Reinforcement Learning - Assignment 1

UID: s1045064, University of Edinburgh 02/10/2016

Contents

1 Before start Programming

In the following answer, we are assuming a discount factor γ=0.9 and a learning rate eta=0.1 for a system with states that yield 0 reward assigned apart from reaching the goal were the reward is 10/step size. In addition, we assume that a passenger cannot be initialised in the goal position, as well as that s/he cannot be initialised in the taxi itself but only in one of the pick-up states. The correctness of those two rules were discussed with and approved by Dr. M. Herrmann.

1.1 Typical time horizon of the problem

In the best case scenario, let us assume that the taxi is in position (2,1) (located underneath R) and that the passenger is in R. Ideally, it will take the algorithm 1 step to get to the passenger. Then, let us assume

that the drop-off destination is B. Given, that the best route is 8 steps assuming we are only considering the optimal scenario and thus set a relatively average time size of 10 (which is equal to $\frac{1}{1-\gamma}$). This can be obtained by counting the steps from the pick-up location to the goal. Considering the episode when the passenger is already in the taxi as our initial time step. This will take 7 steps to get to the goal and one time step to drop the passenger. Regardless, worst case scenario will include the taxi having to go from one side of the map to the other and then take the passenger to the side the taxi started from. Worst case scenario, in an already known environment will take about 20 time steps. That is, given we have the fully learnt policy.

1.2 Mean (immediate) reward for a good policy

The immediate reward for a given state and action can be either -1, 0 or 10/steps to the goal. A good policy assumes that the times the taxi will do things which requires penalties is going to be relatively small. Therefore, we think that the immediate reward will be between 0 and 1. My reasoning is backed up by the assumptions made in the previous sections. If it takes a taxi driver about 10 steps on average to reach a goal destination, the reward will be 10/10, 1.

1.3 Maximal value of a state-action pair

The value of a state-action pair is a result from the so far observed dependency between the learning rate, discount value, epsilon, immediate reward per state at which the taxi has been in a given time in the past, the currently obtained value for the state-action of the present as well as the best action known for the given state. Therefore, the value of a state-action pair will continuously grow until the change becomes negligible, or in fact vanishes. In the so far defined setting, the value for the state-action pair is mostly dependent on the past experience. Assuming that γ is 0.9, and that T is 10000, the following holds. For any state V_t and V_{t+1}, where $t \in T$, the next state's value will increase with a factor of γ, i.e. γ^t. However the bigger t, the closer to zero γ will be until it finally converges - i.e. γ^{8000}=0 and a lot before that, the value of γ^t will be negligible. Regardless, convergence will be checked by using $\eta(rMax + \gamma V(s_0) - Q_t(s_0, a_0))$. In addition, the best reward a taxi can get is around 1.25. Therefore, I am assuming that the maximum value of a state-action pair will be about 10, 11.25, to be more precise. The formula used was $\frac{maxReward}{(1-\gamma)}$.

1.4 Number of trials that a standard algorithm will need in order to find a good solution

Assuming the best case scenario, where the taxi will always start from the same point. The rest of the paragraph is based on assumptions and does not provide exact calculations but rather aims to provide the reader with a relatively accurate expectation of the expected behaviour of our solution. Let us assume that the taxi is in position (2,1) (located underneath R) and that the passenger is in B, implying that the taxi will be able to pick up the passenger in about 8 steps. This will be repeated about 8 more times so that the reward can propagate to the start point. At any state there are 6 possible actions to be taken. On average, each action has about 17% chance of being selected. With each incorrectly chosen action, its chance to be selected again decreases, increasing the chance of the other actions. Regardless, we would have 8 steps with 6 possible states each and we will need to go through the entire path about 8 times to

propagate the reward. We should not neglect the fact that we will also take wrong steps. This will assign negative rewards to some of the states throughout our learning. Thus, the amount of repetitions required for the agent to start picking up a good solution will reduce. Therefore, even if a given state hasn't yet received the propagated reward from the reached goal, there is a high chance it will have already received negative rewards based on other actions taken by the taxi. So, we assume that instead of repeating 8 times, we will only need to repeat about 2 or 3 before obtaining a good enough policy able to achieve good solutions.However, this will increase the steps taken. Having said that, we assume that on average the taxi will have to go through 15 to 20 positions before reaching its goal. This, combined with the 6 possible states the taxi can take results in about 120 (6*20) different steps. Since the taxi will need to repeat those steps two or three times this will lead us to an average of 240 to 360 trials. Since we have four different goal states we will end up having relatively a 1000 trials until finding a good solution to the problem for all four goal scenarios. The position of the pick-up location would not matter as much as if the taxi knows how to get to all four places then it wouldn't matter as much where it starts from.

1.5 How can those evaluations be applied and used in the implementation

Thinking in advance about the answers to the questions posed in the subsections above can significantly help in the process of tailoring a solid rule system and thinking about specific scenarios. For example, we decided to initiate counting of the steps from point A to point B only after a passenger has got in the taxi instead of counting the actual state of getting in the taxi. In addition, we have decided to disregard the cases where a passenger can be initialised in its goal position. The reason for this is that we think that in a potential scenario like the one explained above, the passenger would not practically need a taxi and even if they did, this would bring close to no reward to the taxi itself. Thus, instead of over-complicating our system and diverging from our aim, we decided to not allow for such events. Thinking about those problems in advance enabled us to consider a scenario where the passenger is initialised straight in the taxi- if that happens what would be a potential pick up destination, how could we know and if we didn't how would we estimate the goal reward. Therefore, we decided to remove the option for a passenger to be initialised in the taxi but rather only in one of the four pick-up locations. We discussed this with dr. M. Herrmann who approved of that logic and we have thus based all our reasoning above with those assumptions in mind. Calculating rough estimates of what to expect, given our defined rules and corner cases also allowed us to test our results obtained from the actual implementation. It makes the entire debugging process a lot more sensible and helps for conducting the correct experiments.

