Assignment 4: Reinforcement Learning

[Mingrui Sun]
[msun333@gatech.edu]

I. Introduction - Reinforcement Learning and Markov Decision Process

The word Reinforcement Learning was originate in biological study of the famous Pavlovian effect, where one can condition animal to have stronger/more predictable response in a certain environment by giving rewarding/punishment to animal.[JOA10] In typical RL, a agent learn how to behave in a environment (what action to take) by getting reward (can be postive and negative) associate with each action. Rewards getting from the environment can further shaping the policy produce which leads to better overall outcome. The goal of it was to learn a optimal policy (best strategy to behave) in a environment to get the maximum reward. RL is one of the solution to a real world decision making problem like path selection in driving or trading and picking stock. Deep Mind's Alpha Go and Alpha Star are recent example of RL application that beats humans in corresponding games (Go and Starcraft2).[ACT19] In the future RL may play an important role in development of Artificial General Intelligence (AGI).
Markov Decision Process is a class of decision making problem that have formal mathematical definition. It is normally being state as a four tuple (S, A, P_a, R_a) where:
S = Set of state that agent can be in,
A = Set of action in each state that agent can take,
P_a = The set transition probability between each two state, this is to capture randomness in the system
R_a = Set of reward of taking actions in each state
depending on the exact MDP, the state space can be finite or infinite. A policy in MDP would be a function that map S to a series of action (A). In this work, two different MDP was being analysis through the lens of three RL technique (Model based RL: Value Iteration, Policy Iteration, Model Free RL: Qlearning). The two MDP used here are frozen lakes problem and Forest Management for the following reasons:
1)Forzen lake problem from Open Ai Gym is a grid world problem. The goal of this game is to move from S (Start) to G (Goal) without entering to any H (Holes). In addition to H that can terminate the game, movement on the frozen lake (F) is slippery so it have non deterministic property. The player have equal probability of moving in the exact direction and two direction 90 degree near it. What special about frozen lake is that the grid world nature of it make visualizing policy a breeze. Also transition in this game is easy to model so model free method can easily be used. Also Frozen Lake MDP of different size can be easily made so discussion about scalability of each RL technique can be made.
2)Forest Management is a example of a non grid world MDP. In this game, the player manage a forest with two action, cut and wait. Depending on the tree's age when it was being cut or leave till the ends, different amount of reward was generated. At each time step a chance of the tree catching free so there are a trade off between cutting trees early so it doesn't catch fire or harvest/leave it until the oldest age to get more reward. As a non gird world, the strategy in this game is highly depending on the number of states/time horizon of this game. States number impacts on the policy generated can be shown by this game. Also though Forest is a non grid world, the transition is already implemented in MDPtoolbox[Cor] so VI, PI and Qlearning can be easily used on it without much leg work.

II. Forzen Lake

In this section three different Frozen Lake map was generated with a probability of Holes forming being 20% in map size of 4*4 (16 states), 20*20 (400 states). Value iteration, Policy iteration and Qlearning were all implemented using the hiive fork of the MDPtoolsbox in Python. In the version of Frozen Lake used in this work, the reward of getting to the goal is 1 and falling into holes is -1, for each step took that land not in goal/holes a step reward was used. The step reward and hole reward were added to combat the problem of spares reward space in this problem, in the original frozen lake in openAI gym agent don't get any feedback until reaching the goal. Constant feedback was created through this reward shaping measure.

A. *DP Methods*

Dynamic programming (DP) methods value iteration (VI) /policy iteration (PI) was first applied to the MDP problem. For both of them a 0.9 gamma and -0.1 step reward was used. For the 4*4 world, both method returned the same policy shown in figure 1. This policy is clearly the optimal policy. For all blocks around holes it points to direction that is impossible to fall into it account for the slippery nature of the world. Also one thing to noted is that for the winning step before the goal in (3,2) block, the policy is actually points toward the wall instead of directly to the goal as that also prevent falling into the hole. As the map in figure 1 shown, evaluate policy by eyes is only possible in a small world like the 4*4 due to existence of random movement in this game. Policy was further tested through playing 100 runs of frozen lakes which leads to a 100% win rate, this experiment confirms that DP methods do obtain the optimal policy. These two MDP method performance in the 4*4 FL was further examined in terms of convergence time and run time. As shown in figure

2, both DP method converge to a zero error (at iteration 2) with policy iteration converge (at iteration 17) a lot faster than VI. Error here is the utility function(policy utility) difference between last iteration and the current iteration. This difference was due to the fact that in PI update was made in the policy space instead of utility space. In the utility function of each space perspective, PI only require relative order of all state to converge than value of all state to converge. Run time wise VI and PI take 0.5s and 1.8s respectively which shown that PI is more computational expansive than VI, we would see this different expand more as number of states increase.

Switching to the larger 20*20 world, some change was made.

