# Markov decision process and Reinforcement Learning

An analysis of Value Iteration, Policy Iteration and Q-learning algorithm and application on solving architectural design problems

Zhengyang Chen (zchen612 GTID#903338861)
Georgia Institute of Technology

**Keywords**: *reinforcement learning, Markov decision process, core-tube plan, space syntax*

**Abstract**: *This passage is an interdisciplinary experiment based on solving Markov Decision Process models that demonstrates typical architectural design problems by implementing three reinforcement learning algorithms - Value Iteration, Policy Iteration and Q-learning as well as analyzing the outputs.*

## 1 Introduction and Implementation Preparation

This passage is an analysis of Markov Decision Process(MDP) and its application on real problems. As I have a background of Architecture and Urban Planning, the first problem I choose to implement MDP model is a Building Core Floor plan simulation of evacuation under fire hazards, and the second one is an architectural theory - space syntax.

The passage is mainly divided into 4 parts, the first part is a brief review on concept of MDP and three algorithms to solve MDP model. The second part and third part are both proposing the problem of core plan and space syntax theory, and the construction and solution of MDP models via three algorithms, and the analysis of output as well, the fourth part is a further analysis on those algorithm performance on the problems by doing parameter analysis, and lead to the final conclusion of this passage.

The implementation of supporting material for this passage is the coding experiment in Java, with BURLAP[1] package installed in JRE 1.8.0, the code is based on tutorial and previous project on GitHub repository[2]

## 2 MDP, Model-Based and Reinforcement Learning Algorithms

### 2.1 Markov Decision Process

A Markov decision process is a tuple ($S$, $A$, $\{P_{sa}\}$, $\gamma$, $R$), where:

- $S$ is a set of states.
- $A$ is a set of actions.
- $P_{sa}$ are the state transition probabilities. For each $s \in S$ and $a \in A$, $P_{sa}$ is a distribution over the state space.
- $\gamma \in [0, 1)$ is called the discount factor.
- $R : S \times A \to \mathfrak{R}$ is the reward function.

The core problem of MDPs is to find a policy for the decision maker: a function $\pi$ that specifies the action $\pi(s)$ that the decision maker will choose when in state $s$. Once a Markov decision process is combined with a policy in this way, this fixes the action for each state and the resulting combination behaves like a Markov chain. The goal is to choose a policy $\pi$ that will maximize some cumulative function of the random rewards.

### 2.2 Three solution algorithms

Value iteration and policy iteration are two efficient algorithms for solving finite-state MDPs. Both of them are standard algorithms for solving MDPs, and there isn't currently universal agreement over which algorithm is better. For small MDPs, policy iteration is often very fast and converges with very few iterations. However, for MDPs with large state spaces, solving for $V^{\pi}$ explicitly would involve solving a large system of linear equations, and could be difficult. In these problems, value iteration may be preferred. For this reason, in practice value iteration seems to be used more often than policy iteration.

Q-learning is a reinforcement learning technique used in machine learning. The technique does not require a model of the environment. Q-learning can handle problems with stochastic transitions and rewards, without requiring adaptations.

**2.2.1 Value Iteration** Algorithm

1. For each state s, initialize $V(s) := 0$.
2. Repeat until convergence { For every state, update $V(s) := R(s) + \max_{a\in A} \gamma \sum_{s'} P_{sa}(s')V(s')$. }

This algorithm can be thought of as repeatedly trying to update the estimated value function using Bellman Equations.There are two possible ways of performing the updates in the inner loop of the algorithm. In the first, we can first compute the new values for $V(s)$ for every state $s$, and then overwrite all the old values with the new values. This is called a synchronous update. In this case, the algorithm can be viewed as implementing a "Bellman backup operator" that takes a current estimate of the value function, and maps it to a new estimate. Alternatively, we can also perform asynchronous updates. Here, we would loop over the states (in some order), updating the values one at a time.

**2.2.2 Policy Iteration Algorithm**

1. Initialize $\pi$ randomly.
2. Repeat until convergence { (a) Let $V := V^{\pi}$ . (b) For each state s, let $\pi(s) := \arg max_{a\in A} \sum_{s'} P_{sa}(s')V(s')$}

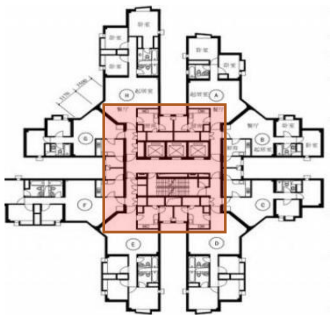
Thus, the inner-loop repeatedly computes the value function for the current policy, and then updates the policy using the current value function. (The policy $\pi$ found in step (b) is also called the policy that is greedy with respect to $V$.) Step (a) can be done via solving Bellman's equations as described earlier, which in the case of a fixed policy, is just a set of $|S|$ linear equations in $|S|$ variables. After at most a finite number of iterations of this algorithm, $V$ will converge to $V^*$ , and $\pi$ will converge to $\pi^*$ .

