# CS 7641 Machine Learning: Project-4

[Arti Chauhan: Apr-21-2018]

## ABSTRACT

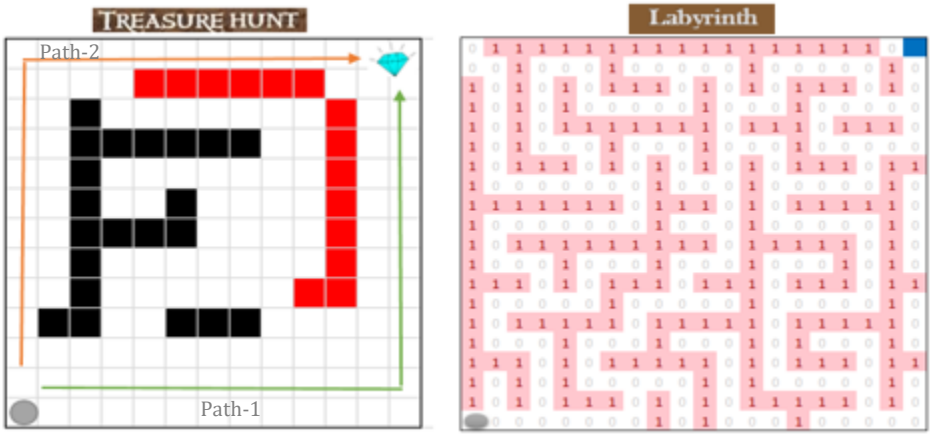
This paper presents analysis of two problems 'Treasure-hunt' and 'Labyrinth', using reinforcement learning algorithms namely Value Iteration, Policy Iteration and Q-Learning.

## PROBLEM INTRODUCTION

Two problems that I have chosen for this assignment model a Treasure Hunt (TH) and a labyrinth. Both these problems were designed to bring out differences in behavior of above mentioned RL algorithms in face of a) different state-space size b) different reward structure. At heart, both these problems are grid world problems and explored as Markov Decision process (MDP) for Policy and Value Iteration.

**Treasure Hunt:** This problem is modeled as 13x13 grid, with a total of 169 states. Black blocks represent wall that agent can't go through. Hitting the wall doesn't cause any penalty. In contrast, going through a forbidden area represented by red blocks results in heavy penalty (-40). Goal of the agent is to reach the terminal state (treasure – top right) with minimal steps, avoiding forbidden area. In this grid world, reward for each state other than wall and forbidden area is -1, which encourages agent to accomplish the task in minimal steps. Agent get as reward of 100 when it reaches terminal state. Agent's actions are probabilistic to emulate real world behavior, where probability of Agent going in intended direction is 0.8 and probability of going other three direction is 0.067.

**Labyrinth**: Second problem that I have chosen is a complicated maze, modeled as 20x20 grid, with a total of 400 states. All states except obstacles (represented by red line) and terminal state have a reward of -1. Agent collects a reward of 100 on reaching destination (top right). Transition probability for Agent moving in intended direction is 0.8 and remaining 0.2 probability is split equally in other three directions.




**Why are these problems interesting?** These problems are of great interest because they emulate several real world challenges.

1. For example treasure hunt grid world can be thought of as a rescue mission where red block represents enemy line and holding a hostage at terminal point. Agent's mission is to rescue hostage, avoiding obstacle and enemy fire at all cost.
2. A different take of treasure hunt can be a floor-plan design for retailers like Macys or Ikea where they want customer to interact with their product as much as possible and maximize time consumer spends in their store. In above setup, there are multiple paths to reach the target state (exit) but retailer can design floor plan such that it puts more rewards on the route it wants consumer to take, there by incentivizing consumer for taking longer path than taking shortcut to exit. On these routes, it can display products it wants consumer to experience or buy.

3. Labyrinth has practical application in traversal of unfriendly space (battle field) or exploration of unchartered territories, where one must find most efficient route to reach target, while overcoming impediments.

## IMPLEMENTATION

Implementation of these MDPs was done using existing Java code [1] and Burlap RL library [2]. Burlap APIs provide several parameters to tune aforementioned RL algorithms. Below I have described some keys parameters that were used to explore behavior of all three algorithms in terms of convergence, steps taken, reward accumulation, computation time and policy evolution.

Following parameters were explored for both problems. Values explored is summarized in table on right.

Table-1

| | Value/Policy Iteration | Q-Learning |
|---|---|---|
| Reward Function | 100 on reaching target | |
| Cost Function | -1 per action | |
| Discount Factor | [0.45 , 0.75 , 0.99] | |
| Transition Probability | 0.8 | |
| Utility/Policy delta | 1.00E-06 | 0.5 |
| MaxIterations | 200 | 3000 |
| Learning rate | | [0.1 ,0.5, 0.9] |
| Epsilon | | [0.2, 0.6, 0.9] |
| Q-Init | | [-50, 0 ,50] |

1. **ϒ (Discount factor)** dictates how much Agent values future rewards. High value of ϒ leads to Agent giving more value to future rewards. When ϒ=0, Agent is short-sighted and makes calls based on current rewards only.
2. **Convergence criteria**: Algorithm terminates when either max iterations are reached or following occurs
   - VI : max change in value function(latestDelta) between consecutive iterations is < 1e-6.
   - PI : max change between policy evaluations (lastPIDelta) between consecutive iterations is < 1e-6.
   - QL : change in 'maxQChangeInLastEpisode' (averaged over past 10 runs) is < 0.5.

   Max-Delta was intentionally kept low to allow enough iterations to bring out any difference in algorithms for two problems.
3. In addition, following set of parameters were explored for Q-learning.
   a. **Learning rate α** allows Agent to determine to what extent recent information overrides old information.α =0 makes the agent learn nothing (exclusively exploiting prior knowledge), while α =1 makes the agent consider only the most recent information (ignoring prior knowledge). It is analogous to step-size. For this assignment I used constant α so that I could compare the performance but in practice I would use dynamic α**,** which decays over time**.**
   b. **Q_Initial** value is the value assigned to all states before first update occurs. High Initial values (Optimistic initial conditions) will encourage Agent to explore. When initial value is low, all states are less viable and Agents puts on 'exploitation' hat.
   c. **Epsilon ε** – it allows control of exploration vs. exploitation. It allows Agent to select random actions with probability ε and greedy actions with probability 1- ε. Ideally, I would have preferred to allow ε decay over time so that exploration is favored during initial phases and exploitation in latter part of the runs but kept it constant for meaningful comparison between different values of ε.

