Reinforcement Learning:

A Comparison of Value Iteration and Q-learning Algorithms in a Markov Decision Process

Corbin Rosset

May 4, 2015

Abstract

In this paper we apply two different reinforcement learning algorithms to a Markov Decision Process (MDP). The first is value iteration (VI) with full knowledge of the environment's stochastic state transition model, the second is Q-learning (Q), which does not require the model. We found that VI converges to a near-optimal solution more quickly, but given the right input functions and parameters, Q yields slightly better performance, especially when the reward function is particularly harsh. Both agents exploited interesting "loopholes" in the environment to achieve better scores. We recommend using VI to solve an MDP whenever enough information exists, even if just as a benchmark, otherwise Q may be the only alternative. And, given enough computational resources, a correctly tuned Q agent will eventually yield optimal performance most of the time. We also discuss some techniques decrease Q's runtime in large state spaces.

1 Introduction

Reinforcement learning addresses the question of how a rational artifical intelligence agent should interact with an environment that it cannot control to achieve a goal. An environment is decomposed into (possibly infinitely many) states $s \in \mathcal{S}$, in each of which an agent may choose an action a from a predefined set (again possibly infinite) of actions $\mathcal{A}$ to reach another state s' [2]. For simplicity, all states are fully observable, so s' is known. Unlike supervised learning, the agent is not told the correct action in a given state because such information, known as a policy π, is too difficult to ascertain directly from complex environments [3]. Instead, an agent must learn the policy given only a scalar "reward" $r(s)$[1] as feedback upon arriving in state s' from previous state s under action a. The objective is then to find a path through the state space that maximizes reward in pursuit of a goal state [1].

We assume that an environment is modeled by a Markov Decision Process (MDP), which is "a sequential decision problem for a fully observable, stochastic environment

[1] a Q agent defines reward for a state-action pair, $r(s, a)$

with a Markovian transition model and additive rewards" [3]. We have already defined the state and action spaces, as well as the reward function. In our case, actions are taken at discrete time steps, but this is a simplification. A MDP is also defined by a stochastic transition model $\delta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \in [0, 1)$ between states, because environments do not always allow agents to execute their intended actions – they are unreliable, so a fixed action sequence has a high probability of missing the goal [1]. We must define a distribution over the possible destination states: $P(s'|s, a) = \delta(s, a)$ exists for each s' reachable from s by action a. Similarly, rewards are usually stochastic as well: a $P(r'|s, a)$ is associated with each s', but in our case rewards are deterministic [2]. A crucial assumption of an MDP is that δ depends only on the previous state and the action taken, and not the prior history of earlier states [3].

The final component of the environment is a value or utility function $U : \mathcal{S} \rightarrow \mathbb{R}$ which is entirely separate from the MDP's reward function. The utility of a particular state s, denoted $U(s)$ encodes the rewards of all previous environment histories which result in s rather than just the reward of s itself. $U(s)$ is not intrinsically a property of the state, but rather the agent's best guess as to the state's usefulness given its prior sequence of states. As the agent explores the state space more fully, arriving at the same place from multiple angles, utility values will converge to their true values, and the policy will become more optimal [3].

With full knowledge of the transition model, δ, navigating the MDP reduces to a path planning problem that can be solved by VI. However, the agent need not know the transition or reward probabilities to learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ which tells the agent the best action to take in any state. Without it, the best policy, π^*, can be learned or infered by iterative trial and error by algorithms like Q-learning. An agent can "explore" a state randomly to accumulate rewards to update $U(s)$ and $\pi(s)$, or if the agent is confident the information collected on its current and surrounding states accurately reflect the true utilities, the agent can assign the best action a in that state, $\pi(s) \leftarrow a$, and focus effort on less explored states [1]. If the "reward" of lingering in the state space is negative, then exploration is costly. Yet, if too little information about the state space is collected, then "exploiting" the known information will result in a suboptimal path to the goal. This is known as the exploration versus exploitation tradeoff; a well-conceived exploration function can immensely improve the accuracy of the utility values to maximize long-term rewards. Irrespective of how the agent approximates it, the optimal policy returns the action that maximizes the *expected* utility of the next state. An agent that follows an optimal policy to the goal will have had a history of locally optimal action sequences. Note it is not necessary to calculate utilities exactly; as long as the truly superior states have the highest utilities, then an optimal path will be found.

