Markov Decision Processes

Benjamin Vargas

Introduction

Two Markov Decision Processes (MDP) are thoroughly examined throughout this analysis through the lens of several algorithms. The driver code and plotting code used to generate the results in this analysis were written in Python 3, and the engine that runs the different algorithms to solve the MDPs is the BURLAP Java library. Additionally, an HTML file was used to generate the grid world MDP arrays that were input into the python driver code. Each algorithm and problem will be discussed individually, followed by an empirical, comparative analysis based on the results generated.

Markov Decision Processes (MDP)

A Markov Decision Process is an agent behavior model. To fit the definition of an MDP, there are four requirements: a finite set of states, a finite set of actions, a reward function that establishes the reward for visiting a certain state (although, this may be generalized), and a transition function that defines the result of taking an action from a state. Put simply, an MDP is a modelling for an agent to behave in a well-behaved universe. Well-behaved refers to having finite states and actions. An MDP can be stochastic (i.e., results of actions, rewards, etc. are probabilistic), in which case expectations based on probability distributions are used to compute values instead of using constants.

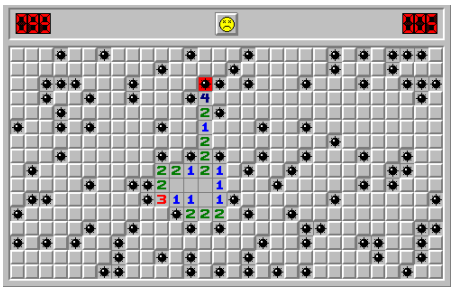


Figure 1 - Minesweeper

In this analysis, two variations of a grid world were created. A grid world is a discrete set of adjacent squares, where each square represents a different state. Each state has a different reward, which can be zero, negative, or positive. An example of a grid world MDP is the classic game of *Minesweeper* (see **Figure 1**[1]). In the game, each state contains a number representing its reward, or a mine that represents sudden death. In the MDPs I generated for this analysis, instead of having mines, I used negative reward values to discourage an agent from visiting a space unless necessary to obtain a reward that makes it worth it. The reward function is simply the reward for the input state. The transition model for a grid world MDP allows an agent four possible actions: It can move up, down, left, or right. If a wall (i.e., no space to move to, or the space is blocked) exists in the direction the agent is trying to move, it will stay in the same place. After an action is taken, the reward of the new state is received, and a new set of actions become available.

In this analysis, two MDPs were created; one problem that is comparatively easy and one that is more difficult.

MDP 1: Hot Summer Oasis

The first MDP defined in this analysis is a 5x5 grid world that simulates a desert environment, as shown in **Figure 2**. Most of the grid contains spaces with negative rewards, simulating the heat of the desert. The start of the grid world is shown in the bottom-left corner, and the goal of the MDP is to get to the "oasis" in the top right, with three spaces having positive rewards (two +3's and one +5). However, to get to the oasis, an agent must suffer through quicksand represented by four -3 spaces, and a -5-space adjacent to the oasis area. This

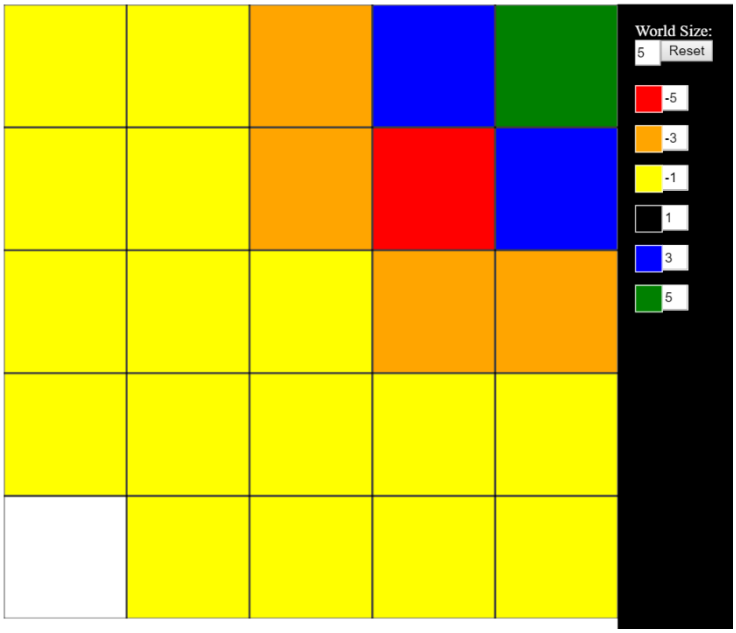


Figure 2 - Oasis Grid World

[1] Image taken while using use the game on the site (http://minesweeperonline.com/)

MDP is purposely small, to compare the different methods of optimizing the MDP using a small scaled world. Additionally, the grid has symmetry, which allows for two different optimal paths; one going right from the start, one going up from the start.

MDP 2: Nightmare Maze

The second MDP created for this analysis simulates a maze that contains booby traps, enticing reward traps, and a painful shortcut. Instead of having 25 states, this grid world has close to 400 states, using a 20x20 grid. The starting and ending state for the MDP are again, the bottom-left corner and the top-right corner, respectively. There are two paths to the goal: one path going up from the start, one path going right from the start. In order to entice the agent to actually explore the maze instead of getting stuck in a local minimum, the goal had to be increased from +5 to +100. It seems the agent does a hard pass at the thought of doing a Nightmare Maze for a measly +5 reward.

Figure 3 - Nightmare Maze Grid World

The path to the right is shorter, however, has two dangers: the hall of suffering that leads to the goal, and the reward trap that may prevent a greedy agent from ever reaching the goal (the five blue squares in the bottom right corner in **Figure 3**). The idea was to make both paths relatively valid to observe how the algorithms converge with different parameters. It was also designed to examine whether the algorithms will suggest an agent to collect the small rewards along the way, despite their dangerous proximity to the booby traps in the red squares (turns out, they suggest ignoring them).

Rewards, Utility, and Policies

Before discussing the various approaches to solve the MDPs in detail, the definition of rewards, utility, and policies for the two MDPs and algorithms examined in this analysis should be provided.

