1. Introduction

This paper applied some of the techniques from reinforcement learning to make decisions. To measure and compare the performance of these algorithms, two discrete, stochastic Markov Decision Processes (MDPs) were explored:

- Frozen lake:
  A grid-world problem with a start state and an end state. The agent can only step into adjacent state on each move, while the directions of each move are not fully determined by the agent. Both reaching the goal or fall into a trap end the game. The agent will be rewarded for finding the end state.
- Forest management:
  A non-grid-world problem without an end state. The moving restriction is not applied. The decisions of the agent maybe interrupted by some random events. The agent will be rewarded when certain conditions are satisfied.

An MDP can be represented as (S, A, T, R, γ), where:

- S (States): a set of all possible states s.
- A (Actions): a set of actions that can be performed on each state *A(s).*
- T (Transitions function): probabilities of moving from one state to another for a given action
  $$T(s, a, s') \sim Pr[s' \mid s, a]$$
- R (Rewards): a matrix of rewards based on a given state and action
  $$R(s, a, s')$$
- γ (Discount factor): a fractional factor introduced to encode time into our utility and model the agent to prefer instant gratification. Decrease how impactful future rewards are, letting us converge to a finite reward after taking infinite steps.

Given the description of the MDP, the goal is to find the optimal policy, a set of actions $\Pi(s) \rightarrow a$, to maximize the total reward and to determine the optimal actions for any given state. The optimal policy can be found by solving the Bellman equation represented by the utility of a state:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} T(s, a, s')U(s')$$

I performed three reinforcement learning algorithms to solve the Bellman equation:

- VI - Value iteration
  Start with arbitrary utilities, update them based on their neighbors, iterate repeatedly until convergence:

  $$\widehat{U}_{t+1}(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s, a, s')\widehat{U}_t(s')$$

  Since the rewards propagate through their neighbors, we'll eventually converge by overwhelming the initial random guessing $\widehat{U}_0(s)$. The iteration can be terminated when maximum change in the value function is less than a maximum delta threshold.
- PI - Policy iteration

Start with a random policy Π_0, calculate the utility by following this policy:

$$U_t(s) = R(s) + \gamma \sum_{s'} T(s, \Pi_t(s), s')U(s')$$

Then, given that utility, we can figure out how to improve the policy by finding the action that maximizes the expected utility:

$$\Pi_{t+1}(s) = \underset{a \in A(s)}{argmax} \sum_{s'} T(s, a, s')U_t(s')$$

The utility and policy are improved alternately until convergence. With this formulation, we have n linear equations which can be solved with linear algebra. The iteration can also be terminated when maximum change in the value function is less than a maximum delta threshold.

- QL - Q-Leaning
  A model-free learner can deal with situation when the states, the transition functions, the rewards may be hidden. Only after taking an action can we see its effect on the new state and resulting reward.
  Start by assigning all states a Q-value:

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

  The utility and policy can be represented as:

$$U(s) = \max_a Q(s, a)$$
$$\Pi(s) = \underset{a}{argmax}\, Q(s, a)$$

  By visiting each state to iteratively evaluate the Q-value based on immediate and delayed rewards.

$$\langle s, a, r, s' \rangle: \quad \hat{Q}(s, a) \overset{a_t}{\leftarrow} r + \gamma \max_{a' \in A(s')} \hat{Q}(s', a')$$

  It's a tradeoff between exploitation and exploration. Exploration factor ε are used to determine this behavior:

$$\hat{\Pi}(s) = \begin{cases} \underset{a \in A(s)}{\arg\max}\, \hat{Q}(s, a) & \text{with probability } 1 - \varepsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

  The agent takes random actions for probability ε and greedy actions for probability (1-ε).

In order to perform VI and PI, the two MDPs have known reward and transition matrix. The results of each MDP are in section 2, 3 respectively.

The implementations of these algorithms and MDPs were pulled from the OpenAI Gym and hiive mdptoolbox library.

2. Frozen lake

The agent controls the movement of a character in the middle of a lake. The water is mostly frozen, but there are holes where the ice has melted. If you step into one of those holes, you will fall into the freezing water. Since the ice is slippery, the moving directions are uncertain and not fully determined by

the agent as intended. The episode ends when the agent reaches the goal or falls in a hole. The agent receives a reward of 1 for finding a walkable path to a goal tile, and zero otherwise.