Please note that in rewards, steps, time and Δutility/policy charts in sections below, lines for all three algorithms don't extend all the way to the end (max-Iterations) because '*change in utility/policy*' criteria is met before max-Iterations is reached and hence algorithm terminates. I chose to display it this way so that reader can easily infer from the graph iteration at which convergence was achieved.

## 1. TREASURE HUNT MDP

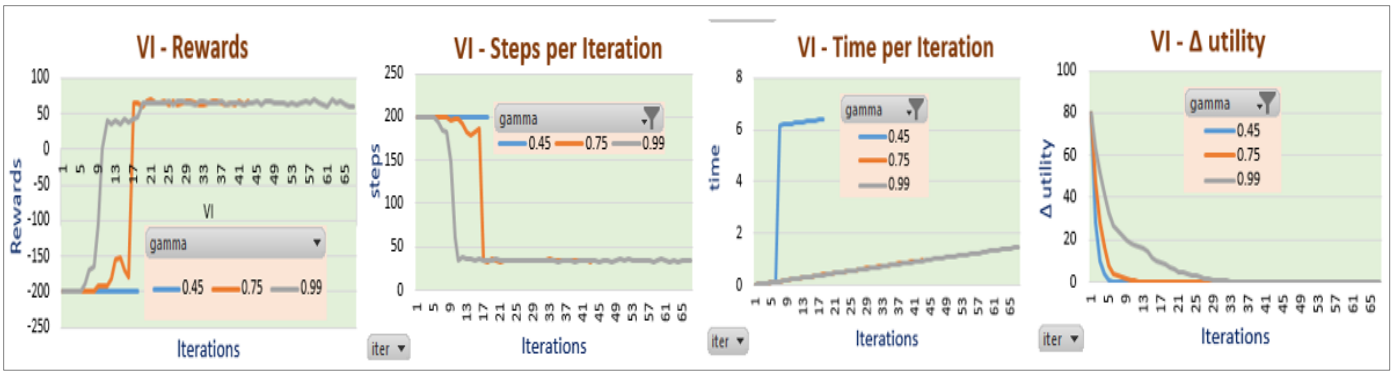
### 1.1 VALUE ITERATION (VI)

Charts below show rewards, steps, time and Δutility per iteration# for Value-Iteration. This information is presented for 3 different discount factors- 0.45, 0.75 and 0.99. Algorithm stops when Δutility < 1e-6 or max-Iterations (200) have occurred.

1. From these charts, it is evident that ϒ=0.45 didn't perform well. In contrast ϒ=0.75 and 0.99 both are able to achieve optimal rewards. With small value ϒ, Agent is myopic, not valuing future rewards as much and giving more weight to current rewards. Hence, not able to arrive at optimal policy. From rewards chart, we

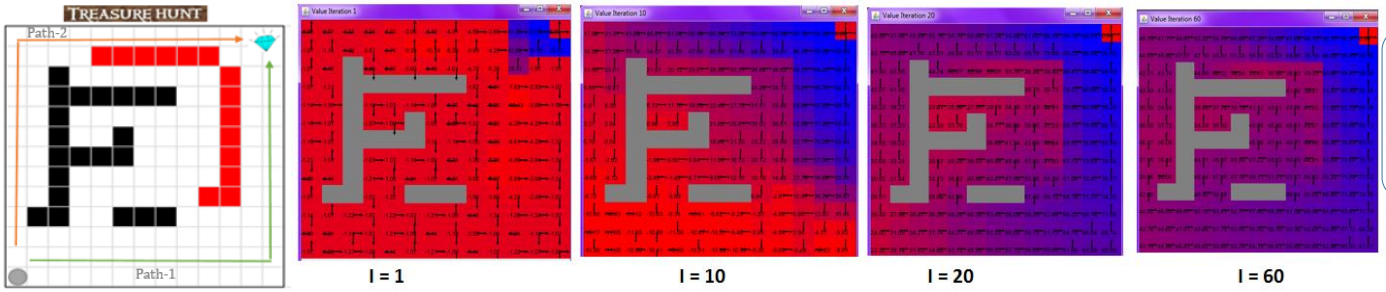
see that it stays stuck at around -200 for a long time (blue line), leading Δutility to fall below 1e-6 and thus causing algorithm to terminate.

2. For ϒ=0.75, 0.99, steps per iterations falls significantly within first 20 iterations, leading to increase in reward# (Recall every step costs agent -1).However, ϒ=0.75 converges faster than ϒ=0.99. This intuitively makes sense.We don't want ϒ to be too high or too low. A happy medium here is ϒ=0.75, where convergence occurred at I=30.
3. Time increased linearly with number of iterations for ϒ=0.75, 0.99, which is expected because of Bellman equation has a constant complexity.




Initially, utility of states is not known to algorithm. Using immediate reward at the terminal state (top right), VI computes utility of the nearest states based on discounted reward. This step is repeated until utility of all states are known and value function converges. This working is evident from policy evolution charts shown below.

- When I designed this problem, I purposely added two paths that would require same amount of state transitions to reach target, except that upper path (orange line) has higher chances of falling into red region due to probabilistic transitions when compared to bottom path (green line). Because of this, my expectation was that states around upper path (orange) would have lower utility compared to states on green path and that's what we see from map below.
- Agent rightfully marks middle region with low values. Agent can move diagonally towards target to minimize number of steps, but will incur high cost if it crosses red barrier.




