Assignment 4: Markov Decision Processes and Reinforcement Learning

Brent Mitton

April 19, 2015

1 Abstract

This paper is a brief look at some simple reinforcement learning techniques as applied to Markov decision processes (MDPs). There will be a focus on three different methods of reinforcement learning and a comparison of how these methods perform on two toy problems.

2 Introduction

2.1 Techniques

This paper will look at three different algorithms for reinforcement learning on Markov decision processes. The first two of these models are policy iteration and value iteration. These two techniques aim to find the optimal policy, π^*, of an MDP given that they are provided a model of the MDP. For the purposes of this paper a *model* means a transition matrix, $T(s, a, s')$, and a reward matrix $R(s, a)$. The transition matrix gives the probability of ending up in state s', give that the agent was in state s and took action a. The reward matrix returns the reward achieved from being in state s and taking action a.

The third technique that will be looked at is called Q-learning. This is one technique that aims to learn the optimal policy of the MDP without being given a model. This sort of method is often more suitable to real-world problems where we don't have prior knowledge of the transition and reward functions, but instead want the agent to learn and react to environmental stimulus.

2.1.1 Value and Policy Iteration

The aim of value iteration is to find the optimal value function, $V^*(s)$, of the MDP. Then, by using the discovered optimal value function the optimal policy can be inferred. The optimal value function of an MDP is defined as:

$$V^*(s) = \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V^*(s')), \forall s \in S \tag{1}$$

The value of a state is the immediate reward of being in the state plus the expected discounted reward of following the optimal path (path of highest reward) from that point on. The optimum policy, π^*, of an MDP can be inferred given $V^*(s)$:

$$\pi^*(s) = \underset{a}{argmax}(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s')V^*(s')) \tag{2}$$

Using a simple iterative process, we can find the correct $V^*(s)$ of an MDP, and thus are able to find the optimal policy. [1]

```
intialize all V(s) arbitrarily
repeat
  for all s ∈ S do
    for all a ∈ A do
      Q(s,a) ← R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')
    end for
    V(s) ← max_a Q(s,a)
  end for
until policy is satisfactory
```

Policy iteration works by trying to find the optimal policy directly instead of relying on finding the correct value function and then inferring the correct policy.

```
choose arbitrary π*
repeat
  π ← π*
  // Find the value function for current policy by solving systems of linear equations
  V_π(S) = R(s,π(s)) + γ Σ_{s'∈S} T(s,a,s')V_π(s'), ∀s ∈ S
  // improve the policy at each state
  π'(s) ← argmax_a (R(s,a) + γ Σ_{s'∈S} T(s,a,s')V_π(s'))
until π = π'
```

2.1.2 Q-Learning

Q-learning works by learning an action-value function $Q(s, a)$, which is defined to be the expected discounted reward that would be obtained by starting at state s, taking action a, then continuing optimally from that point. The slightly modified Q-Learning algorithm used for this paper is detailed below.

```
initialize Q(s,a) arbitrarily
initialize α, γ
for all episodes do
  for all s ∈ S do
    while s ≠ goalstate do
      if not possible to move from s then
        // start new episode
        break
      end if
      a ← action from state using policy derived from Q (ε-greedy)
      r ← reward from doing a
      s' ← state after doing a
      Q(s,a) ← Q(s,a) + α(r + γ max_{a'} Q(s',a') − Q(s,a)))
      if New Q(s,a) not significantly different then prior Q(s,a) then
        // start new episode
        break
      end if
      decrease α
      s ← s'
    end while
  end for
end for
```

This algorithm is essentially Watkins' Q-Learning algorithm[2] but with a couple of modifications to better suit the problems.

1. Ensure that the algorithm does not spend time trying to find best action sequences from states that have no path to the goal.

2. Break into a new episode in the case that $Q(s, a)$ has not changed significantly. This prevents getting stuck in cycles of states, however this may remove the guarantee of convergence.

