GEORGIA INSTITUTE OF TECHNOLOGY

Title: Machine Learning Homework4 – Markov Decision Processes

Author(s):
Weifeng Lyu

Instructor:
Prof. Charles Isbell,
Prof. Michael Littman

TABLE OF CONTENTS

1. Introduction

1.1 Problem

This project implements three different reinforcement learning algorithms, they value iteration, policy iteration and Q-Learning, then apply these algorithms to solve two interesting Markov Decision Processes.

1.2 Computer Configuration

Manufacturer: Dell, Model: Inspiron 7559 Signature Edition, Processor: Intel® Core™ i7-6700HQ CPU @ 2.60GHz, Installed Memory (RAM): 8.00GB (7.88 usable), System Type: 64-bit Operating System, x64-based processor

2. Dataset

Two interesting Markov Decision Processes are defined as grid worlds. The grid world is developed in Java Application Class inherited from Simulated Environment from burlap library. The agent tries to reach the terminal state from one step to the other through each grid. Through each grid step, the agent receives the reward function except hitting the wall. The transition matrix will be computed for the probability that the agent moves to next grid.
We design the grid world problem as an interesting experiment because the following reasons:
1. Many real-world applications such as maze, navigation, autonomous vehicle and even playing chess can be described as Grid World problems as Markov Decision Processes.
2. Each sequential step except the first move depends on previous moves. In real-life, we always complete tasks steps by steps. The better we make the choice from last actions, the easier the next move will be.
3. All the moves should be designated for the shortest and fastest in order to minimize the intermediate negative reward. Any returning effort will cost both time and effort that we should avoid in real-life.
4. There will be always unexpected obstacles and accidents along the path.
5. Many real-world problems are deterministic environment like the grid world, which means that both state transition model and reward model are deterministic functions.

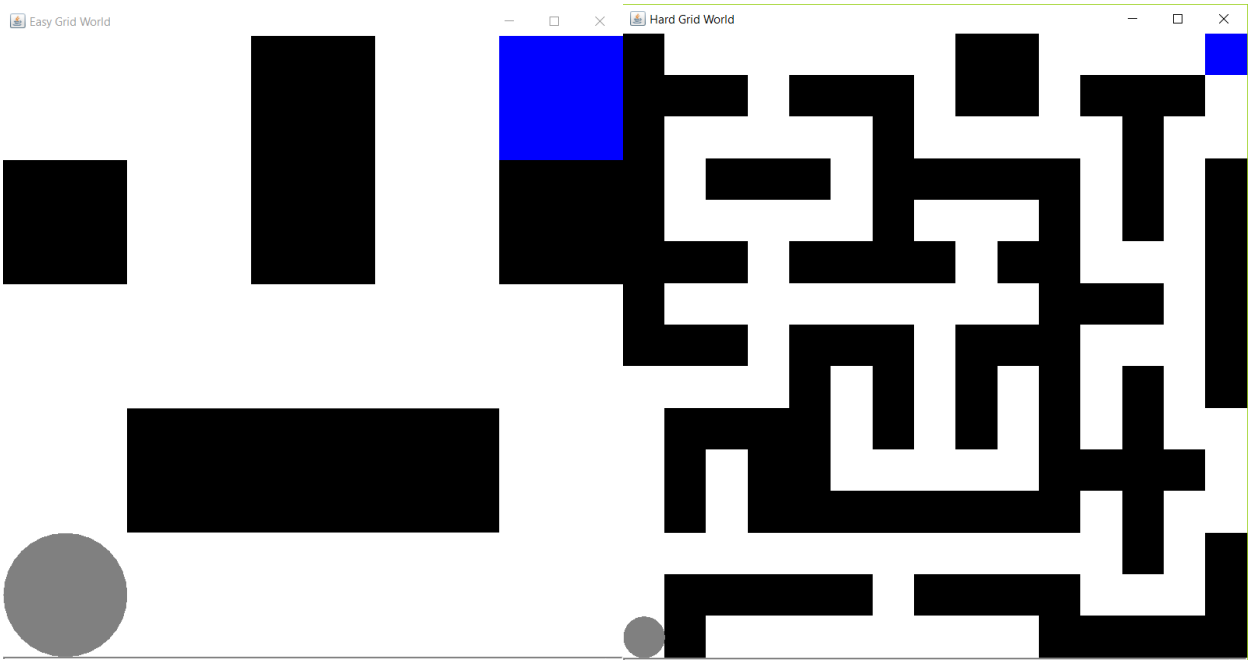


Figure1. Easy Grid World (left) and Hard Grid World (right)

2.1. Easy Grid World

The easy grid world for this problem is constructed by 5*5 grids. There exists one shortest path and several paths from starting point to the end as observed. The agent aims to find the shortest path to reach the target.

