Assignment 4: Reinforcement Learning

CS 7641 Machine Learning
Spring 2017
Karel Klein Cardeña

Introduction

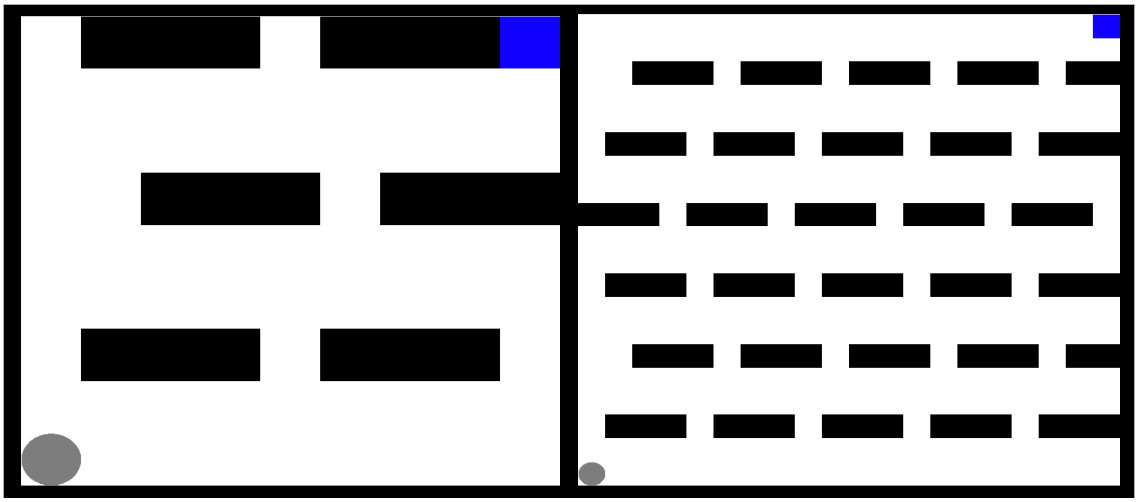
In a Markov Decision Process (MDP), an agent exists in an environment with a set of states S, where it can perform an action a from a set A. At each time step t, the agent perceives the state and decides which action to perform given the current state. The environment then provides feedback to the agent in the form of a reward, and gives the agent access to the succeeding state. In such a world, it may seem advantageous to keep track of all prior states visited in order to accumulate a history from which future decisions can be deduced. However, maintaining a long sequence of events is computationally cumbersome and often times unnecessary. In a MDP , we limit the number of previous states to 1 and aim to calculate the optimal policy in a tractable and efficient manner. Therefore, the next state s' is dependent on the current state s and the agent's action a, but independent of all previous states and actions. Thus, the MPD satisfies the Markov property. Additionally, we make the assumption that the transition probabilities do not change over time, and can therefore be represented by a stationary distribution.

With this environment in mind, the goal of the agent is then to learn a policy -- or a mapping from states to actions that informs the agent what decision to make given its current state. Moreover, the goal of the agent is to learn the optimal policy, that is, the policy which maximizes the sum of the discounted expected utility of each state which it visits. We now present two problems and three algorithms with which to solve them in an effort to contrast these methods' efficacy in discovering the optimal policy.

MDP Problems

To examine the performance of the three algorithms, we set the problem stage to Grid World, the classical setting used for testing MDP algorithms. In Grid world, we have a two dimensional landscape depicted by a square in which a grid defines the states of the environment. Grid world contains an agent and a terminal state, which the agent will aim to reach in the least number of actions. Additionally, the landscape is laden with walls that add complexity to the landscape and make the path to reach the absorbing state less obvious. Each grid block in the map represents a state which the agent may navigate to.

To further differentiate between the performance of the algorithms, we create two worlds: an "Easy" world containing 63 states (left) and a "Hard" world containing 312 states (right). In each grid, walls are represented by black rectangles, the terminal state by a blue square, and the agent by a grey circle.



The agent is incentivized to reach the absorbing state by the reward function set forth, where each non-terminal state represented by the white squares awards the agent -1 point, and the terminal state in blue awards the agent 100 points. The actions

available to the agent are contained to the set {move left, move right, move up, move down}. Furthermore, a transition function defines the likelihood that the agent's desired move will take place. In these grids, the agent's probability of that its selected action is implemented successfully is 80%, while the probability it moves in any other direction is 6.67% for each of the three alternatives.

