Introduction to Machine Learning, Programming Assignment 7

Reinforcement Learning: The Racetrack Problem

Julie P. Garcia

Johns Hopkins University

Abstract

Reinforcement Learning is a dynamic programming strategy that is used to solve problems that involve choosing an optimal sequence of events by allowing an agent to explore and learn it's environment. Generally, this is done through a set of trial-and-error experiments, where rewards are given for actions that move the agent closer to its goal. A Markov Decision Process can be used when there exists sufficient knowledge of the environment model. When the environment requires more exploration, other methods such as SARSA and Q-learning may be utilized.[1,2] In this paper, we compare a value iteration algorithm with the SARSA algorithm and show that the value iteration algorithms always converges in the optimal policy, however, the time complexity for large environments increase exponentially. While using the SARSA algorithm for reinforcement learning, it may not always converge on the optimal policy, but runs significantly faster and converges with a nearly optimal policy.

Problem Statement & Hypothesis

In this paper, we demonstrate that utilizing a value iteration algorithm for reinforcement learning will always converge to the optimal policy, however, the time complexity of the algorithm grows exponentially with the size of the environment. To solve this issue, value iteration was compared with SARSA as an alternative that will converge close to the optimal policy and run significantly faster than the value iteration.

Algorithms Implemented

Reinforcement Learning algorithms are based on an agent that interacts with its environment in order to learn a policy that will help it to achieve its goal. At each time step in the process, the agent takes in its environment and tries to make the optimal decision based on its current state and some reward or penalty given back from the environment. The goal of the algorithms implemented for reinforcement learning is to learn the optimal policy.

Markov Decision Process

Markovian models are models that base their next action solely on the step that came before the current state. The Markov Decision Process consists of five elements. S is the set of all possible states in the problem. A is the set of all possible actions. T is the state transition function defined as the probability of transitioning to another state *s'*, given that we are in state s and we perform action a. R is the reward function which tell us what the reward is when we take action a, while in state *s*. The last element is the discount factor, *gamma*. The agent will go through the decision process by choosing the next action that gives the highest reward from the current state that it is in. The state will then change based on the probability distribution defined by our model.

Value Iteration

The value iteration algorithm computes the optimal policy by iterating until the values converge. The algorithm (shown below), starts by initializing the q values to zero. The loop repeats until V(s) converges. In each loop the values for the combination of each state and action pair are updated by using the Bellmen Formula. The argmax of the values are taken to find the optimal action at the current state.

```
Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
        $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge
```

SARSA Algorithm

Unlike with Value Iteration, when using a SARSA algorithm for reinforcement learning, you are not explicitly given the model ahead of time. SARSA involves a series of iterations of trial-and-error and the best policy is learned when the agent explores it's environment and continually updates it's policy. SARSA is considered an on-policy algorithm, which means that it estimates the q-value for the current policy and updates it accordingly. In SARSA the q-values are arbitrarily set in the initialization step. As, you can see below, the algorithm then iterates until it reaches its terminal state or goal state. In each iteration, an action is chosen, the immediate reward (or punishment) is collected and then the successive state is calculated from the action chosen. In this paper, we used an epsilon greedy strategy to choose successive actions. After each action is chosen, the q-value for the state, action pair with the SARSA update rule.

```
Initialize all $Q(s,a)$ arbitrarily
For all episodes
  Initalize $s$
  Choose $a$ using policy derived from $Q$, e.g., $\epsilon$-greedy
  Repeat
    Take action $a$, observe $r$ and $s'$
    Choose $a'$ using policy derived from $Q$, e.g., $\epsilon$-greedy
    Update $Q(s,a)$:
      $Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma Q(s',a') - Q(s,a))$
    $s \leftarrow s'$, $a \leftarrow a'$
  Until $s$ is terminal state
```

Experimental Approach

We first created a race track simulator that took in three tracks as input, similar to the ones shown here. The test-track was a small sample track used to build the algorithm and run quick tests on the simulator. The experiments were also run on the L-track.txt and R-track.txt.