This is a typical grid-world problem. The surface can be described using a grid:

```
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)
```

I set the probability of fixed tile as 0.8. Three different sizes were chosen, 4*4, 8*8, 15*15.

```
[['SFFF']  [['SFFFFFFH']  [['SFFFFFFHHFFFFHF']
 ['FFFH']   ['HFFFFHFF']   ['FFHFHHFFFFFFHFF']
 ['HFFF']   ['FHFHHFFF']   ['FFFFFFFFHFFFFFF']
 ['FHFG']]  ['FFFHFFFF']   ['FFFFFFFHFFFFFFF']
            ['FFFFFFHF']   ['FFFFFFHFHFHFHFF']
            ['FFFFHFFF']   ['FFFFFFFFFFFFFFH']
            ['FFFFFFFG']]  ['FFFFFFFFHFFFFHF']
                           ['FFFFHFHFFHFHFHF']
                           ['FFHFFFFFFFFFFFF']
                           ['FFFFFHFFHHFFHFH']
                           ['FHFFFHHFFFFFFHH']
                           ['HFFFFFFFFFFFFHF']
                           ['FFFFHFFFFFFFFHF']
                           ['FFFFFFFFFFFFFFH']
                           ['HFFFFFFFFFHFHFFG']]
```

The experiment was run in 3 steps:

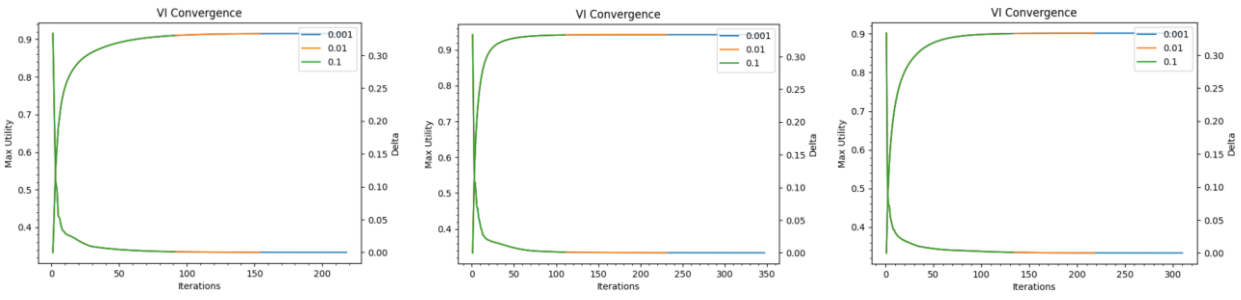
- Step 1, tune the hyper-parameters for each algorithm according to each size, evaluate their convergence properties.
- Step 2, evaluate how problem sizes affect the performance of each algorithm.
- Step 3, change the discount factor (gamma) to observe the impact on the results.

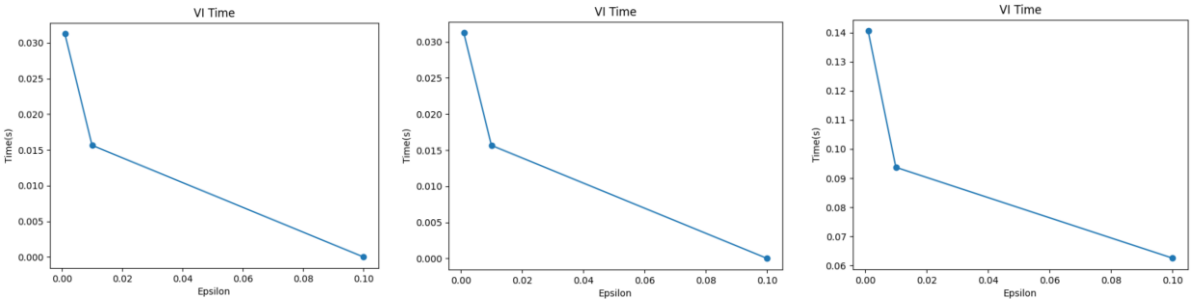
Performance are evaluated based on three factors: utility, number of iterations and time consumption.

