Keith Mertan
CS7641 - Machine Learning
Assignment 4

Markov Decision Processes and Reinforcement Learning

Introduction

Markov Decision Processes

This assignment explores Markov Decision Processes and three methods for finding their optimal policies. A Markov Decision Process is a framework for dealing with stochastic decision making processes. A key component of an MDP is that it satisfies the Markov property. This states that the process is "memoryless", or conditionally independent of all prior states. An MDP can be characterized by its available states, its possible actions at each state, the transition probability function which dictates how likely the chosen action is performed in the stochastic environment and the reward function for entering each state. In this analysis, grid worlds with embedded reward values at each cell are used to represent MDPs.

Two grid worlds, one 10 x 10 and one 20 x 20, are constructed for testing policy finding algorithms. Each grid has obstacles the agent must navigate with varying reward values found in each accessible state. Since there are walls represented numerically as 1 and visually as black grid cells, the number of accessible states is 68 for the small grid world and 263 for the large grid world. For each world, the goal state is in the upper right corner of the grid. The goal state has a reward of 100 and each step is -1 reward in both grid worlds. Each action has a probability of 0.8 to be executed correctly, with each of the other three directions having probability 0.2/3. Rewards embedded in grid cells are multiplied by 10.

Once these grids are created, the goal of this analysis is to solve for each MDP's optimal policy. An optimal policy is the set of actions that, at each state, have the highest expected value of future rewards. Value iteration, policy iteration and reinforcement learning are tested for solving for the MDP's optimal policy.

Algorithms

Value iteration looks at each state, randomly assigns it value, then recursively assigns reward from each discounted future state for each possible action. This continues until the change converges to a small positive number. Once all states have converged, a deterministic policy is constructed by taking the action of maximal utility at each state.

Policy iteration starts by generating a random policy along with a set of values for each state. Each state is then iterated over and its value is calculated recursively until convergence over just the action the policy dictates at that state. The policy is then updated by taking the action of maximal utility at each state like in value iteration. The value-policy update loop continues until the values converge.

Reinforcement learning is implemented as Q-Learning in this analysis. Q-Learning attempts to learn a value Q(s, a) for each <state, action> pair that will inform the creation of a deterministic policy. Q-Learning starts out at a disadvantage to the iterative algorithms explained above because unlike them, it does not have knowledge of the transition probabilities or reward function in the environment. Instead it uses a proxy to the utility function, Q, and attempts to learn it directly. This method is guaranteed to converge to the optimum policy assuming that every action from every state is observed an infinite number of times.

Three tuning parameters for Q Learning are explored in this analysis. Alpha, or the learning rate, controls how much information the learner incorporates from a new observation. Each time a Q value updates, it retains 1 - alpha of its original value plus alpha times the value found for the <state, action> pair in this iteration. Higher alphas favor new information while

lower alphas favor information learned in the past. Although a decaying alpha was tested, the strategy yielded worse results than those with constant alphas across the board. Only results from Q-Learners with a constant alpha are analyzed here. The values tested for alpha are 0.1 and 0.9.

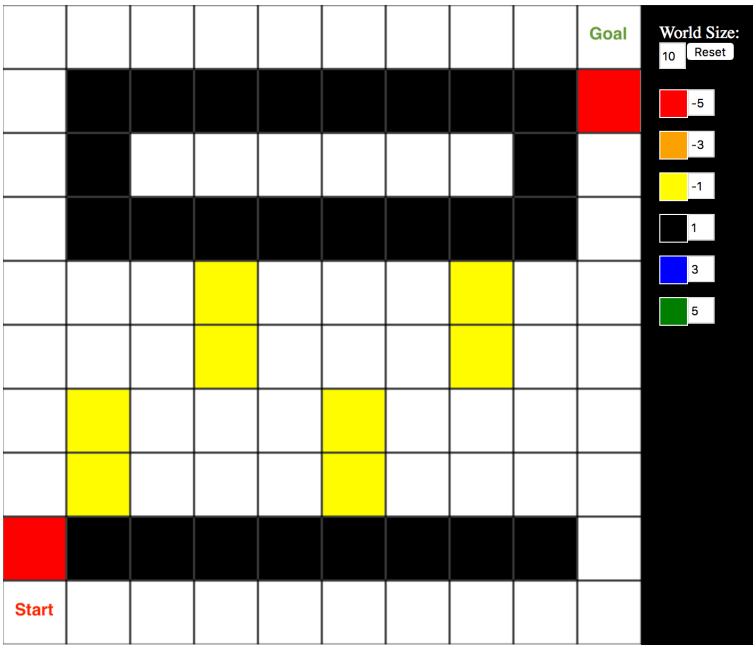
The second parameter tuned is the initial values of Q. The values tested are -100, 0 and 100. -100 begins by telling the learner every <state, action> pair is terrible and will yield bad results. Setting them all to 100 tells the learner that all movements from all states yield amazing results and setting them to 0 makes all actions neutral. If in truth, most <state, action> pairs are detrimental, initializing to -100 would make sense. If most moves are truly beneficial, initializing to 100 may yield better results.

