UNIVERSITY OF CAMBRIDGE

MPhil in Machine Learning and, Machine Intelligence 2018-2019

Module Coursework Feedback

Module Title: Reinforcement Learning and Decision Making
Module Code: MLMI7
Candidate Number: F606F
Coursework Number: 1
I confirm that this piece of work is my own unaided effort and adheres to the Department of Engineering's guidelines on plagiarism. √

Date Marked: Click here to enter a date. Marker's Name(s): Click here to enter text.
Marker's Comments:

This piece of work has been completed to the following standard *(please circle as appropriate)*:

	Distinction			**Pass**			**Fail (C+ - marginal fail)**		
Overall assessment (circle grade)	**Outstanding**	**A+**	**A**	**A-**	**B+**	**B**	**C+**	**C**	**Unsatisfactory**
Guideline mark (%)	90-100	80-89	75-79	70-74	65-69	60-64	**55-59**	**50-54**	**0-49**
Penalties	**10% of mark for each day, or part day, late (Sunday excluded).**								

The assignment grades are given **for information only**; results are provisional and are subject to confirmation at the Final Examiners Meeting and by the Department of Engineering Degree Committee.

MLMI7 Reinforcement Learning and Decision Making (1733 words)

Question a

The value iteration algorithm [1] can be outlined as the following procedures,

1. Initialize a state-value function V as a zero vector with the length of number of states.
2. Initialize a for-loop to repeatly improve the state-value function for each state s via the following equation,

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)\big(r + \gamma V_k(s')\big)$$

, where $V(s)$ the state-value function of state s, k the k-th iteration of the for-loop, a the action state took in state s, s' the next state, r the reward led by taking the action a in state s and γ the discount.

Consider the system is designed to only arrange one reward r to each state-action pair, $p(r|s, a) = 1$, and by Bayes' rule, the probability from the state-value function can be expressed as follows,

$$p(s', r|s, a) \propto p(s'|s, a)p(r|s, a) \propto p(s', s, a)$$

3. Update the policy π via,

$$\pi'(s) = arg\max_a \sum_{s',r} p(s', r|s, a)\big(r + \gamma V_k(s')\big)$$

4. Repeat until the convergence occurs. The convergence is defined as follows,

$$\max_s |V_{k+1}(s) - V_k(s)| < \theta$$

, where θ the threshold $= 0.001$.

```
1: Initialise V_0 arbitrarily
2: repeat
3:     Improve V_{k+1} using the estimate of V_k
4: until convergence
```

Figure 1: Pseudocode of value iteration.

For the testing purpose, the algorithm is implemented for smallworld. The result is shown in Figure 2. The plot looks slightly different at the position $(u, v) = (3,3)$ because whether going

down or right still leads to the same reward, the system can only choose the action (down or right) randomly.

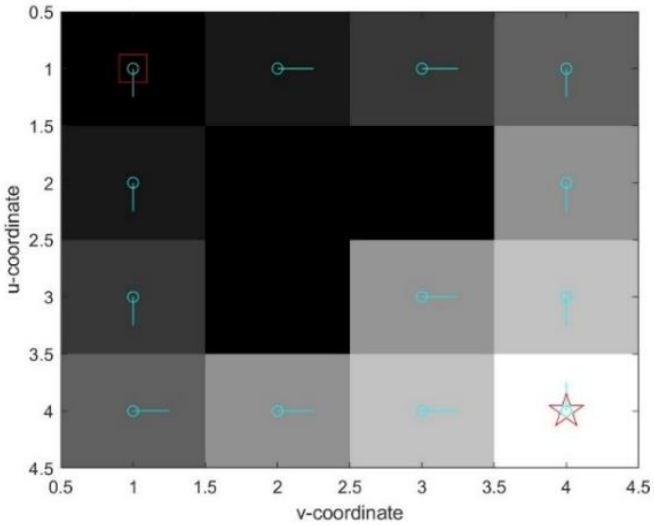

Figure 2: The value function and policy predicted by value iteration for smallworld.

The algorithm is then implemented with 10000 iterations for gridworld. In Figure 3, it is interesting to note that the policies choose to point away from the bad spot to avoid stepping in randomly.