2.1. Convergence property

Value iteration:

I set gamma as 0.99, the algorithm gives most importance to the future rewards. The iteration will stop when max_iter=1000 is reached or small enough delta error is achieved.



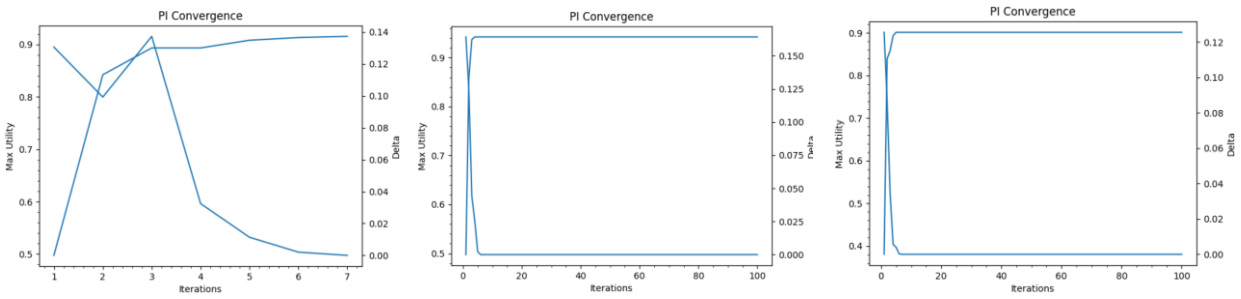


Above are results of problem size 4*4, 8*8, 15*15 from left to right. I plotted Max utility and Delta over iteration to illustrate the converge process. Three different epsilons 0.1, 0.01, 0.001 were used.

As shown in the results, epsilon doesn't change the converge behavior. It's the stopping criteria to determine when the model is converged. The smaller the epsilon, it took more time to find a better policy. While it failed to achieve the optimal policies for both problem sizes.

Policy iteration:

I set gamma=0.99. The iteration will stop when max_iter=100 is reached or small enough delta error is achieved.



Above are results of problem size 4*4, 8*8, 15*15 from left to right. I plotted Max utility and Delta over iteration to illustrate the converge process.

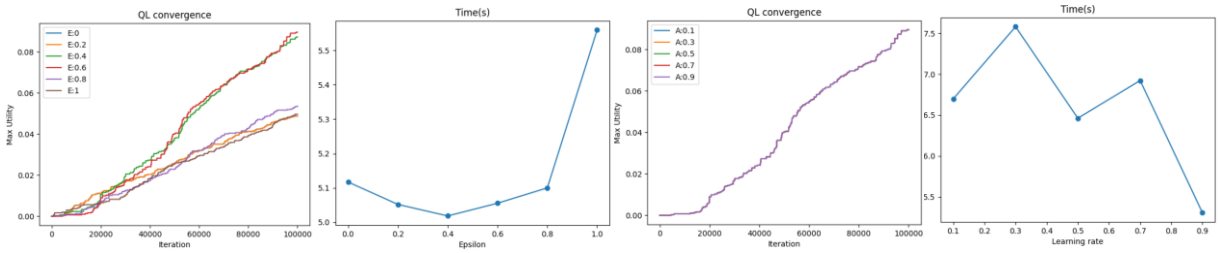
Compare to the results of VI, PI converged much faster with better policies achieved. Since the action in each states is fixed by the policy, PI produces linear equations because the "max" operator is removed, which make it more efficient to solve the Bellman equation.

Q-learning:

I set gamma=0.99 and n_iter as 100000, 200000, 400000 for each problem size. The iteration will not stop until the n_iter is reached.

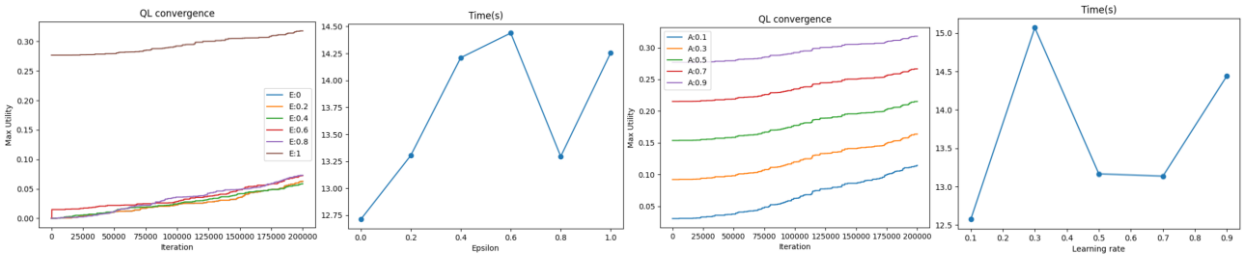
I first set learning rate alpha=0.9 to choose the best exploration strategy epsilon. Then find the optimal alpha based on that epsilon.