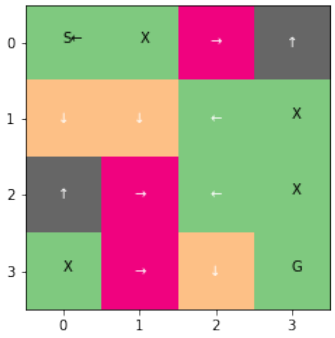

Fig. 1. Policy Map of Value/Policy Iteration on 4*4 Forzen Lake, S,O,G here represent Start, Hole, Goal.

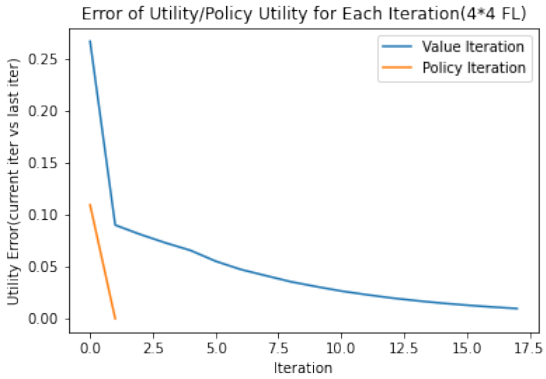

Fig. 2. Error of Utility/Policy Utility During Each Iteration for 4*4 FL

Firstly, the step reward was decrease to -0.1/400. This is a measurement to prevent agents found falling in hole near start is more beneficial than traveling through the world to goal. Secondly, a larger gamma of 0.999 was used to ensure the delay reward of reaching goal get value enough as start to goal distance increase. The converge policy generated by VI and PI were shown in figure 3 and figure 4 respectively. These two policy aren't identical with a few blocks difference primarily on the lower left part of the map which are mostly irrelevant for the path from start to goal. Their performance was further validate through simulation of 100 plays which shown a 22% and 25% success rate. This difference was mostly causing by the randomness of the environment. Currently in OpenAi Gym there seems have a bug with env.step() random seed that prevent a true one to one comparsion. One may fixed this problem by downgrade/upgrade OpenAi gym or overwrite the env.step implementation but it wasnt done here due to time limitation. For the 20*20 world shown here, there are no way to get to the goal 100%. Overall, though tiny difference was shown in these two policy, a similar level of performance was achieved. The error chart of this world was shown in figure 5, PI approach converge(no error) earlier (at 18 iteration) than VI (at around 100 iteration) just like in the smaller world case. In terms of run time, PI (1min 2s) takes longer time than VI (0.9s). Our run time measurement confirms that PI do have higher complexity than VI. According to some study, VI have a time complexity of $O(S^2*A)$ where S and A are the number of states and action in the MDP problem. PI's complexity is highly depends on the the gamma used,exact MDP process it is used on, the terminate method used, for simplex termination PI is $O(S^3*A^2)$.[MS13] This complexity different make VI more scalable compare to PI.

One interesting problem to discuss is the impacts of reward

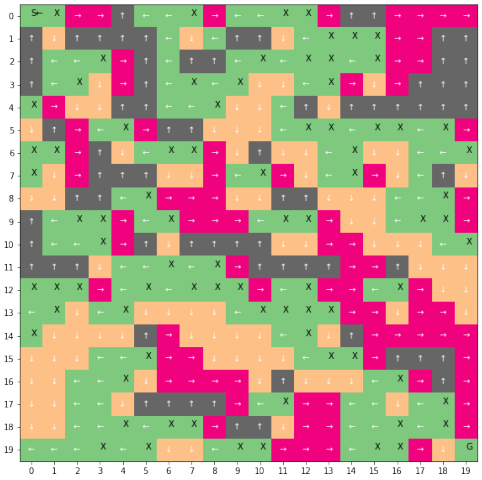

Fig. 3. Policy Map of VI in 20*20 FL

shaping for the 20*20 FL MDP. For FL, step reward was added to punish agent for choosing a longer path and hole reward was added to keep agent from staying away from holes. In the 4*4 world these reward shaping measure don't make much difference as the policy space is too small to have those variant. For a larger world, too negative of a step reward would cause early stopping of the agent. A series of policy for the same 20*20 FL MDP with different step reward was generated on VI. Its error of utility on training was shown in figure 6. It shown that less negative step reward lead to longer convergence until -0.001. In figure 7 the success rate

