Comparative Study of Reinforcement Learning - Markov Decision Processes

Xiangnan He

Nov 27, 2017

I. Abstract

In this study, we explored the Markov Decision Process with value iteration, policy iteration, and Q-learning. Two scenarios are considered: An easy 6×6 grid world was used to simulate a robot movement planning for the abandoned mine exploration and mapping, and a hard 15×15 grid world was used to simulate a nursing robot movement in a room for taking care of people that need help. The Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library was used for this project. The parameters are varied for this comparative study, including learning rate, epsilon, initial state, and discount factors. The output results analyzed include time, reward, steps/actions, and convergence/latest delta. A grid world size study was also included to study the impact of varying grid world size from 2×2 to 30×30.

II. Introduction

I. Markov Decision Processes

Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying a wide range of optimization problems, including robotics, automatic control, economics, and manufacturing.

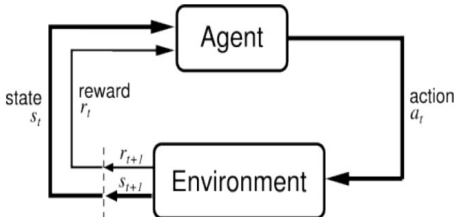
A Markov Decision Process is a tuple (S, A, T, r, γ), where S is a finite set of states, A is a finite set of actions, T is a state transition probability function $T(s' |s, a) = P[S_{t+1} = s' |S_t = s', A_t = a]$, r is a reward function $r(s, a) = E[R_{t+1}|St = s, At = a]$, γ is a discount factor $\gamma \in [0, 1]$.

MDPs can be solved by linear programming or dynamic programming. In what follows we present the latter approach. Suppose we know the state transition function P and the reward function R, and we wish to calculate the policy that maximizes the expected discounted reward. The standard family of algorithms to calculate this optimal policy requires storage for two arrays indexed by state: value V, which contains real values, and policy π which contains actions. At the end of the algorithm, π will contain the solution and V(s) will contain the discounted sum of the rewards to be earned (on average) by following that solution from state s.

The algorithm has the following two steps, which are repeated in some order for all the states until no further changes take place. They are defined recursively as follows [1]:

Step1: $$\pi(s) := \arg\max_a \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V(s') \right) \right\}$$

Step2: $$V(s) := \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s') \right)$$



On the right side, we can see how the Agent interact with Environment. The assumption in this plot is that agent gets to observe the state.

II. Algorithms

1. Value Iteration (VI)

In value iteration, which is also called backward induction. The *π* function is not used, instead, the value of *π(s)* is calculated within *V(s)* whenever it is needed. As shown in the equation below:

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s,s') \left(R_a(s,s') + \gamma V_i(s') \right) \right\},$$

where *i* is the iteration number. Value iteration starts at $i = 0$ and V_0 as a guess of the value function. It then iterates, repeatedly computing V_{i+1} for all states s, until V converges with the left-hand side equal to the right-hand side (which is the "Bellman equation" for this problem).

2. Policy Iteration (PI)

In policy iteration, step one is performed once, and then step two is repeated until it converges. Then step one is again performed once and so on. Instead of repeating step two to convergence, it may be formulated and solved as a set of linear equations. This variant has the advantage that there is a definite stopping condition: when the array π does not change in the course of applying step 1 to all states, the algorithm is completed.

3. Q-learning

Reinforcement learning can solve Markov decision processes without explicit specification of the transition probabilities; the values of the transition probabilities are needed in value and policy iteration. Q-learning is a model-free reinforcement learning technique. Q-learning can be used to find an optimal action-selection policy for any given finite MDP. It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter. One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment.

$$Q(s,a) = \sum_{s'} P_a(s,s')(R_a(s,s') + \gamma V(s')).$$

In reinforcement learning, instead of explicit specification of the transition probabilities, the transition probabilities are accessed through a simulator that is typically restarted many times from a uniformly random initial state. Since the probabilities or rewards are unknown, it is useful to define a further function, which corresponds to taking the action *a* and then continuing optimally (or according to whatever policy one currently has): While this function is also unknown, experience during learning is based on (s, a) pairs. Thus, one has an array Q and uses experience to update it directly.

$$Q(s_t,a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right),$$

Reinforcement learning can also be combined with function approximation to address problems with a very large number of states.

III. Implementation

The implementation of MDP utilized existing Java code [2], which was based on the original source code from Burlap [3]. Jython was used for comparative study by varying parameters [4], such as learning rate, intial state, epsilon, and discount factors. Jupyter notebook was used for plotting the results as CSV files.

The grid world model from Burlap is used. The starting point is on the bottom left corner (0, 0), which has the choices of movement in the four cardinal directions to reach the terminal state while avoiding the wall. We focuses on the single agent model, where the agent is forced to make an action with a small negative cost of -1 per step except the terminal state. A large reward of 100 will be provided if the agent reaches terminal state. The goal is to maximize the net reward by reaching the terminal state in the fewest number of steps. The actions are made probabilistic (not deterministic). The probability of the intended direction is 0.8, and other directions share the same probability of 0.2/3 = 0.0667. A discount factor is used to factor the future rewards into the expected value of each step. The learning rate (LR) is a relative measure of the recent vs. previous information for each decision making process. The qInital value is to initialize all states before updating the values with true data. When it is high, all states are more viable to explore, and less viable or exploratory when it is low, which means more exploitation. Epsilon ε is used as a threshold of random generation, below which an agent will choose to explore. The output plots include time/duration, reward, steps/actions, and convergence/latestdelta.

III. MDP results

Part I. Easy Grid World Problem - Abandoned Mine Exploration robot

Robot system is an interesting method for mapping subterranean spaces such as abandoned mines. Subsidence of abandoned mines poses a major problem for society, as do ground water contaminations, mine fires, and so on. Mine maps, the primary source of information on mine structures, are key components in resolving this threat and preventing mine-related disasters; however, in the case of abandoned mines, such maps are often inaccurate and unreliable if they exist at all. Tens of thousands, perhaps even hundreds of

thousands, of abandoned mines exist today in the United States. Not even the U.S. Bureau of Mines knows the exact number, because federal recording of mining claims was not required until 1976. Most abandoned mines are inaccessible to people, but some are accessible to robots. Autonomy is a key requirement for robots operating in such environments, due to a lack of wireless communication technology for subterranean spaces [5-7]. The objective in this study is to optimize the robot movement planning in a grid world that resemble the abandoned mine.

