Massachusetts Institute of Technology

6.038 Artificial Intelligence in Practice

Problem Set 2A: Tuesday, February 10, 1998
DUE: Thursday, February 12, 1998

1 Background

The objective of this assignment is to get you thinking about machine learning in problems where actions are called for and the "right" answer is not necessarily known ahead of time. While we are at it, we'll review some simple concepts from search and game playing.

2 The `Robots` Game

To have a specific subject to focus on, we'll use a variant of a simple video game called `Robots`. The game is played on a 10×10 grid (normally 20×20), on it are the player (represented by `I`) and some killer robots (represented by `=`). Time is discrete; the player and the robots alternate moves. The player and the robots can move to any of the 8 squares adjacent to their location. The robots always move to the square closest to the player. If a robot reaches the player, the player dies. If two robots collide, they die and leave a junk heap (`@`). If a robot collides with a junkheap, it dies. In a typical `Robots` game, the player can also choose to teleport to a random location on the board, but we have disallowed that in our version. Instead, we have (optionally) allowed the player the ability to wraparound on the board. That is, as far as the player is concerned opposite ends of the board are identical or, in other words, the game takes place on the surface of a torus. To the robots the edges always behave as walls.

We have provided you an implementation of this game where you can control the player or have the computer choose the play. You can find this at `/mit/6.038/ps2/bin/robots` on Athena. The command-line arguments are:

```
-f savefilename          -- default is none
-k keyboard_play (0 or 1) -- default is 0
-d delay_seconds_between_moves -- default is 1
-s random_seed           -- if you want to replay a particular game
-l lookahead             -- depth of search, default is 4
-n num_robots            -- default is 4
-g num_games             -- default is 1
-w wrap_around (0 or 1) -- allow wraparound? default is 1
-r random_I    (0 or 1) -- start I in random location? default is 0
-m max_iterations        -- default is 20
```

In that same directory you can find (most) of the source code for the implementation.

3 Assignment

3.1 Playing the Game

1. Play a few games to get a feeling for the strategy. You can do this by executing: `robots -k 1 -g <n games>`.

2. Play at least ten games and save the trace into a file. You can do this by executing: `robots -k 1 -g <n games> -f <filename>`. Please don't change any other defaults when you do this. Then copy the resulting file into `/mit/6.038/ps2/saved/<your login name>`. We will be using these in Part B of the problem set.

Yes, you can tell your roommate we forced you to play a video game...

3.2 Analysis and Reflection

Look at the code in `/mit/6.038/ps2/src/robots.c`. We've given you the code for `pick_move_by_search`. However, we have only given you an outline for the key functions `evaluate_move` and `static_score`. In Part B of this assignment, you will be asked to complete the code, among other things. The idea of the code is to look ahead a specified number of moves (`lookahead_depth`) and select the current move that leads to the best final board position.

These questions ask you to think a bit about all this. As usual, we are looking for brief answers here. No more than a page total.

1. How would you rate the search-based player (when playing with and without `I` wraparound)? Do you notice any weaknesses in it (using the default lookahead). Can you explain why the weaknesses might be there?

2. Does more lookahead search help? Empirically, what's the best value of lookahead? Answer this both for the game with and without `I` wraparound allowed. Provide data to justify your answer. Can you explain your observations? Note that a lot of the actual time in playing the game is taken up in the enforced delay between moves so you can see what's happening. This is specially true if one reduces lookahead. If you want the machine to quickly play a lot of games just so you can see how many times it wins, you can use the `-d 0` argument.

3. Think about scaling your `evaluate_move` function to play a more complicated video game, e.g. "Super Mario 64" or "Crash Bandicoot". Or if you've never played a video game, think about making the decisions involved in a typical day in your life. Assume somebody else has given you a suitable static evaluation function. What computational bottlenecks would you encounter?

4. How would you set up this general type of problem (not just Robots but also the more complex ones) as Machine Learning problems? What benefits and problems would you foresee?