Modern Robots: Evolutionary Robotics

Programming Assignment 1 of 8*

Overview

In the field of evolutionary robotics 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 have 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 initially 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 $v = e^1, ...e^n$, where the *ith* element in the vector may take on a real number in $[0, 1]$. The fitness of a vector v we will define as the mean value of all of its elements.

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 5, you will use it to evolve robots.

Tasks

You will be creating the hill climber in C++, and create plots of its performance in Python. If you are not familiar with Python, don't worry, the actual coding in Python will be minimal.

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 a tutorial online that uses both/either.

*Original material was graciously provided by Josh Bongard. Jeff Clune slightly modified it. Joost Huizinga heavily modified it.

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, 2, and 3. To create graphics in Python, download and install matplotlib, 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. After you have successfully installed Python it is time to start working on the actual hill climber. A framework for the hill climber is provided in a set of header files that comes with this assignment. Extract the code archive and make sure that it compiles.

5. The first step is to create an individual for your hill climber, in this case a vector of doubles. In `Ind_Vector.hpp` you will find the basis for such and individual. Implement the functions `randomize` and `mutate` to assign a random value to every element, and to assign a random value to random elements, respectively (see the comments for details). Look at `Misc_Random.hpp` for functions that produce random values.

6. The second step is to create a fitness function for our vector. Open `Fit_Vector.hpp` and implement the `evaluate` function to set the fitness of the individual to be the average of its elements. Use `individual.setFitness(fitness)` to assign the fitness to the individual once it is calculated.

7. The third step is to implement the actual hill climber. Open `EA_HillClimber.hpp` and implement the `init` function and the `run` function. The `init` function should set the fitness function as well as set, randomize, and evaluate the parent. The `run` function should run the hill climber for the provided number of generations, and it should write the parent and the fitness of the parent to two separate files. See the comments in `EA_HillClimber.hpp` for details.

8. The fourth step is to actually run the hill climber. You should now be able to open `main.cpp` and run the following code:

```
//Define the type of our individual.
typedef IndivFitness<Vector> ind_t;

//Set the seed to be based on the current time.
//You can also set the seed to a particular number.
seed();

//Set the fitness function.
//This fitness function has no parameters, so it takes no arguments.
VectorAverage fitnessFunction;

//Set the initial individual to have a length of 50,
//a minimum value of 0,
//a maximum value of 1,
//and a mutation rate of 0.05
ind_t initialIndividual (Vector (50, 0.0, 1.0, 0.05));

//Create our hill climber. Note that we have to provide the type of individual
//(VectorIndividual) and the type of fitness function (VectorFitnessFunction)
//between the corner brackets.
HillClimber<ind_t, VectorAverage> hillClimber;

//Initialize our hill climber with the initial individual and the fitness function.
```

```
hillClimber.init(initialIndividual, fitnessFunction);

//Run the hill climber for 5000 time-steps,
//and write the fitness to "hw1_fitness1.csv"
//and the parent for each generation to "hw1_individual1.csv"
hillClimber.run(5000, "hw1_fitness1.csv", "hw1_individual1.csv");
```

9. You should now be able to visualize your fitness over time. Use the provided `plotLine.py` to create a line plot of your fitness over time. The full command would be:

```
python plotLine.py hw1_fitness1.csv
```

The resulting image should look like figure 1.

10. You should also be able to plot the genome of the parent of generations. Use the provided `plotMatrix.py` to create a plot showing the genome of the parent over generations. The full command would be:

```
python plotMatrix.py hw1_individual1.csv
```

The resulting image should look like figure 2.

11. Run your hill climber 5 times, and make sure that each run creates a different fitness file (don't forget to change the seed if you defined it!). Plot the fitness of all runs in a single plot using `plotLine.py`. Note that you can simply provide a list of files to plot multiple lines. For example:

```
python plotLine.py fit1.csv fit2.csv fit3.csv fit4.csv fit5.csv
```

The resulting image should look like figure 3.

Deliverables

Generate the figures looking like figures 1, 2, and 3, and put them into a single pdf document, and name it: hw1_your_name.pdf. Also, take all files you modified and put them into a single zip archive named: hw1_your_name.zip. Email the pdf document and the zip file to the grader.

Notes

1. For those of you working on a Mac, the order in which you import the Python packages may matter. From the command line on a Mac:

```
python
>>> import scipy
```

```
>>> import numpy
>>> import matplotlib
```

2. Make sure you get the right version of numpy for your python version. If, for example, you have python 3.3 you need a python3.3 package.

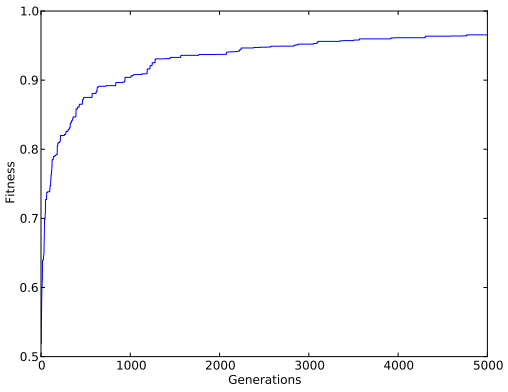

Figure 1: Fitness curves from one run of the hill climber.

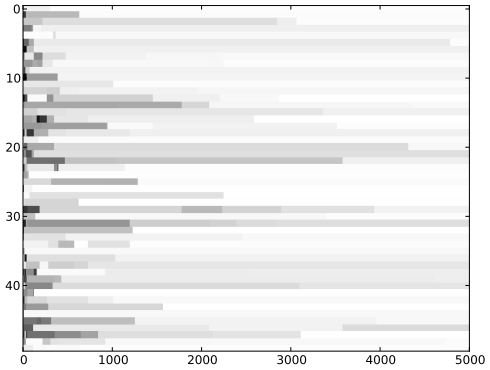

Figure 2: The values of the 50 genes in the vector: lighter values indicate higher values.

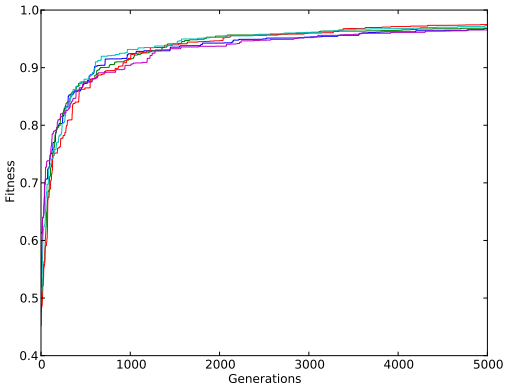

Figure 3: Fitness curves from each of five runs of the hill climber.