Lastly, epsilon is used to account for the exploration vs. exploitation dilemma. As the agent moves around the world to learn the results of each <state, action> pair, it must decide how to move. Instead of taking the action that maximizes Q(s, a) over all possible actions, epsilon gives the a small probability for the agent to explore a seemingly suboptimal route. Since the values of Q are initialized arbitrarily, this helps the algorithm break out of optima fabricated by those initializations. When epsilon is greater than zero, this also ensures that we visit each <state, action> pair an infinite number of times if we iterate infinitely. Epsilon values tested are 0.1, 0.3 and 0.5 to represent low, medium and high rates of random actions, respectively, during learning.

The code used is an adaptation of Jonathan Tay's GitHub repository for assignment four. The repository utilizes the BURLAP library.

Value Iteration and Policy Iteration

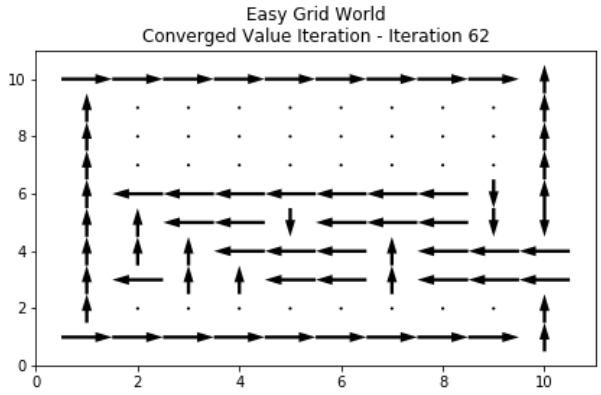
10 x 10 Grid World

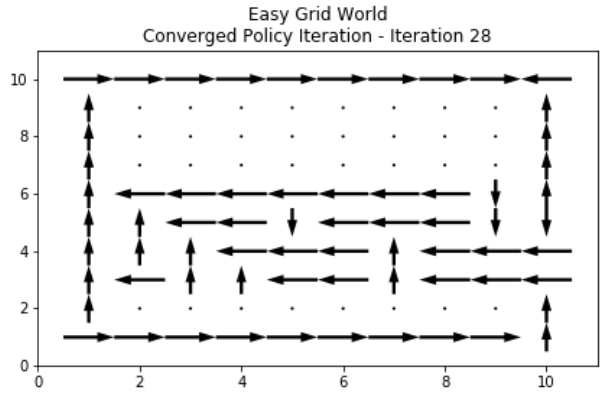


The grid world on the right is the MDP with a small number of states to be solved. This world is interesting because there are multiple routes to the goal state, but the best is obviously to take the long way and loop around the negative rewards. This gives the algorithms opportunities to fall into local optima (all the way up then all the way right or vice versa). Mitchell's book provides a good example to ground this in the real world: a robot looking for its battery docking station. As for the negative rewards, perhaps they are puddles… or fires… or portals to other MDPs of varying difficulty. Feel free to use your imagination.

Value iteration and policy iteration were both employed to solve this MDP for its optimal policy. The two converged policies can be seen below. The two policies only differ by their optimal actions at the goal state. Since the game ends at the goal state, the action is irrelevant and a random one is assigned making the policies identical.

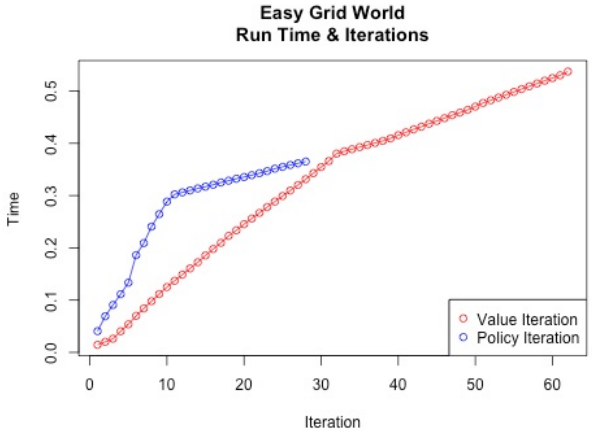
An interesting point to note is the decisions of each policy on row 6, directly below the 3 x 8 grid of inaccessible cells. Both policies choose to travel left in columns 2 through 8, even through the cells with -10 reward. This is contrasted by the choices made on row 3, above the

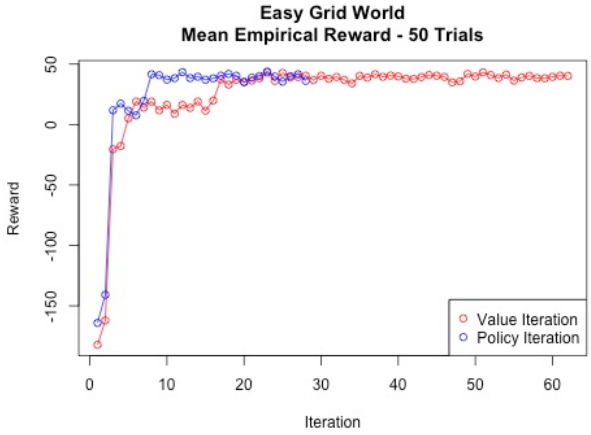




wall near the bottom, for columns 3, 4 and 8. These three cells move up to avoid the -10 reward cells. This means that the stochastic nature of movements actually makes traveling through the negative rewards in row 6 superior to avoiding them. Traveling down to avoid it would cause the agent to move farther from the goal, creating an additional cost of -1 for each step and potentially landing the agent in the -10 cell due to the transition probability. Since movements upward to avoid negative rewards in row 3 still move the agent closer to the goal, they are preferred over the option of moving through them.

