Report 4: Markov Decision Processes and Reinforcement Learning

Yiwei Yan

November 29, 2015

Abstract

The purpose of this report is to investigate and explore two different Markov Decision Processes (MDP) and learn from them to center in how agents work. We first introduce the algorithms that we utilize in this report, and then we analyze two finite MDP cases separately by using value iteration, policy iteration and Q-Learning strategies under both deterministic as well as non-deterministic conditions. Particularly, a flexible exploration strategy is designed in this work in order to simulate explorations with different greediness. One of the cases we selected is with a smaller amount of states, while the other one case is with a relatively larger amount of states.

1 Algorithms

In a wide range of situations, people need to make decisions that consider about not only the immediate benefits, but also the long-term rewards. It means that generally today's decisions will affect tomorrow's benefits, and tomorrow's decisions will impact the day after tomorrow's benefits, etc. Markov Decision Process (MDP) is such a mathematical framework that describe the decision making process in situations where outcomes are partly under the control of a decision maker and partly randomly depends on different conditions.

1.1 MDP

A MDP model consists of *States(S)*, *Actions(A)*, *Transition function(P)*, *Reward function(R)*, and *Discount factor(γ)*. At each time of step, the agent (decision maker) chooses any action a that is available in the start state s, and the process responds by randomly moving into a new state s' at the next time step, which gives the agent a corresponding reward $R_a(s, s')$. The major goal of MDPs is to find a policy π that can maximizes some cumulative function of random rewards. MDPs can be solved by linear programming or dynamic programming.

More precisely, suppose we know the state transition function P and the reward function R to calculate the policy π that maximizes the expected discounted reward. At the end, we can have $\pi(s)$ that contains the solution and $V(s)$ that contains the discounted sum of the rewards:

$$\pi^*(s) = \arg\max_a \sum_{s'} P_a(s, s')V(s') \tag{1}$$

$$V^*(s) = R(s) + \max_a \gamma \sum_{s'} P_{\pi(s)}(s, s')V(s') \tag{2}$$

1.2 Value Iteration and Policy Iteration

The essential idea behind value iteration is: the π function is not used; instead, the value of $\pi(s)$ is calculated within $V(s)$ whenever it is needed. We update $V(s)$ based on each s until convergence:

$$V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{\pi(s)}(s, s')V(s'). \tag{3}$$

For policy iteration, it starts with a random policy, compute each state's utility given that policy, and then select a new optimal policy. It updates $\pi(s)$ based on each s until convergence:

$$\pi(s) := \arg\max_{a \in A} \sum_{s'} P_a(s, s')V(s'). \tag{4}$$

1.3 Q-Learning

Value iteration and policy iteration work well when they assume that agents has all the domain knowledge (transition function and rewards). Unfortunately, the agents may not have access to these in many cases. Q-learning is a form of model-free learning. Q means the reward received immediately upon exacting a from s, plus the value (discounted) of the following the optimal policy thereafter.

$$Q(s, a) = R(s) + \gamma \sum_{s'} P_{\pi(s)}(s, s') \max_{q'} \hat{Q}(s', a') \tag{5}$$

Before learning start, Q returns an (arbitrary) fixed value. Then, each time the agent selects an action a, and observes a reward and a new state s that may depend on both the previous state and the selected action, Q is updated as follows:

$$\hat{Q}(s, a) \longleftarrow R + \gamma \max_{a'} \hat{Q}(s', a') \tag{6}$$

1.3.1 Exploration Strategies

Associated with the Q-learning procedure, we need to set up an exploration strategy. In this work, I designed a **flexible** exploration strategy to control how "greedy" the exploration is. Specifically, the exploration strategy is:

- **Input:** λ– greedy factor, Q– the Q matrix learned so far.
- **Output:** a_i- action choice for iteration i
- **Algorithm:** In iteration i, with the agent in state s_i:

  1. Compute

$$\zeta = \lambda * \left[1 - (\frac{1}{\log(n+2)}\right]; \tag{7}$$

2. Draw a uniform random variable

$$p \sim \texttt{Unif}(0, 1); \tag{8}$$

3. If $p < \zeta$, take the action a_i by taking the best choice from the Q learned so far, i.e.:

$$a_i = \arg\max_a Q(s_i, a); \tag{9}$$

Otherwise, i.e. $p \geq \zeta$, randomly select an available action as action a_i.

