Markov Decision Processes

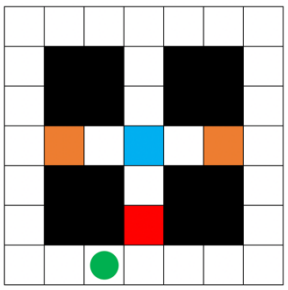
Ting-Yu Kang 903443289 tkang49

Two Interesting MDPs

1. MDPs overview

   (1) Small-State Problem (7x7, 33 states):

   I designed this problem to observe the behavior of an agent (green dot) to reach the goal (blue grid) when encountering different choices of paths. Specifically, there are 4 different paths, but the closer path has larger penalty. The reward of the red grid is -3, the orange grids are -2, the goal is 10, and the remaining grids are -0.3. The probability of the agent following the intended policy is 0.8, while randomly moving toward the other directions with 0.2.



   (2) Large-State Problem (21x21, 201 states):

   This is a randomly-generated maze problem using Aldous-Broder algorithm. The agent (green dot) would move from upper left to the exit (blue grid) at lower right. The reward of white grids are -0.5, and the goal is 10. The probability of the agent following the intended policy is 0.8, while randomly moving toward the other directions with 0.2.

2. Interesting points

   (1) The small-state problem is useful in real life. Imagine the goal at the center is a house on fire, and the agent is a fire truck that trying to find a path to enter. The red grid and orange are gates which are closer but take time to break in, while the path without a gate is farther. This problem is interesting and worth exploring that under what condition the agent might change its choice.

   (2) The large-state problem is also interesting with lots of implementations such as route search for autopilot car. This problem serves as a toy example without losing generality that helps me understand the process of deciding the optimal direction at every location. In this large-state problem, I will focus more on the performance issue.

3. Tool

   In this assignment, I implemented the planning and learning algorithms using **Burlap** library from Brown university. The programming language I used is Java.

4. Reinforcement Planning and Learning Algorithms

   (1) Value Iteration (VI):

   VI randomly initializing utilities on each state, and iteratively update utilities by choosing an action that maximizes the expected reward. To make nearer states more valuable, a discounted factor is added to urge the agent to reach the goal as soon as possible.

   (2) Policy Iteration (PI):

   In each iteration, PI generates policies of each state according to the utilities of VI and calculate new utilities from the policies for the next round of VI. PI usually needs fewer iterations to converge but requires more calculations (policy extraction) than VI.

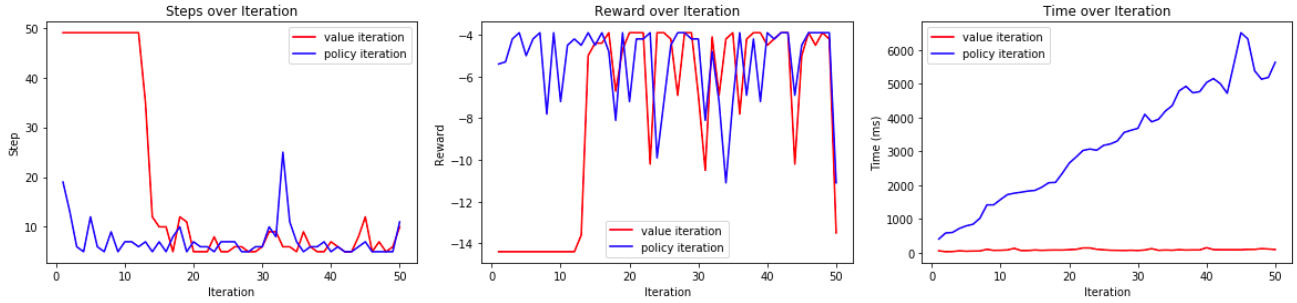
   (3) Q-Learning:

The previous two planning algorithms assume that the model and transition function are perfectly known, but it's unrealistic to always know the exact transition model in real life. Q-Learning is what we want: it learns from observing the outcome reward without a model, and iteratively updates and approaches the optimal Q-value. However, it might take more iterations to converge and get sub-optimal results.