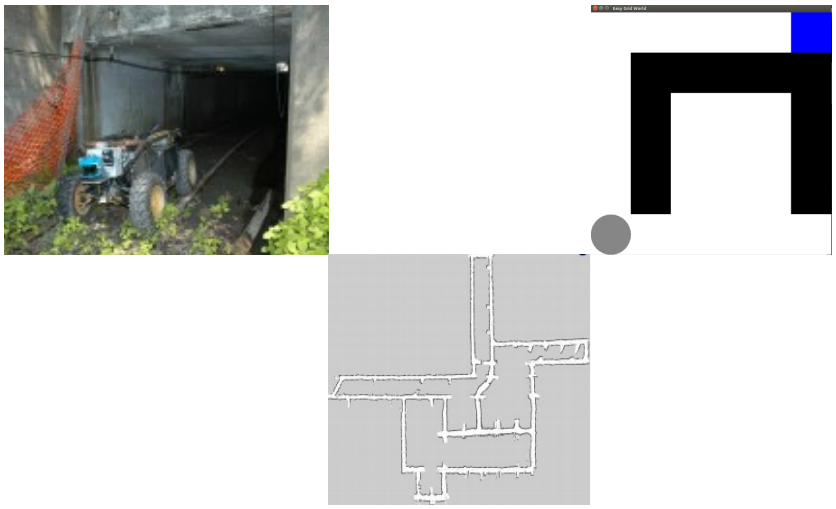


Figure 1. Robot exploration of abandoned mine and a grid world design of a dead end in the mine

Figure 1 left photo shows the ground hog robotic mine mapping platform. For the case here, as shown on Fig1. right image, we built a dead end type of grid world 6×6 pattern to simulate the robot exploration, as we can see from Fig1. middle image, there are many dead ends in the mine path. The agent starts at bottom left corner and the destiny is on the top right corner. The black boxes [11state] represent the wall of the mine that the robot cannot pass. With the MDPs, the policy will be optimized for the robot given such situation. With the simple model here, we only considered the walls, instead of encouraging with higher rewards along the way. This will help make easier comparison with the following hard world case. The terminal state reward is 100. A default settings was used, with a discount factor of 0.99, learning rate of 0.99, and epsilon of 0.1.

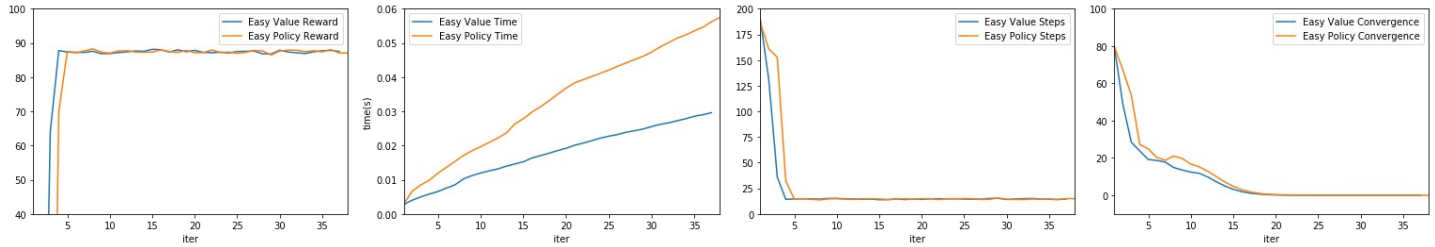


Figure 2. Value iteration vs policy iteration plots

From Fig. 2, from the reward plot, VI and PI performs similarly with the reward reaches the final reward neighborhood at 4 iterations for VI and 5 iterations for PI. They had a large negative utility at the beginning before stabilizing/converging. This effect can also be seen from the action plot, where ~200 steps at the beginning. The reward stabilizes at around 87, which make sense, since the cost of each step is -1.The duration grows with more iterations almost linearly, and PI grows 2 times fasters compared with VI, which make sense given that PI evaluate the π policy function for each iteration. The time unit is second. The linearly growth of the duration also make sense because from the steps plot, the growth of duration will linearly grow with more actions. The steps reduce to around 15 steps after 4 iterations for VI and 5 iterations for PI. If a convergence criteria of 1.0 is used, VI converges at 18 iterations and PI at 19 iterations. When we apply a convergence criteria of 1e-6, the VI converge after 37 iterations and PI at 38 iterations.

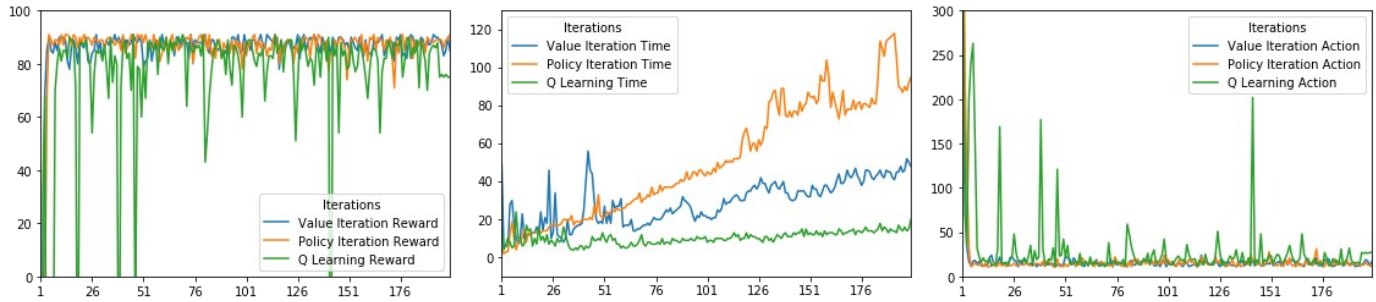


Figure 3. Value iteration, policy iteration, and Q-learning plots on reward, time, and actions.