A *reward* is specified by the value that an agent receives when they arrive at any of the states in the grid worlds. For example, if an agent arrives in a red state, they will obtain a *negative* reward of -5, and conversely, if they land in a blue state, they will obtain a reward of +3. This value is constant in the MDPs in this analysis and is a simple function that maps 1-1 to each state. Utility is similar to reward, except it also considers future rewards in its calculation. Calculating the true utility of a state requires many iterations, because the future rewards slowly propagate to each state, instead of all at once. This future reward may be *discounted* using a discount factor between 0 and 1.

A discount factor of 1 implies that future rewards are as valuable as current rewards, so an agent will behave as if there is infinite time in this type of universe. On the other extreme, a discount factor of zero makes the agent behave wholly greedy, i.e., it would only care about immediate rewards when calculating utility. A discount rate on either end of the extremes (1 or 0) are not often used in practice. Instead, an intermediate value is often used. In my analysis for both MDPs, I experimented with discount factor values of **0.70, 0.90, 0.95**, and **0.99**. In the Oasis grid world, there was no significant change to results for any discount factor tested. In the Nightmare Maze, however, there was an interesting observation when discount rate was changed, which will be discussed in the **Q-Learning** section.

A *policy* in this context is a function that maps states to actions. For example, given an input of state (0,0), the starting point of the grid world MDPs, a policy function will return an action, such as up, left, down, or right. The optimal policy is one which maximizes total utility of an agent for all states. For this analysis, the common notation of $\pi(s)$ is used to denote a policy function, where $\pi(s)$ returns an action given a state s.

Value Iteration

Value Iteration is an algorithm that is used to calculate the true utilities of a state, using the Bellman Equation, $U(s) = R(s) + \gamma \, max_a\{\sum_{s'} T(s,a,s')U(s')\}$. Using the Bellman Equation, the utility of a state is calculated by adding the reward of the state to the discounted future rewards for what is *currently* the optimal action (i.e., one which maximized future utility). Actions are denoted with a, discount rate is denoted with γ.

It's important to note that the action which maximizes future rewards is only maximal with respect to the current values of the utilities for each state. Utility can be initialized differently, such as initializing everything to zero, or initializing everything randomly, and this initialization will affect the time it takes for Value Iteration to converge to an optimal policy. Finding the optimal policy can be done in different ways, however, the way Value Iteration converges to an optimal policy, is iteratively updating the utilities until a specified convergence threshold. In this analysis, $1e^{-6}$ was used as a convergence delta threshold for Value Iteration. The convergence value is calculated by the distance between the new utility values after a single iteration of Value Iteration, and the previous iteration's utility values.

When Value Iteration converges, it implies that the true utility (although, with a convergence threshold marginally above 0, a very close approximation) of each state has been determined. Using these utility values, an agent can, at each state, select the action which maximizes their utility for the state they are currently in. As such, a policy $\pi(s)$ can be easily generated from the return value of Value Iteration U. In practice (including for the two MDP's in this analysis), this policy can be seen as optimal, due to the very small convergence threshold used.

MDP 1 Results

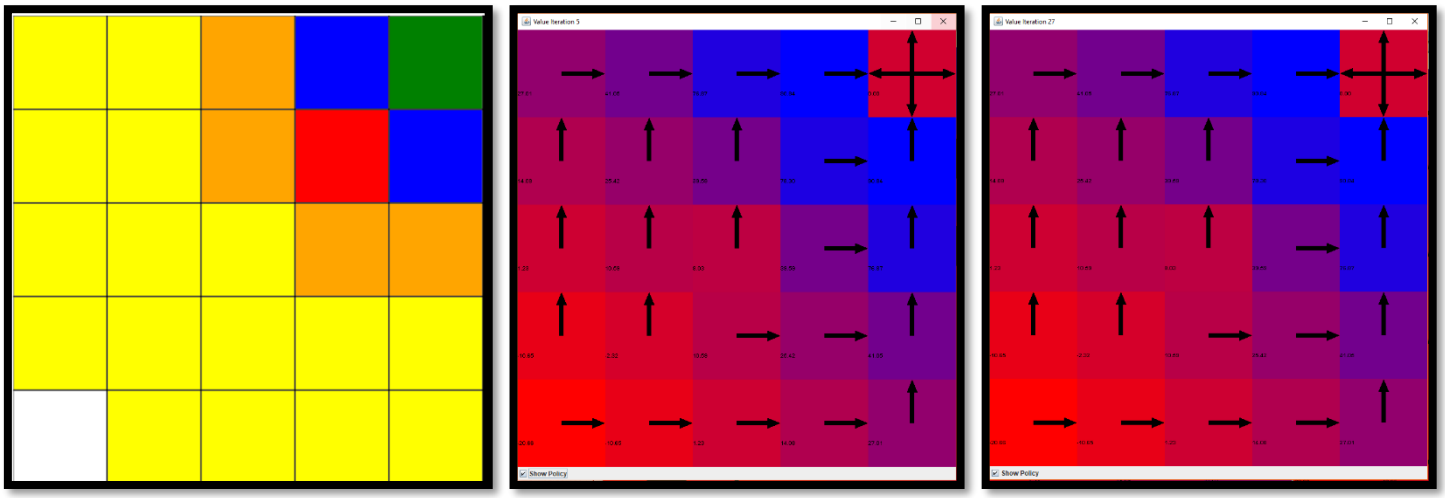


Figure 4 - (From Left to Right): Oasis Grid World, Value Iteration Policy after 5 Iterations, Converged Value Iteration Policy

After running Value Iteration on the first MDP, the optimal policy was found *very* quickly. This is intuitive, because the problem has a small number of states, and relatively low standard deviation of rewards. In **Figure 4**, the original grid world is shown on the left image, the output policy after 5 iterations of Value Iteration is shown in the middle image, and the final, converged output policy after 27 iterations is shown on the right. The two policy visualizations also include a heat map for the calculated utilities based on a **red** to **blue** color scale. **Redder** on the spectrum implies states with lower utilities, and vice versa for states that are **bluer**. **Purple** suggests that a state's utility is somewhere in-between. The goal state should be ignored for this heatmap analysis. An interesting observation is that the optimal policy was found at 5 iterations, however, the magnitude of the reward vectors (as shown in **Figure 4**) did not completely converge.

