Modern Robots: Evolutionary Robotics

Programming Assignment 6 of 10*

Description

In this assignment you will create a population based evolutionary algorithm with tournament selection and use it to evolve the neural network driving the table robot.

Tasks

1. Copy all your code for assignment 5 to the folder you will use for assignment 6. Make sure to keep a working backup of assignment 5.

2. Extract `hw_6_code`, and move `EA_Population.hpp` and `Sel_TournamentSelection.hpp` to your assignment 6 folder.

3. Open `EA_Population.hpp` and implement the `init`, `epoch` and `getBest` functions. Look at their comments for the specifications.

4. Open `Sel_TournamentSelection.hpp` and implement the three versions of the `select` function.

5. Open `main.cpp` and comment out or delete your previous code (backup!). Run the population based evolutionary algorithm to evolve a population of 50 individuals for 10 time-steps. The individuals should be the same as in assignment 5: neural networks with 4 inputs, each directly connected to all 8 outputs. Parent and survivor selection should be set to tournament selection with a tournament size of 2 and 'elitism' enabled. The `RobotDistanceFitness` should be the fitness function.

Remember that, to create an instance of the population based evolutionary algorithm, you will need to supply both the parent and survivor selection objects. Assign them those as follows:

```
RobotDistanceFitness fitnessFunction(&simulator);
TournamentSelector selection(tournamentSize, true);
PopulationEA<fit_t, TournamentSelector, TournamentSelector> ea;
hillClimber.init(initialNetwork, fitnessFunction, selection, selection, populationSize);
```

Here, `tournamentSize` and `populationSize` should be set according to the values mentioned above and `simulator` should be your instance of the simulator.

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

Write the fitness of the best individual over generations and the average fitness of the population over generations to two different files.

6. If the algorithm seems to work correctly, rerun the code 5 more times, each time writing to different files. Plot the 5 files containing the best fitness over generations to a single file with `linePlot.py` (supply all file-names as command line arguments in a single command to do so). Do the same for the average fitness files. Add these files to your document, they should look like figures 1 and 2.

7. Lastly, we'll check for the effect of having a population. Change the population size to 1 and change the number of generations to 500, so the number of evaluations stays the same. These changes effectively turn the algorithm into a basic hill-climber. Rerun the algorithm 5 times and show the best fitness over generations in a single plot. Add this figure to your document, it should look like figure 3. What do you notice? Is the population based algorithm always better than the hill-climber? Why? You do not need to include the answers to these questions in your document.

Deliverables

A pdf document containing the figures resembling figures 1, 2, and 3, and any files you changed.

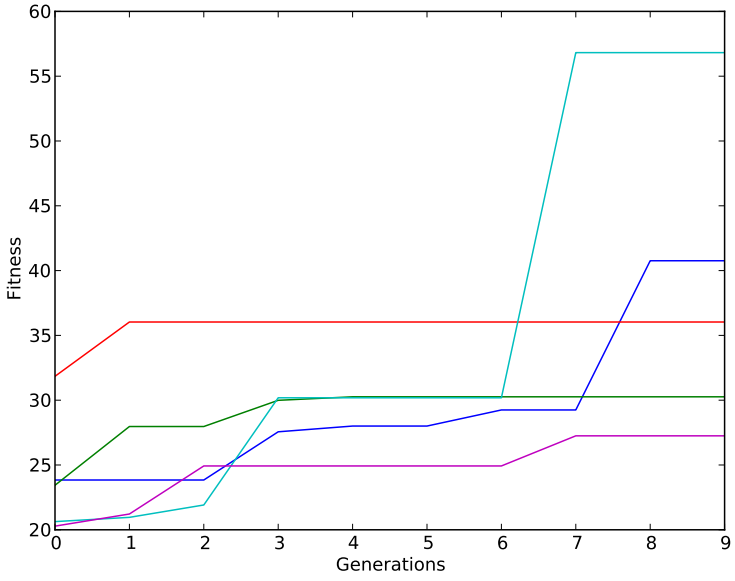

Figure 1: The fitness of the highest performing individual in the population over 10 generations.

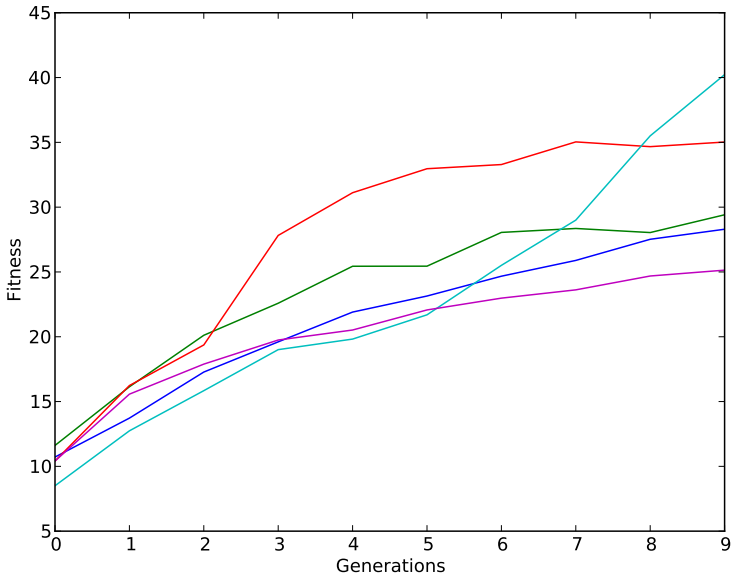

Figure 2: The average fitness of the population over 10 generations.

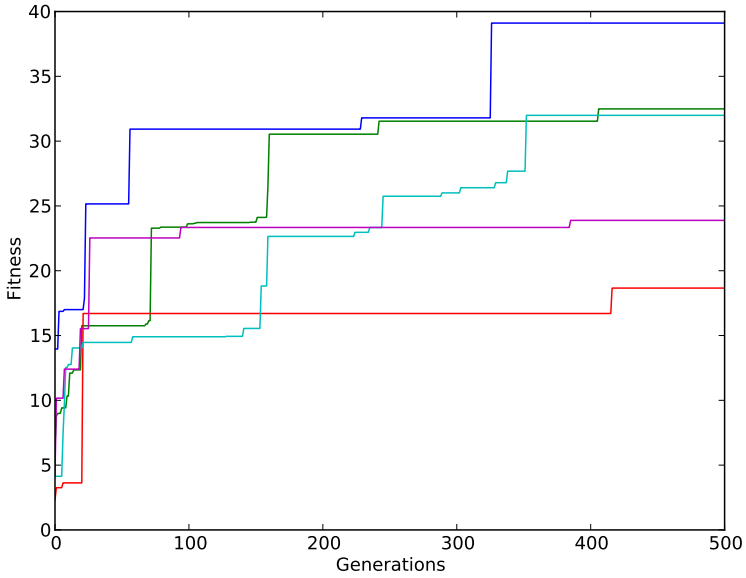

Figure 3: The fitness with a population of 1 over generations.