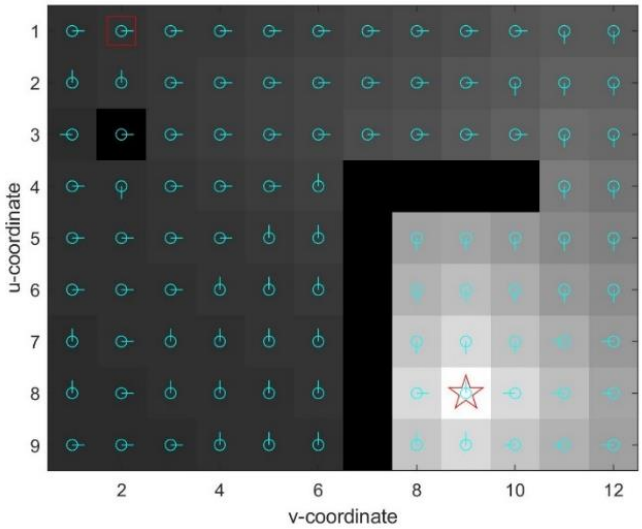

Figure 3: The value function and policy predicted by value iteration for gridworld.

Question b

The policy iteration algorithm [1] can be outlined as the following procedures,

1. Initialize the state-value function V as a zero vector and the policy function π as a one vector, both are in the length of number of states.
2. Initialize the for-loop which includes another for-loop that repeatly improve the state-value function for each state s via the following equation,

$$V_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)\big(r + \gamma V_k(s')\big)$$

   The for-loop within repeats until it converges according to the following equation,

$$\max_{s} |\, V_{k+1}(s) - V_k(s)| < \theta$$

   This step is called the policy evaluation.
3. Update the new policy π′ via,

$$\pi'(s) = arg\max_{a} \sum_{s',r} p(s',r|s,a)\big(r + \gamma V_{\pi}(s')\big)$$

   , if it satisfies the following condition to ensure the new policy must be as good as, or better than, the current policy π,

$$V_{\pi'}(s) \geq V_{\pi}(s)$$

   This step is named as the policy improvement.
4. Repeat until policy becomes stable.

1: Initialise V and π arbitrarily
2: **repeat**
3: Evaluate V using π
4: Improve π using V
5: **until** convergence

Figure 4: Pseudocode of policy iteration.

The policy iteration algorithm is implemented with 10000 iterations for gridworld. The result is plotted as Figure 5.

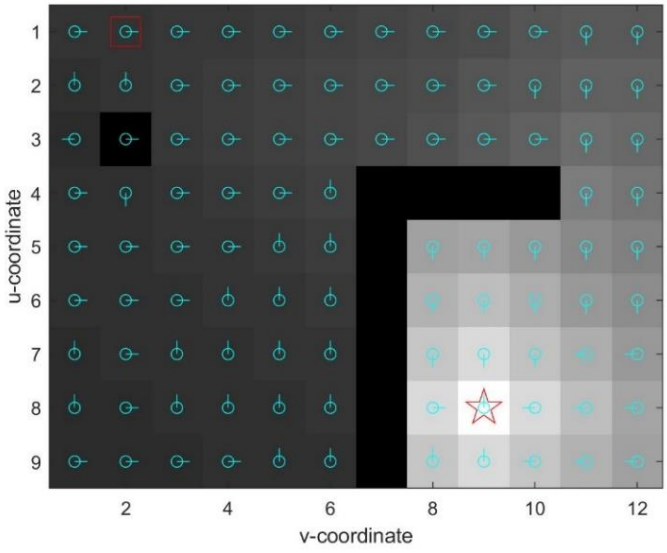

Figure 5: The value function and policy predicted by policy iteration for gridworld.

Question c

The state-action-reward-state-action (sarsa) algorithm can be outlined as the following steps,

1. Initialize the action-value function Q as a matrix of dimension $\mathcal{S} \times \mathcal{A}$, where $\mathcal{S}$ the state space and $\mathcal{A}$ the action state.
2. Initialize the start state as the starting spot.
3. Set the step-size parameter α and the exploration parameter ϵ to be either the constants or the iteration dependent.
4. Choose the action a by $\epsilon-$ greedy algorithm [2].
5. Take action a and observe the following reward r and next state s'.
6. Choose the next action a' by $\epsilon-$ $greedy$ algorithm.
7. Update the Q function via,

$$Q(s,a) = Q(s,a) + \alpha\big(r(s,a) + \gamma Q(s',a') - Q(s,a)\big)$$

8. Update the current state s and action a as the next state s' and action a'.
9. Repeat step 5 to 8 until it reaches the goalstate.
10. Repeat the whole process until the maximum episodes.

1: Initialise Q arbitrarily, $Q(terminal, \cdot) = 0$
2: **repeat**
3: Initialize s
4: Choose a ϵ-greedily
5: **repeat**
6: Take action a, observe r, s'
7: Choose a' ϵ-greedily
8: $Q(s,a) \leftarrow Q(s,a) + \alpha\,(r + \gamma Q(s',a') - Q(s,a))$
9: $s \leftarrow s', a \leftarrow a'$
10: **until** s is terminal
11: **until** convergence

Figure 6: Pseudocode of sarsa.