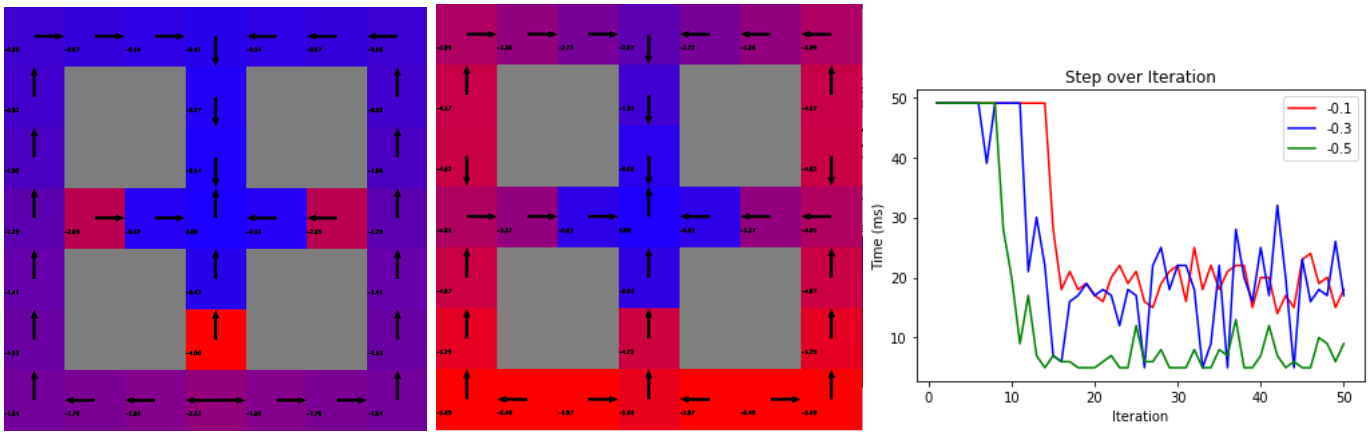
Value Iteration and Policy Iteration

In this part, I will use VI and RI to find the optimal policies for the underlying MDP problems. After that, I will analyze the results to compare both algorithms and how different number of states could affect the outcome.

1. Small-State Problem
   (1) Parameters: discount factor: **0.99**, max iteration: **50**. To observe the behavior without early stop, I set max delta = **-1**.
   (2) Reward: goal: **10**, lower gate: **-3**; left/right gates: **-2**; other states: **-0.3**.



   (3) From the step chart and reward chart, we can see that both planners converge to similar results. It makes sense because they both utilize the Bellman Equation as their core function.
   (4) Value Iteration (red): We can see that the number of steps converges around 20 iterations, at the same time, the reward rises from -14 to -4.
   (5) Policy Iteration (blue): The number of steps converges within 5 iterations, but it takes a lot more time (37.6 times on average) than VI because of back-and-forth conversion between policies and utilities.
   (6) The reason PI converges faster might because there are only four actions (up, right, left, down). Even though the utilities are not that perfect, PI can still generate policies that close to optimum depending on the utilities. Therefore, PI needs less iteration to converge.
   (7) After 20 iterations, the fluctuation of the reward graph might because it's a tiebreak for the agent to decide between down gate or left gate. I will prove it in (8).
   (8) To explore how different reward functions can affect the policy, I use different reward settings (-0.1 and -0.5) and visualize the optimal policies:



(a) In the left graph, I set reward = -0.1 for the states other than the goal and gates. Because of the penalty of changing states is small, the agent would rather travel farther to the top than get a higher penalty at the closer gates.

(b) In the middle graph, I set reward = -0.5, so the traveling penalty is high. The agents would choose the nearest gate to minimize the penalty.

(c) In the default setting, I set reward = -0.3 on purpose. In the right chart, we can see that the red (-0.1) and green (-0.3) lines are stable, but the blue (-0.3) line fluctuates between red and green lines. It can be concluded that, -0.3 is a draw that the agent can either choose the nearest gate or travel farther to enter the top path without a gate. There is little difference between different policies when the reward is set to -0.3, which explains the fluctuation after 20 iterations in (2) graphs.

2. Large-State Problem

(1) Parameters: discount factor: **0.99**, max iteration: **200** (VI) and **30** (PI). max delta = **0.01**.

(2) Reward: goal: **10**, other states: **-0.5**.

(3) Value Iteration:

(a) We can see that VI converges at around 120 iterations, where the steps sharply decrease from 441(21×21, max number of grids) to around 130, and the reward also surges from -220 to -70. This shows that the agent doesn't reach the goal before 120 iterations and bouncing back and forth until terminated because of reaching max steps allowed (441).