Treasure Hunt: VI policy evolution

## POLICY INTERATION (PI)

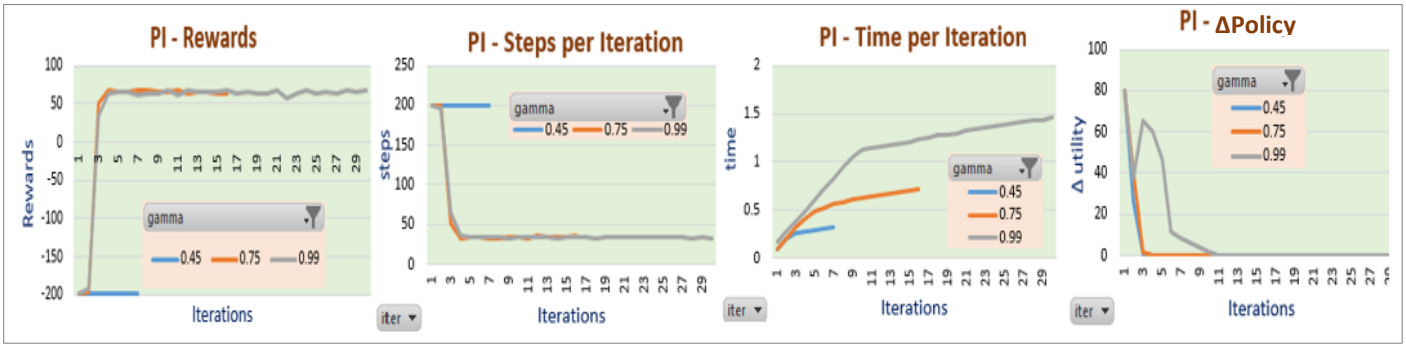
Experiments conducted in this section follow parameters values described in table-1. In addition, max VI iterations to evaluate a policy was restricted to 10. Planning terminates when the maximum change between policy evaluations falls below 1e-6 or max-iteration (200) is reached.

Reward graph below shows that Policy Iteration also converged to same reward (67) as Value Iteration, however it converges much quicker at I=10. In flipside, PI took more time (~2x) per iteration compared to VI. This aligns with my understanding where PI is expected to converge faster because policy can converge much before value convergence. But for each iteration, PI is computing values for all states. Hence more time per iteration.

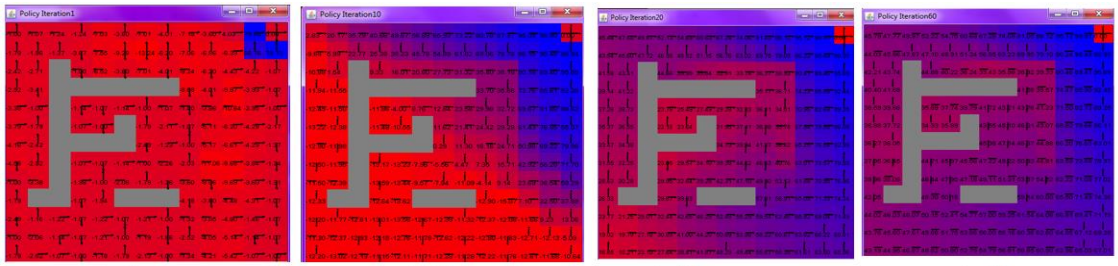
- I evaluated impact of different ϒ values on rewards, time and steps per iteration. Results are shown below. Here again ϒ=0.45 didn't yield good results and this can be tied back to reasons described in section 1.1 above.

- Υ=0.75, 0.99 gave comparable results in terms of reward# and steps as iteration count increases. However, Υ=0.99 shows more variability in Δpolicy in initial runs and higher execution time as iteration# increases.
- Given the data below, Υ=0.75 is the optimal choice in this case, where convergence occurs at I=9 if maxPIDelta is set to 1e-3 and I=15 if maxPIDelta is set to 1e-6.

  This brings up an interesting question –*what's the optimal value for convergence threshold?* More iterations and time is required if convergence threshold is set too low, but it gives more confidence in declaring convergence. So there's a tradeoff. Given results below, I would say 1e-3 is also a reasonable choice for this problem




Policy Evolution -PI starts with an arbitrary policy and evaluates the expected value at all states under that policy. It checks if utility of a given state can be increased by modifying the policy and iterating until there is no more improvement. From chart below, we see that overall policy evolution for PI looks very similar to that of VI, where states on lower path have higher utilities compared to upper path. There is small difference at I=1 (at upper right corner) but by I=10, both algorithms converge to similar policy. It is interesting to note that reward# doesn't change much beyond I=5, yet policy continues to evolve (as seen in I=10, 20, 60 below), having minimal impact on reward#. This illustrates that we don't need to compute true utilities - 'good enough' utilities are sufficient to determine optimal policy.




Treasure Hunt: Policy Iteration policy evolution

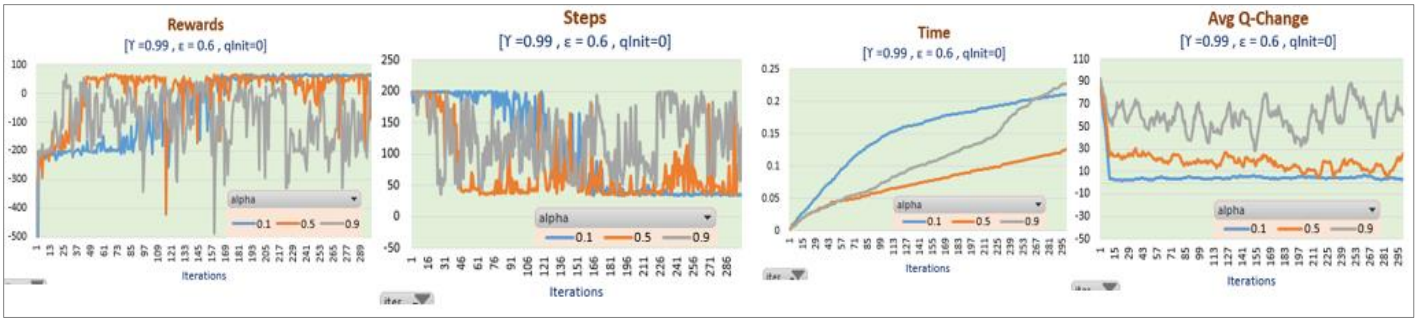
## Q-LEARNING (QL)

In contrast to PI and VI, QL is model free – it doesn't know transition and reward function in advance. It learns it on the fly, by assigning all states a Q-value, and visiting each state to re-assess the Q-value based on immediate and delayed rewards. To explore the behavior of Q-learning, I ran experiments with different values of α, ε, Υ and Q-Init, yielding 256 different combinations to analyze.