The sarsa alogrithm is implemented for smallworld. After 1000 episodes with 50000 iterations per episode, the result is shown in Figure 7.

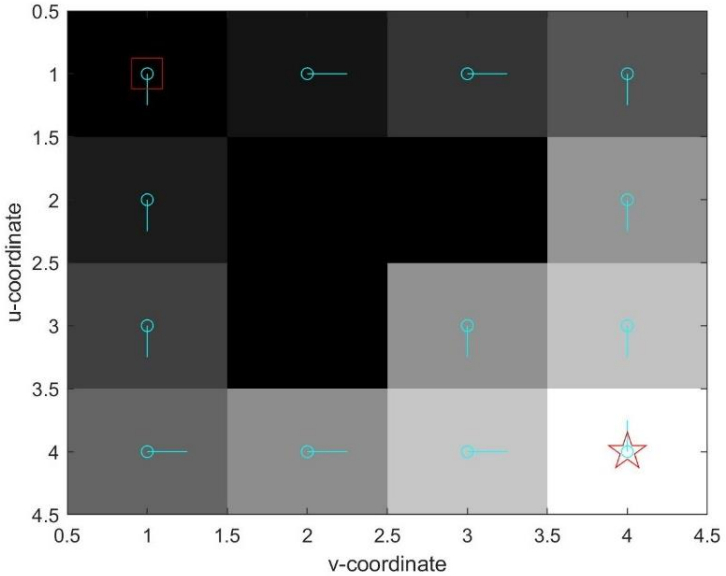

Figure 7: The value function and policy predicted by sarsa algorithm for smallworld.

Question d

The q-learning algorithm can be outlined as the following steps,

1. Initialize the action-value function Q as a matrix of dimension $\mathcal{S} \times \mathcal{A}$.
2. Initialize the start state as the starting spot.
3. Set the step-size parameter α and the exploration parameter ϵ to be either the constants or the iteration dependent.
4. Choose the action a by $\epsilon-$ greedy algorithm.
5. Take action a and observe the following reward r and next state s'.
6. Update the Q function via,

$$Q(s,a) = Q(s,a) + \alpha\left(r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

7. Update the current state s as the next state s'.
8. Repeat step 4 to 7 until it reaches the goalstate.
9. Repeat the whole process until the maximum episodes.

```
1: Initialise Q arbitrarily, Q(terminal, ·) = 0
2: repeat
3:    Initialize s
4:    repeat
5:       Choose a ϵ-greedily
6:       Take action a, observe r, s′
7:       Q(s, a) ← Q(s, a) + α (r + γmax_a′ Q(s′, a′) − Q(s, a))
8:       s ← s′
9:    until s is terminal
10: until convergence
```

Figure 8: Pseudocode of q-learning algorithm.

The q-learning alogrithm is implemented for smallworld. After 1000 episodes with 50000 iterations per episode, the result is shown in Figure 9.

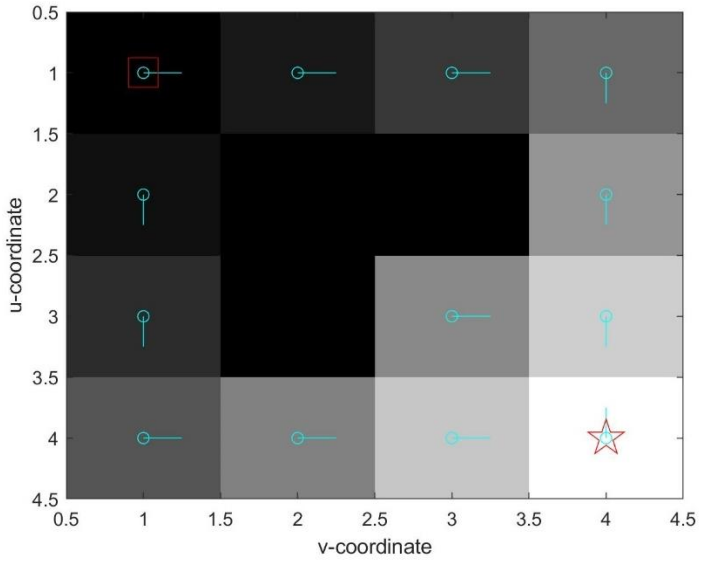

Figure 9: The value function and policy predicted by q-learning algorithm for smallworld.

