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 PISTAR of an MDP given that they are provided a complete model of the MDP. The third technique that will be looked at is called Q-learning. This technique aims to learn the optimal policy of the MDP without being given a model. This is desirable in many real-world cases where models are not known and learning must happen given only environmental inputs.

2.1.1 Value and Policy Iteration

The aim of value iteration is to find the optimal value function 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:

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

and the optimum policy can be inferred through

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

This function can be found through an iterative process that has been shown to converge to the correct values. CITATION

intialize all V(s) arbitrarily
repeat
  for all $s \in S$ **do**
    for all $a \in A$ **do**
      $Q(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$

end for
$V(s) \leftarrow \max_a Q(s,a)$
end for
until policy is satisfactory

CITE

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

choose arbitrary π^*
repeat
$\pi \leftarrow \pi^*$
// Find the value function for current policy by solving systems of linear equations
$V_\pi(S) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s'), \forall s \in S$
// improve the policy at each state
$\pi'(s) \leftarrow \arg\max_a (R(s,a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s'))$
until $\pi = \pi'$

2.1.2 Q-Learning

Q-learning works by learning an action-value function (QSA), 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.

A general outline of the algorithm that was used for this paper follow:

initialize Q(s,a) arbitrarily
initialize α, γ
for all episodes **do**
for all $s \in S$ **do**
if not possible to move from s **then**
// start new episode
break
end if
$a \leftarrow$ action from state using policy derived from Q (ϵ-greedy)
$r \leftarrow$ reward from doing a
$s' \leftarrow$ state after doing a
$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s,a)))$
if New $Q(s,a)$ not significantly different then prior Q(s,a) **then**
// start new episode
break
end if
decrease α
$s \leftarrow s'$
end for
end for

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 EARLY EPISODES

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 from** 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. WRITE AN EXAMPLE OF THIS

2.2.1 5x5 World

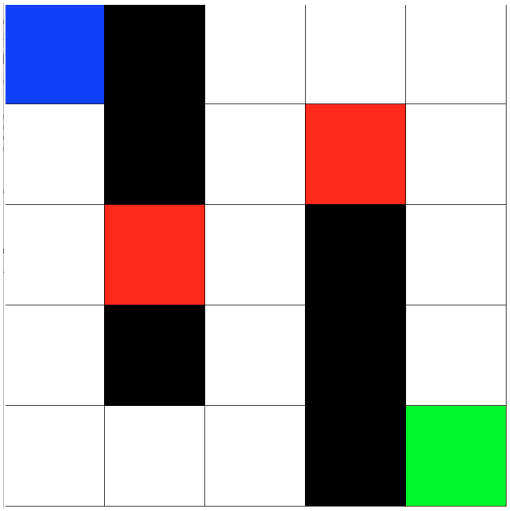


Figure 1: 5x5 World

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 starting position. There are no special rules, rewards, or penalties for entering this state
- Black – a wall state
- Red – penalty state
- Green – goal state

2.2.2 50x50 World

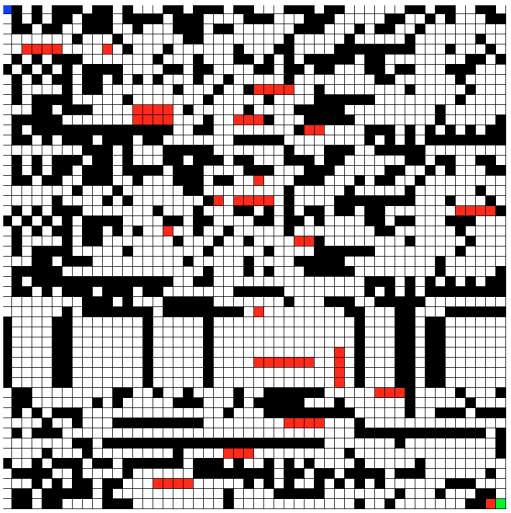


Figure 2: 50x50 World

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 that was generated 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

