

ON-POLICY MONTE CARLO CONTROL OF A RACE TRACK SIMULATOR

Ryan J. Meuth

Department of Electrical and Computer Engineering
University of Missouri at Rolla

ABSTRACT

This report details the implementation of the On-Policy Monte Carlo control algorithm on a race-track simulator with discrete states and actions. Results are given for the "optimal" policy after 1000 iterations of the algorithm.

1. INTRODUCTION

One of the simplest methods for finding the optimal policy in a unknown environment is the Monte Carlo Algorithm. This episodic algorithm explores the state-space, using a policy and receiving a reward R for the action taken in each state. These rewards are averaged, and a new policy is constructed from the resulting information. The policy that the algorithm follows while exploring may be either the current best policy (on-policy MC control) or an arbitrary exploration policy (off-policy MC control). Here, the On-Policy Monte Carlo Control algorithm is implemented in a Race Track Environment.

2. ENVIRONMENT

Consider driving a race car around a turn. You want to go as fast as possible, but not so fast as to run off the track. In this simplified racetrack, the car is at one of a discrete set of grid positions. The velocity is also discrete, a number of grid cells moved horizontally and vertically per time step. The actions are increments to the velocity components. Each may be changed by +1, -1, or 0 in one step, for a total of nine actions. Both velocity components are restricted to be nonnegative and less than 5, and they cannot both be zero. Each episode begins in one of the randomly selected start states and ends when the car crosses the finish line. The rewards are -1 for each step that the car stays on the track, and -5 for if the car tries to drive off the track. Actually leaving the track is not allowed, but the position is always advanced by at least one cell along either the horizontal or vertical axes. With these restrictions and considering only right turns, all episodes are guaranteed to terminate, yet the optimal policy is unlikely to be excluded. To make the task more challenging, we assume that one half of the time steps

the position is displaced forward or to the right by one additional cell beyond that specified by the velocity.[1]

3. ON POLICY MONTE CARLO CONTROL

Monte Carlo control algorithms are simple, episodic learning algorithms that require only *experience* from their environment to discover an optimal policy. Off Policy Monte Carlo algorithms utilize an arbitrary or ϵ -soft policy to generate episodes for optimal policy updates, while On Policy Monte Carlo algorithms use the current optimal policy estimate to generate episodes.

For this problem, the following On Policy Monte Carlo Control algorithm was implemented[1]:

Initialize, for all $s \in S, a \in A(s)$:

$Q(s, a) \leftarrow$ arbitrary

$\pi(s) \leftarrow$ arbitrary

$Returns(s, a) \leftarrow$ empty list

Repeat for 1000 Iterations:

For each start state:

Generate an Episode.

For each s, a appearing in the episode:

$R \leftarrow$ return following s, a

Append R to $Returns(s, a)$

$Q(s, a) \leftarrow$ Average($Returns(s, a)$)

For each s appearing in the episode:

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

4. SIMULATED TRACKS

For this problem, two tracks were simulated, shown in Figure 1 and Figure 2.

5. EXPERIMENTAL RESULTS

The algorithm was run for 1000 iterations on each of the simulated tracks. An example solution episode for track 1 is shown in Figure 3. The colored blocks denote the path of the agent through the track. Here we can see this is not exactly the optimal path, as the distance between blocks along the middle of the path is small, meaning that the velocity of

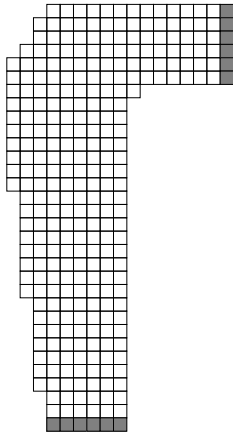


Fig. 1. Track 1 for On Policy Monte Carlo Control

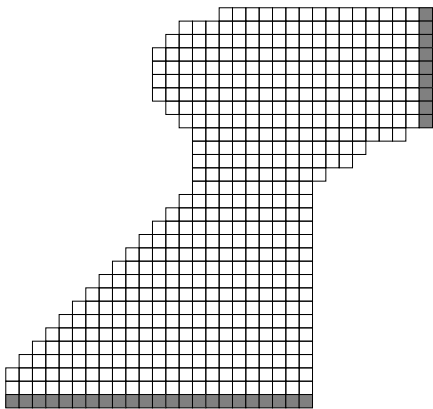


Fig. 2. Track 2 for On Policy Monte Carol Control.

the agent at that time was far from maximum. Figure 4 shows a solution episode that is much closer to optimal. The blocks along the path are distantly spaced, showing that the velocity at those points is much higher.

Similar performance can be shown for Track 2. Figure 5 shows a sample solution path for Track 2. Solution paths for track 2 must be more interesting as the turn in track 2 is much sharper, and two turns may be necessary for some of the leftmost starting states.

Figure 6 shows a very non-optimal path discovered during the course of the algorithms' execution. Note that the path seems to 'bounce' off of the left edge.

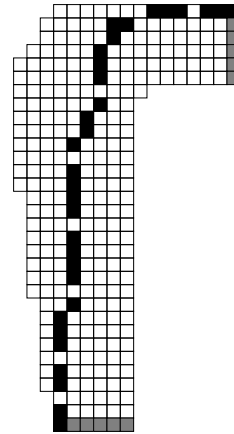


Fig. 3. Sample Solution Episode for Track 1.

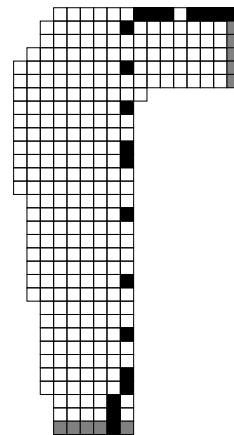


Fig. 4. A nearly optimal solution episode for Track 1

6. CONCLUSIONS

Though some of the paths described above appear to be very close to optimal, it is not likely that they are truly optimal.

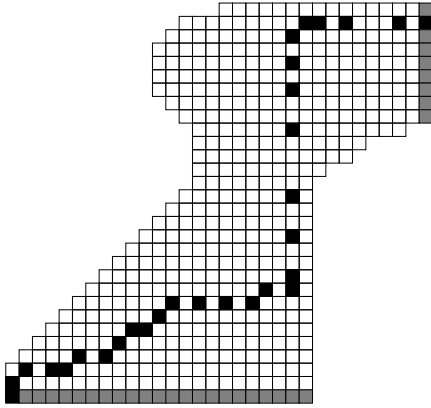


Fig. 5. A Sample Solution Path for Track 2

This may be a shortcoming of the algorithm, as since it is a deterministic on-policy algorithm, there is a danger of premature convergence to a non-optimal path. This may be alleviated by initializing the Q array to an arbitrary high value, and replacing the values with real-valued averages once visited. Alternatively, an ϵ -soft algorithm might be used. Both of these methods would increase exploration, and a policy that is much closer to optimal is likely to resort.

7. REFERENCES

- [1] Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning" *The MIT Press*, 1998.

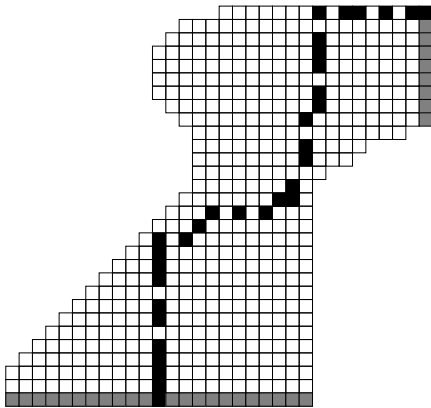


Fig. 6. A non-optimal path on Track 2