Exploring Markov Decision Processes

Rohan Ramakrishnan
Georgia Institute of Technology
rohanrk@gatech.edu

***Abstract*—In this paper, The decision making model known as Markov Decision Processes (MDPs) is observed through two specific examples. Methods of discovering optimal policies for MDPs are analyzed and compared across the two examples. The methods of solving MDPs that are discussed include Value Iteration, Policy Iteration, and Reinforcement Learning.**

I. Markov Decision Processes

Markov Decision Processes is defined as a 4-tuple (S, A, T, R) where S is a set of states, A is a set of actions, T is a transition model, and R is a reward function. T is a function that is formally defined below

$$T(s, a, s') = \Pr(s' \mid s, a)$$

Equation 1: Formal Definition of a Transition Model

The transition model takes in a state, an action, and another state and returns the probability of advancing to the latter state given that the current state and the action taken. An important property about this model is that it only depends on the immediately preceding state. This conforms to the Markovian property

The reward function maps states to real numbers, $R(s): S \to R$. It represents the immediate reward for being in a particular state.

The solution of an MDP is known as a policy π. A policy is a function that maps states to actions, $\pi(s): S \to A$. A policy represents the action an agent should take for every state it is in. Generally, a policy that gives greater long term reward, is considered better than a policy that gives smaller long term reward.

By this definition, the optimal policy, π^*, is the policy that maximizes the long term expected reward. Formally, π^* is defined as follows

$$\pi^* = argmax$$

Equation 2: Formal Definition of an Optimal Policy of an MDP

For the purposes of this discussion, the following two MDPs will be analyzed.

A. Easy Grid World

This is a 6x6 grid world shown in figure 1. The reward function is described as follows.

$$R(s) = \begin{cases} 100, if\ s\ is\ terminal \\ -1, otherwise \end{cases}$$

Equation 3: Reward Function for Grid Domains

The terminating state is the top right of the grid which is (5,5) for Easy Grid World.

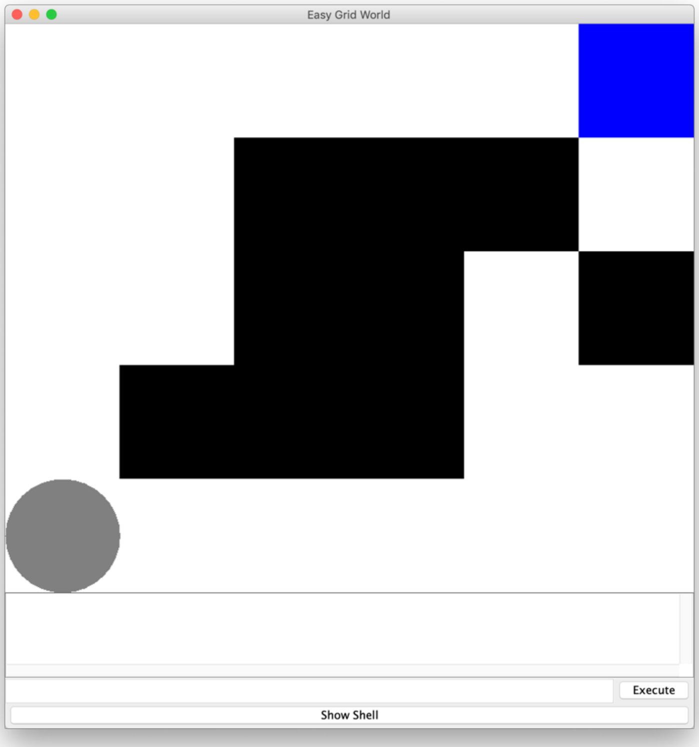

Figure 1: Easy Grid World MDP

B. Large Grid World

This is an 11x11 grid world domain shown in figure 2. This MDP uses the same reward function as Easy Grid World. The terminal condition also remains the top right corner of the grid (10, 10).

Another important point to note for both grids. Actions include moving up, down, left, and right. Actions have an 80% chance to successfully execute with approximately a 6.67% of performing an unintended action.

Grid Worlds are classic examples of MDPs and often used to introduce the concept of MDPs in introductory AI courses and ML courses. So why are these particular grid domains interesting? Simply because these MDPs and their states are easy to visualize. Imagine an MDP that simulates old Atari games such as Lunar Lander. There are several states that are difficult to visualize and one may argue that the state space is continuous (which makes solving that particular MDP an arduous task).

Furthermore, having such a large state space results in long runtimes when solving the MDP.

Grid Worlds are robust, easy to modify, quick to solve, and small changes in their layouts can provide useful intuition on how small changes can affect the optimal policy. As a result, we can analyze two different versions of what is effectively the same problem and still produce meaningful analysis.

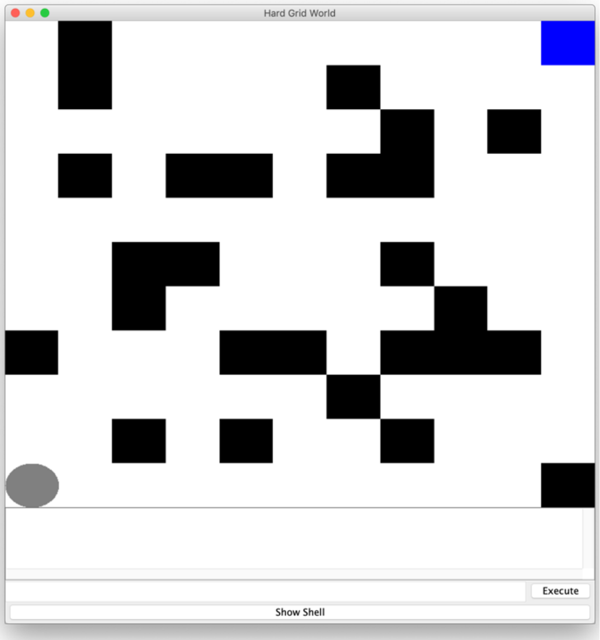

Figure 2: Hard Grid World Domain

II. Solving Markov Decision Processes

Before beginning a discussion on the various methods of solving MDPs, it is important to introduce the concept of discounted reward or utility.

