159.372 Artificial Intelligence
Lab 3: Genetic Algorithms

Stephen Marsland

1 Introduction

In this lab session we are going to investigate the use of the genetic algorithm. This is the first lab session where you actually need to write some code. There isn't much to it, but it will still require a bit of thought. There are two parts to this: (1) working out what you want to do, and (2) actually coding it. I strongly recommend that you separate the two, to the extent that I won't help anybody to write code who doesn't have a comment line (use # at the start of the line in Python) that says what the code is meant to do.

As mentioned in the lectures, the code for a GA isn't very hard, but that doesn't mean that you want to spend an entire lab session writing one. As with the other algorithms in my part of the course, I have provided code for you. The code that you will be using is available on Stream, as 1 file of code for the GA (in `ga.py`) and 2 sample fitness functions (`fourpeaks.py` and `billsfit.py`). Once you have downloaded these 3 files, start Eclipse and make a new project (File>New>Pydev project); give it any name you like. Then import the 3 files using File¿Import.

There are two types of problems that people solve with GAs. The first are so-called 'toy' problems, which are very simple, and are designed to test how well the algorithm works. They usually run very quickly, but aren't that interesting. The two that I've given you fall into that category. The others are real problems, and usually take hours or days to run, which is impractical in a 2 hour lab. I've come up with a vaguely real problem for you to write a fitness function to attempt to solve.

2 Running the Genetic Algorithm

Almost all of the code is in the file `ga.py`. The main loop of the algorithm is in `Run_GA` and should be clear, with the different parts of the GA – selection, crossover, and mutation all being obvious. The parameters are all set in the initialisation. You should use this program as the basis of your work. I've implemented two types of crossover – single-point (`spCrossover`) and uniform (`uniformCrossover`). You can also choose to allow elitism, tournament replacement, or neither, and change the mutation rate.

At the top of the file there is an option to provide the name of your fitness function, and this is also specified in the call to the constructor (see below). These two have to match, unfortunately. The test fitness functions that I've given you solve is the 'four peaks' problem. It isdescribed in the next section, and you should try it out, trying the different options for crossover, etc., but following that I want you to change the fitness function name to `billsfit`, and then to edit that (very basic) file to write a fitness function. I've written the basic parts of the file, but you need to fill in the details, as discussed below.

In order to make the GA work, the Python code you need at the command line is:

```
import ga

g = ga.ga(20,'fF.fourpeaks',101,100,-1,'sp',4,False)
g.runGA()
```

The options in the initialisation line are the string length (which controls how hard the problem is for the two test functions), the fitness function (which has to match the `import` line in `ga.py`), the number of generations to run for, the population size, mutation probability (with -1 meaning that it is $1/L$, where L is the string length), the type of crossover (un or sp), the number of elite species to keep, and the type of selection that is used.

3 The Four Peaks Problem

Four peaks is a toy problem that is often used to test out GAs. The fitness function rewards strings that have lots of consecutive 0s at the start of the string, and lots of consecutive 1s at the end of the string. Have a look at the function, and you should be able to see this. There are two parts to it, the main part of the fitness function just counts the maximum number of 0s at the start or 1s at the end, and returns that as the fitness. However, there is also a bonus if both the number of 0s and the number of 1s is above some parameter T (which is preset to 15), in which case the fitness function has 100 added to it. This is where the four peaks come from – there are two small peaks where there are lots of 0s, or lots of 1s, and then there are two larger peaks, where the bonus is included. The GA should find these larger peaks for a successful run. See if you can find a GA that works consistently when the chromosome length is 100 and $T = 15$.

4 The Exercise

First of all you should play with the four peaks problem, and try out the different options for the selection and crossover functions and for the size of the population, etc. Once you've done that, I want you to use a GA to solve the following problem. You need to write a fitness function, and then run the GA a few times, trying out the different parameters and functions.

The problem is this: you are sitting in your student house and there are 10 bills that need to be paid (electricity, gas, rent, beer delivery, etc.). The values of the bills are in the top of the `billsfit.py` file in the sizes variable. The problem is that between you and all of your housemates you only have $500, which is not enough to pay all of the bills. You cannot pay part of a bill, for each one you either pay all of it or none of it. You want to spend as close to $500 as possible without going over that limit, so you need to design a fitness function that maximises this, and then use the GA to find a solution.The best possible solution for this problem spends $499.91.

There is a second set of bills in the commented out line below, which has 20 bills to pay instead. The best possible solution for this problem spends $499.98.

You need to change the length of the string passed to the GA (it was 20 in my example above, and it should be 10 for this problem) and also the fitness function that is passed and the fitness function that is imported. Then you need to write the fitness function, since it just returns zero at the moment. Note that the code will crash as it is, since my basic code returns all zeros, so when it divides by the total fitness it has a problem.

Things to think about:

- a fitness function should increase until the global maximum is reached
- a fitness function should always be positive
- it is worse to spend $501 than to spend $499, so make sure that is taken into account

Finally, you should run the exhaustive search and greedy search methods of finding the solution (in the imaginatively named `exhaustive.py` and `greedy.py`), and decide whether or not you prefer to use the GA to find the globally optimal solution. I wouldn't bother trying the exhaustive search on the one with 20 bills to pay, as it takes a while to run.