2 Solving the problem using Q-Learning

2.1 Performance of the algorithm for random starting and random goal positions

As in the previous section, the values used for learning rate, discount variable and epsilon are the same. Comparing the estimates from before, we realised we were not that far off with our predictions. On a training with 10000 trials, we obtained a mean reward of 0.9048 and a maximum value V(s,a) of 11.2409. However, observing the results, we noticed that the pre-assumed 1000 trials were not entirely correct. In fact, in this specific attempt, the learning rate started improving comapred to its initial value after the 700-800th trial as shown on Figure 1. The figure illustrates a plot of the average Q reward against all

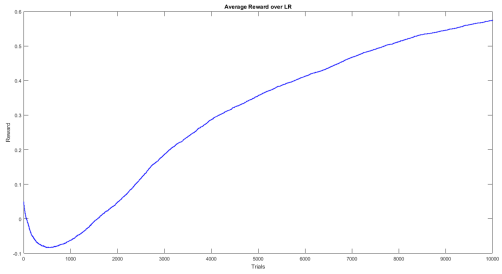

Figure 1: Average reward over learning time

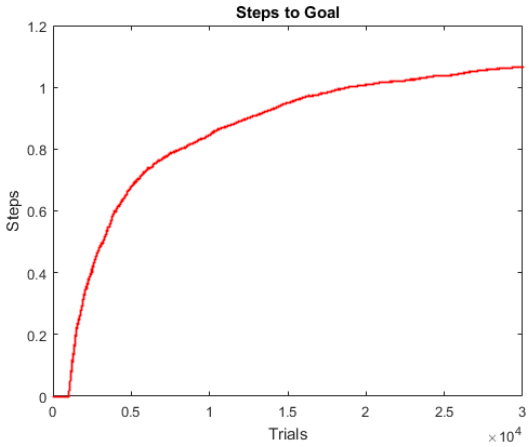

Figure 2: Mean reward per state

episodes. In fact, as it is always the case with probabilities, these values vary (see Figure 5 to Figure 8). Figure 2, however shows a plot of the mean reward taken over time steps. It can be easily seen how for the first couple of hundred steps the mean of the reward obtained was zero but after that started improving. Comparing the maximal value of a state-action pair, again assuming a learning rate of 0.1, discount factor of 0.9 and a random factor of 0.1. Here, depending on the pick up and goal locations, the values will vary. To compare it with our initial estimates, we ran the algorithm with the same as previously assumed start and stop locations. We obtained 11.2409 as stated above which is with less than a 0.01th different than what we proposed. This showed us that the maximum reward values can fluctuate a bit. And this makes a lot of sense as the algorithm can fluctuate between negligible values around the actual one. It is interesting to note that we did not initially take into consideration the randomly assigned Q values and the fact that in our calculations we did not take into consideration the action in which the taxi picks up the passenger and the reward it gets 1.

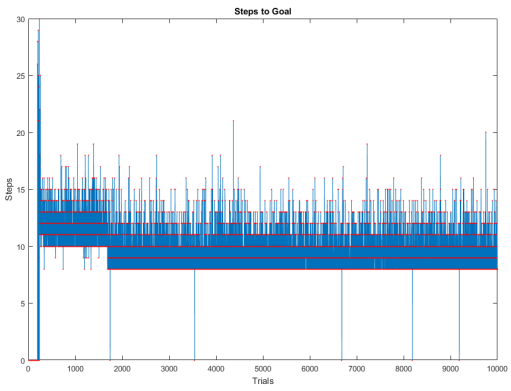

Figure 3: Number of steps to goal

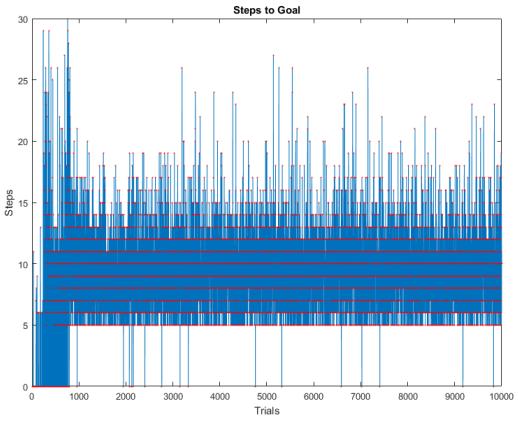

Figure 4: Same but all possible goals and pick-up states considered

In order to comment on the assumption about the typical time horizon of the problem, we had to plot

the number of steps taken for the taxi to take a passenger from point A to point B. Figure 41 shows the relationship between the number of steps taken to get to a goal, spread across the first 10000 trials. Note, how the first 200 trials did not have almost any journeys. The reason for this is that it took the taxi driver more than 30 steps to reach the destination which led to the preliminary leaving of the passenger. This is another evidence that it will take 1000 steps on average for the taxi to learn the good policies. To conclude this, we trained the algorithm with the exact same parameters randomising all possible goals and pickup locations. The results are shown on Figure 42 Regardless, another thing to be mentioned is that some of the episodes did not result to getting the passenger to te goal state. Although this could be due to a bug which had not been found, it could also be caused by the random factor introduced in Q-learning.

