An Analysis of Reinforcement Learning

In this analysis, we will be looking at reinforcement learning. We will look at two Markov decision processes (MDPs). One of which will have a relatively small number of stats and the other will have a relatively large number of states. We will examine the performance of three reinforcement-learning algorithms on each of these MDPs: Value iteration, Policy iteration, and Q-Learning. To analyze these performances, we will look at runtime, iterations, and similarity to the optimum policy.

MDPs

To explore the aforementioned algorithms, we must first understand the problems these algorithms will be saving. An MDP represents a convenient mathematical framework for modeling real world processes and decision-making. Many sequential decision making problems can be described as the following:

- **S**, the set of all possible states of the environment
- **A**, the set of all possible actions the decision maker can perform
- **T(s, a, s') = Pr(s'|s, a)**, a transition model that is the probability action *a*, leads to state s' given state s.
- **R(s)**, a reward function for the immediate value of being in a particular state
- **π(s)**, the policy of the decision maker, a mapping of states to the "best" action to take in the state

The utility of an MDP is dependent on the Markov property, or the Markov assumption, which states the conditional probability of future states is dependent only on the present state, not any previous states. In the case of our MDPs, we will be looking for π*, the optimal policy of maximum reward. We will assume our MDPs have stationary preferences.

Value Iteration

Value iteration is a greedy algorithm. If every state had an assigned utility to the overall problem, we could easily calculate the path of maximum utility to the goal state. However, for most MDPs, we don't know these overall utility values. To perform value iteration (VI) we perform the following steps:

- Assign an arbitrary utility to every state in **S**
- For each *s* in **S**, calculate a new utility for *s* from its neighbors
  - $U'(s)_{t+1} = R(s) + \gamma max_a \sum_{s'} T(s, a, s')U'(s')$
- Repeat until convergence

VI suffers from possibly being very slow to converge.

Policy Iteration

With a simple modification to VI, we can run policy iteration (PI). PI is run as outlined below:

- Generate an initial random policy, a permutation of actions for all states in the MDP.
- Loop until no action in the policy changes
  - Compute the utility for each *s* in **S** relative to the current policy
  - Update the utilities in each state
  - Select the new optional actions for each state, changing the current policy in the process

Q-Learning

Both VI and PI make the strong assumption that the decision maker has a substantial amount of knowledge about the underlying process in the MDP. This equates to how strongly we believe the transition model is accurate. In many situations, this belief is very weak.

VI and PI are actually more similar to path-finding algorithms that reinforcement learning, as they already know all of the information they operate on. Q-learning, on the other hand, learns during runtime.

Q-learning is a family of model-free reinforcement leaning algorithms that, given knowledge of **S** and **A**, attempts to learn the optimal policy from the available information. It does this by first assigning to each state a "q-value". When a state is visited, a reward is received for visiting that state, which is then used to update the q-value for each state as seen below. This may occur many times, since rewards may be stochastic.

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

This value represent the value for arriving in *s*, leaving via *a*, and then continuing optimally. Q-learning updates the q-values until some convergence. Q-learning needs a function to deal with the exploration-exploitation dilemma of reinforcement learning. In preliminary experimentation, we looked at epsilon greedy exploration, Boltzmann exploration, and random exploration. Best results were accomplished with ε-greedy exploration with very small ε, which we will use for the rest of this analysis.

Implementation

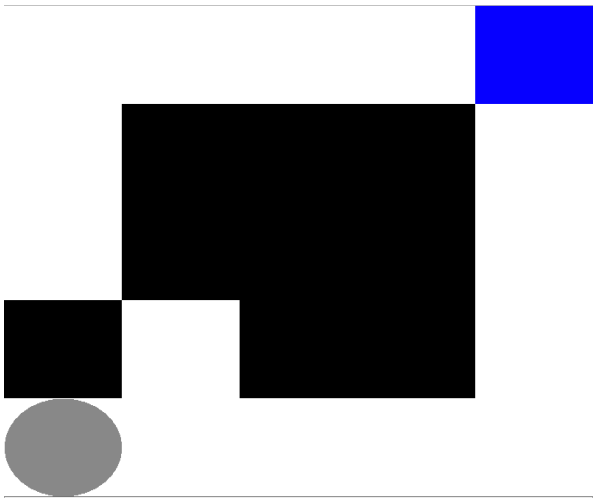
To run these algorithms on MDPs, we will use Juan J. San Emeterio's extension of the BURLAP Java Library.

