CS295/CS395/CSYS395: Evolutionary Robotics

Programming Assignment 1 of 10

Assigned: Friday, September 2, 2011

Due: Friday, September 9, 2011 by midnight

Description: The field of evolutionary robotics is so called because in a typical experiment, an evolutionary algorithm is used to automatically optimize robots so that they perform a desired task. There are many kinds of evolutionary algorithms, but they all share one thing in common: they are a simplified model of how biological evolution works. In short, in any evolutionary algorithm (1) the fitness of each entity in an intially random population is measured; (2) those entities with lower fitness are discarded; (3) modified copies of those that remain are made; (4) the fitness of each new entity is measured; and (5) this cycle repeats until a highly fit entity is discovered.

In this assignment you will be creating the simplest evolutionary algorithm, a serial hill climber. The algorithm is known as a hill climber because one can imagine the space of all entities that are to be searched as lying on a surface, such that similar entities are near one another. The height of the surface at any one point indicates the quality, or fitness of the entity at that location. This is known as a fitness landscape. A hill climber gradually moves between nearby entities and always moves ‘upward’: it moves from an entity with lower fitness to a nearby entity with higher fitness.

In this assignment, rather than evolving robots, you will simply evolve vectors of the form $v = e_1, \ldots e_n$, where the ith element in the vector may take on a real number in $[0, 1]$. The fitness of a vector v, denoted $f(n)$ we will define as the mean value of all of its elements:

$$f(v) = \frac{\sum_{i=1}^{n} e_i}{n} \tag{1}$$

Thus, the hill climber will search for vectors in which all of the elements have values near one. In assignment 3, you will re-use the hill climber you develop here to evolve neural networks; in assignment 10, you will use it to evolve robots.

Tasks:

You will be creating the hill climber in Python. If you are not familiar with Python, this assignment will serve as a good introduction to the language.

1. Download, install and run Python, an open source language available for most computing platforms. **You are free to do all of the programming assignments in a Mac, Windows or Linux environment.** In order to make sure you installed Python correctly, work through the first 10 minutes of any online Python tutorial.

2. In order to use vectors and matrices in Python you will also need to download and install NumPy and SciPy. In order to make sure you installed both of these applications correctly, work through the first 10 minutes of this tutorial.

3. Rather than submitting code to be graded, in this course you will submit screenshots of graphics that visually demonstrate you have successfully completed the assignment. The three screenshots you will create for this assignment look like those shown in Fig. 1. To create graphics in Python, download and install matplotlib (website here; download here), a Matlab-similar set of graphing libraries. In order to make sure you installed Matplotlib correctly, copy and paste the code associated with any of the graphs from the site's gallery.

4. The first step in the serial hill climber is to create a random vector. To do this, create a Python function **`v = MatrixCreate(1,50)`**. This function should return a vector with 50 elements, all set to zero. The vector is actually a 1×50 matrix. **`print v`** will show you whether the vector was created successfully or not.

5. Create a function **`v = MatrixRandomize(v)`** that will put random values drawn from $[0, 1]$ into each element of v. You'll need to use **`random()`**, which returns a random floating-point value in $[0, 1]$.

6. The hill climber must now compute the fitness of the random vector. Create a function **`f = Fitness(v)`** that returns the mean value of all of the elements in v.

7. The hill climber must now create a modified copy of v. Create a function **`c = Matrix Perturb(p,0.05)`** which makes a copy of the parent vector p, and then considers each element in the new vector c. With probability 0.05, the element's value is replaced with a new random value drawn from $[0, 1]$; otherwise, the element's value is left as is. You can cause an event to happen with a given probability using an if statement: **`if ( random()<0.05 ):`**.

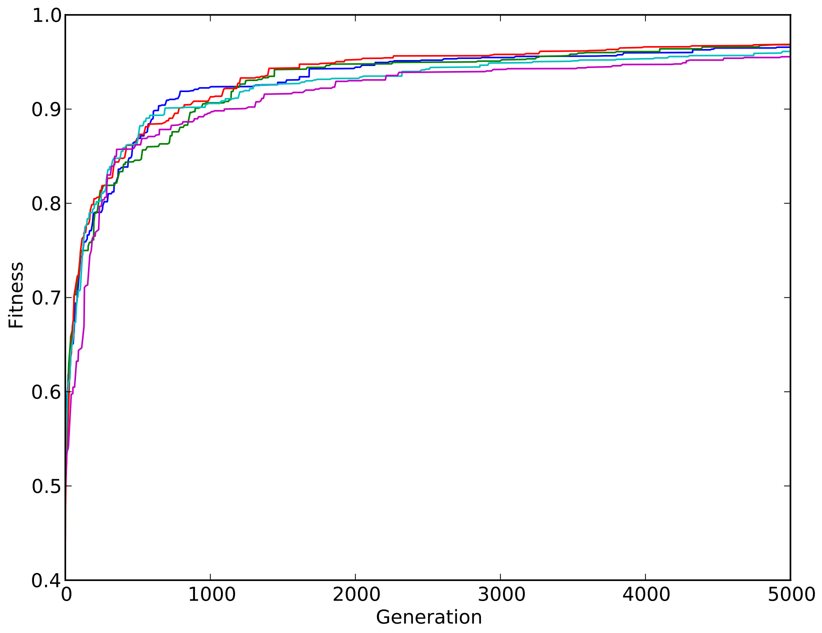
8. You can now use all of these functions to create a serial hill climber:

```
parent = MatrixCreate(1,50)
parent = MatrixRandomize(parent)
parentFitness = FitnessParent(parent)
for currentGeneration in range(0,5000):
    print currentGeneration, parentFitness
    child = MatrixPerturb(parent,0.05)
    childFitness = Fitness(child)
    if ( childFitness > parentFitness ):
        parent = child
        parentFitness = childFitness
```

