Wenjun Wu
April 23, 2017

Analysis of MDP and Reinforcement Learning

Background

In this report, I will discuss Markov Decision Process(MDP) and application of Reinforcement Learning(RL) on MDP problems which are BlockDude and GridWorld implemented by BURLAP java library.

Markov Decision Process(MDP). All MDPs are defined by four components that are user-defined: a set of possible states of the environment; a set of actions that the agent can take; a definition of how actions change the state of the environment and the rewards the agent receives for each of its actions, known as the reward function, which will determine what the best behavior is. In order to solve MDP problems, we can use planning algorithms and learning algorithms. The planning algorithms explored in this report are Value Iteration (VI) and Policy Iteration (PI).

Reinforcement Learning(RL). RL was inspired by behaviorist psychology that studies how agents should take actions in an environment so that some notion of cumulative reward is maximized. Unlike planning algorithms, RL do not need knowledge about the MDP. In this report, I chose to implement Q-Learning as the RL algorithm to solve MDP problems.
Q-Learning

Markov Decision Process Problems

Grid World(GW). GW was shown in the lecture as an example of MDP problems. The domain of the GW is composed of three kinds of grids: wall that the agent can't reach, normal grid that the agent can move to and terminating grid which terminates the game. The domain can have different sized world and supports four actions: east, south, west and north. In this report, I implemented three different domains with various number of states as shown in Figure 1.

1) Single Block World: The initial position of the agent, represented by the gray circle, is at the bottom left corner. The termination position, represented by the green star, is located at the right top corner. This problem has 10 states in total.
2) Four Rooms: The initial position of the agent is at the bottom of the left corner (gray circle). The termination position (green star) is located at the top right corner. This problem has 104 states.
3) Maze: This is a 40*40 maze with the initial position of the agent (gray circle) located at the right bottom corner and the termination position (green star) located at the top left corner. This problem has 800 states.

Block Dude(BD). Besides GW, I also adopted an implementation of the Texas Instruments calculator puzzle game, Block Dude. The goal of the game is to reach the exit. The player has three movement options: east, west and up if the platform in front is only one unit higher. However, in most cases, the agent will never be able to reach the goal by just moving. Instead, the player need to manipulate a set of blocks scattered across the world using pick up action and put down action, through which the agent can raise the block in front and carry the block till put

down at certain location in front of the agent. For this report, two difficulty levels of BD games are evaluated. The easy level has 841 states and the hard level has 14200 states.

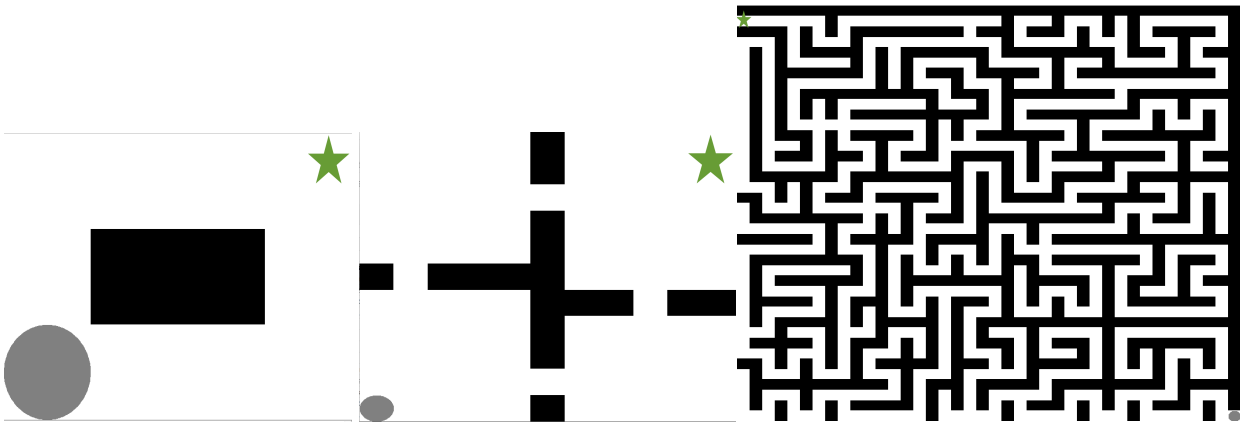

Figure 1 The domain of the three GridWorlds explored in this report. Left: Single Block; Middle: Four Rooms; Right: Maze. The gray point represents agent initial position and the green star represents termination position.

