# CSG2341 Intelligent Systems

## Workshop 6: An introduction to genetic algorithms

**Related Objectives from the Unit outline:**

- Describe the concept of computational intelligence and its associated techniques and algorithms.
- Identify appropriate intelligent system solutions for a range of computational intelligence tasks.
- Demonstrate the ability to apply computational intelligence techniques to a range of tasks normally considered to require computational intelligence.

**Learning Outcomes:**

After completing this workshop, you should be able to describe the operation of an evolutionary algorithm for optimisation, demonstrate an understanding of the resources required to implement a problem solution based on genetic algorithms, and describe and analyse the steps in designing an evolutionary algorithm to solve an optimisation problem.

---

**Background:**

In this workshop, you will experiment with using a Genetic Algorithm to solve a search problem, and see the effect of changing Genetic Algorithm parameters such as elitism, mutation rate, and crossover rate.

**Task:**

The problem is to “find” a certain 26 letter message, by trying possible solutions, and being told how many letters (but not which ones) are correct.

To give an example using a shorter message : suppose the hidden message is “Hit the highway” and the trial solution is “Give me a break”. The matching letters are underlined – there are 4 of them. For technical reasons, this will be given a “fitness score” of 1+4, i.e. 5.

To solve this problem using a Genetic Algorithm, we first have to decide on a way to represent solutions as a “genome”. We are going to use strings made out of letters from

“abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ”

We also need to design a mutation operator, and decide how to carry out crossover.

A solution using the IS library has been created for you. Download it from Blackboard. Unzip it. You should get a folder called GAWorkshop, containing a source folder called src, which in turn contains a folder StringSearch. Create a Netbeans project from this with GAWorkshop as the project folder and src as the source folder. Also link in the IS library (is.jar) as a Library.

You should be able to build the project and run the class EvolveString. You'll see a window like this:


In the top two panels there are two randomly generated strings, old 1 and old 2. Now click on "mutate 1". You might see something like this:


Notice that new 1 has changed (probably – if not, click again). The mutation operator has been applied to old 1 to get new 1. Likewise if you click on "crossover", old 1 will be crossed with old 2, and the new genomes will be seen as new 1 and new 2.

Using this as a tool, and by examining the code in StringSearchEvolvable, and the following questions:

**QUESTION 1**

*Describe how the mutation operator works in this program.*

**QUESTION 2**

*Describe how the crossover operator works in this program. Is this one-point or two-point crossover?*

**QUESTION 3**

*What selection method does this program use in the genetic algorithm?*

If you examine the code in EvolveString, you will find this

```
GeneticAlgorithm ga = new GeneticAlgorithm
  (
    StringSearchEvolvable.class, // this is the class that represents a genome
    constructorParams, // these are used to construct genomes (passed to the constructor)
    POP, // population size
    random, // a random number generator to use
    mutationProb, // probability of mutating a gene
    crossoverProb, // probability of crossing two parents
    eval, // the fitness function
    new FitnessTerminator(eval.maxFitness()), // the termination criterion
    new StochasticUniformSelector(random), // the selector
    true // whether to use elitism : true for yes, false for no
  );
```

Here you can see some of the parameters that can be set when setting up a Genetic Algorithm.