4*4:



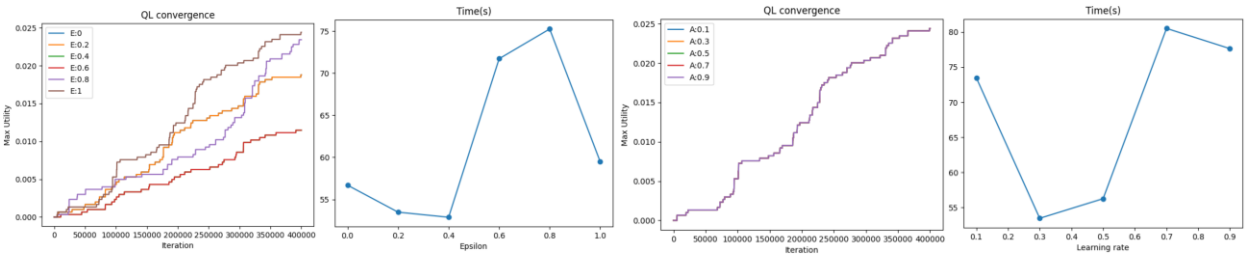
epsilon=0.6, alpha=0.9

8*8:



epsilon=1, alpha=0.9

16*16:



epsilon=1, alpha=0.9

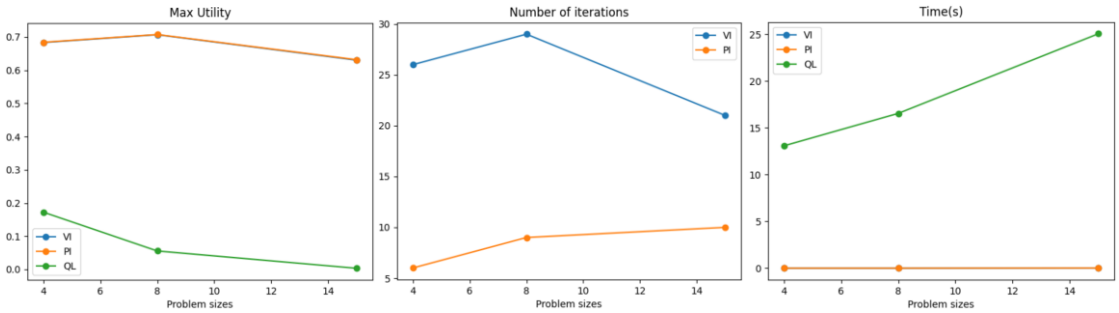
I plotted Max utility over iteration to illustrate the converge process.

From the results above, Q-learning responded poorly to this MDP compared to VI and PI. It may because only stepping into the goal state can receive a non-zero reward. Therefore random explorations were preferred with a larger epsilon (0.6 or 1). The models were still not quite converged after plenty of time. It supposed to take longer because Q-learning is a sequential algorithm and cannot be parallelized. Besides, Q-learning suffers from stochastic exploration because the convergence was determined by the maximum deltas for all states. I would prefer PI or VI to waiting it converge in this case. We may need more immediate reward to improve the performance.

Since the epsilon determines the exploration behavior, we can see an increasing trend of time consumptions along with the increasing of epsilon from the results. The learning rate determines how often the Q-values are updated. With a larger learning rate, the model is supposed to learn more quickly if we rely on the Q-values. This theory cannot be applied to this MDP because of the stochastic exploration, no obvious trend in processing time can be observed.

2.2. Problem complexity

I used three different sizes to show how the number of states affect the results: 4*4, 8*8, 15*15. I set gamma=0.99. The iteration of VI and QI will stop when max_iter=1000 is reached or small enough delta error is achieved. The iteration of QL will not stop until the n_iter=200000 is reached.