Motivation – Why Interesting?

When talking about MDPs, GW is a good starting point because it has small problem space. The agent only has 4 movement actions and the locations of the grid are fixed. Most planner and learner can converge within 100 iterations. Moreover, the policy and states of a GW domain can be visualized directly. As an observer, we can easily tell the effectiveness of the policy in most GW problems intuitively, which also makes the discussion of states, actions, transitions and Q-values easier. Not only GW is useful in the aspect of demonstrating how to solve MDPs and what parameters are important for the solution, GW also has significant real-world applications. The direct example would be a self-driving car which also has a starting location and a destination. The goal is to reach the destination and when navigating through the world, the car also encounters obstacles such as buildings, rivers and other non-drivable area, which are pretty much like GW problem. In short words, GW could represent a miniature of navigation model and surrounding exploration.

Unlike GW, BD is interesting due to its large problem space. It has much higher running time than the GW problem. It also has larger actions and larger states for the planner and learner to explore, which in some sense is close to what real-world situation is: complex and versatile. These characteristics makes BD a good problem for evaluation MDP and RL algorithms in a large problem space with various states and actions.

Methods

Value Iteration(VI) and Policy Iteration(PI). I solved each MDP problem using VI and PI. For these two MDP problem with varying difficulties and problem, I can investigate how does the problem space affect the performance of planners. Then, I will vary the termination reward(TR), step reward(SR) and discount factor to understand the impact of those parameters on the performance of Vi and PI (Table 1).

Parameters	Values Explored
Termination Reward (TR), Step Reward (SR)	{(5,-0.1), (5,-1), (10,-0.1), (10,-1)}
Discount	{0.99,0.9,0.7,0.5}

Table 1. Explored parameters used in performing experiments with VI and PI. Highlighted value is the default value used when varying other parameters.

Q-Learning. Finally, Q-learning was chosen to solve both GW and CB with varying initial value of Q, epsilon and learning rate(LR) to demonstrate the exploration and exploitation dilemma (Table 2). Through sets of experiments performed, I will show that different exploration strategy performs can be drastically different. The performance of VI, PI are evaluated by runtime, iterations took to converge and number of steps used to reach termination position.

Parameters	Values Explored
Initial Q	{0.1, 1, 10, 100}
Epsilon	{0.05, 0.1, 0.4, 0.7}
Learning rate(LR)	{0.1, 0.3, 0.5, 0.8, 1}

Table 2. Explored parameters used in Q-learning. Highlighted value is the default value used when varying other parameters.

Results

Value Iteration(VI) and Policy Iteration(PI)

GridWorld. The domain the GridWorld is set to be stochastic: when agent performs an action, it has 80% chance to success and 20% chance end up in the wrong direction. The performance of VI and PI on GW domains with different sizes are shown in Table 3. The Utilities and policy output from VI and PI is using default parameters (TR = 5, SR = -0.1, discount = 0.99) are shown in Figure 2 and 3.According to Table 3, the as the number of states increases, the number of iteration increases in both VI and PI, but the increase in VI is larger. We can observe that VI takes more iterations to converge than PI. This is because there are only finite number of policies in a finite-state, finite-action MDP just like GW and thus the search of policy will terminate in a relatively small number of steps. However, value iteration starts at the end and works backward, refining the estimate of V*, which really has no end. In this sense, PI has absolute advantage on number of iterations took to converge.

However, the running time of PI is much longer than VI, indicating that each iteration of VI takes longer time. This is because we need to compute the utility for each state given the current policy and update the state utilities to find the optimal action for each state. Thus, we can observe an exponential growth of time spent by each iteration when the number of states increases. On the other hand, the running time of each iteration of VI is very consistent, about 9-12 seconds per iteration over all three GWs. The runtime is grows linearly in regards to the number of states. A potential reason for this might be due to the fact that converge threshold (maxDelta) is set to 0.01. This tolerance is relatively high comparing with the step reward, which may contribute to the consistent converge speed.