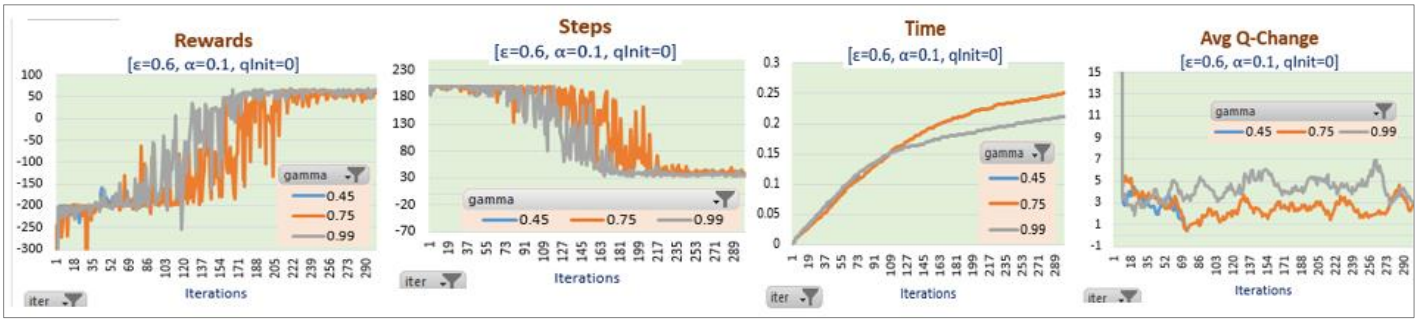
α = [0.1, 0.5, 0.9] | ε = [0.2, 0.6, 0.9] | Υ = [0.45, 0.75, 0.99] | qInit = [-50, 0, 50]

After going through all results, I narrowed down to best set of parameters [Υ=0.99, ε=0.6, α=0.1, qInit=0] for this problem. For the purpose of this discussion, I will use this best set to bring out any potential impact on reward#, computation time and convergence as a function of change in α, ε, Υ and Q-Init.

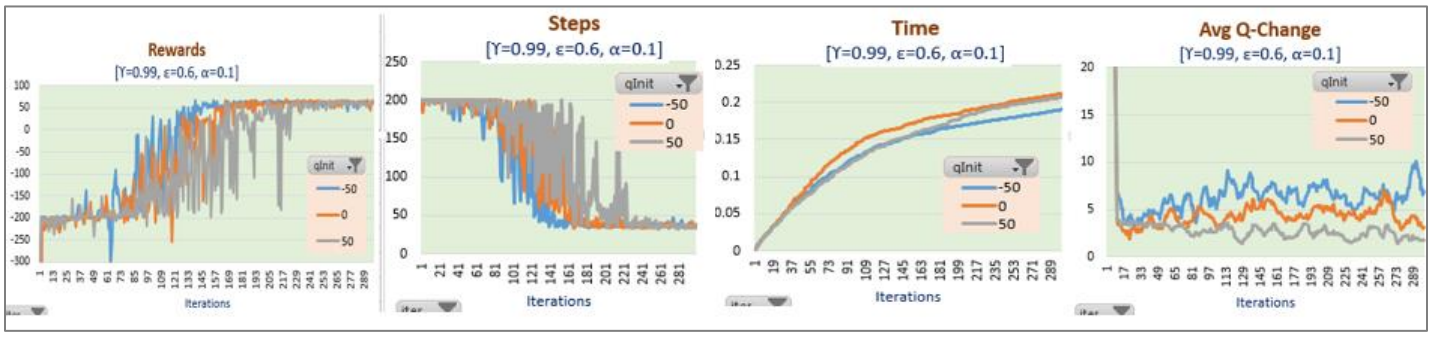
**Impact of learning rate** (α): α is dictates to what extent new information overrides old information. Too small α causes delay in convergence and too big α can cause algorithm to overshoot global optima. Hence it has to be balanced. From graphs below, we can see that with α=0.9 (grey), algorithm overshoots and is not able to converge to optimal policy. With α=0.5 (orange), max rewards is reached at I=65 but with α=0.1(blue) it takes more iteration and hence more time (I=161). However on flipside, average Q-change (rightmost graph) for α=0.5 shows more variability and settles at higher value compared to blue line. Hence, I would pick α=0.1 for this problem.



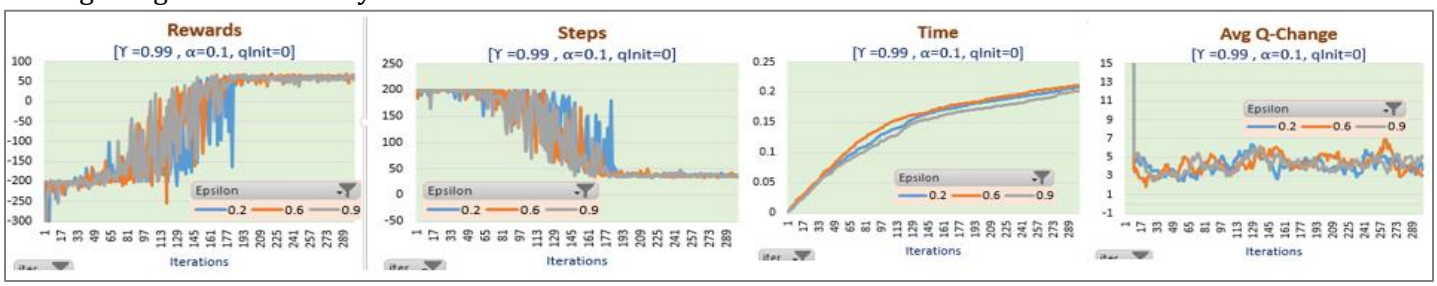

**Impact of Discount factor** (ϒ): As seen for PI and VI also, with ϒ=0.45 (blue), policy was not able to converge within specified criteria. ϒ=0.75 & 0.99 are reasonable choices here but I would prefer ϒ=0.75(orange) as it settles to a lower Qvalue-Change as iterations progressed, compared to ϒ=0.99(rightmost chart)




**Impact of Q-initial:** here I experimented with 3 different Q-Init value [-50,0, 50] to create overly optimistic conditions to encourage Agent to explore and very pessimistic condition to encourage exploitation. With Q-Init=50 (grey), Agent favors exploration and this is evident in charts below-it reaches max rewards little later than for other two Q-Init value, accompanied by huge variations in step# as iterations progress.