As we can see, the greedy factor λ controls how greedy the exploration is. By changing λ, we can simulate exploration ranging from **completely random exploration**(with $\lambda = 0$), to **completely greedy exploration** which always takes the previously learned best action (with $\lambda = \infty$). Another important property of the strategy is, the probability threshold ζ decays logarithmically, such that the strategy will converge to a greedy strategy eventually. This is reasonable, because with more iterations the Q learned will become more reliable and accurate. λ further affects the decaying speed of ζ.

2 Experiments

In order to investigate and evaluate MDPs, we designed two real-life scenarios.

2.1 Why are cases interesting?

Those two cases are particularly interesting because it implies that Markov decision process are existing everywhere around our lives. It has research value at different industries not only for robotics, but also for economics, finance, supply chain management, and even for people's daily buying decisions. Moreover, these cases are flexible and comprehensive enough to fully validate the algorithms. The cases covers deterministic and non-deterministic cases, small and large cases (in terms of number of states), etc.

2.2 Case 1: Career Plan

The first small case we introduce is about how to plan your career path starting from unemployment state to maximize your future rewards. In this case, we have four states that are Unemployed (S_0), To Industry(S_1), To Graduate School(S_2), and To Academia(S_3). Each states can have two choices of actions (a and b).

The goal of the first case is to solve MDPs with value iteration, policy iteration and Q-learning strategy, then observe the changes of total iteration times, values and convergence time with increasing gamma (γ) values, and compare the convergence time among value iteration, policy iteration and Q-learning strategy.

2.2.1 Case 1: Deterministic Version

We divide the first small case into deterministic version and non-deterministic version. Therefore, we can also watch the difference between Deterministic and Non-deterministic situations.

Table 1: Deterministic case: Career Path Selection. Each row illustrates the next state if the agent takes action a or b, as well as the associated rewards.

S_i	S_i'	**Rewards with** a	S_i'	**Rewards with** b
S_0 **Unemployed**	S_2	0	S_2	100
S_1 **Industry**	S_0	0	S_2	30
S_2 **Grad School**	S_3	50	S_2	0
S_3 **Academia**	S_3	30	S_2	-50

Table 2: Non-Deterministic case: Career Path Selection. Each row illustrates the probabilities of transiting to the next state if the agent takes action a or b, as well as the associated rewards.

S_i	S_i'**; Trans. prob.**	S_i'**; Trans. prob.**	**Reward from** a	S_i'**; Trans. prob.**	S_i'**; Trans. prob.**	**Reward from** b
S_0 **Unemployed**	S_2; 1	-	0	S_2; 1	-	100
S_1 **Industry**	S_0; 0.9	S_1; 0.1	0	S_2; 1	-	30
S_2 **Grad School**	S_3; 0.9	S_2; 0.1	50	S_1; 0.9	S_2; 0.1	0
S_3 **Academia**	S_3; 1	S_2; -	30	S_2; 0.9	S_3; 0.1	-50

Deterministic means that the trans probability $P(s, s')$ of states is always 100 %. Table 1 gives the information of the deterministic version of case 1. The results of total iteration times, values and convergence time changes are shown in Figure 1.

We can see from Figure 1, with the growing value of gamma from 0.1 to 0.9 with 0.1 interval, the iteration times of both value and policy iteration are increasing. Additionally, iteration times of value iteration are much larger than that of policy iteration, but the converge time of value iteration is generally less than that of policy iteration except for at gamma larger than 0.8, which means that value iteration is more efficient in time. In terms of the converge values ($V^*(s)$), they are almost overlapping as shown in the subfigure 3. It means that value iteration and policy iteration are almost converge to the same value at different gammas γ.