In the next section we delve into the details of converging to the correct utility values, and how to construct the optimal policy. We also describe the VI and Q algorithms at length. In Section 3 we discuss the test cases and parameters on which the algorithms were run, and compare the results of those tests in Section 4. We then explain why the results occured in the discussion section, and what situations a user interested in applying these algorithms should consider.

2 Finding Optimal Policies in a Markov Decision Process

In our experiments, we assume an "infinite horizon" so that there is no time limit after which the agent "dies". If there was a time limit, or finite horizon, then depending on what the time pressure is, the optimal action in a given state may change [1]. The convenience of an infinite horizon is stationarity of the optimal policy, meaning it depends only on the current state and not time. However, since the goal is to maximize the sum of the rewards along a path or state sequence, an infinite horizon may allow an infinitely long path with infinite utility, renders comparing paths meaningless [3]. We use the concept of "discounted" rewards and geometric series to force path rewards to a finite value. Given a discount factor $\gamma \in [0, 1)$, the reward of an arbitrary state sequence $\langle s_0, ..., s_n \rangle$ is

$$r(\langle s_0, ..., s_n \rangle) = \sum_{t=0}^{n} \gamma^t r(s_t, \pi(s_t))$$

which is bounded by $\frac{R_{max}}{1-\gamma}$ for R_{max} the largest reward in magnitude seen along the path [1]. The optimal policy for a state sequence starting in state s_0 is then

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))$$

for all possible policies, which maximizes the discounted rewards of all states subsequent to s_0 [1]. In fact, the optimal policy is independent of the start state[2]. We now define the true utility U^* of a state s as the *expected* sum of discounted rewards collected by executing an optimal policy in s and all subsequent states in the sequence:

$$U^*(s) = E\Big[\sum_{t=0}^{n} \gamma^t r(s, \pi^*(s))\Big] = \sum_{s'} \gamma \delta(s, a) U^*(s') \tag{1}$$

where the sum is taken for all s' adjacent to s. Notice that the expected reward of a state sequence of arbitrary length beginning in s can be described by the utilities of those states immediately adjacent to s because utility encodes information about all subsequent states. Of course, any non-optimal utility found by executing a non-optima policy is identical to the equation above, just without the asterisks. We similarly define the utility of a state sequence beginning in s_0 as the *expected* utility of s_0 and all subsequent states visited by executing policy π as

$$U(\langle s_0, ..., s_n \rangle) = E\Big[\sum_{t=0}^{n} \gamma^t r(s_t, \pi(s_t))\Big] = \sum_{t=0}^{n} \gamma^t \delta(s, a) U(s_{t+1}, \pi(s_t)) \tag{2}$$

where δ returns the probability of arriving in the next state, s_{t+1} under action a [2].

[2] yet the action sequence is not

Notice the recursion in these definitions – the utility of state is dependent on the utilities of surrounding states, for every state. This has significant implications on algorithm design and runtime. Since the optimal utility can only be approximated by sampling during pseudo-random walks, VI and Q-learning algorithms are careful not to assume independence of utilities. The purpose of any agent is to estimate with as little error as possible the optimal policy for every state s, which is the action that leads to the adjacent state s' with maximum expected utility.

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s,a)U^*(s') \tag{3}$$

where $U(s')$ is shorthand for the utility of s', one of the possible states reached by a (remember, a stochastic transition model can map an action to many states non-deterministically). We could have added γ as a coefficient for the right hand side, but the equality holds without it [2]. In the next section, we formalize the definition of utility and use it to find the best policy.

2.1 Bellman Equations and Policy Evaluation

We may finalize the relationship between the utility and reward in what is known as the Bellman equation. The utility of a state $U(s)$ is the current reward of being in that state plus the sum of the surrounding states' discounted utilities:

$$U(s) = r(s) + \gamma \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s,a)U(s') \tag{4}$$

Which can be derived from equation (1) above by decomposing the sum into its first term plus the rest of the terms. It has been proven by [2] that for a finite MDP, the Bellman equations uniquely identify the optimal policy. To summarize, the problem of finding the optimal path to the goal state from any start state depends on finding the optimal policy for all states, which reduces to finding the true or nearly-true utilities of each state.

To ascertain the true utility of each of n states, n Bellman equations must be satisified, each with its own $U(s)$ an unknown. Since the Bellman equation contains the *max* operator, the system of n equations is nonlinear. The following two algorithms propose ways to solve for utilities under different assumptions.