Fig.3 left image shows the reward comparison among VI, PI, and Q learning, where VI and PI show similar trend. VI and PI rewards stabilized very quickly after ~ 5 iterations, and stay at a small fluctuation range, however, Q learning is spiking downward frequently, suggesting it is still exploring new solutions not knowing a model existence. From Fig.3 mid image, both VI and PI costs more much more time than Q-learning, which make sense in that Q-learning runs at a constant speed since it hashes the actions taken and rewards until a policy is calculated, whereas PI requires calculation of policy each iteration. The time unit here is millisecond. For Q-learning, the reward stabilizes slower, provided that Q learning is not based on a model, it is expected to take longer time to converge. The right image shows similar trend in steps/actions, VI and PI quickly reach a state at about 15 steps, whereas Q learning jumps to high steps every now and then. The fluctuation in all three plots is expected due to the probabilistic movement of the agent.

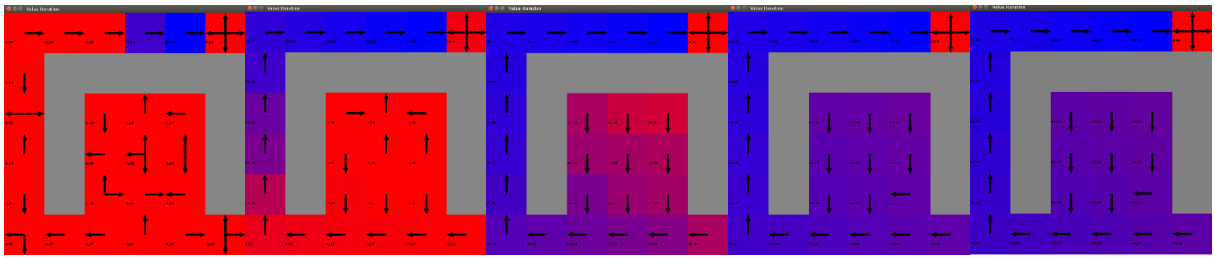


Figure 4. Value iteration policy map at 1, 5, 10, 20, 100, 200 iterations.

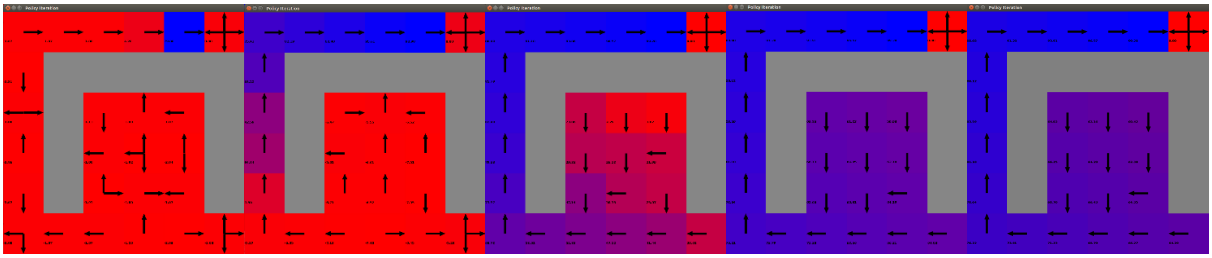


Figure 5. Policy iteration policy map at 1, 5, 10, 20, 100, 200 iterations.

As we can see from Figures 2 and 3, the policy map is constructed with VI and PI, respectively. At iteration = 1, the closest states to the terminal state are evaluated for utility. With more iterations, more states are updated with the true value with this kind of backward induction. The red colored states are where the agent has not reach and update, and the arrows are not optimized. At iteration = 5, more states are updated, and the arrows are sorted and pointing to the terminal state. With more iterations, more red states are initialized.

The policy maps show no change after 20 iterations, and both PI and VI shows identical policy map after that. Although the reward stabilize at iteration = 5, the policy maps keep updating after 10 iterations, suggesting that the remaining changes has minimal impact on the reward. From the arrow, we can see the policy map guide the robot to move away from the dead end. Comparing the policy map at iteration = 10, the PI policy map shows three states not optimized (comparing with the iteration at iteration = 20), whereas VI shows one, which make sense and correspond to the plot in Fig. 2 well, where VI reaches convergence faster.

The comparison between VI and PI suggests slight difference between them with a convergence difference of only 1 iteration, very tiny. However, the expectation was that PI should be able to converge faster than VI given PI calculate the policy π function in each iteration and takes longer duration. A study on the following nursing robot case will be carried out to improve the PI to show the advantages of PI on faster convergence compared to VI.

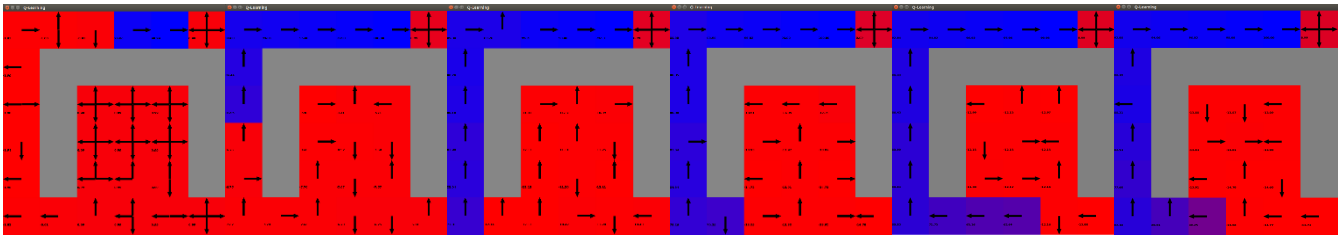


Figure 6. Q-learning policy map at 1, 5, 10, 20, 100, 200, 500 iterations.