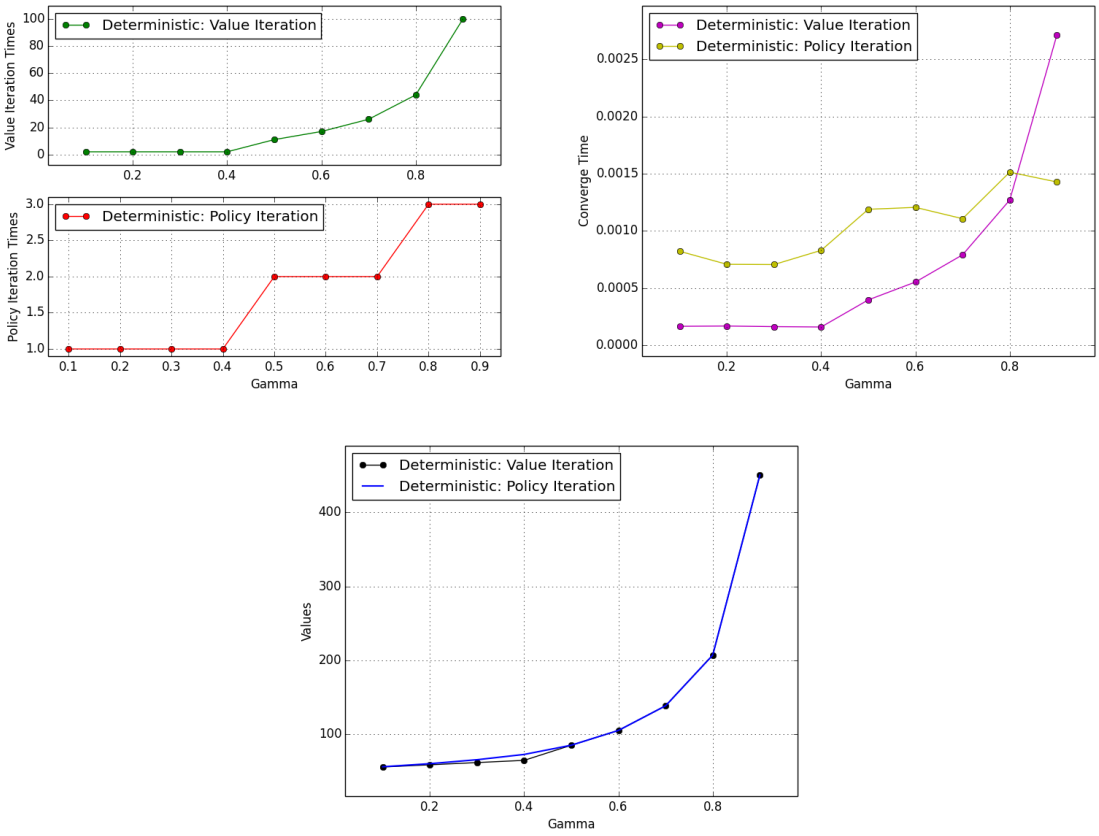
2.2.2 Case 1: Non-Deterministic Version

On the other hand, the non-deterministic version has the trans probability $P(s, s')$ of states is not always 100 %. We assume that in each selected action (a) from a starting state (s), there are two possibilities for which the next state will be.

Table 2 gives the information of the non-deterministic version of case 1, and its results are shown in Figure 2. It gives a very similar results with Deterministic version.

The iteration times of value and policy iteration are increasing with the growing gamma from 0.1 to 0.9 with 0.1 interval, the iteration times of value iteration is larger than that of policy iteration, the converge time of value iteration is less than that of policy iteration except for with gamma larger than 0.8, and they almost converge to the same value ($V^*(s)$) at each gamma γ.

Figure 1: Deterministic case: comparisons between Value Iteration and Policy Iteration, by changing the discount factor γ. Subfigure 1: total number of iterations to converge. Subfigure 2: run time (in seconds). Subfigure3: the value V computed by the algorithm.