With a reward function, transition function and terminal state in place, a Markov Decision Process is defined in which our three algorithms can now experiment.

Grid World Motivation

To the unrefined eye, using grid world as a setting for reinforcement learning may seem overly simplistic. Indeed, the layout of each grid contains only one type of obstacle in a 2-dimensional environment. It is worth considering, however, that generic environments often offer the best of testing grounds. By constructing an environment whose simplicity can hardly be outdone, grid world provides the bare elements which make up a Markov Decision Process. As a result, this setting contributes little bias to our goal of finding the "best" algorithm.

To further defend its simplicity, a grid with obstacles is analogous to a physical map with obstructions through which a robot must navigate to reach a goal state. A (near) futuristic robot tending to patients in a hospital, for example, would have a physical start state such as a docking station, and a terminal state such as a hospital room. In order to navigate the often chaotic map of the hospital effectively, it would need not to act on preprogrammed instructions, but rather to learn to take actions which avoid walls, passing employees, and sensitive areas while still reaching the target location with the shortest path. Of course, such an automaton would have a much larger set of possible states and set of potential actions than those of grid world. However, the essence of the problem remains quite similar: given a world with finite

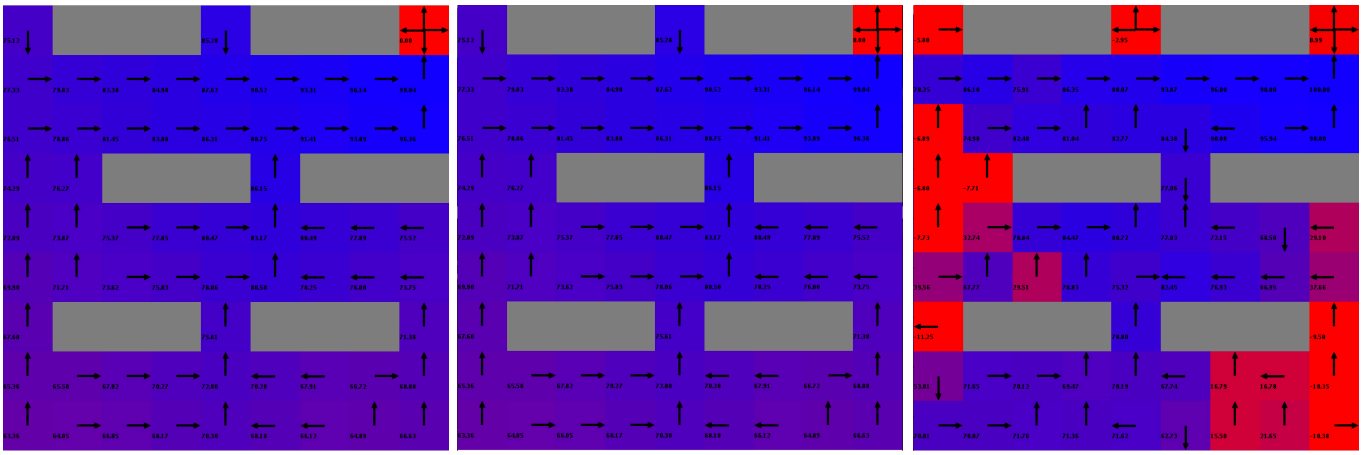
states and actions, an autonomous agent must learn the optimal policy to reach its absorbing state.

While grid world may present an slightly overly simplistic setting in which an agent is to operate, it models well the applications which we may see robots take in the near future. Furthermore, its visual representation allows for intuitive understanding of results and provides insights into the decisions reached by various algorithms.

Easy World Results

To compare the relative performance of the Value Iteration, Policy Iteration, and Q-Learning algorithms, we apply them first to the aforementioned "Easy" grid and analyze the optimal policies realized. Taking advantage of the intuitive design of grid world, we can then visualize the policies each algorithm reached after 100 iterations.

[Value Iteration] [Policy Iteration] [Q-Learning]



In these grids, we notice a red-to-blue color gradient. This gradient helps visualize the discounted future value the policy calculates for each state -- a value closer to the terminal state reward of a 100 is represented by a light shade of blue, whereas a low value is closer to the red end of the spectrum. As we can see, both Value and Policy Iteration were able to converge to the optimal policy, which we can verify as the correct

optimal policy by ensuring that each state's assigned action is the one producing the shortest path to the absorbing state.