2.2 Value Iteration

Assuming that the transition probabilities are known for every action, we can take an iterative approach to update the utilities of each state simultaneously using the Bellman equation until convergence is reached. It is common to initialize utilities to zero, and in each iteration i, update the utility of every state s based on the Bellman Equation (see Algorithm 1):

$$U_{i+1}(s) = r(s) + \gamma \arg\max_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s,a)U_i(s') \tag{5}$$

Algorithm 1 Value Iteration Algorithm from [3]

Input: an MDP with fully specified state space S, transition model δ, and reward function r; discount factor γ; convergence threshold ϵ
Output: a near-optimal policy
Local Variables: U, U' vectors of utilities for every state, real number ρ

procedure VI(MDP, ϵ, γ)
  repeat
    $U \leftarrow U'$, $\rho \leftarrow 0$
    for all states s in S **do**
      $U'[s] \leftarrow r(s) + \gamma \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s,a)U(s')$
      if $|U'[s] - U[s]| > \rho$ **then**
        $\rho \leftarrow |U'[s] - U[s]|$
      end if
    end for
  until $\rho < \epsilon$
  generate π from U by $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} U[s]$ for all s
end procedure

The convergence condition is met in iteration i when the maximum difference $|U_i(s) - U_{i-1}(s)|$ for any state s is less than a convergence threshold, ϵ. As this difference decreases, the error between the true utility and the assigned utility is also assumed to decrease.

Note the agent needn't be present to explore the environment at all. This algorithm is a "pre-processing" step to inform the agent of the policy to follow wherever it may be placed in the environment. The runtime of a single iteration is linear in the size of the state space and the number of transitions; the runtime of the entire algorithm depends on the tightness of the convergence criterion.

2.3 Q Learning

Without any knowledge of the transition model or the reward function, we must learn the policy, as opposed to solving for the policy as VI does. This "model free" approach is the more general scenario because often an agent is placed in an environment with no prior information other than to seek a goal. The agent must explore the environment with pseudo-random walks to sample the effects of the transition model so it can infer utilities for (hopefully) every state [2]. This is much more risky[3] and computationally intense.

In this scenario, the agent learns an action-utility, or Q, policy that returns the expected utility of taking action a in state s, denoted $Q(s, a)$. Other algorithms, however, learn just the utility functions on states or the correct action to take, but these require more information about the environment. A Q agent has the advantage

[3] meaning there are either dangers to the agent itself, or ample opportunities to accrue excessive cost

that it does not need to know the transition probabilities, δ, associated with its actions. While this generalizes to nearly all environments, the agent is unable to "look ahead"; in fact, the Q-learning algorithm can only back-propagate information to the previous state, which severely hinders runtime [3].

Even with knowledge of some approximate utilities for some or all of the states, a greedy or purely "exploitative" agent will seldom converge to the optimal policy because it most likely only has partial information of the true transition model. In the extreme case, a greedy agent may stick to the first policy that reaches the goal, even if that policy is suboptimal, and never investigate whether a better policy exists. To avoid such a scenario, we force the agent to explore other states, even if it is expensive, because we may significantly regret not having found a better policy. How much to explore is a canonical problem known as the "exploration-exploitation" tradeoff [2, 1]. It has been proven (by [3]) that in order to ascertain the optimal policy for every state, each state must be visited an infinite number of times. Since this is not practical, a Q agent estimates[4] the optimal state-action value function Q^*, which is related to the optimal policy:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a) \tag{6}$$

$$U^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) \tag{7}$$

The best policy is the action associated with the highest Q value for that state, and the optimal utility U^* for that state is the value Q takes on for the best action [2]. The algorithm samples a new state s' by either exploiting the currently known best action or exploring a new state according to an exploration function, f, which depends on the number of times that state-action pair has been visited, $N(s, a) \in \mathbb{N}$.

Upon visiting a new state, the agent retroactively updates the Q-value for the previous state and the action taken by a modified Bellman update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(N(s, a))\big[r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)\big] \tag{8}$$

where the learning rate, α is a function of $N(s, a)$, $\alpha : \mathbb{N} \to \mathbb{R} \in (0, 1)$ (which is usually monotonically decreasing). It is difficult to find the correct α function, but it should weight updates to less-visited state-action pairs more heavily, because the initial occurances of (s, a) are more likely to be the result of very different random walks. Again, we would like to arrive at (s, a) from as many different paths as possible, and those paths are most likely to be different earlier in training when the agent is highly explorative.