GridWorlds

GridWorlds are two-dimensional spaces divided into squares of uniform size. Each of these squares can be one of several types: the agent (grey circle), walls (black squares), and the goal state (blue square). Upon taking an action, the agent has an 80% chance to move in the desired direction (North, South, East, West). The agent has a 6.67% chance of moving in any of the other directions. Every GridWorld also has a reward function specifying the rewards of performing various actions.

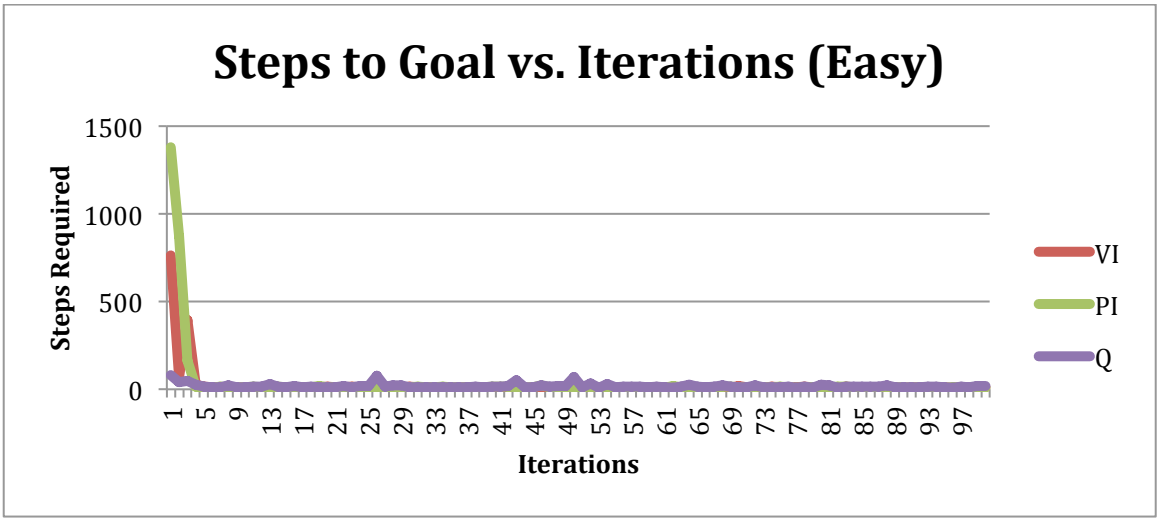
An example of GridWorlds usefulness in the real world is for robot planning algorithms. The decision space for a robot can be mapped to a GridWorld and algorithms can be created to operate on this space. The two GridWorlds we will use will be the easy and hard GridWorlds in Esmeririo's BURLAP extensions.

Easy GridWorld

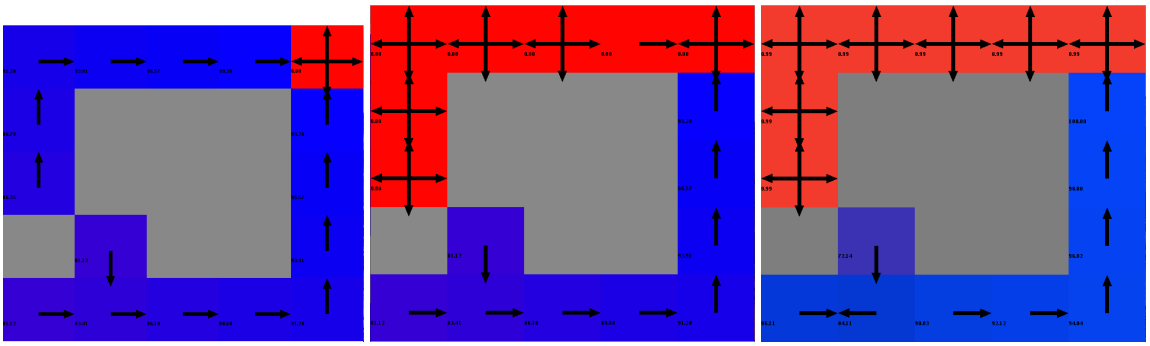


To begin our analysis of our algorithms on GridWorlds, we will use the above GridWorld as our relatively smaller MDP. A point of interest is the state directly east of the agent in the above image.