2.2 The Problems

The problems used in this paper to compare the techniques are two different sizes of maze/gridworld problems. The environment for the problem is an NxN grid where each cell is a state. From each cell the agent can take an action $a \in A = \{$Go North, Go East, Go South, Go West$\}$. There are four different types of states to be considered in the environment:

- regular states, can be entered from adjacent states
- wall states, cannot be entered from other states
- enalty states, give a large negative reward to the the agent
- goal states, give a large positive reward to the agent and terminates the problem

Generally, there is also a step penalty introduced which will give the agent a negative reward for each step taken. The optimal policy of one of these problems should be the one that maximizes the reward gained.

Gridworlds are an interesting problem to consider for Markov decision processes not only because they are easy to visualize and understand, but also because it is often possible to generalize more complex problems into variants of gridworlds.

2.2.1 5x5 World

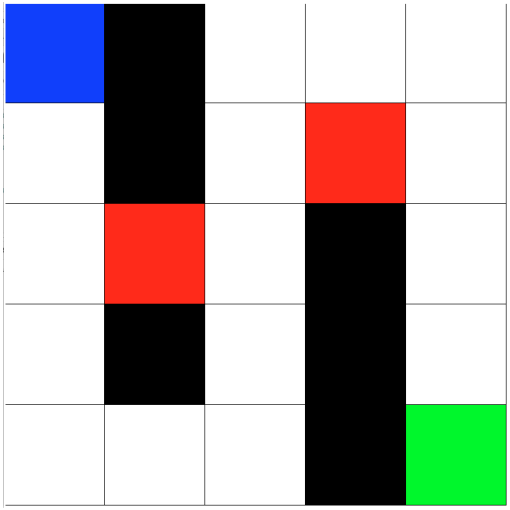


Figure 1: 5x5 World

The 5x5 gridworld/maze is a small environment used to demonstrate the three techniques. It has a total of 25 possible states that the agent could end up in. Each of the coloured squares in *Figure 1* represents one of the different state types.

- Blue – not a state type, indicates the position the agent will start in
- Black – a wall state
- Red – penalty state
- Green – goal state

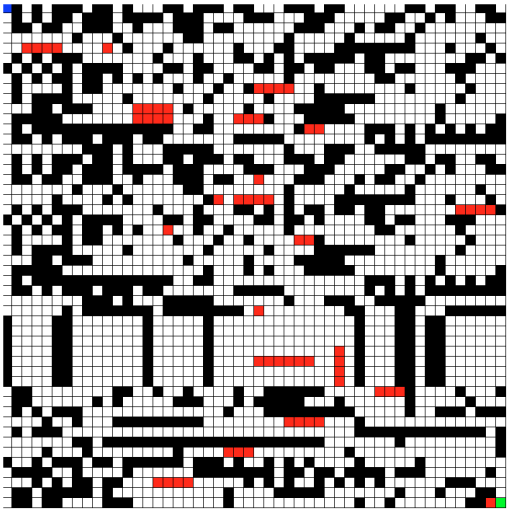


Figure 2: 50x50 World

2.2.2 50x50 World

The 50x50 gridworld/maze is a larger environment used to demonstrate the three techniques. There are a total of 2500 possible state the agent could end up in. *Figure 2* shows the pseudorandomly generated environment used for experimentation.

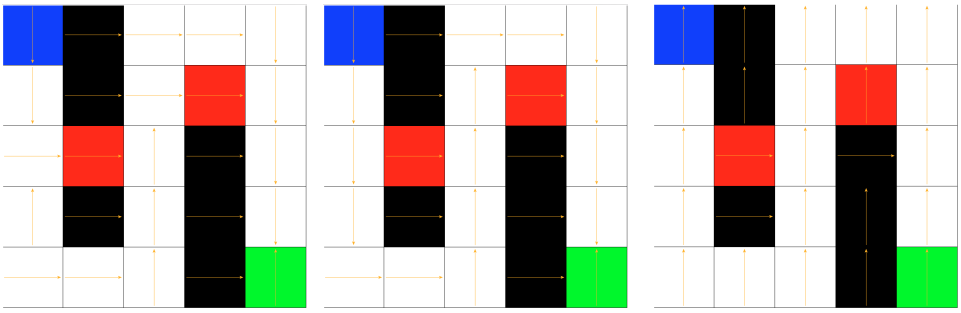
3 Comparisons

3.1 5x5 World

For comparison it is interesting to try each of these methods on different reward schemas. That is, different reward values for: step penalties, entering penalty states, entering the goal states. The reward schema used for applying the three techniques to the 5x5 World will be