Run time is critical in analyzing these two approaches in this case because they converge to identical policies for all practical purposes. On the left below are the number of iterations it takes each algorithm to converge and the clock time taken as well. On the right is the average empirical reward received when running 50 simulations of agents following the optimal policy at each iteration.



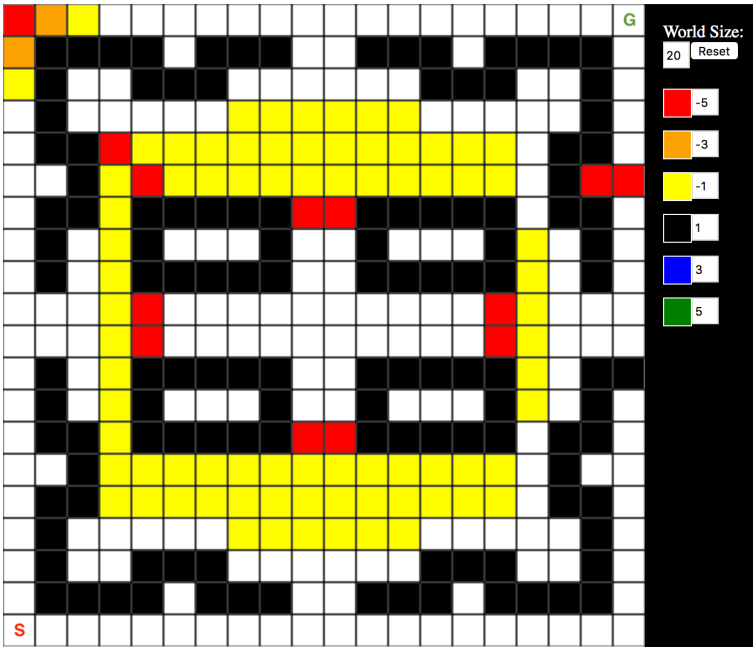


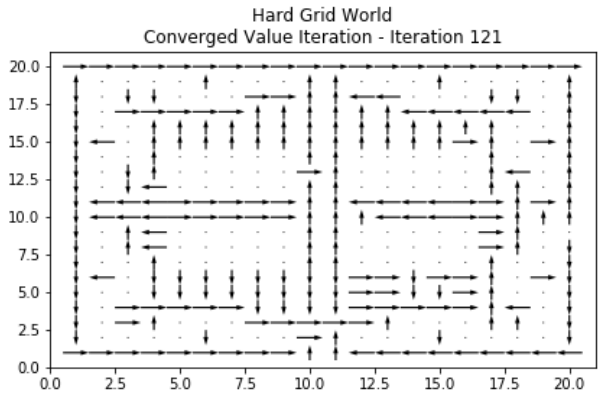
Policy iteration converges in far fewer iterations compared to value iteration. Because policy iteration runs something similar to value iteration at each time step, it makes sense that it requires fewer iterations to converge. In this case, policy iteration not only converges in fewer iterations but also converges in a shorter time. Value iteration takes almost 50% longer to converge. Since changes in values are being evaluated at each iteration and not changes in policy, it is possible that the policy in either algorithm converged before the values did and was unchanged for some number of iterations. This is an area for further analysis, however; given this implementation of the two algorithms, policy iteration is preferred for this small grid world. This point is further supported by the average empirical rewards seen on the right. Both

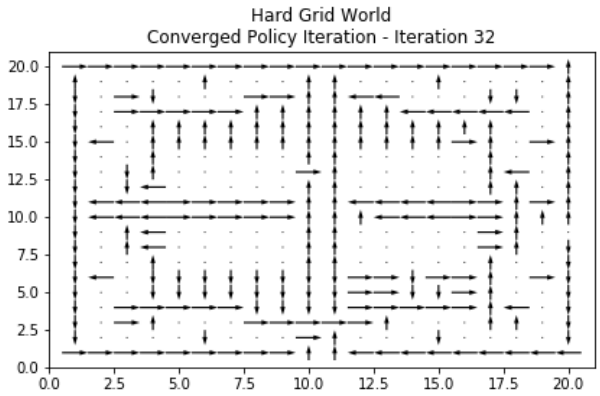
algorithms converge to a similar total reward near 50, making the run time the main differentiating factor between the two. As a final note on the run time, both methods see an elbow at some point during their iterations. Since each algorithm has an inner loop for each state which runs until the value update is small enough, these inflection points must be where nearly all state values were constant and required few to zero cycles through that inner loop.

20 x 20 Grid World

The second MDP on the right has a larger number of states (~4x as many) and far more obstacles than the previous grid world. There is a blocked path in the bottom right and suboptimal paths through the middle, along the left wall and along the right wall. The MDP was designed such that the best path is along the bottom, up through the bottom opening, around the inner right walls, out the top then right toward the goal. Again, value and policy iteration are used to solve the MDP and comparisons are made between their solutions, run time and average empirical reward. Below are the optimal policies produced by value and policy iteration.



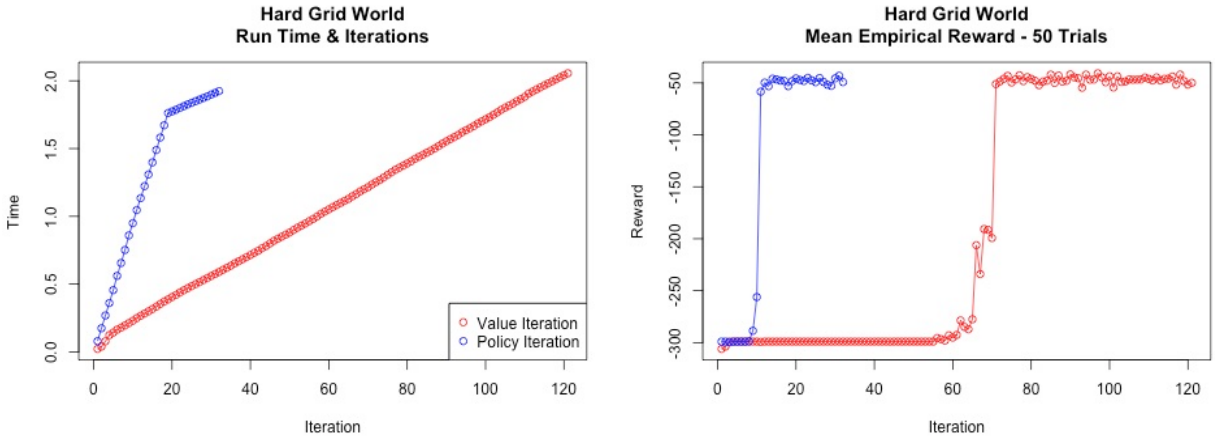




As is the case with the easy grid world, we see nearly the same policy from each algorithm. The actions to take at the goal state and row 18 column 2 are different for each algorithm. The goal state difference is explained by the same reasoning as the small MDP while the other cell simply has a tie in the utility it gains from a move downward or rightward. For all intents and purposes, the two algorithms converged to the same optimal policy.

Since again, as in the case with the small MDP, we see practically equivalent policies from both algorithms, run time becomes paramount when choosing between value and policy iteration. Like the small grid world, far fewer iterations are required for policy iteration to converge than value iteration. This is expected because of the different iteration complexities of

the two. The clock time required for value iteration to converge is only ~7% greater than that of policy iteration in this case. This is a significant improvement from the ~50% convergence time difference seen on the small grid world. Again, both solvers may continue running at some point while the underlying policy has already converged. Later in the analysis convergence time as a function of grid world size (with no obstacles) is examined to see if the decreasing trend in the ratio of value iteration convergence time to policy iteration convergence time continues as the grid world grows.



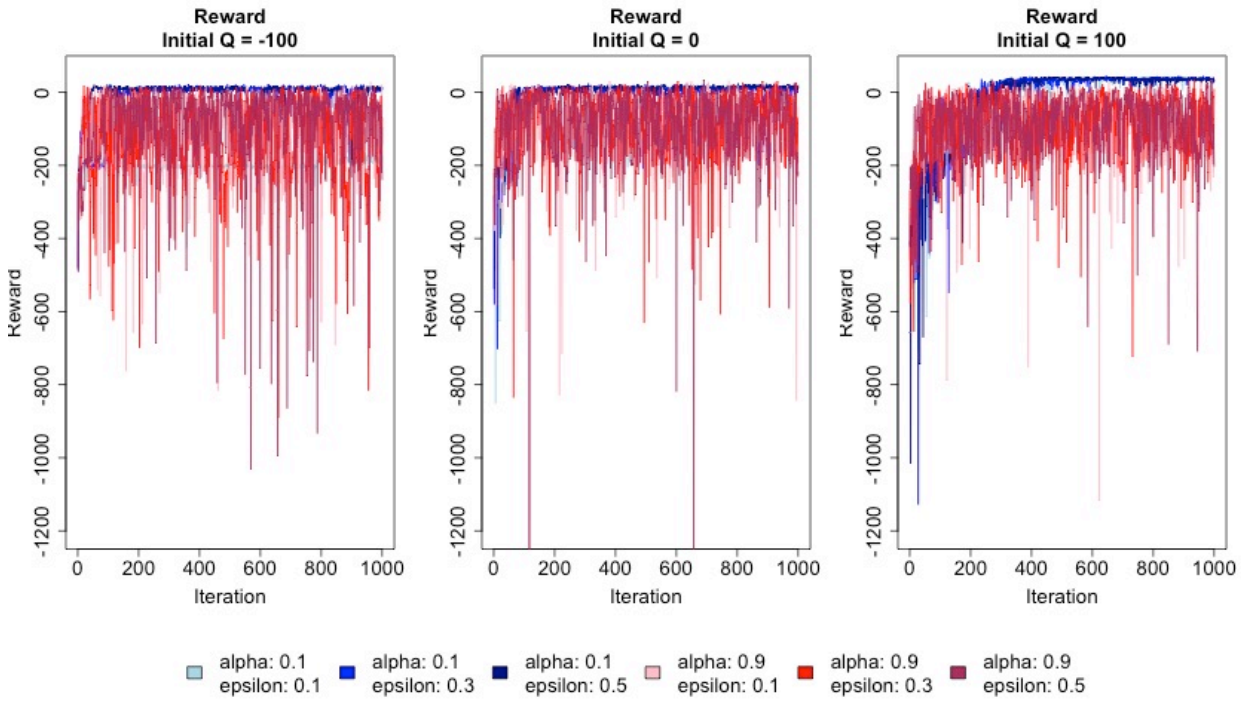
The mean empirical reward chart shows that both algorithms stabilize to a reward around -50 many iterations before convergence. Value iteration finds a local optimum between iterations 60 and 80 but quickly breaks out of it. Policy iteration also finds a local optimum where the reward is approximately -250, but discovers the best route on the next iteration. Again, policy iterations provide more drastic updates than value iterations so these two phenomena make intuitive sense. The reward plot shows that both algorithms explored the space and possible optima but are robust enough to discover the roundabout path that represents the global optimum.

Reinforcement Learning

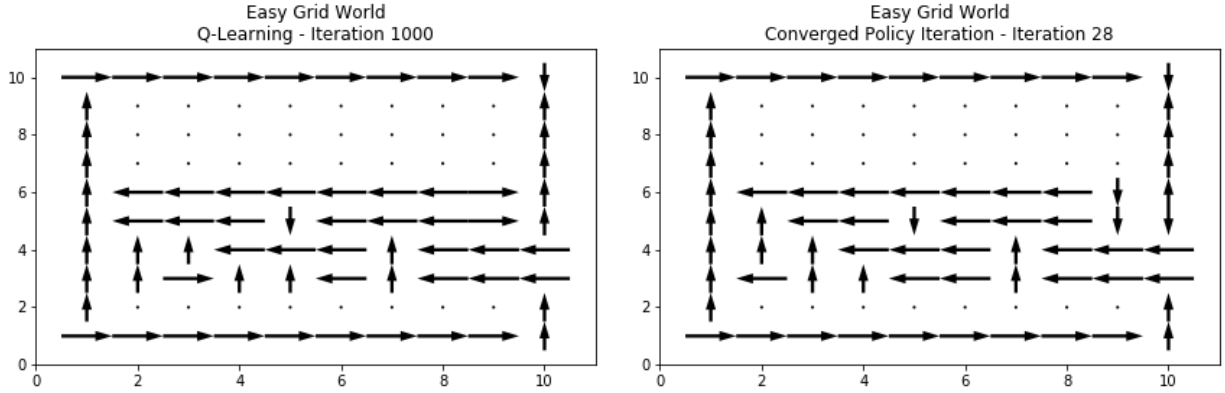
10 x 10 Grid World

To begin solving the small MDP with reinforcement learning, optimal parameters for the Q-Learning algorithm must first be selected. Each plot below represents the mean empirical reward of 50 trials given an initial value for Q. The lines within each plot represent combinations of alpha and epsilon. It is clear that the high learning rate of alpha = 0.9, shown in hues of red, varies far more than when alpha = 0.1. This makes sense since 90% of each update comes from the most recent observation of Q(s, a) and there is no alpha decay. Alpha influences the reward most significantly, averaging over 100 reward points higher for the conservative learning rate across all other parameter settings. Q's initial value also increases reward significantly when set to 100 over 0 or -100. The rightmost chart below reflects this and shows a cluster of alpha = 0.1 and Q = 100 learners breaking off from the others. Within this cluster epsilon does not significantly alter results. With few states to explore, it makes sense that the exploration parameter, epsilon, isn't massively influential here. Ultimately epsilon = 0.5 produced the best result, achieving an average reward of 41.18 in its final iteration.

The convergence criteria for the above algorithms was an average of maximum Q value updates each 10 training episodes less than 10^{-6}. None of the algorithms converged given this criteria, but lowering the threshold to even 10^{-5} resulted in far worse performance on all



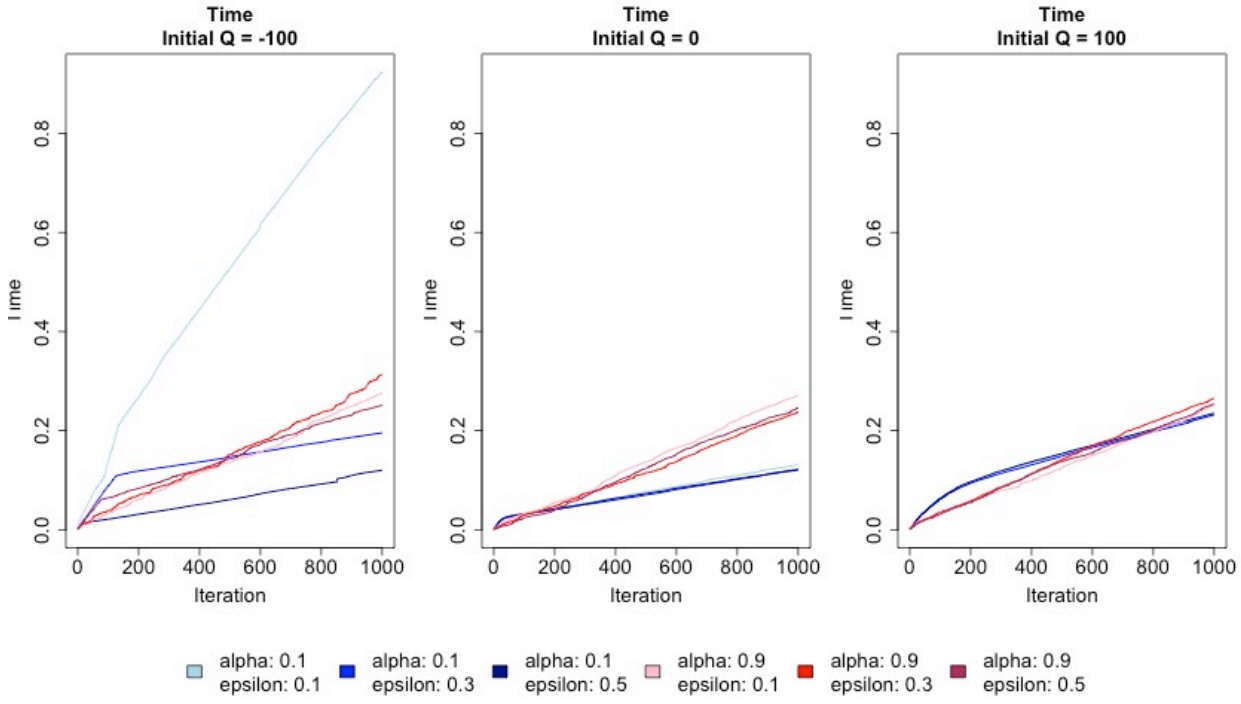
parameter combinations. Although they did not converge, the mean reward for the final iteration is used to compare performance. The maximum reward over all iterations could have been used and the associated policy extracted, but given the stochastic nature of the environment the highest reward doesn't necessarily represent the best policy. Choosing the policy from the final iteration provides more stability since Q values converge as the number of iterations approaches infinity.



Above is the policy provided by the optimal set of parameters: Initial Q = 100, alpha = 0.1 and epsilon = 0.5. The converged policy iteration solution is provided again for comparison. The optimal route from the start cell is the same for each. The differences come in the off-route solutions. An example is row 3 column 3. The Q-Learning solution suggests moving right while the policy iteration solution moves the agent upward. It is difficult to determine which move actually provides the maximum utility, but it should be noted that the 41.18 reward recorded by Q-Learning is slightly greater than the 38.08 and 39.48 that policy and value iteration recorded, respectively. For minor differences in policy like this, an interesting follow on would be to

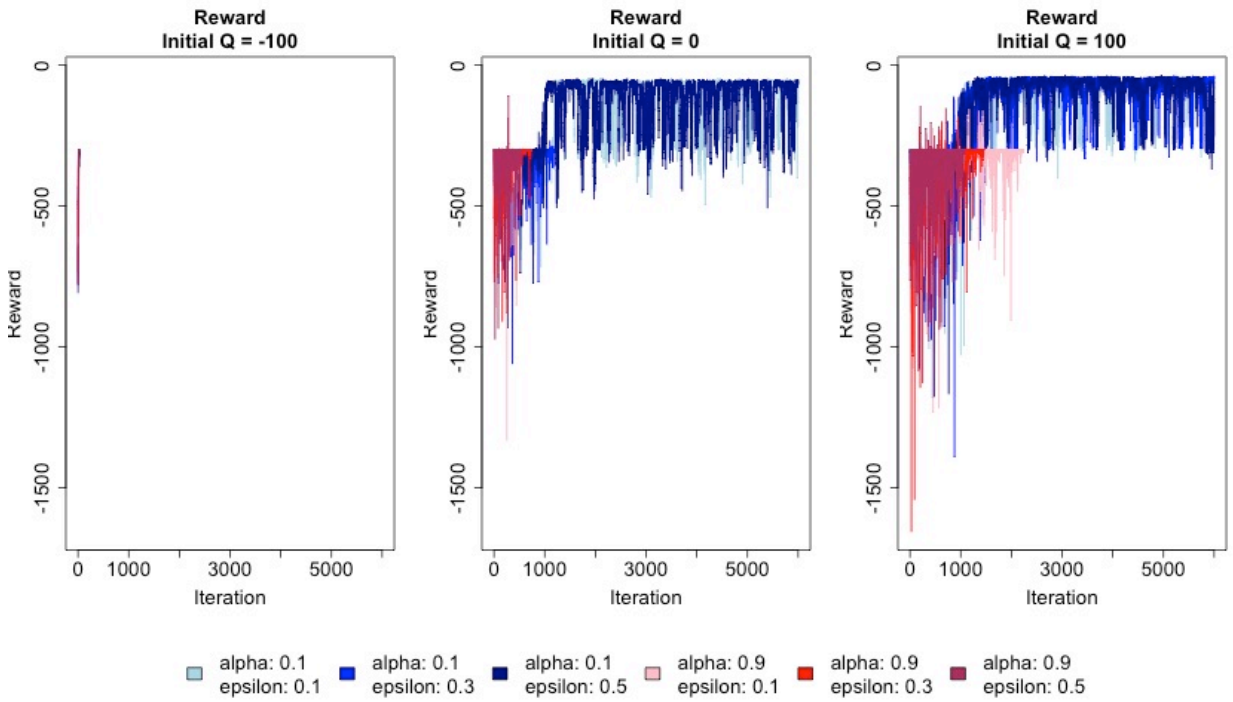
inspect the distribution of empirical rewards (instead of just the mean) and run t-tests to see if one is significantly better than the other, statistically.

Finally, the run time as a function of iterations is shown for each combination of Q-Learning parameters below. At first glance, Q = -100, alpha = 0.1 and epsilon = 0.1 jumps out. This combination clearly has the longest total run time and run time per iteration. Given the results already discussed, this confirms that true Q values are closer to 100 than -100. With a small learning rate, small exploration rate and far from true Q values it takes the learner extra time to converge to a reasonable solution. In all other cases, alpha = 0.9 takes longer in total than alpha = 0.1. The parameters that gave the best reward finished 8th of 18 parameter combinations in terms of run time. Compared to the policy iteration solution, the optimal parameter Q-Learning solution ran faster, in just 65% of the time. Given the comparable rewards and faster training time, Q-Learning seems to be the preferred solver for the small MDP.

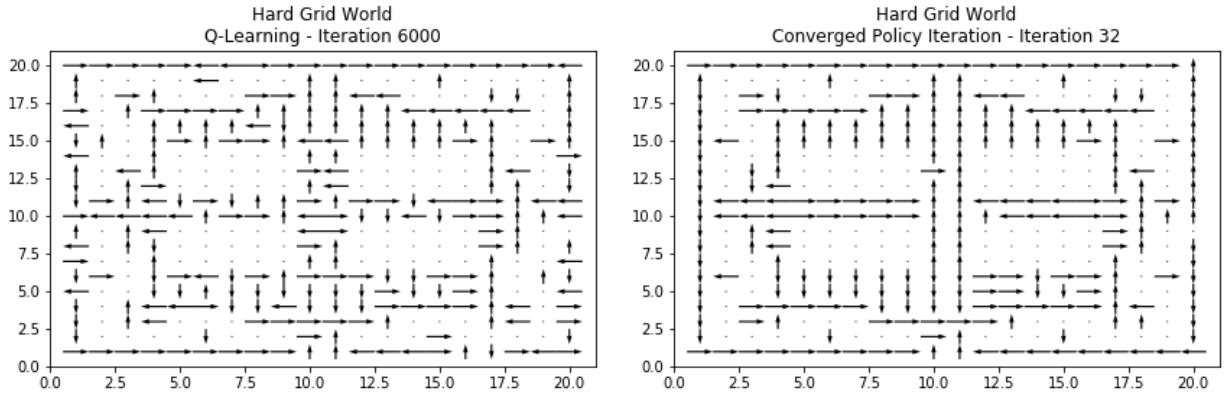


20 x 20 Grid World

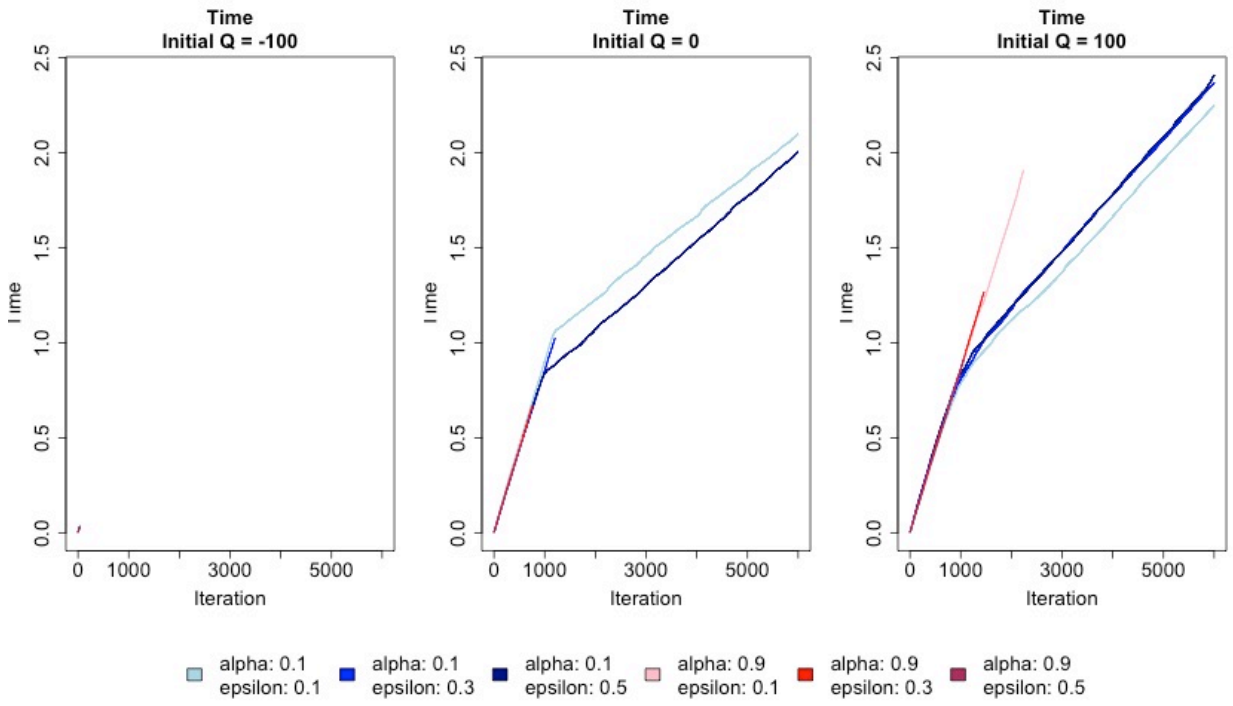
Below are the mean rewards received from each Q-Learning parameter combination when learning on the large MDP. The immediate observation is that for Q = -100, there is a quick convergence to a score near -300. This score happens when the agent stays near the beginning of the map and avoids obstacles, accepting the small negative reward of -1 at each step in preference to exploring the map and running into larger penalties. Since all moves look “painful” to the agent at first due to the large negative Q value, this decision makes sense but ultimately provides a suboptimal policy. For Q = 0 and Q = 100, much better performance is achieved. As with the 10 x 10 grid world, alpha = 0.1 greatly outperforms alpha = 0.9, this time by over 150 reward points on average. Epsilon = 0.1 performs better on average than the other two values. There is a need to explore the large map with many different possible routes to the goal, but epsilon values greater than 0.1 seem to not exploit enough of the information gained already.