This means that, effectively, the optimal policy was already found after 5 iterations, however, the algorithm did not terminate, because it is calculating the approximate true utility values. Policy Iteration is the next algorithm discussed, and specifically seeks the optimal policy, and should therefore need less iterations to converge.

MDP 2 Results

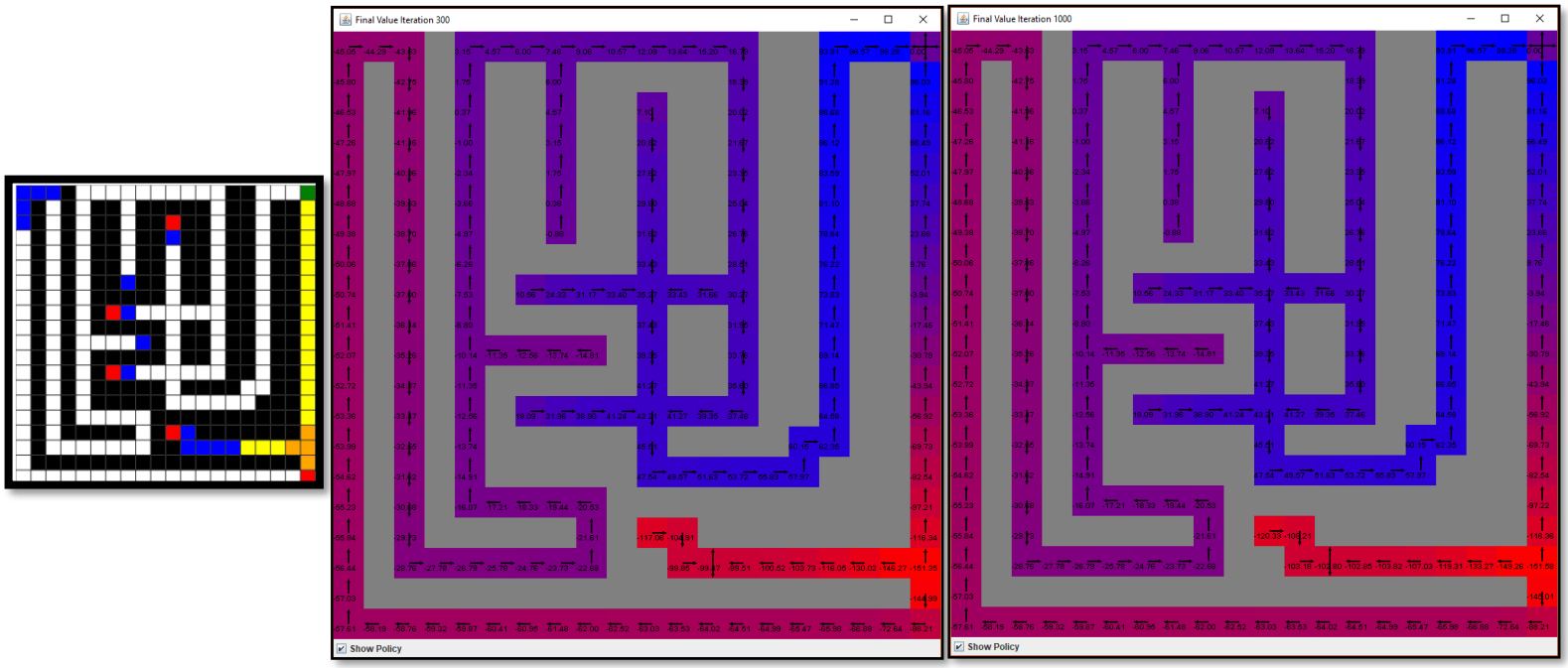


Figure 5 - (From Left to Right): Nightmare Maze Grid World, Value Iteration Policy after 300 Iterations, Converged Value Iteration Policy

Despite not converging within the maximum allowed iterations of 1000, Value Iteration found an optimal policy for the Nightmare Maze. Similar to the results for the Oasis MDP, although the policy stopped changing from 300 to 1000 iterations, the algorithm continued to fruitlessly run in an effort to find the stable utility values. As the problem scales up with respect to the number of states and actions, it is clear that Value Iteration has trouble converging the utility values. Another characteristic that may have made it difficult for the algorithm to converge is the booby trap areas, specifically the one in the bottom right. The utility values for these states likely fluctuate as the algorithm struggles to find a way out. It does seem that, if the shortcut is taken to reach the goal (using a discount rate of 0.99), then the reward deficit is very minor, so the vertical-from-the-start path won out only slightly.

Through experimentation with different discount rates, it seems that when a more short-term policy is searched for, it is too risk-averse to reach the goal from the start. The policy seems to oscillate an agent back and forth to try and avoid that massive losses associated with the shortcut and booby traps. This result was also observed for the other algorithms.

Policy Iteration

In the Value Iteration section, it was noted that, although the optimal policy was already found after only a few iterations of Value Iteration (for example, in **Figure 4**), the algorithm did not terminate because the utility values were still propagating. This is an obvious inefficiency, to which Policy Iteration does not suffer from. Instead of approximating the true utility values of each state, Policy Iteration instead attempts to approximate the optimal policy only, ignoring the magnitude of the action vectors (i.e., the action from the state providing the maximum utility).

It accomplishes this by using the Bellman Equation in a different manner. For each iteration, the utility values for a single policy is evaluated, i.e., the MDP follows the policy to determine the utility values for each state. Then, for each state, the utility resulting in the max utility is compared with the utility values evaluated for the policy. If a better utility is found, then the policy is updated for that state to reflect a more optimal action. When the policy stops changing from

one iteration to the next, then the algorithm converges and returns the optimal policy. The utility values are not guaranteed to be accurate, but the policy itself is. This effectively turns the max function problem from the original Bellman Equation into an argmax function, which allows the problem to be solved in deterministic polynomial time. Although Value Iteration has a polynomial time worst case complexity, the complexity can become non-polynomial with respect to the discount factor. Policy Iteration does not suffer from this problem, and a linear algebra solution can be used to guarantee a polynomial time solution.