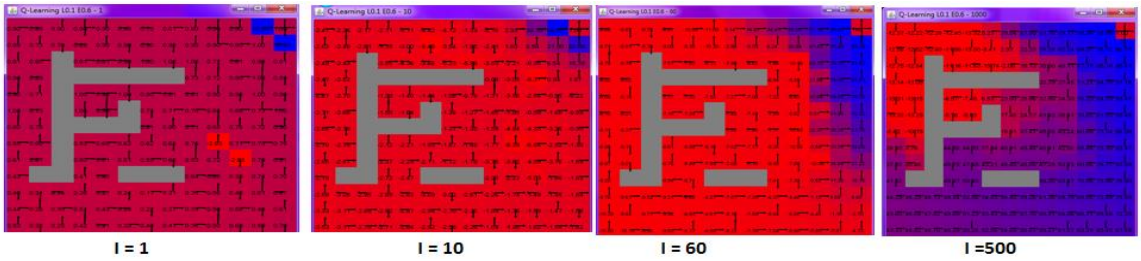



**Impact of Epsilon** (ε): ε controls agent's exploration vs exploitation behavior. For the set of parameters I chose, impact of ε didn't come out as starkly as I would have liked. From charts below, changing ε is not making considerable difference for this problem. But in practice it does, as I have seen it in my other projects for stock-trading using RL and for Labyrinth MDP.



**Policy evolution** for QL looks very different from that of PI and VI. At I=60, policy is still evolving, which is expected as QL is model-free and expected to take more iterations. At I=500, it strongly discourages upper route (more red) compared to PI & VI.

Treasure Hunt: Q-Learning policy evolution

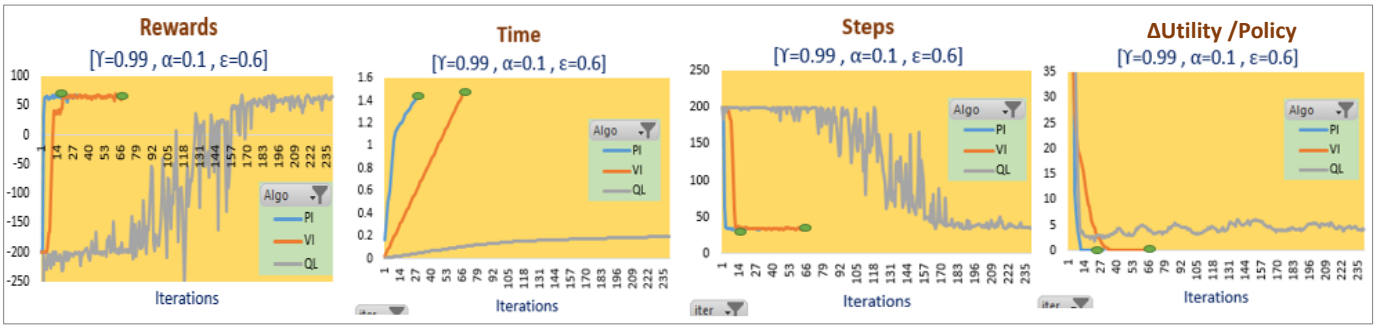



## 1.4 VI / PI /QL COMPARISON – TREASURE HUNT

Charts below compare performance of three RL algorithms for treasure hunt MDP in terms of reward#, computation time, steps and convergence as function of number of iterations, using best set of parameters determined in previous section.

**Key points**

- Green dots show iteration at which convergence occurred for each algorithm. Please note QL didn't meet convergence criteria (<0.5) even after max-Iteration (3000) hence no green dot for QL. However, it is able to accumulate same rewards as PI and VI, which is remarkable as QL didn't have the advantage of knowing the model (transition and reward function of MDP).
- QL takes more iterations to reach max reward compared to PI & VI but time per iteration is low. This is because in each iteration it just hashes action taken and reward and computes the policy.
- PI takes the longest time per iteration (almost 2x as VI) but converges in much fewer iteration than VI. This is expected because policy can converge much before value convergence. PI evaluates the utility of each state/action pair for a given policy, which takes more time to compute.
- As expected, number of steps and rewards are highly correlated. As number of steps taken decrease, rewards increase because with every step agent incurs a penalty of -1.



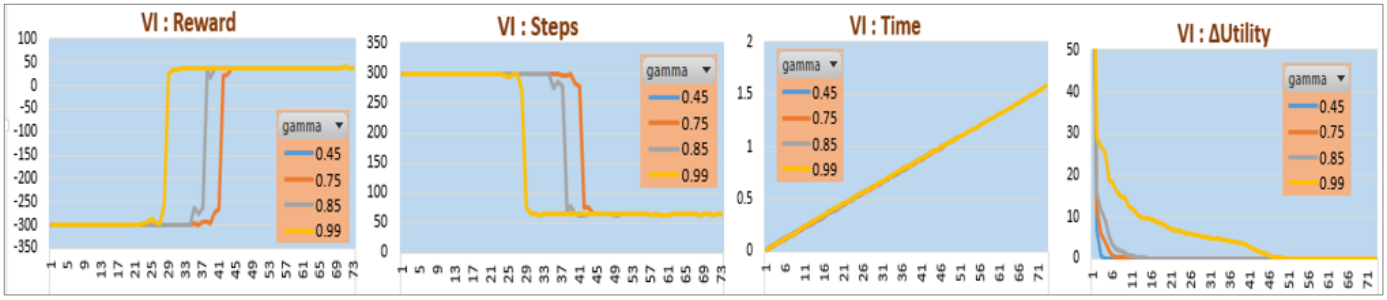

# 2 LABYRINTH MDP

Analysis of Labyrinth MDP follows similar procedure as described for treasure hunt MDP above. Hence in this section I will refrain from repeat it and rather focus on any differences that may arise due to larger (2.3x) state-space for Labyrinth MDP when compared to Treasure Hunt MDP (400 vs. 169). Parameters explored for this analysis are summarized in table-1 (page2)

## 2.1 VALUE ITERATION (VI)

Charts below show reward, time and Δutility for different ϒ values.

- While with ϒ between 0.75-0.99, algorithm performed reasonably well, with ϒ=0.45, algorithm is not able to reach max-reward within specified convergence criteria. That's because with smaller ϒ agent is myopic and gives more importance to current rewards than long term rewards.
- Rate of convergence slows as ϒ approaches 1. This is evident from right most chart where ϒ=0.99 takes more iterations to converge than with ϒ=0.75.
- Time increases linearly with I (number of iterations) which makes sense as utility computation for each state occurs I times.
- Compared to TH MDP, here we see algorithm takes more iterations (40 vs 61 using ϒ=0.75) to converge and time increases by a factor of 1.4x. This can be attributed to larger state space for Labyrinth MDP.