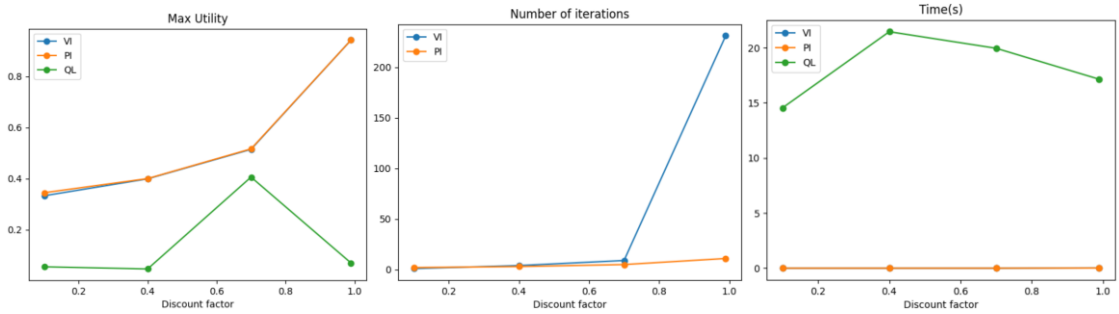
I used the max value of the value function learned by each algorithm to evaluate the quality of the policy. As shown in the first figure, although not optimal one, PI and VI returned similar policies. And their performances were not affected by the size of MDPs since the implementations of VI and PI are 100% vectorized in numpy. While the QL performed badly in this case, because it took way too long to fully converge. We may need to add more immediate reward to assist it. The quality of the policies QL learned deteriorated with increasing number of states. As we mentioned in the previous section, exploration behavior were preferred in this case. It took longer to sample the states space for larger MDP.

Since no early termination for QL, we only compare the number of iterations for PI and VI. From the second figure, VI needed more iterations to converge than PI. We cannot say for sure how number of states affects this results because of the random nature of this two algorithms.

As shown in the third figure, both PI and VI were computational efficient and were not affected by the size of MDPs, benefiting from parallelism. Combined with the results from the second figure, PI required more calculation on each iteration. As expected, QL took much more time.

2.3. Discount factor

I used different discount factors to show how they affect the results: 0.1, 0.4, 0.7, 0.99. Mathematically, the discount factor needs to be set less than 0 for the algorithm to converge. The environment is the 8*8 Frozen lake. The iteration of VI and QI will stop when max_iter=1000 is reached or small enough delta error is achieved. The iteration of QL will not stop until the n_iter=200000 is reached.



I used the max value of the value function learned by each algorithm to evaluate the quality of the policy.

The discount factor measures how valuable future rewards compared to immediate rewards. With a smaller gamma, the agent prefers instant gratification and is more likely to return a local optima, which is proved by the first figure. And with a smaller gamma, the model is supposed to converge faster. Since the randomness of these algorithms, the results of a single run may not sufficient to establish statistically significant comparisons on iteration and run-time.

3. Forest management

A forest is managed by an agent with a state in {0, 1, … s - 1}. On every year, the forest grows to the next state, an action is decided to 'Wait' or 'Cut'. If 'Wait' is performed, the old forest is maintained for wildlife. If the forest is in its oldest state, the agent is rewarded with r1, and zero otherwise. If 'Cut' is performed, the forest turn back to its youngest state. The agent can sell cut wood and get a reward of r2 if the forest was in oldest state, and 1 otherwise. No matter which state the forest in, after a wild fire occurs with a probability of p each year, the forest is in the youngest state.

With 'Wait' be action 0 and 'Cut' be action 1, the transition and reward matrix can be defined as follow:

$$P[0,:,:] = \begin{bmatrix} p & 1-p & 0 & \cdots & \cdots & 0 \\ . & 0 & 1-p & 0 & \cdots & 0 \\ . & . & 0 & . & & \\ . & . & & . & & \\ . & . & & & 1-p & \\ p & 0 & 0 & \cdots & 0 & 1-p \end{bmatrix} \quad R[:,0] = \begin{bmatrix} 0 \\ . \\ . \\ . \\ 0 \\ r1 \end{bmatrix}$$