2.2 Hard Grid World

The hard grid world for this problem is constructed by 15*15 grids. There exist multiple alternative paths from starting point to the end as observed, the graph is constructed more like a maze. The agent aims to avoid the obstacles and find the shortest path to reach the target.

3. Project Architecture

3.1. Java Implementation

This project implements in Java Eclipse, we alternate the iterations for each experiment to analyze different effects and plot the graphs in Python 3.6's matplotlib using the data from Java implementation. The following parameters will be set for our experiment:

1. We assign -1 reward for each state except the obstacles and the terminal state, which means every step the agent performs will result 1 penalty.
2. If the agent gets to the terminal state with the smallest number of actions dictated by the reward function, once the ultimate state is reached, the agent will be rewarded 100.
3. The probability set for moving to the next grid is 0.8, and (1-0.8)/3=0.0667 for moving to the other 3 directions besides the target.

The application includes two main components defining the Grid World described below: EasyGridWorldLauncher.java and HardGridWorldLauncher.java. More importantly, there exists other utilities for the user to analyze the agent activities:
AgentPainter.java, MapPrinter.java, LocationPainter.java and WallPainter.java constructs the background and styles for the agent, maze. AnalysisAggregator.java gathers all the values from reward functions and perform computations. AnalysisRunner.java is the application that runs the actual algorithms such as Policy Iteration, Value Iteration, Q-Learning. AtLocation.java, BasicRewardFunction.java, BasicTerminalFunction.java control the location at which the agent is on the grid world, assigns the appropriate reward for the agent. Movement.java manipulates the action performed by the agent.

3.2. Algorithm Review, Project Demonstration and Sample Result

The value function represents how good is a state for an agent to be in. It is equal to expected total reward for an agent starting from state s. The value function depends on the policy by which the agent picks actions to perform. So, if the agent uses a given policy $\boldsymbol{\pi}$ to select actions, the corresponding value function is given by:

$$\mathrm{V}^{\pi}(s) = E\left[\sum_{i=1}^{T} \boldsymbol{\gamma}^{\boldsymbol{i-1}} r_i\right] \forall s \in \mathrm{S}$$

For all possible value functions, there exists an optimal value function that has highest value for all states, and the optimal policy $\boldsymbol{\pi}^*$ is the policy that corresponds to optimal value function.

$$\mathrm{V}^{*}(s) = max V^{\pi}(s)\ \forall s \in \mathrm{S}$$
$$\pi^{*}(\mathrm{s}) = \arg max V^{\pi}(s)\ \forall s \in \mathrm{S}$$

The relationship between Q*(s, a) and V*(s) is easily defined by the following equations, V*(s) is the maximum expected total reward when starting from state s , it will be the maximum of Q*(s, a)over all possible actions:

$$V^*(s) = maxQ^*(s,a)\ \forall s \in S$$
$$\pi^*(s) = \arg maxQ^*(s,a)\ \forall s \in S$$

Value Iteration and Policy Iteration relies on the following equations to compute the optimal values:

$$Q^*(s,a) = R(s,a) + \gamma E_{s'}[V^*(s')]$$
$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in 0}^{n} p(s'|s,a)V^*(s')$$

Their values of V are computed by the following equations:

$$V^*(s) = maxQ^*(s,a)$$
$$V^*(s) = maxQ^*(s,a)(s,a) + \gamma \sum_{s' \in 0}^{n} p(s'|s,a)V^*(s')$$

Q-Learning approximates the state-action pairs Q-function from the samples of Q(s, a) that we observe during interaction with the environment. This approach is known as Time-Difference Learning.

$$Q(s,a) = (1-a)Q(s,a) + aQ_{obs}(s,a)$$
$$Q_{obs}(s,a) = r(s,a) + \gamma maxQ(s',a')$$

The optimal Q-function Q*(s, a) means the expected total reward received by an agent starting in sand picks action a, then will behave optimally afterwards. There, Q*(s, a) is an indication for how good it is for an agent to pick action a while being in state s.
Value Iteration uses Bellman equation to evaluate the states till there reaches convergence. At the beginning, the unity is not known other than the immediate reward. Value iteration computes the optimal state value function by iteratively improving the estimate of V(s). The algorithm initializes V(s) to arbitrary random values. It repeatedly updates the Q(s, a) and V(s) values until they converges. Value iteration is guaranteed to converge to the optimal values. The algorithm can be described as the following pseudo-code referenced from Alpaydin Introduction to Machine Learning, 3rd edition:

```
Initialize V(s) to arbitrary values
Repeat
    For all s ∈ S
        For all a ∈ A
            Q(s,a) ← E[r|s,a] + γ Σ_{s'∈S} P(s'|s,a)V(s')
        V(s) ← max_a Q(s,a)
Until V(s) converge
```

