

CS51 Project 2b: Rolling Down the Hill

Due: Friday, April 17th 2009 at 4:59 PM

Total Points: 51 (including 10 style points)

1 Assignment Overview

In this portion of the project you will use least-squares to estimate properties of a graph and make decisions based upon those estimates. “Powerups” fall from the sky following some probability distribution and have a size property dependent on the node they appear at – it is up to you to collect as many large powerups as possible over a short period of time.

2 Solutions For Project 2a

The code you wrote for finding the shortest path between two nodes in the graph may be helpful for the final section of this assignment. If you would like to look at our solution for Dijkstra’s algorithm, you can find it in the file “dijkstra.ss”, which we will make available on the website. This version is included by default in scheme file, however, we encourage you to use your old code!

3 Introduction to the World

Again we have given you some code to initialize an engine with the game rules for this week. If you run `proj2b.scm` after downloading it you should see a graph appear with several agents. The yellow agent represents the agent you will write code for. The other agents are “powerups” that appear on the graph and wait for several turns or until they are picked up by an agent before disappearing. After some time they will reappear again at a random node. (The fact that they “fly” onto the graph is purely aesthetic – your agent cannot detect them until they arrive at a node on the graph).

Powerups appear with some probability distribution over the nodes in the graph. Each node stores the size of the last powerup that appeared at that node and a number alpha.

When a new powerup appears at that node, the new powerup will be the size of the last powerup at that node plus alpha, with some random variation.

Your job will be to create an agent that can observe the graph over time and estimate both the alpha values and the probability that a powerup will fall at each node. Using this information, your agent will attempt to maximize the total size of the powerups it picks up while minimizing the number that it does not collect in time.

Although we are quite proud of the sound effects in the game, to turn off sound change the `#t` in the following lines of code to `#f`:

```
(define p1 (make-object powerup% #t))
```

4 Least Squares

(15 points total)

First, you are going to implement the linear least squares regression that we discussed in lecture. You can assume that the data is “good enough” to use straightforward least squares. That is, you don’t need to worry about removing outliers before computing the regression. You’re getting to be experienced coders by now, so we’re leaving the design and implementation of least squares up to you, but we’d like you to implement it by means of iterative search rather than solving for a closed form solution. Your data should be a list of points with x and y coordinates, and your function should return some representation of a best-fit line.

You’ll probably want some representation for a line (recall that in class we used slope-intercept to represent a line), an error function (look at the lecture slides for the formal definition of least-squares error), and a function that does the actual iterative search. Think of this last function as searching over a space of lines for the line that minimizes the error function. How many parameters define a unique line in your representation? This is the number of dimensions in the space you’re searching over.

We suggest that you write an auxiliary function (or set of functions) that perturbs the parameters of a line a little bit in some direction. Choose any perturbed line that results in a smaller error function than the current line, and simply repeat this process to find the minimum. Remember that the least squares function has a unique minimum, so if you find a point where the error function stops going down and begins to go up no matter how you perturb your line, you’ve found the global minimum of the function and can stop your search.

Exercise 1.

15 points

Implement linear least squares regression. Define all needed structures and functions. Make sure to comment your code and include test cases.

5 Editing the Agent

(25 points total)

Now we are ready to begin editing the agent to attempt to intelligently pick up powerups.

Exercise 2.

25 points

(a) [10 points]

Implement the function `collect-data`. This function will be called every time new powerups appears or disappears from the game graph. It takes as an argument an association list that has an entry for each *new* powerup, and maps nodes to the powerup agent that is currently at that node. The purpose of this function is to collect the history of powerup drops on the graph so that you can complete the following exercises. We leave the structure of the data you store up to you.

(b) [5 points]

Define a new public function `estimate-alphas`, which returns a dictionary that maps nodes to the estimated alpha value at that node. Recall the meaning of the alpha value: if a powerup of size N appears at a node with alpha value $alpha$, the next powerup to appear at that node will have a size approximately $N + alpha$.

(c) [5 points]

Define a new public function `estimate-dist`, which returns a dictionary that maps nodes to the estimated probability that a powerup will land at that particular node. Note that your agent's best estimate for the probability that a powerup will next appear at node n is just the fraction of all the powerups that have ever appeared that appeared at n . You should assume that if no powerups have fallen on the graph yet, the next powerup has an equal probability of appearing at each node.

(d) [5 points]

Using at least one of the functions from parts (b) and (c), override `new-goal-node` so that your agent does well in this simulation (as a rough indicator on the 3 node sample graph, a good agent should pick up more powerup-points than it misses when run for a few thousand steps). You do not need to play optimally or even very well, so long as you have a reasonable explanation for your strategy and use at least one of your `estimate-alphas` and `estimate-dist` functions. Write a comment explaining your strategy. You may also find it useful to import code from Project 2a. Doing well in this section of the project may help you create a better agent in the next two parts of the assignment.

6 Testing Your Code

We have supplied some code for you at the bottom of `proj2b.scm` that will help you test the correctness of your agent's functions, or run the simulation without the GUI. For instance,

you may run some of these functions and check that the alpha values and probability distributions that you estimate approximately match the actual values in the engine.

7 Finishing Up

(1 point total)

Exercise 3.

1 point

Please tell us how much time you spent on this problem set. You may use this space to add any thoughts or questions you want to send to your TF as well, if you so desire.

8 Supplied Code

What follows is a brief description of the code that we supply that you will need to override or that may be helpful in writing this assignment.

8.1 Dictionaries

The dictionary module we provide has changed a little since last project. `dict-get` and `dict-put` have been changed to use builtin Scheme functions `dict-ref` and `dict-set` instead of `hash-ref` and `hash-set`. Now these functions will work on multiple types of dictionaries, for instance, association lists as well as hash-tables.

We also have removed the contract specifications in the dictionary module, as the builtin Scheme functions already define their own. Now you can, for instance, supply an optional third argument to `dict-ref/dict-get` that indicates what to do if the key being searched for is not in the dictionary. For more information, look up these functions in the Scheme documentation.

8.2 Game Specific Functions

The following functions may be useful in your solutions.

```
;; Anytime you have an argument "query-obj", it is of type pickup%
(class pickup%
  ;; Returns a dictionary mapping a node to the powerup% at that
  ;; node.
  (define/public (get-powerups) ...)
  ...
)
```

```

(class powerup%
  ;; Returns the current size of the powerup.
  (define/public (get-size) ...)
  ...
)

;; Overrideable functions in your agent class
(class graph-agent%
  ;; This is called the first time you add an agent
  ;; to the engine. You may use it to initialize any
  ;; game specific data for the agent. Using this function
  ;; is optional.
  (define/public (init-agent query-obj) (void))

  ;; Used to customize the agent's looks. bm-dc stands for bitmap-dc%
  ;; init-bitmap is called when the agent is first created, while
  ;; draw-bitmap is called everytime the agent is drawn.
  (define/public (get-default-pen) (make-object pen% "GREEN" 2 'solid))
  (define/public (get-default-brush) (make-object brush% "GREEN" 'solid))
  (define/public (init-bitmap bm-dc) ...)
  (define/public (draw-bitmap bm-dc) ...)
)

;; Functions written by you
(class my-agent
  (define/override (new-goal-node query-obj) ...)
  (define/override (collect-data powerups) ...)
  (define/public (estimate-alphas) ...)
  (define/public (estimate-dist) ...)
)

```