CNSCC.361 Artificial Intelligence

Lab Session 5 Genetic Algorithms

In this lab, you will see an example of a genetic algorithm implemented in MATLAB. We will use it to solve a simple knapsack problem. This is just an example, as genetic algorithms can be used to solve many different kinds of problems. The idea here is for you to understand how genetic algorithms work.

You may not finish everything in this document during the lab session. You can look at the code, try to understand it, and play with the parameters to see how it affects the results. Then, you can continue with the rest and think about them in your own time.

Knapsack Problem

The problem we have is that we are given 16 items, each occupying some specific units of space and worth some amount of money. You have to figure out which of these items to take (or not take), in order to keep the total units of space to within a specified maximum, while trying to maximise the total amount of money your items are worth.

The units of space and the amount for each of the 16 items are:

units of space	3	5	6	6	7	3	1	6	2	9	3	7	3	4	6	9
£	1.1	5.1	0.2	4.4	9.1	0.4	9.5	1.8	3.7	2.2	0.9	4.3	2.8	1.1	4.9	2.4

For example, if you were to take the 1st, 2nd and 4th items, the total units of space would be 14, and the total value would be £10.6. You have to take a set of items so that the total units of space is 40 or below, and the total value is as high as it can be.

First, try to think about which items you might pick manually. What is the highest total value that you can get by doing trial-and-error manually?

Genetic Algorithm

Manually trying to find a solution to the above problem is not so easy and maybe somewhat tedious. We apply a genetic algorithm to solve it. We use a binary encoding of possible solutions, where each solution is represented by 16 bits. The string of 16 bits together tells you which item you are or are not selecting. For example, this string "1101000000000000" means that you are picking the 1st, 2nd and 4th items, and none of the rest.

From Moodle, you can find the code for this in the "Lab 05" folder. The file "knapsack_problem_GA.m" has most of the code you need. Look through the code to try and understand what each part of the code is doing. Many of the ideas were also explained in the lecture.

You have to add a few lines of code to this part of the file:

```
% crossover prob 0.8 and random pick cross point
if (rand < 0.8)
    % need to add your own code here ...



end
```

And you also have to add a few lines of code to this part of the file:

```
% mutation prob 0.2 and random pick bit to switch
if (rand < 0.2)
    % need to add your own code here ...



end
if (rand < 0.2)
    % need to add your own code here ...



end
```

In the above code, you have to implement the crossover and mutation parts of the algorithm (it's only a few lines of code!). For the mutation part, you can do it separately for each of the "temp_chromosomes" (that's why there are two "if" loops).

After adding the code, you should be able to execute it and see the results. If you print the "population" variable, you can see the 20 chromosomes in the population (each with 16 bits and the last column is the fitness score). What is the largest fitness score (which is the same as the total worth of the items in pounds) that you can get?

Adjusting Parameters

There are many parameters that you can adjust in a genetic algorithm.

(1) Try changing the value of "`iter`" or the number of times a new population is generated. Execute the code, and print the "`population`" variable. How does this affect the results? Also try to change the population size "`population_size`", which is currently 20. Can changing the population size change the overall algorithm and results?

(2) Try different mutation probabilities and crossover probabilities. How do these change the results? Think about using different crossover strategies.

(3) The code currently implements elitism as a replacement method. Try not to use elitism at all.

(3) The code currently implements a selection methods (there is a selection function implemented in "Selection.m" file) discussed in the lecture. What type of selection is it?

(4) What is the highest total value that you can get in the knapsack problem? Try to change some of the parameters mentioned above.

Additional Questions to Think About (At home)

(5) Think about using Stochastic Universal Selection. How would this change the code and the results?

(6) In the knapsack problem, we limited the number of each item that we can select to one. What if we can select multiple items? So we may, for example, select three of the 1st item and five of the 4th item. How can this change the overall problem, code, and results? You don't have to do any implementation. Just think about the problem in general.

(7) In our problem, we had 16 items, each of which we can either select or not. We could have exhaustively listed every possible solution. How many possible solutions are there? In general, our genetic algorithm should evaluate the fitness function for a smaller number of times, compared to this exhaustive number of solutions. Otherwise, it does not make sense to use a genetic algorithm, and we could have just use exhaustive search.

MATLAB Toolbox

There is a MATLAB toolbox that has an implementation of GA. You can see an example of this here: http://www.mathworks.co.uk/help/gads/examples/genetic-algorithm-options.html. It tries to find the minimum of a specific "SHUFCN" function. There are only two values in the possible solutions, so it is not so complex. But the function itself is not simple, so it is not easy to find a solution for it either.