As compared with VI and PI, Q-learning is a model-free method which assigns a value to all states and modifies them by considering the immediate rewards and explore the delayed rewards. From Fig. 6, the major difference between Q-learning and PI and VI are the fact the red states in the dead end always there even at iteration of 500, suggesting the algorithm does not learn anything new during exploring the immediate and the delayed rewards of that section. Q-learning continues to explore policies even after 500 iterations with lower utilities. The policy maps vary after iteration = 20, but overall not major change. The Q-learning actively explore new policies and not settling on a policy (not same as VI and PI), explaining why the average reward and total utility per iteration is lower than VI and PI.

Part II. Hard Grid World Problem – Nursing Robot

The US population is aging at an alarming rate, it is widely recognized that this ratio will increase as the baby-boomer generation moves into retirement age. Meanwhile, the nation faces a significant shortage of nursing professionals. This acute need provides significant opportunities for robotics and AI researchers to develop assistive technologies that can improve the quality of life of our aging population, and help nurses become more effective in their activities [8].

Figure 7. Nursing robots and the grid world design of a room

As shown in Fig. 7, the left photo shows a guide robot that help leading the elderly people that are not capable of going to the places they intended to. The middle photos shows a nursing robot that can take care of people with disabilities and take them to where they want to go. On the right, we constructed a room with a grid world with 15×15 grid with the robot on the bottom left corner as a round dot and the place to go on the top right as a blue square. There are 40 sites used for wall showing as black boxes, which is used to simulate furniture and room walls. A default settings was used, with a discount factor of 0.99, learning rate of 0.99, and epsilon of 0.1. The objective is to optimize the movement of the robot while it navigates to the intended place, either to pick up the patient or to deliver the patient to the place.

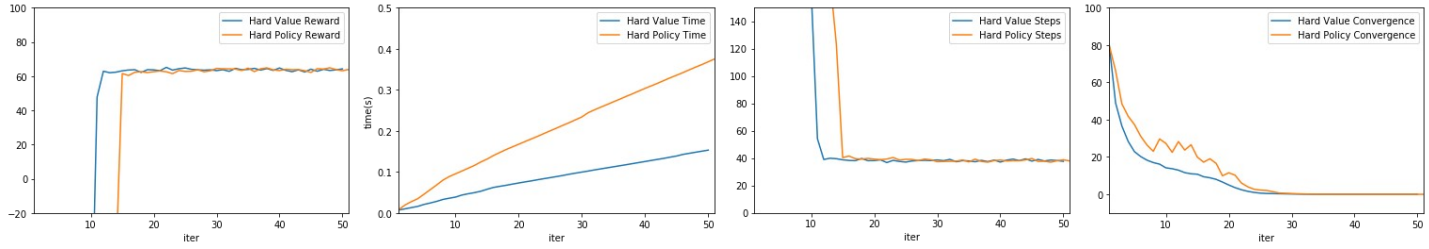


Figure 8. Value iteration vs policy iteration plots on reward, time, steps, and convergence.

From the reward plot in Fig.8 on the left, the rewards stay at around 60, which is expected because the grid world is larger and more states are involved to reach to the terminal state. VI reaches the stable reward of around 60 at 11 iterations, whereas PI reaches there at 14 iterations. Again, the time plot shows a linear duration with more iterations, and PI shows almost 2 times the growth rate of VI. The time unit here is second. The steps plot shows the steps stay at around 40 after 14 iterations. The convergence plot shows VI and PI

converges at 25 and 28 given a criteria of convergence < 1.0. The trend is very similar to the mine robot case, except that the convergence takes longer with more steps and iterations due to a larger grid world in a room compared with a mine dead end.

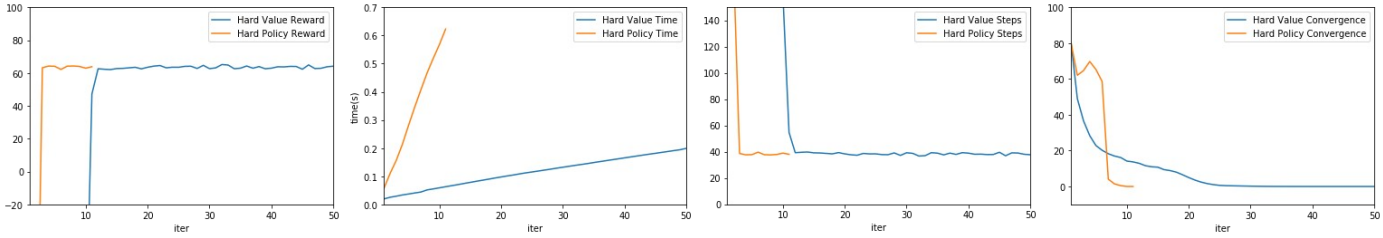


Figure 9. Value iteration vs policy iteration plots after change max evaluation iteration from 1 to 10.

As shown in Fig.9, the plots of the same room grid world, but with max evaluation iteration changed from 1 to 10 in PI, which means in each policy iteration, there will be maximum 10 evaluation iteration of the adjustment made in the policy. With this change, the policy reaches the reward of 60 after only 3 iterations, much faster than VI now. The time spent on PI also drastically change due to the max evaluation iteration change, which means more steps in each policy iteration. The orange line stops due to the setting of convergence limit of 1e-6.