In practice, this implies that Policy Iteration will scale better as the number of states increases, due to a fixed horizon of convergence. It also implies that Policy Iteration will require fewer iterations, although, the computation required for each iteration of Policy Iteration is heavier than what is required for Value Iteration. These hypotheses will be examined in the MDP results for Policy Iteration and the comparison of algorithms section.

MDP 1 Results

The results for Policy Iteration are shown in **Figure 6**. Unsurprisingly, the results for this MDP are nearly identical to the results for it in the Policy Iteration section. Again, to get a more granular understanding of the performance differences between Value Iteration and Policy Iteration, more detailed metrics will be examined in the algorithm comparison section. Observing the optimal policy for Policy Iteration is not as informative as it is for the other algorithms used to solve the MDP.

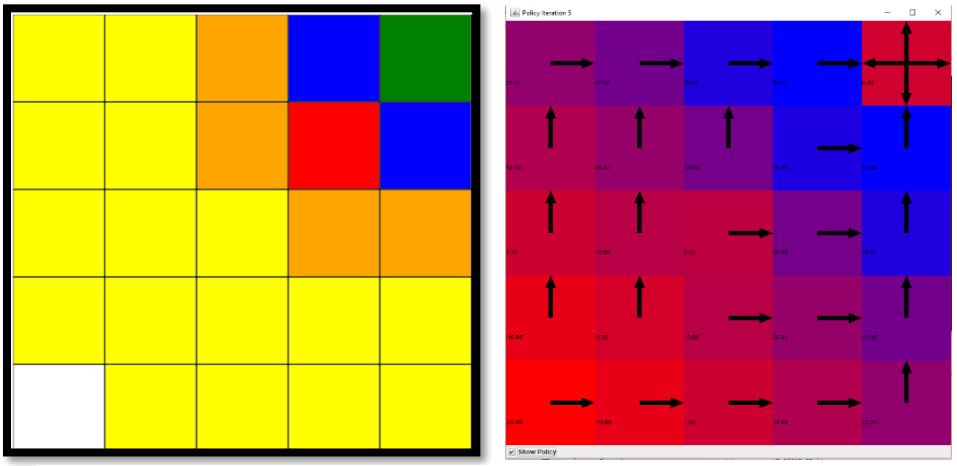


Figure 6 - (From Left to Right): Oasis Grid World, Policy Iteration Policy after 5 Iterations

MDP 2 Results

Because the scale of the MDP is significantly increased in the Nightmare Maze MDP, compared to the Oasis MDP, the differences between Policy and Value iteration become clearer. The optimal policy was found in 300 iterations, whereas Value Iteration failed to converge within 1000 Iterations. By comparing the results in **Figure 7** and **Figure 5**, the claim that the resulting policy is optimal is strongly supported, because both algorithms converged to the same policy.

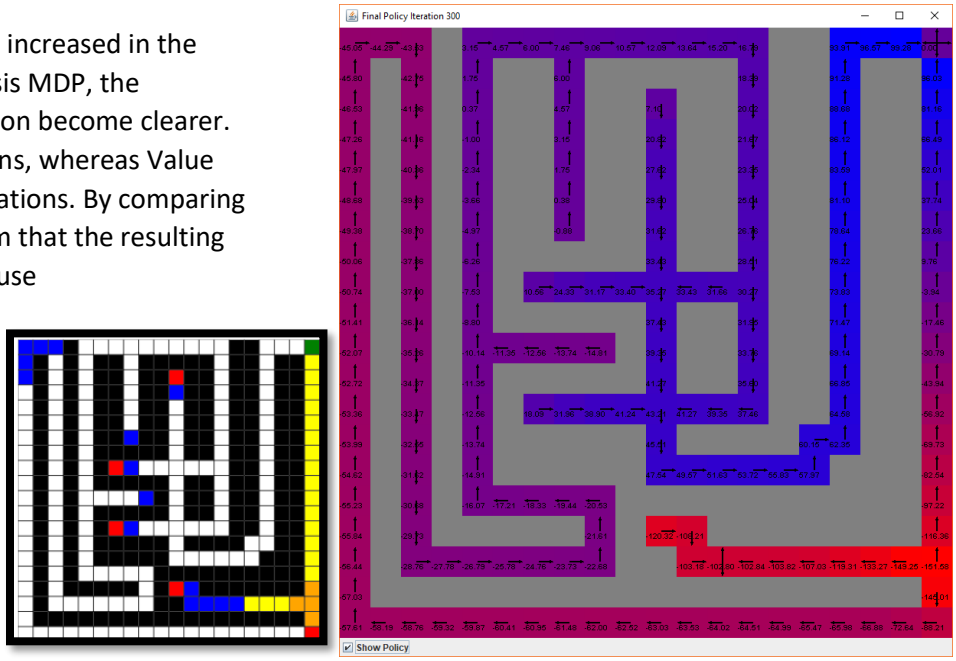


Figure 7 - (From Left to Right): Nightmare Maze Grid World, Converged Policy Iteration Policy

The convergence threshold was set conservatively such that the likelihood of an optimal policy being presented is high. This is important because for the final algorithm, Q-Learning, an optimal policy is not necessarily guaranteed with the implementation in BURLAP. As such, to accurately benchmark Q-Learning results, they must be compared to a policy that has a high likelihood of being optimal, such as the resulting policies in the Policy and Value Iteration sections. The convergence criteria for Policy Iteration could be decreased to simply be

when there are no changes to the policy from one iteration to the next, however, to be conservative, the convergence criteria were stricter. This also helped obtain more data to compare the different algorithms.

Q-Learning

Both algorithms previously discussed assume the transition model and reward function are known, however, this is rarely the case in practice outside of a subset of well-behaved problems. Instead, for cases where this model and function are not known, a model-free approach is required. The final algorithm examined in this analysis is the model-free approach for the Bellman Equation: Q-Learning.