Once α is set and $Q(s, a)$ is updated, the exploration function, f, will then recommend the action for the next state-action pair based the best utilities. Since an agent gains more information visiting relatively unexplored states, the exploration function will favor them by artificially boosting their utility. State-action pairs whose visitation, $N(s, a)$ is less than a threshold, N_e, will have their Q-value set to an unrealistically high yet permissable reward[5], R^+.

[4]The strong law of large numbers almost certainly guarantees a good approximation

[5]By permissable we mean R^+ should be less than the reward of any goal state

Algorithm 2 Q-learning Algorithm from [3]

Input: current state s' and current reward r', convergence threshold ϵ, function α
Output: the action a to take
Local Variables: Q a table of utilities for every state-action pair, N a counter of visitations to every state-action pair, s the previous state, r the previous reward, real number ρ.

```
1: procedure Q(s', r')
2:     repeat
3:         ρ ← 0
4:         if s' is not a goal state then
5:             increment N[s, a]
6:             Q[s, a] ← Q[s, a] + α(N[s, a])(r + γ max_{a'∈A} Q[s', a'] − Q[s, a])
7:             if |Q'[s, a] − Q[s, a]| > ρ then
8:                 ρ ← |U'[s] − U[s]|
9:             end if
10:            execute action a ← argmax_{a'∈A(s)} f(Q(s', a'), N(s, a))
11:            to yield new state, s'; s ← s'; r ← r'
12:        end if
13:    until ρ < ε
14:    generate π from Q by π(s) = argmax_{a∈A} Q[s, a] for all s
15: end procedure
```

$$f(Q(s,a),n) = \begin{cases} R^+ & \text{if } n \leq N_0 \\ Q(s,a) & else. \end{cases} \tag{9}$$

If the visitation is above this threshold, then the utility is just the Q-value, as it should be. During training, the next action to take from s is then:

$$\arg\max_{a' \in \mathcal{A}(s)} f(Q(s',a'), N(s,a)) \tag{10}$$

This process is repeated until the maximum change in any Q update between iteration i and $i+1$, $||Q_{i+1}(s,a) - Q_i||$, is less than ϵ. Note every iteration must end in a goal state. See Algorithm 2.

3 Experimental Methods and Data Sets

The MDP is instantiated as a race track in two-dimensional plane. The state space is described by position-velocity pairs such that valid positions are non-walls and valid velocities are two dimensional vectors of the form (v_x, v_y) where both v_x, v_y are integers on $[-5, 5]$, and the action space is two dimensional acceleration vector (a_x, a_y) with a_x, a_y integers on $[-1, 1]$. The transition probability $P(s'|s,a)$ in any state s under action a is 0.9; there is a ten percent chance that the action will not be taken, that is, the action will be (0, 0). There are two types of restrictions: crashing

into a wall under "hard crashing" conditions resets the agent to a random position on the starting line. "Soft Crashing" puts the agent in the nearest valid position with velocity (0, 0). There are three racetracks: L, O, and R which describe their shape. Please see figures 5 and 6 for images of the L and R tracks (where 'S' denotes a start state with zero velocity, 'F' denotes finish position, '#' denotes wall, and '.' denotes racetrack pavement). The R track is most difficult because it contains several sharp turns with narrow passages. For all experiments, every VI agent had a convergence threshold[6] of $\epsilon = 1 \times 10^{-12}$ and every Q agent had $\epsilon = 1 \times 10^{-5}$. Finally, the reward of every non-goal state was -1, and the goal states had reward of 0.

The alpha function is given by the $O(1/n)$ function for $n = N(s, a)$ below:

$$\alpha(n) = \frac{1}{1 + \frac{n}{10}} \tag{11}$$

This $\alpha(N(s, a))$ decays slowly enough so as to encourage exploration, but discourages exploration after ten visits. R^+ for the exploration function was arbitrarily set to -0.47.

4 Results

It was difficult to exact highly granular trends for the input parameters γ the discount factor and α the learning factor. For any Q-learner with $0.6 < \alpha < 0.95$ and $\gamma > 0.8$, all else being equal, an agent will achieve roughly the same nearl-optimal score on a map, within some small standard deviation for the stochastic Q-learning algorithm. It is apparent that not much can be interpreted from such results other than high discount factors were needed for more complex state spaces. Figures 1 through 3 show the impact that the discount factor has on the score of VI and Q agents on the different tracks under different crashing conditions. Under soft-crashing conditions, both agents stabilized to the values shown in Table 1 for nearly any input parameter. Figure 4 shows the number of iterations a VI agent needed to converge under hard-crashing conditions. A Q agent followed a similar trend with a bit more noise – anywhere from 100,000 to 1,000,000 iterations were needed, depending on the parameters. More iterations were needed for harsher reward functions (hard crashing), higher discount factors, and more explorative α and exploration functions. Soft-crashing agents typically did not require more iterations with higher discount or alpha parameters because they must have found optimal policies relatively early.