Utility or value is defined as the long term expected reward for a particular agent from being in a particular state. For the purposes of this analysis, utility of a state is predicated on the assumption that the agent(s) have infinite time to work with. If they did not the utilities would drastically change.

Utility is defined with the following equation.

$$U(s_0, s_1, s_2 \ldots) = \sum_{t=0}^{inf} \gamma^t R(s) \; s.t \; 0 < \gamma < 1$$

Equation 4: Definition of Utility

Where gamma is the discount factor. This discount factor is critical as it allows this infinite geometric series to converge to a value which allows

This value is necessary to consider as there are many states with equal reward, but those states could have drastically different utilities. This means that an agent should attempt to progress towards states of higher utility as they yield the highest expected reward over time.

Utility is often parameterized by the current policy an agent is following as those actions output by the policy directly define the utility of a state.

$$U^{\pi}(s) = E[\sum_{t=0}^{inf} \gamma^t R(s) \mid \pi]$$

Equation 5: Definition of Utility Parmeterized by Policy

Finally, the Bellman Equation introduces utility as a function of the reward of the current state and the future expected reward. What this means is that the agent considers the utility of the state to be the reward it gains from being in the state plus some discounted estimate of the long term reward it can gain from being in its current state.

$$U(s) = R(s) + \gamma max_a(\sum_{s'} T(s,a,s')U(s'))$$

Equation 6: The Bellman Equation

Note that $U(s) = U^{\pi^*}(s)$.

With this background information on utility, we can now begin a discussion on using utility to solve MDPs.

A. Value Iteration

Value Iteration is a method of estimating the true utility of each state over time. One can observe that the Bellman equation is nonlinear due to the max function so it cannot be solved using linear algebra methods. Therefore, in order to find the true utilities, we need to iteratively calculate U(s) for every state. Once we have the true utilities, the optimal policy is defined below.

$$\pi^*(s) = argmax_a(\sum_{s'} T(s,a,s')U(s'))$$

Equation 7: Optimal Policy Equation Utilizing True Utilities of Every State

Since we start out with arbitrary utilities for the states (for this analysis all initial utilities were initialized to 0 at time step 0), an observant reader may point out that this method is flawed as we cannot possibly start out with arbitrary utilities and hope to converge to the correct reader.

However, that is not the case. The reason we can start off with inaccurate predictions of the utility and still converge to the correct solution is due to the first term of the Bellman Equation. The reward of the current state represents 'ground truth'. This means that with each update, the predicted utility is growing slowly toward its 'true' utility especially since the predicted utility update is discounted more over time. This process is similar to update rules for neural nets

Value Iteration was run on both MDPs with a discount factor of 0.99. There are some important observations to note about the results of Value Iteration on the two Grid World MDPs. The first is that Value Iteration found solutions very quickly for both MDPs.