**2.2.3 Q-learning**

Q-learning is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.The method works by assigning all states a Q-value, and visiting each state to re-assess the Q-Value based on immediate and delayed rewards. Q-learning strikes a balance between executing on immediate rewards and learning to explore delayed rewards.

## 3 Reinforcement Learning Application on High-Rise Core-tube Plan Design

With a background in architecture and urban design, I choose 2 problems within architecture and urban scope to explore Markov Decision process and Reinforcement Learning algorithms. By modeling the core tube plan, it will be interesting to see the policy in condition of fire evacuating, as the VI and PI may fit the real condition of the residents in building and Q-learning can be a good reference of proposing the behavior of visitors in emergency.



### 3.1 High-Rise Design Mode and Core Tube Plan

The first application problem is the high rise floor plan. The design code for high rise is different from the common residential or industrial buildings due to the safety concerns. It must obtain the method to resist lateral forces such as wind load, seismic load, etc. Thus the most common strategy of high rise floor plan design is to have a core tube construction with reinforced concrete shear walls ,and the most common spatial function for the core is vertical transportation including elevator and stairs. In typical conditions like fire warning the stairs take the responsibility of evacuation.

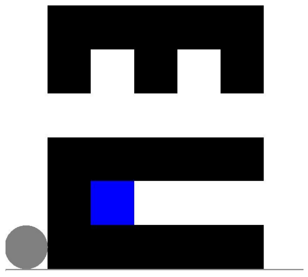


Figure 3.2.1

### 3.2 MDP Modeling

Based on these knowledge, I choose one sample floor plan to construct the MDP space and demonstrate the emergency condition of fire, as shown in figure 3.2.1, the space of core plan is interpreted into the 7*7 grid world, and the origin agent in gray is at (0,0) to simulate the exit of south part of building, with a target position in blue square at (2,1) to indicate the safety exit of stair, the successful transition probability is set to 0.8 to simulate the urgent situation and instability of building as well as crowds, and the goal is to successfully exit the floor. The gamma value is set to -1 and the reward value set to 100.

## 3.3 Result of Policy Finding and Reinforcement Learning Algorithms

The next step is to solve this MDP model by the three algorithms above, figure 3.3.1 shows the process of Value iteration algorithm solving the MDP model as the iteration increases. Figure 3.3.2 shows the Policy Iteration process with also an increasing iteration. From the gradient solution we can figure out the process of algorithm executing - randomly set the initial policy, and then search and find the best policy by discovering the optima and renew the Bellman function.

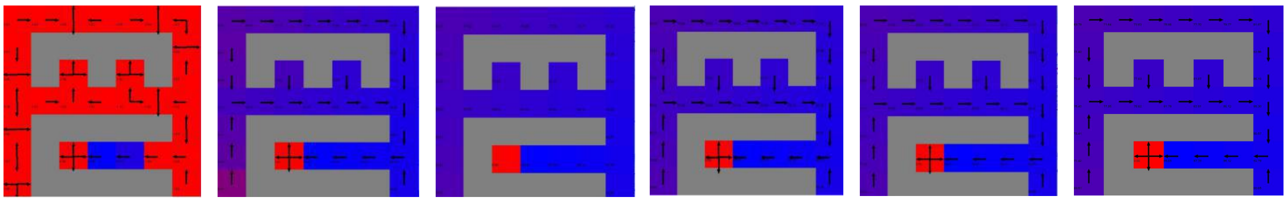


Figure 3.3.1

And we do the same thing on policy iteration, and gain the output plotted by figure 3.3.2, it also developed by seeking the best increment of current policy.

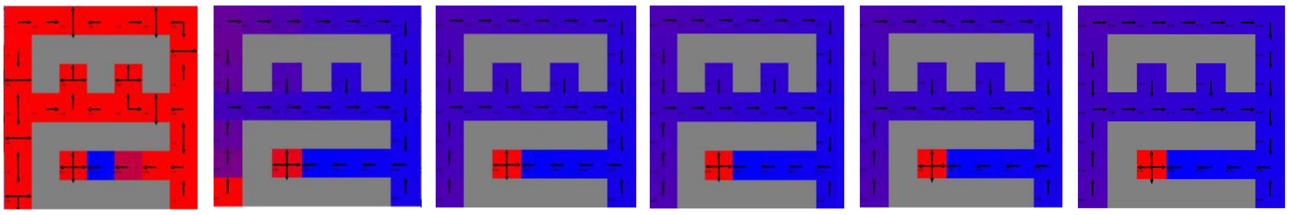


Figure 3.3.2

From these two set of outputs we can discover two obvious character - the first discovery is that both of the algorithm found the target policy in limited iterations low as 10 or less, and from what we re to present in following sections, we can say that they are quick to converge. The next discovery is that they seems to do the same thing! Both set of the output diagrams are showing distinct similarity, and converged to the same policy. Thus what is the reason behind?