2.2.3 Case 1: Q-Learning Strategies

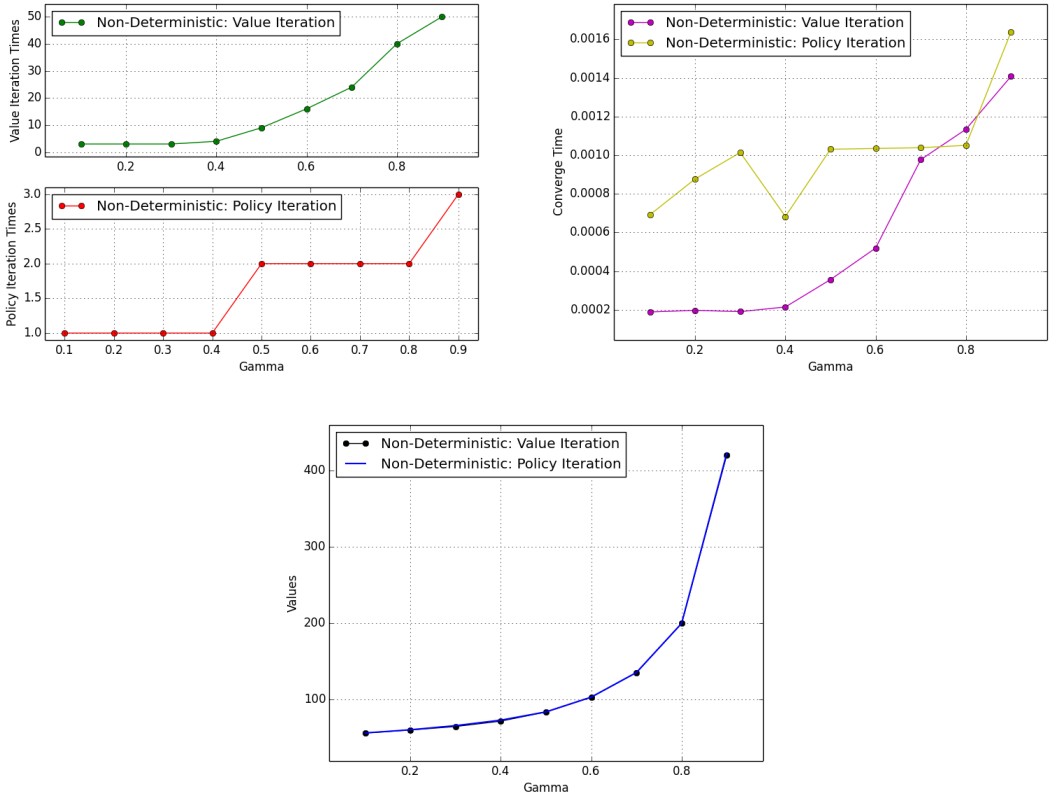
In the next step, we solve case 1 MDP using Q-Learning strategies by adjusting the "greedy" levels (Here we can easily generate different Q-Learning strategies by changing its greedy factors). We want to explore its iteration times, converge time and values, and then compare them with those from value iteration and policy iteration.

Figure 3 provides the results of Q-learning iteration times and converge values upon different greedy levels of the exploration strategy from 0 to 2 with 0.2 interval. The red dashed lines are the value computed from policy iteration and value iteration at $\gamma = 0.9$, we see them as the baseline of "Real Value". The subfigure 1 is for Deterministic version, and the subfigure 3 is for Non-deterministic version.

Before the greedy level 0.8, Q-learning strategies can approach converge values ($V^*(s)$) that very close to the real values baseline. But their performance become worse after greedy factors increase beyond 1.2. This indicates that values computed from lower gamma are more accurate and closer to the "real value" base line. This result directly reflects the importance of the exploration strategy.

For Deterministic example at the lower subfigure 1, its convergence time of Q-learning is wavelike decrease, while for Non-Deterministic example at the lower subfigure 2, the convergence time of Q-learning is wavelike increasing.

Figure 2: Non-Deterministic case: comparisons between Value Iteration and Policy Iteration, by changing the discount factor γ. Subfigure 1: total number of iterations to converge. Subfigure 2: run time (in seconds). Subfigure3: the value V computed by the algorithm.



In Figure 4, we can compare the converge time among Q-learning, value iteration and policy iteration. As the blue lines shown, Q-Learning strategies are generally much slower than that of value and policy iteration. It means that Q-Learning is a time-consuming methodology comparing with value and policy iterations.