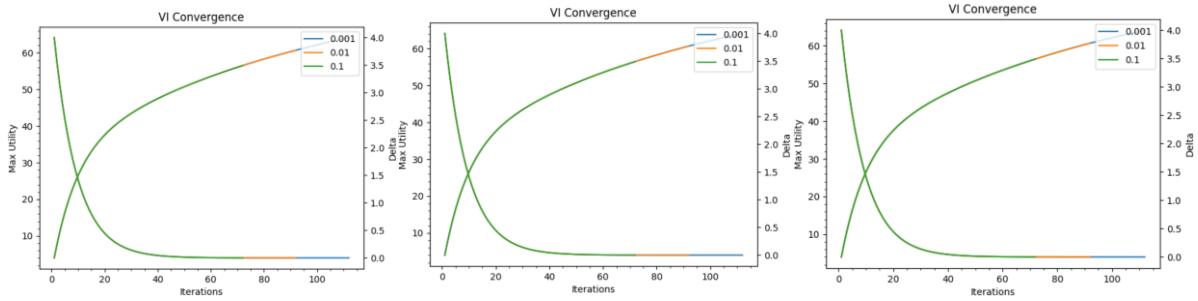
$$P[1,:,:] = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \\ . & . & & & & . \\ 1 & 0 & \cdots & \cdots & \cdots & 0 \end{bmatrix} \quad R[:,1] = \begin{bmatrix} 0 \\ 1 \\ . \\ . \\ 1 \\ r2 \end{bmatrix}$$

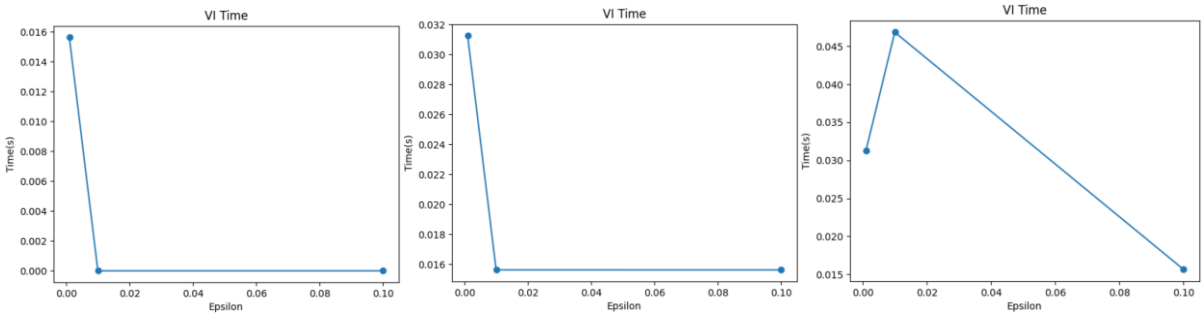
I set the probability of wild fire as 0.1, r1 as 4 and r2 as 2. Three different sizes are chosen, 20, 100, 400. This experiment followed the same steps as described in section 2.

3.1. Convergence property

Value iteration:

I set gamma as 0.99 and max_iter=1000.



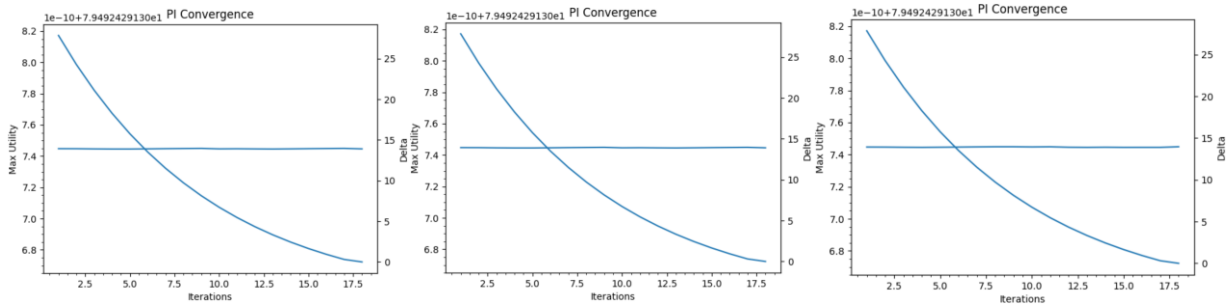


Above are results of states 20, 100, 400 from left to right.