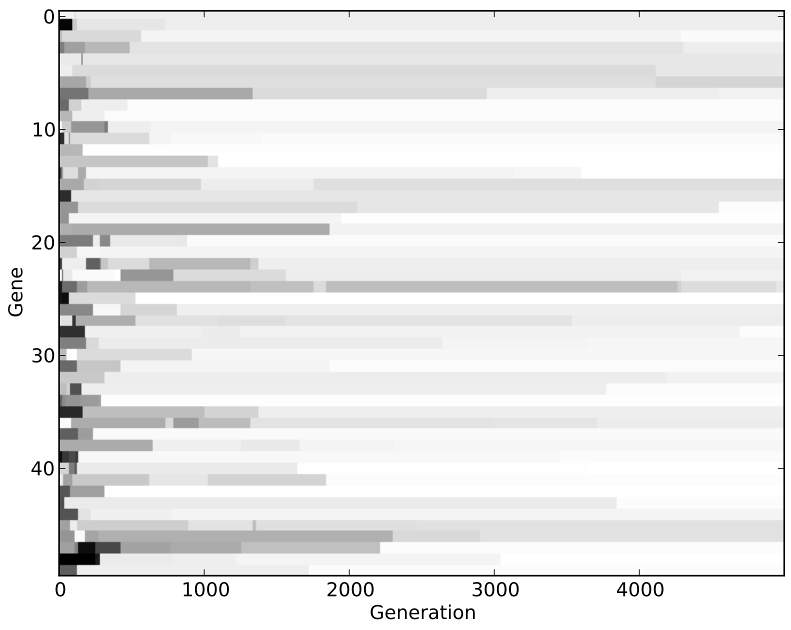
9. You should see that as the fitness of parent is printed, the fitness value goes up as the generations pass.

Deliverables:

1. The first graph you will create will visually show how the fitness of the best vector climbs as the generations pass. In your existing code, create a new vector **`fits = MatrixCreate(1,5000)`** that stores the fitness value of the parent at each generation. Print **`fits`** after your code has run to make sure the fitness values have been stored.



a



b

Figure 1: Results from running five serial hill climbers. a: The fitness curves for the five runs. b: The values of the 50 genes in the vector: lighter values indicate higher values.

2. Create a function **`PlotVectorAsLine(v)`** that plots the vector as a line (use **`plot()`** from matplotlib). The graph should show one line with a curve similar to those of the lines in Fig. 1a. Save this figure, and paste it into either a Word document or a Latex document.

3. Wrap the Python code from step 8 above in a loop that runs the hill climber five times, each time starting with a different random vector. At the end of each pass through the loop, add another line to your graph, so that you have a picture similar to that in Fig. 1a.

4. Create a matrix **`Genes`** with 5000 columns and 50 rows. After each generation j of the hill climber, copy each element of the parent vector into the jth column of **`Genes`**. After the hill climber has run, **`print Genes`** to ensure the elements were stored correctly.

5. The matplotlib function **`imshow(M)`** will print a matrix M as an image, where each pixel p_{ij} corresponds to element e_{ij} in M. Calling **`imshow(Genes`**, **`cmap=cm.gray`**, **`aspect='auto'`**, **`interpolation='nearest')`** after the hill climber has terminated will produce a figure similar to that of Fig. 1b. **`cmap=cm.gray`** will ensure that the image is shown in grayscale: elements with values near 1 will be plotted as near-white pixels; elements near zero will be plotted as near-black pixels. **`aspect='auto'`** will expand the otherwise very long, flat image to fill the figure window. **`interpolation='nearest'`** will stop any blurring between the pixels.

6. Copy and paste the resulting figure into your document with the other two figures. Now, convert your document to pdf using a pdf conversion application such as PrimoPDF. Email the pdf to the TA.

Notes:

1. For those of you working on a Mac, the order in which you install the Python packages matters. From the command line on a Mac:
   ```
   python
   >>> import scipy
   >>> import numpy
   >>> import matplotlib
   ```
   Do not `install matplotlib.pyplot`; you can call the matplotlib routines after installing these three packages