Figures 5 and 6 show some of the the "loopholes" the agents found under soft crashing conditions, when the cost of crashing into the wall was relatively low.

[6] thresholds lower than these did not improve the score on any track for any agent

Soft Crashing	L-track	O-track	R-track
VI	-10.48	-22.6	-23.7
Q	-10.68	-22.62	-23.22
Best	-10	-20	-21
Hard Crashing			
VI	-12.98	-30.9	-31.82
Q	-11.66	-28.64	-30.32
Best	-11	-22	-23

Table 1: The average of 50 simulations by each agent on each track (hard crashing) given the parameters that lead to an optimal solution. The "Best" row is the minimum score ever achieved by any agent on any simulation, which was rare for the O and R tracks. Becuase Q is stochastic, these scores are averaged over ten agents, with standard deviations 0.29, 1.04, and 2.41 for L, O, and R tracks respectively. These standard deviations were more or less consistent for both types of crashing conditions.

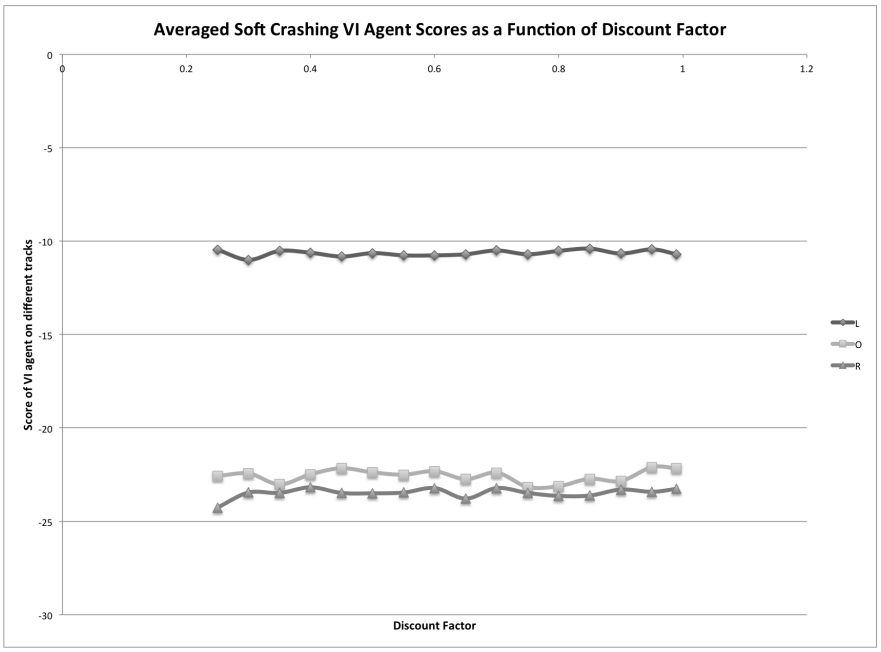

Figure 1: Plot of the average score of 50 simulations of a trained soft-crashing VI agent on each of the three tracks as a function of discount factor, γ under soft crashing conditions. When mistakes are relatively inconsequential, virtually any discount factor is acceptable. This graph for a soft-crashing Q agent also stabilizes near the optimal score for virtually any discount factor.