Policy Iteration is an algorithm that creates arbitrary policies, with each policy, the algorithm will calculate the utility value. The algorithm tries different policies and test if the new policy can increate the

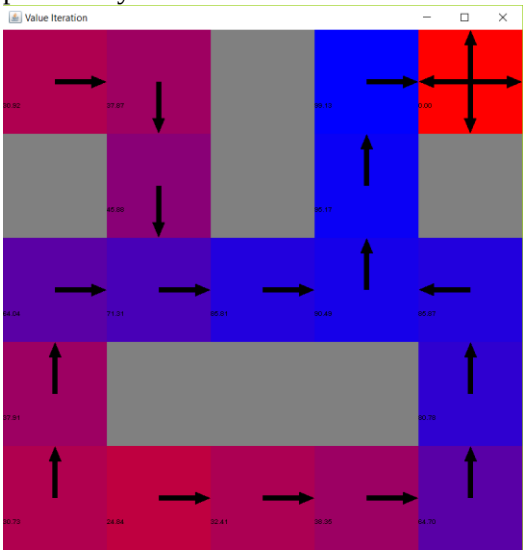
utility value. If there no longer exists any improvement for the utility value. The method converges, then it will re-define the policy at each step and compute the value according to this new policy until the policy converges. The algorithm can be described as the following pseudo-code referenced from Alpaydin Introduction to Machine Learning, 3rd edition:

Initialize a policy π' arbitrarily
Repeat
    $\pi \leftarrow \pi'$
    Compute the values using π by
        solving the linear equations
            $V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^{\pi}(s')$
    Improve the policy at each state
        $\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a) V^{\pi}(s'))$
Until $\pi = \pi'$

Q-Learning is a model-free algorithm. the agent will not try to learn explicit models of the environment state transition and reward functions. However, it directly derives an optimal policy from the interactions with the environment. More specifically, Q-Learning assigns q values to all states, then the agent visits each state and assigns new q values based on immediate and delayed rewards. In addition, there is a model-based Q-Learning that the agent will interact to the environment and from the history of its interactions, the agent will try to approximate the environment state transition and reward models. Afterwards, given the models it learns, the agent can use value-iteration or policy-iteration to find an optimal policy, ultimately q value will converge to its true values for each state.

(1). Easy Grid World with shortest path found

Easy Grid World with Iteration = 5:
We start with running 5 and 100 iterations for Easy Grid World. We can see on the figure.2 that value iteration finds an optimal policy. From the grid world described, the agent tends to find the path with dark blue, avoid red. Since every iteration, it will compute the probability of step taking in every grid, the shortest path from beginning to end would be most blue path with least possible red. The values for probability are shown in left-bottom corner for each grid.



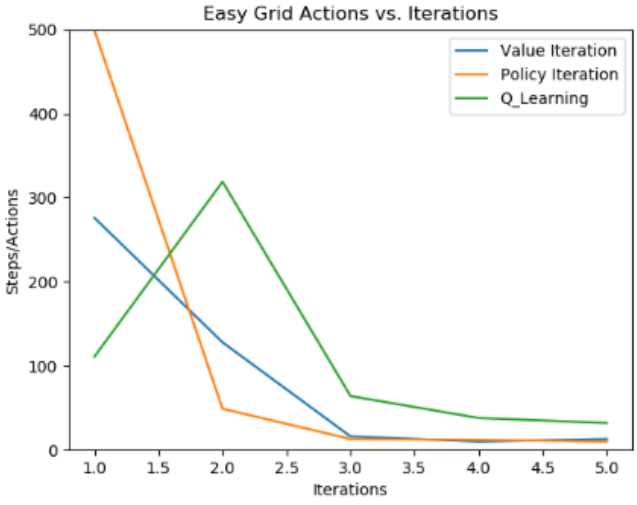


Figure2. Value Iteration paths (left) and Actions Vs. Iterations for Easy Grid World (right)