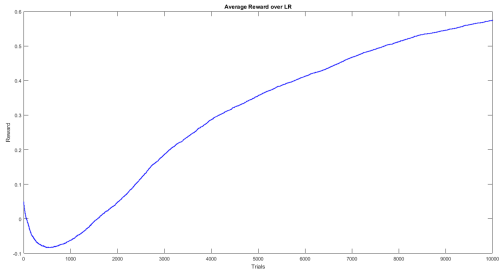

Figure 5: First Run

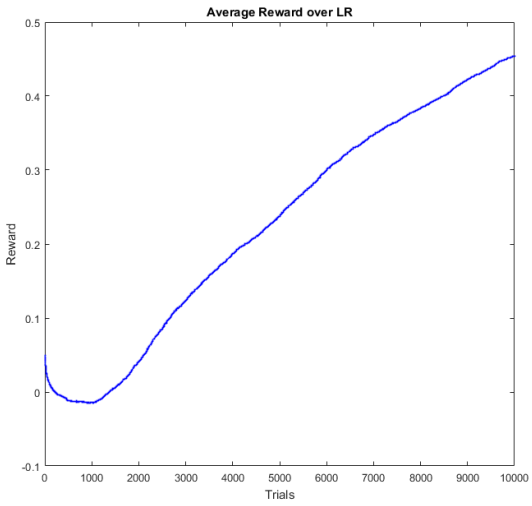

Figure 6: Second Run

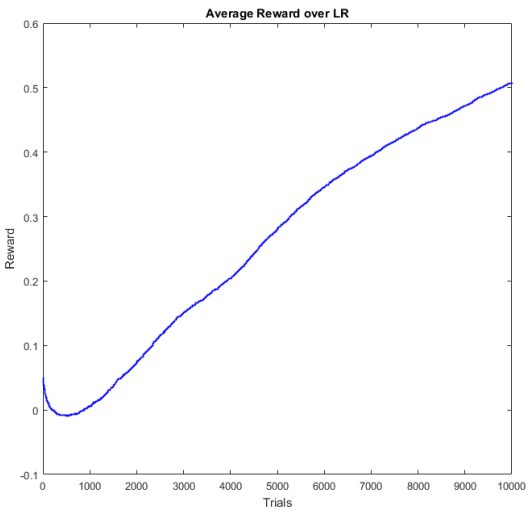

Figure 7: Third Run

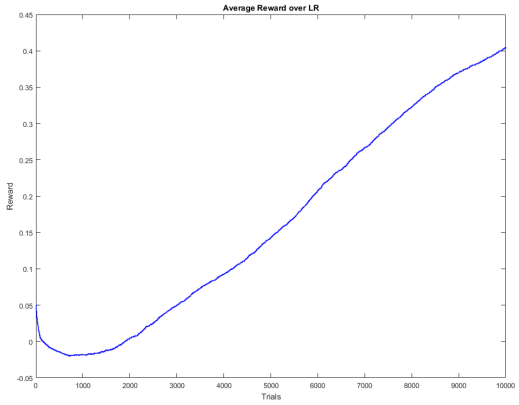

Figure 8: Fourth Run

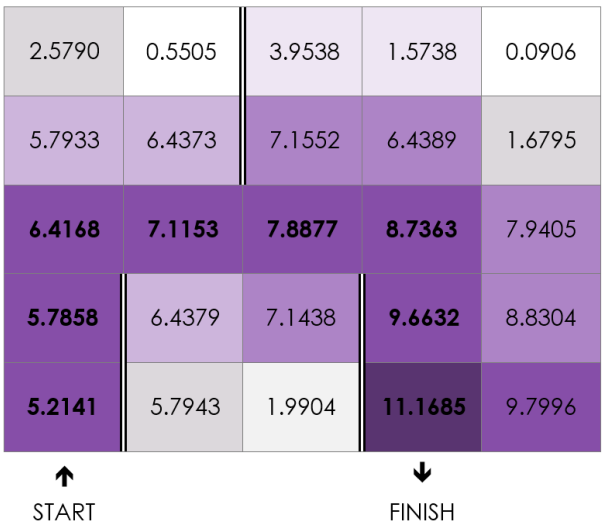

Figure 9: Spatial spread for Maximum Values

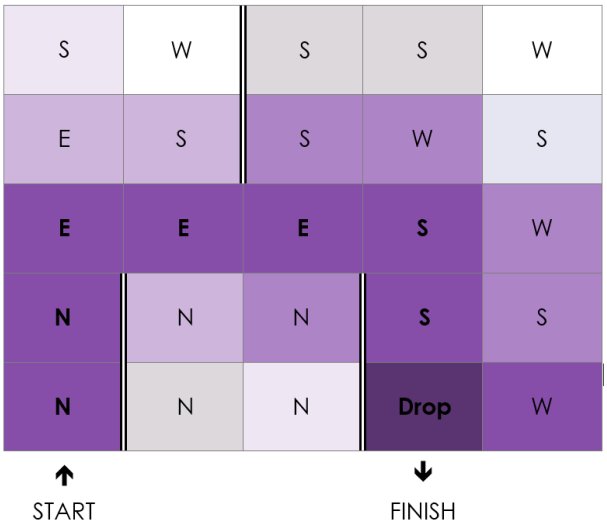

Figure 10: Spatial spread for best actions

2.2 Represent your solution using an example

To show that our solution works we set the initial passenger position at "Y" and goal at “B”. We then plot the value of the spatial states and the best action for each state as shown in Figures 9 and 10. In the former, the values presented are the actual maximum values obtained for each state. The bold numbers trace the path through the maze from ’START’ to ’FINISH’. The higher the value, the darker the purple colour. The lowest values are in white squares. The latter figure follows the same colour coding, however has letters instead of numbers in it. Each letter represents an action. ’N’ is for North, ’S’ is for South, ’E’ is for East, ’W’ is for West and Drop represents the drop-off state. Note how the reward value has propagated back to the start position. Another thing to be noted is that the stepsize from ’START’ to ’FINISH’ is again 8, discarding the step in which the passenger gets in the taxi. However, the reward is less than the calculated 11.25 from the previous question. As already discussed, the value can fluctuate within some small variations and that is completely acceptable. Another interesting thing to be pointed is that the algorithm has managed to explore the entire maze assigning values as expected. Another representation of our solution is plotted in Figure 11 which shows how the closest path to the destination has obtained the highest reward and has thus resulted in the shown figure. It is important to note that the figure represents the five by five grid where each edge represents a state as opposed to each square. Note how the goal has received the highest value. Another representation of the example is shown in Figure 12. There, it is easier to see the different values obtained and compare them with the rest of the options taken while exploring the path and rewarding each action in the states obtained. To get from position 21 (point A) to the goal in position 24 (point B), the taxi had to first get to the passenger and then take them to the desired destination. On the graph, the circles represent positions the taxi has been to. It is easy to notice how the path, presented in Figure 11 has the highest values in Figure 12. The quickest route to point B is the following: starting from point A, in 21, the taxi goes through points 16,11,12,13,14,19 until it reaches point B in 24. However, Figure 12 has all 6 states plotted. Compare the results with Figure 13 where the only plotted values are from the vector representing state 6, drop-off. The results from there show how the reward from the goal state has propagated through the states until the pick-up location. Figure 14 is

