Genetic Algorithm Robots

Professor Caleb Fowler

Sp16

Problem.

You are going to reproduce an experiment first conducted at Harvard University in 1968. You are going to study the effects of evolution on a population of robots. The robots need to maneuver around a room collecting energy. The robots must collect more energy than they expend to survive.

The key is the robot behavior. Each robot has a collection of direction sensors and a move counter. It is also capable of several behaviors (turn, forward, do nothing). The key is mapping the behavior to specific sensor readings. The robots start with a random mapping, but over time, the mappings evolve into successful strategies.

The genetic algorithm is the key to the robot brain. This is a mapping of stimulus (via sensors) and behavior. The idea is we read the sensors and if they match a pattern specified on one of our genes, we execute the behavior. Think of this as an array. Each element holds a gene comprised of characters that represent the sensor and the behavior. For example, the first character always represents the state of the N sensor. It will hold either a 0 - nothing there, 1 - battery, 2 - don't care, or a 9 - wall. The next three characters represent the East, South and West sensors. A 2 codes for don't care, so you see a problem with my coding scheme. The fifth character is the direction the robot is currently facing. Assume we have a partial gene like this: 922N2. That means the associated behavior will activate if there is a wall to the north of the robot, the robot is facing North and we don't care what is in the other directions.

We continue with our gene by adding behaviors. The robot reads the gene and executes the behaviors from left to right. Let's add a M for move and XXX. Our completed gene will look like 9222NMXXX. Add a variable which holds the direction the robot is facing (don't inherit this). Assume for this example our robot is facing North. If the sensor pattern matches the first 4 characters of this gene, and the current facing matches the facing specified on the gene, then the corresponding behavior will execute. This means the robot will move north (right into the wall). This is not going to end well for our robot.

However, this gene would be more successful in this situation: 9222NRMXX (R - rotate right). In this case, the robot would detect a

N wall and notice it was facing north; triggering the rotate 90 degrees to the right and move one square behaviors - a far more successful strategy. Don't forget to update the robot's facing variable after it executes a rotation.

There is nothing special about this way of encoding the genetic information. You could use structs or parallel arrays if you wanted to. You can also use whatever characters and symbols you want to code the information. The key is you are recording the mapping between sensor data and subsequent behavior.

Start with 24 genes of 9 characters (or chromosomes). Each turn your robot will iterate through the 24 genes and try to find a match. It is up to you to decide if you want it to use the best match or the first match. The best match will probably yield better results.

You will keep score for each robot by measuring how much energy it harvests from the environment. A robot must move over top a battery to harvest it's energy. Once harvested, the battery is gone and removed from the map. Run each robot for a fixed number of turns (12-25). Record the score after that. Robots can run out of energy and die. They start with a fixed amount of energy (7-10 power units).

Once all the robots in a population have had a turn (and acquired energy scores), record the total energy harvested by the entire generation and breed your robots. Sort your population by energy harvested and kill off the lowest 50%. Then pair up the top 2 robots and produce 2 children by combining genes from both parents. The children enter the gene pool with the parents in the next round. Then, breed the 3rd and 4th highest scoring robots. Repeat until all the robots have reproduced. Keep the number of robots at a fixed number for the entire simulation - I suggest 200 individuals to start.Genes are randomly created for the first population only - the robots breed after that.

Each parent supplies 12 of the 24 genes for a child. The simplest way is for one parent to supply the first 12 and the other to supply the last 12. You will not want the siblings to have the same genes (unless you are investigating the effects of identical twins). However, you can use a different swap scheme if you would like.

Swapping genes is a tricky business and mistakes happen. In 5% of the individual genes swapping there is an error - a mutation. Randomly change one character on the gene the child has inherited from it's parent. Just generate another value and insert that value over the old one.

Sensors.

- Sense if there is an object (wall, battery, nothing, don't care) next to them in a given direction: (North, South, East, West).
- The direction it is facing (and moving).
- Keep track of total energy.

Behaviors.

- Move forward 1 square. This costs 1 energy.
- Rotate left 90 degrees for no cost.
- Rotate right 90 degrees for no cost.
- Pick up battery (if standing on battery) and gain +5 units of power.
- Do nothing.
- You can have up to 5 behaviors per gene. Code Do nothing for empty spaces if you use less than that.

The degree to which the robot successfully harvests energy from the environment is called 'fitness'. We measure that by the total amount of power harvested when each individual robot's time ends. When we finish with the entire population of 200, we calculate an average fitness score for the population. Save the average fitness for each generation. You will most likely see slow and steady improvement over time - evolution at work. When the simulation completes, print out the average fitness scores on the console. This is even more effective if you are able to draw a console graph (not a requirement).

Constraints.

- Create a 12 by 12 square room.
- Start your robot on a random square with 5 units of power.
- Populate 40% of the squares with batteries.
- Each robot will move a total of 12-25 times unless it runs out of power before that (you decide at the start of the simulation). The robot's score is the amount of power it has when it reaches this exit condition.
- Robots that hit walls with extra moves will still try to move (and fail and lose 1 power each time they hit the wall).
- Use a robot population of 200 individuals. Run all the robots through the room before you breed them.

Bonus Features.

- No bonus features are needed with this assignment. However, past students have:

- Added obstacles for the robots to avoid.
- Added predator robots.
- Added vision so the robots can 'see' 2 spaces away.
- Added memory of moves.
- Plotted fitness over time on the console (a common modification).
- Created Don Juan robots that have children with several partners.
- Stored many of the constraints as constants so they could explore the effects of modifying them (ex: change the mutation rate to 8% from 5%) - another common modification.

Due Date and Turn In.

This assignment is due before the first class of the week on the week it appears under Hw Due in your syllabus. It is due by the end of the class. Do not count on D2L holding multiple copies of your homework (it should not do so). I grade the most recent version only (if I made a mistake and allowed multiple submissions).

TURN HOMEWORK IN by uploading to the appropriate D2L Dropbox folder. You do not need to put your name in the **filename**; Homework1, 2 whatever will be just fine. D2L appends student information to the files when I download them, so I will see all this information automatically. I will review your work using the rubric at the end of the assignment.

DO NOT SAVE your code as a .cpp file! Save it as a .txt file instead. Don't zip or otherwise compress your files. I will be able to read them once you get them on D2L. I have a script which converts the files to .cpp and automatically executes them. This script also runs other tests with them as well.

Client's System

Your code must run on a Ubuntu 16.04 linux system. I will compile it with the following

```
g++ -std=c++11 -g -Wall <filename.cpp>
```

Using the Work of Others.

This is an individual assignment, you may use the Internet and your text to research it, but I expect you to work alone. Copying code from someone else and turning it in as your own is plagiarism. However, you **may** discuss code and the assignment. I have

opened discussion groups in D2L to do this. I will monitor this, but not interfere. D2L will check your code against a database of other assignments. It tells me how similar your code is to someone else's. I consider isomorphic homework to be plagiarism. Do your own work.

Comments.

Genetic Algorithms are the real deal in Artificial Intelligence. It is still an active area of research. You will see how powerful this technique can be with this homework. Students frequently tell me they get a real sense of accomplishment when they turn this in. They often feel like they can handle any programming assignment after this - which is exactly what we are trying to do in this class!