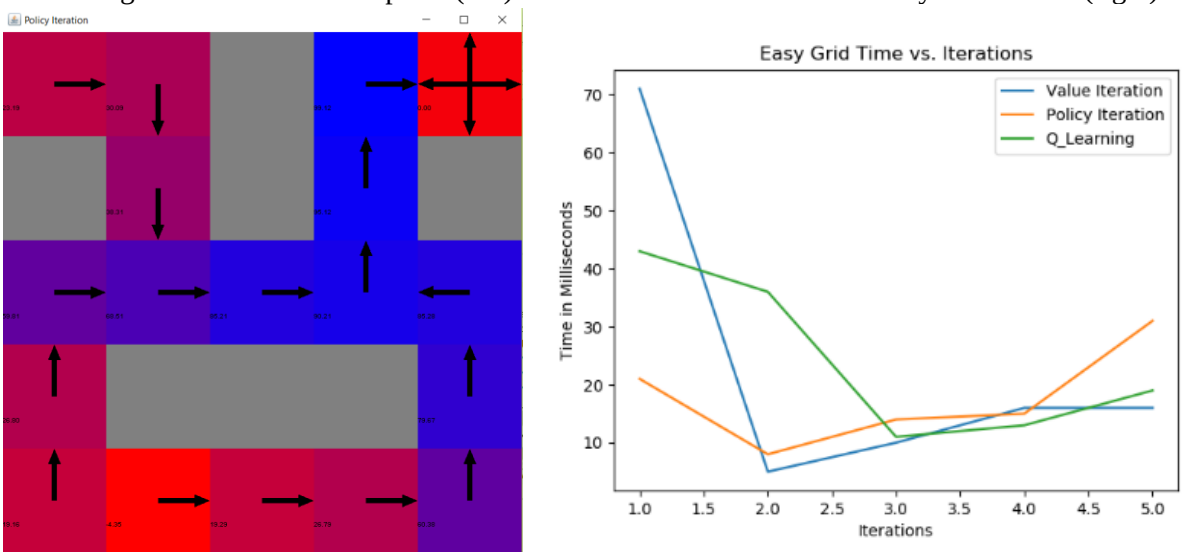


Figure3. Policy Iteration paths (left) and Time in Milliseconds Vs. Iterations for Easy Grid World (right)

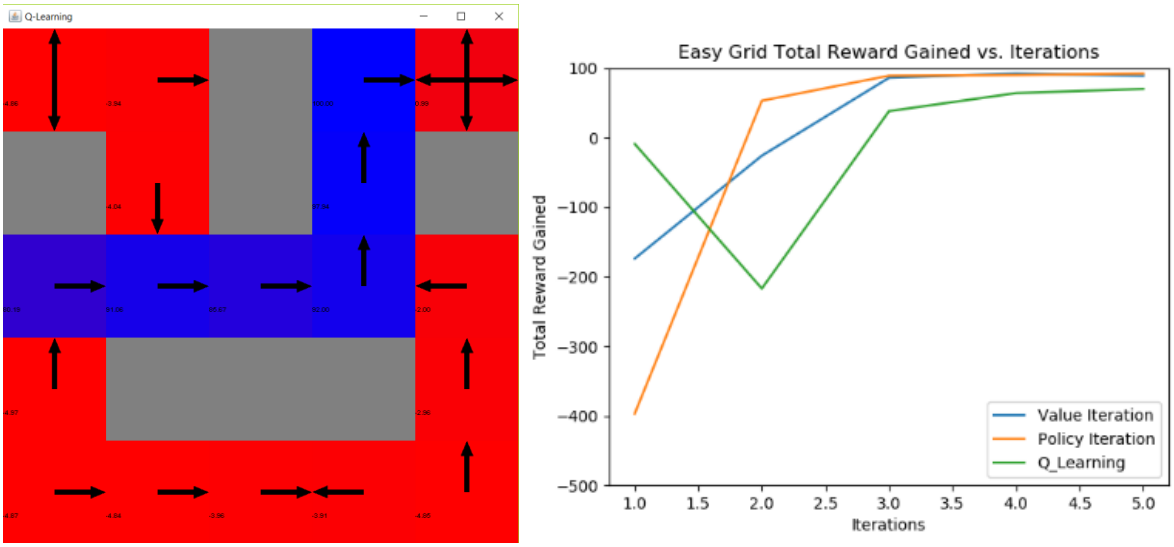


Figure4. Q-Learning paths (left) and Total Reward Gained Vs. Iterations for Easy Grid World (right)

Firstly, we run very low iterations to see how quick the algorithm can converge. From figure3 and figure4 for the above group of graphs depicts iterations=5 for value iterations, policy iterations, they quickly converge to 90+ approximately in 3 iterations, Q-Learning also converges quickly with one alternation between 1-3 and the reward stays about 90+. The time used for completing the first iterations normally is highest for policy iterations and value iterations, between them policy iterations is higher. for value iterations and policy iteration, but for this case even the first iteration computational time is minimal. There is another interesting fact that Q-Learning in such a lower iteration can't seem to find a solid shortest path because the agent only knows current states and actions, and the agent has to actively learn through limited experience of interactions with the environment.