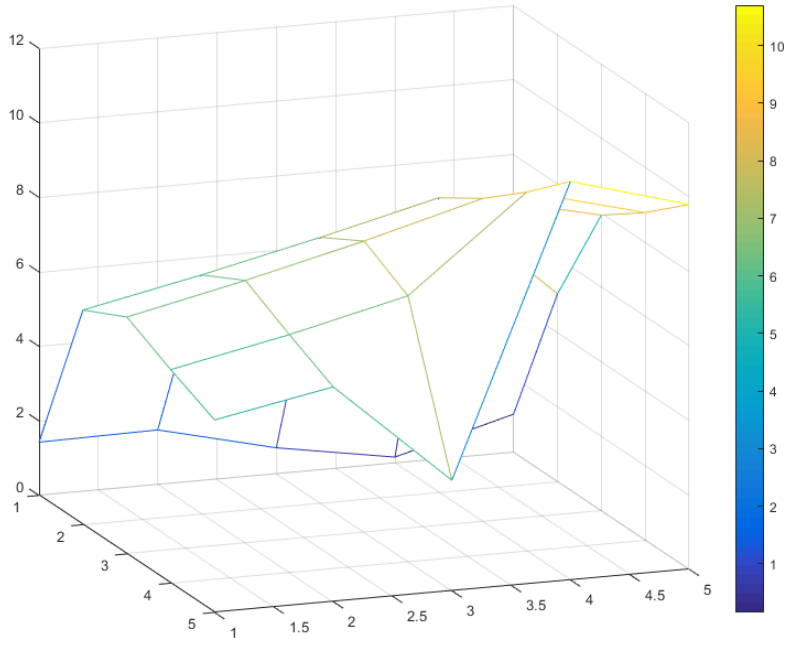

Figure 11: Spatial representation of reward obtained.

a 5 by 5 grid representation of the solution. Here, the warmer the colour the higher the reward obtained for it. Note how it's not the blocks that are of interest but their edges and walls. The taxi will go through edges (1,5) to (1,4), (2,4), (3,4), (4,4), (4,5). However, the grid cannot simply represent the dots it went through but it represents each wall. Which is why the block starting from (4,4) and ending in (5,5) has different coloured walls. Note how the (3,5) to (4,5) wall is blue, representing low value. Relate with Figure 13, the value at 25 (i.e. 5*5) is low compared to 24, for example. 24 is represented as the wall starting at (4,4) and ending at (4,5).

2.3 Consider the time course of the values and the reward and define a convergence time for the algorithm

The following consideration is done under the observation of the Q-learning algorithm, with $\eta = 0.1$, $\gamma = 0.9$ and $\epsilon = 0.1$. The pick-up state is at 'Y' and the goal state at 'B'. The time course of the reward, as show in Figure 15 shows the spread of reward throughout the trials for a set goal and a set passenger location. A dot represents a single reward, where some rewards fall to -1, others stay at 0 (which are then represented by a simple dot) and others go above to 10/step-size. This is useful as it can be clearly seen when the taxi has started reaching the goal destinations at around the 100th trial. Moreover, it can be easily seen that the taxi learns how to get to the goal through the shortest path quite quickly and starts picking up maximum rewards around the 300th trial. This is a good example to show that even though a taxi learns a specific route to a goal, it will keep on improving until it finds a better one and continue doing so until convergence.

To effectively show the performance of our algorithm, we ran it with the same parameters as above, however for all possible states and goals. Our results showed the convergence time as shown on Figure 16.

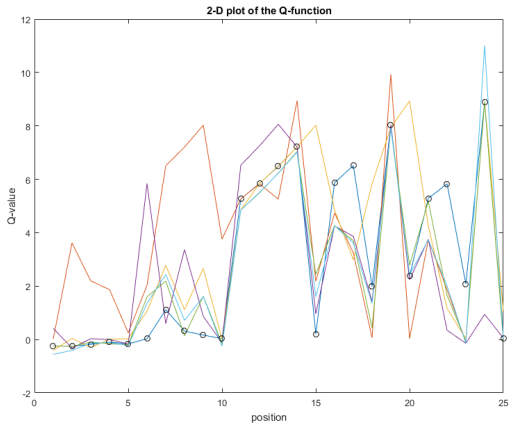

Figure 12: 2D representation of the spatial state-value representation.

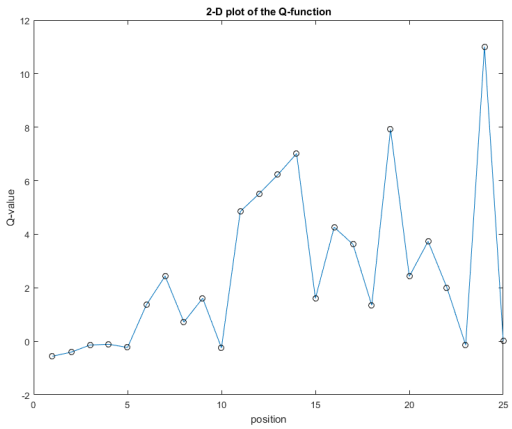

Figure 13: More constrained and thus clear representation of the spatial state-value relationship.

Figure 14: A 5 by 5 grid representation of the solution.

