Andrew Cudworth
CS XXXX Machine Learning (Spring 2020)
Assignment 4 Markov Decision Processes

CS XXXX Assignment 4 Markov Decision Processes

Intro/MDPs:

Three reinforcement learning algorithms were explored via two Markov decision processes (MDPs). One MDP is a "frozen lake" grid world environment with stochastic results whose size is varied from small to large. The other MDP is based on a small tower of Hanoi problem. Both MDPs use openAI gym [5] where an environment returns rewards, a state, and terminal indicator for allowed actions. Value Iteration (VI), Policy Iteration (PI) and Q Learning (QL) were applied to each MDP and explored.

The first MDP is the ai-gym environment "frozen-lake" (FL) [SOURCE] which is a grid world that has start "S" and goal a "G" state that returns positive reward of 1. Every other grid is either "F" a frozen cell that can be moved to or "H" a hole that ends the episode and provides -1 reward. The state space is an N x N grid initially set to N = 8 giving 64 possible states and increased up to N = 100 for analysis. The action space has a size 4 corresponding to up, down, left, and right. The environment executes the submitted action with a probability of 66%.

This problem is interesting because it requires probabilistic reasoning to choose a best move which means an average performance of a policy must be understood vs a single result. Additionally, the state space can be increased while keeping the action space constant.

```
SFFF       (S: starting point, safe)
FHFH       (F: frozen surface, safe)
FFFH       (H: hole, fall to your doom)
HFFG       (G: goal, where the frisbee is located)
```

Figure 1 Frozen Lake (FL) N=4 Example

The second MPD is an ai-gym implementation of the classic game "Tower or Hanoi" (TOH) where three poles have disks of decreasing size stacked on the left-most pole. The goal is to move all the disks to the right pole in the same order. Only the top disk of a pole can be moved, and a disk cannot be placed on disk smaller than itself. In this environment, invalid moves are ignored, and the goal state results in a reward of 1 [8]. The number of disks can be varied from 3 to N. The state space is represented by a vector of the disks with the pole number the disk is on. The legal state space is 3^N.[6] The action is represented by a tuple where the first index is the pole to move the disk from and the second index is the pole to move the disk to. There are 7 possible actions, but generally fewer than 4 legal actions. Actions are deterministic.

This problem is interesting because it is a well analyzed computer science problem with a known recursive solution. The minimum length of a valid solution is 2^N.[6] which can be used as a baseline to compare learned performance against. Additionally, the problem has a large state space, but the fraction of legal states visited decreases with the size of the problem. This suggested an opportunity for exploration on small state spaces and exploitation on large. Implementations for large N will have high memory requirements, but the nature of the solution does not fundamentally change.

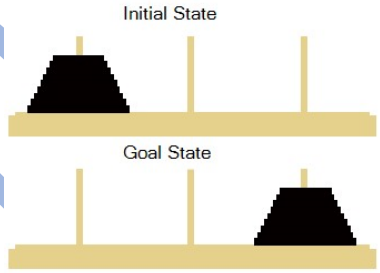

Figure 2 Tower of Hanoi (TOH) Example [6]

Table 1 TOH State Space Growth [6]

Number of Disks n	Length of Solution Path 2^n	Number of Legal States 3^n	Fraction of Visited States $(2/3)^n$
1	2	3	0.6666667
2	4	9	0.4444446
3	8	27	0.2962963
4	16	81	0.1975309
5	32	243	0.1316872
6	64	729	0.0877915
7	128	2187	0.0585277
8	256	6561	0.0390184
32	4294967296	1853020188851841	2.32×10^{-6}

Value Iteration:

Value iteration was implemented as outlined in Algorithm 1 [1]. The convergence criteria $\Delta < \theta$ was replaced with two criteria to be made practical. Criteria 1 was a large number of iterations intended to be more than needed but forcing termination within a reasonable time for the test case. Criteria 2 was no change in the Value function V(s)

generated for 10 iterations in a row. For the large state spaces, this was increased to 100. The majority of problems explored terminated due to criteria 2.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Algorithm 1 Value Iteration [Sutton]

Convergence Implemented:

-Maximum Iterations of Loop (generally set to 1000)

OR

-No V(s) Change for X iterations (generally set to 10)