One possible assumption to this question rely on the nature of two algorithm - both of them are doing the recursive searching based on single function, the difference is that the Value iteration is finding the optimal for each round of iteration and extract one policy from the result, meanwhile the Policy iteration is doing the continuous exploring to find the best. We can make the analogy that the value iteration is the 'naive' version of policy iteration, and given this solution, as both of them are model-based, the grid world is simple and have the explicit path defined by the proposed policy - and from domain knowledge we could even guess it. Thus in this situation, the inner iteration of PI algorithm may only be 1 round , which makes no difference from Value iteration. And as PI executes in java, we do get the proof of this assumption by printout the iteration times.

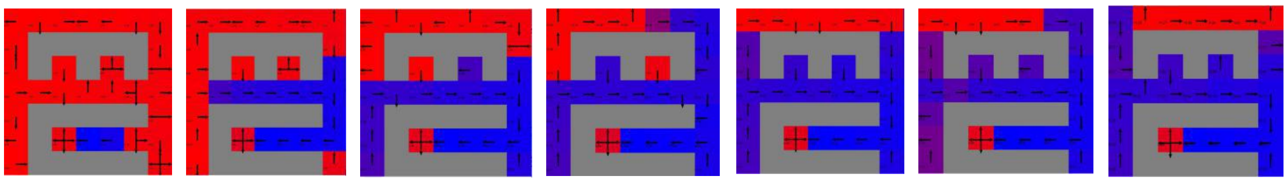


Figure 3.3.3

Then we come to the Q-learning algorithm. Significant difference can be recognized from the output diagram set from those of VI and PI, and the reason is also obvious - Compared with both VI and PI, the Q-learning is not model-based, which means it does not ‘know’ the final status. Thus with each round it will do the search until reach the final status, and by each round- what we call episode. By episode iteration the mind of the agent is ‘reinforced’, and to the end of centuries it will eventually figure out the best way. So what does the diagram set tell us? The reason why it differs from VI and PI is that - they are still exploring. However, as the iteration goes, we can figure out that the mind of agent is ‘reinforced’ - as the shortest path which will lead to greatest reward was gradually found and blued.

### 3.4 Analysis and Comparison

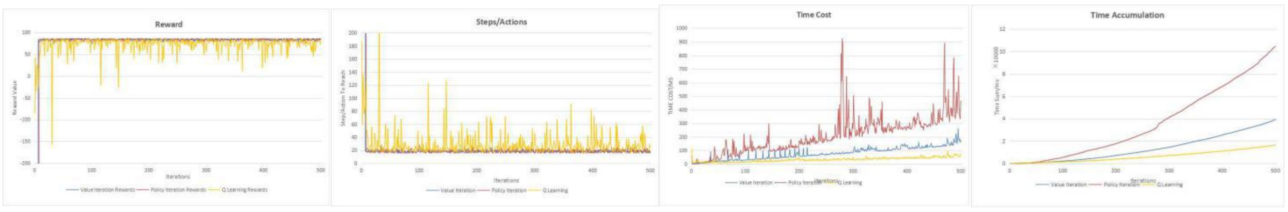


Figure 3.4.1

As we go through these three algorithms, we can have the output in figure 3.4.1 evaluating the reward of the MDP world, the Steps the agent takes, and the time cost as well. From the first two diagram we can see the explicit result that demonstrate the difference between model-based PI and VI and agent-based learning method - The reward and step performance of VI and PI come to convergence quickly and stayed in a stable curve extending far away along x-axis, however the Q learning curve is still under fluctuation. The answer to this phenomenon is apparent- both VI and PI algorithm is based on the knowledge of model, thus the iteration updates are just on the way towards the ending status, however the Q-learning agent are continuously exploring the world episode by episode, it is not doing the goal based thing, but strengthen the mind by reinforcement learning. The instability in curve, indicates the forwarding process in learning.

Another discovery is that the time cost for PI is greater than VI, and the reason is also obvious- that PI is doing iteration to search for the precise policy while VI is just finding the optimal, the time cost of PI is thus more, or even times of VI, and we can also figure out the tendency from time cumulative figure at the most right of figure set.

## 4 Reinforcement Learning Evaluation on Space Syntax Theory

To do further observation on a ‘larger’ status, this passage will continue the discussion over another case named space syntax, which is a classic and popular theory in architectural space and urban planning. But is it that the planning proposal can real fit the need of public? The interesting thing is to see how the policy oriented algorithms and reinforcement learning tells.