We can observe that algorithm converges at around 3000th trial. The fluctuation can be explained by the fact that the maximum reward obtain can belongs between +/- 1 from the actual maximum reward obtained

by $\frac{maxR}{1-\gamma}$. The time course value comparison presented in Figures 21 to 24 show different examples of the convergence time presented. However, a more detailed explanation is provided in the next subsection. The red lines represent the change in $\eta(r + \gamma(maxQ(s', a)))$ and the blue ones represent the changes in the new Q-value over time.

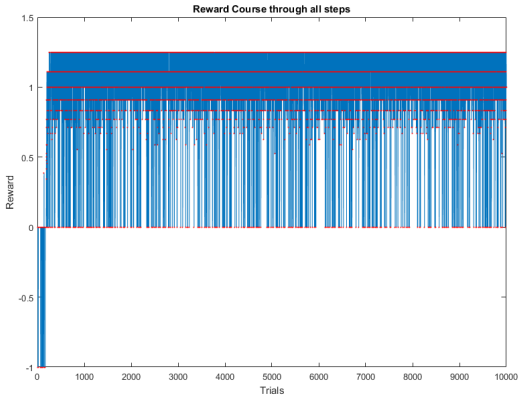

Figure 15: Time course of single reward for one case.

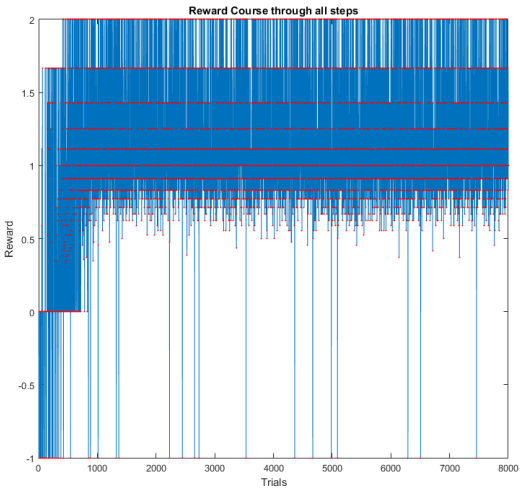

Figure 16: Time-course reward for all cases.

2.4 Discuss, possibly using numerical examples, how the convergence time depends on the exploration strategy and other parameters of the algorithm

Fixed goal and position: To learn to make good decisions, the Q-learning algorithm is dependent on three parameters namely the exploration factor, ϵ, the learning rate η and the discount property, γ. Usually, often picked values for those three parameters are the following: $\epsilon \in [.1, .3], \eta \in [.01, .1]$ and $\gamma \in [.9 - .96]$. In this subsection, we plot different values for those parameters and assess the results obtained, showing how the overall end result depends on the good selection. Figures 17 and 18, Figures 21 to 24 and Figures 25 to 28 contain various plots with different parameters, the title of each plot contains the values set for those parameters. It is interesting to observe that for learning rate of 0.5 the actual learning is fluctuating a lot and does not converge. That is normal because the step the algorithm adapts a the new values is quite big resulting in jumping from one side of the convergence slope to another, missing the actual value at which it would converge. As expected, reducing the learning rate reduces the time of convergence - i.e. it happens after 5000th trial and reduces the amount of influence the newly obtained rewards have to the updates of the Q values. Keeping η and ϵ 0.1 and reducing γ to 0.1 leads to a drastic change of the learning rate in comparison to the smooth raise observed so far. The reward drops exponentially and the new values accumulated from the newly obtained rewards have prior to none influence on the updates of Q. This is due to the smaller discount rate of rewarding. Further reducing the discount rate will increase the gap. Using 0.5 as discount rate and keeping η at 0.1 leads to a good compared to the examples obtained so far convergence at about 5000th trial. This is very interesting as in fact shows that decaying the value by a half discounts the reward obtained but also shrinks the influence the new values have on Q. Increasing the learning rate by a bit to 0.2 leads to more 'adventurous' behaviour of the algorithm as

presented on Figure 24. It does however impose more 'noisy' values for $\eta(r + \gamma(maxQ(s', a)))$ which prevent the algorithm from complete convergence. Exploiting the random factor in the equation leads as expected to more random behaviour. Figures 25 to 27 show the behaviour of our algorithm when using different values for ϵ. Notably, this value can be very useful in fine-tuning the algorithm when attempting to obtain the best possible results though it can be quite tricky to adapt as it has unpredicted outcomes. While randomly exploring new states can be advantageous it can often impose danger and in fact lead to a never-converging algorithm.

Once we obtained a sufficient understanding of how each variable behaves, we attempted to find a better value which will result in a closer to zero value in comparison to our base case of 0.1,0.9 and 0.1 as introduced so far. To evaluate and compare our results we used $\eta(rMax + \gamma V(s_0) - Q_t(s_0, a_0))$. Although we did not manage to manually find the best results, we suspect that after running a grid search over all possible parameters we will find the values that will lead to convergence, i.e. the ones that result in $\eta(rMax + \gamma V(s_0) - Q_t(s_0, a_0)) = 0$. However, due to time constraints we could not perform that experiment. Regardless, the closest value to zero we achieved, 0.0066, was with $\eta = 0.01$, $\gamma = 0.85$ and $\epsilon = 0.06$. We plot the Q values and the mean in Figures 19 and 20.