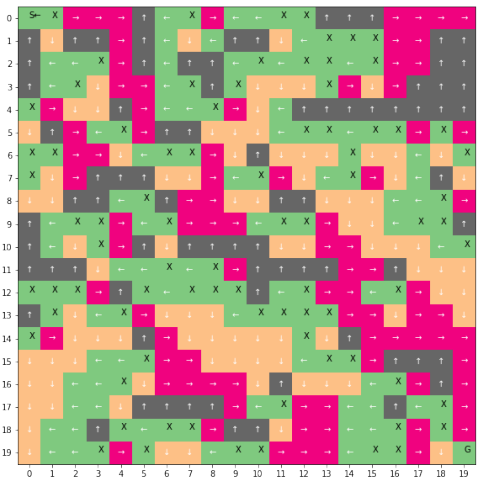

Fig. 4. Policy Map of PI in 20*20 FL

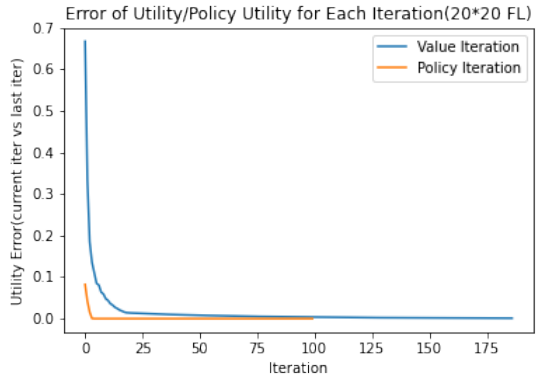

Fig. 5. Error of Utility/Policy Utility During Each Iteration for 20*20 FL

of 100 plays simulation result was shown. At too negative step reward cause zero success rate as agent encounter the problem of optimal policy was to fall into holes early in the game. For step rewards level like -0.001 and -0.0001, the final policy reach a similar level of performance and it is hard to say which is better as only 100 plays was simulated. Result from figure 6,7 also confirm that the -0.1/400 step reward used in this work is appropriate for the 20*20 FL MDP. The process of fine tuning reward value for each event can be seems as adding domain knowledge to help the learning process. Just like in supervise learning, the more domain knowledge we can provide to our learner the better it would performed assume those domain knowledge are true. One more factors that is important in MDP is the selection of discount factor gamma which control the preference between short term reward and long term reward. Low gamma would produce policy that only care about the reward in the last stage when high gamma(close to 1) the policy would have more weight on long term reward.

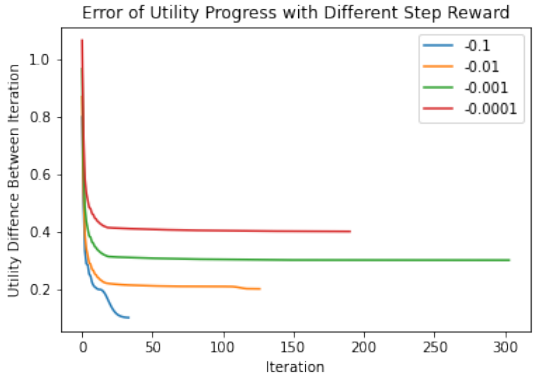

Fig. 6. Step Reward Impact on VI Utility Progression for 20*20 FL, a offset of 0.1 was added for each series to shown its shape.

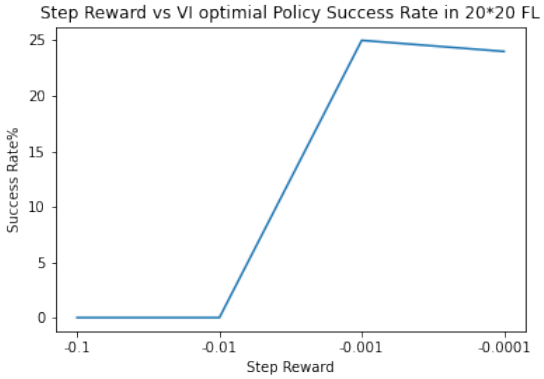

Fig. 7. Success Rate Under Different Step Reward

For 20*20 FL MDP particular, the only positive reward is the reward of reaching the goal on the other side of the map which is a long term reward that matter so a larger gamma should be used to put more weight on that. Gammas impact on VI's training and optimal policy generate were shown in figure 8,9. From 0.1 to 0.999, increase in gamma result in longer converge for VI, but it drop back once passed 0.999. Policy success rate wise, the success rate peak at 0.999 gamma. For the 20*20 FL, a larger gamma was more preferable like explained before, but too high of a gamma like 0.9999 also would result in sub part performance from VI generated policy. Just like the reward assignment, a reasonable gamma selection also added domain knowledge that would help the learning process.