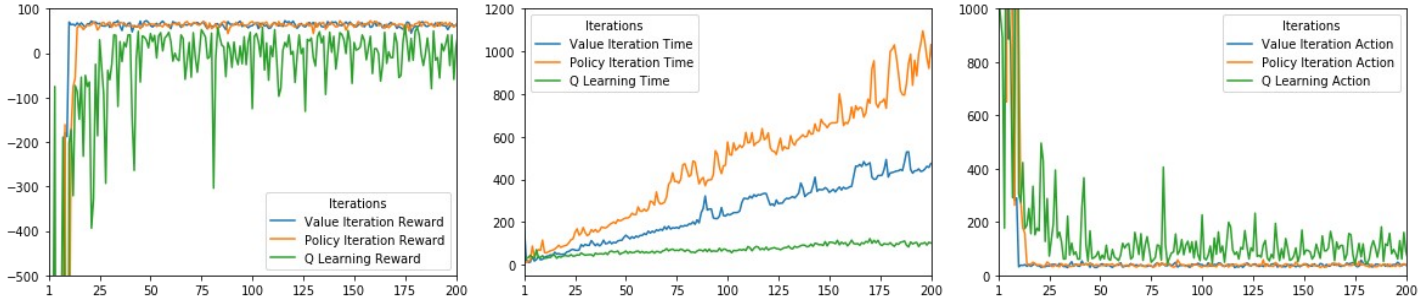


Figure 10. Value iteration, policy iteration, and Q-learning plots on reward, time, and actions.

In Fig. 10, the setting of the max evaluation iteration in PI is still set to 1 for easier comparison with the easy grid world scenario. From the reward plot on the left, we can see that VI and PI performs similar with VI converges slightly faster, and Q-learning is still fluctuating much more due to its creativity/exploratory nature discussed in the mine robot case. The time spent is much more significant than the mine robot case due to more states (more states/actions to reach the terminal state). Again, PI requires almost two times the duration compared with VI, but Q-learning stays very low. From the action plot on the right, the VI and PI performs similar to mine robot case with higher average action #. Q-learning action overall not changing that much after it converges.

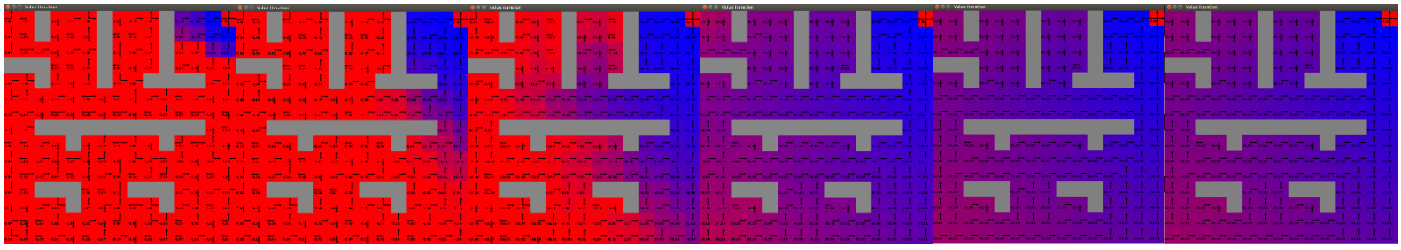


Figure 11. Value iteration policy map at 1, 5, 10, 20, 100, 200 iterations

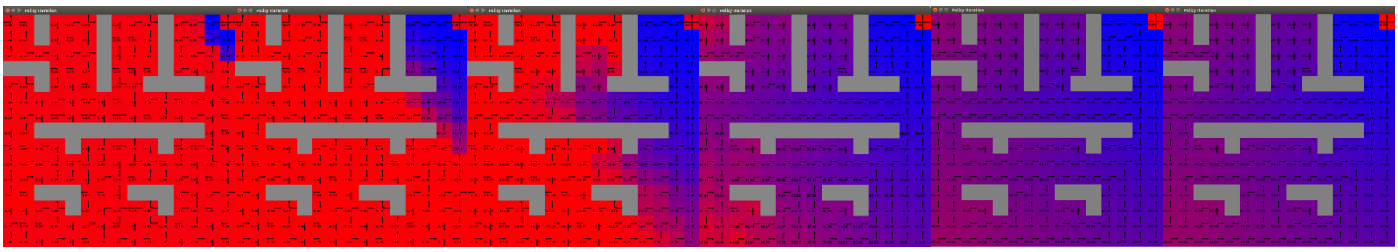


Figure 12. Policy iteration policy map at 1, 5, 10, 20, 100, 200 iterations

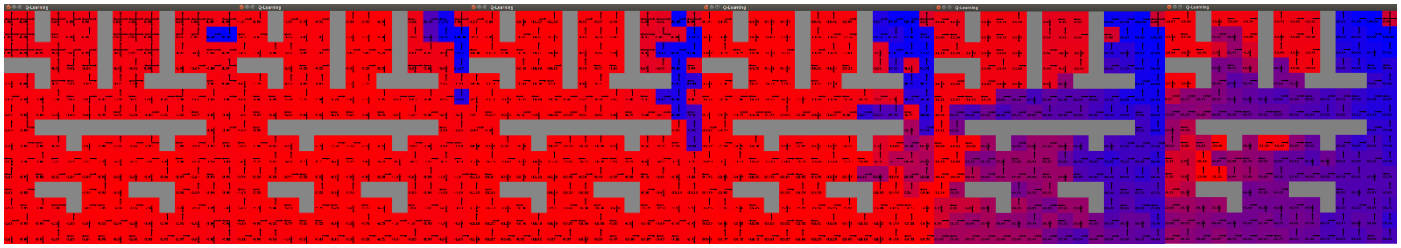


Figure 13. Q-learning policy map at 1, 5, 10, 20, 100, 200 iterations

From Fig. 11 and 12, VI and PI is similar with VI converge slightly faster than PI. A more preferred path is toward the right side of the grid world with more blue colors, which make sense that it is less likely to get trapped in one of the rooms on the top left quadrant. The Q-learning plot in Fig.13 shows that the algorithm does not learn anything on the top left quadrant from the immediate reward and the delayed rewards. Similar to VI and PI, it prefers the path on the right side of the grid world.

Part III. Algorithm Parameter Study

In this Q-learning study, we varied learning rate, initial state, epsilon, and discount factor to study the impact to the performance of the nursing robot room grid world.