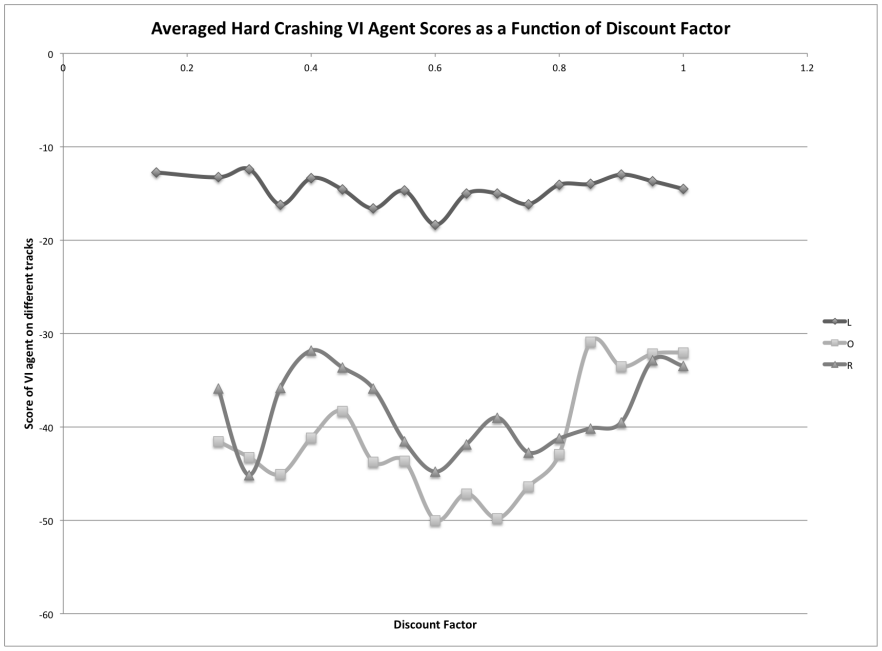

Figure 2: Plot of the average score of 50 simulations of a trained hard-crashing VI agent on each of the three tracks as a function of discount factor, γ. With more complex tracks, a higher γ was required to achieve consistently good scores

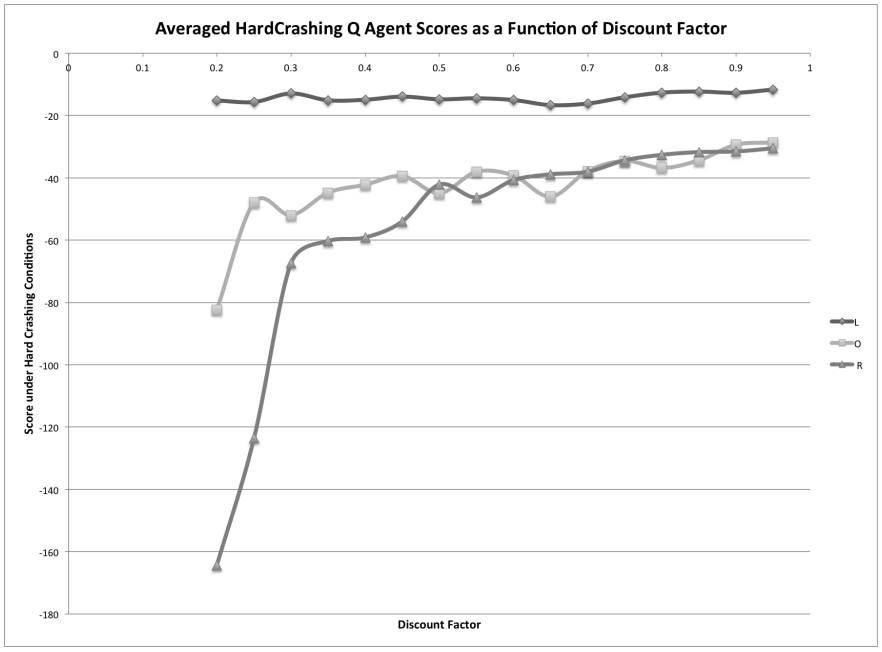

Figure 3: Plot of the average score of 50 simulations of a trained hard-crashing VI agent on each of the three tracks as a function of discount factor. For better performance, a higher discount factor is needed.