(b) The calculation time stops increase at around 150 iterations, showing that the difference of utilities between iterations is below the max delta threshold (0.01). Therefore, VI terminates around 150 iterations.

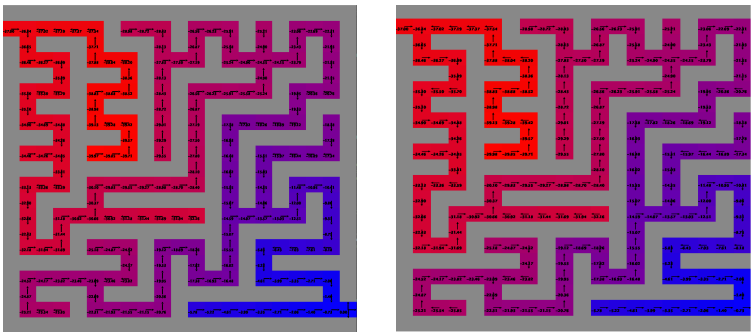
(4) Policy Iteration:

(a) In contrast to VI, PI needs only 8 iterations to converge. The optimal number of step and reward are very close to the result from VI, showing that PI is as effective as VI.

(b) However, the running time of PI is about 20 times more than VI. Same as the small-state problem, PI needs additional calculation for policy evaluation. That's also the reason why PI only needs 8 iterations to converge but VI requires 120 iterations. The imperfect value function can still generate perfect policy.

(c) The time stops increase at 10 iterations, showing that the difference between policies is smaller than max delta threshold (0.01).

(5) Optimal policies:

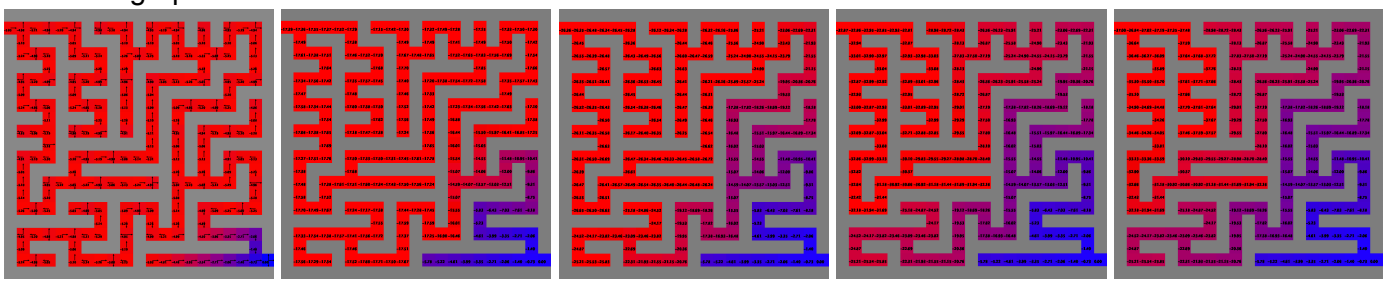


(a) The left graph is VI of 200 iterations, and the right one is PI of 30 iterations. We can see that the color regions of the two graphs are nearly the same. Also, the utilities of the starting point are both -37.00.

(b) The comparison shows that VI and PI have the same ability of policy generation. I will make another comparison with Q-learning in the next part.

(6) Convergence analysis:

(a) Because VI and PI have nearly the same computation logic, I choose VI to demonstrate the policy graph in different iterations:



10 40 70 100 130

(b) We can see the utilities of the goal propagated from the lower right to upper left. Because there is only one way on the top right that connects the upper left and lower right part, the agent cannot find a path to the goal until the utilities propagate to the upper right states: 100-130 iterations. Therefore, the sudden improvement of steps and rewards in around 120 iterations makes sense.

(c) Changing the learning rate and discounted factor doesn't affect the convergence result too much. The reason might because the maze is not complex enough and the number of states isn't large enough either. Also, the rewards of all the states except the goal state are negative, no matter how unimportant the far states are, it still better for the agent to reach the goal instead of wandering around and being punished.