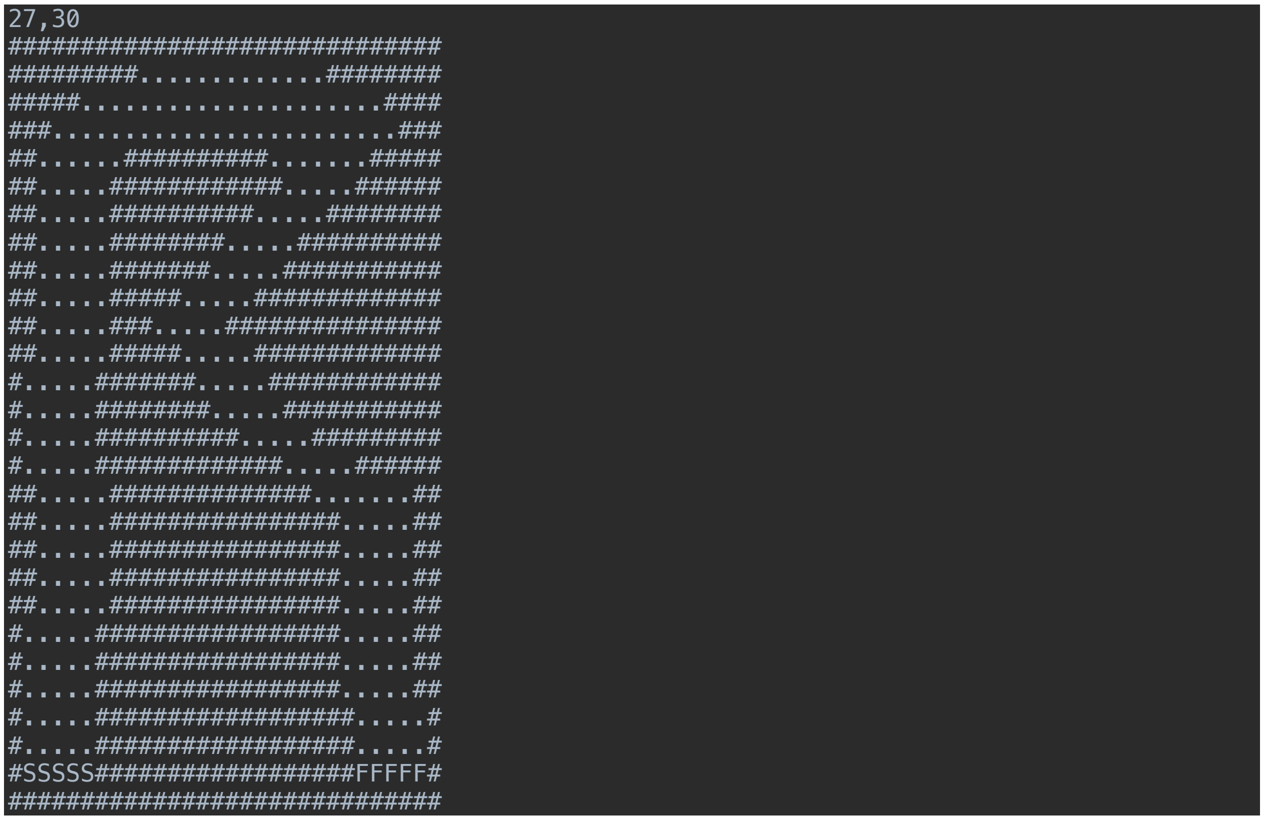
Test-track.txt

```
6,6
######
#FFFF#
#....#
#S...#
#S...#
######
```

L-track.txt

```
11,37
#####################################
################################FFFF#
################################....#
################################....#
################################....#
################################....#
#S..................................#
#S..................................#
#S..................................#
#S..................................#
#####################################
```

R-track.txt



```
27,30
##############################
#########.............########
#####.....................####
###........................###
##......##########.......#####
##.....############.....######
##.....##########.....########
##.....########.....##########
##.....#######.....###########
##.....#####.....#############
##.....###.....###############
##.....#####.....#############
#.....#######.....############
#.....########.....###########
#.....##########.....#########
#.....#############.....######
##.....##############.......##
##.....################.....##
##.....################.....##
##.....################.....##
##.....################.....##
#.....#################.....##
#.....#################.....##
#.....#################.....##
#.....##################.....#
#.....##################.....#
#SSSSS##################FFFFF#
##############################
```