Value Iteration (VI) seeks to choose the action that maximize expected utility. However, a state's true value (utility) is not initially known to the agent. To derive expected utility, VI calculates the sum of the immediate reward of the state and the expected discounted reward if the agent were to act optimally from then on:

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

For each state, VI then calculates each state's utility based on its neighbors utilities, and repeats this process until convergence is reached. This algorithm, while guaranteed to converge, works by estimating the true value of each state in order to derive the optimal policy.

In contrast, Policy Iteration (PI) works by iterating through policies. It first starts with a random policy, and then calculates the utility of each state given that policy. With these utilities, it then selects the optimal action for each state. This loop is repeated until no updates are made to the policy, indicating that the optimal policy has been reached.

The key assumption that both VI and PI make is that the agent has knowledge of the transition function as well as the rewards for each state in the environment. While this is not a far fetched assumption in our grid world setting, having access to this information is simply infeasible. A robot dropped on the surface of Mars, for example, would be hard pressed to have access to this information prior to landing. With this in mind, it is unsurprising to see that both VI and PI were able to discover the optimal policy given 100 iterations. In each case, the optimal policy found is verifiably the the solution to the Easy grid: we can see the concatenation of actions at each state yield the shortest path to the terminal state. Looking at Q-Learning's policy, however, we notice it did not achieve the same level of efficacy in computing the optimal policy.

Q-Learning (QL) attempts to construct the optimal policy without knowledge of the transition function or the rewards of all states. Using only knowledge of which states exist and the set of actions that the agent can take, QL proceeds as follows. First, it estimates a Q-value for each state. It then visits a state and uses the reward received to update the Q-value for that state by adding the immediate reward received with the best possible state-action pair for the next state. Using this method, the Q-value of a state-action tuple can be updated with the rule:

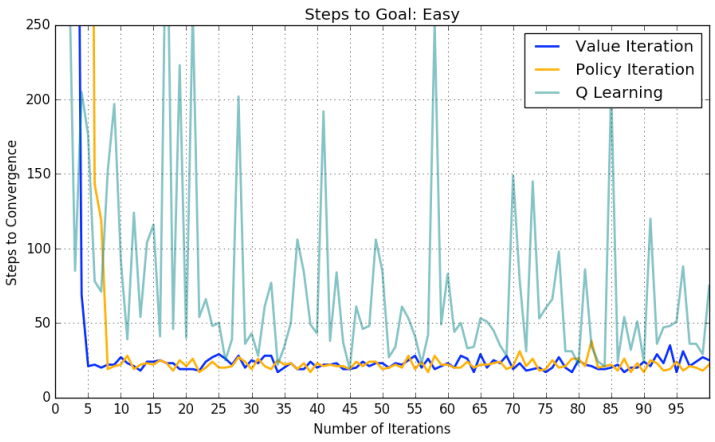
$$Q_{t+1}(s,a) = r + \gamma max_{a'} Q_t(s',a').$$

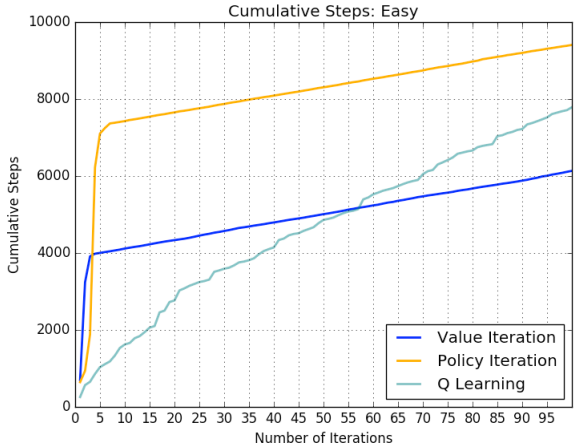
The resulting tradeoff is the one of explorations versus exploitation. If we explore more than exploit, we risk not learning an optimal policy efficiently, where as the opposite strategy may mean we do not explore states enough to find new, potentially better, paths to the goal.

In this implementation of QL, we use an ϵ-greedy exploration approach, where 90% of the time we choose to exploit by selecting actions with the highest Q-value. As we can see in QL's policy grid, it struggles assigning optimal actions to states near the edges and corners of the grid. This is likely a result of both our selected exploration approach as well as the number of iterations allotted for Q-Learning to learn the policy. The algorithm is unlikely to visit often the states in the periphery (as they do not produce high Q-values), and as a result we can expect suboptimal actions to be assigned to these states. In contrast to the policies learned by VI and PI, it is expected that QL would not achieve the same level of proficiency given its much weaker assumptions about the model in which it operates.