3. Conclusion

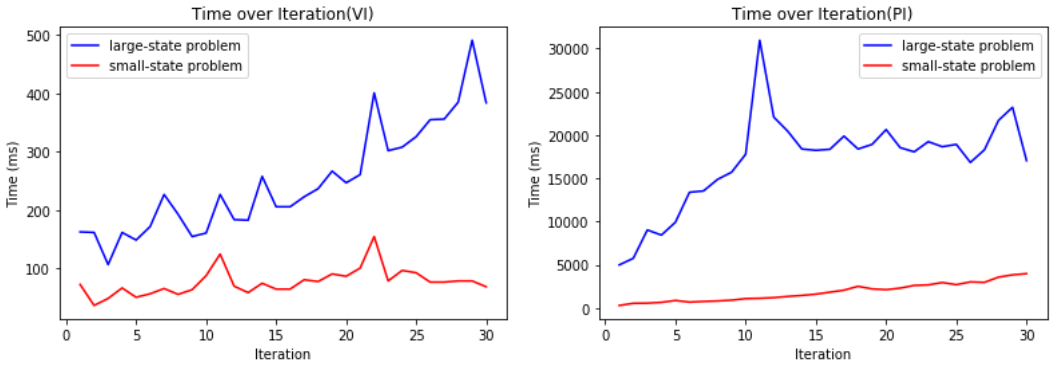
(1) The two Model-based planning algorithms give us very similar results because of using the Bellman equation as core function.

(2) Value Iteration needs more iterations than Policy Iteration to converge. Because for VI, utilities are continuous variables with arbitrary large value combination, so it requires many iterations to fine-tune the value functions. In contrast, Policy Iteration only needs several iterations to converge. Because the action space is limited, it is a lot easier to get optimal policy given imperfect utilities.

(3) However, Policy Iteration needs huge time per iteration compared to Value Iteration. PI needs to generate a policy from the utilities of previous iteration and calculate new rewards with this policy for the next iteration. In contrast, value Iteration is a lot faster because it only needs to update utilities

from the previous iteration. The different magnitude of computing complexity causes a huge gap of per-iteration running time.

(4) The larger number of free states (excluding black obstacles), the more executing time it takes to compute optimal policy:



The ratio for the executing time of large-state/small-state is roughly equal to the ratio of the number of states. It's because VI and PI are both dynamic programming problem with polynomial time complexity that iteratively updates utilities for each state. Therefore, the execution time will be in proportional to the number of states.

Favorite Reinforcement Learning Algorithm: Q-Learning

In this part, I choose Q-learning as my favorite. Q-Learning is the most popular RL algorithms that is widely used in robotics, traffic signal controls, web networks configuration, and so on. As a learning algorithm, Q-learning is model-free and able to learn from the reward received. Another reason I choose Q-learning is that it is taught in class, and I aspire to implement and feel what I learned instead of just learning math formula. In the following part, I will use the same MDP problems and run Q-learning on them to find anything interesting.

1. Small-State Problem

(1) Parameters: discount factor: **0.99**, max iteration: **1000**, initial state: **0,** epsilon: **0.1**, learning rate: **0.1**.

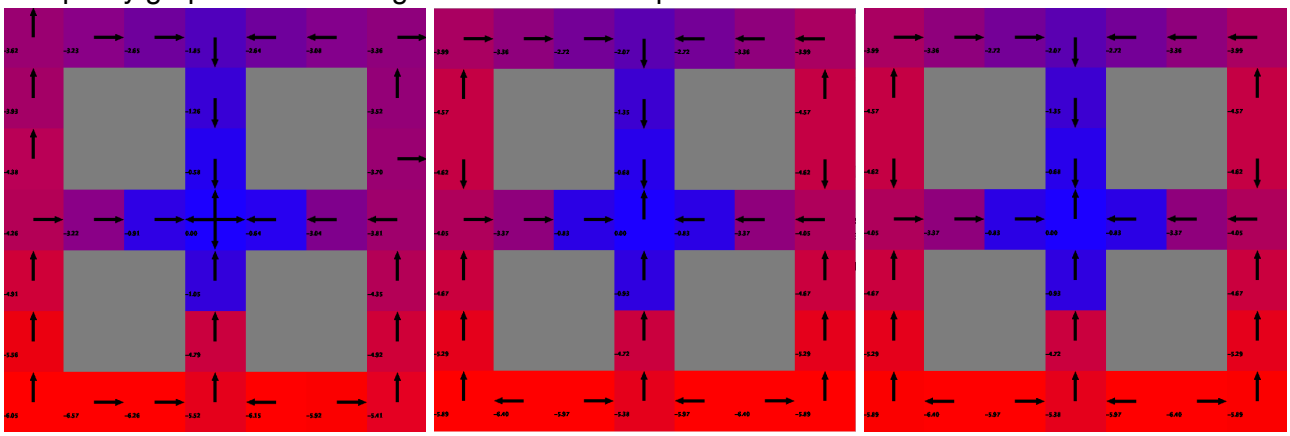
(1) Reward: goal: **10**, lower gate: **-3**; left/right gates: **-2**; other states: **-0.3**.