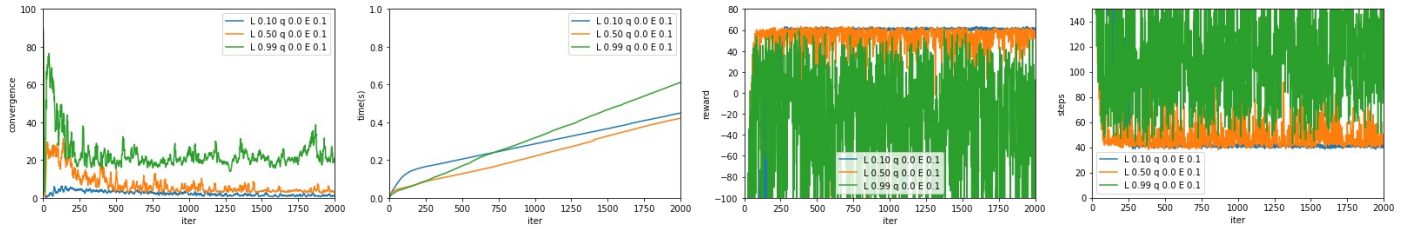


Figure 14. Hard grid world plots with varying learning rate at 0.1, 0.5, and 0.99

The learning rate α_t determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything (more exploitation), while a factor of 1 would make the agent consider only the most recent information (more exploration). In fully deterministic environments, a learning rate of α_t =1 is optimal. When the problem is stochastic, the algorithm still converges under some technical conditions on the learning rate that require it to decrease to zero (more exploitation). In practice, often a constant learning rate is used, such as α_t =0.1. In this study, we compare the Q-learning results with different learning rate of 0.1, 0.5, and 0.99 (was used in the previous default setting). In Fig. 14, convergence plot, higher learning rate converges at ~20 after about 1250 iterations, whereas L=0.5 at ~8 and L=0.1at ~3. The time plot shows with higher learning rate, the algorithm explores more with more steps and causing the time cost to be higher. Although L=0.1 is higher at the beginning due to more steps at the beginning, the slope decreased after 125 iterations, and trending to be lower than L=0.5 after 2000 iterations. The reward plot shows less variation in L=0.1 and highest variation in L=0.99, which is expected given L=0.99 is more explorative. Same idea in the steps plot, where L=0.99 show much higher average and variation, whereas L=0.1 shows the least in both average value and variation. Overall, It looks like a lower (more exploitive) learning rate help achieve a more favorable convergence in this case, which make sense in that the best route in this room grid world design is not hard to find. If the world was deliberatively design so the best solution is not easy to find, then the algorithm with higher learning rate will probably excel.

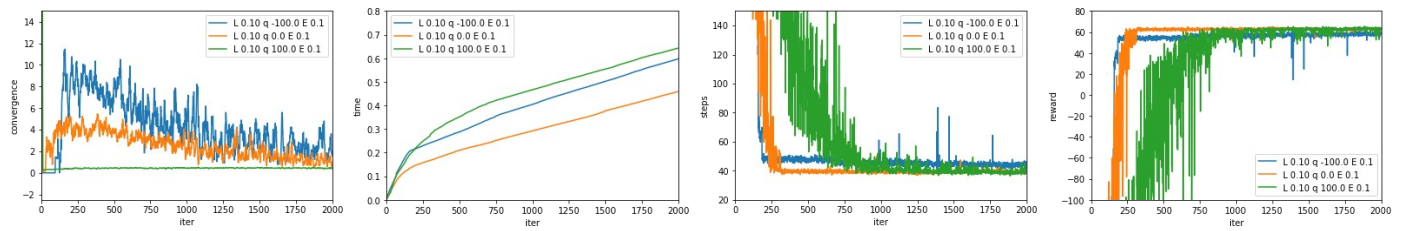


Figure 15. Hard grid world plots with varying q initial states at -100, 0, and 100.

Given that Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs. High initial values, also known as "optimistic initial conditions", will encourage exploration, which means no matter what action is chosen, the update rule will cause it to have lower values than the other alternative, thus increasing their choice probability. On the other hand, when the initial condition is set to low values, algorithm will encourage exploitation or less exploration.

In this study, we set the initial value (qInit) of the hard grid world to -100, 0, and 100 to study the effect. In Fig. 15 convergence plot, we can see that qInit = -100 shows much higher convergence (or latest delta) than qInit = 100, and qInit = 0 is in the middle. In the time plot (unit is second), qInit = 100 is higher and qInit = 0 and -100, suggesting more actions or exploration compared with lower qInit conditions. Interestingly, qInit = 0 has the lowest time cost in the three, probably because it reaches reward stable state faster and the final steps number (at 40) is lower compared with qInit = -100(at 50). The time is growing fast before iteration = 250, and then slow down because the steps/actions start to be more stable. The steps vs iteration plot show that qInit = 100 reaches steps = 40 much slower than qInit = 0 and -100, at about iteration = 800. Also the plot shows that qInit = -100 end up at different level of steps around 50, higher than q = 0 and 100. The reward plot correlates with the steps plot well, suggesting lower reward for qInit = -100 overall at around 55, whereas qInit = 0 and 100 at around 60. qInit = 100 reaches the stable reward after iteration = 800. Both steps and reward plot show qInit = -100 has more spike in the curve. Overall, qInit = 0 helps the algorithm to reach the best solution/policy with reasonable reward and less time cost.

Recently, it was suggested that the first reward *r* could be used to reset the initial conditions. According to this idea, the first time an action is taken the reward is used to set the value of Q. This will allow immediate learning in case of fixed deterministic rewards. Surprisingly, this resetting-of-initial-conditions (RIC) approach seems to be consistent with human behavior in repeated binary choice experiments. Therefore, as future work, RIC can be studies to further optimize the initial condition.