1. Goal State: +10; Step Penalty: -0.05; Penalty State: -1;
2. Goal State: +10; Step Penalty: -1: Penalty State: -5;
3. Goal State: 0; Step Penalty: 1; Penalty State: -2;

3.1.1 Value Iteration



(a) Reward schema 1
Reward: 7.65
Time: 0.0054
Iterations: 84

(b) Reward schema 2
Reward: -5
Time: 0.0061
Iterations: 81

(c) Reward schema 3
Reward: ∞
Time: 0.0002
Iterations: 2

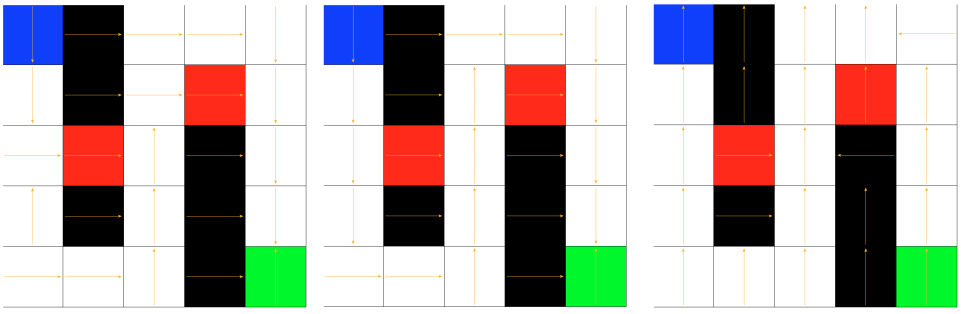
Figure 3: Policies Learned by Value Iteration

There are some interesting things of note that are happening in value iteration for these simple problems. Firstly in *Figure 3a* the policy learned was not optimal given the reward structure. For a problem of this size it possible to trace through by hand and find that the correct policy is the one detailed in *Figure 3b*. This is most likely an error in the implementation of value iteration.

Another interesting but not surprising result can be seen in *Figure 3c*. For this reward schema there is no incentive for the agent to enter the goal but there is an incentive for making arbitrary steps. This causes the agent to try to crash into the upper wall and gain infinite reward.

One further note to make for all subsequent policy figures, the goal state is always terminal though the arrow may suggest otherwise.

3.1.2 Policy Iteration



(a) Reward schema 1
Reward: 7.65
Time: 0.0094
Iterations: 10

(b) Reward schema 2
Reward: -5
Time: 0.0112
Iterations: 11

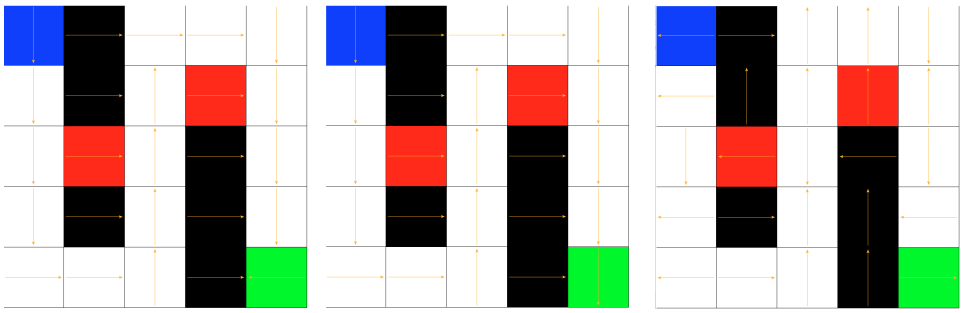
(c) Reward schema 3
Reward: ∞
Time: 0.5076
Iterations: 1000 (max)

Figure 4: Policies Learned by Policy Iteration

Policy iteration demonstrates the same results as *Figure 3*. Again, this is most likely due to the implementation.