Question e

The function scripts for the sarsa and q-learning algorithms are modified to include the inputs of α and ϵ, this facilitates the investigation of how they affect the state-value function, policy and the accumulated rewards.
Since the learning process looks noisy, the results are smoothed by averaging the reward sums from 100 successive episodes (similar to 10 successive episodes suggested by [3]).
Before replicating Figure 6.4 from the Sutton and Barto text [3], some experiments on how those parameters affect the state-value function, policy evaluation and rewards sum per episode are done.

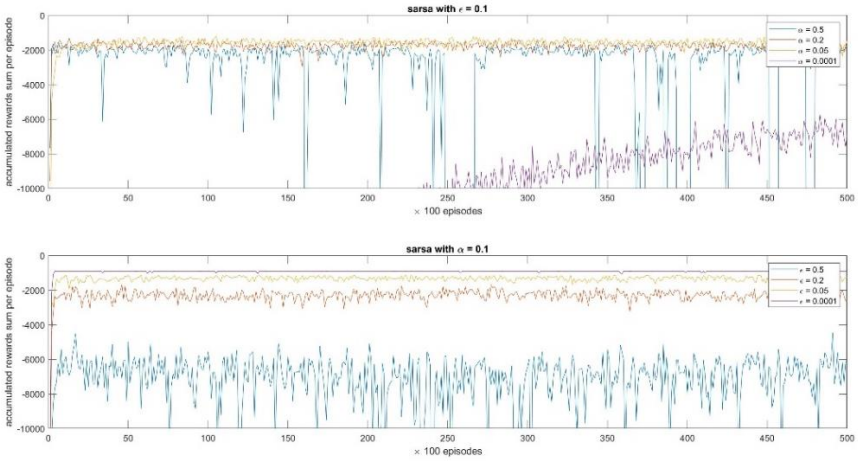

Figure 10: Compare how the rewards sum per episode are affected by α and ϵ in sarsa.

For sarsa, the step-size parameter α is shown to affect how early the model converges in the upper part of Figure 10. When it is too small around 0.0001, the sum of accumulated rewards per episode is still growing so it has not converged yet. But α also cannot be too high, the sum per episode fluctuates more shown by the blue line in the upper part of Figure 10.

In the lower part of Figure 10, the policy parameter ϵ seems to affect the degree of exploration. The higher the value is, the more explorative the algorithm is. When ϵ is around 0.5, the sum of rewards per episode tends to fluctuate more and its variance is greater, which shows it is exploring; In comparison, when it is around 0.0001, the sum per episode over episodes is flatter.

