CS 7641: Markov Decision Processes

Miles Rufat-Latre

Part 1: Selecting Two MDPs

Context

While some Markov Decision Processes (MDPs) would demonstrate tradeoffs in Q-Learning hyperparameters, the goal in this assignment is to identify the practical differences between Value Iteration (VI), Policy Iteration (PI) and Q-Learning (QL) overall. For that purpose, I've devised one large and one small, modified version of the Frozen Lake gridworld.

MDP Selection Criteria

The stochasticity and early termination states of my selected gridworlds should make Q-learning harder to scale up to larger worlds. It is hard to explore the world if you keep slipping into bottomless pits, or wandering into hard-to escape nooks and crannies. These MDPs should also cause larger differences in utility estimate between PI and VI, based on the intuition that a long and narrow optimal path, to escape from a given nook or cranny, will grant PI more freedom in how it ramps up the utility between adjacent states on that path, and therefore more freedom in the utility estimate of the deepest, darkest nook of that cranny.

MDP Details

Entering an ice state confers a reward of -0.1. $water$ states are terminal and are worth $-k$ points, where k is manually increased with the size of the world. There is one $goal$ state, which is terminal and confers $+k$ reward. In my case, the goal state is always in the top right corner of the map, and the starting state is an ice state in the bottom left. I've also added $wall$ states into which the agent can't move. If the transition would lead the agent into a wall state, it stays in place instead.

Agent actions succeed with probability 0.9. Otherwise, a movement direction is chosen uniformly from the alternatives. Each state's type (with the exception of the start and goal states) are chosen from the distribution $P(ice) = 0.8$, $P(water) = 0.1$, $P(wall) = 0.1$. State type and action-transition probabilities remain constant across all world sizes. The random seed is manually chosen to select a world with water near the start *and* finish of the gridworld, and a viable path from the start to the finish.

The small world has width and height $n = 7$, and goal reward $k = 5$. The large world has $n = 40$ and $k = 30$. I would have tried a larger world, but BURLAP's dynamic programming approach has memory complexity $O(n^3)$, and I think there were other memory size issues. Unfortunately my hardware reached its memory limit close to $n = 40$.

Part 2: MDP Planning Comparison

VI and PI Implementation and Hyperparameters

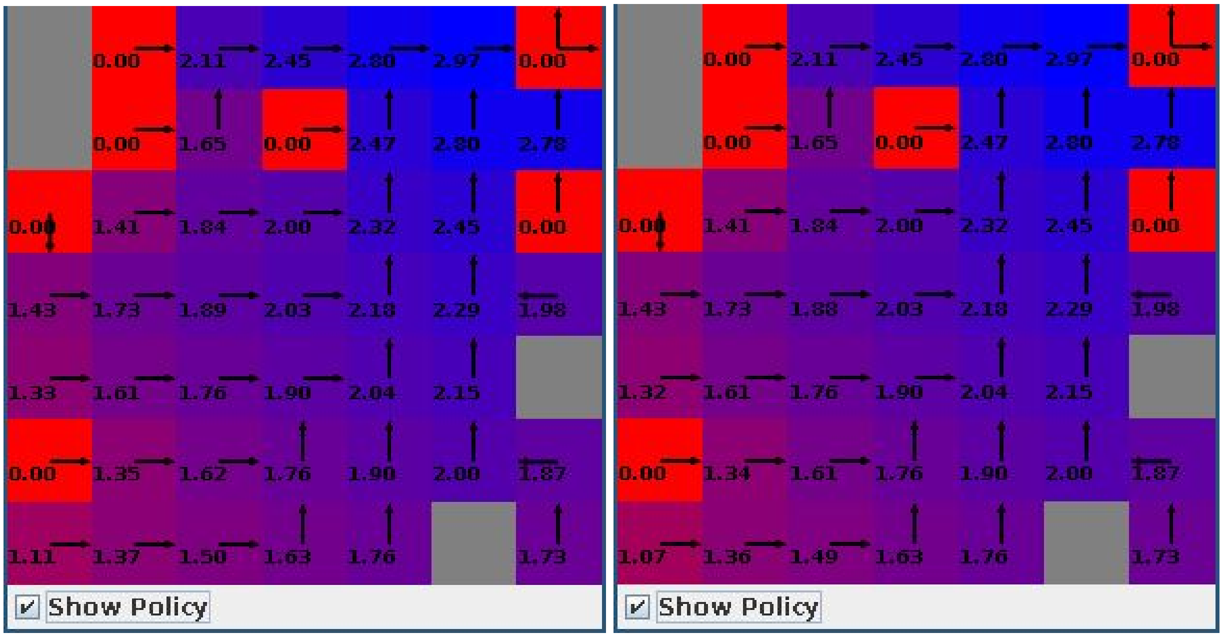
Any VI implementation has a discount γ and convergence criterion *c*. I used $\gamma = 0.999$ for both VI and PI, to ensure that the reward propagates far enough for convergence even on the large world, which will be good for comparison against Q-Learning. The convergence criterion is whether, during the iteration, the maximum change in value over all states exceeded a certain threshold. Reducing this threshold to 0 caused VI to run for more than 100,000 iterations even on the small world, without any policy change at all, as it attempted to exhaust double precision in its value estimates. I kept it at $c = 0.001$ which results in estimates with less than 0.01 error from that optimum on the small world.