To more precisely differentiate the rate of convergence of the three algorithms, we conduct an experiment running each for 100 iterations, and simulating how many steps each takes to reach the absorbing state. Since all states carry a rewards of -1 save

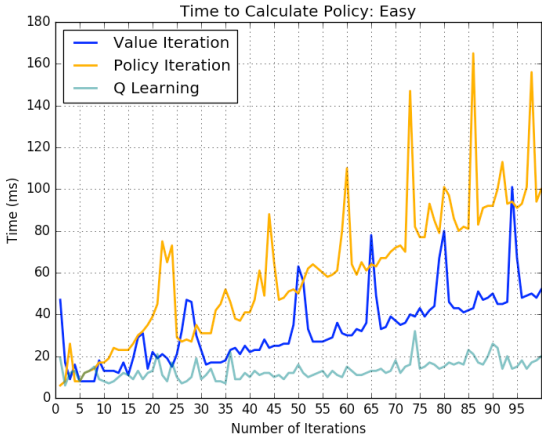
the terminal state, we can expect the number of steps to be commensurate with the reward achieved by the policy. [*y-label (left) should read *"Steps to Goal State"*]

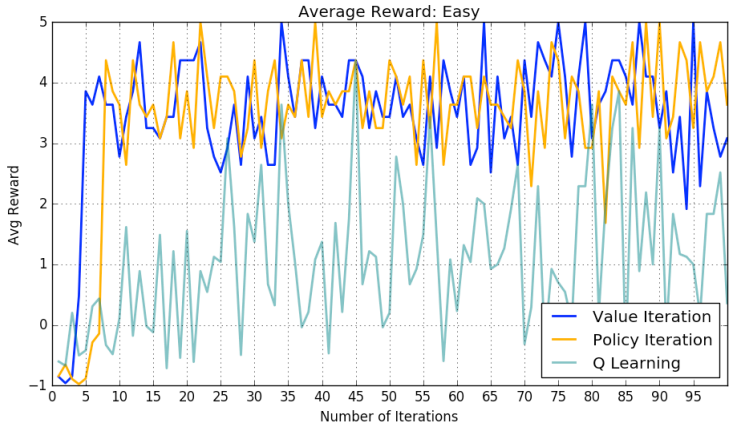




We can see Value Iteration converges to the optimal policy in just 5 iterations and Policy Iteration in just 7. Meanwhile, Q-Learning requires 25 iterations to reach the same point, after which it remains volatile and fails to reach the minimum number of steps with a much greater rate than the other two algorithms. These results meet our expectations. Utilizing far fewer resources to conduct its search, QL will almost always take more iterations to come to a policy of comparable efficacy.

To expand the analysis of runtime, we can also measure the time in milliseconds that each algorithm takes to calculate a policy.





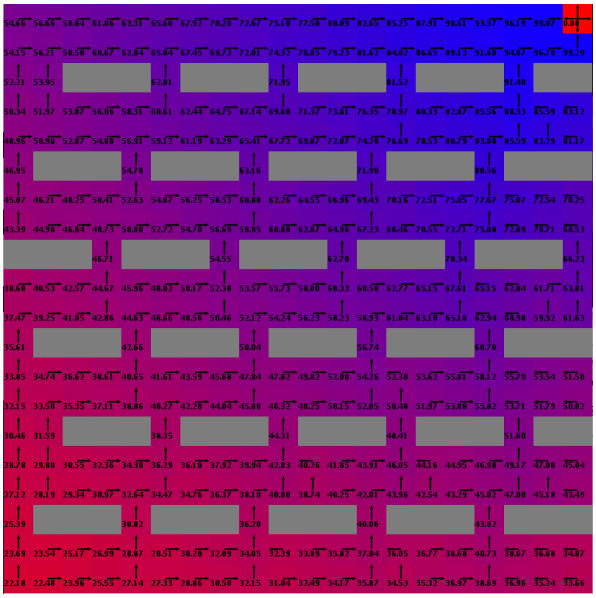
VI and PI have running time linear in the number of iterations, with Policy Iteration having the largest slope. Since Value Iteration avoids calculating distinct policies and utilizes values to update its optimal policy, it will avoid the heavy computational cost associated with computing a new policy per iteration step. Q-Learning appears to have near constant time for computing the policy regardless of the number of iterations, an advantage which would prove very relevant when the number of iterations required to reach an optimal policy is huge.