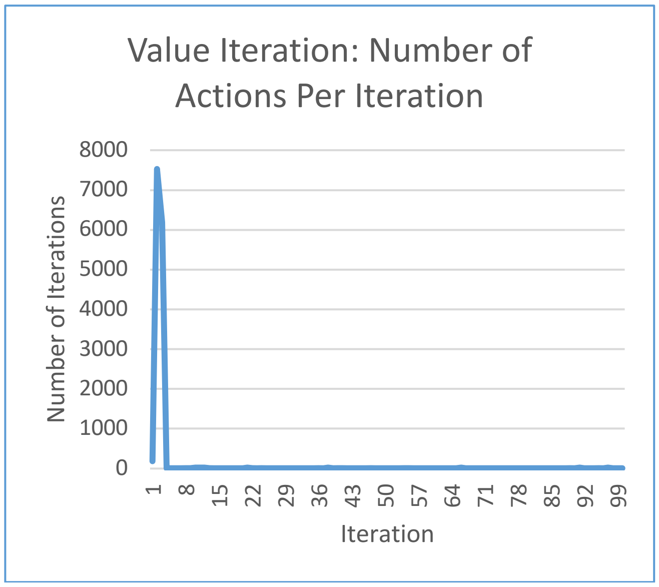

Figure 3: Small Grid World. Number of Actions Per Iteration for Value Iteration

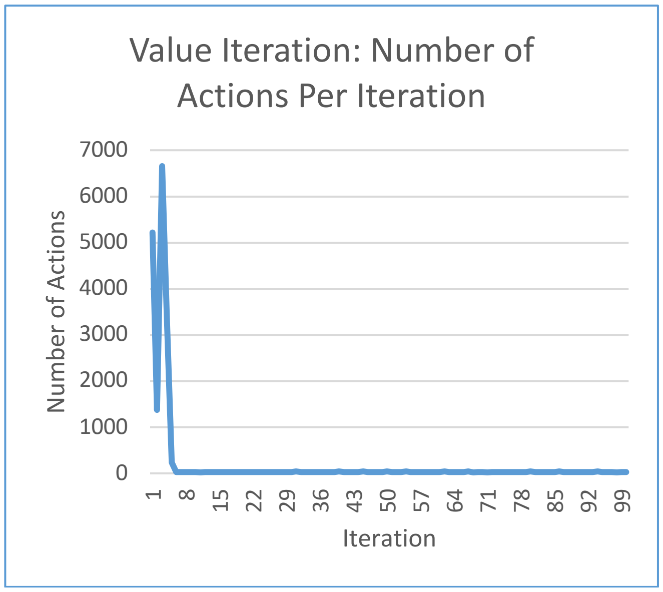

Figure 4: Large Grid World. Number of Actions Per Iteration for Value Iteration

For both MDPs, Value Iteration converges within the first 10 iterations. It is interesting that Value Iteration takes fewer actions to reach the terminal state for the Large Grid World (6656 at iteration 3) than it does for Easy Grid World (7528 at iteration 2). This can be attributed to the stochasticity of the agent's actions. The second observation is the reward that Value Iteration accumulates per iteration. These results are shown in figures 5 and 6.

These results are consistent with the number of actions per iteration, as the longer it takes for the agent to reach the terminal state, the more negative reward it accumulates. The agent does accumulate a significant amount of negative reward due to the time it takes to find a consistently well performing policy, but it very quickly levels out and converges within the first 10 iterations.

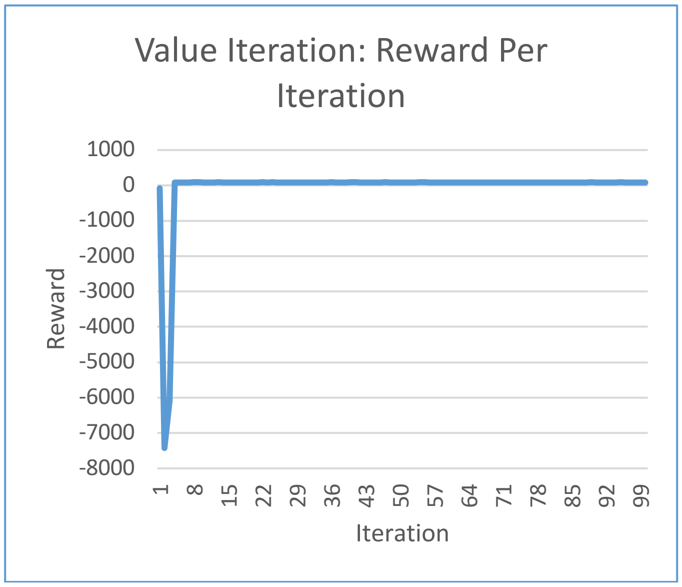

Figure 5: Small Grid World. Reward Per Iteration for Value Iteration

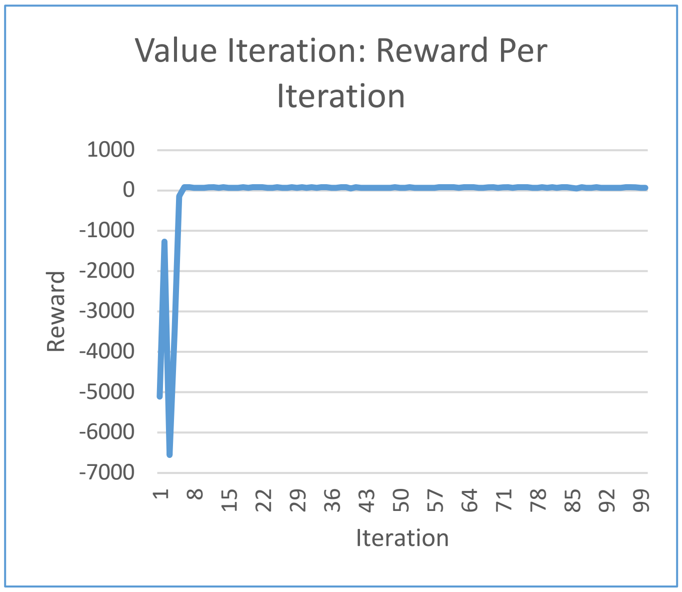

Figure 6: Large Grid World. Reward Per Iteration for Value Iteration

As expected, the reward per iteration is inversely correlated with the number of actions per iteration. In fact, figures 3 and 4 appear to be reflections of figures 5 and 6 across the x-axis. However, the reward slowly evens out and the agent consistently performs better racking up reward that's closer to the reward from reaching the terminal state. This implies that the agent is reaching the terminal state faster over time.