B. Q-Learning

Thanks to our previous experiment carry out on 20*20 and 4*4 FL MDP through DP method. In this section, we reuse setting of gamma and reward setting in this section. First the Qlearning method was applied to the large 20*20 FL world to solve it. Unlike DP method that have convergence requirement built in, most qlearning implementation run until the max iteration value is meeted, the user have to decide whether the policy is good enough or not to decide whether further training is needed. For our case of the FL MDP, policy

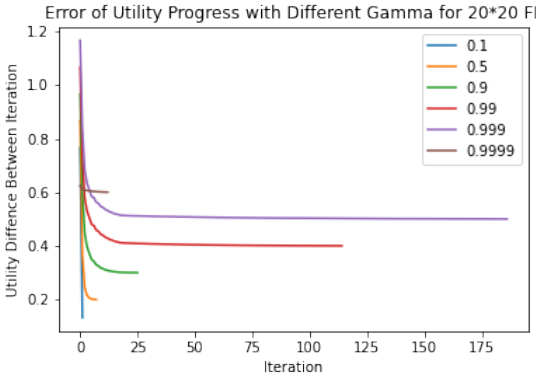

Fig. 8. Gamma Impact on VI Utility Progression for 20*20 FL, a offset of 0.1 was added for each series to shown its shape.

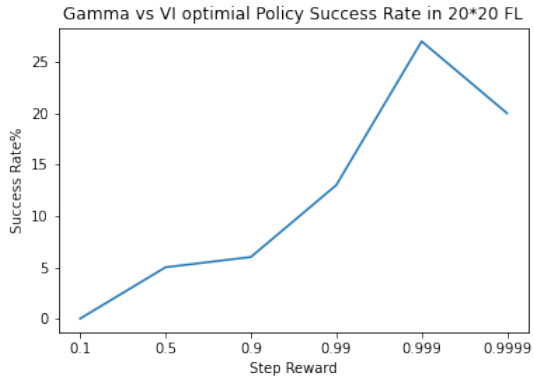

Fig. 9. Success Rate Under Different Gamma

wise we only care about the success rate of the agent getting from start to goal so we used the policy success rate as the convergence test. For the 20*20 world, a 25% success rate level was used the stopping criterion of qlearning, the success rate only being access per 100000 episode of learning. Using a learning rate alpha of 0.1 and a high alpha decay value 0.999999, a qlearning optimal policy was generated for 20*20 FL. The success rate evolution in the qlearning training process was shown in figure 10. The success rate fluctuate around 5% to 20% level for most time during its life time can stop in the end peaking at 28%. This shown a problem with qlearning that is the training towards a stable q-agents with stable performance show no sign of convergence. The whole training process of this qlearner takes 15min 19s which is almost one magnitude longer than the PI implementation shown before, but performance wise the policy basically about the same (28% vs 26%). One interesting to note is that a 0.9 alpha decay was also tried before in the 20*20 world but it runs forever and ends up reach 26% success rate in 1h 50 mins which is too long for the taste of this project so a larger alpha decay was finally used. The policy map of qlearning was shown in figure 11. Comparing to figure 4, majority of the difference happend in the upper right and the bottom left region which most of the time are irrelevant for the optimal start to goal path (about the diagonal of the map) so their performance happened to be similar.

Why qlearning is so much a pain for FL MDP compare to

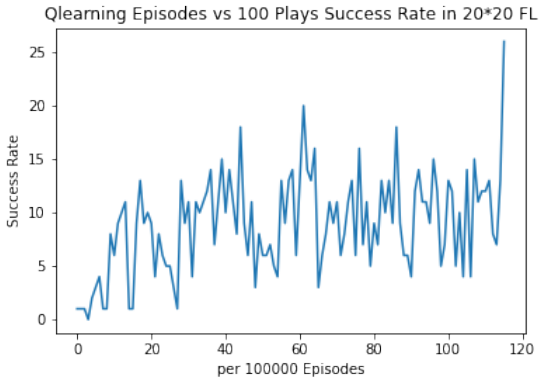

Fig. 10. Success Rate of 100 Plays During Qlearning Training in 20*20 FL

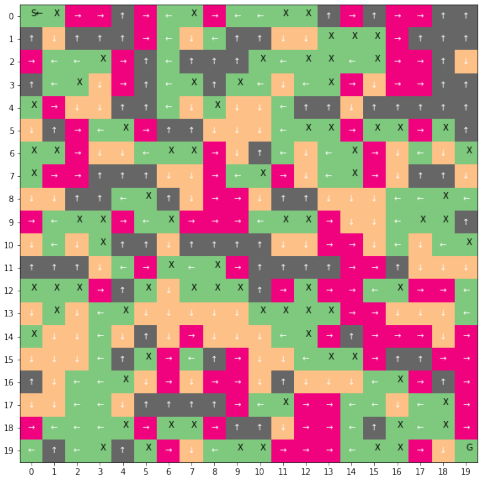

Fig. 11. Optimal Qlearning Policy for 20*20 FL

our DPs solver? In terms of training time, a lot more time was used without much improvement. The simplest answer would probably be the model free features of the qlearning. Model free learning is great as no model involve so it can almost due with any problem whether the model in those MDP are well defined or not. Also model free ensure the learner don't have inherent bias from the current model of the MDP. But it also means that qlearner dont get any boost from domain knowledge contains from those model. For a problem like FL, the model is well defined so DPs can easily get a good solution with those domain knowledge help. So in the ends it take qlearner longer to generate a good enough policy as it don't know all transition/rewards of each states

from the start but need a lot of experience in the world to learn those information. There are trick to combat this problem, one is to initialize the qtable using a good heuristic function. For example for the case of FL, q-table of each block can be initialize with the city block distance between the block to goal. In way domain knowledge of how the FL world is become avaible right from the start. Unfortunately in MDPtoolbox implementation of Qlearning there are no easy way to perform this as it only support initalize with 0 or random number.