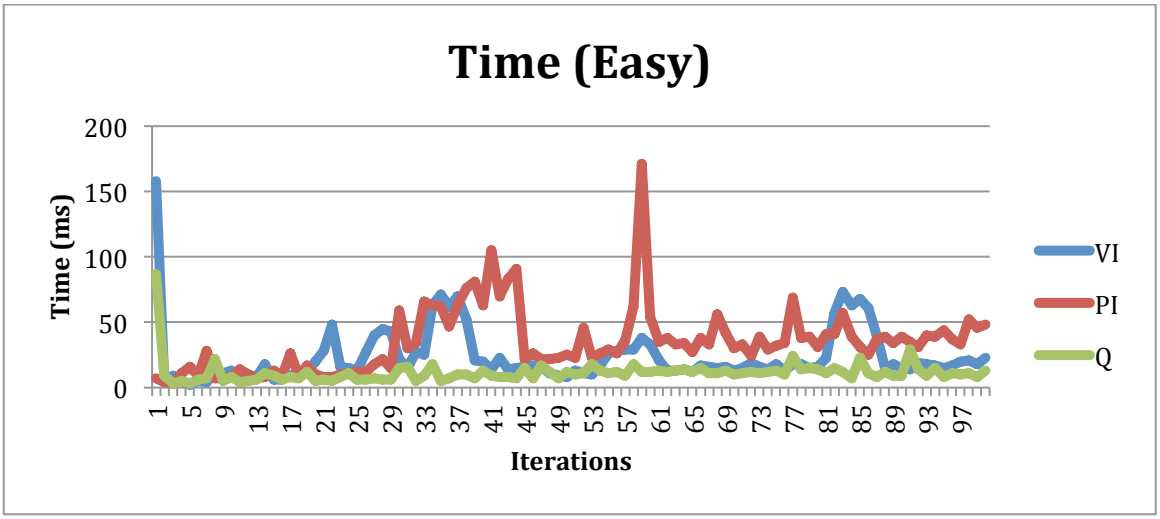
Now we will look at the performance of our three algorithms (VI, PI, and Q-learning) on this GridWorld. We will first look at the steps to goal vs. iterations for each algorithm.

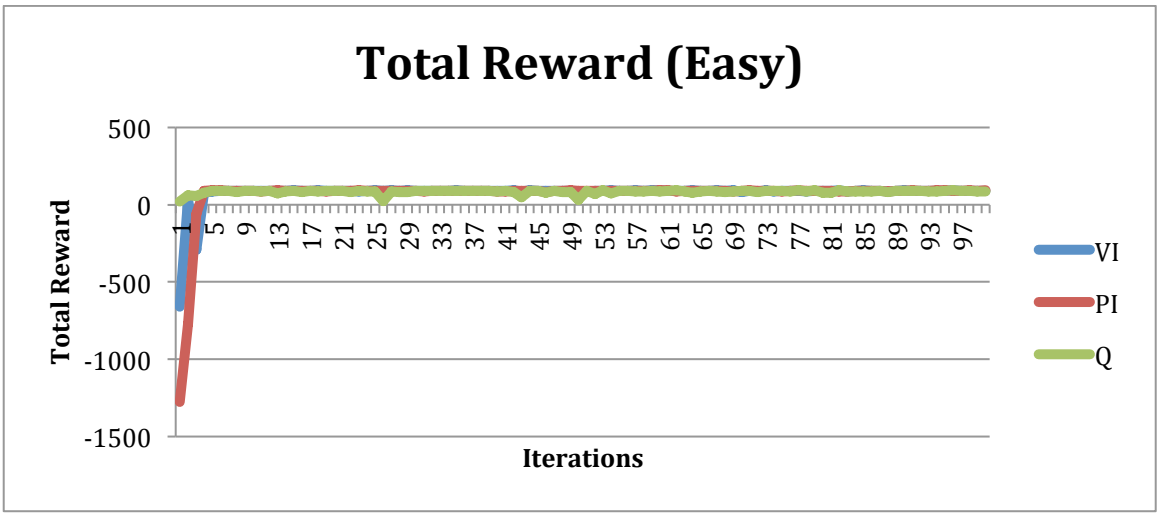


For this GridWorld, Q-learning performs very well. Although, VI and PI perform very well since they have perfect information of their environment. Below, we will look at the policies created by each algorithm.