The discount factor γ was varied on the three MDPs with average reward and iterations of 100 VI episodes presented in Figures 3-5. Note that 1 is the maximum reward for each MDP and each MDP only provides reward at the goal state. The discount factor controls how much future reward increases the value of a state. No discount (γ = 1) would require knowing the reward for the goal state for the given MDPs else all states would have Value of 0. This can be seen in the increased iterations as γ increases for the FL MDPs. This is not seen in TOH because the small deterministic state space allows an accurate value function with few iterations. In Contrast, the FL MDP requires more exploration and accounting for stochastic actions. So a trade-off between iterations and performance is needed. In the case of the large state space, no performance can be achieved unless the convergence criteria are relaxed to allow many more iterations. This result is not presented due to run time constraints. γ was set to 0.9 for future VI experiments.

Table 2 Value Iteration Discount Factor Impact

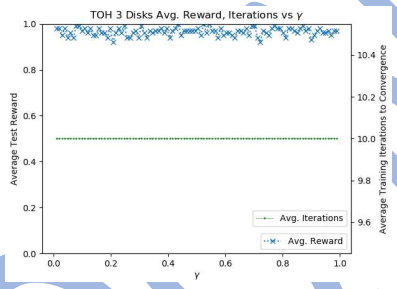

Figure 3 TOH Average Iterations and Average Reward vs Discount Factor

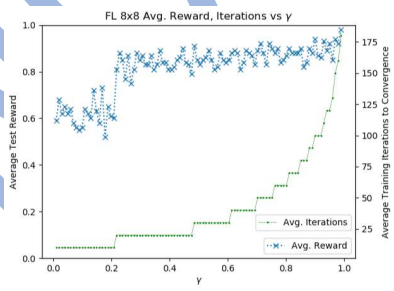

Figure 4 FL 8x8 Average Iterations and Average Reward vs Discount Factor

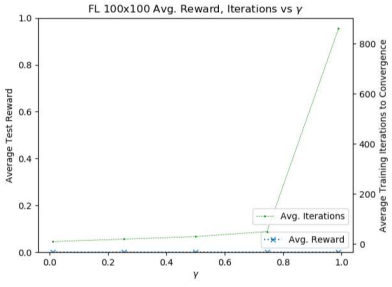

Figure 5 FL 100x100 Average Iterations and Average Reward vs Discount Factor

Policy Iteration:

Policy iteration was implemented as outlined in Algorithm 2 [1] with some changes. Only one step of value iteration was performed at each policy evaluation step rather than a VI loop to convergence. Convergence was similar to VI implementation with a maximum iteration criterion of 1000. However, the second convergence criteria were changed to policy π(s) not changing as opposed to no change in the value function. The choice to run 1 step of V(s) updates in the policy evaluation step allows more direct comparison between VI and PI. As outlined in Sutton, PI would have fewer iterations, but each PI iteration would have a longer run time and a variable number of Value function updates making comparison difficult. As implemented, VI and PI have similar iteration run times with very different convergence criteria. VI requires the Value function to converge vs PI only requires the value of states relative to each other to converge. For

the MDPs explored, this results in slightly faster convergence in terms of iteration and run time (see results section for further discussion). However, PI has more dependence on future discount γ.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Algorithm 2 Policy Iteration [Sutton]

Convergence Implemented:

-Maximum Iterations of Loop (generally set to 1000)

OR

-No Policy Change π(s) for X iterations (generally set to 10)

As in VI discount factor γ was varied and average rewards + iterations of 100 episodes are presented in Figures 6-8. Similar to VI, Performance is easily achieved for the small deterministic TOH environment and no performance is achieved for the large stochastic FL100 state space. Discount Factor γ serves a similar function as VI, but t interacts very differently with the new convergence criteria. Because a policy can converge before a Value function when V(s) is changing PI requires fewer iterations to achieve good rewards even when less future reward is considered(low γ). In other words, the Value function only needs to be accurate about which state is has the highest value relative to each other. Additionally, stochastic results force the update of a state's Value, but will only force a policy change if that value changes relative to other stochastic outcomes. This will not occur if the nature of stochastic operations (P(action = 66%) does not change over time. All of this leads to good performance on FL8 and TOH. For future PI experiments, γ is set to 0.9.