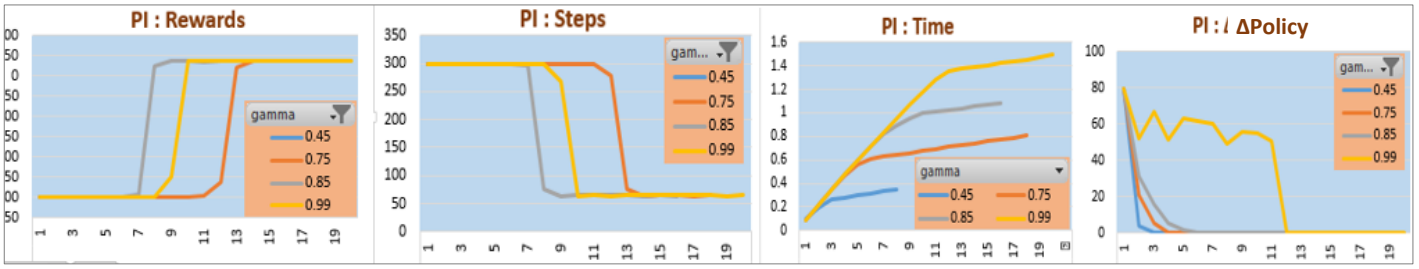

## 2.2 POLICY ITERATION (PI)

- Impact of ϒ for PI is not very different from what I saw in VI analysis. ϒ too high (0.99) or too low (0.45) were not ideal for this problem as well. Low ϒ failed to find optimal policy under convergence constraint, while high ϒ led to longer convergence time.
- While VI keeps improving the value function at each iteration until the value-function converges, PI cares only about finding optimal policy and sometimes the optimal policy will converge before the value function. This is what I witnessed for both MDPs. PI took less iterations to converge compared to VI. In terms of convergence time, PI was same or less than VI (table on right).

**Convergence time**

| ϒ | PI | VI |
|---|---|---|
| 0.75 | 0.8044 | 1.0161 |
| 0.85 | 1.0875 | 1.0870 |
| 0.99 | 1.4943 | 1.5826 |

- Compared to TH MDP, we see that algorithm takes more iterations (11 vs 16 using ϒ=0.75) and longer convergence time, which can tied to increased number of states in labyrinth MDP.

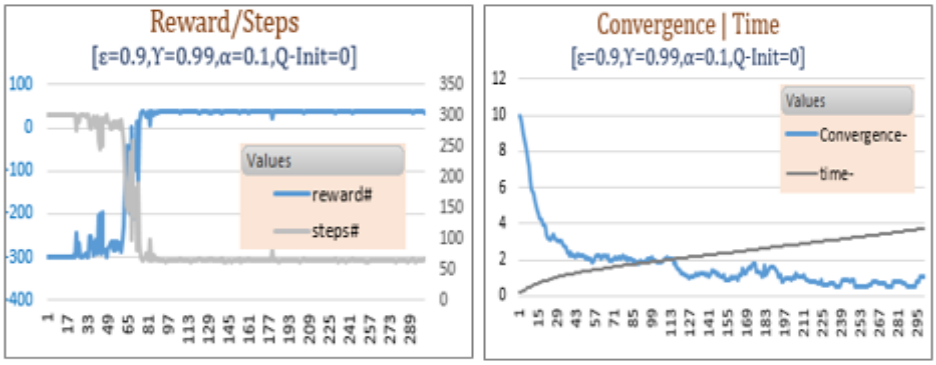



## 2.3 Q-LEARNING (QL)

To explore the behavior of QL for Labyrinth problem, I ran similar experiments as described in section 1.3, using different values of α, ε, ϒ and Q-Init and selected best of parameters

[ϒ=0.99, ε=0.9, α=0.1, qInit=0].

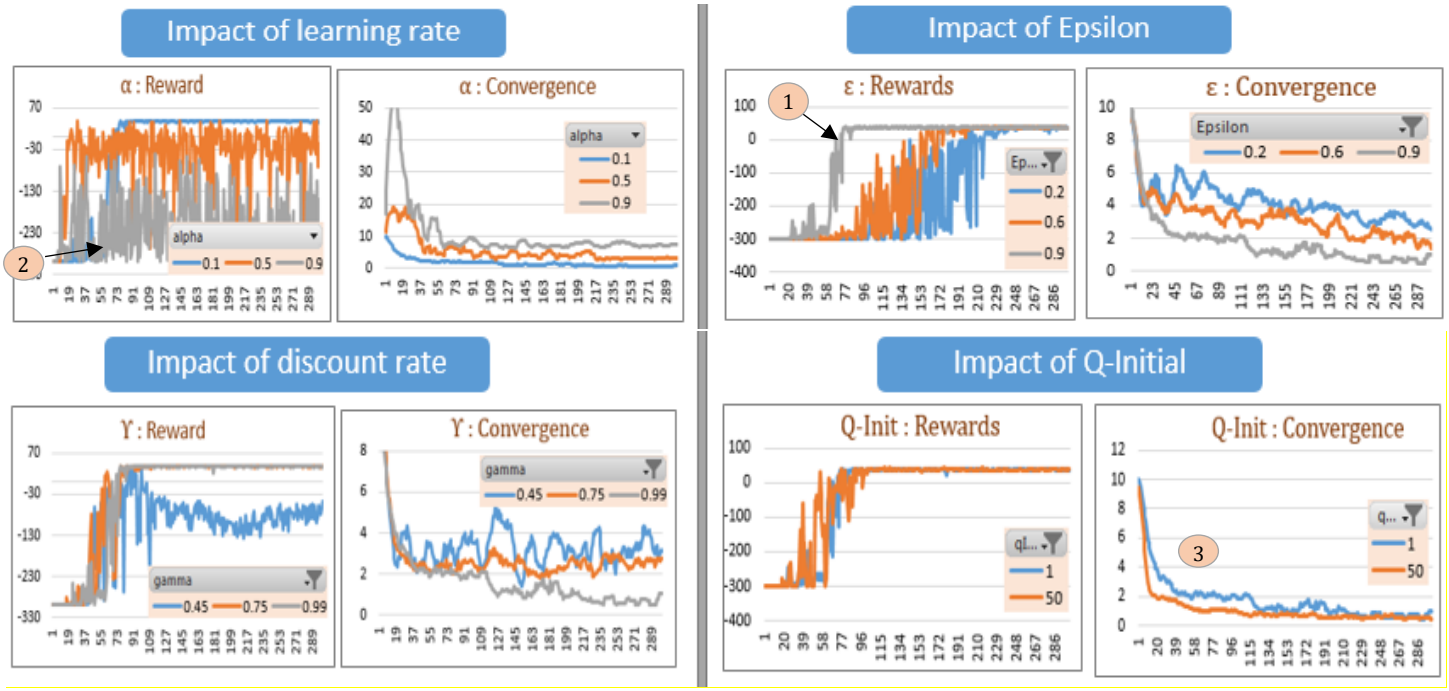
Performance of QL using above parameter set is shown on right.