	Single Block	Four Rooms	Maze
Number of States	10	104	800
Performance of VI			
iterations	8	22	188
Runtime (ms)	79	171	2399
Steps taken to exit	7	18	158
Performance of PI			
iterations	3	7	15

Runtime (ms)	138	1010	13293
Steps taken to exit	12	40	167

Table 3. Performance of VI and PI on GridWorlds with different number of states

Moreover, from Figure 2 and 3 below, we can observe that VI has more accurate utility calculated for each state. The utility gradient of Four Rooms problem solved using VI is more obvious than that solved using PI. However, the two methods arrived at the same policy for all GW problems. In this sense, VI performs better in GW domains because of its lower running time and less number of steps to exit (Table 3) although it requires more iterations to converge.

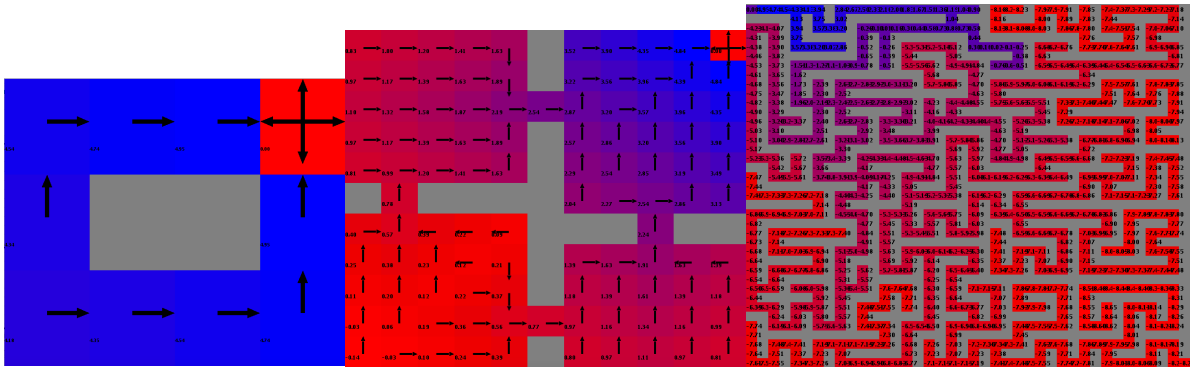

Figure 2. Utilities and Policy of GW solved with VI with Termination Reward = 5, Step Reward = -0 and Discount = 0.99.

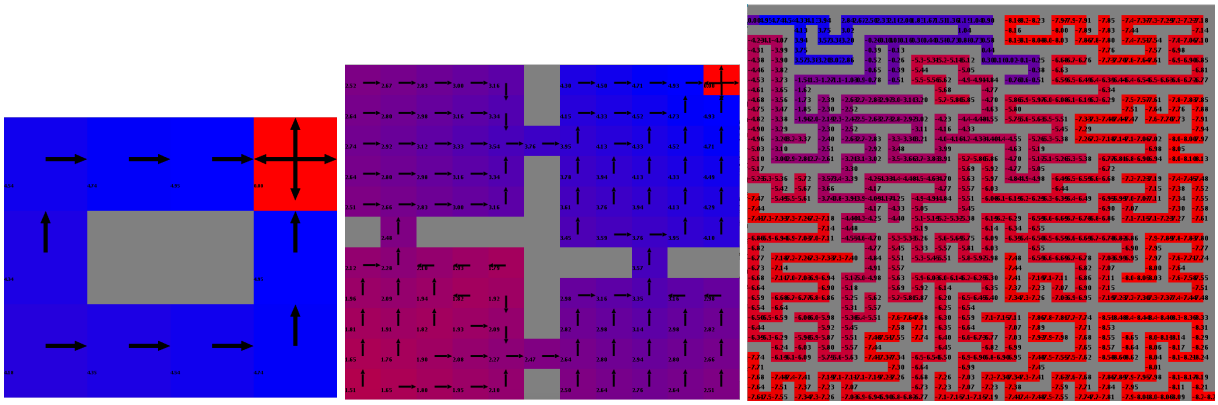

Figure 3 Utilities and Policy of GW solved with PI with Termination Reward = 5, Step Reward = -0 and Discount = 0.99.

We also vary the combination of termination reward and step reward. The result of Four Rooms problem is shown in Table 4. Similar trend was displayed in Single Block and Maze domain and due to the space limitation of this report. I will not specify the result here and if interested, please find complete result in "result.xlsx". As can be seen from the table, the variation of TR and SR does not change the performance of VI much, which is strange since higher TR is expected to draw the agents toward the goal. A potential reason for this could be that the determination of converges counteract the effect of attraction. For PI, the run time and iterations increases as SR and TR increases. This makes sense because the change of reward determines the convergence and with larger reward value, convergence may take longer.