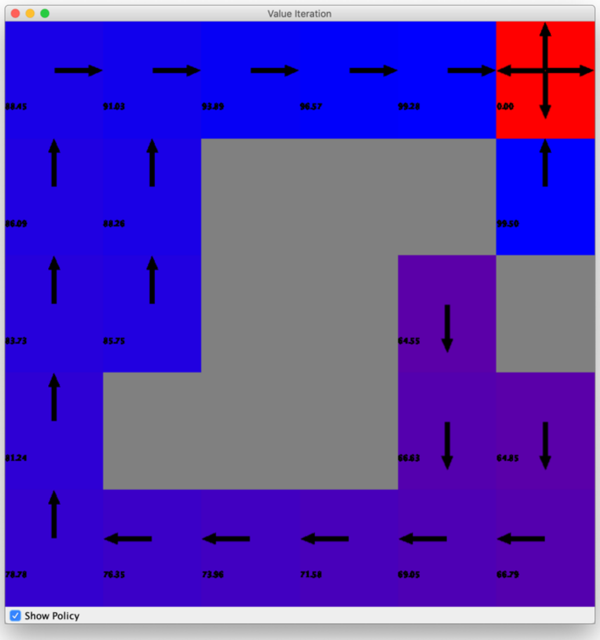

Figure 7: Value Iteration Convergence On Easy Grid

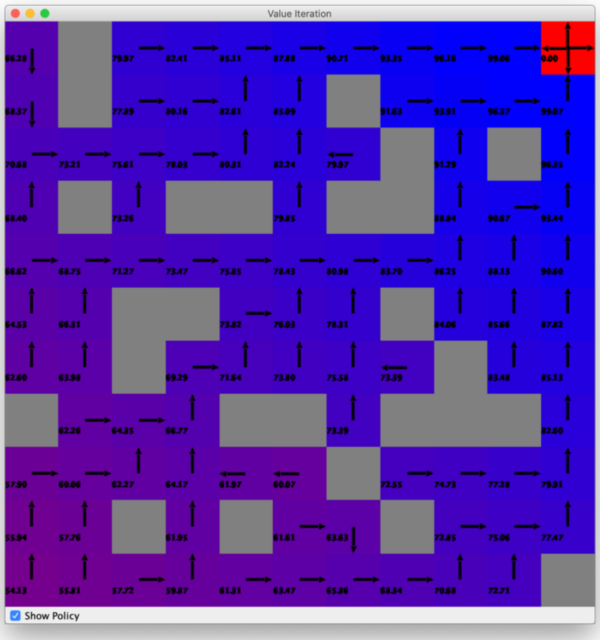

Figure 8: Value Iteration Convergence On Large Grid

Figures 7 and 8 show the final policy that the agent converges to. This appears to be an optimal policy as every action brings the agent closer to the terminal state. The terminal state has a value of 0 in the diagram because the utility of each state is calculated as the agent enters and exits the state, however the run terminates at the terminal state so the value is never calculated. Regardless the agent clearly recognizes that it should take actions that move itself towards the top right corner.

B. *Policy Iteration*

Though Value Iteration provides a useful iterative method to estimate the true utility of every state, one may observe that the solution of an MDP is a policy, but Value Iteration focuses on discovering utility. This observation leads to the realization that it is unnecessary to know the exact value of the utility of each state so long as they are accurate relative to each other. For example consider two states, s_1 and s_2 . Let $U(s_1) = 1$ and $U(s_2) = 2$. If our estimate of the utility is -1 and 0.5 respectively, we can still find an optimal policy because although we have an inaccurate value of the utilities, the key concept is that s_2 has greater utility than s_1 .

This leads to a new iterative method to solve MDPs known as Policy Iteration. Policy Iteration starts with an initial policy and then iteratively improves that policy over time using the following equations.

$$U_t(s) = R(s) + \gamma \sum_{s'} (T(s, \pi_t(s), s')(U_t(s))$$

Equation 8: Policy Iteration Version of Bellman Equation

$$\pi_{t+1} = argmax_a(\sum T(s, a, s')U_t(s'))$$

Equation 9: Policy Update Equation

Equation 8 is actually linear! We could solve it through linear algebra techniques. However it turns out that those calculations are cumbersome and computationally expensive, so it is often better to utilize iterative procedures rather than attempt to solve equation 8 especially because we would need to evaluate this equation through several policies.

Running Policy Iteration on the aforementioned MDPs produces familiar results.