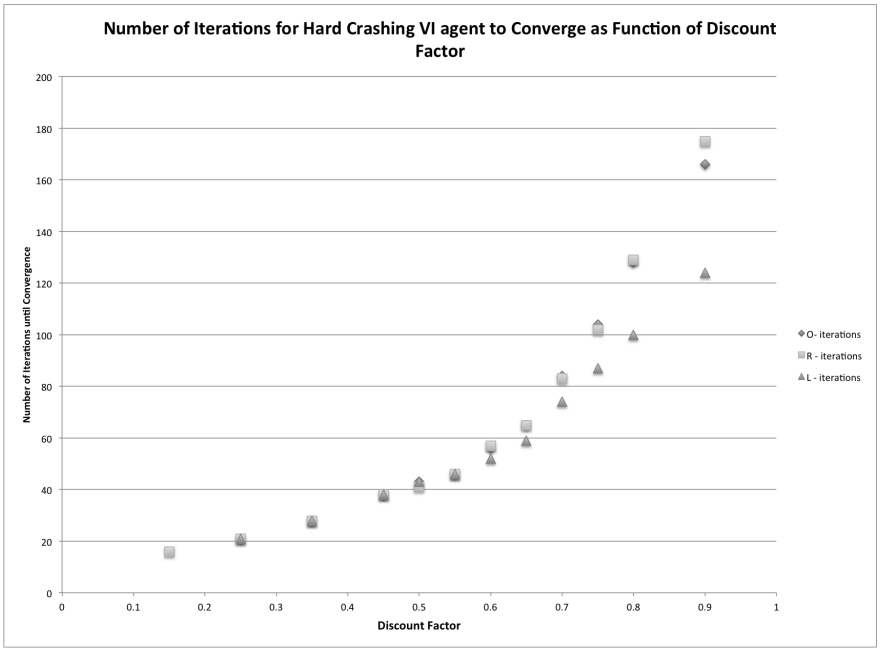

Figure 4: The number of iterations required for a VI agent to converge is nearly linear up to a point with increasing discount factor for hard crashing conditions. Nearer to $\gamma = 1$, the number of iterations increases super-linearly. For a VI agent under soft crashing conditions, this graph was always perfectly linear.

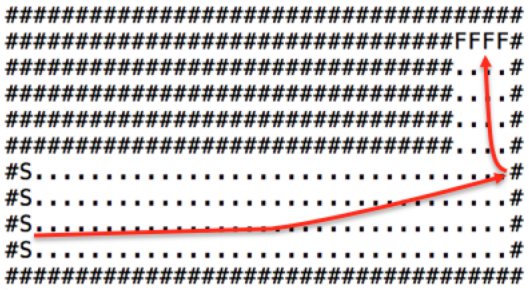

Figure 5: A soft-crashing VI agent on the L-track who used the wall to change velocity more rapidly than the agent itself could. This "loophold" strategy led to the best possible score under these conditions because the agent could recklessly traverse the horizontal stretch without having to decelerate in anticipation of a sharp turn.

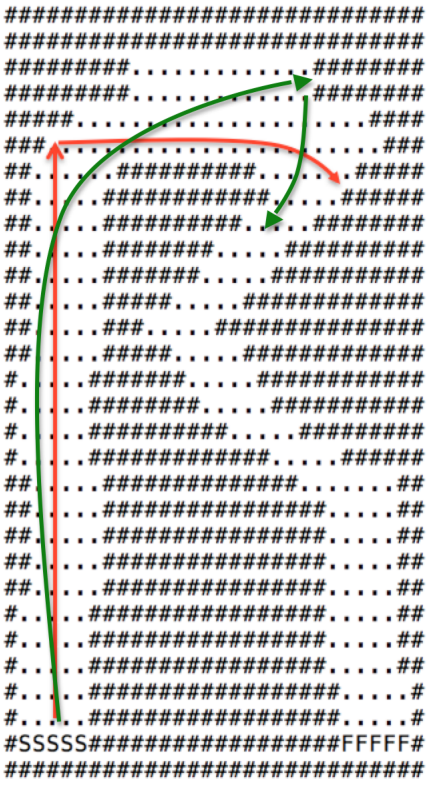

Figure 6: A soft-crashing Q agent on the R-track who also used the wall in two different ways. The red line was generated occasionally by an agent with a higher convergence threshold. The arguably smarter green line, which led to an optimal solution, was found by an agent with a lower threshold.

5 Discussion

The remarkable aspect of these two algorithms is that they are very forgiving with respect to the learning rate and discount parameters. The VI algorithm was able to achieve scores very close to optimal for any value of $\gamma > 0.2$ on all tracks for soft crasing conditions. For hard crashing conditions, any $gamma > 0.6$ was needed for the L-track, $\gamma > 8$ was needed for O-track, and $\gamma > 0.85$. The Q agent was able to reach optimal results with $\gamma > 0.8$ for hard crashing, and $\gamma > 0.6$ for soft. Such a distinction between the necessary gamma values under different crashing conditions shows that the agent is quite responsive to the consequences of its actions. The discount factor represents how much influence neighboring states have on the value of the current state. A high discount factor will make the agent more conscious of neighboring states that happen to be harmful or costly – such as walls that reset the agent to the start line – and it will respond in a risk-averse fashion. The user can tune the discount factor to reflect how conservative the agent should be in its actions