As the name of this family of algorithms (somewhat) suggests, a Q-table is used to help guide the learning process to solve an MDP. Each entry of the Q-table is represented by the following modified Bellman Equation:

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

Hyper-Parameters, Action Selection and Convergence: Exploration Vs. Exploitation

The challenging part of any reinforcement learning problem is balancing exploration with exploitation. That is to say, to learn about the world an agent exists in, exploration is required, however, if the world is fully explored, then the results from the previous exploration are not informing the exploration, and the exhaustive nature of the exploration defeats the purpose of the algorithm in the first place. If the world can be exhaustively explored and explained, then there is little reason to generate a learning algorithm for that world. Conversely, if an agent only exploits what they learned instead of exploring possibly bad actions, then they are prone to falling into a local minimum, similar to the problem of overfitting. To combat this issue, it was necessary to tune hyper-parameters to receive good performance with Q-Learning on the MDPs, especially on the Nightmare Maze.

The first parameter to tune was the **learning rate**. This essentially weighted how much the Q-table is adjusted as new values are found through taking actions. A higher learning rate puts more emphasis on exploitation by weighing past experiences more, and vice-versa with a lower learning rate. In practice, I found that this value had significant impact on the results.

Initial Q Values bias the algorithm towards a certain convergence. In my analysis, I used three extremes to observe some obvious biases associated with initialization. Obviously, if the Q-table is initialized with values that approximately approach the median reward values, then it will likely converge quickly, however, the reward for each state is not always known without requiring an action, and the utility requires exhaustive search. As such, initializing Q values inconsistent with the states may insert bias into the algorithm, preventing it from finding a solution to the MDP.

Discount Rate and **Convergence Threshold** are two parameters previously already discussed, but it seems they had a more pronounced effect on Q-Learning results specifically. If the convergence threshold is too high, then the results were poor. Additionally, I found that lower discount rates for my MDP made Q-Learning very short sighted and poor performing with respect to approximating the optimal policy, getting trapped in local optima.

The final parameter tuned was **epsilon**, which represents the rate at which a random action is taken instead of the optimal action given the Q-table. If only optimal actions were taken, then there may be unexplored paths that are never found due to biases in the short-term rewards. By including random actions in exploration, the chances of getting stuck in a local optimum are reduced. This parameter was important; however, it was overshadowed by the other parameters, broadly speaking.

Each of these parameters will be discussed in detail below for each of the two MDPs. Once the parameters were selected, then the best performing Q-Learning runs will be compared with Value Iteration and Policy Iteration with respect to a few key metrics in comparison of algorithms section.

MDP 1 Results

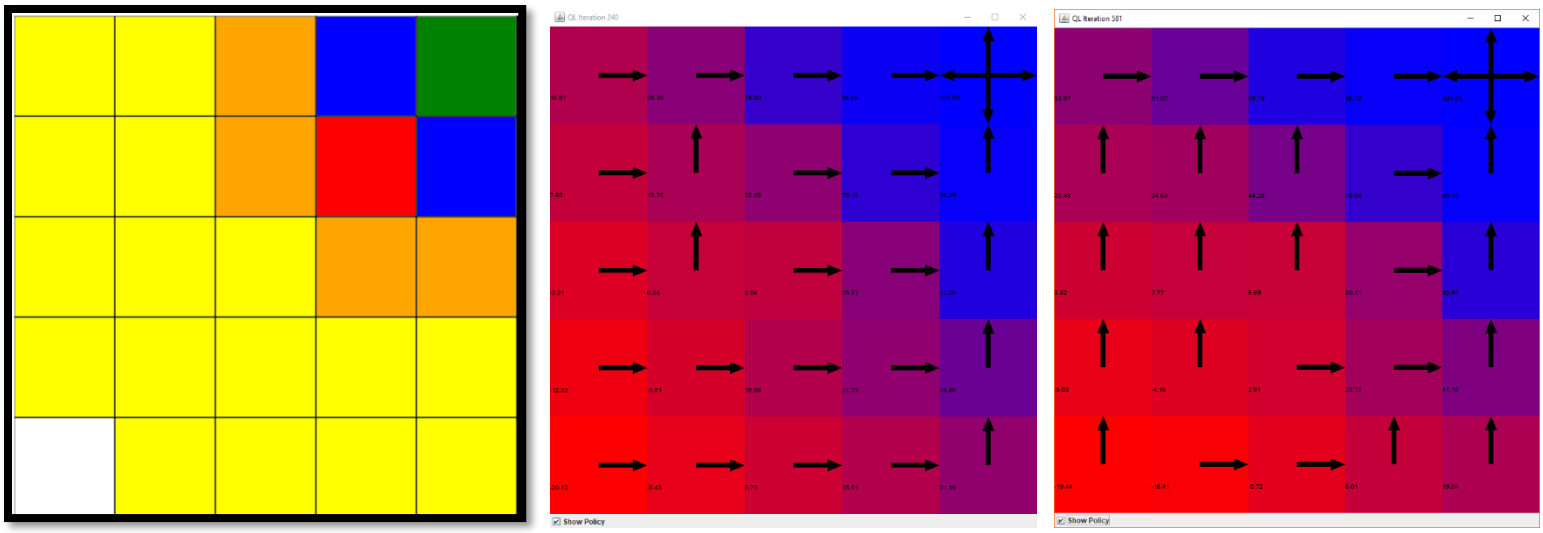


Figure 8 - (From Left to Right): Nightmare Maze Grid World, Q-Learning Policy after 240 Iterations, Converged Q-Learning Policy

As shown in **Figure 8**, a rough approximation for the best policy was found relatively well. There are some differences between the policy shown here, versus the policies shown in **Figure 4** and **Figure 6**. It seems that Q-Learning actually overfit the first problem, surprisingly. This could be fixed by decreasing the learning rate even further than, but Q-Learning simply a poor choice for a problem this small. As will be discussed in more detail later, Value Iteration and Policy Iteration perform extremely fast on small state spaces, such as this 5x5 grid. Due to the exploration and exploitation balance of Q-Learning, there are large deviations in the policies that prevent Q-Learning from converging on a good policy using standard param values.

Nevertheless, based on a series of runs using the parameters shown in **Figure 9**, the best values were selected before comparing results more empirically to Value Iteration and Policy Iteration. To identify the best discount factor, a series of output policies (such as shown in **Figure 8**) were examined, and it was clear that the discount factor had very little effect overall for this small of an MDP.