TR, SR	{5,-0.1}	{5,-1}	{10,-0.1}	{10,-1}
Four Rooms solved by VI				
iterations	22	24	24	24

runtime (ms)	171	187	239	219
steps taken to exit	18	22	23	24
Four Rooms solved by PI				
iterations	188	223	162	219
runtime (ms)	2399	2934	1758	2287
steps taken to exit	158	236	223	224

Table 4. Performance of VI and PI on Four Rooms with different combination of TR and SR. Similar trend was observed on two other GWs.

The discount parameter is responsible for the delayed reward. A discount close to 1 means the long term feedback is more important and a small discount means that short term gains is more important than long term gains. According to result from Table 4, we can see that larger discount value results in longer running time and more iterations to converge in VI than smaller discount value. However, smaller discount value results in more steps taken to exit. This makes sense since the agent is less willing to move toward the destination and only the states near the end position have high utility values. This is also confirmed by the experimental result from maze GW. When applying discount value less than 0.99, the agent couldn't find the destination within 100,000 iterations (Figure 4). This makes sense since when dealing with complex maze, long-term feedback is very important in order to reach the final destination. However, as shown by Figure 4, smaller discount values result in the limitation of high utility blocks to be only near destination position.

Discount	0.99	0.9	0.7	0.5
Four Rooms by VI				
iterations	22	18	8	6
runtime (ms)	171	173	130	116
steps taken to exit	18	23	99	3276
Four Rooms by PI				
iterations	7	7	6	4
runtime (ms)	1010	958	1210	919
steps taken to exit	40	67	95	170

Table 4. Performance of VI and PI on Four Rooms with different discount.

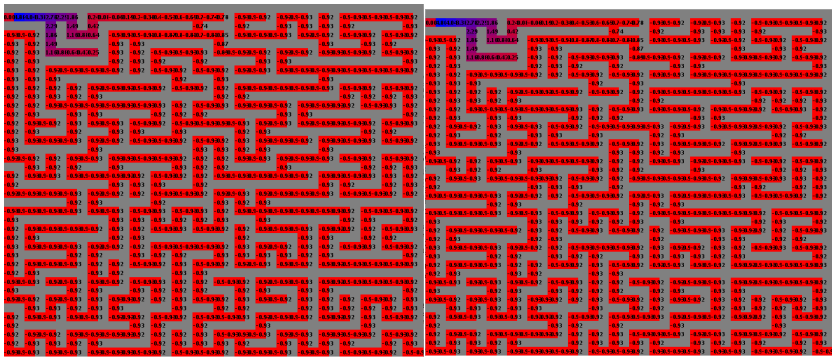

Figure 4 Utility obtained by VI (left) and PI (left) on Maze GW using discount = 0.9, TR = 5, SR = -0.1.

Block Dude. Unlike GW problem, the distance between the initial location and final destination of the agent is complicated. This is not only due to the fact that the number of states are higher, but also due to the increase in possible transitions from 4 to 5. For the BD problem, I also varied two parameters (Table 1) for BD problems with 2 levels of difficulty. The layout of the game is

determined by the number of levels. The performance of VI and PI on GW domains with different sizes are shown in Table 5.

	Easy	Hard
Number of States	841	14200
Performance of VI		
iterations	223	236
runtime (ms)	5089	82753
steps take	19	94
Performance of PI		
iterations	21	52
runtime (ms)	15938 ms	1135842
steps take	19	94

Table 4. Performance of VI and PI on Block Dude MDPs with different sizes using default parameters. TR = 5, SR = -0.1, Discount = 0.99.

Similar to the result from GW, the run time and iterations both increases as the difficulty increases. Also, due to the increase in problem difficulty, the number of steps taken also increases for both VI and PI. PI takes much less iterations to converge when comparing with VI. However, the running time of Vi is much shorter than PI. The same reasons discussed above applies. Also, the time taken by each iteration increase dramatically, about 30 times longer as number of states increases from 841 to 14200. Again, since the converge threshold was set to 0.01, the convergence speed of VI, or the time spent for each iteration, increases slower over the two BD domains comparing with PI. The same reasons applies: the program needs to solve a large number of linear systems in order to evaluate each state, the time spent for each iteration increases exponentially as the number of states increases. Lastly, despite the difference in runtime and iterations, VI and PI have the same number of steps taken in both domains. This makes sense because the change of reward determines the convergence and with larger reward value, convergence may take longer. Checking the actual actions generated by VI and PI also indicates that the action sequences are the same.

We also vary the combination of termination reward (TR) and step reward(SR) to explore the effect of those on performance of VI and PI. However, since PI takes about 20 minutes each run. Only easy mode BD problem is used in parameter tuning. Like result obtained above, the number of steps taken to exit from VI and PI are 19 regardless the Termination Reward and Step Reward. Figure 5 below shows the effect of TR and SR on the iteration took to converge and runtime of VI and PI. In general, the number of iterations used by PI is very consistent regardless of TR and SR. The runtime of PI increases as SR or TR increases. In comparison to PI, the run time and number of iterations both doubles for VI when SR or TR increases. This is different from what we observed in GW problems, in which the run time and number of iterations were not affected so much by TR and SR. This makes sense since when comparing with simpler problems like GW, BD has 5 transitions models and much higher number of states. Thus, a larger reward would have more impact of the utilities of the states that are very far from the destination position.

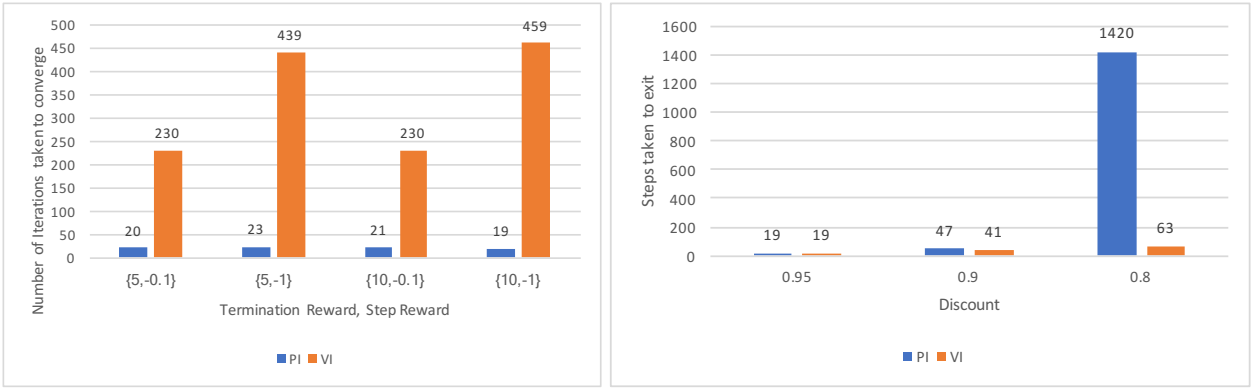

Figure 5 Effect of Termination Reward and Step Reward on Performance of VI and PI on DB Problems.

Finally we vary the discount parameters as described in Table 1. Similar trend as in GW Problems are observed here. The results are consistent with my reasoning that smaller discount parameters may lead to non-optimal actions and thus longer run time while higher discount values result to less steps to exit, time and less number of iterations to converge.

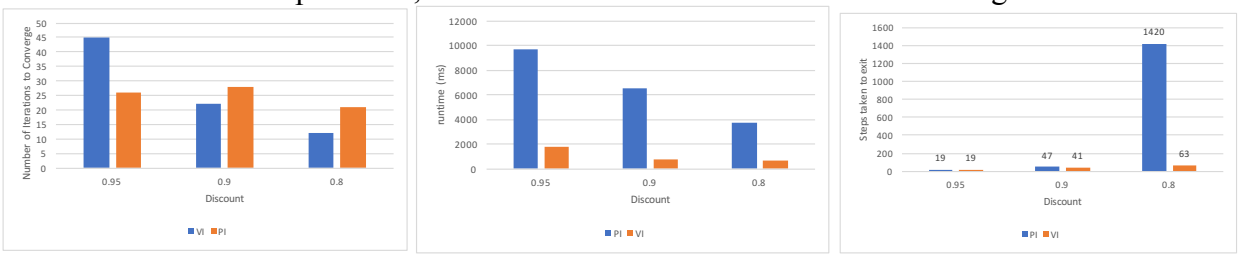

Figure 6 Effect of Discount on Performance of VI and PI on DB Problems.

In general, we can conclude that in both problems, VI has advantage in its ability to generate similar result with PI but using much less runtime in both GW and BD problems. The advantage of VI in runtime is more obvious in complex problems like BD Problem. However, in terms of number of iterations to converge, the number of states does not have a huge impact on PI, but for VI, the number of iterations increase linearly with number of States presented in the problem. Also, in complex problem as BD, reward has larger impact on behavior of VI as discussed above.

Q-Learning(QL)

I chose Q-learning as my Reinforcement Learning (RL) algorithm. I varied the initial Q values, epsilon values according to Table 2 above. The performance metrics will be used to evaluate QL are Number of Steps Per Episode and Average Reward Per Episode. Through the first metrics, the convergence of QL can be observed and through the latter, the trend of convergence of reward and the reward value of the final solution. Finally, the reward value will be compared to the results from VI and PI.

The fundamental tradeoff in QL is the exploration-exploitation dilemma: there is only one agent but we need to perform two conflicting tasks. On one hand, we need to take action that results in maximum reward, but on the other hand we need to take actions that we don't know much and need to learn about. QL uses a process like simulated annealing to avoid the problem caused by action selection strategy with greedy-nature. Epsilon acts as the decaying factor, or the cooling factor for reducing the initial Q values, which acts as the initial temperature in context of simulated annealing. Higher epsilon values mean more exploitation while lower epsilon values

represents more exploitation. Similarly, higher initial temperature, which is initial Q values, would lead to more exploration and vice versa. The learning rate in Q-learning determines the significance of updated information regarding old information. Learning rate of 1 indicates that the newly acquired Q-value will completely override the previous value.

GridWorld. The result of QL in solving GW problems are shown in Figure 7 to 9. Due to lack of memory error, only initial Q value exploration was performed with the maze domain (Figure 9). In general, smaller initial Q value, smaller epsilon generates better performance, which might because GW has limited number of states to be explored and the convergence is rather quick, about 500 episodes even with initial Q as large as 100. The behavior of Q learning is fully demonstrated in the graph. With a larger initial Q value, the average reward starts low and start to become more positive as the temperature cools down. Also, with larger initial Q values, convergence is harder and takes longer time (Figure 10). However, when the problem space is simple, like GW problem, not much exploration is needed to find the optimal solution and exploitation is more efficient. Similarly, a high epsilon value would result in excessive exploitation and limits the performance in the case of GW problems, which can be clearly seen from the graphs on the left of Figure 7 and 8. However, unlike initial Q value, epsilon does not affect the convergence behavior. Moreover, the learning rate does not have huge impact on the performance of QL (Figure 7 and 8 middle).

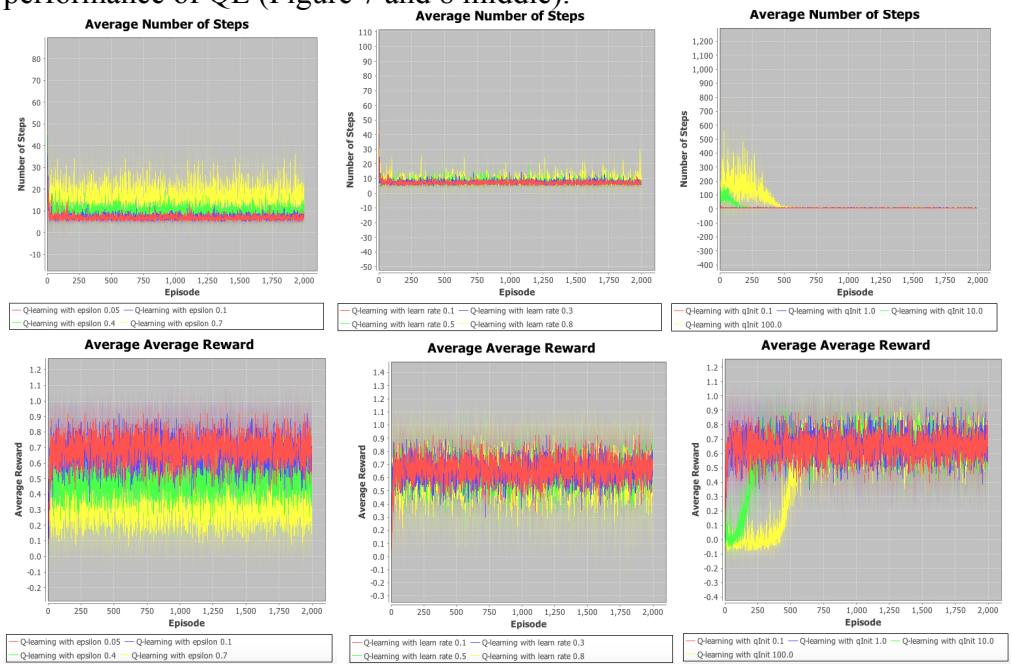

Figure 7 Effect of initial Q, learning rate and epsilon on performance of Q-Learning on Single Block (GW) Problem. The upper figures exhibit the Number of Steps; the lower figures exhibit the average reward.

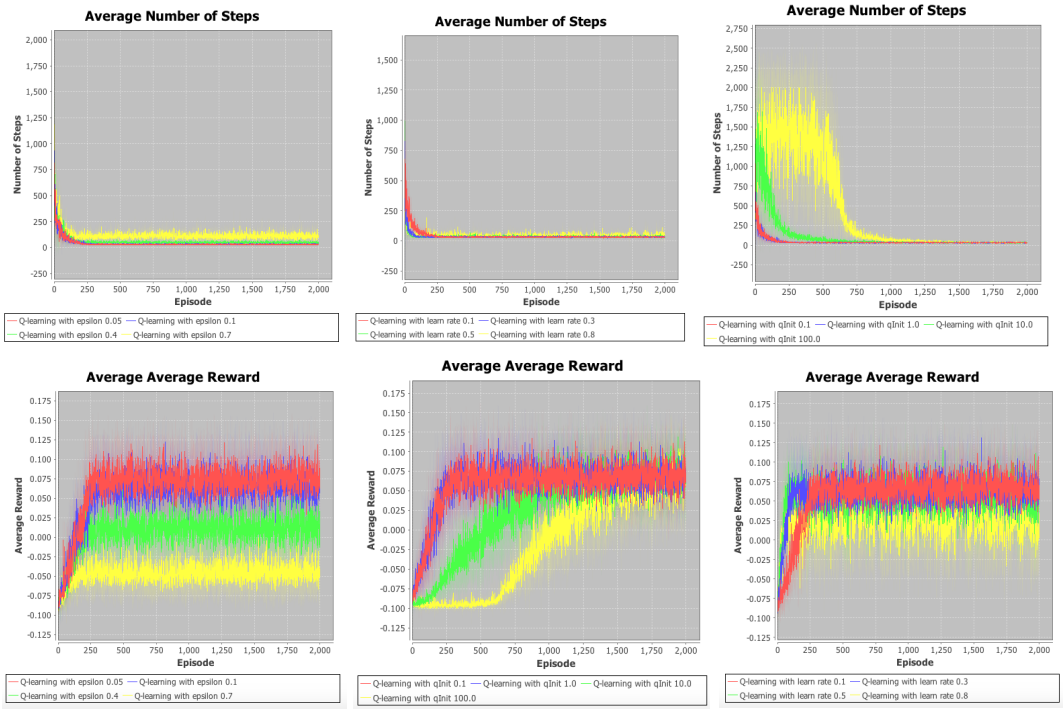

Figure 8 Effect of initial Q, learning rate and epsilon on performance of Q-Learning on Four Rooms (GW) Problem. The upper figures exhibit the Number of Steps; the lower figures exhibit the average reward.

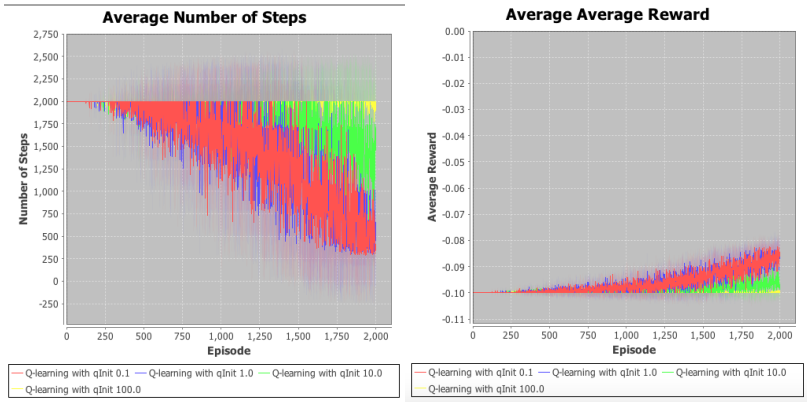

Figure 9 Effect of initial Q, learning rate and epsilon on performance of Q-Learning on Maze (GW) Problem. The upper figures exhibit the Number of Steps; the lower figures exhibit the average reward.

Block Dude. Same as what I did for GW problems, I varied initial Q, epsilon and learning rate according to Table 2. Due to the lack of memory error, QL was only successful on easy mode. The result of QL in solving GW problems are shown in Figure 10. Similar trend as we observed in GW problems are also observed here. This indicates that both problem do not require much exploration to reach the optimal solution. From the graphs on the left of Figure 10, we can see that the best performing epsilon is 0.05. As epsilon increases, the variation of average number of steps per episode becomes larger. Besides, epsilon does not affect convergence behavior and QL with epsilon = 0.05 has the highest average award per episodes, which is similar to we observed in GW problems. From the graphs in the middle of Figure 10, we can see that again, learning rate is not a role player in performance, but higher learning rate decrease episodes took to convergence. From graphs on the right, we can conclude that the best performing initial Q = 0.1. Large initial Q makes convergence extremely hard, as observed before.

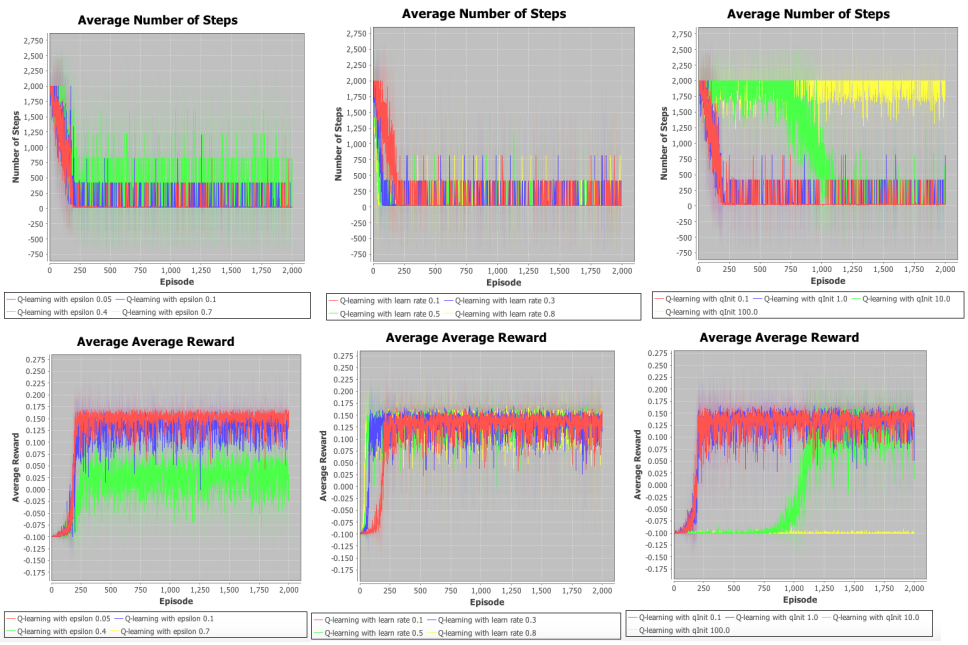

Figure 10 Effect of initial Q, learning rate and epsilon on performance of Q-Learning on easy mode of BD Problem. The upper figures exhibit the Number of Steps; the lower figures exhibit the average reward.

Conclusion

Using the best performing parameters and episode = 1000, I re-performed QL and the comparison between planner and learner is demonstrated below. Again, Q learning in hard block dude failed due to lack of memory error.

Average Reward Per Episode			
	VI	PI	Q-Learning
Simple Block	0.6778	0.6882	0.73
Four Rooms	0.15	0.14	0.075
Maze	-0.084	-0.1208	-0.075
Easy Block Dude	0.0139	0.1595	0.168
Hard Block Dude	-0.018	-0.083	NA (lack of memory)

Table 5. Comparisons between VI, PI and QL on all MDP problems in regard to average reward per episode.

As can be seen from the results above, Q learning performs better in Simple Block, Maze Easy Block Dude. However, the difference between the performance of learner and planner are very alike in most problems.

Besides, as discussed in previous sections, VI performs better than PI in terms of runtime and steps taken to exit. PI has advantage in number of iterations to converge. Also, from the table, we can also see that in terms of average reward per episode, VI and PI have very similar performance.