2.3 Case 2: Machine Maintenance

Case 2 is a process about machine maintenance decisions with larger amount of states. In this case, we have in total 40 machines (Number of states), which is 10 times larger than that of case 1 (4 states). Since one of the key objectives of case 2 is to see the effects of changes states size, we also perform tests by changing the machine numbers (states numbers) from 10 to 100.

The problem statement is: in the machine maintenance process, there are machines for our factory to manage. Suppose that these machines can be in 40 descending levels of productivities/conditions (from newest to oldest). The newest machine can purely produce and make the highest benefit of 10 for the factory. The benefit that a machine can brings will decrease by a certain percentage based on the quality of maintenance.

There are 5 types of maintenance actions (Actions a) for each machine to choose from, and every type of maintenance has different labor costs and resource costs. Those actions are Action 0 (no maintenance), Action 1 (level-1 maintenance), Action 2 (level-2 maintenance),

Figure 3: Results of Q-Learning algorithm: the computed value V and the convergence time, by changing the greedy factor λ of the exploration strategy. Subfigure 1: results from the deterministic case. Subfigure 2: results from the non-deterministic case.

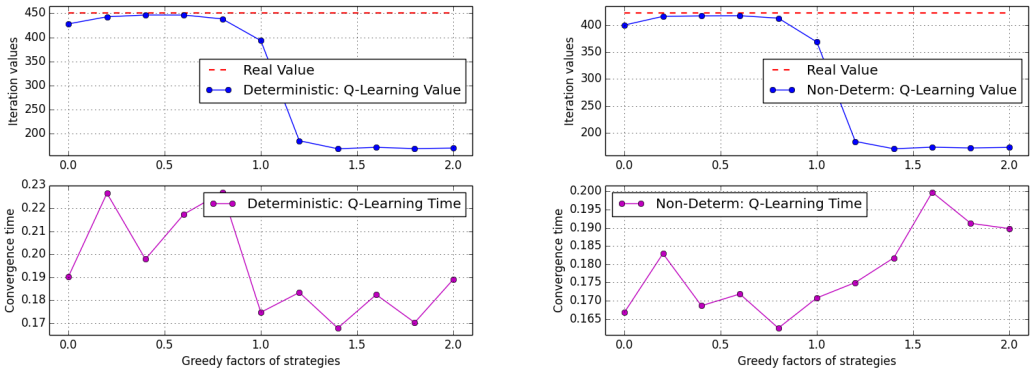
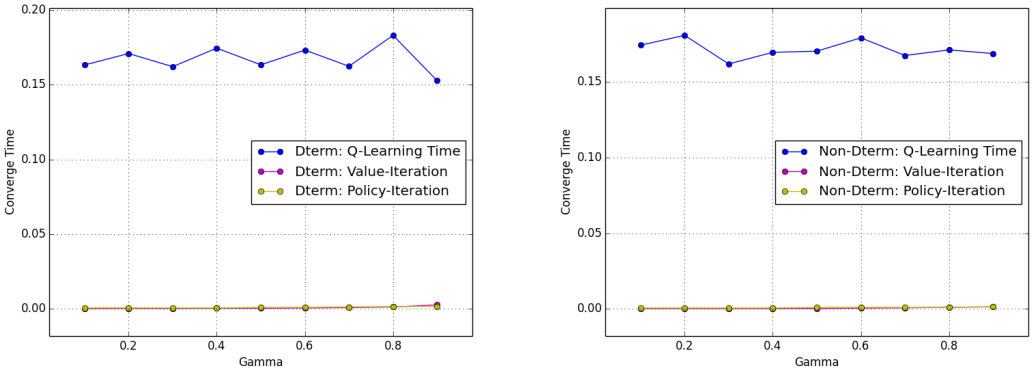


Figure 4: Convergence Time Comparison Among Q-Learning, Value Iteration, Policy Iteration



Action 3 (level-3 maintenance), and Action 4 (the best-level maintenance).

Without maintenance, there is a 90% chance that a machine will depreciated to the next level of condition (s'). With the best maintenance, the cost will be the highest but then the machines will have the highest percent to remain at their current conditions rather than depreciated to the next condition levels. In general, the maintenance qualities will directly affect the depreciation speed, total cost, and total values of those machines.