Hyper-Parameter	Values Tested	Best Value
Discount Factor	[0.7, 0.9, 0.95, 0.99]	0.99
Learning Rate	[0.9, 0.5, 0.3, 0.1]	0.1
Initial Q Values	[-100, 0, 100]	0
Epsilon	[0.1, 0.3, 0.5]	0.1
Convergence Delta Threshold	[0.1, 0.5, 1.0, 1.5]	1.5

Figure 9 - Hyper-parameters tested for Oasis MDP

The learning rate, however, was more drastic in impact. In **Figure 10**, the convergence over iterations was observed for each learning rate. As learning rate increases from 0.1 to 0.9, there was a clear increase in the variation of convergence. This suggests that the policy is wildly fluctuating and will not converge smoothly to a policy. This is also due to the fact that BURLAP uses an epsilon greedy approach instead of applying a slow epsilon decay (similar to slowing temperature in simulated annealing). Thus, 0.1 was clearly the best learning rate from the values tested.

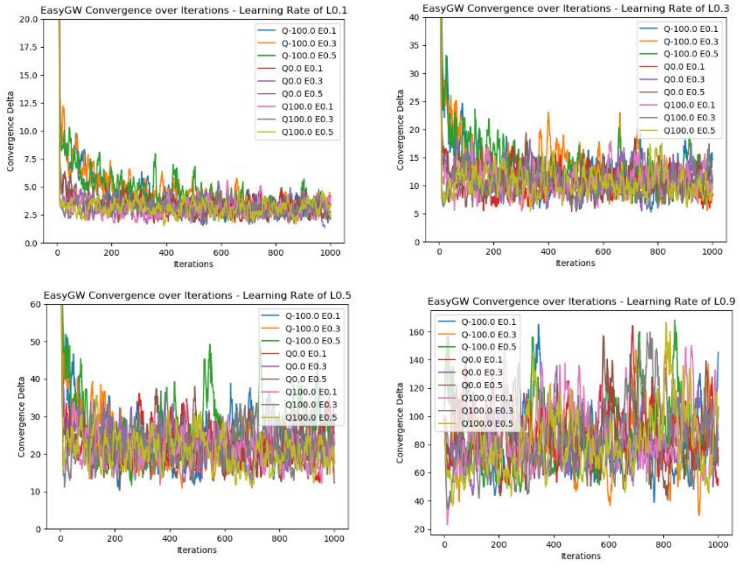


Figure 10 - Learning Rate Convergence Variance (Oasis)

The convergence delta threshold was tested by printing out convergence values to the console and observing their behavior relative to the output policy at hard-coded

iteration checkpoints. With this approach, it was clear that convergence minimized around 1.5 at best, and thus this was set as the threshold.

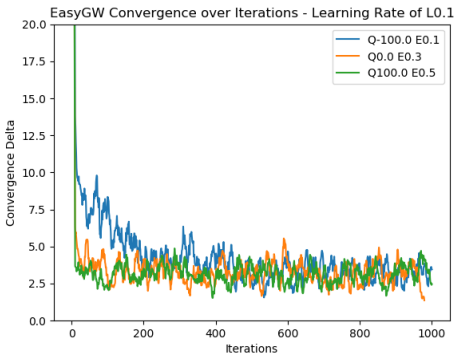


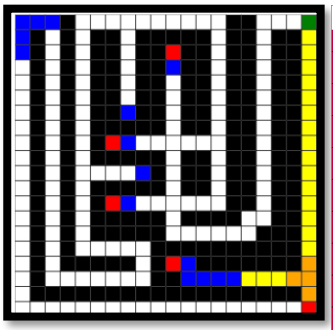
Figure 11 - Tuning Initial Q (Oasis)

To tune initial Q-values, one could just look at the grid and assign the initial value to the most common reward value, however, this is cheating somewhat when using a model-free approach. Instead, convergence over iterations was again plotted and observed in **Figure 11** using the best runs from each q-value.

A quick observation at the results show that initializing Q values to 100 results in the fastest convergence. While this is true, a sanity checks on the policies shows that this actually results in extremely poor policies, because initialization is so wrong that it goes into a local optimum. The other two initializations do not cause this, so initializing to 0 is the best option.

The final value, epsilon, was tested similarly as initial Q values, with 0.1 being the best value of epsilon for the Oasis MDP.

MDP 2 Results



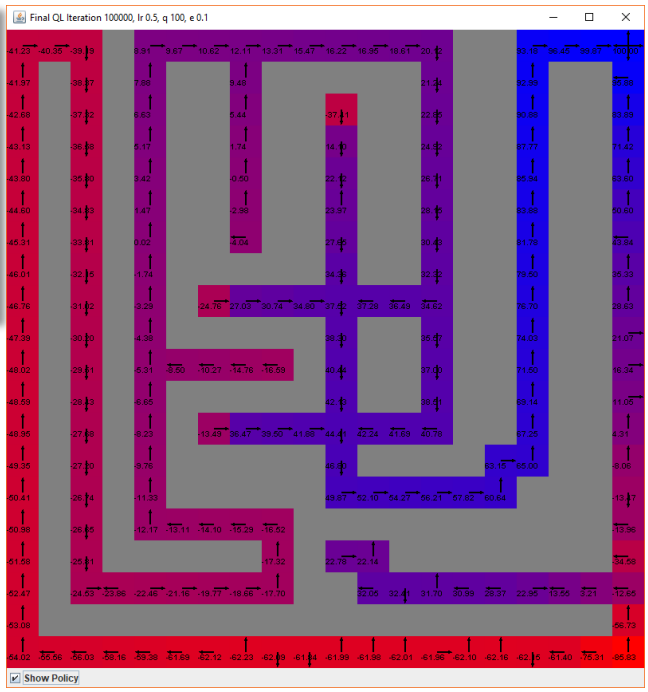


Figure 12 - Q-Learning Policy Nightmare Maze

The final policy for the Nightmare Maze MDP does find the best path to the goal, using the policy from Value and Policy Iteration as a benchmark. However, this required a **large** number of iterations, and excessive computation time. On the initial run, with 30,000 maximum iterations, the resulting policies were extremely poor, and even with 100,000 max iterations, the Q-table values fluctuated wildly in equilibrium, although the optimal path from the starting state was preserved, as shown in **Figure 12**.