(2) Because Q-learning doesn't know the model of MDP and learn from the reward returned, it needs more steps than VI and PI to solve it. Moreover, Q-Learning is a light-weight algorithm for it just needs to find the next state and update it, while VI/PI needs to update the whole domains iteratively. To be more specific, Q-learning only needs 20-40 ms per iteration whereas VI needs 70-120 ms per iteration and PI needs thousands of ms per iteration.

(3) The result fluctuates a lot, but we can still see the result converges at around 200 iterations. The fluctuation might because of the epsilon factor isn't large enough. By increasing ε (randomness), the variance might be lower and more compact. I will discuss different exploration strategies in (7).

(4) Compare the optimal reward and steps with the result from VI/PI (blue line), we can see they are very close, but VI/PI is slightly better with a higher reward. It makes sense because Q-learning doesn't know the models, but VI/PI does.

(5) The policy graph of Q-Learning shows that it is as powerful as VI/PI:

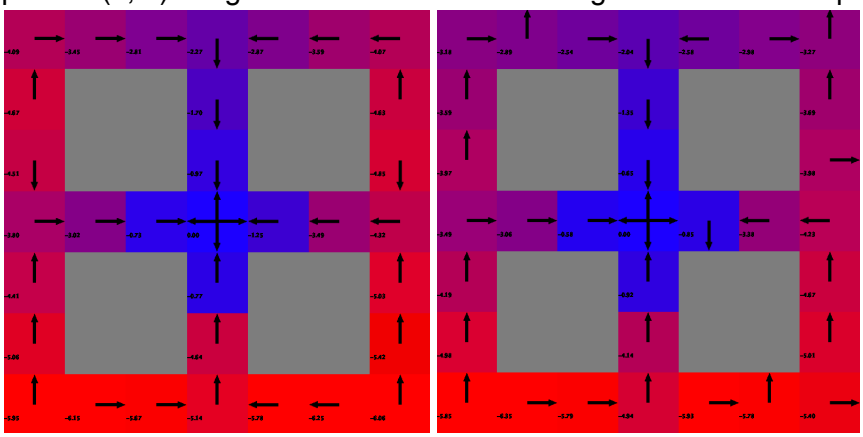


Q-Learning Value Iteration Policy Iteration

We can see that in this case, the optimal policy is to reach the goal as soon as possible regardless of the penalty of gates.

(6) Different Exploration Strategies ($\varepsilon = 0$, $\varepsilon = 0.1$, and $\varepsilon = 1$):

(a) Similar to the simulated annealing algorithm, Q-Learning uses the epsilon (ε) variable to prevent trapping in local optima by moving to some random states with probability ε. I tried two extreme cases of epsilons (0, 1) to figure out how different strategies can affect the policy returned:

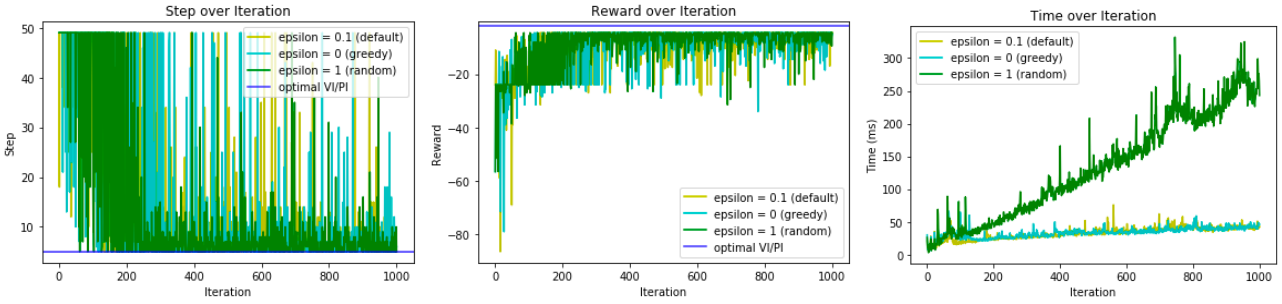


$\varepsilon = 1$(always random) $\varepsilon = 0$(always greedy)

(b) We can see that the random policy graph (left) is the same as the default case where $\varepsilon = 0.1$, it is because the algorithm will always explore the whole domain, and more likely to find the optimal policy.