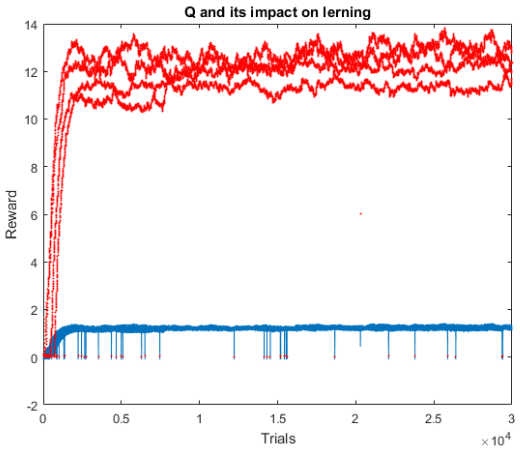

Figure 17: $\eta = 0.1, \gamma = 0.9, \epsilon = 0.1$

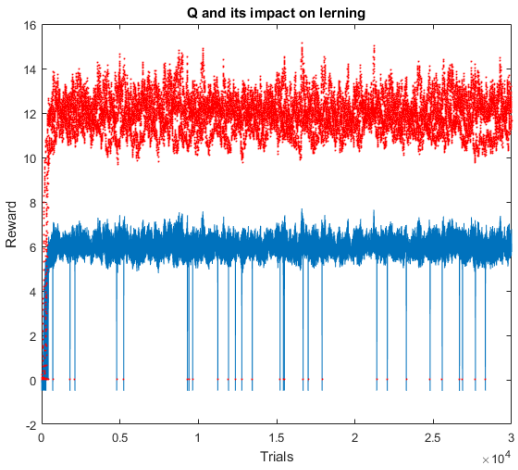

Figure 18: $\eta = 0.5, \gamma = 0.9, \epsilon = 0.1$

3 Solve the problem using SARSA and answer the same questions as above.

To make sense of the rest of this section we define the parameters we used in evaluating SARSA. As in the previous section, we used the same initial parameters - namely $\eta = 0.1$, $\gamma = 0.9$ and $\epsilon = 0.1$. We ran our algorithm on a fixed goal and pick-up state -i.e. a client in 'Y' who wants to go to 'B'. The averaged reward over the time steps we obtained was 0.0323 which is three times smaller than that obtained for Q learning. Compared to question one's estimates, this still fits within our estimated bounds between 0 and 1, however tends to go towards zero instead of one. The maximum value V, is 9.7757, which is smaller than that of Q, and does not fit in +/- 1 of the estimated value in section 1. The Q values, however seem to converge at about the 100th time step (see Figure 33), which is considerably fast. Regardless, they are twice as far away from leading to a convergence according to $\eta = 0.01$, $\gamma = 0.9$ and $\epsilon = 0.1$, averaging

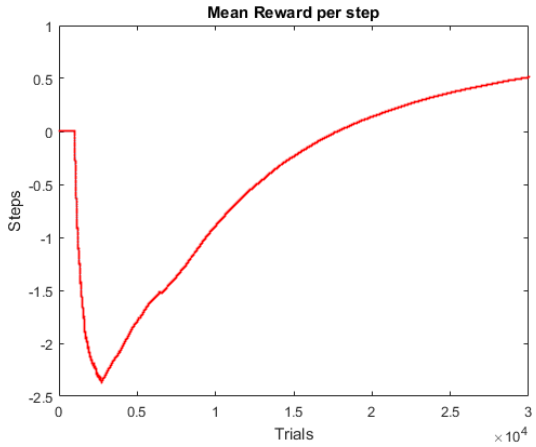

Figure 19: Mean Reward for $\eta = 0.01, \gamma = 0.85, \epsilon = 0.06$

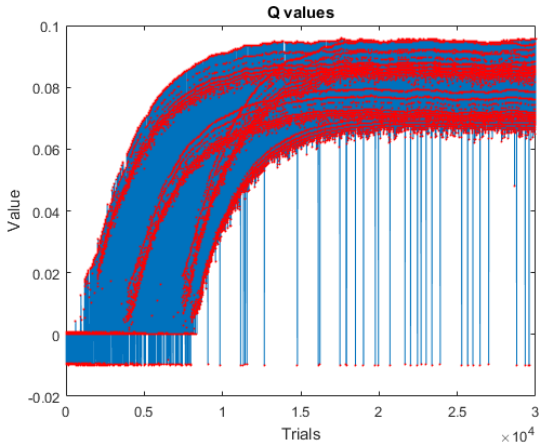

Figure 20: Q values for $\eta = 0.01, \gamma = 0.85, \epsilon = 0.06$

around 0.0391, as opposed to 0.01 for Q-learning. This implies that we will need different values for those parameters in order to optimise our solution. However, this was expected as we are observing a different algorithm. In order to comment on the assumption about the typical time horizon of the problem we plot the steps to goal calculation where we again assume that the taxi should not take more than 30 steps to get the passenger to the goal. comparison shows that SARSA manages to start getting less than 30 times steps to take a passengers from pick-up point to goal before the 100th episode (see Figure 32). Figures 34 and 35 show the steps taken for the grid again assuming the same states as goal and destination. Here, we do not account for the state of the passenger getting into the taxi. However, the reward is less than the calculated 11.25 from the previous question. As already discussed, the value can fluctuate within some small variations and that is completely acceptable. Another interesting thing to be pointed is that the algorithm has managed to explore the entire maze assigning values as expected. Figure 36 shows a different representation of the same maze, however in 3D. Again, the figure represents each state as an edge on the plot rather than an entire square. Notice, how regardless we use a different algorithm and apart from a few values changed, the steps taken are again the same. This is because we do not have a 'cliff' representation in our task. More on that is presented in the next section. Figure 37 shows the spread of reward throughout the trials for a set goal and a set passenger location. Again, a dot represents a reward where some rewards fall to -1, others stay at 0 (which are then represented by a simple dot) and others go above to 10/step-size. We can see that for this scenario, where we consider only one case, the value converges at around 500. Figures 41 to 43 show that the dependency learnt from before still applies - namely the three parameters are quite important for handling the performance of the algorithm. Something else can also be noticed. SARSA looks quite similar to Q learning, however the values are always slightly lower. Regardless, the reasons why that is the case are pointed in the next section. An interesting observation is the dependency of SARSA with the random factor. The random factor results in very low and random values as one would expect, given that SARSA is an On-policy algorithm always aiming at finding the best policy that still explores. In this case, SARSA does not find the path to the goal. Another such observation is the correlation between SARSA and the learning rate. Here, the same observations apply for the learning rate results- smaller values yield better accuracy in fact, they get a lot