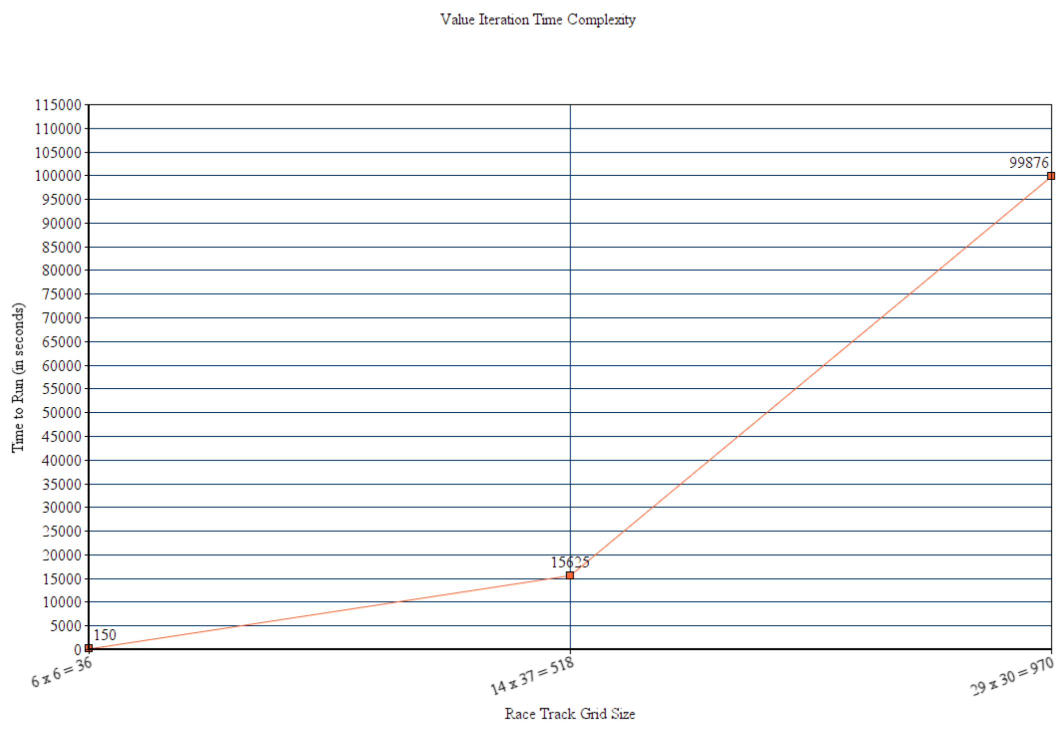
The track simulator moves a race car from the start line to the finish line by following a sequence of actions. The set of possible actions are a{x,y} ∈ {−1, 0, 1}. Where a is the acceleration. At each step the acceleration is applied to the current velocity (initialized to 0, 0) by adding the acceleration to the velocity and then moving the car based on the current velocity, by adding the velocity to the current (x, y) position. The velocity at any state can never exceed [+-5, +-5]. A non-deterministic element is added with the probability of acceleration succeeding as 80% and 20% of not exceeding. Any attempt to move without accelerating will succeed 100% of the time.

If the car runs into a wall, denoted by the '#' character, or attempts to move off the track, two strategies were implemented. One puts the car back on the track where it left, the other puts the car back at the start. The latter applies a more severe penalty for crashing. In both cases, the velocity is set back to (0, 0).