### 4.1 Space Syntax Theory in Architectural/Urban Design

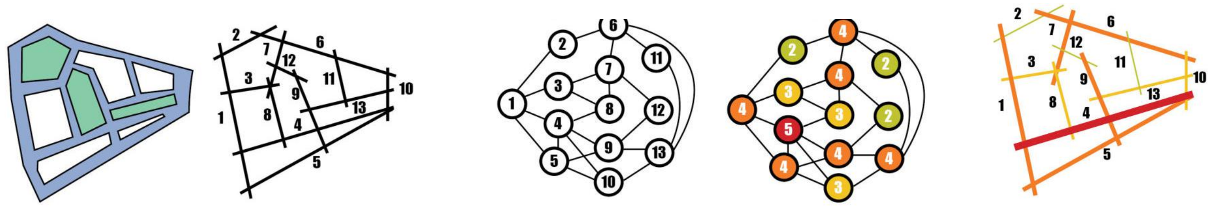


Figure 4.1.1

To begin with the experiment, we need to firstly know what this theory is - Given the space in whatever scale - whether as small as a house, or as big as a city, we can describe the space in topological way - simplify the space and convert them into axial map( 2nd pic of figure 4.1.1), and the axial map reveal the discrete relationship between spaces (3rd and 4th pic), based on this relationship we calculate how much adjacent space it connects to define the ‘Connectivity’ of each space, and then explore from each space by calculating the average ‘depth’ to define the ‘Integration’, and analyse based on these two concepts.

## 4.2 MDP Modeling

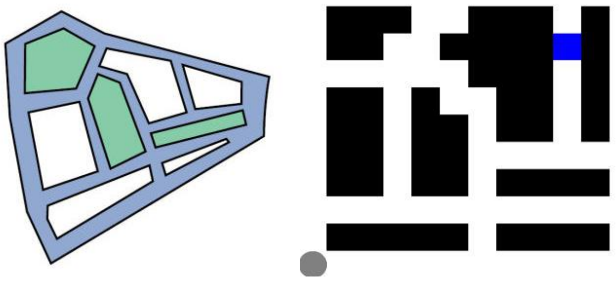


Figure 4.1.2

The modeling of MDP is based on the original street network and transformed it into topology relationship - the same way space syntax proposed. The origin is set at the left bottom with medium ‘connectivity’ and ‘integration’, and the final state is at (9,8), near the top right, which is on the street with lowest ‘connectivity’ and ‘integration’. The goal is to do validation of the theory and maybe challenge it by proposing an agent behavior via three algorithms, where the first two describe the urban planner’s idea and the last one describe the agent’s.

By launching the solution of three algorithm on this model, the policy coming out will describe the directed behavior proposal of people in the space, and the learning of this space can be simulated by agent with reinforcement learning process.

## 3.3 Result of Policy Finding and Reinforcement Learning Algorithms

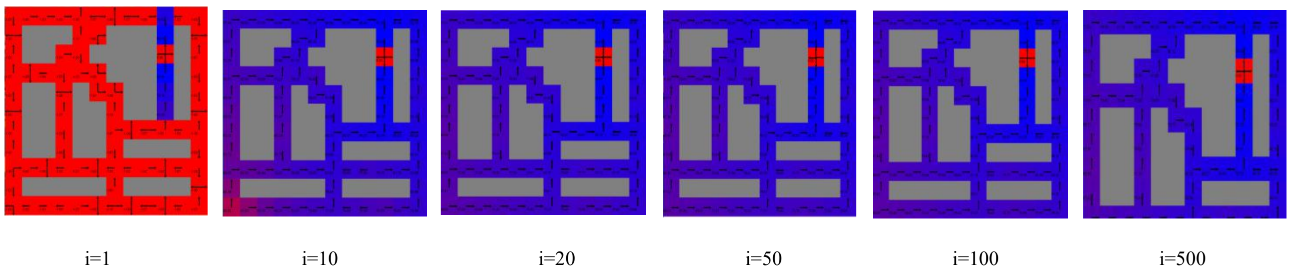


Figure 4.3.1

Value iteration algorithm, again, are doing the convergence in limited iterations, while it still took a bit long for it to converge than on the ‘easy’ grid of core plan. Figure 4.3.1 is showing that the policy and rewards, though without significant difference, are in a gradual development towards solution. Notice that this grid world has a dimension of 12*12, a bigger state than before, and thus the time consume and iterations to converge takes longer than easy one.