It shows that epsilon doesn't change the converge behavior. The smaller the epsilon, it took more time to find a better policy. Since this is a non-grid MDP, it has no termination state, the utility keeps increasing. While both models with different numbers of states returned the same policy. It may because the long-term rewards r1 (4) and r2 (2) were not attracting compared to immediate reward (1). Instant gratification was preferred.

Policy iteration:

I set gamma=0.99 and max_iter=100.

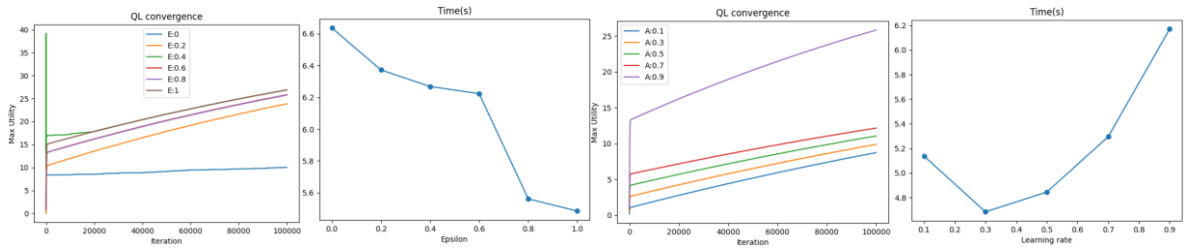


Above are results of states 20, 100, 400 from left to right. Same policy was return by PI as well. And since PI converges faster, it stop early with a not good enough policy. We may need to decrease the short-term reward or increase r1 and r2 to see a difference.

Q-leaning:

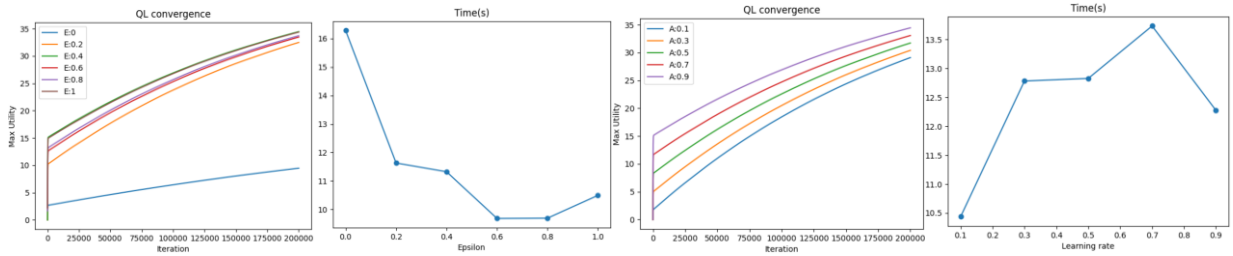
I set gamma=0.99 and n_iter as 100000, 200000, 400000 for each number of states. I first set learning rate alpha=0.9 to choose the best exploration strategy epsilon. Then find the optimal alpha based on that epsilon.

20:



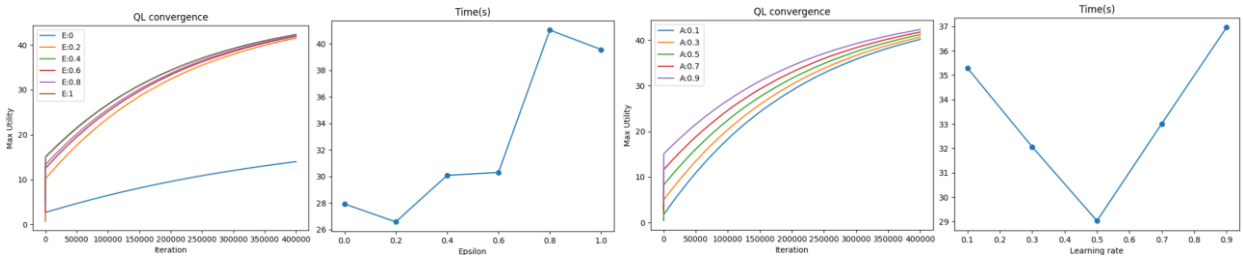
epsilon=0.4, alpha=0.9

100:



epsilon=0.4, alpha=0.9

400:



