CSG2341 Intelligent Systems

Workshop 10: Hybrid Intelligent Systems

Related Objectives from the Unit outline:

- Identify appropriate intelligent system solutions for a range of computational intelligence tasks.
- Demonstrate the ability to apply computational intelligence techniques to a range of tasks normally considered to require computational intelligence.

Learning Outcomes:

After completing this workshop, you should be able to describe and analyse the steps in designing a hybrid intelligent system. You should also be able to use a computer package to implement a hybrid intelligent system that uses an evolutionary algorithm to train a neural network, or uses an evolutionary algorithm to train a fuzzy system.

Background:

In this workshop, we return to the cruise controller from Module 3. The simulation has been changed just a little to include the effects of air resistance. We experiment with using evolutionary algorithms to evolve controllers with different characteristics.

Task:

Step 1:

Download the file `EvolveCruiseControl.zip` from BlackBoard and extract it to a working directory. Load up in NetBeans and build the project. You will find three packages, `CruiseControl, EvolveFuzzyCruiseControl` and `EvolveNeuralCruiseControl`. Some key source files:

In CruiseControl:

`CruiseEvaluator.java` – this is the source file for a class that tests out controllers and returns a fitness value. It can be used as the evaluator for an evolutionary algorithm.

In EvolveFuzzyCruiseControl:

`MyCruiseController.java` – this is a source file for a cruise controller based on a fuzzy rule set. The fuzzy rule set doesn't work very well. Run `CruiseControlTest` to test it.

`EvolveController.java` – this is a driver class that lets you evolve controllers based on `MyCruiseController`, using `CruiseEvaluator` to measure fitness.

Run `EvolveController.`

When the evolution is complete, a window will pop up that lets you try out the best evolved controller. The program will also test the best evolved controller and print out a message in the output window like this:

```
Overall fitness = 0.8509245011950851

Average absolute speed error = 0.1753323673529607

% of OK steps = 99.867

Average force used = 1990.3708437963162
```

The first number gives the fitness value for the best ever controller evolved during the running of the evolutionary algorithm, averaged over a number of trials. The second number is the average absolute difference between the target speed and the actual speed of the car over the test period. The third figure is the % of time the car was within 3 m/s of the target speed. The last figure is the average amount of force used by the controller over the test period.

Run the program and record these figures.

Step 2:

The average amount of force used is related to wear and tear on the car, fuel consumption, emissions, brake life etc. The car's manufacturer wants to modify the controller to reduce these costs, while still keeping the car within 3m/s of target speed at least 90% of the time.

Design a new fitness function and use it to evolve a new controller that meets these requirements. To do this, you will need to change the method `evaluate()` in `CruiseEvaluator.java`. This method has a code segment:

```
// load the parameters into the controller
controller.load(evolvable);
// simulate the setup for steps timesteps, trials times
totalError = 0.0;
totalOK = 0;
totalForce = 0.0;

for(int t = 0; t < trials; t++)
{
    setup.init();
    for(int nsteps = 0; nsteps < steps; nsteps++)
    {
        totalError += Math.abs(setup.getSpeedError());
```

```
            if(Math.abs(setup.getSpeedError()) < TOLERANCE)
               totalOK++;
            totalForce += Math.abs(setup.getForce());
            setup.update(
                controller.computeForceChange(setup.getSpeedError(),
                setup.getA()), TICK);
        }
    }

    return 1.0/(1.0+totalError/(steps*trials));
```

The inner loop calculates the total error (difference between the target speed and the actual speed) over a trial run, the number of steps where the speed was within 3 m/s of the target speed, and the total force used.

At the moment, the fitness value returned is highest when the total error is lowest. You will need to change this method to return a value that will help to meet the manufacturer's requirements.

Step 3 (hard!!):

Repeat the above with the package `EvolveNeuralCruiseControl` which uses an evolutionary algorithm to evolve a neural network to solve the cruise control problem.

Answers for submission:

1. Provide the code for your modified fitness calculation from Step 2.
2. Compare the controllers evolved in Step 2 with the two different fitness functions.
3. Notice that the number of generations has been set much higher in Step 3 than with the fuzzy controller. Why do you think that might be?
4. Compare the performance of the fuzzy and neural evolved controllers. Note that the lecturer was not able to get the neural controller to work well with the new fitness function. If you succeeded, well done and I'll be interested to see how you did it!

When you have completed this workshop, submit your CruiseEvaluator code and your other answers to your tutor using the submission facility on BlackBoard.