Stochastic Processes 160B, Project

Ted Tinker, 3223468

March 15, 2017

Summary

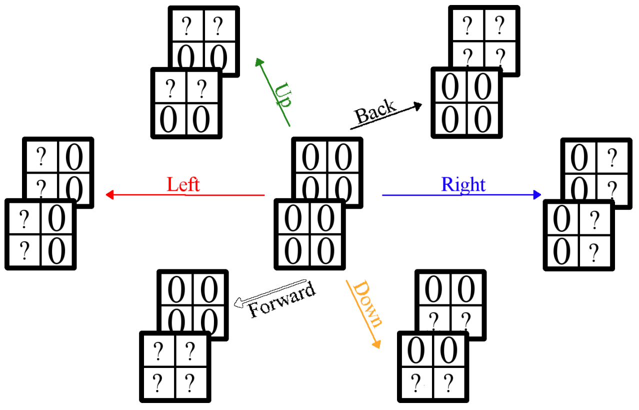
For this project, I elected to simulate a Markov Decision Process (MDP). A Markov Decision Process with a state-space S is like a Markov Chain, except each state $s \in S$ has a list of *decisions* A_s, and the probability of moving from state s to state s' given decision $a \in A_s$ is given by probability function $P_a(s, s')$ (a generalization of the Markov Chain's transition matrix). A reward function $R_a(s, s')$ assigns a value to moving from s to s' given decision a, and decisions are chosen by a Markov Decision policy $\pi(s)$, based only on the current state s. Policies can be made to maximize or minimize the expected value of the reward function (depending on whether the reward function represents a value or a cost).

Motivation

These decision-making processes are not only fantastic extensions of the skills we've learned in PSTAT 160A and B, they are also invaluable in Machine Learning and related fields like Dynamic Programming. Therefore, they have use in Economics, Automation, and Business Analytics. I hope to expand this project with a Summer Undergraduate Research Fellowship!

Methods

In my Markov Decision Process, 256 states represent all possible $2 \times 2 \times 2$ arrays with entries 0 or 1; they are like cubes with 0 or 1 on each corner. The Markov Decision policy relates each state to a decision: *forward, back, up, down, left,* or *right.* At each step, the state changes to a random state one 'move' in that direction. One state is mapped below:



Every decision shifts four entries to the opposite half of the cube while revealing four random 0s and 1s. Thus, there are $2^4 = 16$ next states possible, chosen uniformly at random. This establishes the states $s \in S$, the decisions $a \in A_s$, and the probability function $P_a(s, s')$.

Let the reward function value states containing many 1s, while disliking states containing 0s. If $n(s)$ describes the number of 1s in state $s \in S$, define $R_a(s, s') = 2n(s') - 8$, the difference between the number of 1s and the number of 0s in state s'.

We will evaluate two Markov Decision methodologies: $\pi_r(s)$ will relate each state to a random decision to be made; while $\pi_s(s)$ will relate each state to the decision whose expected reward is highest. In tracking over multiple trials the cumulative reward over time with each policy, $R(t|\pi)$, we verify that $\mathbb{E}R(t|\pi_r) < \mathbb{E}R(t|\pi_s)$.

Implementation

My Python code generates 256 nodes of a custom class, each having six lists of 16 nodes which could be *in front of, behind, above, below, right,* or *left* of the node.

Then it generates policy $\pi_r(s)$, which associates each node with a random direction, and $\pi_t(s)$, which associates each node with the direction of the node's list containing the most 1s (or a maximal amount of 1s, in the case of a tie).

Finally, my code begins in the node representing a cube with 0 on each corner and increments the process step-by-step based on either policy π_r or π_t. By printing the time and cumulative reward for each step, we can compare the two policies and see which performs better.

Results

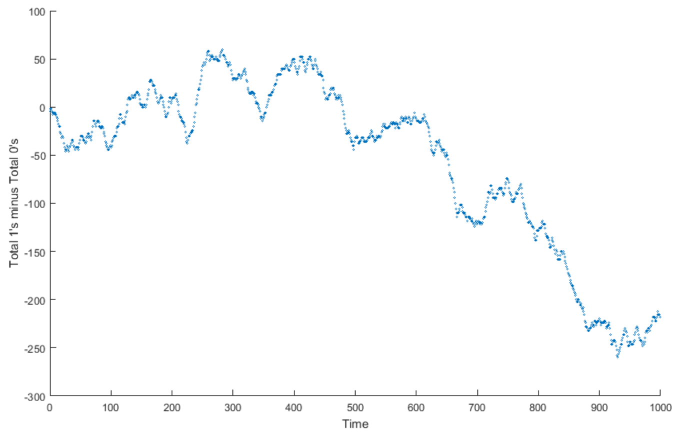


Figure 1: $R(t|\pi_r)$

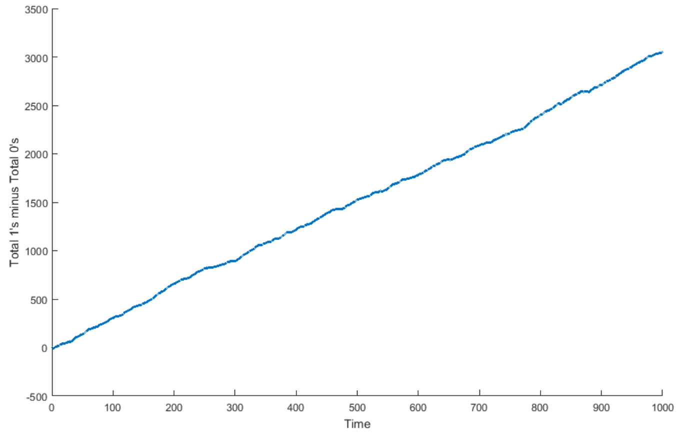


Figure 2: $R(t|\pi_s)$

Discussion

When using π_r, the cumulative reward function appears Brownian (which makes sense, as it is essentially a random-walk on the state-stace). 1,000 trials of different randomly generated Markov Decision policies had an average reward of -7.376 at time $t = 1,000$, which is close to 0, and the characteristic Brownian dips and spikes are present.

Using π_s produces a plot with striking linear correlation; 1,000 trials produced an average reward of 3083.218 at $time = 1,000$, indicating the process spent most of its time in states with six, seven, or eight 1s.

However, there are still issues with my policy $\pi_s(s)$. It only looks one step into the future, for example; the ideal policy would sum the expectations for many steps or infinitely many steps (modified by a discount factor between 0 and 1 so the overall expectation converges).

At the same time, moving to a random state is a poor abstraction for movement in a three-dimensional array. If we used $\pi_s(s)$ to direct a $2 \times 2 \times 2$ window through a cubic lattice of 0s and 1s, it would quickly become trapped alternating between a few suboptimal states, not collecting as many 1s as it could. In order to dislodge the Markov Decision Process, we must alter our model to remember some information about its past states.

Combining these notes, we can imagine a $2 \times 2 \times 2$ window piloted by a more sophisticated policy. The window moves to contain as many 1s as possible, accounting not only for the infinite sum of reward expectations of surrounding cubes discounted by distance, but also for its memory of the entries it has observed.

As a SURF Fellow, I would implement a system like this in Python, hoping to report any interesting emergent behaviors and valuable applications. As a stretch goal, I would like to make an website where people at the event could move the $2 \times 2 \times 2$ window themselves, and try to beat my algorithms.