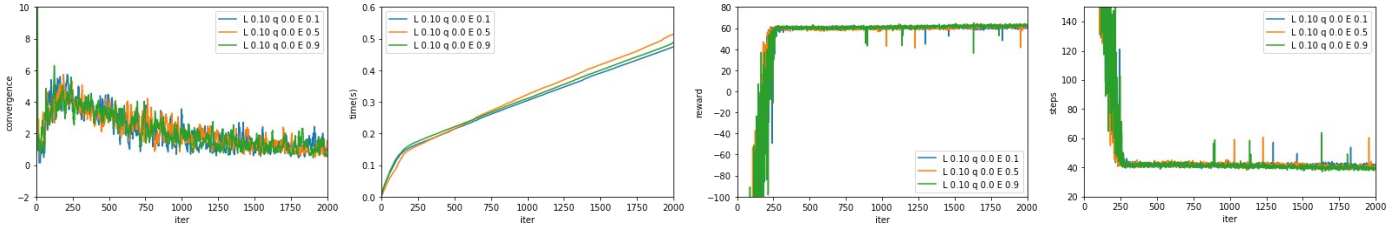


Figure 16. Hard grid world plots with varying epsilon at 0.1, 0.5, and 0.9 (learning rate at 0.1)

I would like to discuss the idea of the introduction of epsilon here to better understand the impact of epsilon to the results. For instance, consider a labyrinth game where the agent's current Q-estimates are converged to the optimal policy except for one grid, where it greedily chooses to move toward a boundary that results in it remaining in the same grid. If the agent reaches any such state, and it is choosing the Max Q action, it will be stuck there for eternity. However, keeping a vaguely explorative / stochastic element in its policy (like a tiny amount of epsilon) allows it to get out of such states. The standard way to get exploration is to introduce an additional term, epsilon. We then randomly generate a value, and if that value is less than epsilon, a random action is chosen, instead of following our normal tactic of choosing the max Qvalue. The introduction of epsilon (ε) is to nullify the negative effects of over / under fitting. Using epsilon of 0 is a fully exploitative choice whereas epsilon = 1 is a fully exploratory choice.

In the study, we varied the epsilon at 0.1, 0.5, and 0.9. From the convergence plot, we can see it is similar in all three cases. In the time plot, epsilon = 0.5 is lower and later on higher, but overall very identical to the other two cases, suggesting similar time performance. The reward and steps plots also suggest identical trend between the three different epsilon settings. Maybe the difference exists, but it will be very tiny. Therefore, the learning rate was changed to 0.99 (more exploration) to see is any different, as showing in Fig.17.

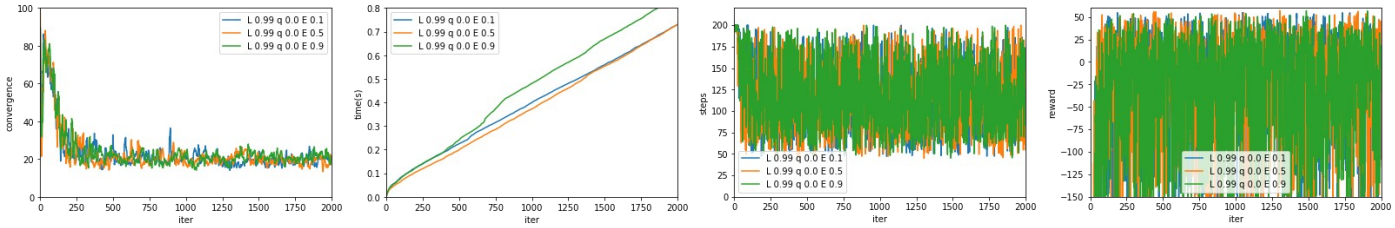


Figure 17. Hard grid world plots with varying epsilon at 0.1, 0.5, and 0.9 (Learning rate at 0.99)

In Fig.17, the plots at learning rate of 0.99 shows similar trend on convergence plot, steps plot and reward plot. In time plot, epsilon = 0.9 is more time consuming compared with the other two, suggesting more actions/steps in the process, although there is not a significant difference between epsilon =0.5 and 0.1. It is possible that this grid world design is not suitable for comparing the epsilon impact on the algorithm. To give another try, we look back to the easy grid world case of the mine mapping robot, see Figure 18.

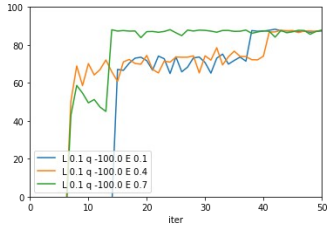


Figure 18. Easy grid world Q-learning reward plot with epsilon at 0.1, 0.4 and 0.7 (learning rate at 0.1)

In Fig. 18, we also zoom in the plot, from which we can see the epsilon at 0.7 reaches the reward of 87 faster, probably because it was most explorative. Epsilon = 0.4 reaches 65 faster than E = 0.1, and later on reaches 87 almost at the similar iteration compared with E = 0.1 at around iteration = 40. It seems like we find the case where we can compare the epsilon impact on Q-learning.

In addition, to continue our discussion of the epsilon, interestingly let's consider what if we've explored all the options and we know for sure the best option, we still sometimes choose a random action, which means the exploration feature does not turn off. There are a number of ways to overcome this, most involving manually adjusting epsilon as algorithm learns, so that it explores less and less as time passes and it has presumably learned the best actions for the majority of situations.

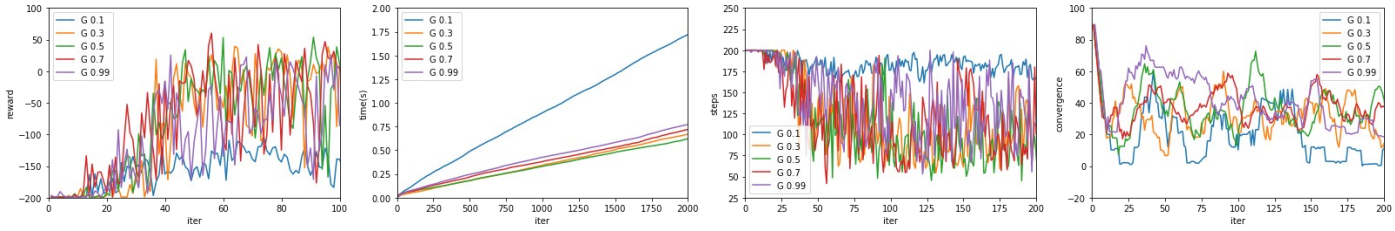


Figure 18. Q-learning plots with varying discounts at 0.1, 0.3, 0.5, 0.7, and 0.99.