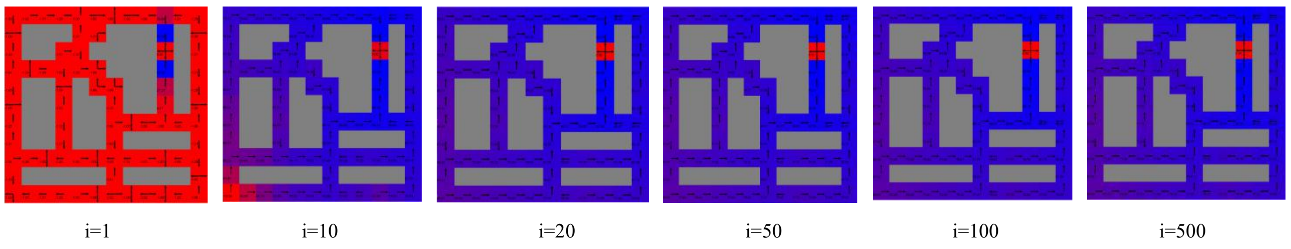


Figure 4.3.2

Policy iteration Algorithm, is again showing the result similar to Value iteration output, figure 4.3.2 proposed the iteration steps, and we can see that together with VI, they are again finding the policy within limited iterations. This is again due to the same nature of search pattern, and the difference in time lies in the fact that the Policy iteration is doing more, detailed data description will be provided later this section.

| item | algorithm | Core Tube MDP | Space Syntax MD | times |
|---|---|---|---|---|
| total time 500 iteration(s) | Value Iteration | 39.288 | 200.929 | 5.114 |
| | Policy Iteration | 104.736 | 412.647 | 3.940 |
| time to converge (ms) | Value Iteration | 0.042 | 0.197 | 4.69 |
| | Policy Iteration | 0.077 | 0.264 | 3.42 |

Table 1

However from table 1 we can compare this duplicated grid world to the easy grid world, the significant difference in time consuming comparison indicates that the lower efficiency in these two model-based algorithm, even if the grid is not of significant larger in size due to the amount of void spaces in world. It can be assumed that if the world continues to get larger and more complex, the time cost for these two policy searching may be too expensive. And unfortunately if the situation comes to the space syntax problem in urban scale, it may lead to extreme low efficiency in calculation.

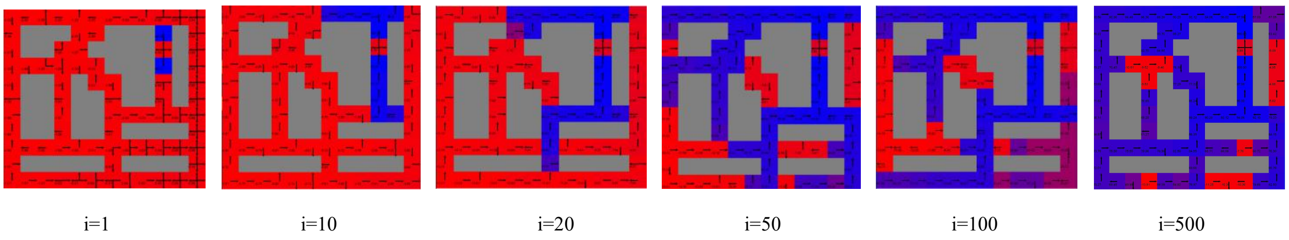


Figure 4.3.3

For Q-learning in this part, it is good for demonstrating the behavior of agent path finding in this situation, though it does not converge. The steps are continuing while it comes out something interesting to the problem - according to the theory of space syntax, the second bottom street have the most connectivity and integration- which means that the relationship strength between this street and the block should be of the best, however, the first unlocked street is not this one, and moreover, the agent's behavior seems totally different to what space syntax tell us.The conclusion may be lead to a challenge to the practice of this popular theory in architectural and urban design field.

## 4.4 Analysis and Comparison

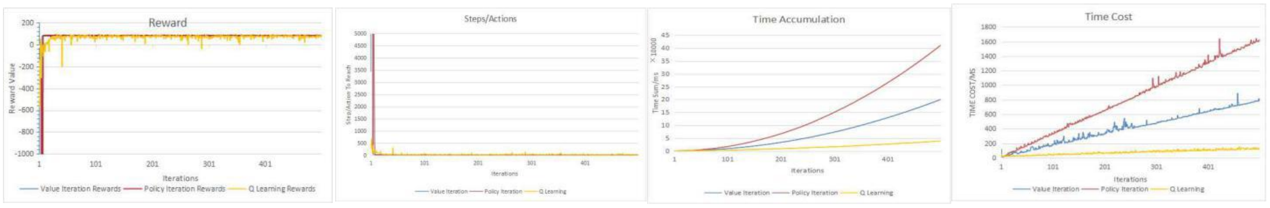


Figure 4.4.1

To draw conclusions from the output for this section, we can see the similar performance in curve shape in figure 4.4.1 from what we get in section 3, the PI is still takes twice as long as VI. However, a slight difference is observed - the distinct in time cost between Q-learning and PI/VI seems larger, and this discovery can also be another evidence in addition to table 1, that the PI/VI are of low efficiency dealing with larger states, and as the states amount grows this problem tend to be more severe.