The policy iteration implementation is as simple as possible, with no special tricks in the value estimation stage. In that stage it runs some number of rounds of value iteration (with the same hyperparameters mentioned above), starting from its existing value estimates. I chose to do only 1 iteration of VI in this step, to allow maximal divergence between the two algorithms and by the same stroke allow PI to converge quickly. The convergence criterion was based on the largest change in value for any state during a given round of policy evaluation. I chose a threshold of $0.05n$, to allow the threshold to grow with the world automatically. This should keep PI converging to a near-optimal policy without requiring total value convergence.

Planning Results and Analysis

On the small world, value iteration and policy iteration converge to very similar value estimates (within 0.01), with the exception of the start state which differed by 0.04. That can be explained by the same “nooks and crannies” reasoning mentioned before: PI can change the slope of the utility ramp into a cranny state without affecting the resulting policy, so it won’t bother to refine the deep-cranny utility estimates. The start state in my seeding of the small MDP is tucked behind a water state, making it fairly cranny-like, and it is also the furthest from the goal state, giving PI more flexibility in what values it can assign without changing the policy.

On the small world, despite the slightly differing utility estimates, the policies are identical. This illustrates how policy iteration didn't need to reach the exact value estimates to come up with the optimal policy. PI ultimately executes 11 iterations vs the 12 of VI. PI is most likely stopping early due to its larger convergence threshold $0.05n = 0.35$ on the small world. Execution time for VI on the small world is volatile, ranging from about 0.05s to 0.09s, even though iteration count is consistent. PI remains stable around 0.025s. I’m not clear why these algorithms differ in execution time by so much, although it could be because of more debug printing or memory allocation in VI which would result in chaotic I/O waits. In any case the implementations are not optimized so this experiment will only be able to reveal time complexity differences on the larger world.



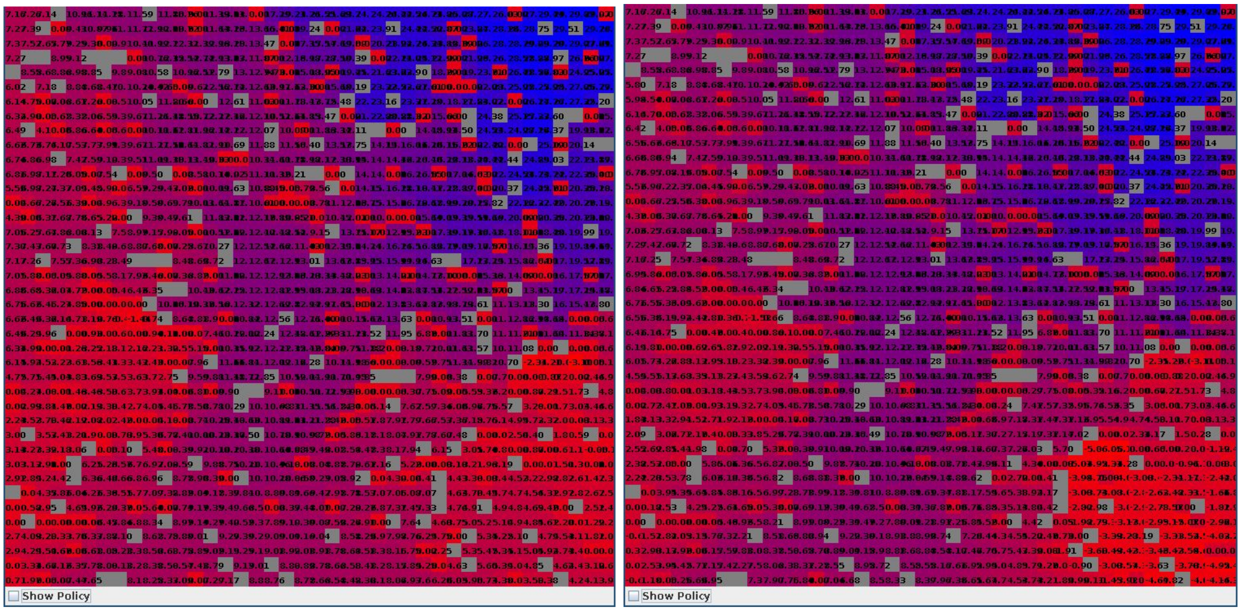
	0.00	2.11	2.45	2.80	2.97	0.00
	0.00	1.65	0.00	2.47	2.80	2.78
0.00	1.41	1.84	2.00	2.32	2.45	0.00
1.43	1.73	1.89	2.03	2.18	2.29	1.98
1.33	1.61	1.76	1.90	2.04	2.15	
0.00	1.35	1.62	1.76	1.90	2.00	1.87
1.11	1.37	1.50	1.63	1.76		1.73

☑ Show Policy

	0.00	2.11	2.45	2.80	2.97	0.00
	0.00	1.65	0.00	2.47	2.80	2.78
0.00	1.41	1.84	2.00	2.32	2.45	0.00
1.43	1.73	1.88	2.03	2.18	2.29	1.98
1.32	1.61	1.76	1.90	2.04	2.15	
0.00	1.34	1.61	1.76	1.90	2.00	1.87
1.07	1.36	1.49	1.63	1.76		1.73

☑ Show Policy

Figure 1. Small world value estimates and policy. Arrows show policy. Color and numbers show value estimates. VI on the left, PI on the right. Terminal states show a value of 0, and walls are shown in grey.



☐ Show Policy

☐ Show Policy

Figure 2. Large world value estimates. Color shows value estimates. Policy could not be shown at this scale. VI on the left, PI on the right. Terminal states show a value of 0 (brightest red), and walls are shown in grey. Slightly duller reds in PI are due to more extreme negative estimates in the bottom right corner, which shifted the bottom end of the color scale slightly.

Planner	World Size	Iterations	Time (s)	Utility Est.
VI	7	12	0.0576	1.11
PI	7	11	0.0272	1.07
VI	40	81	0.996	0.74
PI	40	44	0.566	-0.63

Table 1. Shows comparison of all planning results on small and large worlds. Shows iterations and time in seconds until convergence. Utility estimate is the planner's value estimate for the starting state.

In the large world, there were noticeable differences in value estimate from VI vs PI. In particular, states which were far away from the optimal path diverged most in value between the two algorithms. This is especially pronounced in the bottom right corner where we can see that some value estimates differ by as much as 8.5, almost ⅓ of the final goal reward. But if we want to see the most telling evidence for my analysis, we can observe the path most directly from the start to the goal: On this path, the policies for VI and PI are only slightly different at the beginning, and become identical once we are about ¼ of the way through the path. The values are only slightly different in the first half of the path, and are identical closer to the goal.

I believe this shows that not only is PI terminating earlier, and not quite propagating value estimates back to the start state, it is also neglecting to update its policy for states that it has learned to avoid due to previous iterations (the top left and bottom right). The result is that the evaluation phase has a different policy to work with in these areas, and comes to very different utility estimates than pure VI. In particular it is not surprising that the value estimates are so much *lower* in these areas for PI: An unoptimized policy is generally less useful, and since we never update the policy in these states, we are really running a kind of policy-parameterized value iteration that (perhaps accurately) estimates the value of the suboptimal policy.

This explains why VI consistently needed 81 iterations and about a second of processing vs PI which needed 44 iterations and 0.5 to 0.6 seconds to converge. PI is simply doing stopping optimization when it wouldn't change the policy, and removing states from consideration when it determines that their impact on the utility of the optimal policy is small. This effect only shows up when we increase the world size. The most obvious reason for this is that the small world is so small that convergence to the optimal policy happens before PI can start to avoid suboptimal states, which it must do to begin to diverge from VI on non-optimal-path states.

The fundamental advantage of PI over VI seems to be that it can stop optimizing when it wouldn't improve the policy, and it can neglect to optimize its policy in a given state if it determines that the state should never be reached under the optimal policy.

Part 3: Reinforcement Learning

Q-Learning, Implementation and Hyperparameters

I chose Q-Learning since it is among the simplest and most popular of reinforcement learning algorithms, and will demonstrate the deep differences between reinforcement learners and MDP planners. Q-Learning has an initialization i, learning rate α, discount γ and "curiosity" ε (curiosity is the likelihood of taking an action randomly rather than by following the learned policy). My implementation is provided by BURLAP, and is tabular with one row/column per state.

To choose hyperparameters, I produced training profiles similar to figures 3 and 4, which show cumulative reward averaged over multiple training runs. I used these plots to compare up to 5 models at a time. I used these to evaluate tradeoffs between learning quickly or optimally, and incurring high or low cost before converging.

I found two local optima. I call them BasiQ and OptimistiQ. BasiQ has $i = 0,\ \alpha = 0.5,\ \varepsilon = 0.05$ and OptimistiQ has $i = ln(n),\ \alpha = 0.2,\ \varepsilon = 0$. BasiQ relies on random exploration to encounter new state/action pairs, but if we take that too far it will never learn enough about the states near the goal, because it will keep falling into water. Therefore it must learn quickly when it sees a new state. This is a catch-22 because the high learning rate can allow stochastic actions to confuse the learner. OptimistiQ does on-policy exploration by assuming that each action it hasn't taken will be fairly rewarding, until it sees it a few times in practice. It will not converge until it has explored enough to feel that the goal is the most valuable thing worth pursuing. Increasing i with the log of world size is intended to avoid causing excesses in exploration as the number of states increases, and seems to be important for the large world. I suppose $\sqrt{n}$ could be better.

Reinforcement Learning Results and Analysis

To test the Q-Learners, I run 1000 episodes strictly on-policy ($\alpha = 0,\ \varepsilon = 0$), and measure the mean reward per episode. I also examine some of the test episodes by replaying them. I can't show the examined episodes in this paper, but I will describe what I see. I estimate convergence iterations by looking for where the slope of the average reward curve becomes constant. I use this to estimate convergence *time* based on the total execution time.

On the small world, both learners converge to almost the same policy. The difference in reward slope in figure 3 is due to BasiQ still having its $\varepsilon = 0.1$ throughout training. This is supported by the scoring results which show that BasiQ achieved slightly *better* reward on-policy. BasiQ looks to have converged first around 75 episodes, while OptimistiQ took until the 100th episode. This is because OptimistiQ had to work through its "optimism backlog" before it would switch from exploration to exploitation.

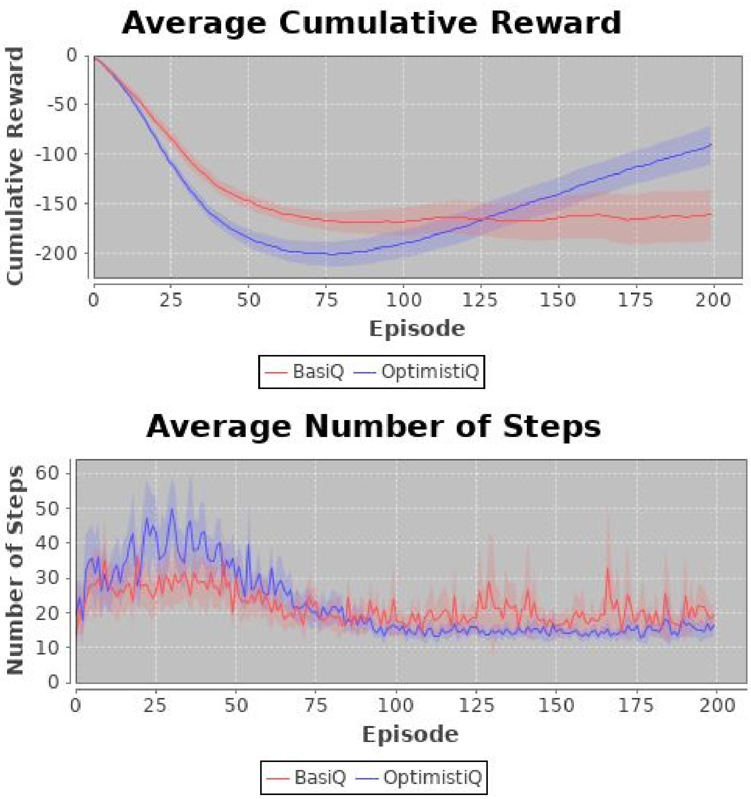


Figure 3. Small world Q-Learning. Cumulative reward and steps per episode, averaged over 50 runs.

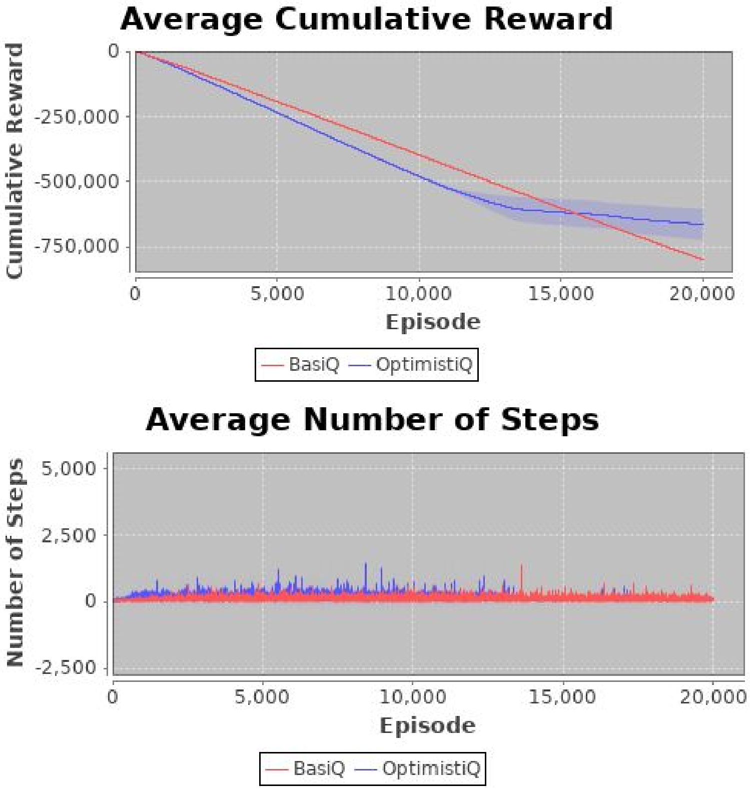


Figure 4. Large world Q-Learning. Cumulative reward and steps per episode, averaged over 5 runs.

Algorithm	World Size	Time (s)	Iterations	Mean Reward
Value Iteration	7	0.0576	12	1.343
Policy Iteration	7	0.0272	11	1.306
BasiQ	7	0.00298	~75	1.076
OptimistiQ	7	0.00324	~100	1.101
Value Iteration	40	0.996	81	1.17
Policy Iteration	40	0.566	44	1.37
BasiQ	40	N/A	N/A	-545.73
OptimistiQ	40	1.943	~14,000	-2.605

Table 2: Shows comparison of all algorithms (learning and planning), on small and large worlds. Time and iterations to convergence, and mean on-policy reward per episode. For Q learners, time and iterations are estimated based on apparent convergence of slope in the cumulative reward curve.

The policies produced by the Q-Learners on the small world are slightly lower in utility than the planning algorithms' policies. Based on observing the episodes directly, I believe this is because the Q-Learners were slightly more likely to fall into certain water states due to a policy difference, not because of wasted time "walking in circles". Otherwise the difference would be smaller, since the only other cost is time spent moving, and the policy I observed on the optimal path was straightforward for all planners and learners. I was surprised by the Q-Learning convergence times on the small world. Convergence occurred in about 3000-4000 iterations total, but these were much shorter iterations than in VI or PI. It is possible that JVM JIT optimizations also played a role, by inlining large chunks of the code, since the inner loop of the QL algorithm has only one iteration per action type, vs VI or PI where the inner loop iterates over all states.

On the large world, BasiQ fails to converge at all. This is because its method of exploration is extremely inefficient, and causes it to visit each state-action pair far too rarely. When BasiQ moves it into a state it hasn't fully explored, the agent is faced with inaccurate estimates that it doesn't "know" to be inaccurate, and it fails to explore the unknown actions. It instead settles into a suboptimal policy when visiting that state. Even when it does choose an unexplored action, it can be easily thrown off by the stochasticity of the world dynamics. So if the number of visits to a given state is already small, the agent is unlikely to converge properly in that state. Even worse, the probability of reaching the goal state in an episode decays *at least exponentially* as the world gets larger, because the water states have uniform density and *even on the optimal policy*, as we pass by any one of these we have a chance 1/30 of falling into it. Random exploration compounds this issue, and means we will almost never reach the goal

state, so the correct utilities will take an at-least-exponentially increasing amount of time to propagate to the start state.
OptimistiQ, on the other hand, converges to nearly the same policy as VI and PI, incurring only a small value difference from PI/VI. I think the reasons for this discrepancy are similar to when we saw it on the small world: OptimistiQ has developed a different policy that gives it a slightly higher risk of falling into certain water states, which reduces its average reward per episode slightly. This convergence was extremely expensive, and the training time grew much more rapidly than for VI or PI. In fact, I would predict that as the frozen lake world grows beyond 100 states, even OptimistiQ will fail to converge, because it will fall victim to the uniform distribution of water states, and exponentially decreasing probability of reaching a given state even under the optimal policy. This would overpower the influence of the quadratic growth in number of states, since it represents an exponential decay in reachability of those n^2 states.

One risk with OptimistiQ is that once it converges, there is no exploration at all. This means that the small utility difference between its policy and VI/PI will never be resolved, no matter how much additional training we put it through. One way to deal with this could be to add a very small $\varepsilon = 0.01$, which would allow it to experiment with very small policy changes.

4. Possible Future Work

This experiment was helpful in uncovering the differences between the naive implementations of PI, VI and Q-Learning. However, there are a number of subjects I would be interested in experimenting with further.

Optimizing the implementations for memory footprint would allow us to experiment with larger world sizes, and to test the hypothesis that OptimistiQ will eventually fail to converge. This would mean reimplementing the environment, agents and experiment in a language with smaller and more controllable memory consumption such as C or Rust. This would also likely improve the consistency of timing operations for small world sizes, as these languages have no runtime JIT. Once we can experiment with larger worlds, we could make a high-resolution plot of optimal policy utility (using VI) vs world size to see what the curve looks like, and to determine whether and when the exponentially-decaying reachability issue kicks in. That experiment could set a baseline from which to measure how *additional* exploration problems grow in different reinforcement learning algorithms.

As an improvement on my measurement techniques, I would like to (given the time) solve for the utility of the Q-Learning policies rather than statistically estimating them, by using a modified version of VI that assumes the agent will always follow the Q-Learner’s policy. Another improvement would be to generate new random worlds for each training run rather than manually selecting a seed. This would eliminate biases in my experiment that might result from the particular seeds I chose. We would need to automatically filter out worlds that are invalid (i.e. path from start to goal being fully obstructed).

Finally, there are improvements for the algorithms under test that I would like to try, to mitigate some of the issues I saw, and to evaluate their use in industry, where these optimizations would probably be required. Firstly, I would like to try some of the “tricks” Dr. Isbell mentioned in the class lectures for reducing the cost of Policy Iteration’s evaluation phase. Second, I would like to try explicitly KWIK (Knows-What-It-Knows) Q-Learning models, and to compare them against the implicitly-KWIK OptimistiQ which simulates that through initialization. For example, using a learning rate that is independent for each state-action pair, and decays only when the action is taken. And lastly, I’d like to try to mitigate the severe Q-Learning exploration issues I observed, by changing the percepts to the Q-Learning agents. I might do this by using a Q-function that takes the x and y distance from the goal, and a 5x5 snapshot of the states around the agent. Such a Q-function could no longer be a simple table, but I predict that the agent would be able to effectively explore arbitrarily large FrozenLake environments, at least to the extent allowed by the exponentially-decaying reachability issue.

5. Conclusions

I chose the FrozenLake environment expecting to observe moderate increases in time consumption by the algorithms I was testing, as the size of the world increased. I discovered that FrozenLake actually demonstrates a universal problem of “reachability decay”, and observed how each algorithm responded.

VI and PI reached identical policies on the small world. The difference in policy utility between VI and PI was small enough to be negligible on both the small and large worlds. VI training time *and* iterations grow faster than PI on the large world, as PI is able to save time by omitting improvement for states that are not relevant under the optimal policy and by stopping early when further iterations will not improve the policy. The number of irrelevant states increases for larger worlds, which is why this is a difference in growth characteristics. VI and PI policies and utility estimates diverge from one another as the world grows. This divergence is greatest in the “irrelevant” states that PI does not bother to optimize for. There is also slight divergence for states that are far from the goal state. Divergence in value is more widespread than divergence in policy.

When QL did converge, its policy’s utility was slightly lower than that of PI or VI. This is due to slight differences in policy that make it more likely to fall into the water due to the environment’s stochastic transition mechanism. QL’s visibility into connections between state utilities is limited by its exploration mechanism, so this is to be expected. Q-Learning’s convergence time and iterations grew much faster with world size than VI or PI. An optimistic Q-function initialization and greedy exploration policy were found to be necessary for any convergence on the large world. This requirement on the exploration mechanism helped me identify the “exponential reachability decay” issue, which affects *all* policies on the FrozenLake environment, and severely impacts exploration for Q learners.