Due to long run time requirement of qlearner, it is unrealistic to discuss the impact of each hyperparameter in a large world like 20*20, so for this project we did that in the simple 4*4 world. A 100% success rate level was used as the convergence check for the q-policy produce. Using a 0.9 gamma and default qlearner setting, a optimal policy for 4*4 world was produce. The policy map wasn't shown here as it is exactly the same as in figure 1. The training ends up take 7.6s with success rate progress shown in figure 12. For the 4*4 case we can a better progression of success rate improve to 100% which cant be improve anymore but it still have some of the problem discussed for figure 10. If we don't know that a 100% success rate for 4*4 FL is possible then it is really hard to decide that level as the stopping level. The qlearning ended up converge at 1340k episode which is a lot higher compare to VI or PI. Each qlearning episode is a lot cheaper than VI or PI (per iter) but far more episode was required to reach a similar performance.

One big aspect in Qlearning is the trades off between explore

Fig. 12. Success Rate of 100 Plays During Qlearning Training in 4*4 FL

and exploit. So explore means take random action so a large portion of our states map can be experience. Exploit means take we take advantage of the currently qtable so we only pick the actions with the largest q value. For the epsilon greedy qlearning used in this work, this trades off was control by the value of epsilon and its decay rate. With a high epsilon, the agent are more probe to take random action than exploit the qtable. The epsilon decay slowly which generate more exploit as learning progress. In a perfect world, we would perfer the qlearner to do all explore until the qtable is ready to be exploit which the epsilon greedy process simulate this. The impact of initial Epsilon on the training process was shown in figure 13. The 0.3 epsilon setting was clearly the best in this game as it require only 10k episode to converge. A smaller epsilon than 0.3 cause too early exploit which leads to longer convergence. A large epsilon than 0.3 cause over exploring which also leads to longer convergence. But the good news is that thanks to the small size of the 4*4 FL, all epsilon setting reach the same optimal policy. This wont be the case for a larger state problem as a not appropriate epsilon value can leads to exponential longer run time which make qlearning unrealistic for those case. The impact of Epsilon decay was also tested in this world with a fix epsilon 0.1 in figure 14. Unfortunately not much information was generate in this case as the 4*4 FL is not complicated enough to shown difference between those decay level. The same discussion on the 20*20 world wasn't done due to time constrain. Overall the explore/exploit trades off tuning with epsilon/decay is essential in qlearning as this process also are adding domain knowledge of the problem into the learner.

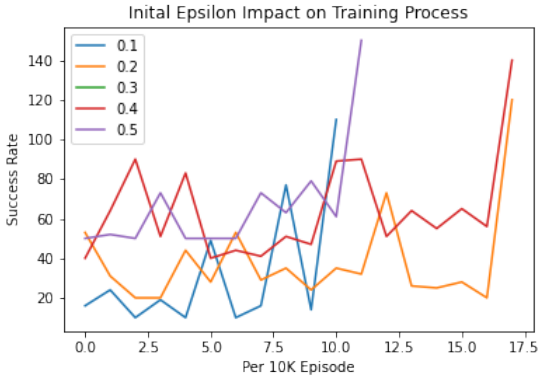

Fig. 13. Starting Epsilon Impact on the 4*4 FL Qlearning Training Process, a offset of 10% was added between each series. Noted the 0.3 Epsilon converge at the first 10k episode so it was not shown here.

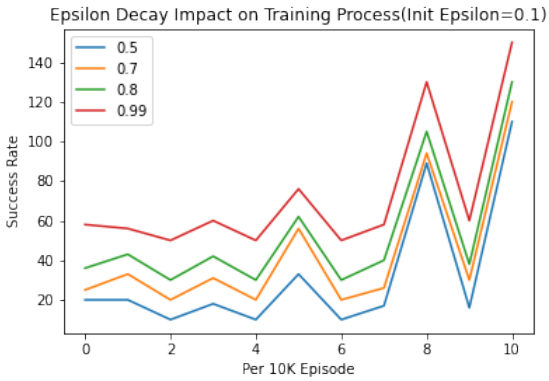

Fig. 14. Epsilon Decay Impact on 4*4 FL Training Process

III. Tiny Forest Management Problem

In this section, Forest Management Problem of three different size was solve with VI, PI and Qlearning. In this game,

tree have small stage: youngest (first year), mid-age (every year in between), old-ages(the last year). The reward of cutting tress are 0,1,2 for these three stage respectively. The reward of waiting only occur when trees are at the old-ages which is 4. At each time step a chance of 10% that the tree would catch on fire to go back to youngest stage. Overall objective of this game is to get the most amount of discounted reward by the ends of the game. Size of 3, 10, 100, 1000 was discussed in this section.