For a Q agent, a constant learning factor could be replaced by a function of the number of times a state-action pair is visited. Devising an alpha function and an exploration function that strike a balance in the exploration-exploitation tradeoff

depends on the size and properties of the environment. For the alpha function, any exponential decay too aggressively approached zero and hindered exploration, as well as the function 1/n. We had to stretch a $O(1/n)$ function along the independent variable's axis so that alpha remained above 0.5 for less than ten visits to a state-action pair. For our racetracks, the finitely many paths to the goal (assuming no reversals) were very narrow and uniformly directed, so it was unlikely that there were more than ten ways to reach the same state-action pair, thus the choice for the alpha function in Equation 10.

In large state spaces, VI will be expensive to run, but polynomial in the size of the input state space (depending on the input parameters). Q-learning, on the other hand, will have great difficulty even finding the goal state on the first iteration, because there is no guarantee that a random walk will reach the goal state in time linear in the size of the state space. The authors of [2] have proposed combining several neighboring states into one superstate to exponentially reduce the number of states. Another solution, which the author of this paper implemented, was to initialize an agent's random walk at states other than the start states. Starting the agent nearer to the goal increases the chance that the random walk will terminate quickly, having provided information about the locality near the goal. Then in subsequent iterations, initialize the agent a bit further away so it has a high probability of intersecting some of the previously visited states. Since the state spaces in the environments tested in this paper were smaller than 50,000 states, random initializations did not have much impact. But in environments with more than say 10^7 states, such a strategy could be helpful.

A faster approximation algorithm for the optimal policy could terminate iterating either a VI or Q agent before convergence if the policies for the vast majority of the states haven't changed in some time. An optimal policy for every state can be found faster than the correct utilities because only the relative ranking of utilities for state-action pairs, not the precise values themselves, determine the policy. And learning the optimal policy is the agent's only goal.

The performance of the agent is dependent on the accessibility of the goal states, which in turn depends on the prevalance and positions of dangerous traps and easy shortcuts. Whether an agent falls for the traps or takes the shortcuts depends on how the agent *expects* it will be rewarded. An agent may prefer a quick but dangerous path to the goal if the reward function harshly penalizes every action and the discount factor is relatively low because the agent pushes itself to recklessly "escape" as quickly as possible. On the other hand, another agent may prefer a longer but safer path to the goal if actions are relatively cheap and the discount factor is relatively high because the agent prefers to be both mindful of, and comfortable in, its surroundings.

This behavior surfaced in the racetrack tests. All agents assumed lower scores for hard crashing conditions than soft-crashing, because the costs of being reset to the start are much higher than remaining in place. Thus agents tended to avoid walls as much as possible, meaning that sharp turns near corners were less favorable than gradual but safer turns, which led to lower velocities and thus lower scores. For soft-crashing conditions, both agents exhibited cunning strategies which usually involved intentionally crashing into a wall in order to change velocity rapidly, see Figures 5

and 6.

6 Conclusions

We contend that a user should implement VI whenever possible because optimality is almost guaranteed with relatively few iterations. And while it is not hard to brute force the possible values for γ with a VI agent, it is trickier to assign good exploration and alpha functions for Q. Q also does not scale well to large state spaces; if there are infinite states, Q may never reach the goal on its first random walk. We discussed techniques improve Q in such situations. Despite these shortcomings, Q-learning is usually necessary because only very rarely do researchers know every minute detail about the transition probabilities.

We also noticed that with enough iterations, Q was able to marginally but consistently outperform VI in the hard-crashing cases (where the reward function was particularly harsh). We don't know whether this was idiosyncratic of our racetrack environment; whether Q, because it is stochastic, was just "getting lucky"; or if Q really had better utility estimates because it visited states so many times. We find it difficult to believe a Q agent can outperform VI in general because it has so much less information that VI – a topic for further investigation.

7 Acknowledgements

The author would like to thank Zach Sabin and Stefan Reichenstein for peer-reviewing this paper, as well Zachary Palmer for his scaffolding code.

References

[1] Mitchell, Tom M. Machine Learning. New York: McGraw-Hill, 1997.

[2] Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of Machine Learning. Cambridge, MA: MIT Press, 2012.

[3] Russell, Stuart J, and Peter Norvig. Artificial Intelligence : a Modern Approach. 3rd ed. Upper Saddle River, N.J.: Prentice Hall, 2010.