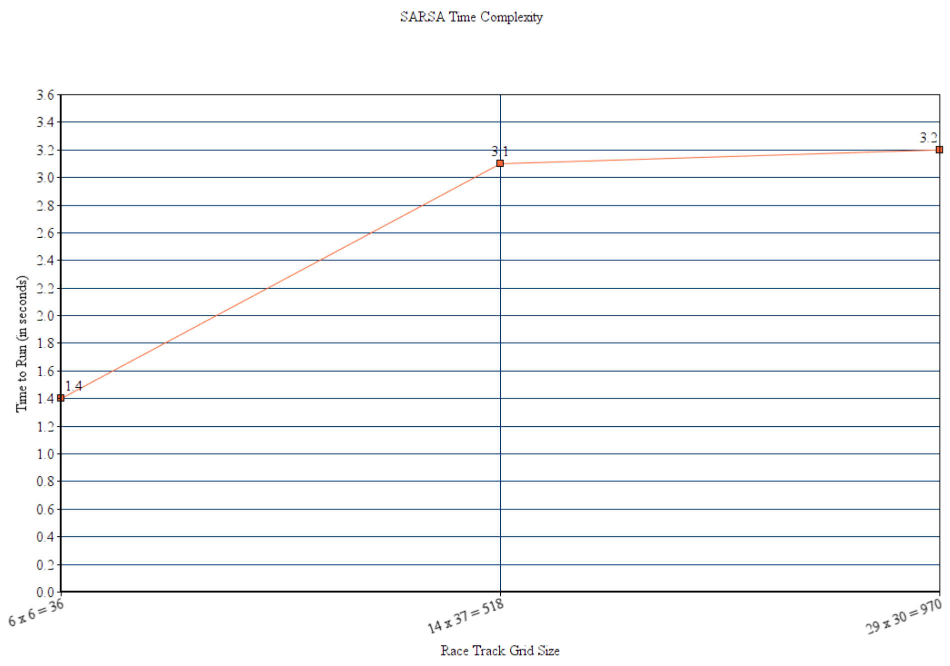
Value iteration was run on each of the tracks to build the optimal policy, and then 10 experiments were run on each track for each strategy (both restart_on_crash and continue_on_crash were run for each of the 10 experiments). The SARSA algorithm was run 10 times on each track (with both crash strategies) and the results were compared with the results of the value iteration algorithm. Both time complexity and accuracy were compared for both algorithms.

Results

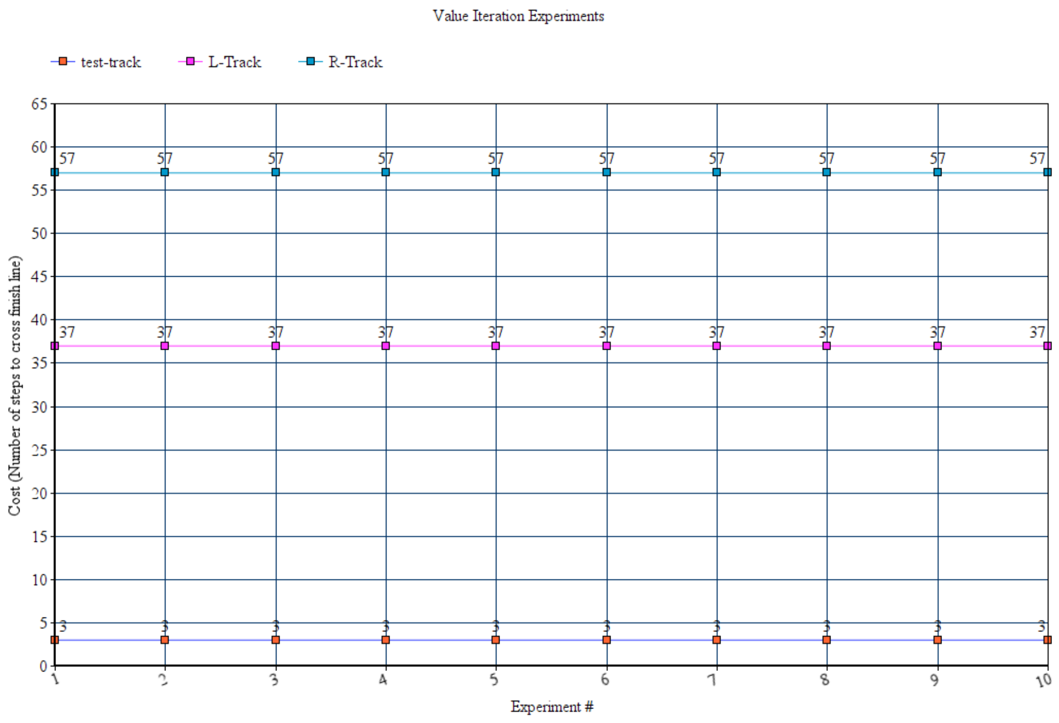
The first set of results demonstrates the time complexity of the two algorithms compared. The value iteration algorithm time to run grew exponentially with the size of the environment, which in this case was a race track grid. Keeping in mind that once the model was created with value iteration, it could be used again for subsequent runs of the race track simulator.