Easy Grid World with Iteration = 100:

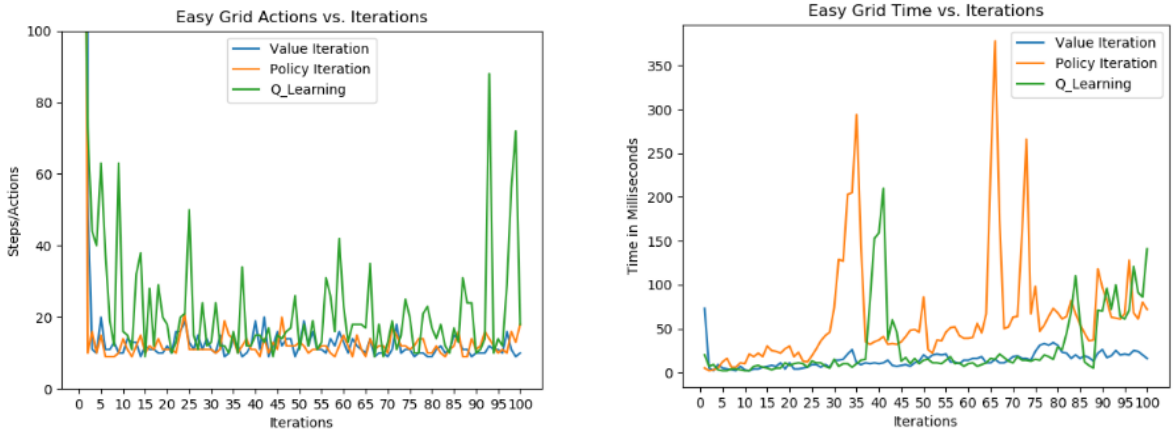


Figure5. Actions Vs. Iterations for Easy Grid World (left) and Time in Milliseconds Vs. Iterations for Easy Grid World (right)

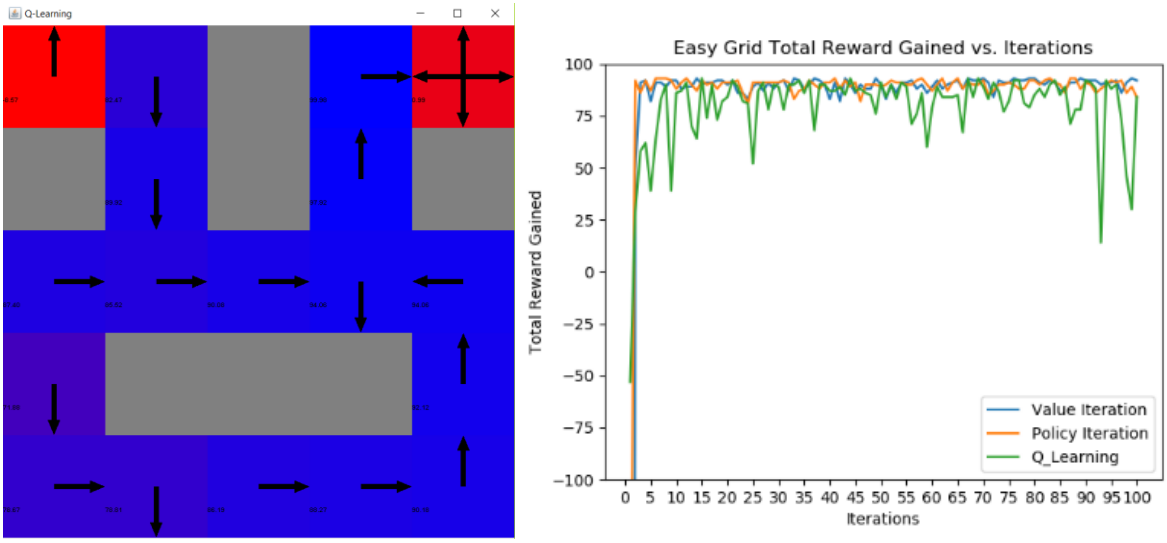


Figure6. Q-Learning paths and Total Reward Gained Vs. Iterations for Easy Grid World (right)

The last example with only 5 iterations is just a glance, can't establish any solid conclusion, we increase the iterations to 100 now we can see its long-term effect.