Looking at the average reward per iteration graph, we can intuit that despite believing that QL was the most volatile in reaching the optimal policy, all three algorithms exhibit similar variance when measuring the achieved reward relative to the number of iterations performed.

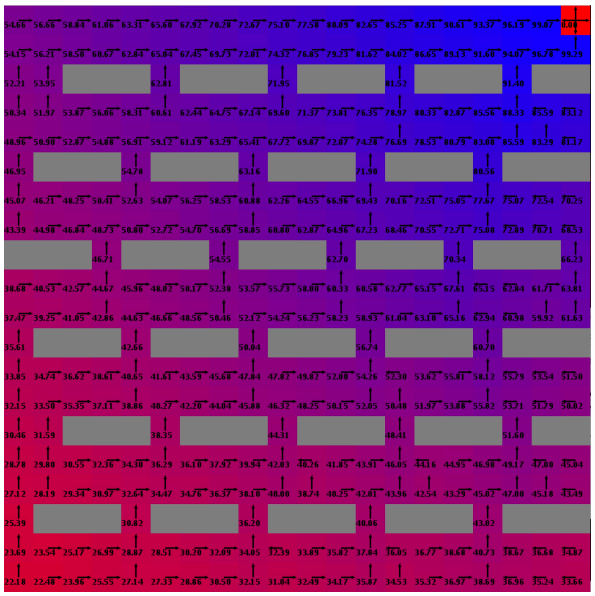
Hard World Results

With results from a small state set in hand, we look to expand experimentation to include the large grid with 312 states and an increased number of obstacles. Applying the three algorithms to this "Hard" grid, we find the below optimal policies reached by each.

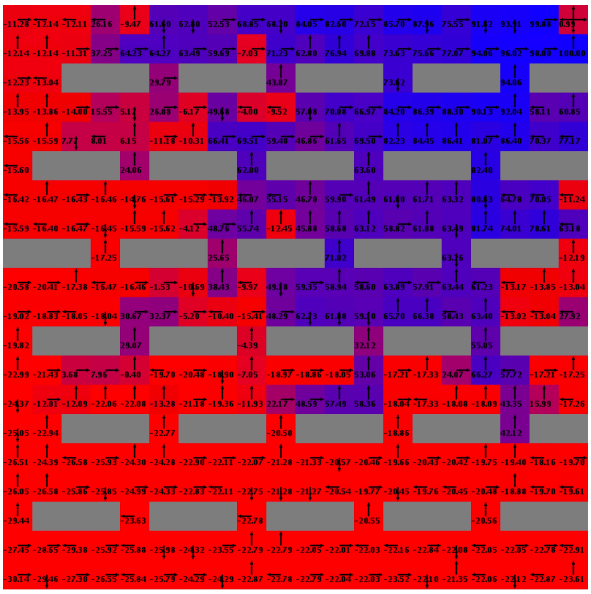
[Value Iteration]



[Policy Iteration]



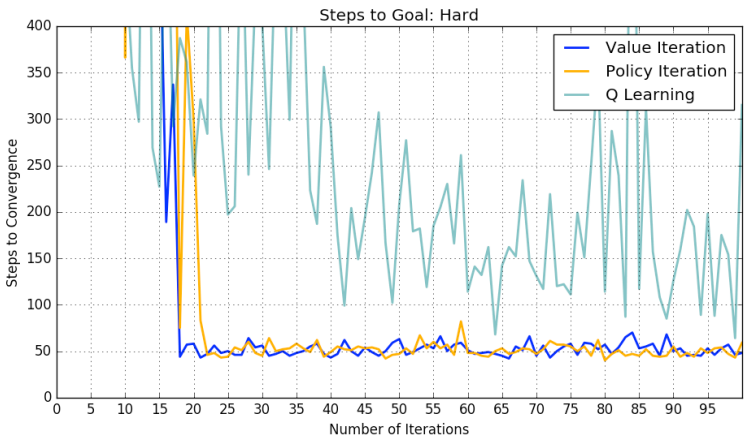
[Q-Learning]