## 5 Algorithm Parameter Analyzing

After evaluating the performance of three algorithms in MDP world and compared them with each other, we will make assessment for those algorithms themselves by analyzing the parameters.

## 5.1 Value Iteration and Policy Iteration

The most important parameter shared by both PI and VI algorithm is the discount value, which expressed as gamma in the equation, thus the next part aims to figure out the performance when this value changes.

### 5.1.1 Value Iteration Discount Value

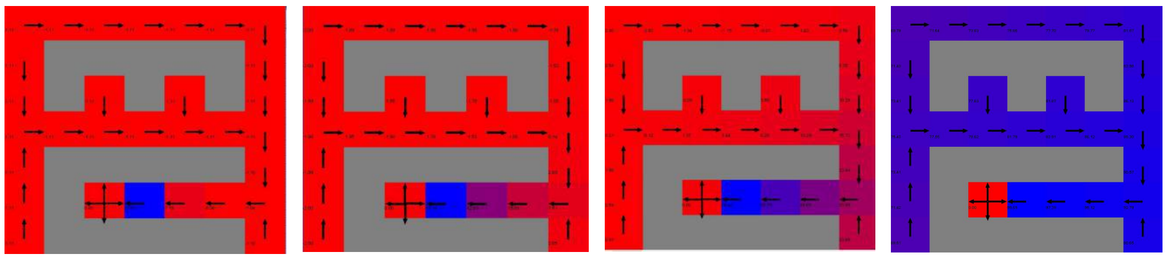


Figure 5.1.1 a

Figure 5.1.1 shows the different performance on quick grid world policy finding within the same iteration via VI algorithm, we can discover the difference between the four pictures with discount value(gamma) set to 0.10,0.50,0.75 and 0.99. The reward of these four situations turns out to be different, with lower in left and higher in right. We may come out with two explanation - the reward is that it should be, or this is because they did not converge yet. Which one is better?

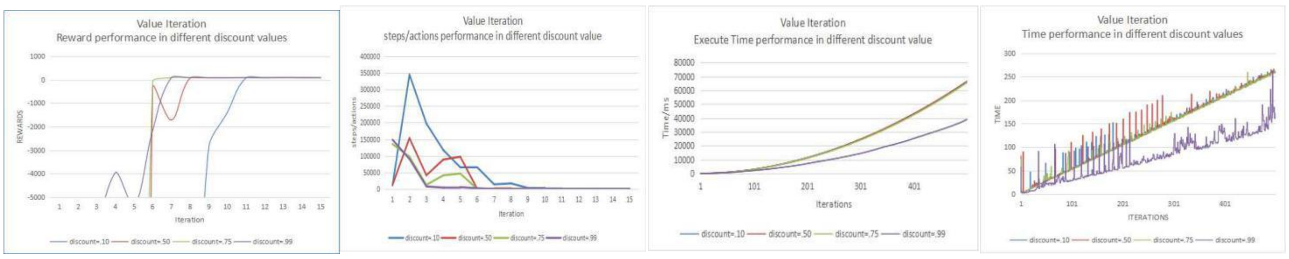


Figure 5.1.2 b

Then we come to further analysis by producing the experiment on different discount values while iteration increases. From figure 5.1.2b we can discover that the answer to the previous question may be the latter, and further more, the lowest gamma tend to be the last to converge while the highest gamma take shorter time in calculation. The reason to these two phenomenon lies in what the parameter is and why - in the equation, the discount value, is to state the ‘long term reward’, which can be explained by providing the example of college education - the investment for now you lose money, the future reward will be more you gain. And in this case, the low discount value indicates that the agent is ‘short sighted’ and not wisdom enough to see the long-term profit, where he search for the current increment in reward and policy, thus it takes much longer for the agent to reach the optimal one, and of course, longer time- which demonstrate the output of time curve in pic 3 and 4.