A. *DP Method*

The S=3 Forest MDP utility function progression for VI and PI was shown in figure 15. The VI curve shown a clear sign of convergence at 71 iteration as the mean of value function approach plateauing. For PI it converge in 2 iterations so the plateauing is harder to see but the mean of value function approach similar level to VI. Actually both method reach the same policy that is "never cut" (0,0,0), which made perfect sense as for S=3 the largest reward can be generate is to wait for tree to growth into the old-ages. The only other option would be cut tree at the mid-ages to get 1 reward which would be 0.95 after discount in the ends, which made no sense after adjust for fire risk in the second stage. One interesting things to note is the runtime difference between these two method, in this case PI (0.1s) actually solve it faster than VI (0.2s). This is due to the small policy space of this MDP which there are only 2*3 policy in this case compare to a 3*3 value spaces. So it is not always the case that PI is more expansive than VI. Same procedure was carry out on the S=10 Forest MDP. PI (0.0028s at 9 iteration) show faster convergence and smaller run time compare to VI (0.0032s at 72 iteration) in figure 16. Just like in S = 3, both method built a "never cut" policy.

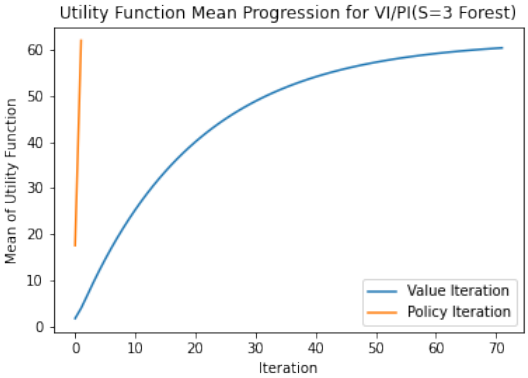

Fig. 15. Utility Function Progression of VI/PI(Forest S=3)

Things changes as S approach a large number 100 as shown in figure 17. Though PI still converge at lower iteration(13 vs 72), runtime wise VI start to outperform PI as policy spaces increase (0.0043s vs 0.00743s). Policy wise both method converge the same policy. This optimal policy is basically start cutting every year since the second year and stop until the ends for the last 13 years. This policy sense make sense as if we

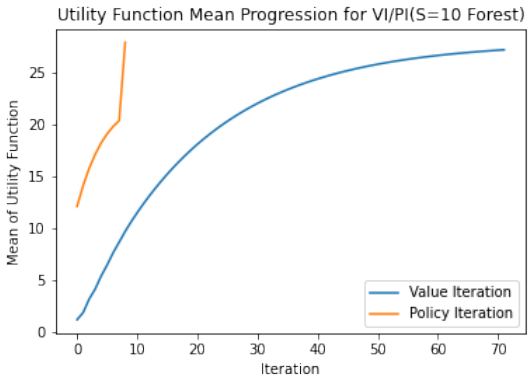

Fig. 16. Utility Function Progression of VI/PI(Forest S=10)

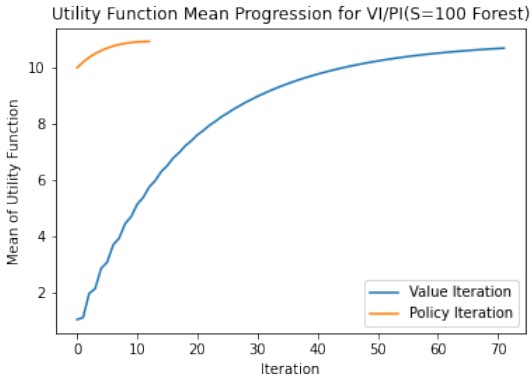

Fig. 17. Utility Function Progression of VI/PI(Forest S=100)

want to cut, the best day is actually at its second year old to get the reward instead of doing it at the old-ages. When approach near the ends of the horizon, the problem swift back to our S=10, S=3 case that never cutting to get the full old ages reward become more preferable. Following the same logic, any S larger than 100 would give a optimal policy of cutting from the second day to the last 14 day. This theory was tested on the S=1000 case, and both VI (converge at 72 iteration in 0.01s) and PI (13 iteration in 0.01s) produce this exact policy as predicted.

The length of the never cut periods in the optimal policy probably have something to do with the discount rate gamma used in this game. If discount rate is high enough, it make more sense to extend this period as the later old-ages wait reward won't experience much discounts compare to the earlier cut reward. Gamma impact on the no cut period length was shown in figure 18 using the S=1000 case, which confirm our expectation before. High gamma do leads to longer no cut period.