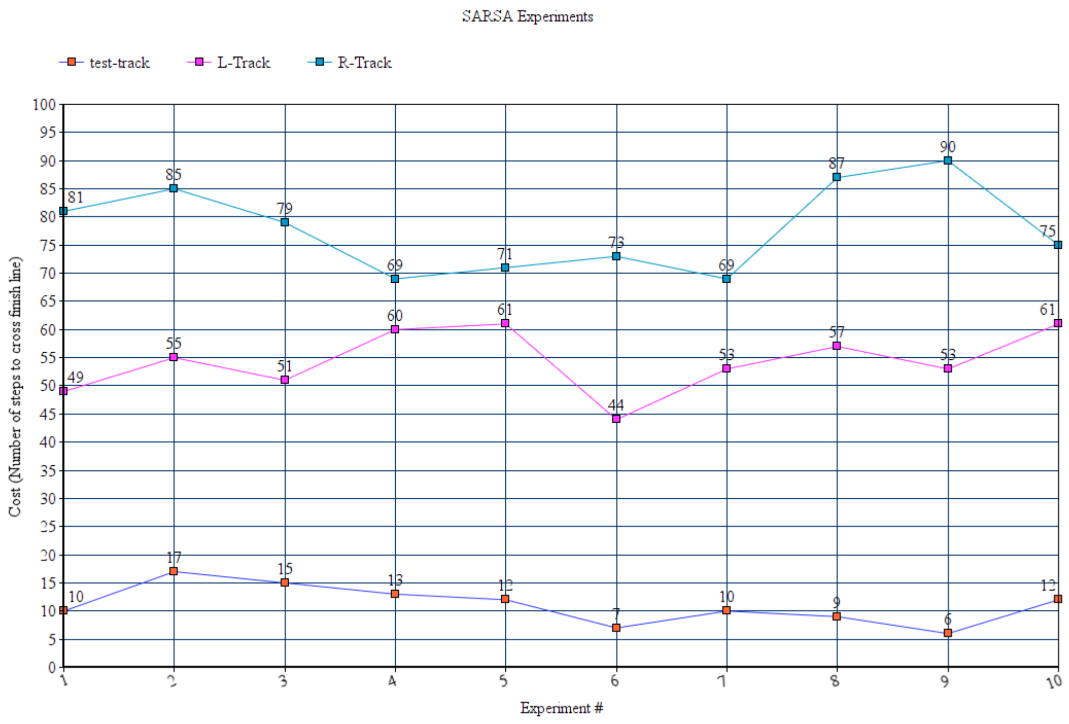


However, the time to run the SARSA algorithm grew relatively slowly as the environment grew.



In the next set of results, we see that the value iteration algorithm always converged on the optimal policy, therefore subsequent runs of the car on the race track simulator produced the exact same optimal results. The SARSA algorithm, however, produced different results with each run while still providing a good enough strategy.





Summary

In summary, when the environment is known and the set of possible states is relatively small, utilizing a value iteration algorithm for reinforcement learning is the best choice, because it will always converge on the optimal policy. As the environment and number of possible states, action pairs grow larger, an alternative algorithm such as Q-learning or SARSA should be used for reinforcement learning problems.

References

1. Alpaydin, E. (2014). *Introduction to machine learning*. Cambridge, MA: The MIT Press.
2. Gosavi, A. (2009). Reinforcement Learning: A Tutorial Survey and Recent Advances. *INFORMS Journal on Computing*, *21*(2), 178–192. doi: 10.1287/ijoc.1080.0305