The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. If the discount factor meets or exceeds 1, the action values may diverge. For γ =1, without a terminal state, or if the agent never reaches one, all environment histories will be infinitely long, and utilities with additive, undiscounted rewards will generally be infinite. Even with a discount factor only slightly lower than 1, the Q-function learning leads to propagation of errors and instabilities when the value function is approximated with an artificial neural network. In that case, it is known that starting with a lower discount factor and increasing it towards its final value yields accelerated learning. It is an important parameter, therefore we varied it (we name it G for Gamma here) at 0.1, 0.3, 0.5, 0.7, and 0.99. From Fig. 18 reward plot, with lower G value at 0.1, the reward has lower fluctuation but stay at lower level at -150, seems like not in a convergence situation. With high G value at 0.99 (purple curve), however, the variation is huge compared with others. Other G values at 0.3, 0.5, and 0.7 look identical. In the time plot, we can see G = 0.1 is much higher than other G values, which are similar with them. The high time cost in G = 0.1 can be explained in the steps plot, where G = 0.1 run much higher steps and is kind of stable there. In the convergence plot, G = 0.1 shows lower overall convergence (or latest delta) compared to others, while G = 0.99 has the highest delta, which make sense from reward plot discussion. In this study, it seems like we should choose something within 0.3, 0.5, and 0.7 to avoid the (stuck) low average reward of G = 0.1 and high variation at G = 0.99.

IV. Grid World Size Study

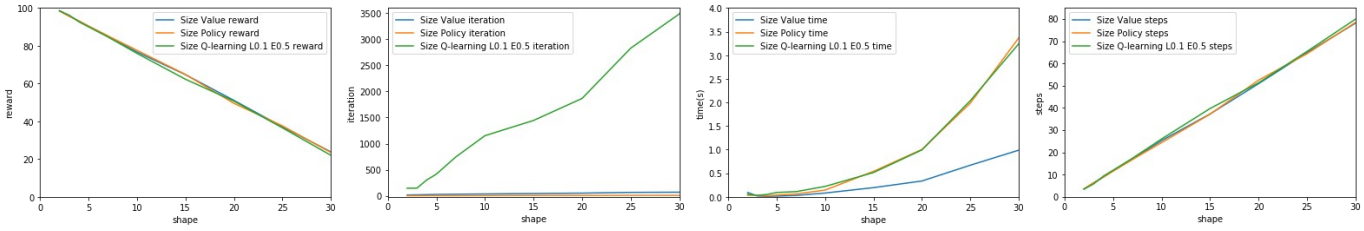


Figure 19. Plots for comparing value iteration, policy iteration, and Q-learning with varying grid world size at 2, 3, 4, 5, 7, 10, 15, 20, 25, and 30.

When we consider just a simple grid world without walls, by increasing the size of the grid world with all states of 0, we can study how this impact the performance of the algorithms. In this study, we varied the size of the grid world from 2×2, 3×3 all the way to

25×25, 30×30. As can be seen from the Fig.19 reward plot, there is a linear decrease of reward due to more states in the grid or more action (with a cost of (-1)) with larger grid size. PI and VI reward is slightly higher overall than Q-learning which is expected after our study of the mine robot (easy grid world) and nursing robot (hard grid world) cases. From the iteration vs shape plot, the Q-learning with learning rate of 0.1 and epsilon of 0.5 shows much more iteration number and grows with larger grid world. VI runs more iterations compared with PI, which make sense from our previous analysis.

In the time vs shape plot, surprisingly, policy and Q-learning shows similar trend, and both are much higher than value iteration. Q-learning runs much more iteration which probably increased the time to be comparable to PI. In the steps vs shape plot, there is a linear relationship, which is expected since more steps needed with larger grid world to reach the terminal state, overall Q-learning is slight higher than VI and PI.

V. Conclusions

In this study, we compared VI, PI, and Q-learning by studying a 6×6 grid world and a 15×15 grid world using BURLAP library to study the MDP process of a mine mapping robot in a dead end and a nursing robot in a large room. VI and PI can converge fast and reaches a stable stage without changing the policy map, whereas Q-learning requires more iterations to reach the converged policy map, but the time cost is much less due to its model-free algorithm. The time cost of PI is around 2 times the cost of VI. VI converges slightly faster than PI, but with increase max evaluation iteration, PI converges quicker than VI with a higher time cost. Policy maps was included and discussed at different iteration. Parameters in Q learning were varied to study their impact on time, reward, steps, and convergence, including learning rate, initial condition, epsilon, and discount factors. The parameter study was mainly based on the hard grid world (nursing robot), but in the epsilon study, easy grid world was used to better show the comparison on impact to the results.

A grid world size variation study was also included in the end of the study to compare the performance of all three different algorithms, VI, PI, and Q-learning (learning rate at 0.1 and epsilon at 0.5). The grid world was varied from 2×2 to 30×30 without walls. The agent starts at the bottom left and the terminal state is on the top right corner.

VI. References

[1] Kaelbling, Littman, Moore. Reinforcement Learning: A Survey. https://arxiv.org/pdf/cs/9605103.pdf
[2] CS7641 Assignment 4 BURLAP Extension https://github.com/stormont/cs7641-assignment-4
[3] Brown-UMBC Reinforcement Learning and Planning (BURLAP) java code library http://burlap.cs.brown.edu/
[4] CS7641 assignment 4 with jython template https://github.com/JonathanTay/CS-7641-assignment-4
[5] robots.stanford.edu/papers/Thrun04a.pdf
[6] ri.cmu.edu/pub_files/pub4/ferguson_david_2003.../ferguson_david_2003_1.pdf
[7] J.J. Belwood and R.J. Waugh. Bats and mines: Abandoned does not always mean empty. Bats, 9(3), 1991
[8] J. Pineau et al. / Robotics and Autonomous Systems 42 (2003) 271–281
[9] Drawing from Sutton and Barto, Reinforcement Learning: An Introduction, 1998