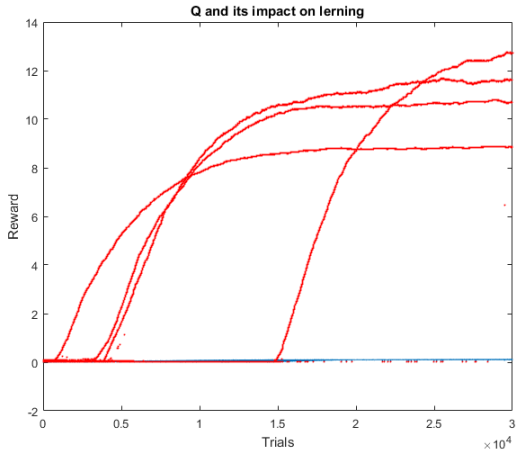

Figure 21: $\eta = 0.01, \gamma = 0.9, \epsilon = 0.1$

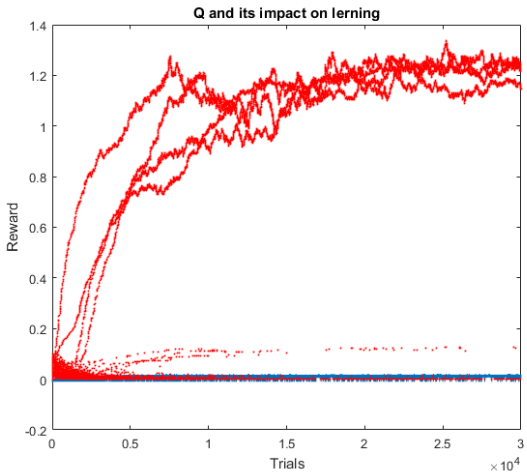

Figure 22: $\eta = 0.1, \gamma = 0.1, \epsilon = 0.1$

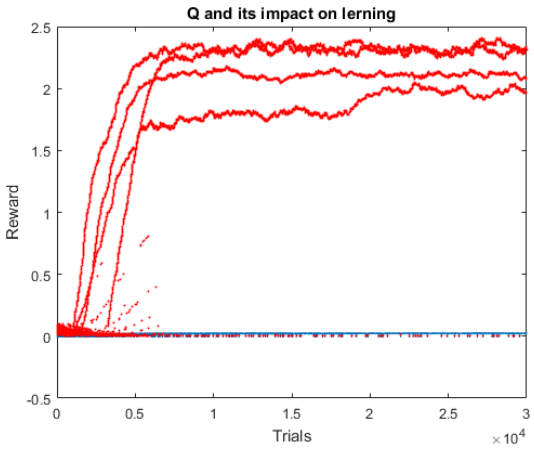

Figure 23: $\eta = 0.1, \gamma = 0.5, \epsilon = 0.1$

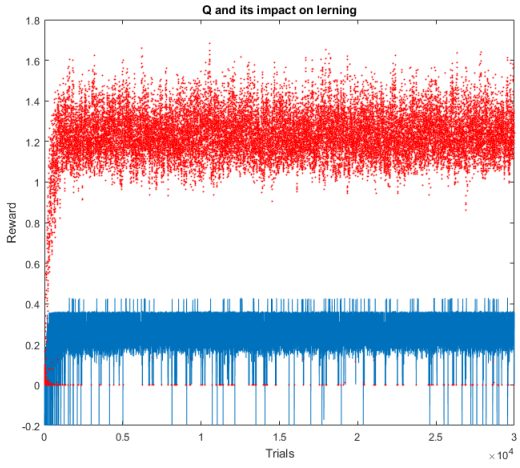

Figure 24: $\eta = 0.2, \gamma = 0.9, \epsilon = 0.1$

closer to convergence than what we would expect. $eta = 0.1, \gamma = 0.85, eps = 0.06$ resulted in 0.0023 after calculating the significance of the next change. This is quite surprising as we would expect SARSA to perform better with higher learning rate.

Regardless, the best results we could have obtain, can be again achieved through the adoption of a grid search algorithm. However, due to time constraints, we were forced to run a few experiments manually and try and predict how to obtain the best value. As already mentioned, we did not see what we expected. Namely that increasing the learning rate by a bit improves SARSA's performance. However, once we obtained the results we realised that this was the case because there was no 'cliff' to keep the agent away from and the fact that the penalties were uniformly distributed lead to obtaining the currently assigned results.