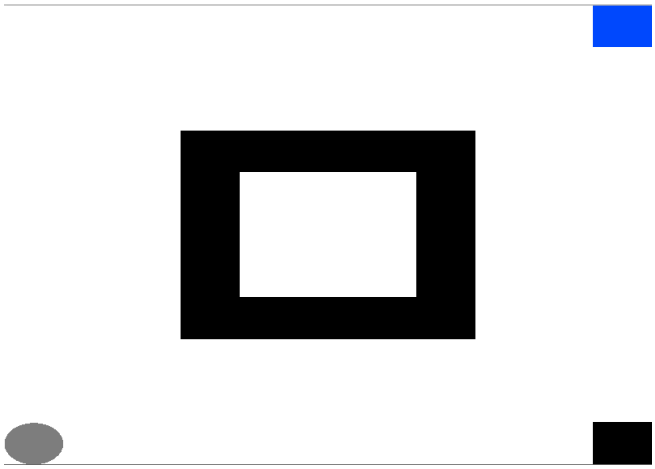
It is important to note that VI is able to create a policy for the separated northwest sector. In reference to the kidnapped robot problem mentioned earlier, VI would have a clear advantage. To further examine these algorithms, we will look at time and total reward with respect to iterations.





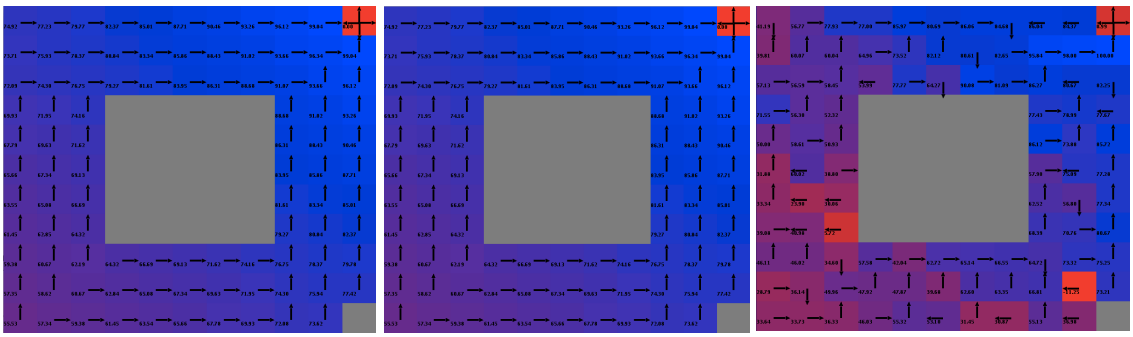
Here we can clearly see that Q-learning outperforms the others, with VI outperforming PI with respect to time. All three perform very similarly with respect to total reward. However, PI slightly outperforms VI which slightly outperforms Q-learning. We see an inverse relationship between speed and accuracy, as expected.