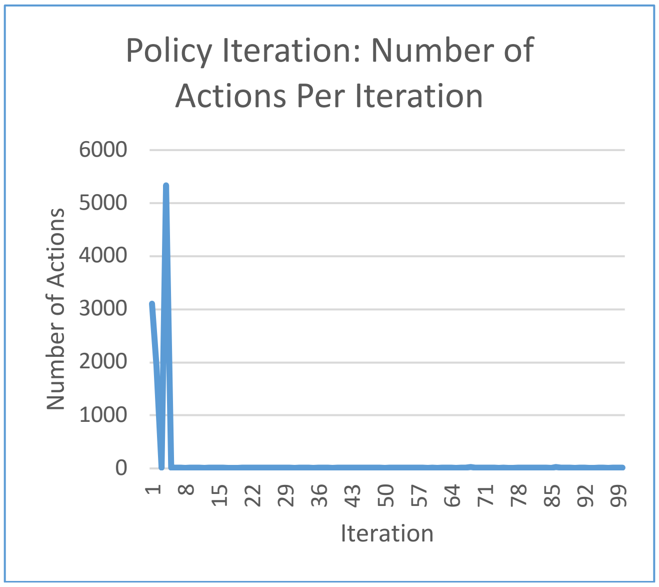

Figure 9: Small Grid World. Number of Actions Per Iteration for Policy Iteration

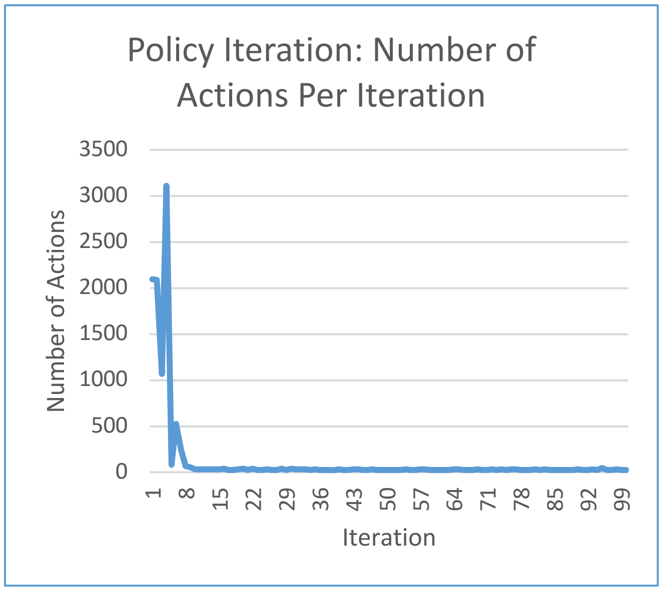

Figure 10: Large Grid World. Number of Actions Per Iteration for Policy Iteration

Similar to Value Iteration, Policy Iteration produces consistently good results quite quickly. This may be attributed to the simplicity of the MDPs, it may also be because the agent learns very quickly which states are closer to its goal state.

As expected, the reward per iteration is inversely correlated to the number of actions taken to reach the terminal state. As with Value Iteration, the agent is quickly able to produce consistent results and accumulate large reward given the reward function of both MDPs.

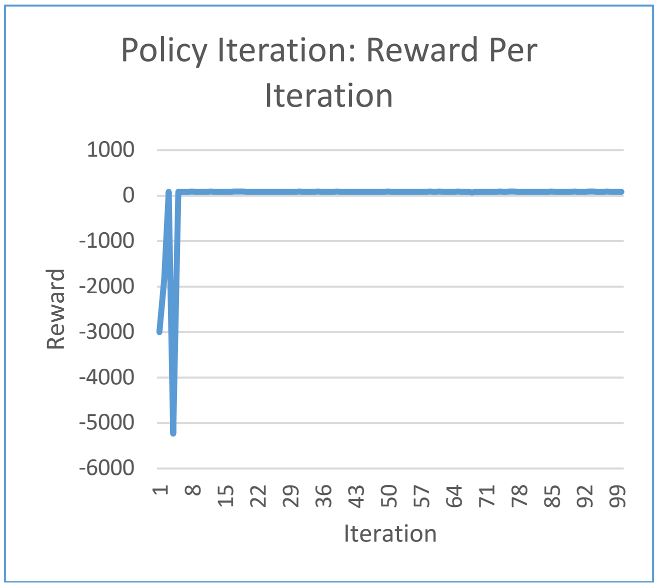

Figure 11: Small Grid World. Reward accumulated per Iteration for Policy Iteration

Despite the similarity in their results, Policy Iteration does perform slightly better. It takes fewer actions overall and accumulates higher total reward. This implies that Policy Iteration performs well on simpler MDPs with smaller state spaces and action spaces. Even the Large Grid is a rather simple MDP. This is consistent with Policy Action's method of estimating utilities based on some current policy and improving said policy. Given simpler MDPs, it would be easier to estimate utilities correctly with respect to the other states.

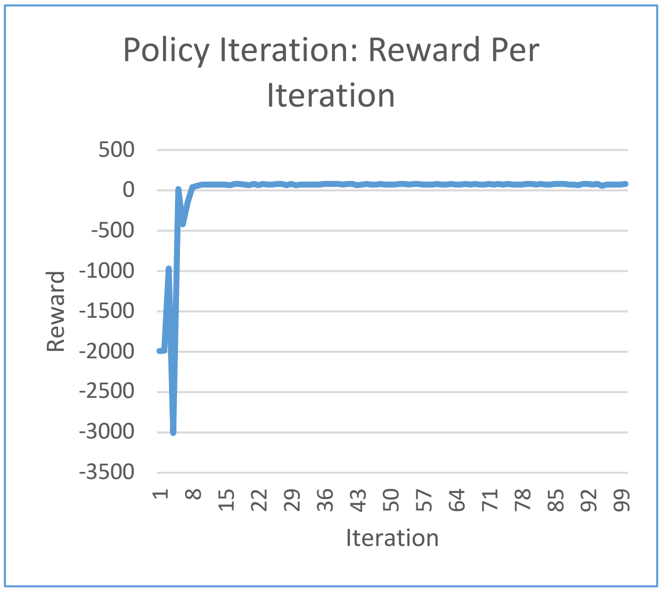

Figure 12: Large Grid World. Reward accumulated per Iteration for Policy Iteration