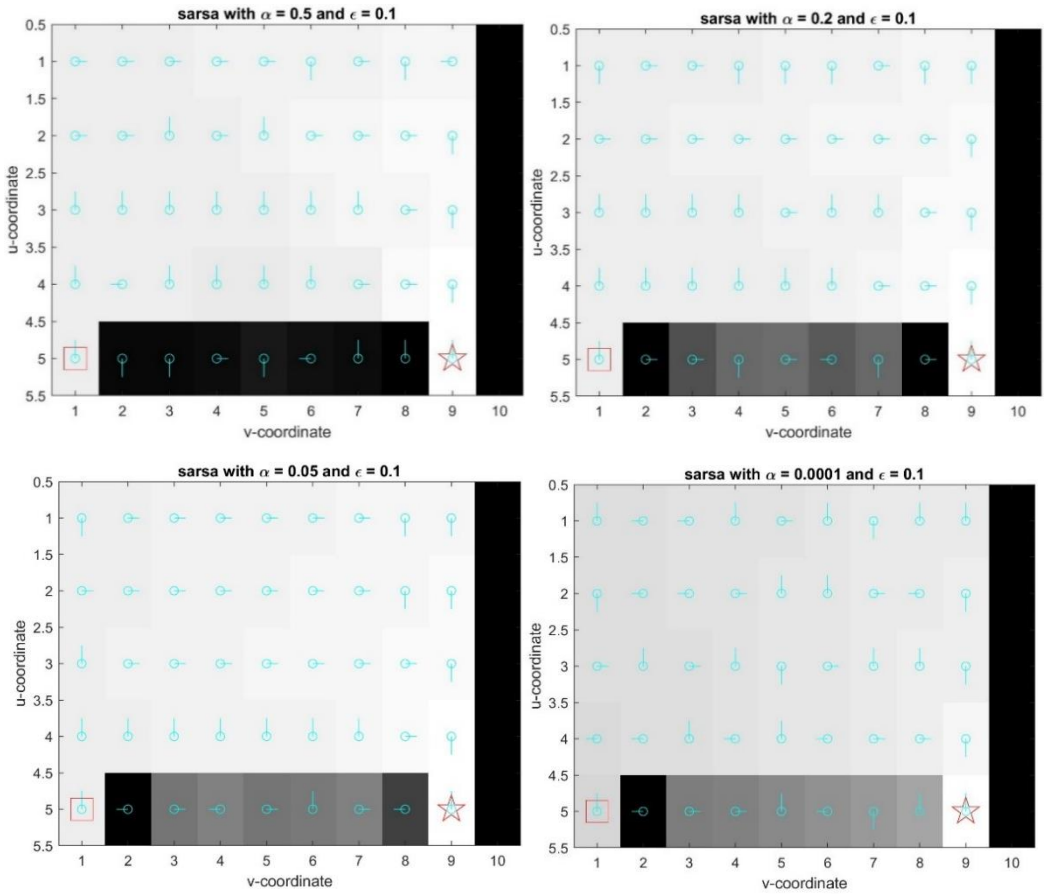

Figure 11: Compare how different values of α affect the state-value functions and policies.

Figure 11 shows a small value of α can lead to the slow convergence which can be illustrated by the non-optimal policy predicted for $(u, v) = (1,1), (4,1), etc.$ in the plot with $\alpha = 0.0001$.

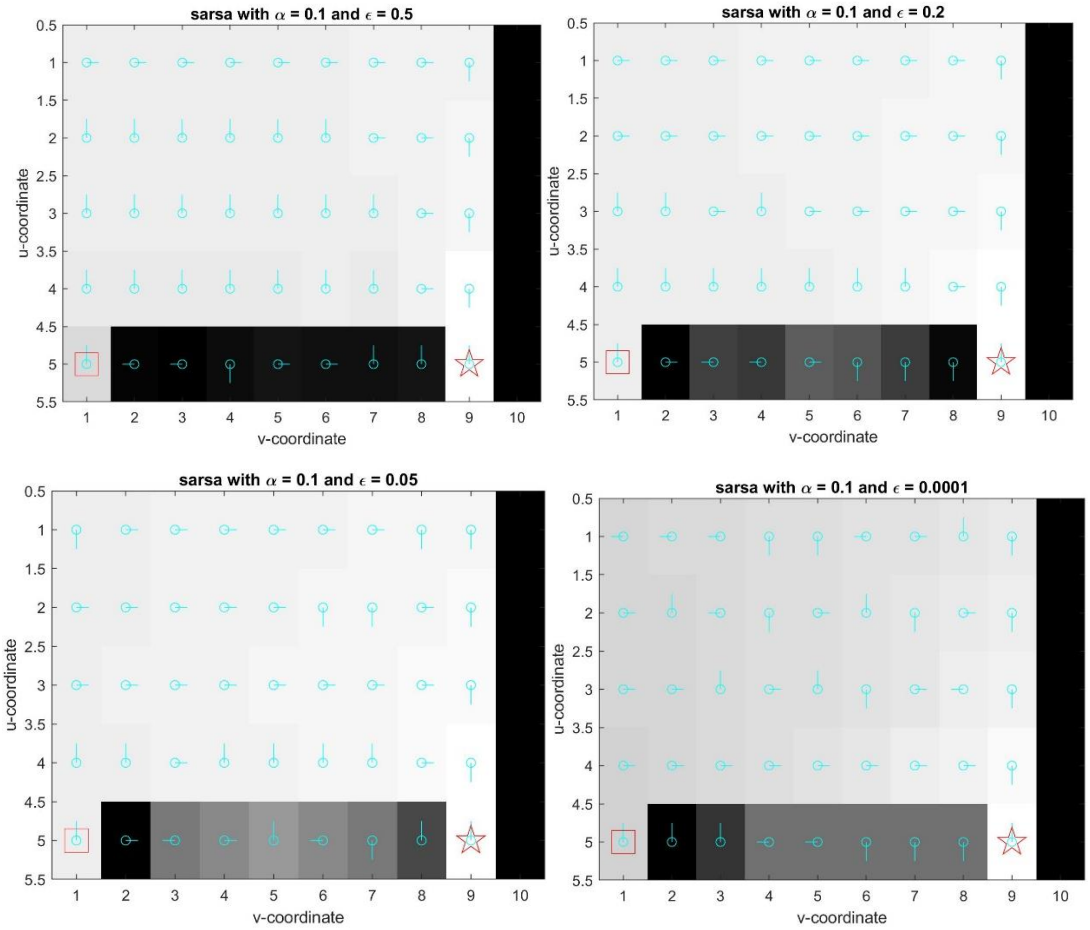

Figure 12: Compare how different values of ϵ affect the state-value functions and policies.