As expected policy iteration takes fewer iterations than value iteration (except in *Figure 4c*). It does take slightly longer in all cases to find what it believes to be the optimal policy. This would suggest that policy iteration has more computationally costly iterations, but does fewer of them overall. *Figure 4c* is an exceptional case since there is no optimal policy that causes the MDP to achieve a terminal state.

3.1.3 Q-Learning



(a) Reward schema 1
Reward: 9.25
Time: 1.353
Iterations: 25013

(b) Reward schema 2
Reward: -5
Time: 1.340
Iterations: 25597

(c) Reward schema 3
Reward: ∞
Time: 1.677
Iterations: 65465

Figure 5: Policies Learned by Q-Learning

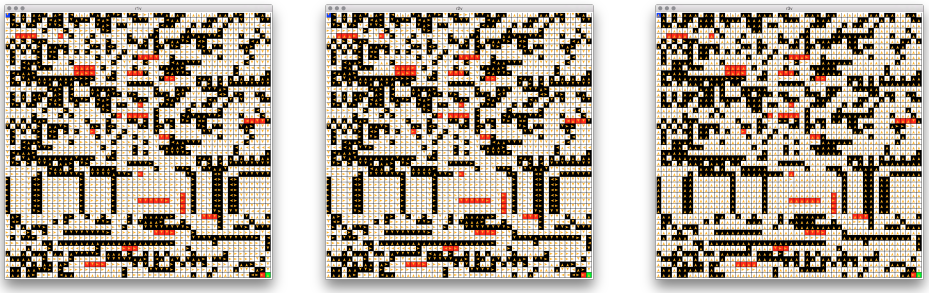
The implementation of Q-Learning is completely separate from the implementations of both policy and value iteration, so it is not running into the same pitfalls as either of those algorithms. The number of iterations for Q-Learning is much higher than either of the other techniques. This is to be expected from the implementation of the algorithm. The number of episodes has been set to to run a minimum of 1000 times and each episode will iterate over each state at least 1 time. This leads to a minimum number of iterations of 25000. This number of episodes was chosen to increase liklihood of convergence for large MDP problems. It could be liberally adjusted to improve performance for smaller problems.

3.2 50x50 World

The reward schema for the larger gridworld is modified to take into account the large number of steps that are needed to make it to the goal.

1. Goal State: +1000; Step Penalty: -0.05; Penalty State: -3;
2. Goal State: +1000; Step Penalty: -5: Penalty State: -20;
3. Goal State: 0; Step Penalty: 1; Penalty State: -2;

3.2.1 Value Iteration



(a) Reward schema 1
Reward: 995.15
Time: 1.5505
Iterations: 131

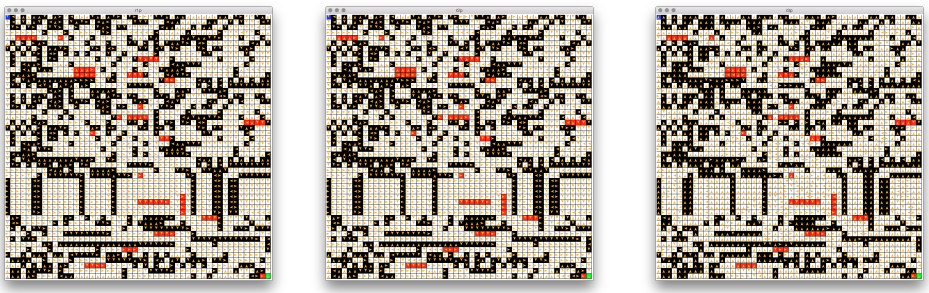
(b) Reward schema 2
Reward: 515
Time: 1.6819
Iterations: 131

(c) Reward schema 3
Reward: ∞
Time: 0.03668
Iterations: 3

Figure 6: Policies Learned by Value Iteration

Value iteration seemed to be able to get the correct optimal policy very quickly and in relatively few iterations. The times listed in *Figure 6c* do not include the time to build $T(s, a, s')$ or $R(s, a)$. The time need to construct this model was much greater than the time needed to find the optimal policy.