**5.1.2 Policy Iteration Discount Value**

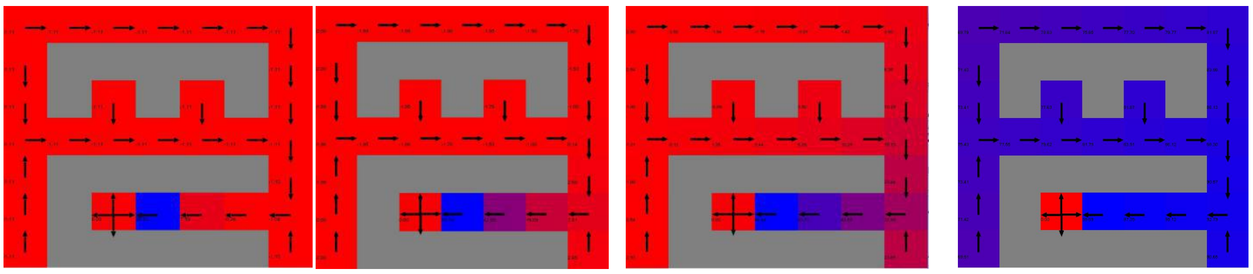


Figure 5.1.2 a

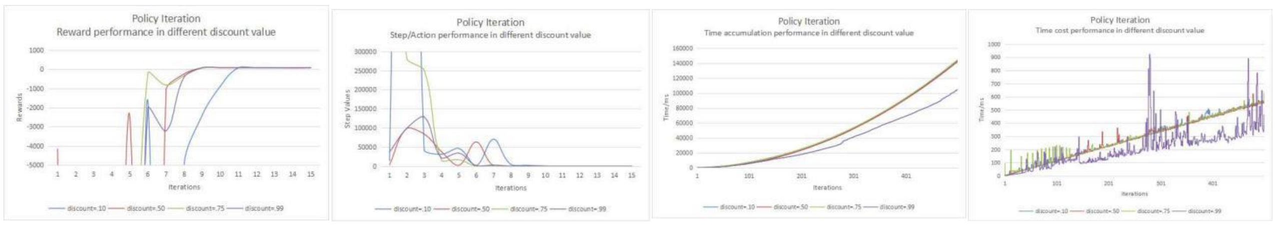


Figure 5.1.2 b

Figure 5.1.2a is doing the same thing as figure 5.1.1a, again we witness the similarity between these two algorithm, the shorter sight the agent is, the longer time it takes to converge.

Figure 5.1.2b is also doing the same thing, however, compared with figure 5.1.1b, the difference in between is that the policy iteration algorithm seems to take one or two more iteration to converge, and together with the result from table 1, the low gamma together with low performance in time weakened the PI algorithm in this status one more step.

## 5.2 Q-Learning

The final part in this section comes to Q-learning, where we compare the algorithm performance itself by adjusting two main variables- the gamma and the learning rate.The strategy for the algorithm is greedy epsilon exploration with a default setting of epsilon=0.1.

### 5.2.1 Discount Value

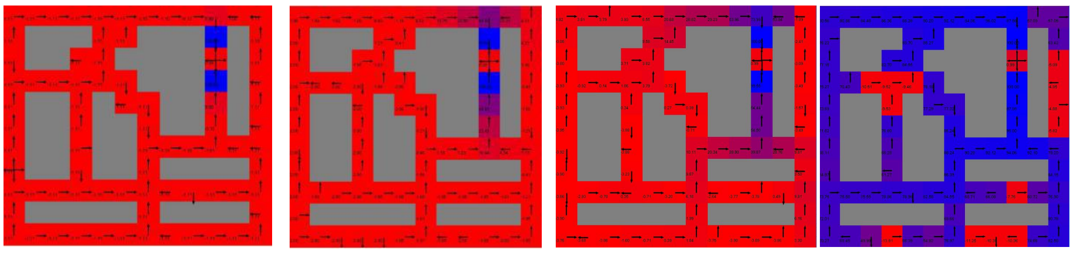


Figure 5.2.1 a

The discount value in this case are similar to those in PI/VI, which indicates the sight of agent, where the high value demonstrate that the agent value the future reward as much as present.

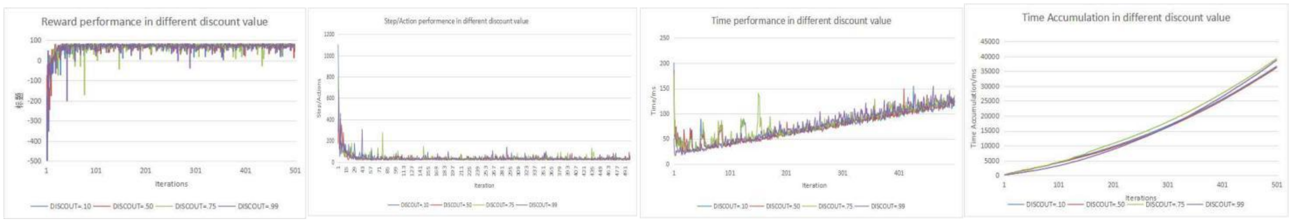
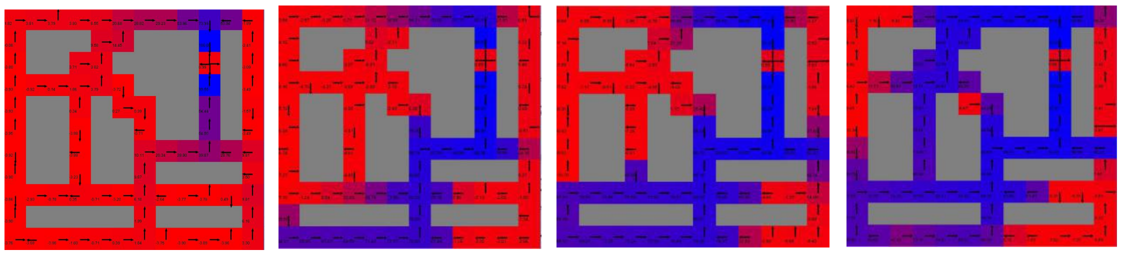