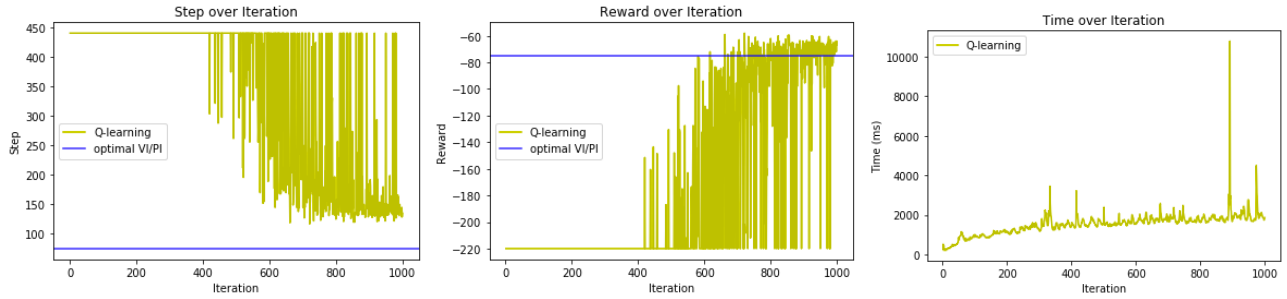
(c) The greedy policy graph (right) is not absolutely correct because it doesn't explore the domain at all, so the Q-value of the states would easily stick to the local optimum.

(d) Compare different strategy with charts:

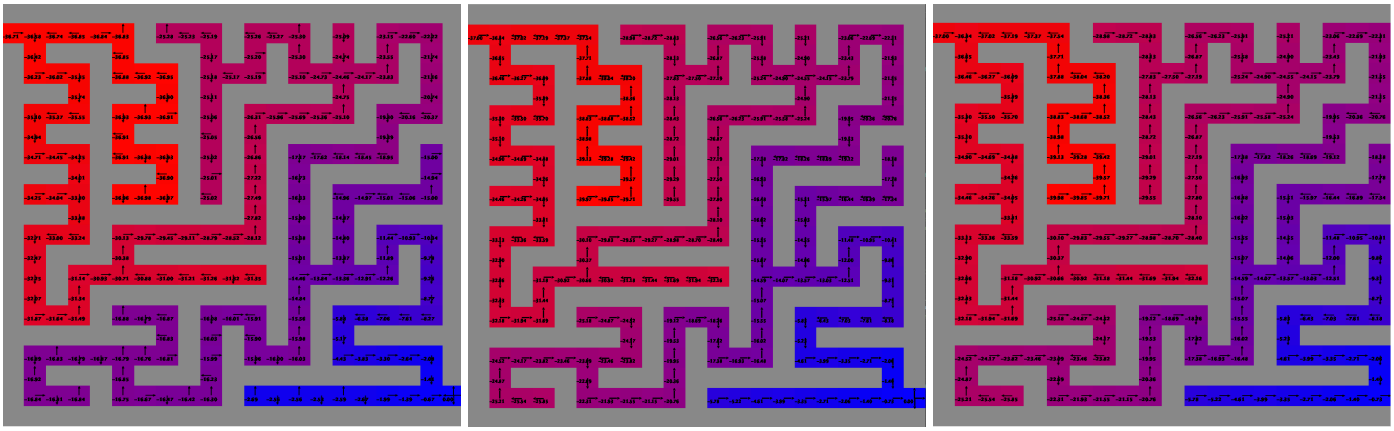


- The random strategy (dark green line) has the least variance compared to the other two cases. Because the random approach explores the domain all the time, it will converge faster and more compact.
- The random strategy also spent more time to learn because it needs heavier calculations to explore unknown states. In contrast, the greedy strategy always chooses the action with max Q-value, so it takes less time to compute.