3.1.2 Policy Iteration

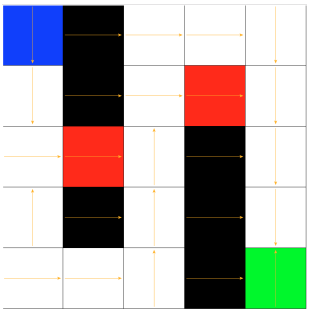
3.1.3 Q-Learning

3.2 50x50 World

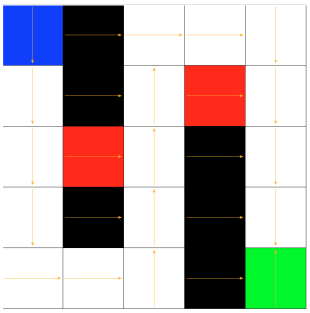
3.2.1 Value Iteration

3.2.2 Policy Iteration

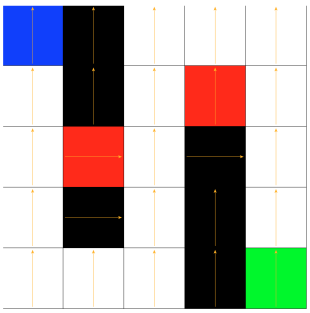
3.2.3 Q-Learning



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

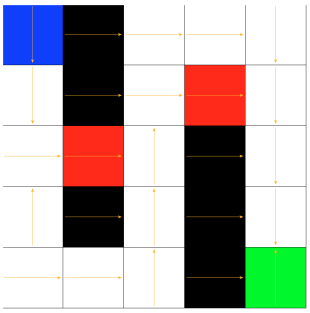


(b) Reward schema 2
Reward: -5
Time: 0.0061
Iterations: 81

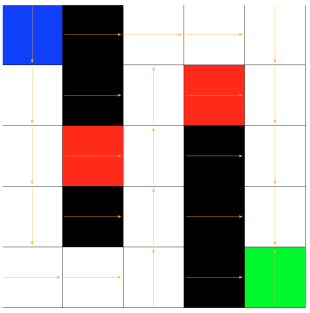


(c) Reward schema 3
Reward: ∞
Time: 0.0002
Iterations: 2

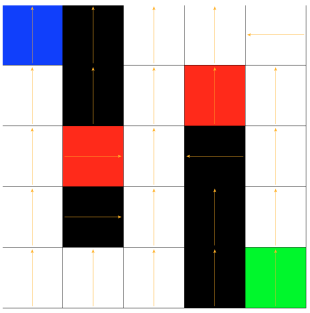
Figure 3: Policies Learned by Value Iteration



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

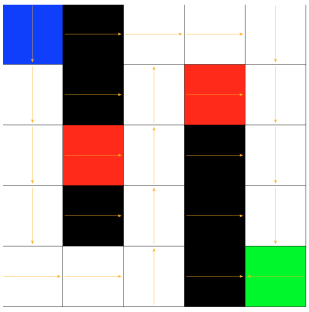


(b) Reward schema 2
Reward: -5
Time: 0.0112
Iterations: 11

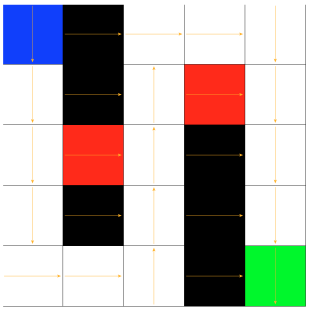


(c) Reward schema 3
Reward: ∞
Time: 0.5076
Iterations: 1000 (max)

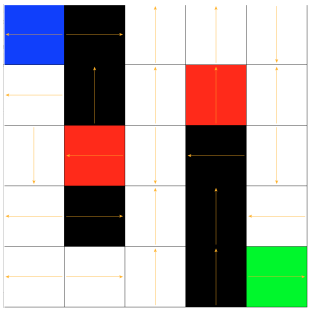
Figure 4: Policies Learned by Policy Iteration