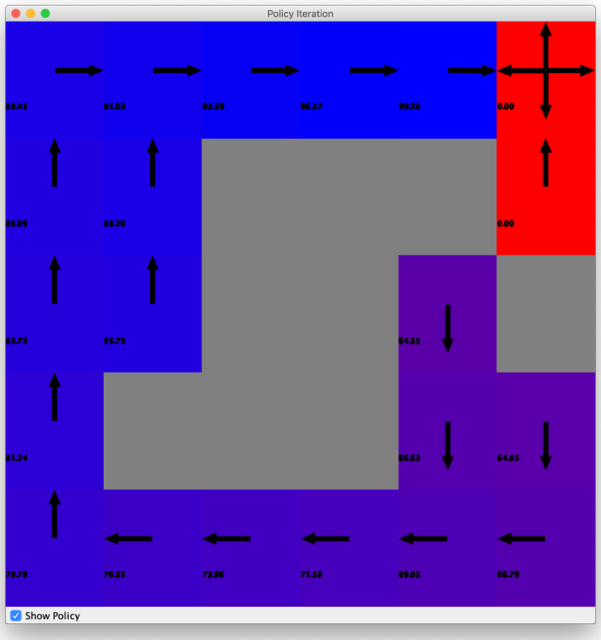

Figure 13: Policy Iteration Convergence on Small Grid

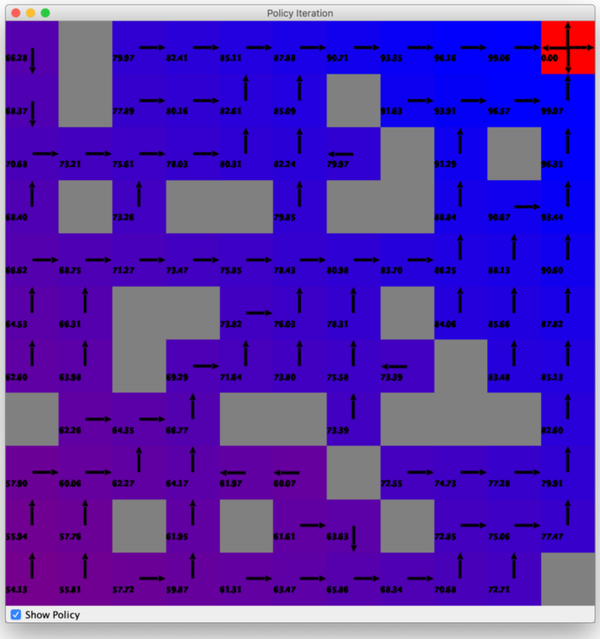

Figure 14: Policy Iteration Convergence on Large Grid

The convergence images are identical to Value Iteration which means both methods result in an optimal policy.

III. Reinforcement Learning

Up until now, the discussion has been focused on iterative methods that use model information to discover the optimal policy. But, it is bold to assume that every agent will have this information. In fact most agents will be interacting with a world that they initially know very little about. Therefore, agents must have some method of 'exploring' the world in order to learn aspects of the model and then once enough information is obtained the agent can 'exploit' the newly gained information to interact with the world. In this section, the results of running Q-Learning with a learning rate of 0.1 on both MDPs is discussed. These results are then compared to Value Iteration and Policy Iteration and the

A. Background

Q-Learning is closely related to Value Iteration. The difference is that the agent does not know its transition model. Therefore Q-Learning is known as a 'model free' reinforcement learning algorithm. Q-Learning is an algorithm where an agent tries to learn some function $Q(s,a)$ which is defined recursively below.

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') * max_a(Q(s,a))$$

Equation 10: Q-Value Update Equation.

$$U(s) = \max_a Q(s,a)$$

Equation 11: Utility in terms of Q Function

$$\pi^* = argmax_a\, Q(s,a)$$

Equation 12: Optimal Policy in terms of Q Function

B. Results

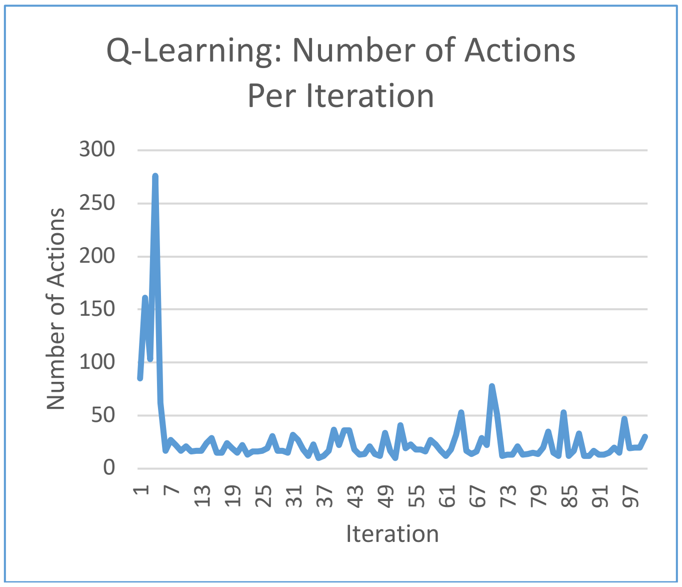

Figure 15: Small Grid. Total Number of Actions Taken before reaching Terminal State Per Iteration for Q-Learning

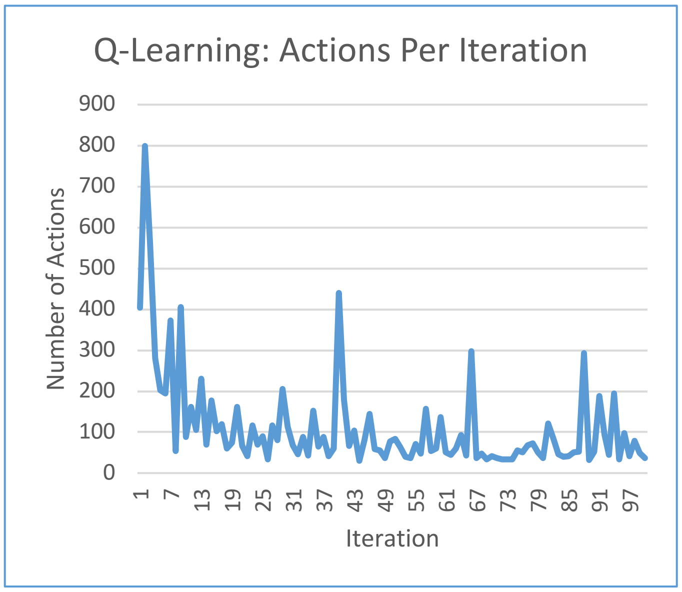

Figure 16: Large Grid World. Total Number of Actions Taken before reaching Terminal State Per Iteration for Q-Learning