From the observation for the grid world, we don't find significant difference compared to 5 iterations in this case for value iterations and policy iterations. And their implementing actions are quite same to each other and show solid blue paths, while Q-Learning cannot find the shortest path in 5 iterations but is able to determine the shortest path in 100 iterations, it did not learn good experience to run the actions in short experience.

From figure5 and figure6 plots for required actions and rewards, we find that first 5 value iterations and policy iterations are quite consistent with the last example with 5 iterations case. This could conclude that both value iterations and policy iterations converge after three iterations, so they have found the solid shortest path with convergence about 90 reward. In contrast, Q-Learning decreases a lot slower even at smaller convergence start. Since Q-Learning does not know the model, it's proven that it expects longer time to converge. To see if it is really converging, we increase the iteration to 1000 but can still see huge alternation in higher iteration number. Therefore, it is not worthwhile to increase iterations to achieve the convergence. Q-Learning tends to learn slower and perform worse in this easy problem.

In addition, regarding computational time perspective, in this easy grid world, value iteration performs much better than Q-Learning and policy iteration in long-term. However, policy iteration will somehow

need unexpected longer time to complete an iteration after convergence. Policy iteration is different from value iteration, which searches directly for the optimal policy while value iteration searches for the optimal value then extract the optimal policy. Overall, before the convergence, policy iteration performs better in this Easy Grid World case. Q-Learning sporadically needs more time in several iterations because it applies random actions, drop the rewards. Therefore, regarding to time-performance and accuracy convergence performance, Q-Learning performs the worse in this Easy Grid world, and value iteration performs the best.
Based on the data results generated from three algorithms with 100 iterations, the maximum reward is 93 and deduct 100, it should yield -7 reward for each step for the best policy, and the minimum action is 9, so the shortest path length is 9-1=8.

(2). Hard Grid World with obstacles avoided

Hard Grid World with Iterations = 250:
We run 250 and 1000 iterations for Hard Grid World.

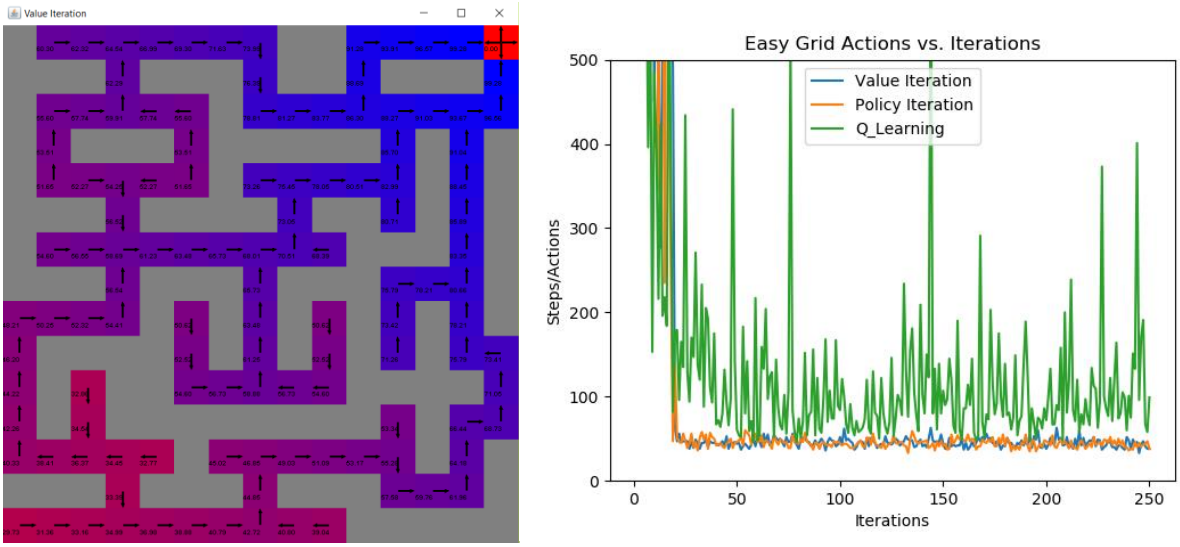


Figure7. Value Iteration paths (left) and Actions Vs. Iterations for Hard Grid World (right)