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

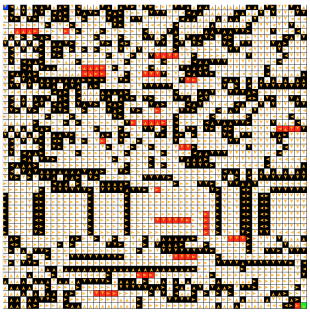


(b) Reward schema 2
Reward: -5
Time: 1.340
Iterations: 25597

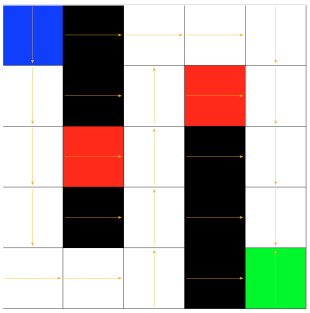


(c) Reward schema 3
Reward: ∞
Time: 1.677
Iterations: 65465

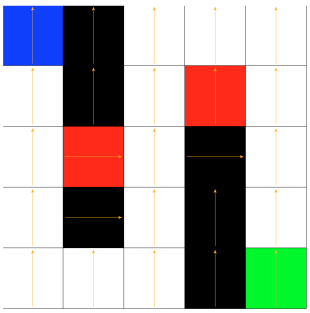
Figure 5: Policies Learned by Q-Learning



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

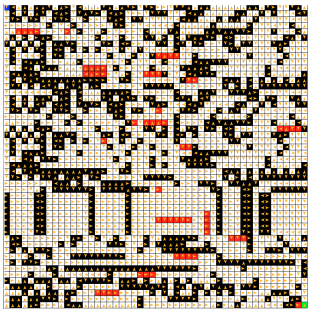


(b) Reward schema 2
Reward: -5
Time: 0.0061
Iterations: 81

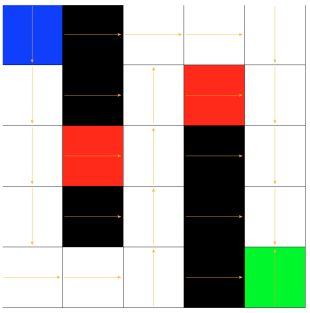


(c) Reward schema 3
Reward: ∞
Time: 0.0002
Iterations: 2

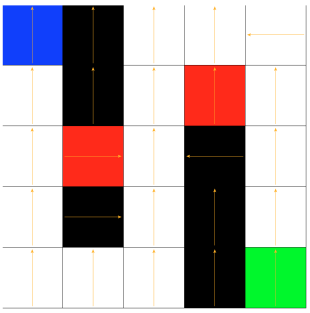
Figure 6: Policies Learned by Value Iteration



(a) Reward schema 1
Reward: 995.15
Time: 39.5257
Iterations: 81

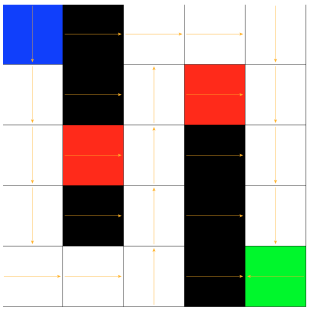


(b) Reward schema 2
Reward: -5
Time: 0.0112
Iterations: 11

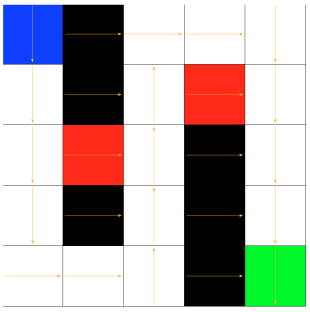


(c) Reward schema 3
Reward: ∞
Time: 0.5076
Iterations: 1000 (max)

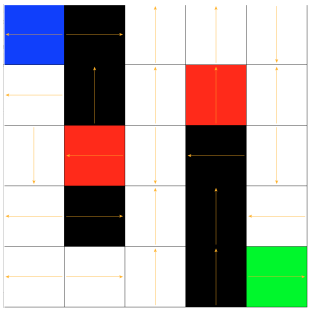
Figure 7: Policies Learned by Policy Iteration



(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 8: Policies Learned by Q-Learning