Below we give an example of our States and Rewards matrices of a small example with only 4 machine status:

Transition probabilities of machines status: from Action 0 to Action 2

$$P(S,a_0)=\begin{bmatrix}0.1 & 0.9 & 0.0 & 0.0\\ 0.0 & 0.1 & 0.9 & 0.0\\ 0.0 & 0.0 & 0.1 & 0.9\\ 0.0 & 0.0 & 0.0 & 1.0\end{bmatrix};P(S,a_1)=\begin{bmatrix}0.4 & 0.6 & 0.0 & 0.0\\ 0.3 & 0.2 & 0.5 & 0.0\\ 0.09 & 0.21 & 0.2 & 0.5\\ 0.0 & 0.09 & 0.21 & 0.7\end{bmatrix};P(S,a_2)=\begin{bmatrix}0.5 & 0.5 & 0.0 & 0.0\\ 0.3 & 0.3 & 0.4 & 0.0\\ 0.12 & 0.18 & 0.3 & 0.4\\ 0.0 & 0.12 & 0.18 & 0.7\end{bmatrix};$$

$$P(S,a_3)=\begin{bmatrix}0.6 & 0.4 & 0.0 & 0.0\\ 0.3 & 0.4 & 0.3 & 0.0\\ 0.12 & 0.18 & 0.4 & 0.3\\ 0.0 & 0.12 & 0.18 & 0.7\end{bmatrix};\quad P(S,a_4)=\begin{bmatrix}1.0 & 0.0 & 0.0 & 0.0\\ 0.6 & 0.4 & 0.0 & 0.0\\ 0.24 & 0.36 & 0.4 & 0.0\\ 0.0 & 0.24 & 0.36 & 0.4\end{bmatrix}$$

The rewards matrix of 5 actions associated with the 4 states (each column corresponds to the reward of one action) is

$$R(S,A) = \begin{bmatrix} 10.0 & 8.0 & 6.0 & 3.0 & -1.0 \\ 9.3 & 7.4 & 5.6 & 2.8 & -1.1 \\ 8.6 & 6.8 & 5.2 & 2.6 & -1.2 \\ 7.9 & 6.2 & 4.8 & 2.4 & -1.3 \end{bmatrix}.$$

2.3.1 Case 2: Value Iteration and Policy Iteration

We set the number of machines (states) to be 40, which is 10 times more states than that in case 1. Figure 5 shows the results of iteration times, converge time and converge values for value iteration and policy iteration. Similarly, we see that with the increasing of gamma from 0.1 to 0.9 in 0.1 interval, the iteration times are rising. Additionally, iteration times of value iteration are larger than that of policy iteration, but its converge time is less than that of policy iteration except for gammas larger than 0.8. This intimate that value iteration is more efficient on time.

The converge values ($V^*(s)$) are overlapping as shown in the second graph of Figure 5. It means that value iteration and policy iteration are almost converge to the same value at different gammas γ. Besides, the converge values increase a little and then become smaller after gamma 0.5. It achieves the lowest converge value when $\gamma = 0.8$, and it reaches the highest converge value when $\gamma = 1$.

2.3.2 Case 2: Impact of States Number Change

In terms of explore that if the number of states affect the results from value iteration and policy iteration at all, we change the machine numbers from 10 to 100 and plot the changes of iteration times and convergence time for both value iteration and policy iteration.