Table 3 Policy Iteration Discount Factor Impact

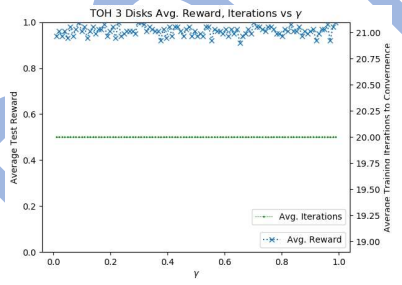

Figure 6 TOH Average Iterations and Average Reward vs Discount Factor

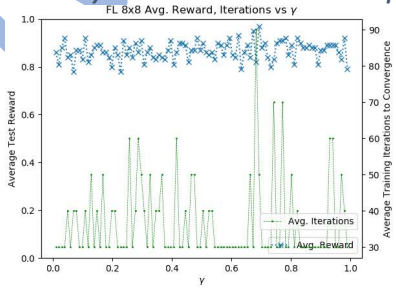

Figure 7 FL 8x8 Average Iterations and Average Reward vs Discount Factor

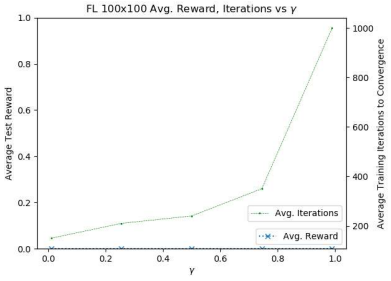

Figure 8 FL 100x100 Average Iterations and Average Reward vs Discount Factor

Q Learning:

Q Learning (QL) was implemented as outlined in Algorithm 3 [1] using a tabular representation of the state and action space. A check that an episode does not run more than 5000 steps was added to force reasonable run times. A few important distinctions in QL vs PI and VI are worth noting. First, QL does not require visiting each state as in VI and PI. Second, the Q table is much larger than the state table for VI or PI. It requires (number of actions)*(number of states)

vs (number of states) in VI and PI meaning it is less memory efficient. Finally, in a given episode, only the sates and actions visited are updated. This requires a careful explore vs exploit tradeoff.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Algorithm 3 Q Learning [Sutton]

Convergence Implemented:

-episode ends

OR

-5000 steps within an episode

The discount factor γ, learning rate α and ε-greedy parameter were explored in QL. Training episodes were conducted via the Algorithm 3 for a variable number of episodes. The Q Table was then frozen and run for 100 test episodes. The average rewards in test episodes are plotted as 'X' and the average iterations in test episodes are plotted as 'O' in the following figures.

In QL, γ plays a similar role as VI and PI. However, not all sates are guaranteed to be visited. As discussed, reward will only propagate if the goal is found. This requires a critical number of training episodes to ensure the goal is reached on average. Below that threshold, no reward is found regardless of discount factor. After the threshold, γ has a high impact on the reward in the stochastic case because non visited sates will have a Q value of near 0. So it is important to value visited paths more highly. A high γ considers the whole path rather than current or very next (s,a). This can be seen in significant performance improvement as γ increases on FL8 in Figure 10.

Table 4 Q-Learning Reward and Iterations to Convg. vs Training Epiosdes at Discount Factor (γ) = [0.05, 0.25, 0.5, 0.75, 0.9]

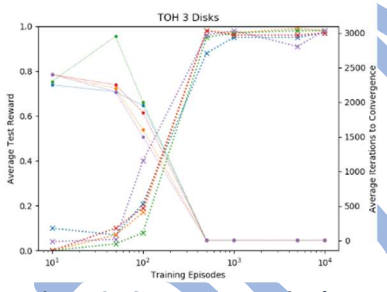

Figure 9 TOH Avg. Test Episode Reward and Steps vs Total Training Episodes

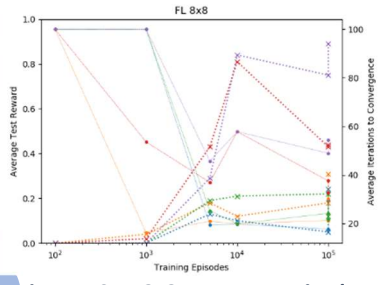

Figure 10 FL 8x8 Avg. Test Episode Reward and Steps vs Total Training Episodes

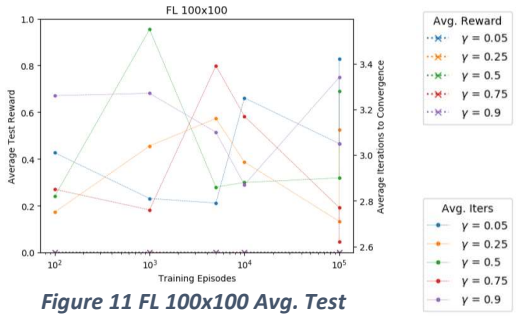

Figure 11 FL 100x100 Avg. Test Episode Reward and Steps vs Total Training Episodes

In QL, α is the learning rate. It modulates how much each step in an episode influences the current Q table. This is especially important in the stochastic case when an unlikely scenario should not replace the Q(s,a) learned over many consistent iterations. This is why a high-performance variation is seen on FL8 in figure 13 with performance and iterations to convergence generally increasing as α decreases. α Is generally set low to favor Q values learned over time vs new information. In this regime, more exploration via training episodes can be utilized to make up to the slow updates to Q values. Figure 12 shows a successful implementation of this with a slight improvement in performance as α decreases. For this reason QL α is set to 0.01 in all other QL experiments.

Andrew Cudworth
CS XXXX Machine Learning (Spring 2020)
Assignment 4 Markov Decision Processes

Table 5 Q-Learning Reward and Iterations to Convg. vs Training Episodes at Learning Rate (α) = [0.01, 0.05, 0.1, 0.15, 0.2]

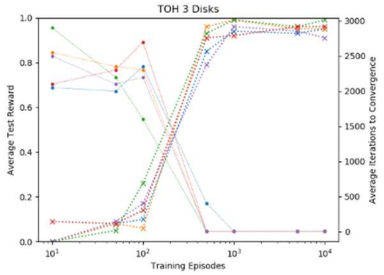

Figure 12 TOH Avg. Test Episode Reward and Steps vs Total Training Episodes

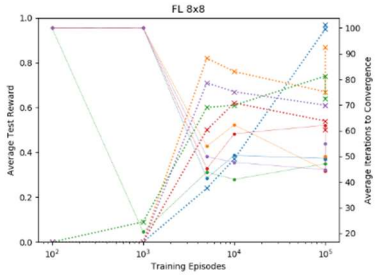

Figure 13 FL 8x8 Avg. Test Episode Reward and Steps vs Total Training Episodes

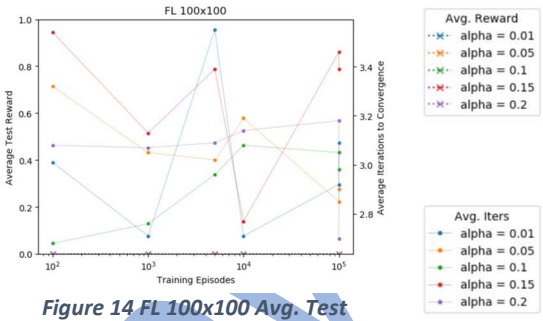

Figure 14 FL 100x100 Avg. Test Episode Reward and Steps vs Total Training Episodes

The step "Choose A from S using policy derived from Q(ε-greedy)" has important implications to how QL will explore the state space. An ε-greedy policy will choose the (s,a) with highest Q value 1-ε % of the time ($\varepsilon \in [0,1]$). In other words, it will act randomly ε % of the time as implemented. Random action is important to exploring the state space as it drives exploration. A high exploration rate would be expected to provide a more robust Q-Table perhaps even in less training episodes. However, high random action will also prevent reaching the goal state despite an accurate Q-table. The results in Table 6 are not conclusive and worse than general QL experiments at ε = 0.5. This is likely due to too much random action preventing "exploitation" of an accurate Q-Table.

Table 6 Q-Learning Reward and Iterations to Convg. vs Training Episodes at ε-greedy = [0.05, 0.25, 0.5, 0.75, 0.8]

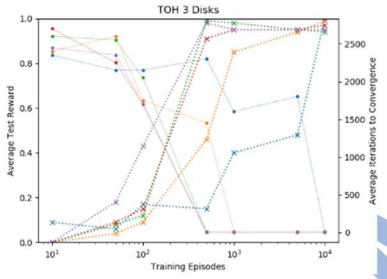

Figure 15 TOH Avg. Test Episode Reward and Steps vs Total Training Episodes

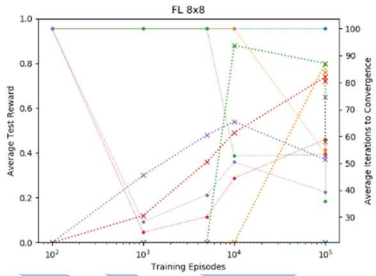

Figure 16 FL 8x8 Avg. Test Episode Reward and Steps vs Total Training Episodes

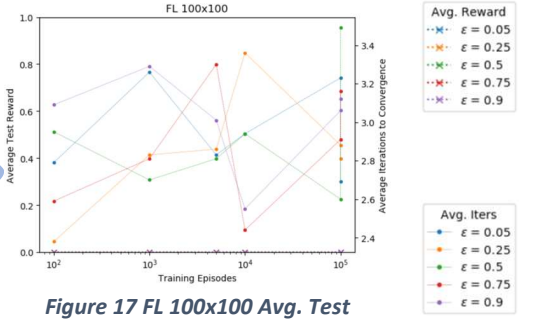

Figure 17 FL 100x100 Avg. Test Episode Reward and Steps vs Total Training Episodes

Finally a regime was attempted to trade off exploration and exploitation. ε is initially set to 1.0 and decayed at a rate *v* for each random action taken. This means the Learner will initially act entirely randomly "exploring" and slowly begin to take its best action "exploiting." The expectation is good reward performance can be achieved in fewer training episodes than without decay. Table 7 presents results where positive reward is almost never achieved. This is most likely due to implementation mistakes. However, it is possible the MDPs with a single game ending goal do not lend themselves well to random exploration as it requires randomly finding a goal that needs a specific action sequence.

Andrew Cudworth
CS XXXX Machine Learning (Spring 2020)
Assignment 4 Markov Decision Processes

Table 7 Q-Learning Explore-Exploit Curve
Reward and Iterations to Convg. vs Training Episodes at ε decay rate (ν) = [0.001, 0.01, 0.05, 0.1]

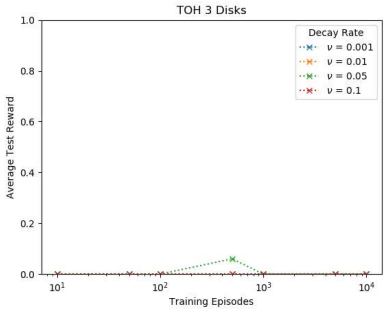

Figure 18 TOH Avg. Test Episode Reward and Steps vs Total Training Episodes

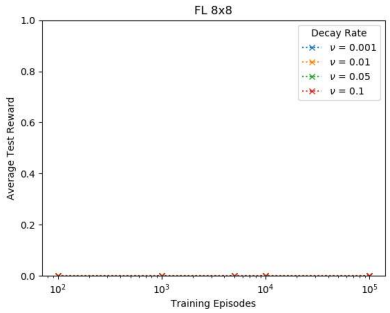

Figure 19 FL 8x8 Avg. Test Episode Reward and Steps vs Total Training Episodes

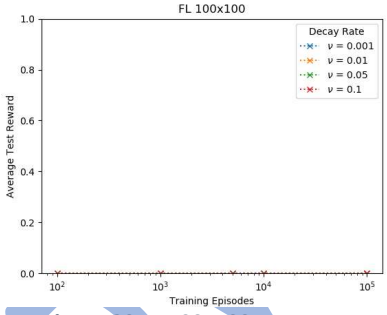

Figure 20 FL 100x100 Avg. Test Episode Reward and Steps vs Total Training Episodes

Results:

QL, VI and PI had their training time compared directly in seconds rather than iterations. Each Learning method used the best performing parameters from the previous section with convergence conditions discussed. Result for TOH are presented in Figure 21 and 22. In the 3-disk state space previously explored, QL achieved similar rewards to VI and PI, but ran for many more iterations (Figures 3,6,9 above). VI and PI converged to the same solution, but QL did not. Training time in seconds shows that the higher iterations for QL translates to roughly the same training time in real time. Above 4 disks QL did not perform well though. As an experiment, the convergence conditions on QL were relaxed to allow a longer run time. Minor improvements could be made in QL > 3 disks by relaxing the convergence conditions, but the run time for such experiments were not feasible.

VI and PI focus on finding a value for each state. Whereas QL focus on finding a value for each state, action pair and requires they be visited. As the state space grows in TOH, relative value of states to each other does not really change only the number of disks. VI and PI perform well in this MDP because there is an objective value for each state and every state is guaranteed to be visited. A QL state, action pair also has an objective value, but requires the Q-learner visit the state action. Additionally, reward is needed for accurate valuation, but becomes much less likely to be found via random exploration as the state space grows.

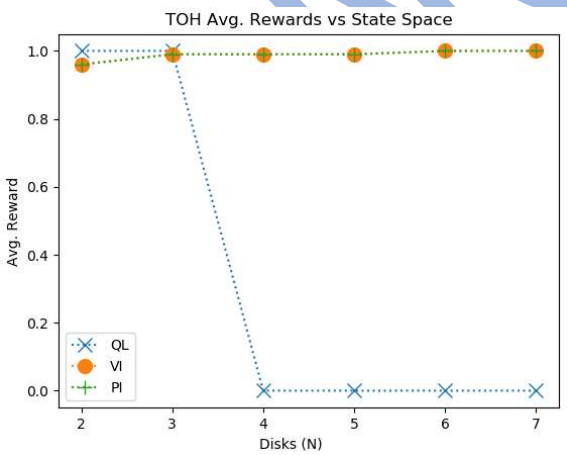

Figure 21 TOH Average Reward Vs. Disks

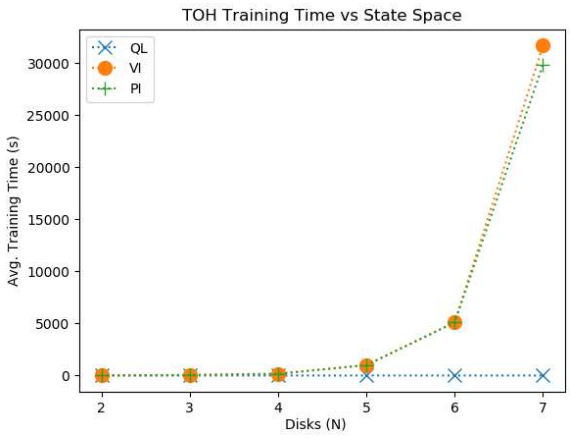

Figure 22 TOH Average Training time vs Disks

Finally, the policy length of VI,PI, and QL was compared to the known optimal solution of 2^N $steps$ and presented in Figure 23. Note that QL is not finding any reward above 4 disks, so its length is less relevant. QL and PI are finding suboptimal solutions by a wide margin though. Training time was increased on each via relaxing the convergence conditions. However, the results did not change significantly from Figure 23. A likely explanation is relates to VI and PI

backpropagating reward based on the transition function $V(s) \leftarrow max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$. This reward shows up in prior states proportional to gamma, but also reinforces the path taken to the reward. This means that VI and PI will value the first solution that finds the reward rather than the fastest solution. This could be addressed in future work via negative incremental reward or reward shaping.

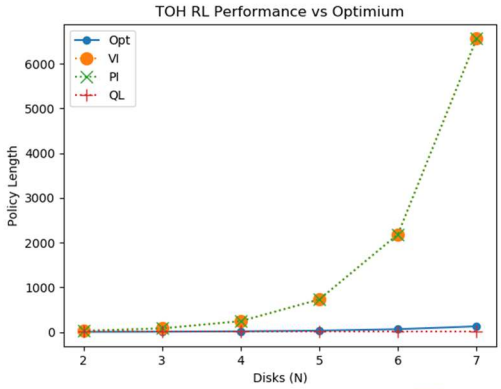

Figure 23 TOH RL and Optimal Policy Length vs State Space

For the Frozen lake problem, QL, VI and PI did not perform well above a 20x20 grid world even with much higher training time in real time (S). This is interesting when contrasted with TOH performance. The legal state space of a 7 disk TOH in figure 22 is 2187 states. The grid world loses its performance at N = 20 a state space of 400. Notably, training time is far higher, and results are far worse for the same VI and PI with the same parameters. This is explained by the stochastic nature and game ending "holes" in the frozen lake problem. States are harder to find a value function for because their next state is random to some degree. Additionally, holes can create negative reward that prevents finding a path to positive reward if reinforced too much within a training episode.

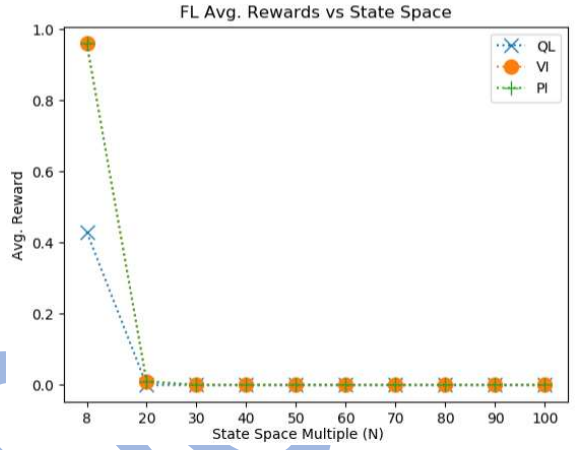

Figure 24 FL Average Reward Vs. Grid Size

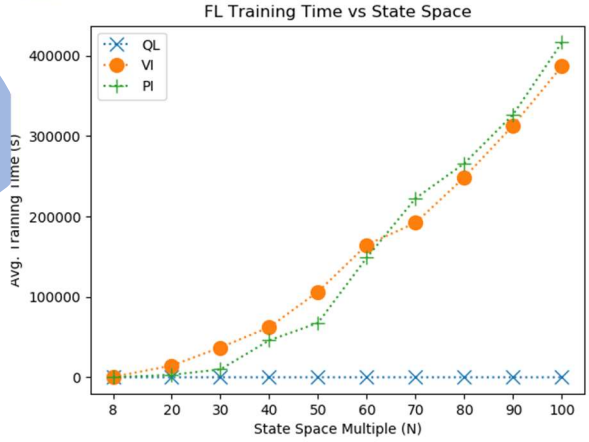

Figure 25 FL Average Training time vs Grid Size

Conclusion

QL, VI, and PI were run on a small deterministic Tower of Hanoi problem with a known optimal solution. Hyper parameter tuning was performed for each, and all three achieved good performance with similar wall clock run times. PI had the least iterations followed by VI then QL. As the number of disks in the TOH problem were increased, VI and PI still found well performing solutions at the expense of exponentially more training time. QL was not able to solve higher state space TOH problems likely due to insufficient exploration. Both VI and Pi were well below the known optimal solution in terms of policy length due to favoring the first rather than shortest solution. The same process was repeated on a small 8x8 and large 100x100 stochastic grid world with game ending grid squares called "Frozen Lake." Reward performance was worse for all methods on this MDP than TOH as the methods were heavily influenced by the stochastic

action. At the expense of training iterations, positive performance was achieved by all methods for the small 8x8 grid world, but never for the large 100x100 grid world. In fact, positive performance was never achieved for a grid world larger than 20x20 showing a much worse performance for small stochastic state spaces vs large deterministic state spaces.

SOURCES

[1] Barto, Andrew G., and Richard S. Sutton. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. 2nd ed., MIT Press., 2018.
[2] Kalita, Diganta. "Realdiganta's Gists." *Gist*, gist.github.com/realdiganta.
[3] Kalita, Diganta. "Solving the FrozenLake Environment from OpenAI Gym Using Value Iteration." *Medium*, Analytics Vidhya, 2 Dec. 2019, medium.com/analytics-vidhya/solving-the-frozenlake-environment-from-openai-gym-using-value-iteration-5a078dffe438.
[4] LearnDataSci, Brendan Martin Founder of. "Reinforcement Q-Learning from Scratch in Python with OpenAI Gym." *Learn Data Science - Tutorials, Books, Courses, and More*, www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/.
[5] OpenAI. "A Toolkit for Developing and Comparing Reinforcement Learning Algorithms." *Gym*, gym.openai.com/envs/FrozenLake-v0/.
[6] Snapp, Robert. "Tower of Hanoi." *Hanoi.pdf*, University of Vermont, Sept. 2012, www.cs.uvm.edu/~rsnapp/teaching/cs32/lectures/hanoi.pdf.
[7] Waqasqammar. "Waqasqammar/MDP-with-Value-Iteration-and-Policy-Iteration." *GitHub*, 22 Sept. 2018, github.com/waqasqammar/MDP-with-Value-Iteration-and-Policy-Iteration/.
[8] Xadahiya. "Xadahiya/Toh-Gym." *GitHub*, github.com/xadahiya/toh-gym/blob/master/toh_gym/envs/toh_env.py.