strings, genetic alphabet=26 lower case letters plus the space character.

Note: (i) The graph has a short title, (ii) both axes are labelled, (iii) the typeface is large enough to be clearly legible, (iv) the lines are thick enough to be clearly legible, (v) the lines are formatted such that they are distinct from each other, (vi) the legend is clear, (vii) the caption is detailed and explains what we are looking at.

All of these are necessary features of a good graph – save that sometimes you can do without (i).

During the lab, check out the graphs that other people around you generate. Are they better than yours? Are they using an approach that is better than yours? Use this opportunity to crowd-source graph plotting utilities/libraries and techniques that you like and do a good job…