Figure 5.2.1 b

From figure 5.2.1b we can find that the factor of discount value slightly affect the time for calculation, while can also have a impact on the convergence measured by reward values.However when we look back to figure 5.2.1a we can see that the large value of 0.99 in discount have come out with a result that the reinforcement learning by agent in this MDP world seems to converge faster, which can also be seen in the first pic of figure 5.2.1b. The reason for this point is that the more agent values future, and in this case with thin and long path of world grid, the more ‘understanding’ of the world the agent will see. And thus the feedback from each episode of learning is different, with high discount value lead to more strengthened learning.

### 5.2.2 Learning Rate

The last part comes to the learning rate, which is special in Q-learning compared with those other two.



Rate=0.1 Rate=0.5 Rate=0.75 Rate=0.99

Figure 5.2.2 a

From figure 5.2.2a we can conclude an obvious observation that the higher learning rate lead to the fater convergence to the optimal policy, and the apparent reason is that each episode the feedback for the agent is new - by definition of the learning rate, the highest learning rate r=1 indicates that the agent will only accept new knowledge of information.

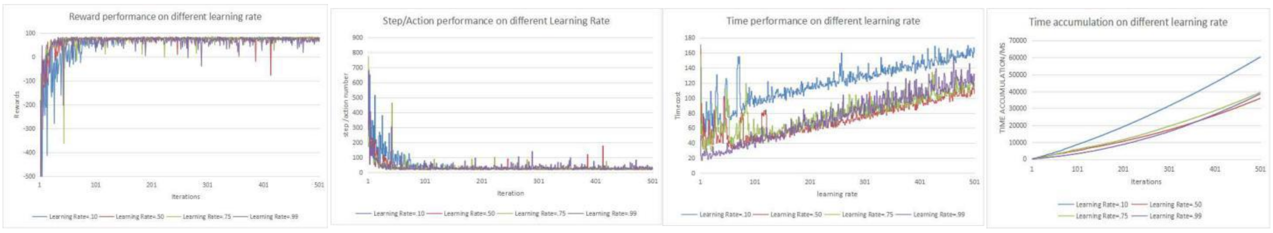


Figure 5.2.2 b

However, if we take a look back into figure 5.2.2b, we may figure out that the highest learning rate not usually lead to best performance, and in this case, the learning rate of 0.5 seems perform best on the general view of time consuming, the convergence, and the reward. The answer to the broad question of which learning rate is best definitely could not be answered, and the best learning rate in this case is due to the condition of our designed Markov Decision Process world, where in this case, a weight of new information acceptance by agent defined by learning rate of 0.50 is the best, and to figure out each condition, the unique test should be done in each specific MDP world.

## 6 Conclusion

In a nutshell, the three different approach of solving MDP world has their character which lead to pros and cons in different situation. Normally speaking,the Value Iteration method and the Policy Iteration method are of the similar algorithm - where both of them are model-based, and need to have the domain knowledge of reward and state in Markov Decision Process model. Compared with Q-learning, it does not need to deal with the puzzle game, and in some case of problem solving and path-finding when we construct Markov Decision Process models, we should have the sense to take advantage of that.

The Q-learning ,on the contrary, are not the model-based iteration method, It does not have the full view of Grid world generated by Markov Decision Process. And it is rather a ‘learning’ algorithm, which takes episodes to discover the reward and absorbed in the final state. It is of most suitable when we have limited domain knowledge to perform the learning method and do data analyzing.

Compared with each other, the Value Iteration have the advantage of low calculation time to the Policy Iteration, however,  the Policy iteration is more exhausting on the MDP world while finding the best policy and get precise returns than Value Iteration. Both of VI and PI algorithm have the limitation of solving large state problems which lead to significant increment on calculation cost, in which the Q-learning have the advantage over them.

References:

[1]BURLAP package: http://burlap.cs.brown.edu/

[2]GitHub Reinforcement Learning Repository:  https://github.com/juanjose49

[3]Machine Learning, Tom Mitchell

[4]Theory of City, Hillier B