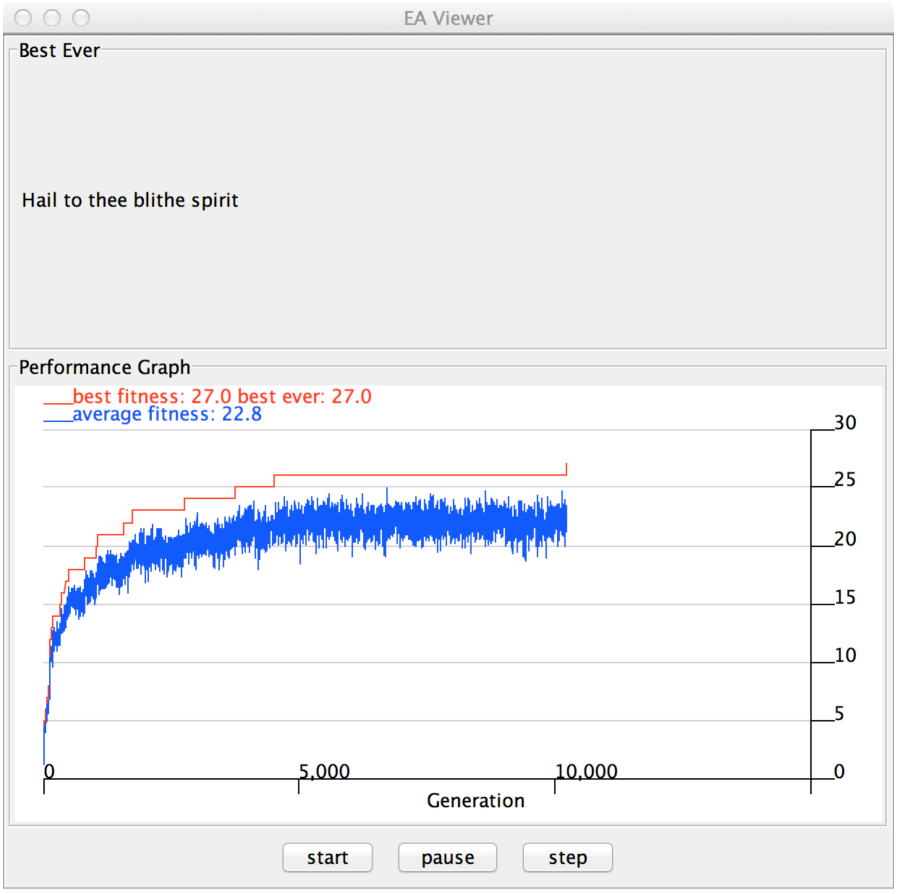
To begin with, mutation probability (the probability that a gene will be mutated when creating a child) is set at 1.0/26, crossover probability (the probability that two selected parent will be crossed to form a two children) is 0.6, and elitism is enabled. We are going to see the effect of changing these.

First, find this section of the code:

```
  // uncomment the line below to test out mutation and crossover operations
ga.test();
// uncomment the line below to run the GA via the GUI interface
//ga.display();
```

Comment out ga.text() and uncomment ga.display(), and run EvolveString. You should see something like


Click on the “start” button. You should see something like this



In this example, the Genetic Algorithm has run for about 10,000 generations before finding the hidden string. As the population size in this case was 20, about 200,000 strings were tested.

**QUESTION 4**

*What is the size of the search space here? i.e. how many different random strings can be made using a sequence of 26 letters from the 49 allowed letters?*

Run the GA a few times (click on “start” again) to see how many generations it usually runs for.

Now we will see what happens if we change the mutation probability. Stop the program, go to the code, and change MUTATION_FACTOR to 0.0. Now the GA will run with crossover only, and no mutation.

Run it a few times (if the generation count gets up to over 500,000, stop the run by clicking on “stop”).

**QUESTION 5**

*Describe what happened. Attempt to explain why.*

OK. Now stop the program, set the mutation factor to 30.0, and run the GA a few times.

**QUESTION 6**

*Describe what happened and attempt to explain why.*

Set the mutation factor back to 1.0, and the crossover probability to 0. Run the GA a few times. Set the crossover probability to 0.9. Run the GA a few times.

It’s a bit hard to tell what’s happening because there is some randomness. Find this piece of code

```
double total = 0;
for(int i = 0; i < 30; i++)
{
   ga.run();
   total += ga.getGeneration();
}

System.out.println("Mean number of probes = " + POP*total/30);
```

Comment out ga.display() and uncomment the above code. Now run the GA. It will take a few seconds to run, and then print out a message with the average number of probes taken to find the hidden string over 30 trial runs (note this is the number of probes, not the number of generations).

**QUESTION 7**

*Try different amounts of mutation and crossover until you find a combination that gives an average number of probes of near to 40,000. What were the mutation factor and crossover probability that gave this result?*

*Can you say anything from this about what mutation probability and crossover probability should be used in a Genetic Algorithm?*

**When you have completed this workshop , submit your answers for the questions above to your tutor using the submission facility on Blackboard.**