The left-upper part of Figure 12 shows the algorithm learns the longer but safer path through the upper part of the grid. This also implies the higher ϵ encourages exploration, and the behaviour was suggested in the Sutton and Barto text. As the ϵ decreases, the policy chooses the path with shortest distance, and it discourages exploration but encourages exploitation, so the right-lower part of Figure 12 also shows non-optimal policy in $(u, v) = (1,1)$.

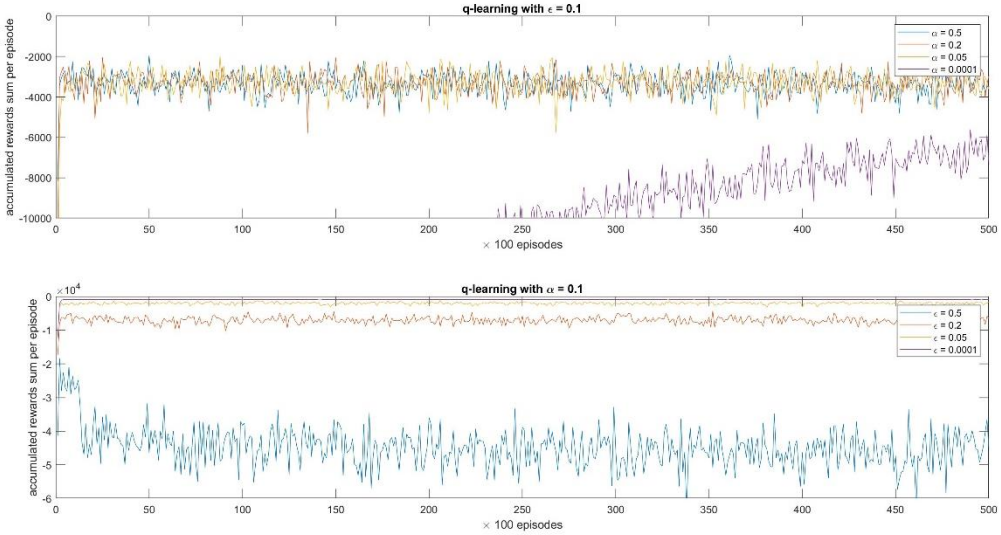

Figure 13: Compare how reward sums per episode is affected by α and ϵ in q-learning.

Similar effects of α are also observed on q-learning algorithm in Figure 13. It also affects the convergence rate or the learning rate. However, the higher value of α does not induce more periodic fluctuations in sum of rewards per episode when compared to how it affects sarsa. Likewise, ϵ also affects the degree of explorations in q-learning algorithm. The sum of rewards per episode fluctuates more when it is set to a higher value.