2. Large-State Problem
   (1) Parameters: discount factor: **0.99**, max iteration: **1000**, initial state: **0**, epsilon: **0.5**, learning rate: **0.1**.
   (2) Reward: goal: **10**, other states: **-0.5**.



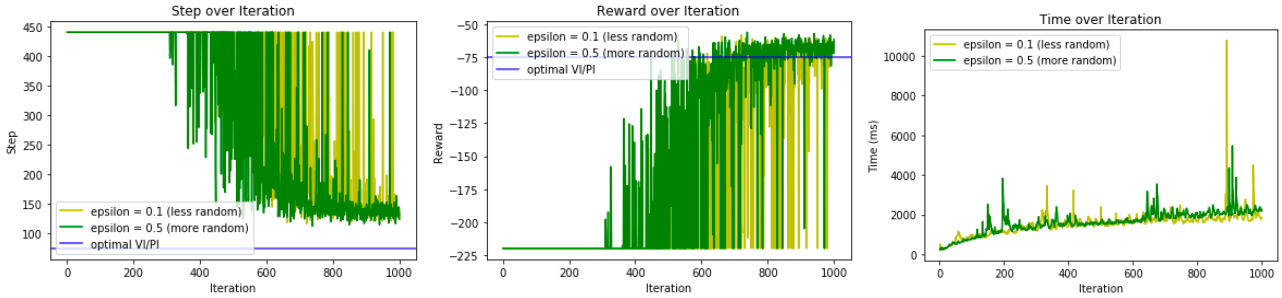
   (3) In the large-state problem, Q-learning needs around 700 iterations to converge, more than VI (120) and PI (8) for it doesn't know the model. Although it cannot find a path with steps as small as VI/PI does (left chart), it indeed reaches the optimal reward as well (middle chart).
   (4) The running time per iteration (~2,000) is larger than VI (~1,500) but still smaller than PI (~20,000). It shows that the running time of Q-learning is exponentially increasing as the number of states gets big.
   (5) The reason for reward fluctuation is similar as the small-state problem that the epsilon (randomness) isn't large enough (I will improve it in (7)). It might also because it is easy for the agent to get trapped in blind alleys that penalized the agent all the way. Therefore, it is much harder to converge than the small-state problem.
   (6) Compare the policy graphs of Q-learning with VI/PI:



Q-Learning Value Iteration Policy Iteration

We can see that the colors of three policy graphs are nearly the same. However, there are still few states with wrong actions in Q-Learning graph. It can be improved by running more iterations or use a larger epsilon value to explore more random states.

(7) Different Exploration Strategies ($\varepsilon = 0.5$ and $\varepsilon = 0.1$):



(a) The strategy with $\varepsilon = 0.5$ (dark green line) has smaller variance and more compact compared to $\varepsilon = 0.1$. Same reason as the small-state problem, the more randomness, the more likely to explore the whole domain and find the optimal Q-value.

(b) The higher ε takes more time to explore and learn than to the case with lower ε. But in this case, there is not a big difference of running time between $\varepsilon = 0.5$ and $\varepsilon = 0.1$. It might because 0.1 and 0.5 are still too close.

3. Conclusion

(1) Q-learning is a model-free learning algorithm, so it needs more iterations than VI/PI to converge. However, the running time for each iteration is much smaller than VI and PI because it only focuses on the current and the next state while VI and PI must update all states new utilities and policies.

(2) Increasing epsilon value gives Q-learning more randomness when choosing the next action. Higher epsilon will make Q-learning explore random states with higher probability and prevent trapped in local optimal Q-values. The result will converge with lower variance and more compact. However, it takes a lot more time to compute during exploration than greedy approach ($\varepsilon = 0$) that only needs to find the next state with max Q-value.

(3) Compared to VI/PI, Q-learning is more realistic because most problems in real life don't have a well-defined model and transition function. From the above analysis, Q-learning can achieve the optimal policy nearly as VI/PI do just with more iterations. Therefore, it is recommended to use Q-learning to solve real-life problems without models.

Reference

[1] Burlap. Retrieved from http://burlap.cs.brown.edu/

[2] John Stilley. Mazelib. Retrieved from https://github.com/theJollySin/mazelib

[3] Santiago Valdarrama. Markov Decision Processes. Retrieved from https://github.com/svpino/cs7641-assignment4