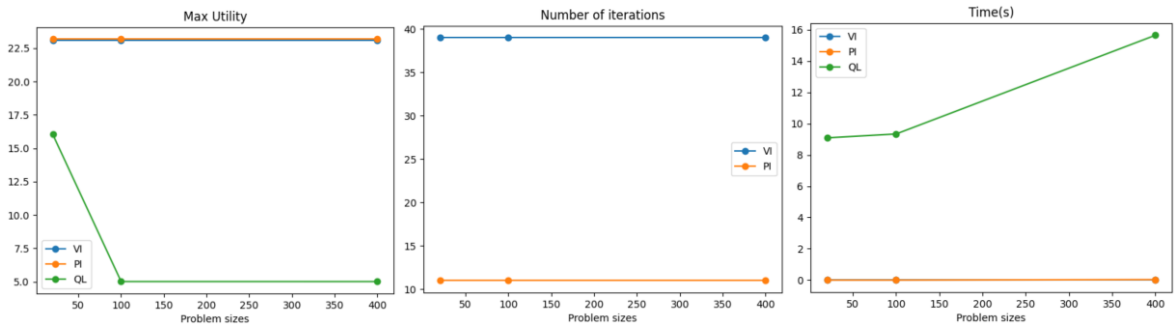
epsilon=0.4, alpha=0.9

I plotted Max utility over iteration to illustrate the converge process.

From the results above, Q-learning learned this MDP better compared to the first grid-world one. There were plenty of immediate rewards, therefore exploitations were preferred with a smaller epsilon (0.4) and a larger alpha (0.9). The result of epsilon=0 was the worst, it proved the fact that we should not exploit the Q-value too early. There may have states we haven't explored lead to better rewards. Since there is no termination state, the utility keeps increasing. It still took longer to converge as we mentioned before.

3.2. Problem complexity

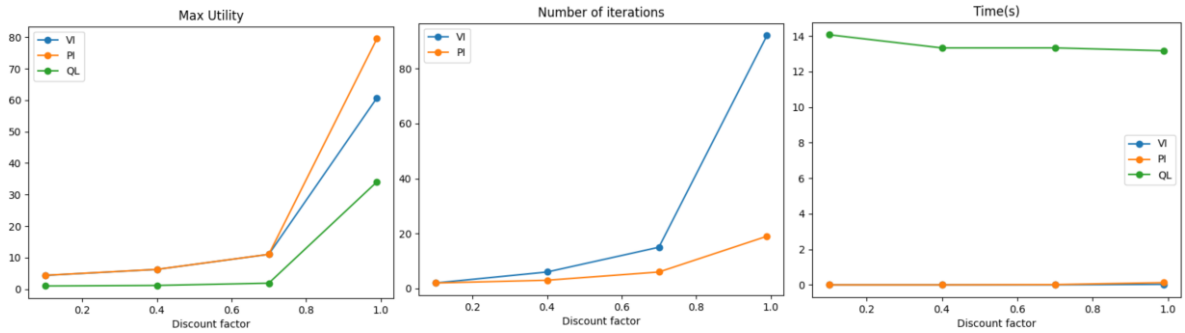
I used three different sizes to show how the number of states affect the results: 20, 100, 400. I set gamma=0.99, max_iter=1000 n_iter=200000.



Still, QL needed more time to have better understanding of all states. No more interesting findings from this results.

3.3. Discount factor

I used different discount factors to show how they affect the results: 0.1, 0.4, 0.7, 0.99. The environment is the forest with 400 states. I set max_iter=1000 and n_iter=200000.



It perfectly proved the fact that with a smaller gamma, the agent prefers instant gratification and the model will converge faster.

4. Conclusions

The environment's long-term and short-term rewards largely affect the behavior of learning algorithms. VI and PI are both computational efficient except that VI needs more iterations to converge. While QL takes much longer. The exploration strategy is crucial for QL. If exploiting the Q-value too early, there may have states haven't explored lead to better rewards. If exploiting the Q-value too late, the run-time may suffer. The discount factor will affect the convergence behavior and speed.

Reference

Kaelbling, Littman, Moore. Reinforcement Learning: A Survey.

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction