Enmao Diao CS4641
Project 4

MDPs and Reinforcement Learning Report

MDPs

GridWorld. A basic grid worlds as MDP problem. The domain allows for the creation of arbitrarily sized grid worlds with user defined layouts. The grid world supports classic north, south, east, west movement actions that may be either deterministic or stochastic with user defined stochastic failures as probability. The domain consists of only two object classes: an agent class and a location class, each of which is defined by and x and y position. We implement two kinds of locations and one agent for our problem. There is one agent moving toward the final location. There are one final location and possibly several wall locations. The number of wall locations are specified as percentage by the user. At the final location, the user will exit and halt moving behavior. When encounters wall, the agent will stay at its original position.

BlockDude. An implementation of the Block Dude puzzle game. The goal is for the player to reach an exit. The player has three movement options: west, east, and up if the platform in front of the player is only one unit higher. However, because of the landscape, the player will never be able to reach the exit just by moving. Instead, the player must manipulate a set of blocks scattered across the world using a pick up action (which raises the block the player is facing above their head to be carried), and a put down action that places a held block directly in front of the agent. The user can specify the number of levels that the blocks stacked and generated different world for each level.

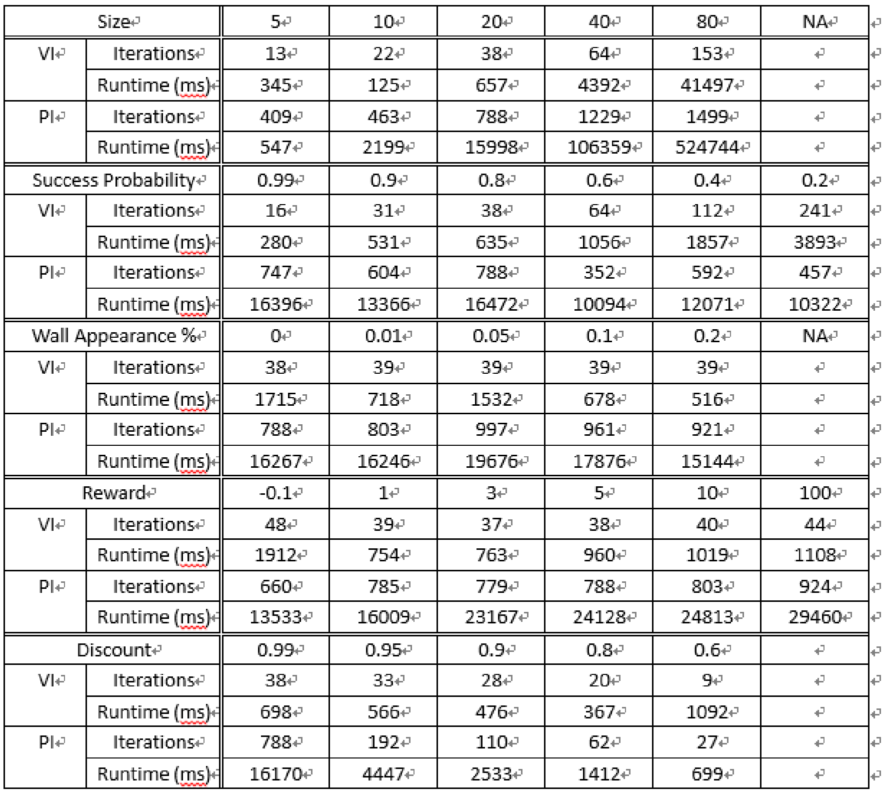
WHY INTERESTING

The reason that the GridWorld problem is interesting is that it has a relative small problem space and mostly the planner and learner will converge within 40 iterations. Meanwhile, there are a lot of parameters we can tune for this problem in order to have a better understanding the impact of the model on the solutions. For example, we can have different number of walls scattered at random locations and have different stochastic success probability of each move. The flexibility on the parameters of the model gives us a better understanding the MDP problem as well as the planner and learner. The reason that BlockDude is interesting is that it can have a relative large problem space. Its runtime increases significantly as the level increases which create much more states for the planner and learner. This problem is also very close to the real world games because it was actually a game Texas Instrument calculator back in the old days.

APPROACH

In order to have a clear view of the MDP problem and its impact on different planners and learners. First we solve each MDP using Value Iteration (VI) as well as Policy Iteration (PI). For the GridWorld (GW) MDP problem, we varying the size of the GW, the stochastic success probability and the percent of walls appearing for the GW models. The size of the GW can help us understanding the impact of problem space on the planners and learners. A large GW has more states because each grid of GW represents one state. The stochastic success probability also helps understanding the influence of probability on the MDP problem. We also evaluate with a very low probability and the result actually shows the nature of MDP solvers. We fix the size of GW as 20x20 and we vary the probability that walls appearing. Sometimes the solution is not reachable because the random generating walls actually blocks the path from the start coordinate to the final location. For the BlockDude (BD) problem, we vary the number of levels and the number of states and run time increases significantly. For VI and PI planners, we also vary the reward of the final location and the discount parameter to show that impact of algorithms on models. For Q-Learning (QL), we very the initial value of Q and epsilon to show the exploration and exploitation dilemma. With different exploration strategy, the result actually can be quite different.

TABLE I
GW PARAMETER TUNING RESULTS



Size		5	10	20	40	80	NA
VI	Iterations	13	22	38	64	153	
	Runtime (ms)	345	125	657	4392	41497	
PI	Iterations	409	463	788	1229	1499	
	Runtime (ms)	547	2199	15998	106359	524744	
Success Probability		0.99	0.9	0.8	0.6	0.4	0.2
VI	Iterations	16	31	38	64	112	241
	Runtime (ms)	280	531	635	1056	1857	3893
PI	Iterations	747	604	788	352	592	457
	Runtime (ms)	16396	13366	16472	10094	12071	10322
Wall Appearance %		0	0.01	0.05	0.1	0.2	NA
VI	Iterations	38	39	39	39	39	
	Runtime (ms)	1715	718	1532	678	516	
PI	Iterations	788	803	997	961	921	
	Runtime (ms)	16267	16246	19676	17876	15144	
Reward		-0.1	1	3	5	10	100
VI	Iterations	48	39	37	38	40	44
	Runtime (ms)	1912	754	763	960	1019	1108
PI	Iterations	660	785	779	788	803	924
	Runtime (ms)	13533	16009	23167	24128	24813	29460
Discount		0.99	0.95	0.9	0.8	0.6	
VI	Iterations	38	33	28	20	9	
	Runtime (ms)	698	566	476	367	1092	
PI	Iterations	788	192	110	62	27	
	Runtime (ms)	16170	4447	2533	1412	699	

PLANNER

GRIDWORLD

For the GW problem, there are five different parameters varying for VI and PI including the size of GW, the stochastic success probability, the percent of walls appearing, the reward of the final location and the discount parameter. Table I shows all the results of varying parameters including metrics like number of iterations to converge and run time. We will analyze each parameter variation to have a better understanding of GW and planners.

First, we varying the size of GW from 5x5 to 80x80. As the size of the world increases, of course, the value of states also increases because the number of states is equal to the number of grids in GW which means size of 80 GW has 6400 states. With more states need to be iterated, both VI and PI need more iterations and run time to converge. The iterations of PI here means the sum of iterations in inner VI for policy evaluation. As the size increases more, the runtime of VI and PI increase tremendously. As you can see when size increases from 40 to 80, the run time of VI increases 10 times and that of PI increases 50 times. This is due to the nature of GW problem that increasing the size makes the problem exponentially harder. Meanwhile, comparing VI and PI we can see that the VI converges faster than PI because within each iterations of VI, each state updates the utility function based on its neighbor states but each iteration of PI has to consider all actions in each state iterations. Although we evaluate the policy with linear equations, as the size increases, the possible improvement also increases because there will be more optimal answers to the GW questions. An example result of size 20x20 GW of both VI and PI is shown in Figure 1. As you can see, most of their actions of the policy are consistent, but some minor differences results from the fact that there are many optimal solutions to this problem.

Success probability here means the stochastic success probability of each move in GW. As you can see from the table, the run time of VI increases as the success probability decreases. However, the run time of PI decreases as the success probability decreases. Figure 2 shows the result of VI and PI with variant success probability. As you can see, they do have similar optimal solutions and mostly differ on the diagonals. This means that PI is better with handling uncertainty and randomness in the MDP problem in terms of efficiency. This is due to PI evaluate the actions directly while VI has no idea about the higher level. What VI evaluates is just the value function based on neighbor states. When the success probability decreases, VI has a better chance accept the action toward the final position, but in the next few states, it will have higher possibility of going to a wrong direction. Besides, the reason that the run time of PI is greater than that of VI is due to the nature of GW problem as we discussed in the previous paragraph. The large number of possible shortest paths slows down the converging process of PI since it evaluates the actions in the higher level. Another interesting fact is that, when the probability decreases to 0.4 or 0.2, the policy actually asks the agent to go to the opposite direction compared with the policy of 0.99 or 0.9 probability (see Supporting Files). This is because going in the opposite direction will actually have higher probability of entering the final location when the success probability is less than 0.5.

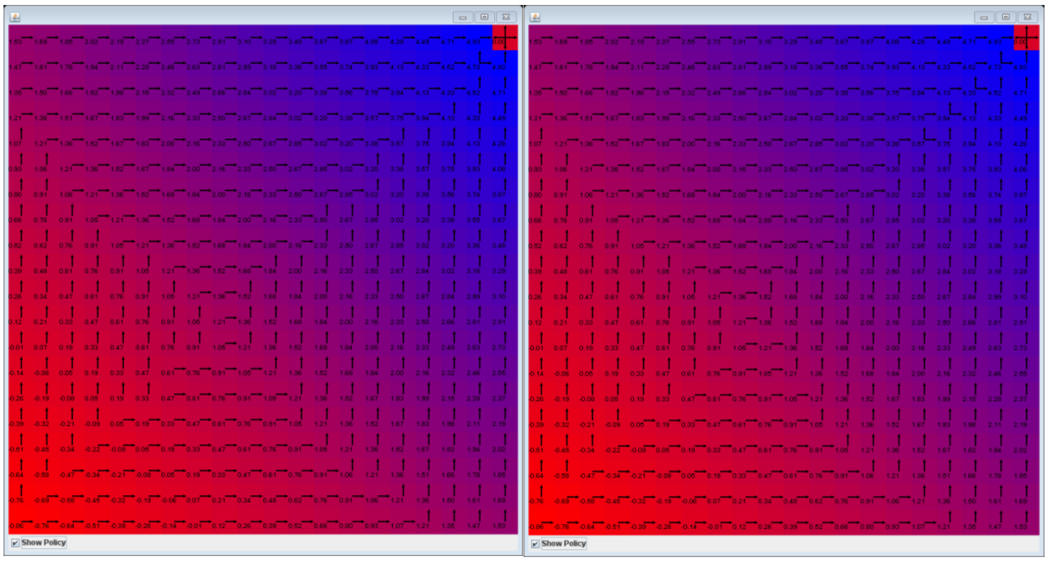


Figure 1. Size 20 GW VI and PI result.

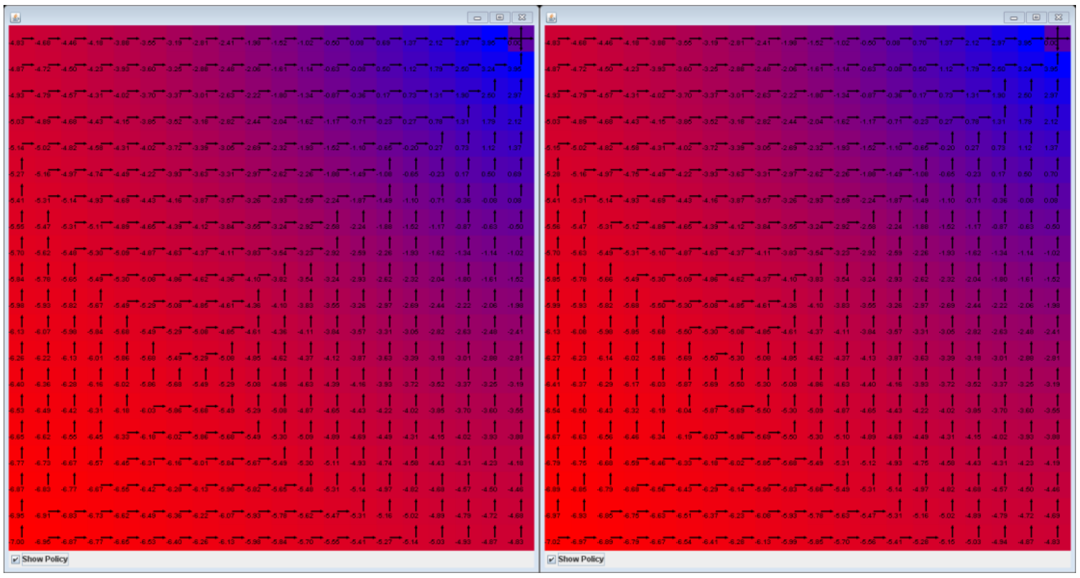


Figure 2. Success probability 0.6 GW VI and PI result.

We also add walls to our world with a parameter wall appearance percent. With a higher percent value, there will be more walls in the world which are not considered as states, and thus slightly less states to be considered by planners. When we have less percent value, for both VI and PI, the iterations tent to slightly increase but the run time decreases. Especially, the run time of VI drops significantly. The reason is that the actual run time is mostly determined by the number of states need to be evaluated. Obviously, with less states, the run time decreases. I believe the reason that the inner VI iterations of PI slightly increases is because the action of returning to the original state when reaching a wall add more uncertainty to the problem. When the percent value equals to 0.2, the iterations required to converge actually drops back to the original value when there are no walls. This is possibly because the uncertainty becomes a kind of certainty because 20% of the locations are actually having the similar transitions now. While with the percent value equal to 0.05, PI need more inner VI iterations to converge with those uncertainties. The optimal policy is not easy to converge because some actions at some states have different transitions from others. The run time of PI, due to slightly increasing iterations, only decreases a little. As you can see from Figure 3, their results do mostly agree with each other, since both of them reaches the optimal policies. Minor differences are due to the fact that there are not only one optimal policy.

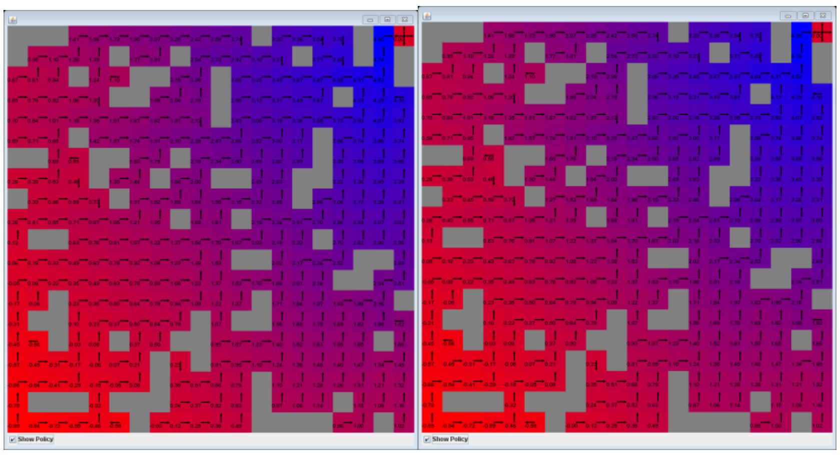


Figure 3. Wall appearance percent 0.2 GW VI and PI result.

The reward is specified as the reward value of the final location. The default reward value for all non-final locations is always -0.1. This is the reason we start the analysis from reward equal to -0.1. We can imagine that when we have even smaller reward value than the default reward value, we will take a long time to converge to the optimal policy. As you can see from the table, when we start from default reward, the run time of VI is extremely high because only the locations near the final location know to enter the final location but all other states do not have a tendency because they only updates the value function based on their neighbors. However, the run time of PI is very small with reward value -0.1. This is because, when considering the actions in the high level, not only the states near the terminal state but also all other states know that the action of moving toward the final location will terminates the decrease of reward. The run time and iterations of PI increase when reward increases is simply because the convergence is determined as the change of reward. With a large reward value, the change will take some time to become small enough to trigger convergence. The run time and iterations of VI are interesting. They first decreases from a large value because the reward attracts more near neighbor to enter the final location. However, they reaches a large value when the reward is 100. This is, I believe, because the determination of convergence overcomes the effect of attracting more neighbors. We reach an optimal reward value at about 3. It means all states now have a tendency to move toward the final location. When we keep increasing the reward value, the time it takes to converge also increases which is the same reason of PI. As you can see from Figure 4, their results mostly agree with each other as they can definitely reach the optimal policy with such large reward value.

The discount parameter, gamma, is used to weight the delayed reward. When we have a discount value close to 1, it means we are willing to take short term negative feedback and believe that we will possibly have long term positive feedback. Gamma is used in the utility evaluation function and this differs the utility from the reward value. One takes following consequences into account and one is only an immediate feedback. You can see from Figure 5 that VI and PI do agree with each other and more importantly, when having small value of discount value which means we are willing to take short term gains instead long term gains, we end up not having the optimal policy because the states near the agent start position will not tend to move toward the final location. Only the states near the final location enters it because of the short term feedback. This is similar to the case where we have small reward value for VI. As you can see from the table, large discount value actually takes more time and iterations to converge for both VI and PI. It makes sense since we are taking long term feed back into account. Actually, in the worst case the number of iterations grows polynomially in 1/(1-gamma), so the convergence rate slows considerably as the discount value approaches 1.

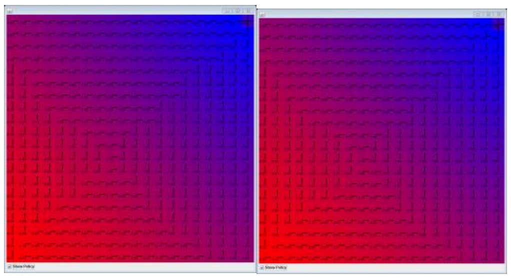


Figure 4. Reward 3 GW VI and PI result.

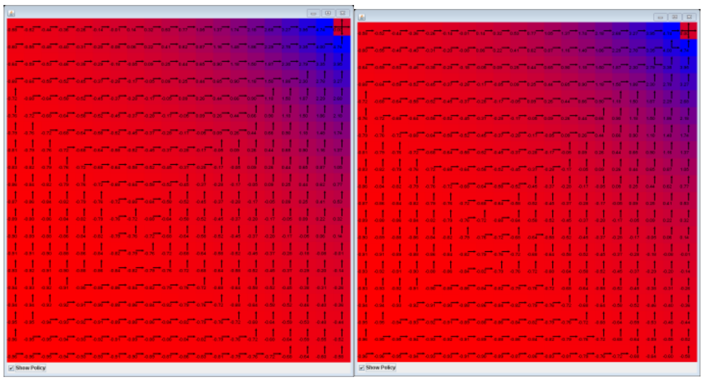


Figure 5. Discount 0.9 GW VI and PI result.

BLOCKDUDE

We learning most of planners from the previous section. This section will emphasize how the performance planners differ in different MDP problems. For the BD problem, there are three different parameters varying for VI and PI including the level of BD, the reward of the final location and the discount parameter. Like Table I, Table II shows all the results of varying parameters including metrics like number of iterations to converge and run time. We will analyze each parameter variation to have a better understanding of BD and planners.

TABLE II
BD PARAMETER TUNING RESULTS

Level		1	2	3			
VI	Iterations	459	1	459			
	Runtime (ms)	6905	868147	150476			
	States	841	230996	14200			
PI	Iterations	724		2679			
	Runtime (ms)	19364		1553309			
Reward		-0.1	1	3	5	10	100
VI	Iterations	459	459	459	459	459	459
	Runtime (ms)	6877	6433	6410	6477	6567	6317
PI	Iterations	710	741	707	666	700	684
	Runtime (ms)	17249	23425	17869	16758	17697	16678
Discount		0.99	0.95	0.9	0.8	0.6	
VI	Iterations	459	90	44	21	10	
	Runtime (ms)	7055	1811	859	445	225	
PI	Iterations	668	328	198	112	34	
	Runtime (ms)	19445	11732	7650	4246	1414	

The level of BD problem is the maximal height of blocks in this game. The layout of the game is set based on the number of levels. Generally, higher level BD games are harder than lower level BD games. However, this does not mean that a two-level BD game is necessarily easier than a three-level BD game. As you can see from Table I, we generate a two-level BD that has the most number of states compared with the one-level and three-level game we generate. After all, for the same game, the possible transitions are determined, so the number of states generally determines the difficulty of the problem. For the two-level game, we only run it over VI for one iteration and you can see that the runtime is extremely large. This is consistent with our previous argument that increasing the number of states significantly increases the run time and iteration to converge.

Recall that in GW, the run time of VI with small reward value is high. With increasing reward value, the run time of VI first drops because of the attraction of the final location and then increases because the large reward value slows down the convergence rate since the convergence is determined with the change of reward. In BD, the iterations of VI is similar with that in GW. Both do not vary much. The different part of VI in BD is that although we do have an optimal reward value at reward equal to 3, at reward equal to 100, the run time drops a lot. This is because the exit location of BD can be quite far from the current state. There is a limit of the distance between the final location and the agent location in GW game. However, in BD game, the limit is very large compared with that in GW since the states are not limited to the geometric coordinates but rather complicate. Not only because the number of states increases but also because the possible transitions increased from 4 to 5. Therefore, unlike in GW, a large reward value does practically have more impact on the states that are far from the final state than a smaller reward value. There is one point worth mentioning for PI. At reward equal to 1, there is a jump of run time. If this is not due to computation latency, one possible reason could be that with lower reward value the solution is not optimal and it soon converges only because it does not explore all states and the policy only correctly direct states near the exit location.

The result of discount value are consistent with our previous argument that smaller discount value only takes short term gain into account and as a result the policy after it converges does not direct the agent to the exit location. With higher discount value, the run time and iterations to converge increase significantly, but they do converge to an optimal solution.

Since the visualization is hard to show for BD, one way to look at if the result of VI and PI are the same is that the number of iterations of the converged optimal solution of VI and PI can be found from Level in Table II as 459 and 724. Iterations around these number should tell you that they can reach the exit location at the inner VI last iteration within small number of steps. However, it does not mean that the sequence of actions are completely the same (See Supporting Files).

In summary, small number of states and discount value will yield higher convergence rate. However, small discount value may not yield optimal solution. Depending on how the convergence is determined, a large reward value may have small convergence rate. Without considering how convergence is determined, VI will takes more time and iterations to converge with lower reward value while PI is robust enough to tackle with lower reward value. For the GW problem, large size of world increases the time and iterations that required to converge significantly, especially for PI. PI is more efficient on tackling transitions with success probability. However, it will take more iterations to converge if the states have rare transitions, like to themselves. For the BD problem, higher-level games do not mean harder games, and therefore, a second-level game can have more states than a third-level game. With more states, the run time and iterations to converge are also larger. Varying reward in BD game yields slight different result from GW game. Larger reward value actually yields smaller run time and iterations. This is because each state have more neighbor states and the number of states is also large. The argument for discount value still holds for BD game as in GW game.

LEARNER

We use Q-learning as our Reinforcement Learning (RL) algorithm to solve GW and BD. We evaluate the performance of QL based on two metrics which are Number of Steps Per Episode and Average Reward Per Episode. Through the number of steps per episode metric, we can see when QL converges and converges to how many steps. Through the average reward per episode metric, we can see the increase of reward vs each episode and converges to the reward value of the optimal solution. In this way, we will have a direct view of QL learning from the past episodes. Moreover, we can compare the final reward value with the results we retrieved from the planners. We also varying two parameters to help us understanding QL better including initial Q values and epsilon values.

GRIDWORLD

In the GW problem, we vary initial Q values and epsilon values and run the QL. The initial Q values and epsilon values we use are shown in Table III. The best performance of initial Q value is qInit = 0.3 and worst is qInit = 30. The best performance of epsilon value is epsilon = 0.05 and worst is epsilon = 0.8.The comparison of these two results are shown in Figure 6 and 7. Results of other values can be found in Supporting Files.

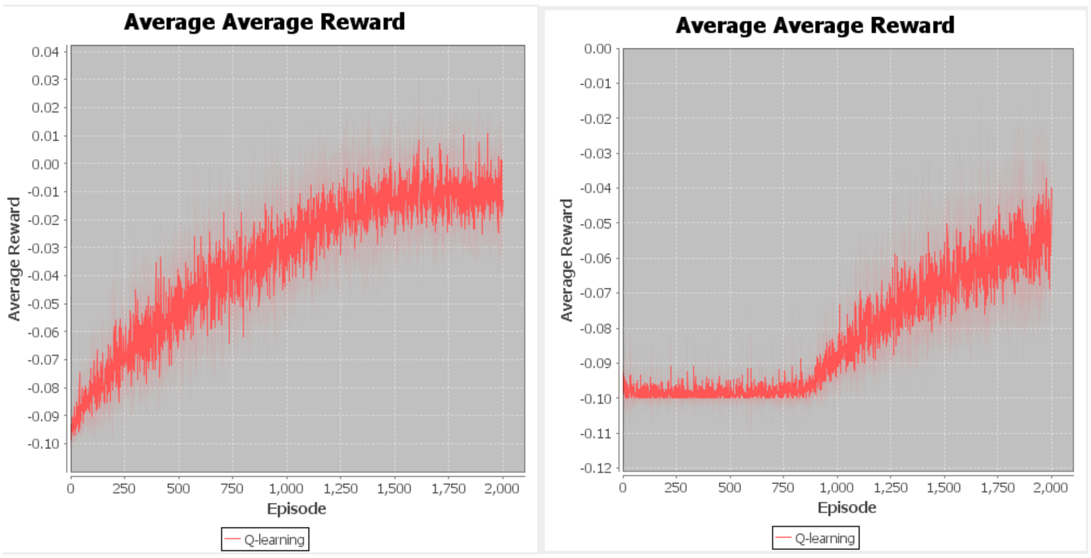


Figure 6. Average reward QL for GW at initQ as 0.3 and 30.

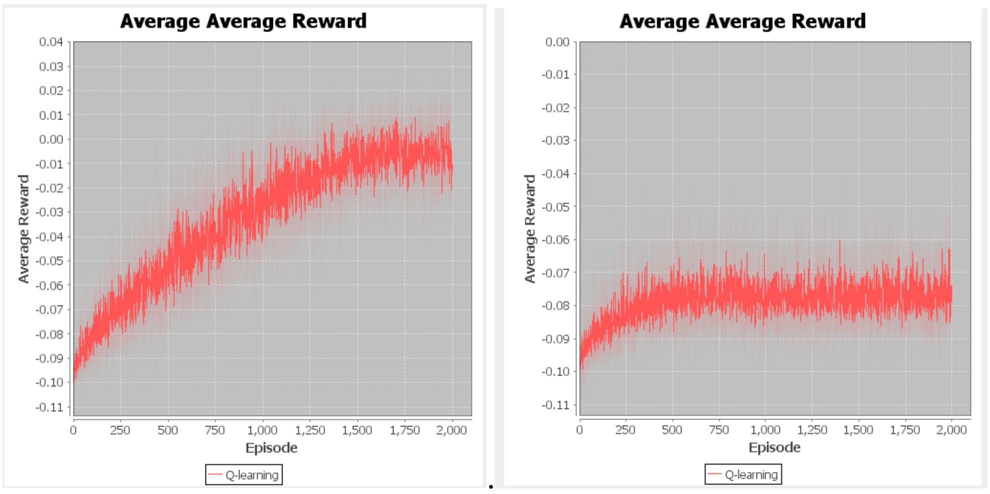


Figure 7. Average reward QL for GW at epsilon as 0.05 and 0.8.

As we know, QL needs random restart to avoid the problems caused greedy action selection strategy. If we do not use random restart for QL, we will mostly use what we know and only use what we know in our calculation because the result based on our learning are almost always better than random walk. We will always choose the same action and therefore soon converges to a local minimal. So we implement epsilon greedy exploration in which we do something like simulated annealing. Epsilon is the decaying factor for cooling down the temperature and initial Q values are just the initial temperature in terms of simulated annealing. With lower epsilon value, QL will choose the action even it is not the best action to take. Lower epsilon value means cooling the temperature slower and do more exploration. While larger epsilon value means doing more exploitation. With a large initial Q values, we are optimistic about the problem and we believe we can explore more before exploitation. In GW, smaller initial Q value yield the best performance. This is because there are not many things need to be explored for this GW. We can quickly find the optimal solution with QL. With a large initial Q value = 30, the graph shows that we do not achieve any improvement from learning before the first 800 episodes. This is because, we haven't finished lowering the temperature and what we are doing before 800 episodes are mostly exploration with random restart. Indeed, we fully explore the problem space, and when the temperature becomes low after 1000 episodes, we start to get positive reward increase back from our learning experience. However, as mentioned before, if there is not much things we need to learn, deep exploration is not necessary and it will only slows down the learning process. Although in the end, we may achieve the same performance but excessive exploration is not efficient.

The initial Q value controls how much we will do exploration before exploitation. With a fixed epsilon value, excessive exploration caused by large initial Q value will slows down the process. On the other hand, the epsilon value actually limits the performance. As you can see from Figure 7, the average reward when converged at epsilon = 0.8 is only about -0.07 which is much smaller than -0.01 with epsilon = 0.05. With epsilon = 0.8, we mostly do exploration and without any exploitation. Since the epsilon value is fixed, we will not have any chance doing more exploration in a new episode. Therefore, the performance is limited by the epsilon value.

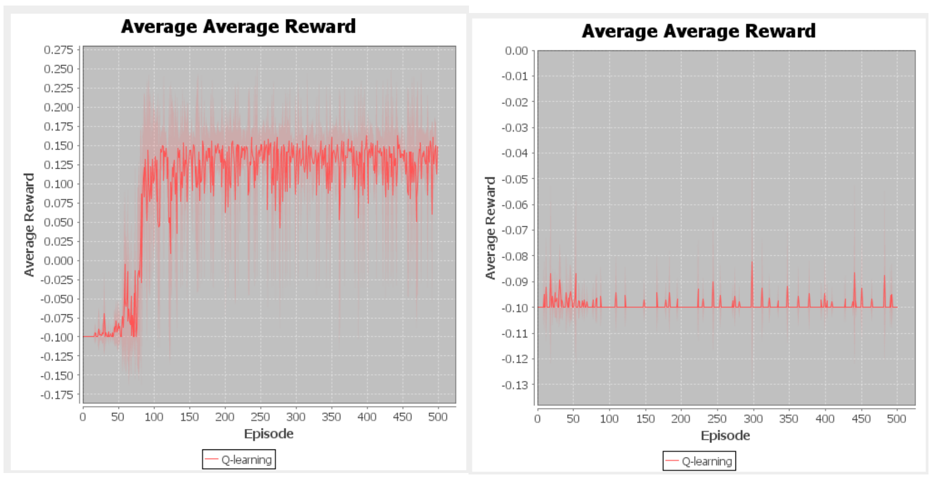


Figure 8. Average reward QL for BD at initQ as 0.3 and 30.

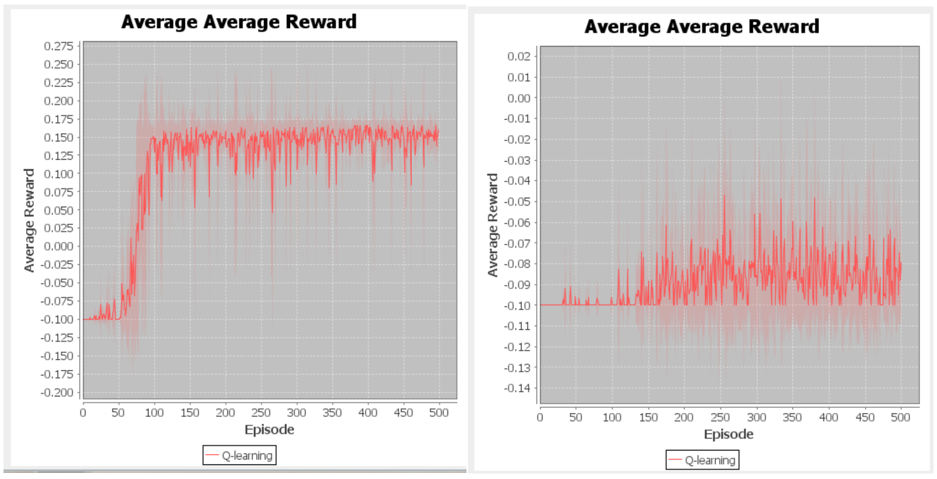


Figure 9. Average reward QL for BD at epsilon as 0.05 and 0.8.

BLOCKDUDE

In the BD problem, we vary initial Q values and epsilon values and run the QL. The initial Q values and epsilon values we use are shown in Table III. The best performance of initial Q value is qInit = 0.3 and worst is qInit = 30. The best performance of epsilon value is epsilon = 0.05 and worst is epsilon = 0.8.The comparison of these two results are shown in Figure 8 and 9. Results of other values can be found in Supporting Files.

The best performance appears at the same positions as it does in GW problem. This means that, both problems do no need much exploration. Therefore, we might guess that the result of QL should also be worse than VI and PI. Actually, this depends on the metrics. If we evaluate the performance in terms of reward, then VI and PI for GW is better than QL for GW. As you can see from Figure 1, most reward values are greater than zero while the result of QL is still smaller than zero. However, if we compare these two methods in terms of run time, then QL is better than PI. As you can see from Table I, II and III, most run time measurements of PI are greater than 10000ms while QL takes less than 5000ms. Therefore, I believe, in terms of GW and BD, VI and PI are generally better than QL in terms of reward, while QL is better than PI in terms of run time. Also as you can see from Table III, higher initial Q value and higher epsilon value have longer run time. Obviously, this is due to excessive exploration with Q value and high epsilon value.

TABLE III

GW AND BD Q LEARNING RESULTS

GW	qInit	0.3	0.5	1	5	30
	Runtime (ms)	1955	1835	1791	2799	13573
	epsilon	0.05	0.1	0.3	0.5	0.8
	Runtime (ms)	1938	1838	1949	2383	4456
BD	qInit	0.3	0.5	1	5	30
	Runtime (ms)	802	682	673	819	3292
	epsilon	0.05	0.1	0.3	0.5	0.8
	Runtime (ms)	599	724	1007	1549	2729