Programming Assignment 3: Reinforcement Learning

CS 436 – Artificial Intelligence – Fall 2008

Due: 12:00 Noon, December 15, 2008

Abstract

For the third programming assignment, you will implement a reinforcement learner and apply it to the racetrack problem. The racetrack problem is a standard control problem that will serve to illustrate several interesting properties of reinforcement learning algorithms.

The writeup for this assignment will consist of a conference-paper style report on your findings. For this assignment, you will not be provided with any explicit template. You should use the information from the template for the second assignment and your feedback on the paper you wrote for that assignment to figure out what your paper should look like and contain.

For this assignment, you are being given the option of pairing up with at most one other student in the class. In this case, you may submit a common set of software; however, everyone is required to submit an individually-written report. All submitted code must be written by you or your team. You may only discuss this assignment with the instructor. Any references used while working on this assignment must be cited properly in the reference section of your paper.

1 The Racetrack Problem

The racetrack problem is a standard control problem. The goal is to control the movement of a race car along a pre-defined racetrack. You want your race car to get from the starting line to the finish line in a minimum amount of time. In this version of the problem, your agent will be the only racer on the track, so it is more like a time trial than a full competitive race.

More formally, at each time step, the state of the agent can be encoded in four variables: x_t and y_t are the x and y coordinates corresponding to the location of the car at time step t (we treat the racetrack as being layed out on a Cartesian grid). The variables $\dot{x}_t$ and $\dot{y}_t$ represent the x and y components of the car's velocity at time t. The control variables for the car are a_x and a_y, which represent the x and y components of an acceleration vector to be applied at the current time step. The system is governed by the standard laws of kinematics:

$$\begin{aligned} x_t &\equiv x \text{ position} \\ y_t &\equiv y \text{ position} \\ \dot{x}_t = x_t - x_{t-1} &\equiv x \text{ speed} \\ \dot{y}_t = y_t - y_{t-1} &\equiv y \text{ speed} \\ a_{x_t} = \ddot{x}_t = \dot{x}_t - \dot{x}_{t-1} &\equiv x \text{ acceleration} \\ a_{y_t} = \ddot{y}_y = \dot{y}_t - \dot{y}_{t-1} &\equiv y \text{ acceleration} \end{aligned}$$

At any given time step, your car only has active control over the values of a_x and a_y and must use these control variables to influence the car's state. This essentially gives you the ability to accelerate, decelerate, and turn. There is a further restriction that the set of values that may be assigned to a control variable is -1, 0, and 1. That is,

$$a_{\{x,y\}} \in \{-1, 0, 1\}.$$

The speed of a car at any given time is limited to $(\dot{x}_t, \dot{y}_t) \in [\pm 5, \pm 5]$. Any attempt to accelerate or decelerate beyond these limits will be ignored.

As an example, if at time $t = 0$ your car is at location $(2, 2)$ with velocity $(1, 0)$, it is essentially moving towards the east. If you apply an acceleration of $(1, 1)$, then at timestep $t = 1$ your position will be $(4, 3)$ and your velocity will be $(2, 1)$. At each time step, your acceleration is applied to your velocity before your position is updated. This is not strictly a good model of Newtonian physics, but we're dealing with integer position and velocity values, so some simplifying abstractions need to be made.

If this were the extent of the problem, it would be quite easy to solve for any given track, so to make things a bit more interesting, we are going add in a small amount of non-determinism. To do this, we assign a probability to each attempted action for each possible outcome. The simple example we will use in this problem is that, for any attempt to accelerate, there is a 10% chance that attempt will simply fail, and the velocity will remain unchanged at the next timestep. Thus, at each timestep, the probability of accelerating as specified is 90% and the probability of having no acceleration is 10%.