From charts above we can see that impact of larger state space is much more pronounced for QL.

- QL achieved same max-rewards as PI and VI but took considerably more iterations and time (1.5 vs 3.7sec)
- Compared to Treasure hunt, Labyrinth required more time and iterations to converge (217 iterations for Treasure hunt vs. 325 for Labyrinth).

I would like to point out that, while I strived to keep all the parameters same when performing analysis for both MDPs for fair comparison, I had to modify maxEpisodeSize (max steps agent will take in a learning episode) for Labyrinth problem. For Treasure hunt QL analysis, I had restricted maxEpisodeSize to 300 but this value didn't work well for Labyrinth. Algorithm was not able to reach a steady state and huge variations (-200 to +68) in reward and Q value was seen even after 3000 iterations. Hence I had to increase maxEpisodeSize to 2000. This had an impact on time per iteration, as expected.




1. Unlike TH MDP, where impact of Epsilon didn't come through as clearly, here we can see the impact distinctly (upper right quadrant above). With large value of ε, we see faster convergence. Small value of ε discourages exploration and hence agent takes more time to discover optimal path.
2. Impact of learning rate is very noticeable as well (upper left quadrant). With α=0.9 (grey), rewards never go in positive territory. With α=0.5(orange), Agent manages to accumulate positive rewards but performance fluctuates a lot. α=0.1 worked best for this problem. With larger α, Agents seems to keep overshooting global optima.
3. Higher value of Q-initial showed faster convergence. I suspect this is because high value encourages Agent to explore more at the beginning. With small Q-Init, Agent is more conservative and hence takes more time to converge to optimal policy.

## 2.4 VI / PI / QL COMPARISION - LABYRINTH

Fig-2.4a and b below show comparison key stats and policy evolution for different RL algorithms for Labyrinth problem.

**Key observations**

- **VI:** At I=1, we see that VI starts by assigning action arbitrarily and computes expected utility to create a policy, emanating backwards from terminal state at top right. By I=35, VI converges from reward perspective ie it found an optimal path to goal, but value function doesn't converge until I= 65.

- **PI**: PI starts out with arbitrary policy and refines the policy at each iteration and computes the value according to this new policy until the policy converges. Map looks very similar to VI, except that PI (and QL) assigns very low value (red) to state left of terminal state where as VI assigns it blue. This was bit unexpected as there's no outlet from that state.
- **QL**: Policy evolution for QL looks very different from PI and VI, which is not surprising. Unlike PI & VI, QL doesn't know transition and reward function and hence takes more iterations to learn this info. While PI & VI improve policy/value function incrementally, QL takes a different approach – it balances 'exploration vs. exploitation' strategy to avoid getting stuck in local optima and discover optimal policy in reasonable time. Because of exploratory approach, we see that reward and steps trend for QL is not as smooth as for PI & VI and varies quite bit before settling in around I> 85 (Fig-2.4a).By I=60, both PI and VI have mapped out entire domain and converged, QL doesn't converges until I>300.
- While all three algorithms reach same reward, convergence time and iteration differed for each. PI converged the fastest at I=16, followed by VI at I=61 and QL at I=325. Consequently convergence time increased in similar fashion.
- Number of steps and rewards are highly correlated – as steps-taken decrease, rewards increase because with every step agent incurs a penalty of -1.