B. *Qlearning*

Since optimal policies were already generate in the last section for S=3 to S=1000, qlearner here could used those policies as the stopping criterion. Qlearner converge run time

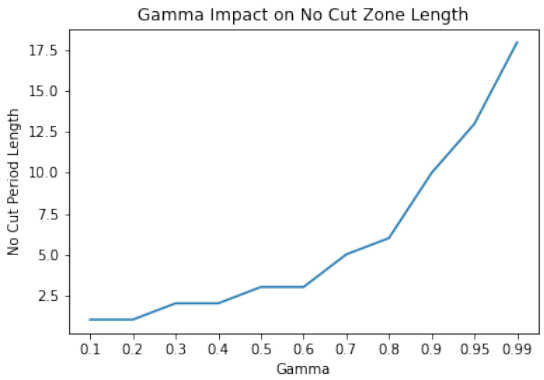

Fig. 18. Gamma Impact on the No Cut Period Length

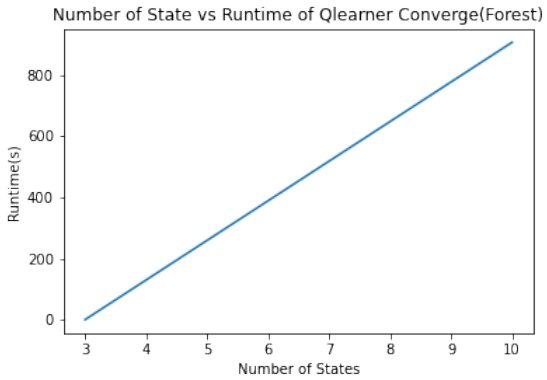

Fig. 19. Qlearning Converge Time at S=3 and S=10(Forest with MDPtoolbox Default Qlearner Hyper-Parameter)

in S=3 and 10 was shown in figure 19. At S=10 it already took the default qlearner 15min to get to the optimal policy. For S=100, attempt was made using the same setting but qlearning run time excess two hour and have to stop the process mid run due to time limitation of this project. This indicate similar problem we discussed before for qlearning: Qlearning lacks the boost of domain knowledge from the start, this leads to longer experience time for qlearner to get to the same level compare to model base methods. Though forest MDP is a non-grid world problem, the model describe it is also well defined so model based approach here would out perform model free approach. Unfortunately the runtime discuss here have to stop at S=10. Progression of mean Q value for S = 10 was shown in figure 20. Though the policy converge to optimal policy, the means Q value show a sudden jump to higher at the end. This showcase a problem with using mean Q as a sign of convergence as policy may converge earlier than it so further training are just a waste of time. This is show case in figure 21 which 1200K more episode (23s more run time) have to be added to show the clear convergence of mean Q value. For S = 10 this added training is bearable but would be a problem as Forest problem get larger. Policy reward based stopping criterion was deeded to be superior than observing roll over of q value means. Three approach can be used to make the forest problem more manageable for Qlearning at high S number: 1) Play with the initialization of Q-table, currently Qtable of all state/action pair was initialize with 0. Changing the starting Q-value of some of those cutting period to 1 maybe help with the training process. 2) Fine tuning hyper parameter of the qlearner used to achieve more efficient explore/exploit balance. 3) Implement more advance Qlearning methods like Dyna-Q which utilize hallucination to learn from previous experience between each update. The forest world is pretty predictable so this approach may work as previous experience do have good predictability for the future. The second option was discussed using the S = 10 Forest Problem in the next paragraph.

The impact of learning rate alpha on convergence time was first shown in figure 22. Too high (0.1) and too low (0.6) of a training rate both result in more training episode. A

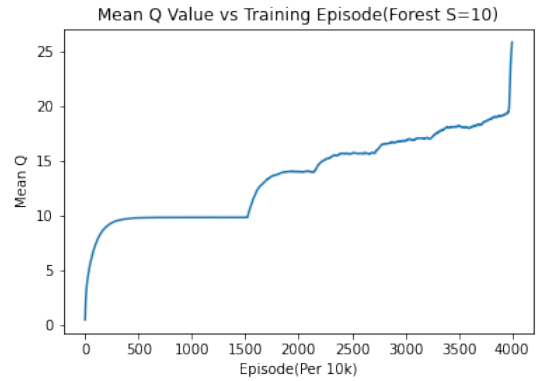

Fig. 20. Mean Q Value Progression in S=10 Forest

training rate of 0.5 (265s) was found to be favor about for this problem. Too little of learning rate would cause all learning take small steps that leads to long training time. Too high of a learning rate would also be unfavorable as random feature like wildfire also take longer to converge as the learning overshoot compare to the weighted means of the true Q value. Changing to 0.5 alpha make a huge lap from 900s training time to 265s, but still it was embarrassing compare to DPs method shown in this work. On tops of tuning alpha, the initial epsilon impact was also tested and shown in figure 23. For the range of epislon tested, the training time requirement increase as epsilon increase, we confirmed that the current used of 0.1 epsilon was favorable. But overall these fine tuning didn't offer the extent of over time improvement that make solving S=100 or 1000 possible in limit time like 20min. So for this section discussion have to stop at S=10 States.