Hard GridWorld

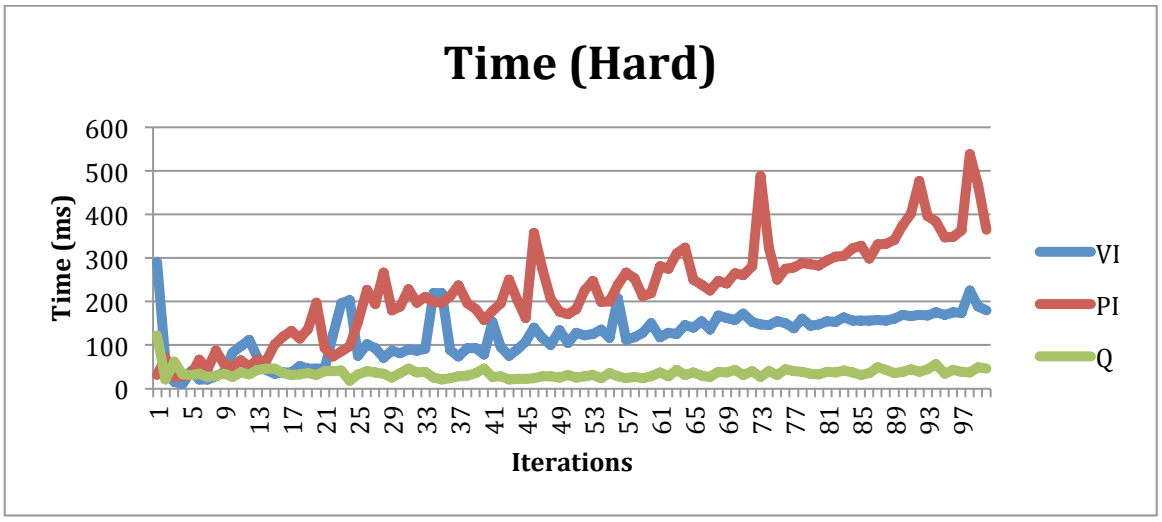


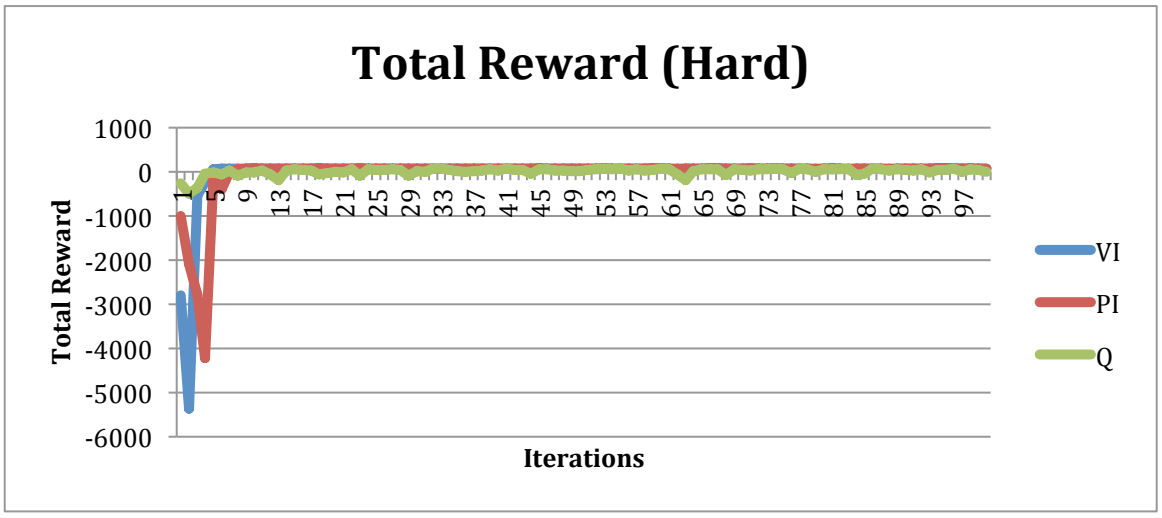
In order to further our understanding of these algorithms, we will look at them in the context of a larger GridWorld. This GridWorld will present the algorithms with many more choices, and we expect to see this accentuate the differences between the algorithms. To begin, we will look at steps to goal with respect to iterations.

At lower iterations, Q-learning clearly outperforms the other algorithms. At higher iterations, VI outperforms PI which outperforms Q-learning. The differences are much larger now, with VI requiring 33 steps, Pi requiring 100 steps, and Q-learning requiring 180 steps at 100 iterations. Here are the policies each algorithm created.



It seems that Q-Learning performs much worse with the larger GridWorld. This is because Q-Learning must learn, while VI and PI have perfect domain knowledge. Below, we will look at time and total reward with respect to time.





It can be concluded that Q-learning converged on reward sooner than and is faster than the other algorithms. Even though it seems to have found a suboptimum policy, the policy it found is better sooner.

Conclusion

In conclusion, we can see that VI and PI perform very well, but are subject to the constraint that they require very accurate and complete domain knowledge. Q learning fills this void, and acquires a policy that is better faster than PI and QI, despite their domain knowledge advantage. Furthermore, for the kidnapped robot problem, we found that VI has distinct advantages that allow it to create policies for normally unattainable states in the space of the MDP.