Tuning the hyper-parameters was performed using the same methodologies as described in the Q-Learning MDP 1 results. Something interesting to note, is that when the learning rate is too low (e.g., 0.1), the policy gets trapped in a local minimum per the booby traps. A higher learning rate was required in order to reach the goal without getting stuck in the shortcut trap. Similarly, initializing the Q values too incorrectly (i.e., negative 100) caused the same issue to occur, and the optimal path was never found in that case. So even though Q-Learning converged, it converged extremely fast (in approximately 15 iterations) to a poor policy. Lowering the discount factor from 0.99 caused this issue as well. To find the optimal epsilon value, the policy maps were examined, with 0.3 providing the results that most closely matched the results for Policy Iteration and Value Iteration.

Hyper-Parameter	Values Tested	Best Value
Discount Factor	[0.7, 0.9, 0.95, 0.99]	0.99
Learning Rate	[0.9, 0.5, 0.1]	0.5
Initial Q Values	[-100, 0, 100]	0
Epsilon	[0.1, 0.3, 0.5]	0.3
Convergence Delta Threshold	[0.1, 0.3, 0.5, 1.0, 3.0]	3

Figure 13 - Hyper-parameters tested for Nightmare Maze

The best hyper-parameters found for the Nightmare Maze are summarized in the table in **Figure 13.**

Comparing the Algorithms

After determining the best Q-Learning parameters for each MDP, the best run (using these parameter values) was compared to the results of Value and Policy Iteration. The metrics compared between the algorithms are wall clock time, iterations to converge, and the reward from the respective policies. Each value was plotted over iterations to provide a consistent analysis between the different metrics.

MDP 1 Results

The first comparison between the algorithms was the number of iterations to converge. In **Figure 14**, the results are shown. Within the first few iterations, Policy Iteration (PI) spikes, however, it then plummets to converge to the optimal policy very quickly, closely followed by Value Iteration (VI). Q-Learning lags behind the other algorithms, which is expected because it is a model-free approach. An interesting thing to note is that, while the other algorithms converge to nearly zero, Q-Learning never settles on the optimal path. This is explained by the fact that the algorithm is probably too greedy to converge to the absolute best policy and would require an epsilon and learning rate decay strategy to converge to zero.

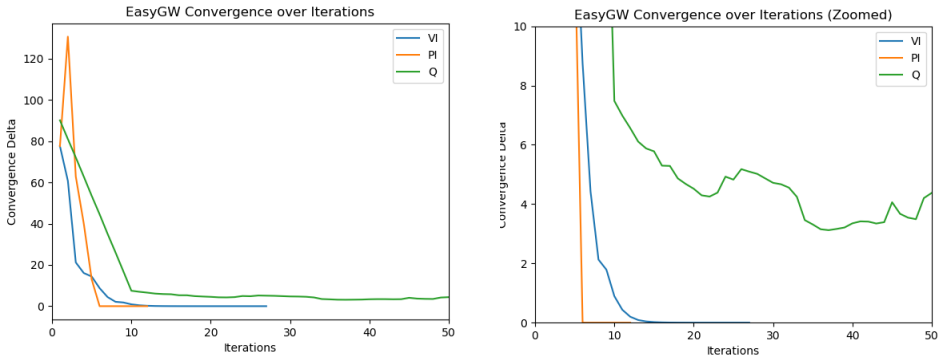


Figure 14 - Comparing Iterations to Converge

The next comparison (**Figure 15)** was with respect to wall clock time. As expected, although PI converged in fewer iterations, it required more time overall to converge than VI. Thus, at small scales, Value Iteration is probably a better choice overall. Q-Learning clearly took more time to compute, due to the fact it never truly converges. It does appear that the time to compute scales linearly, whereas VI and PI scale in quadratic or worse time, which makes a case for using Q-Learning with very large-scale problems.

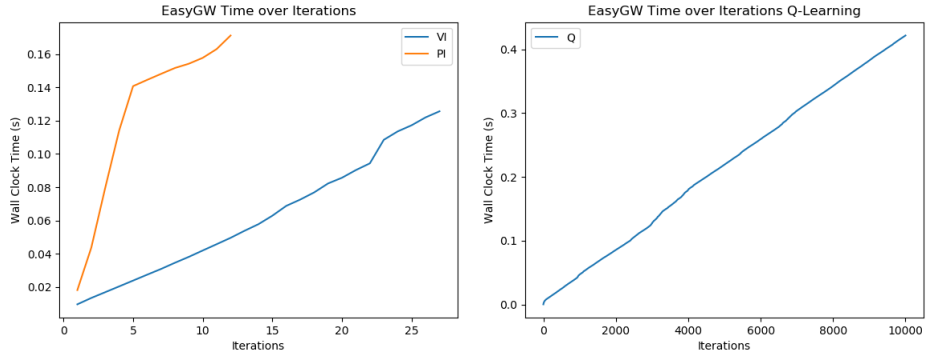


Figure 15 - Comparing Wall Clock Time

The final comparison in **Figure 16** reveals an interesting detail. Even though **Figure 14** suggests convergence for PI at 6 iterations, it seems in fact that convergence was achieved at 2 or 3 iterations instead, even better than previously suggested. Rewards plateau at this value, suggesting changes are not significant from that point forward. Similarly, Q-Learning actually converges around 75 iterations effectively, since it hovers around the best-case reward of zero around that number of iterations. As Q-learning continues to run, it only ever fluctuates for the worse, so the algorithm could have been stopped at 75 iterations, making it actually faster than the other two if convergence is stopped at 75 iterations. The scale of the time graph makes this difficult to visualize, but even at 2000 iterations, Q-Learning is under 0.1 seconds!