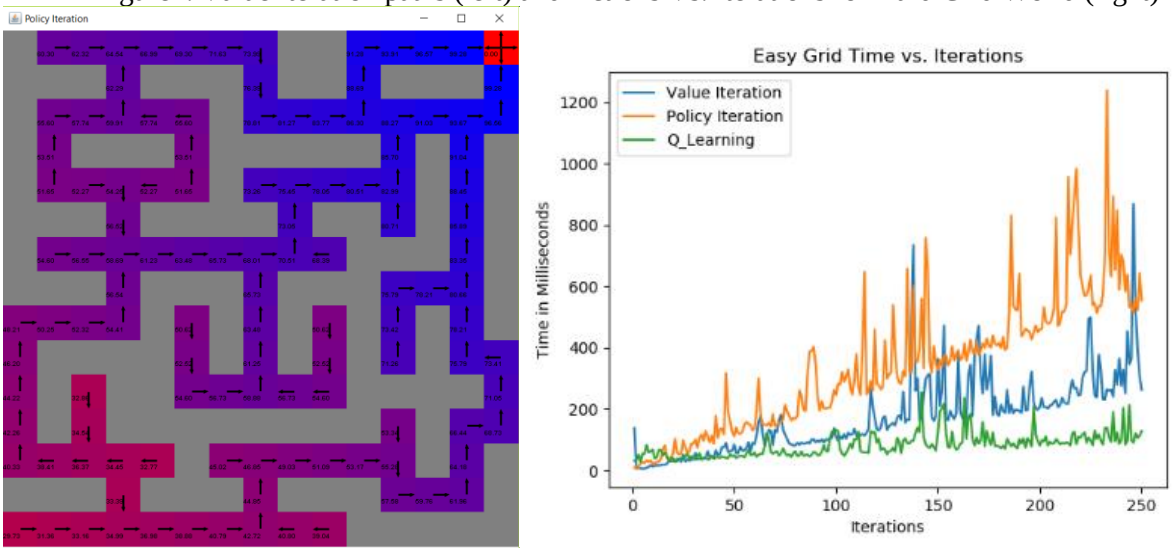


Figure8. Policy Iteration paths (left) and Time in Milliseconds Vs. Iterations for Hard Grid World (right)

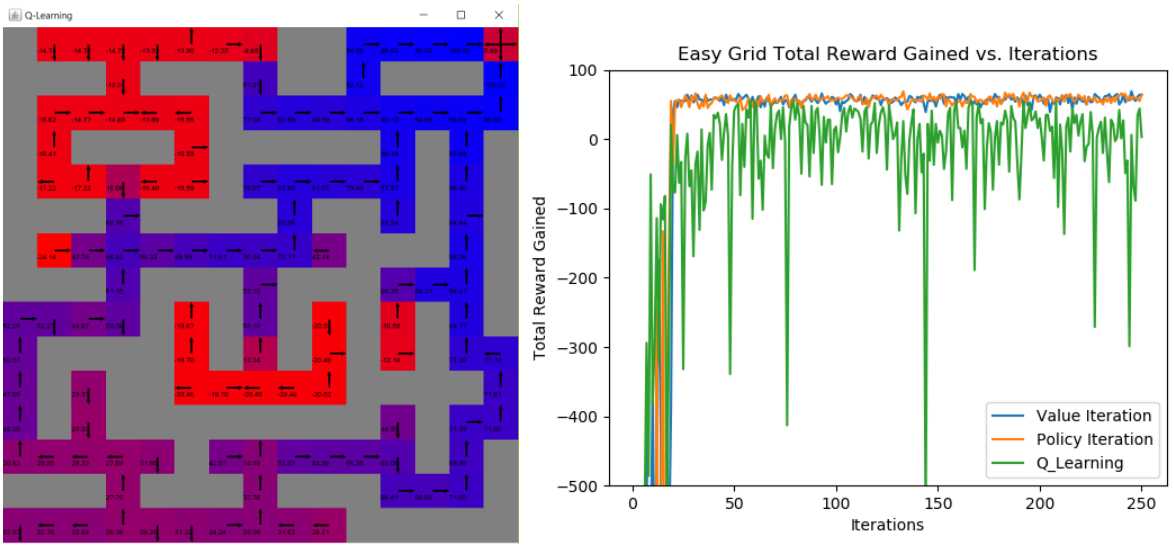


Figure9. Q-Learning paths (left) and Total Reward Gained Vs. Iterations for Hard Grid World (right)