Figures 15 and 16 are much more interesting than those produced by Value Iteration and Policy Iteration. They are much sharper and less stable. The sharp jumps in number of actions is a representation of 'exploration versus exploitation' in reinforcement learning which is fundamental. Exploration is represented by the sharp jumps as the agent is exploring more of the environment in order to gain more information about the environment. Exploitation is represented by the general decrease in number of actions over time. This implies that the agent is starting to use the information it learned about its environment to make better decisions. This assertion is supported in the following figures.

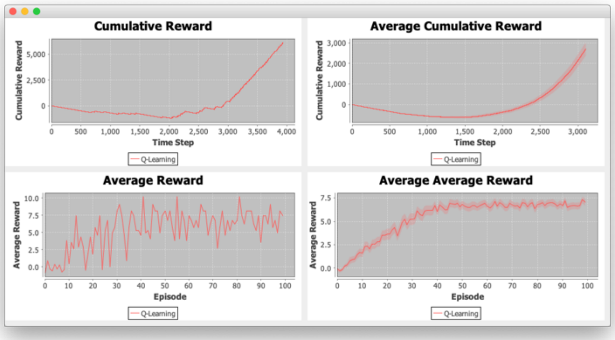

Figure 17: Easy Grid. Relevant Plots when running Q-Learning

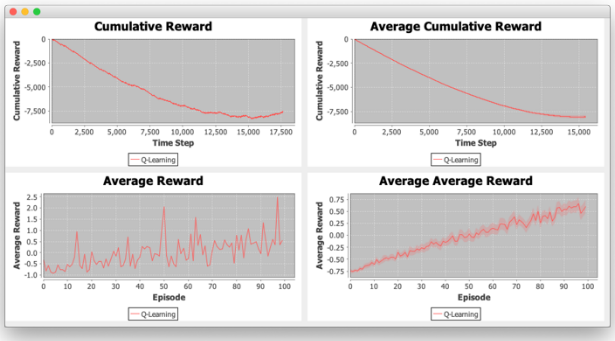

Figure 18: Large Grid. Relevant Plots when running Q-Learning

These charts show the results of Q-Learning when run for 100 episodes on both MDPs. The cumulative and average rewards start by decreasing but eventually slowly start increasing as the agent begins to learn relevant info about the environment and takes actions to exploit its knowledge. With a learning rate of 0.1, the agent prefers to exploit rather than to explore, but this can affect the starting episodes as the agent knows very little and could make some incorrect assumptions as it learns Q-Values. A possible modification would be to increase the learning rate for the first few episodes (let's say 10) and then decrease it after so that the agent performs mostly random actions to gain information and only exploits once it knows enough to make use of its information. It is important to note that the actions are still stochastic so sometimes the agent does attempt to execute optimal actions but is unsuccessful in doing so.

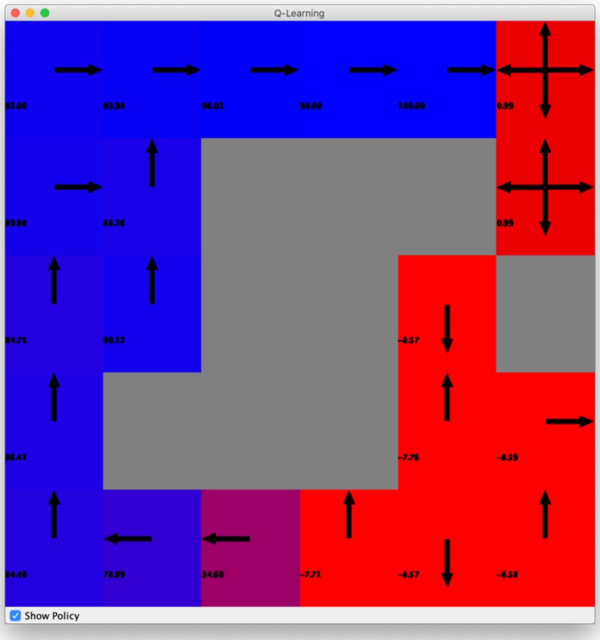

Figure 19: Q-Learning Convergence for Easy Grid

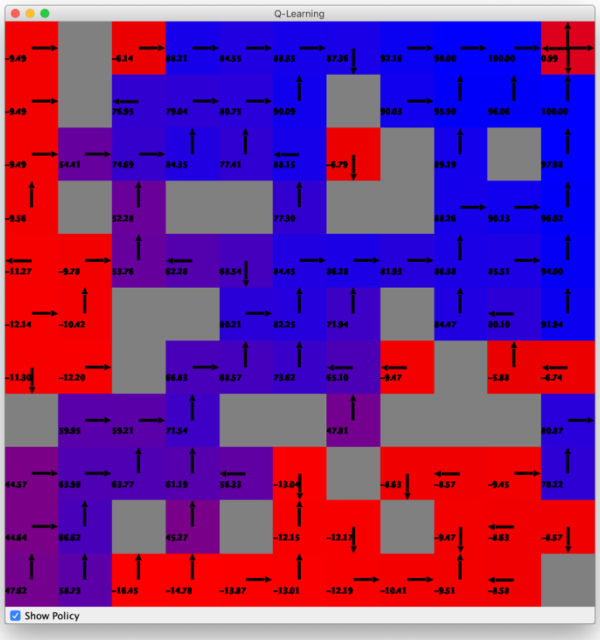

Figure 20: Q-Learning Convergence for Large Grid

The convergence charts for Q-Learning are much more interesting as they show exactly where the agent has explored and what states it has avoided exploring. Red states are states where the agent has either avoided exploring due to high negative Q values or because it already learned the relevant states and therefore is exploiting its knowledge to take the same familiar paths toward the goal state.