3.2.2 Policy Iteration



(a) Reward schema 1
Reward: 995.15
Time: 850.5694
Iterations: 1000 (max)

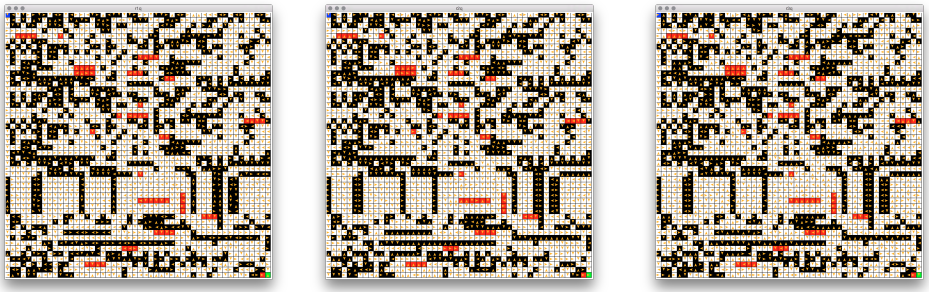
(b) Reward schema 2
Reward: 515
Time: 880.0052
Iterations: 1000 (max)

(c) Reward schema 3
Reward: ∞
Time: 846.6438
Iterations: 1000 (max)

Figure 7: Policies Learned by Policy Iteration

Policy iteration was also able to find the correct optimal policy in all cases. However the results are much less impressive. Each run used the maximum number of iterations allowed to it. It's possible that the optimal policy was found much earlier than the one-thousandth iteration and policy iteration spent most of it's time trying to refine this.

3.2.3 Q-Learning



(a) Reward schema 1
Reward: ∞
Time: 1443.6557
Iterations: 7493022

(b) Reward schema 2
Reward: 515
Time: 2058.5023
Iterations: 11688558

(c) Reward schema 3
Reward: ∞
Time: 1745.3181
Iterations: 9459882

Figure 8: Policies Learned by Q-Learning

In *Figure 8a*, Q-learning was unable to find the optimal policy. This seems to be the effect of size of the step penalty. *Figure 8b* had a larger step penalty that was able to encourage the agent to quickly find a path to the goal and hence achieve the correct result.

In all cases Q-learning took much longer than the methods in which the model was already known. This makes intuitive sense, since when given a model the agent is able to directly look up the consequences of future actions.

4 Conclusion

For model-based methods value iteration was able to outperform policy iteration in terms of speed. This is most probably due to the computational intensity of each of policy iteration's iterations where it needed to solve a system of linear equations in order to update the policy. In straightforward cases, however, policy iteration was able to converge in many fewer iterations than value iteration. In non-straightforward cases (such as those where the terminal state gives no positive reward), it seemed that value iteration was able to more quickly find the optimal policy.

Q-learning, in all cases, took longer than either of the model-based methods. This makes intuitive sense, since the model based methods were able to easily look ahead to discover future reward sequences whereas Q-learning had to rely on exploration. However, Q-learning is still probably a more commonly useful approach. Knowing the model and reward function may be unlikely in real world scenarios, and when they are known it is often improbable to store them in memory. The 50x50 gridworld was the "large" problem presented in this paper, however it only contained 2500 states. With 2500 states the size of the transition probability matrix was $4 \times 2500 \times 2500 = 25,000,000$ elements.

---

[1] Kaelbling, Leslie Pack & Littman, Michael L & Moore, Andrew W (1996). Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4. 237-285

[2] Watkins, C.J.C.H (1989). Learning from Delayed Rewards. Ph.D. thesis, Cambridge University