In this problem, there is an additional requirement that you stay on the track; crashing into the wall is bad. You will experiment with two different versions of how "bad" it is to crash. The first variant says that, if the car crashes into a wall, it is placed at the nearest position on the track to the place where it crashed, and its velocity is set to zero. The second, harsher variant says that when a car crashes, its position is set back to the original starting position, as well as zeroing its velocity. Essentially, in this latter case if you crash, you have to start over from the beginning. You should implement both variants in your program so that you can experiment with what effects they have on the strategies your car learns. Since you have a limited time to run experiments, however, you are only expected to do a side by side comparison of the two definitions of a "crash" on the R shaped track (in the file `R-track.txt`). For the other tracks, use the version where crashing stops the car but leaves it in the location it crashed.

The cost function is 1 for each move, except when you reach the finish line. The finish line locations are absorbing states with cost 0. Since you are attempting to minimize cost, this translates to attempting to complete the race in as few time steps as possible.

2 Program Requirements

2.1 Submission Requirements

You are welcome to develop your programs on any computer, using any "traditional" language. All code should be well structured and well commented. "Self-documenting" coding practices are encouraged, but are almost never sufficient to replace comments entirely.

Your program should be submitted as an archive, either a zip file or a tarball, with the filename consisting of your first and last name and the assignment name, delimited by underscores, all lower case. For example, '`jane_doe_pa3.tar.gz`'. You should submit this archive via email to the instructor at `john.sheppard@cs.montana.edu`.

Your archive should contain any and all files required to build and run your program. It should *not* include the provided data set(s), but it *should* include any data sets used by your program that were identified as part of the assignment. It should also contain a README file, in which you list the files in your archive and a brief description of what each file contains. Your README should also contain any relevant notes on the functioning of your program, such as known bugs, parameters that result in excessive runtime or poor behavior, etc.

Your archive should also contain a `./data/` subdirectory which should contain output from sample runs, performance information, and other information as requested. Sample runs are required; however, you may provide sample runs on a much smaller track than those provided for your actual experiments.

2.2 Feature Requirements

You should implement the racetrack problem as described above. Then you should implement both the Q-Learning and Value Iteration algorithms for reinforcement learning and apply them to the racetrack problem. You should apply these algorithms to the tracks in the data files described below. It is up to you to determine appropriate design parameters such as learning rate, discount factor, exploration strategy, etc.

Keep in mind that reinforcement learning is often very slow. Since you need to run a number of experiments, you are advised to start writing your program early, since even after it is working, you need time to run experiments and collect data. Remember that you *may not use* code from any external source.

2.3 Data Descriptions

The data files for this problem are ASCII representations of racetracks. The first line lists the size of the track as an comma delimited pair $\langle rows, cols \rangle$. The rest of the file is a grid of the specified dimensions with one character at each point. The legend for the files is:

- `S` – This square is on the starting line.

- `F` – This square is on the finish line.
- `.` – This square is open racetrack.
- `#` – This square is off the racetrack (i.e., a wall).

As a simple example:

```
5,5
FF###
..###
..###
....S
....S
```

Note that any square off the edge of the defined map is considered to be off the track (ie. a wall). You are responsible for determining if you have intersected with a wall or other obstacle. This may require determining whether or not you "clipped" a corner.

You will be provided with three sample tracks to use in your experiments. The files are named `L-track.txt`, `O-track.txt`, and `R-track.txt` and can be found on the course website. You are required to perform experiments using these tracks. If you wish to create additional tracks to highlight interesting behavior of your algorithms, you are free to do so as long as the results add more to your paper than length.

3 Report Requirements

As with the previous assignment, your report should take the form of a scientific paper. No new template will be provided with this assignment; you should use the template from the previous assignment and your judgement as a writer to decide how to format your paper. The paper should be submitted as a PDF and will be graded based on formatting, quality of writing, and quality of content.

While the template for the previous assignment gave a number of suggestions as to what kinds of experiments to run and what data to provide, for this assignment all such decisions are left to you. Always keep in mind, the goal is to be clear and informative. You will be graded on your decisions about what kinds of experiments to perform, how you present your data, and your discussions about the causes and meaning of the data. As always, be sure to properly cite all resources used. At a minimum, you should cite the course lecture notes and the text book.