We can see the grid world that both value iteration and policy iteration seem to converge in about 20+ steps and yields the actions of 30+, and the rewards converge to 60+, they seem to behave similarly with Easy Grid World environment setup.
There are some outstanding spikes for value iterations and policy iterations from the beginning of iterations, it will expect longer time to complete in the first few iterations, but quickly it converges a well-defined path better than Q-Learning while Q-Learning still oscillates in later iterations.
Besides of that, they start requiring more and more time to complete in later iterations, this implies that neither value iterations nor policy iterations is suitable to learn this complicated model well. In this case, Q-Learning start performing better in computational time, although it generates a lot of outstanding spikes for rewards and actions in the graph with iterations increased, which means that it applies many random and bad actions may hurt the reward of that iteration dramatically. as it's model-free algorithm so it learns from its previous experience without relying on reward function. After the convergence of value iterations and policy iteration, Q-Learning is relatively quicker to handle each iteration in such a complicated maze. Now that we think 250 iterations is enough for Q-Learning to construct the best actions and rewards. Therefore, from the maze, we can see that an accurate solution is not a significant problem to consider, all 3 algorithms can converge within 100 iterations. We should attach more importance on saving time for computation.

Hard Grid World with Iteration=1000:
When the iteration increases to 1000, we can't see significant optimization for the grid world path.

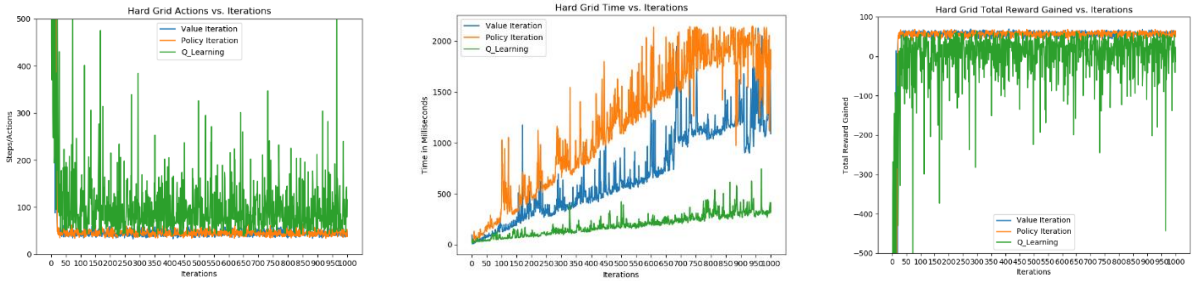


Figure10. Actions Vs. Iterations for Hard Grid World (left) and Time in Milliseconds Vs. Iterations for Hard Grid World (middle) and Total Reward Gained Vs. Iterations for Hard Grid World (right)

From figure10, we can see it follows 250 iterations' pattern, both value and policy iteration achieve convergence very fast, while Q-Learning seems to converge within 100 iterations but keep alternating in very large spike. However, Q-Learning performs better in long-term regarding time-saving.

In this experiment, based on three algorithms above, we can find that the minimum actions needed is 33 for escaping from 15*15 Grid World computed by value iteration and policy iteration, 34 computed by Q-Learning. The values computed by value iteration and policy iteration are exactly the shortest path, while 34 is acceptable. However, with 250 iterations, Q-Learning only gets to 46 actions for the best action while value iteration and policy iteration obtain the best. The best rewards for value iteration and policy iteration are 69-100=-31, Q-Learning yields the best reward of 56-100=-44 for 250 iterations and 64-100=-36 for 1000 iterations. From the observations, we can conclude that the penalty for wrong direction should be set high for this maze problem because sometimes wrong decision will have the long-term negative influence.

4. Conclusions, Discussion and Improvement

In this assignment, we explore three algorithms of reinforcement learning and applied it into Markov Decision Processes. Both Markov Decision Processes shows that value iteration and policy iteration can obtain faster convergence than Q-Learning. However, Q-Learning runs at a constant time while value iteration and policy iteration consume much more time in later iterations and policy iteration takes even much more. Therefore, to improve the model, we should find the relationship between the complexity and the iteration, then use those parameters to run the experiments. To resolve the optimal-required problems, we should let policy iteration and value iteration achieve the optimal value and policy. It is evident that with higher number of states, Q-Learning takes an exponential increase of iterations to converge, so we should not use Q-Learning to achieve the optimum. If the model allows non-optimal solution and time-sensitive, Q-Learning would be the best algorithm to implement. Q-Learning as model-free algorithm, can be described as long-term investment, although it does not have extraordinary effect in short-term, it will achieve greater and greater in long-term learning.
Besides of reinforcement learning, this assignment enhances the Java programming experience for me by exploring Java, Eclipse and Burlap. Not many data manipulation in Python is required, but there are more solid understandings to the context developed for result analysis. Reinforcement Learning arouses my interest and it behaves more like human nature mindset, which inspires me to study those unexplored topics for reinforcement learning in the future.