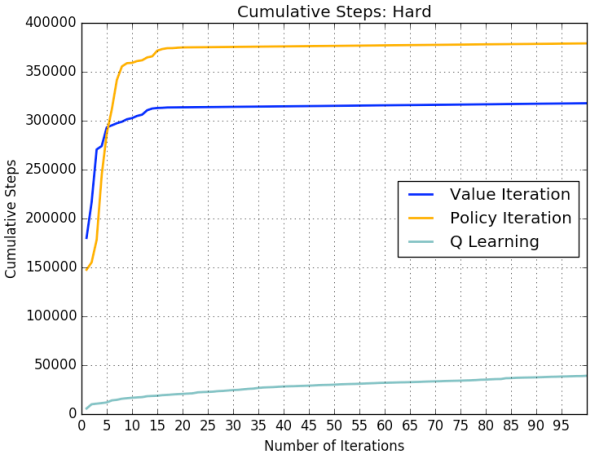


As we did with the "Easy" grid, we notice that VI and PI correctly solve the MDP by constructing the optimal policy. The color gradient observed in these two maps, in addition to the arrows, serve to demonstrate the smooth increase in value as states approach the goal (given the respective policies). Q-Learning seems to struggle with this large-state grid. We can see proof of this by looking at the opposing arrows in state's near walls and obstacles. The drastic change in hue from red to blue indicates that the QL is beginning to learn the behavior desired, but has not refined its policy enough to reach optimality. With more time to conduct experiments, I would look forward to increasing the number of iterations for QL in an effort to see whether it would eventually come to the same conclusion we saw VI and PI come to.

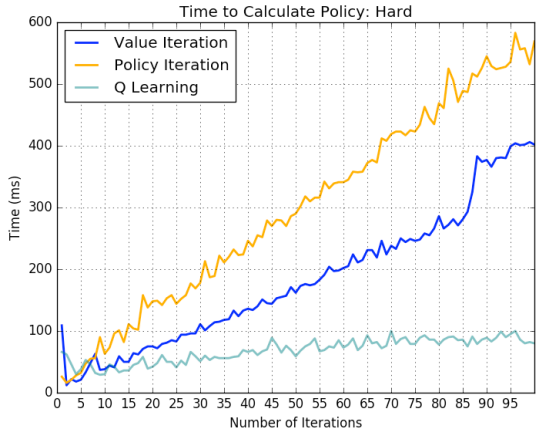
Although we did not see QL converge on the optimal policy, we can look at the rate of convergence graphs to get a better idea of trends that may exist.

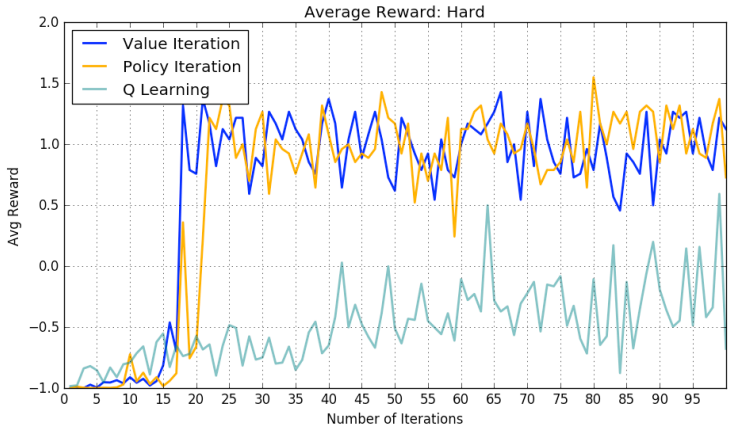
[*y-label (left) should read *"Steps to Goal State"*]





The results are interesting and reinforce the desire to conduct an experiment with more iterations. As we saw in the small grid, we see VI and PI reach the optimal policy quickly, the former in 17 iterations and the latter in 24. In a grid with 5 times as many states, the iterations required to reach convergence grow linearly. In the case of QL, there is a downward trend as the number of iterations increase. Although the optimal policy is not reached, this slope of QL line forecasts convergence to the optimal path provided a larger number of iterations. Upon doing so, we can predict the policy map to look a lot more similar to those produced by VI and PI.





In terms of time to calculate policy, the same results are observed as in the easy grid, where relative speed of each algorithm is the same, but each multiplied by a factor equal to the increase in grid size (each takes 3 times longer than in Easy grid).

Conclusion

By constructing two MDP problems of differing size, we were able to apply three algorithms and analyze how each performed in finding the grid world's optimal policy. We found Value and Policy Iteration the quickest the reach convergence, but asserted the key difference that they operate with full knowledge of the transition and reward functions. Q-Learning thus proved to be a computationally efficient and robust method of reaching an optimal policy in a model-free setting, making it applicable to a much larger set of problems in which an agent is to learn an optimal behavior.