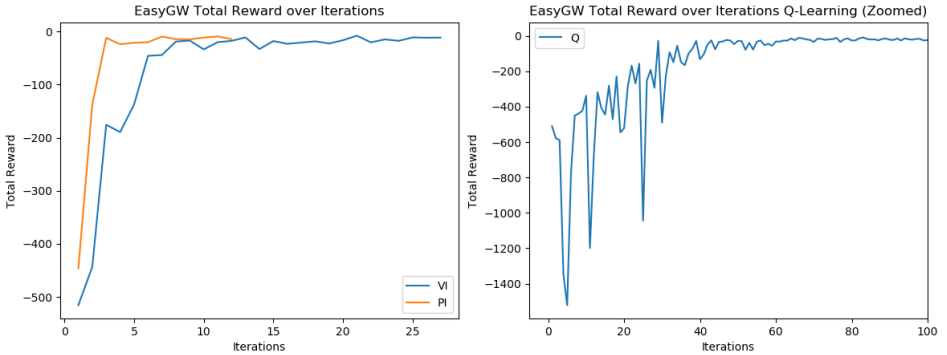


Figure 16 - Reward over Iterations

MDP 2 Results

The results (**Figure 17**) for the Nightmare Maze MDP support most of the previous hypotheses and illuminate some interesting characteristics for each of the algorithms. By simply observing the convergence over iterations plot in the top-right of the **Figure 17**, PI unsurprisingly converges in the least number of iterations, but surprisingly, Value Iteration required more iterations to converge than Q-Learning given tuned hyper-parameters. In a realistic simulation, the correct hyper-parameters would be difficult to tune, however, at extremely large scale this does support an argument in favor of Q-Learning. In fact, it seems possible that Q-learning may do better than policy iteration even, given a large enough scale.

Through observation of the time over iterations plot for the Nightmare Maze, the results seem largely consistent with the results from the Oasis MDP. It seems that, regardless of difficulty of the MDP, Q-Learning is unaffected with respect to clock time per iteration!

The remarks made regarding the reward over iterations in MDP 1 for VI compared to PI also hold consistent for the Nightmare Maze, but a new trend is indicated by the bottom-right Q-Learning reward over iterations plot in **Figure 17**. Once the optimal reward policy was found, Q-Learning oscillates wildly between two optima, one being a local optimum, the other the global optimum. It seems Q-Learning had a hard time converging consistently, likely due to the purposely designed booby traps in the maze and the high values of learning rate and epsilon, which explains the exhaustive computation time required to collect the data. A version of the algorithm with a slow decay on learning rate and epsilon would help the algorithm settle on the global optima and would be interesting to examine more closely.

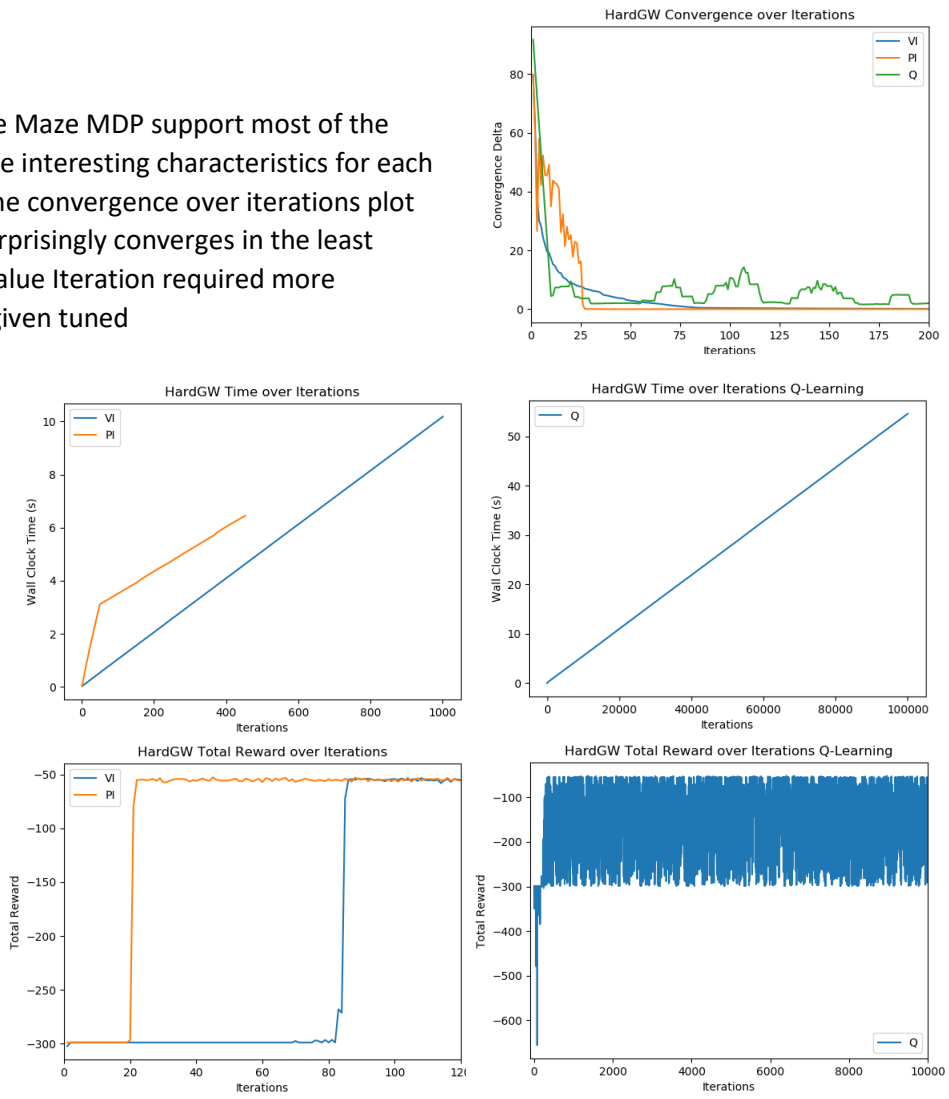


Figure 17 - Comparison of Algorithms (Nightmare Maze)

Closing Remarks

The results of the analysis provide strong arguments for each of the algorithms depending on the use case, which seems to be a relentlessly recurring motif of machine learning concepts. The apparent difficulty of generalizing an algorithm for the wildly different nature of various problems has been made clear throughout the several in-depth analyses performed on the popular machine learning algorithms. It is no different for reinforcement learning and MDPs. That being said, for the problems designed for this analysis, it seems that Policy Iteration works best for all things except computation time, which matches the theoretical understanding about these three algorithms, especially for well-structured problems like the grid world MDPs (i.e., Oasis and Nightmare Maze) discussed in sections above.