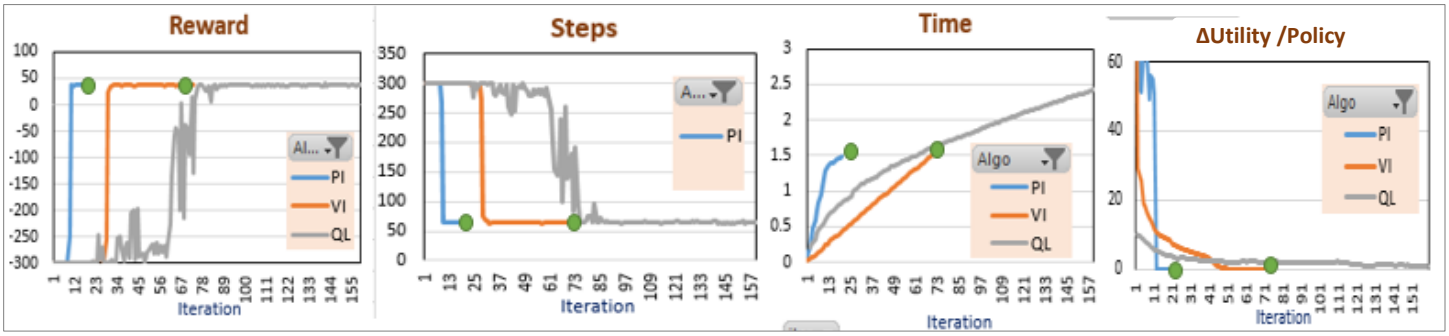



**Fig-2.4a**

**Policy evolution** - *Labyrinth*

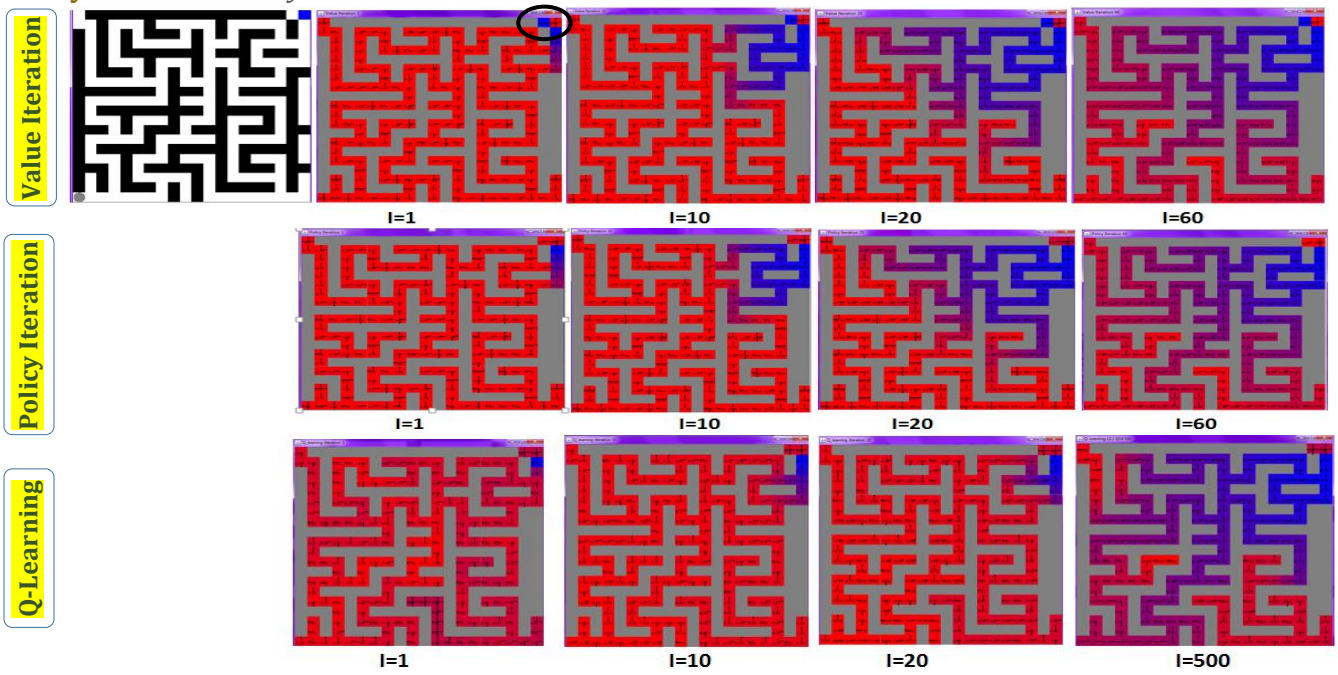


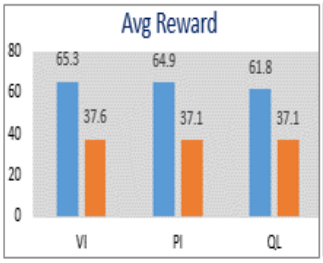

**Fig-2.4b**

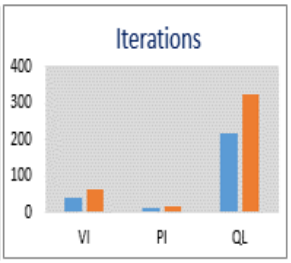
## TREASURE HUNT VS. LABYRINTH

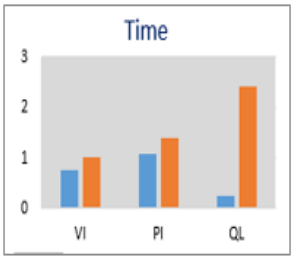
To observe performance difference between three algorithms across two MDPs, I have summarized key indicators in table on right.

| Treasure Hunt | | | | | |
|---|---|---|---|---|---|
| | Iteratio | Reward | | steps | time |
| | | Avg | Max | | |
| VI | 40 | 65.286 | 67.72 | 33.984 | 0.777 |
| PI | 11 | 64.94 | 67.74 | 33.94 | 1.086 |
| QL | 217 | 61.78 | 66.78 | 35.84 | 0.2403 |

| Labyrinth | | | | | |
|---|---|---|---|---|---|
| | Iteratio | Reward | | steps | time |
| | | Avg | Max | | |
| VI | 61 | 37.57 | 38.68 | 64.42 | 1.01 |
| PI | 16 | 37.07 | 38.08 | 64.9 | 1.41 |
| QL | 325 | 37.05 | 37.88 | 65.13 | 2.43 |




- As the problem size (state-space) increased from 13x13 to 20x20, we see that iterations required for convergence also increased. This is more pronounced for QL and VI and much less for PI. This could be explained by that fact that PI cares for optimal policy and not for 'true' utilities like VI and policy can converge before value-function.
- Even though iterations taken by PI were less, each iteration is computationally expensive and hence we can see from the table that total to convergence is slightly higher than VI.
- Time taken by QL increases considerably as number of states to be explored increased. I had to allow significantly more 'steps per Episodes' to allow QL to explore optimal path and achieve max reward.

## CONCLUSION

In conclusion, analysis of these two MDPs helped me gain better understanding of inner workings of these algorithms. While QL took more iterations/time, I find it very promising as for many real-world problems we don't know transition & reward model in advance.

These experiments also helped me understand a) how to influence Agent's behavior by tuning algorithm's parameters such as Discount factor, Epsilon, learning rate, Q-init, which at heart tries to strike a balance between exploration vs. exploitation. b) How reward structure influences ultimate policy.

I acknowledge that there are several aspects of real world RL challenges that my MDPs didn't cover in this analysis-

- In hindsight, I should have picked a MDP with continuous state-space, which would have given me a chance to explore how 'curse of dimensionality' is handled in RL world. Discretization of state-space is one solution but how efficiently and meaningfully state-space is discretized without compromising performance of learnt-policy is key.
- In this assignment we didn't worry about number of interactions agent has with the environment. In many cases, interactions with real world are expensive in terms of time, resources and sometimes money and hence prohibits training in production environment. (eg: When training a trading-agent, we can't let it trade live stocks just for the purpose of training). This emphasizes need of effective simulations.(Dyna)
- We made an implicit assumption that agent can observe all states. But in many real world tasks, Agent can't observe its environment in entirety. It has to remember the past states and take the best action w.r.t current observation.
- I kept α and ε constant for QL experiments to make comparison meaningful and bring out any differences but in practice we want dynamic α and ε (decays over time) so that larger learning steps and random actions are favored at the beginning but as time passes, Agent is forced to take smaller steps and greedy action as its honing into global optima.

I hope in future RL course I will have a chance to explore and work on these challenges.

### References


[1] BURLAP java code library http://burlap.cs.brown.edu/

[2] https://github.com/JonathanTay/CS-7641-assignment-4