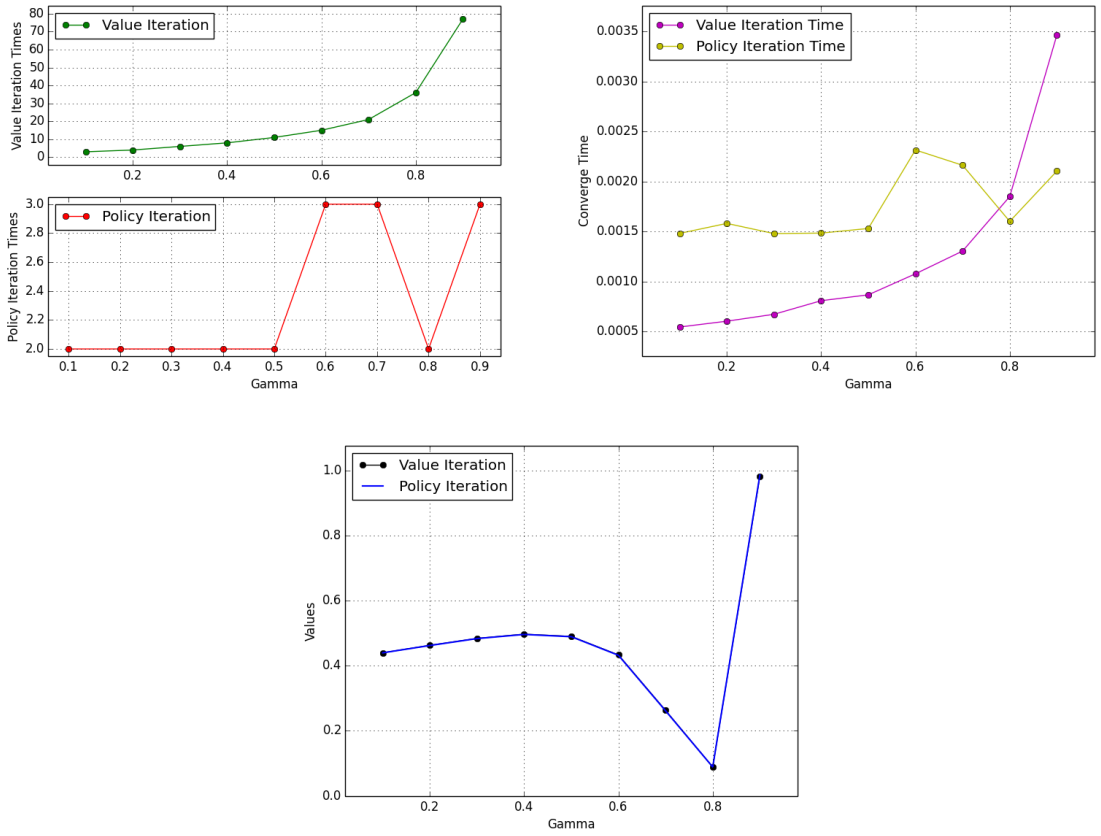
As shown in Figure 6, when we keep adding states into the MDP process, we can see from the subfigure 1 that iteration times for value iteration increase, while the iteration times for policy iteration remains quite stable except for a sudden change when $S = 14$. This means that the number of states change has larger impact on the iteration times of value iteration than that of policy iteration. Moreover, increasing number of states also cause the increasing converge time for both value iteration and policy iteration.

2.3.3 Case 2: Q-Learning Strategies

The same as we did for case 1, we adjust the greedy factor of Q-learning to generate different Q-Learning strategies. We want to explore the iteration times, converge time and values when solving case 2 utilizing Q-Learning strategies, and compare them with those from value iteration and policy iteration. Figure 6 provides the results on different greedy learning levels from 0 to 2 in 0.2 interval.

In terms of the converge values, the red dashed line at the upper subfigure 1 is the value computed from policy iteration and value iteration at $\gamma = 0.9$ with the global information (transition matrix and reward matrix) known *a-priori*. They can be seen as the "groundtruth Real Value". The values computed from Q-learning strategies that with different greedy levels are in a decreasing trend, which indicates that values computed from lower gamma

Figure 5: Case 2: comparisons between Value Iteration and Policy Iteration, by changing the discount factor γ. Subfigure 1: total number of iterations to converge. Subfigure 2: run time (in seconds). Subfigure3: the value V computed by the algorithm.



are more accurate and closer to the "real value" base line. For the convergence time of Q-learning strategies, it fluctuates in a decreasing trend when the greedy factors grow. The reason why the discrepancy between the Q-Learning value and the real value increases when the greedy factor increases, is that when the exploration strategy is not so greedy and more random, the agent is more likely to explore different state-action pairs to gather more reward information. This is particularly important when the number of states is large. The trade-off is, if the exploration is less greedy, the convergence rate will be smaller and the running time is longer.