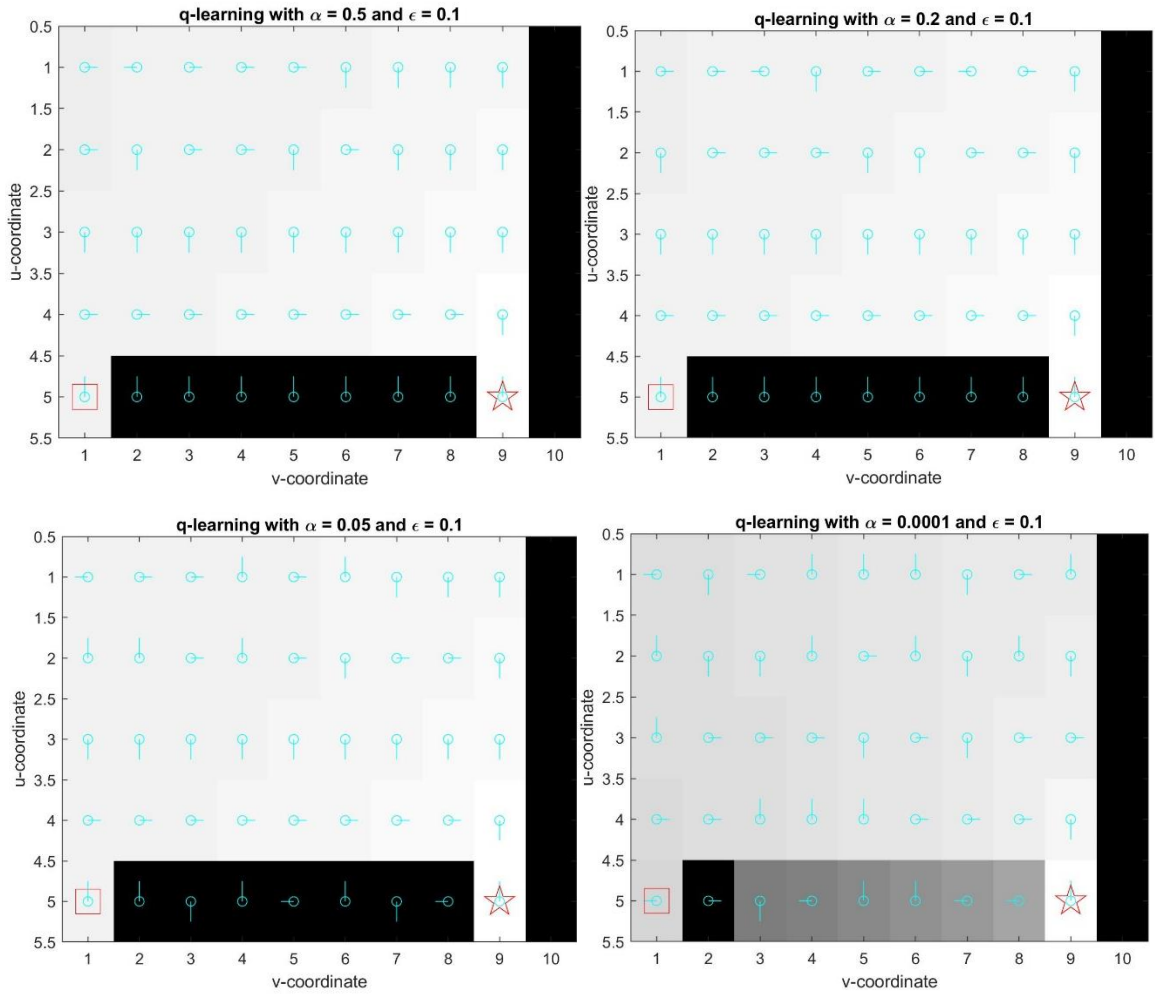

Figure 14: Compare how different values of α affect the state-value functions and policies.

The right-lower part of Figure 14 shows the policy is still not yet converged with low α since some policies for some states are not optimal. And the higher value of step-size parameter seems to help the policy to be converged more quickly as shown in the left-upper part of Figure 14.