IV. Conclusion

In this work, three kinds of MDP solver was used on two different MDP problem (grid vs non grid). In both of those case, DPs base method perform better than Qlearning methods as its' convergence can be easily shown and also much lower run time was achieved. In terms of the difficult of non-grid

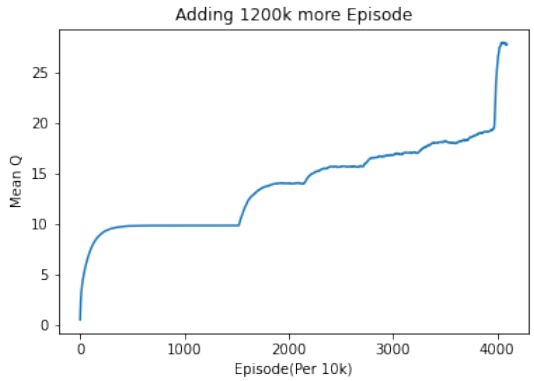

Fig. 21. Adding More Training!

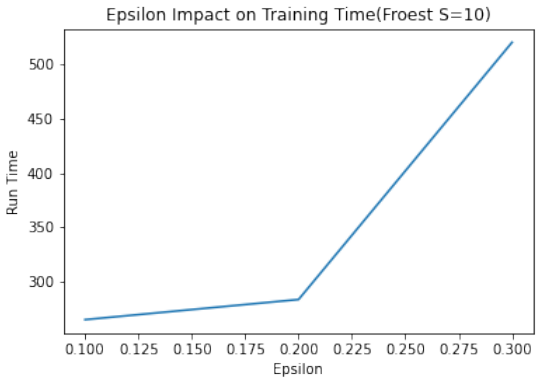

Fig. 23. Epsilon Impact on Training Time(Forest S=10)

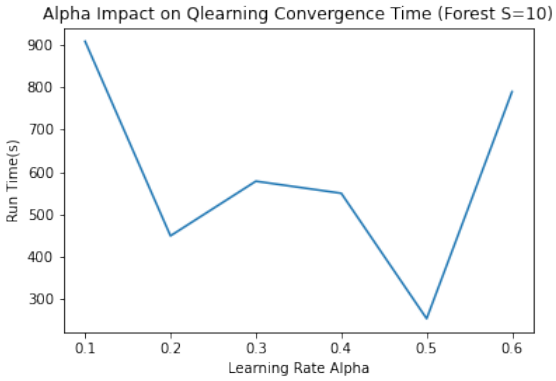

Fig. 22. Learning Rate Impact on Training Time(Forest S=10)

world vs grid world, for DPs method the forest non grid world problem are more easy to solve runtime wise compare to the grid world forzen lake problem. But for qlearner the non grid world(forest) was more difficult as the large state version cant be solve in reasonable amounts of times. In theory qlearning should do better in non grid world problem than DPs' methods. This is probably due to limitation of MDPtoolbox's qlearning implementation. This also showcase the needs for tricks in qlearning like smarter qtable initialization, Dyne q hallucination, fine tuning explore/exploit trades off. Also failure of qlearning was mostly due to the well define nature of the environment model in both games. Comparing DPs's method, we found that VI almost always have lower run time compare to PI except when number of policy space was low enough. It would be useful to always try both method and compare the result policy to ensure a true policy convergence. Overall, this work shown the robustness of DPs based method in solving MDP.

V. References

References

[JOA10] Thomas Jaksch, Ronald Ortner, and Peter Auer. "Near-optimal Regret Bounds for Reinforcement Learning". In: *Journal of Machine Learning Research* 11.51 (2010), pp. 1563–1600. URL: http://jmlr.org/papers/v11/jaksch10a.html.

[MS13] Yishay Mansour and Satinder Singh. "On the Complexity of Policy Iteration". In: *arXiv e-prints*, arXiv:1301.6718 (Jan. 2013), arXiv:1301.6718. arXiv: 1301.6718 [cs.AI].

[ACT19] Kai Arulkumaran, Antoine Cully, and Julian Togelius. "AlphaStar: An Evolutionary Computation Perspective". In: *arXiv e-prints*, arXiv:1902.01724 (Feb. 2019), arXiv:1902.01724. arXiv: 1902.01724 [cs.NE].

[Cor] A W Cordwell. *MDPtoolbox*. URL: https://pymdptoolbox.readthedocs.io/en/latest/api/mdptoolbox.html.