As shown on the second graph of Figure 7, when compare the converge time among Q-learning strategy, Value Iteration, and Policy Iteration. Q-Learning strategies are much slower than that of value iteration and policy iteration. It means that Q-Learning is much more time-consuming and less efficient. This makes sense, because in Q-Learning the agent does not know the global information like the transition matrix and reward matrix. In stead, the algorithm needs to learn and infer the information, which is much more computational expensive than simply solving the Value Iteration or Policy Iteration.

Figure 6: Case2: impact of number of states. Subfigure 1: number of iterations. Subfigure 2: total runtime (in seconds).

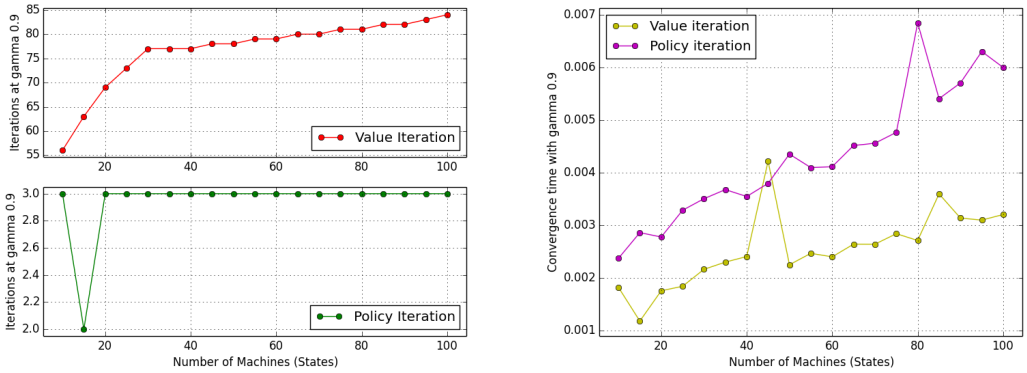
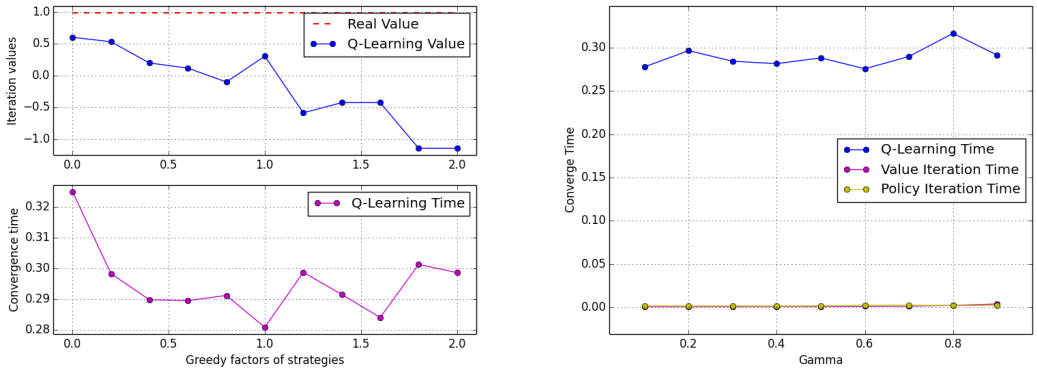


Figure 7: Case 2: validation of Q-Learning exploration strategies (Subfigure 1), and runtime comparison among Q-learning, Value Iteration, and Policy Iteration (Subfigure 2).



3 Summary

To sum up, in this report, we solve two MDPs cases using value iteration, policy iteration, and Q-Learning algorithms. For Q-Learning, we employ and validate different exploration strategies under different greedy levels. From the experiments, we observe that:

- Value iteration is usually faster to converge than policy iteration, and they can converge to the most same values $V^*(s)$.
- Value iteration needs more iterations in each period than policy iteration (at least in our experiments).
- Number of states do have effects on converge time and number of iterations needed.
- Q-Learning algorithm is generally slower and less accurate than value iteration and/or policy iteration on convergence, but it does not require *a priori* global information and thus is more realistic in real applications.
- The exploration strategy of Q-Learning plays an important role in the performance. The less greedy the exploration is, the more accurate the solution is, but in trade-off the convergence rate is smaller and the running time becomes longer.