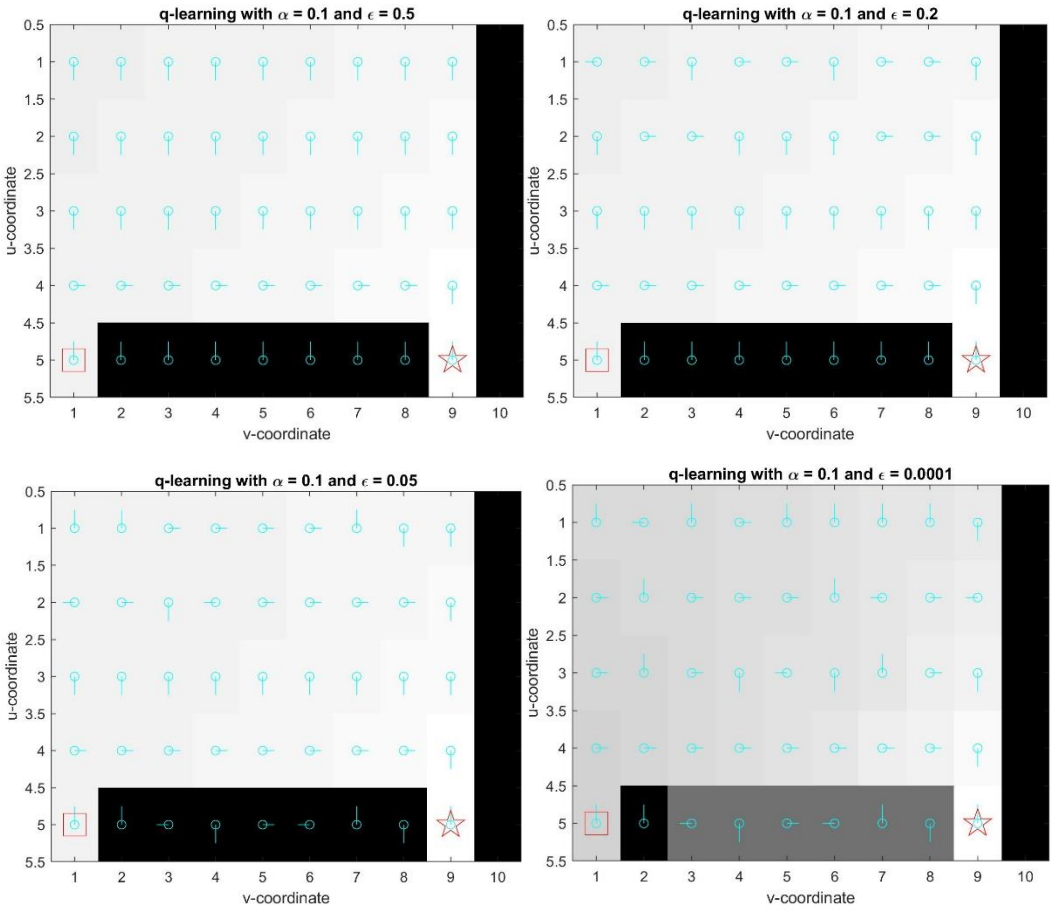

Figure 15: Compare how different values of ϵ affect the state-value functions and policies.

Since the higher value of ϵ encourages greater exploration, the left-upper part of Figure 15 shows optimal policies for all states.

With the paramters of $\alpha = 0.1$ and $\epsilon = 0.1$, the results are plotted in Figure 16.

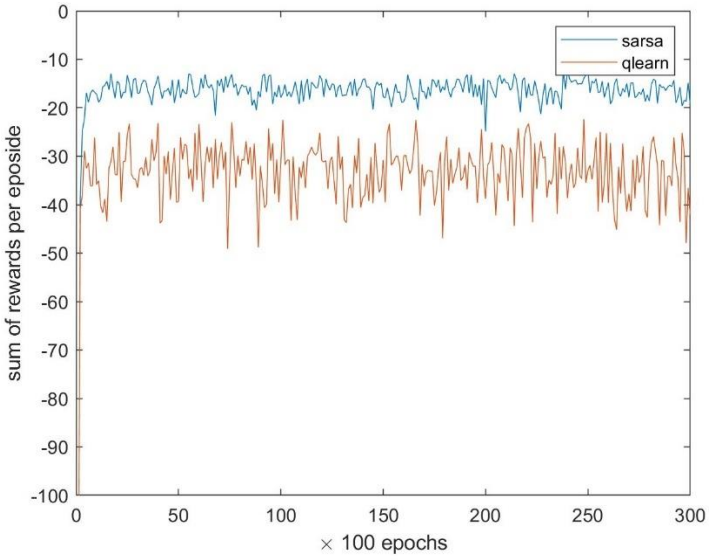

Figure 16: The accumulated rewards per episode for sarsa and q-learning algorithms.

Similar to Figure 6.4 from the Sutton and Barto text [3], the reward sum per episode increases at the beginning and becomes stable afterwards with fluctuations. Although they also converge around 50×10 episodes, their fluctuation is shown to be flatter. This might be due to different settings of gridworld and α.

In general, the reward sums per episode from the sarsa algorithm are greater than those from the q-learning algorithm. Around 5×100 episodes, the reward sums per episode stop to grow but just fluctuate. The reward sums per episode from the q-learning algorithm fluctuates more vigorously than those from the sarsa. This might be because q-learning is an off-policy TD method and sarsa is an on-policy TD method.

Conclusion

Four different reinforcement learning algorithms are tested in this report. All of them are successfully implemented in smallworld, gridworld and cliffworld scripts.

One of the disadvantages of value iteration and policy iteration algorithms is that their predicted policies cannot be adjusted by changing parameters. This might be problematic when the researchers would like to find various policies or adjust the policy characteristics (either explorative or exploitative). Also, value iteration and policy iteration require greater computational power and training time if there is a large number of state variables.

In the experiment, sarsa finds more alternative paths than q-learning. Q-learning can only find one shortest path, and force other policies in states which do not belong to that path to point towards the optimal path.

About the step-size parameter α and the exploration parameter ϵ, the former seems to affect the convergence rate and the latter affects the exploration level. With the greater α, the algorithm converges quicker. With the greater ϵ, the algorithm tends to explore more states.

Reference

[1] Michael Herrmann. University of Edinburgh, School of Informatics. Lecture Slides: RL8: Value Iteration and Policy Iteration. February 6, 2015. http://www.inf.ed.ac.uk/teaching/courses/rl/slides15/rl08.pdf [01/02/2019]

[2] Tim Eden, Anthony Knittel and Raphael van Uffelen, University of New South Wales, Sydney. Reinforcement Learing: TD-Learning. https://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html#aselection

[3] Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction (2nd edition). November 5, 2017. http://incompleteideas.net/book/bookdraft2017nov5.pdf