An interesting question to consider is what happens if there are multiple goal states? The convergence charts below show the effects of having an extra goal state in the top left corner that provided a reward of 150. The agent quickly learned to prefer this terminal state for all 3 algorithms.

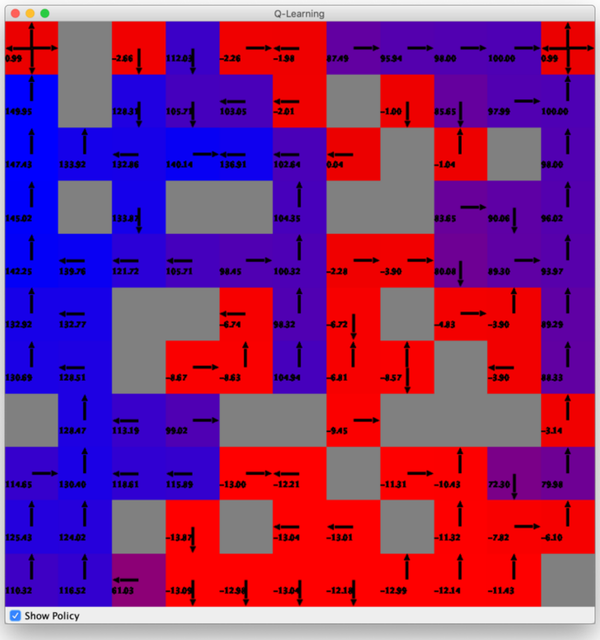

Figure 21: Q-Learning Convergence on Large Grid with Multiple Terminal States

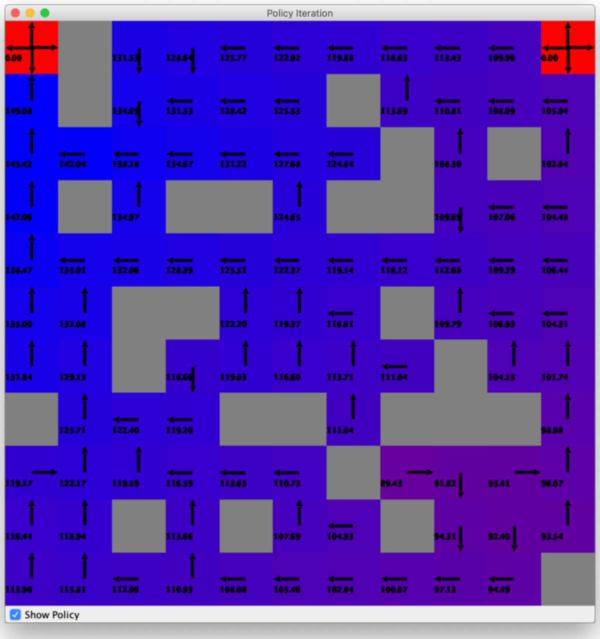

Figure 22: Value Iteration and Policy Iteration Convergence on Large Grid with Multiple Terminal States

IV. Conclusion

Through this introduction to MDPs, the two Grid domains have shown the effectiveness of Value Iteration, Policy Iteration, and allowed us to compare these algorithms with Reinforcement Learning algorithms. Considering all the information that an agent is forced to learn in a Reinforcement Learning environment, the agent still performed very well. Although Value Iteration and Policy Iteration outclassed its performance, Reinforcement Learning is more practically applicable, especially to partially observable environments. It would have been interesting to see the changes to utility calculations given more goal states. Also Policy Iterations seems to outperform Value Iterations for these Grid MDPs, however the intuition is that for a larger state space and with more possible actions, the policy update will become more inaccurate and will require more iterations in order to properly converge whereas value iteration's update will always move a step closer to the true utilities. Hence, although both runtimes will increase for both algorithms, Value Iteration will most likely begin to outclass Policy Iteration. Of course this hypothesis would require analysis on more complex MDPs and it is a step forward from the current analysis.

Acknowledgement

Special thanks to all the developers and contributors of BURLAP (cited below) for providing a useful intuitive framework for creating MDPs and for making several introductory tutorials on how to utilize the library. Also special thanks to Juan Jose San Emeterio for open sourcing some boilerplate code used for this analysis.

References

[1] Mitchell, T. M. (1997). Machine Learning (1st ed., Graw-Hill Series in Computer Science). McGraw-Hill education

[2] "Intro to Machine Learning Course | Udacity." Course | Udacity. Udacity, n.d. Web November 2017

[3] Scholz, J. (n.d.). Markov Decision Processes and Reinforcement Learning. Retrieved April 13, 2019, from https://s3.amazonaws.com/ml-class/notes/MDPIntro.pdf