Below is the optimal policy discovered by Q-Learning: Initial Q = 100, alpha = 0.1, epsilon = 0.1. Though some of the algorithms did converge for the large MDP, the best performance came a parameter combination that did not converge before the iteration ceiling. Comparing to the policy iteration solution, the optimal path to the goal is the same. Again, difference appear in off-route cells that are rarely visited by the Q-Learner or by any stochastic agent following either policy. Column 1, for example, sees many Q-Learned actions that conflict with those in policy iteration. Since they're so far off the beaten path, the empirical reward from both algorithms is similar. Q-Learning scores -43.82 while policy iteration scores -42.75 on average.

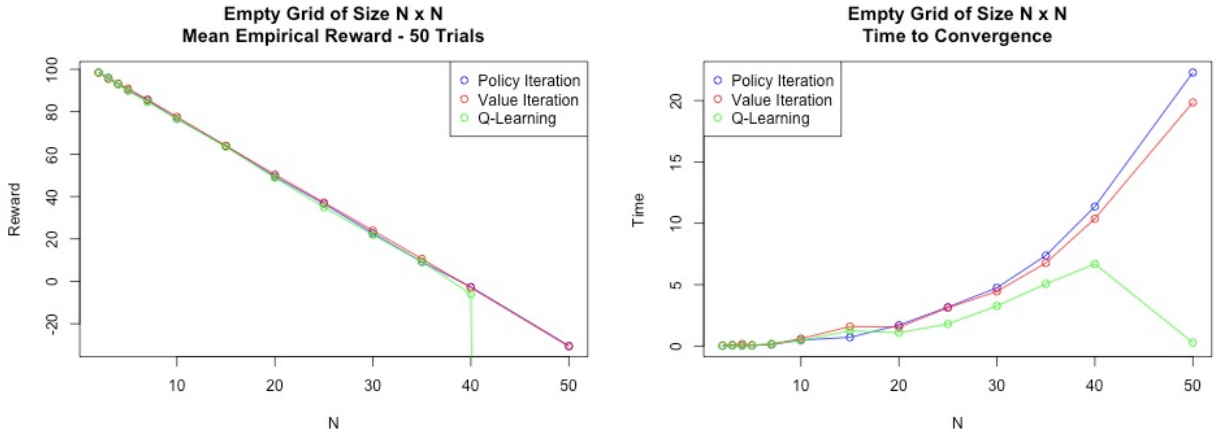


Fast convergence is seen for combinations containing Q = -100 or alpha = 0.9. These are suboptimal solutions so their time benefits are irrelevant. Of the Q = 100 combinations, the optimal solution is the fastest to run. Compared to policy iteration, Q-Learning takes ~10% longer to run in this case. For the large grid world, policy iteration seems to be the best choice as an MDP solver.



More Fun!

Thanks to Jonathan Tay, this analysis is able to look at the performance of value iteration, policy iteration and Q-Learning as a function of solely MDP size. Below are the mean empirical rewards received on empty grid worlds that have no walls or rewards other than the goal. The values of N are the lengths of the sides of the square grid worlds. A grid world with N = 10 has 100 states, for example. For Q-Learning both sets of optimal parameters found in the small and large MDPs were tested. Initial Q = 100, alpha = 0.1 and epsilon = 0.5 far outperformed initial Q = 100, alpha = 0.1 and epsilon = 0.1 so results below reflect performance of the prior. It seems that with so many routes to the goal, a more curious exploration strategy is preferred to a more conservative one.



Looking first at the reward plot on the left, it is impressive that Q-Learning stays competitive with the value and policy iteration scores up until the 50 x 50 grid world. It has no information about transition probabilities or the reward function yet is still able to perform nearly as well, and in considerably less time, than the other algorithms. But as grid world size grows, issues of infrequently visited <state, action> pairs start hurting the empirical agent's score as we see in the 50 x 50 grid (reward ~-4000). Reinforcement learning is useful for quick optimal routes with a small number of states but degrades once the number of states is too large and may not be useful if having optimal actions globally is desired.

Value and policy iteration have nearly identical rewards to one another across all of the tested grid world sizes. As was noted earlier in the analysis, value iteration seems to take longer than policy iteration for smaller grid worlds. After 20 x 20, value iteration actually converges more quickly than policy iteration while still retaining an competitive reward score. It seems that for larger grid worlds, value iteration becomes the preferred MDP solver.

Conclusion

In conclusion, each of the three solvers analyzed has its time to shine. Policy iteration is useful for a robust policy on small grid worlds. Value iteration becomes quicker than policy iteration for larger grid worlds while still optimizing the reward received. Finally, reinforcement learning is a quick way to find a path to a goal when a global policy isn't as important. Moreover, reinforcement learning is a remarkable substitute for either of the other two approaches when the model of the environment is unknown. For beginning at such a disadvantage, it yields impressive results nonetheless.