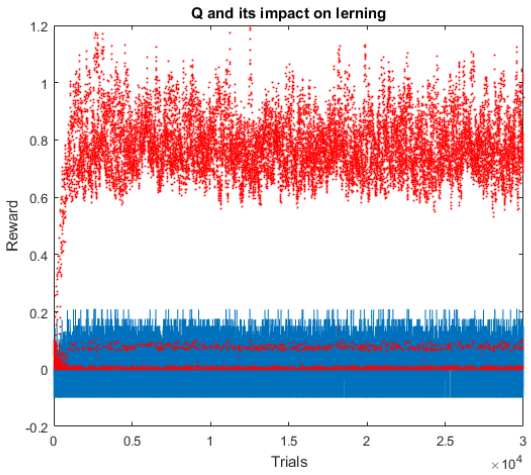

Figure 25: $\eta = 0.1, \gamma = 0.9, \epsilon = 0.5$

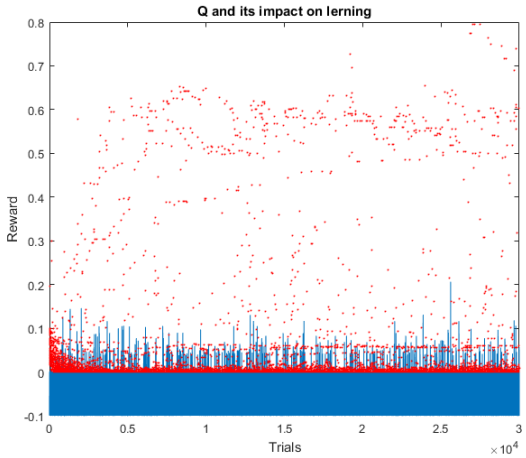

Figure 26: $\eta = 0.1, \gamma = 0.9, \epsilon = 1.0$

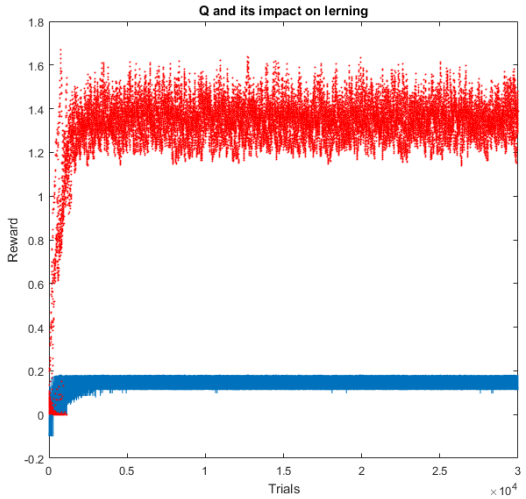

Figure 27: $\eta = 0.1, \gamma = 0.9, \epsilon = 0.0$

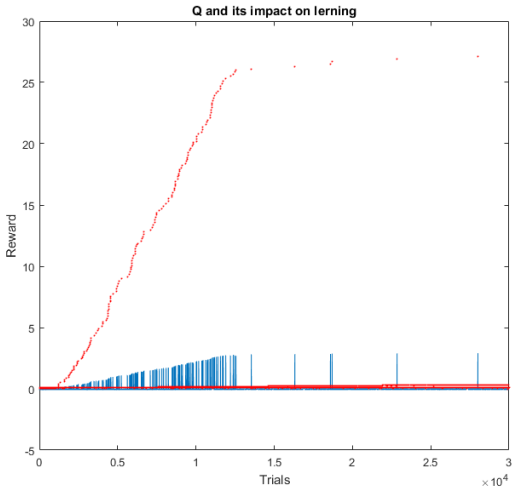

Figure 28: $\eta = 0.1, \gamma = 1.0, \epsilon = 0.1$

4 Compare the performance of Q-learning and SARSA on the present version of the Taxi problem.

SARSA, as an on-policy algorithm attempts to improve the policy that is used to make decision, compared to Q-learning which is an off-policy algorithm. In the case for Q-learning, the algorithm evaluates one policy while following another. Even though in general the latter can be slower since only the part after the last exploration is reliable, we saw that this is not really the case for our problem as we do not have big penalties for specific states. Even though it takes about two times longer for Q-learning to converge, we find those values negligible compared to the tens of hundreds episodes the algorithms ran for. Over 10000 trials, Q learning managed to just as quickly learn the problem and understand how to get from point A to point B. On the other hand, even though SARSA is committed to always exploring and trying to find the best policy, it still performed somewhat worse in comparison with Q-learning. However, Q learning obtains higher reward values overall (see Figure 29). In fact, the averaged reward over learning time

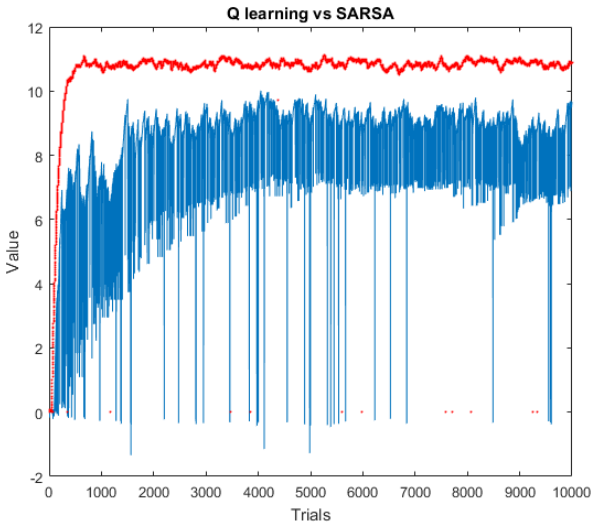

Figure 29: Q learning reward values in red Sarsa in blue.

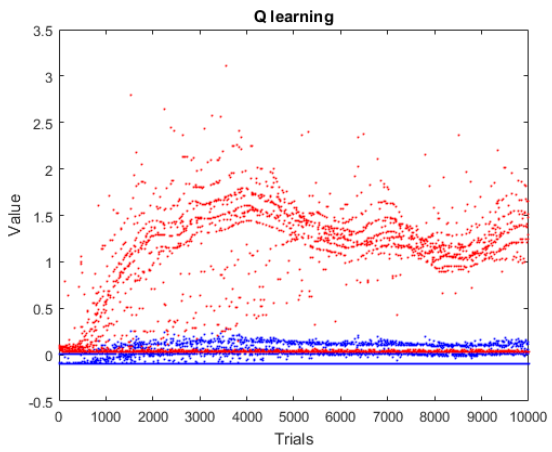

Figure 30: Q learning with $\eta = 0.1, \gamma = 0.9, \epsilon = 0.99$

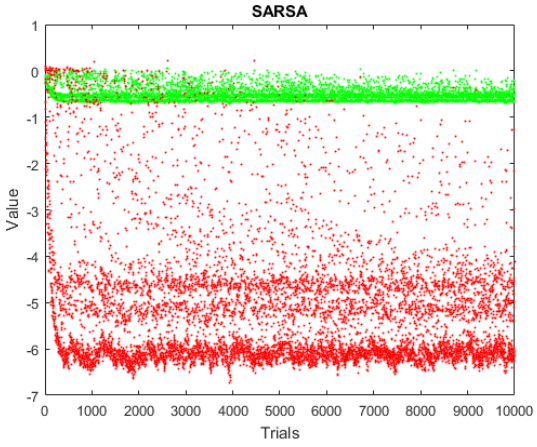

Figure 31: SARSA with $\eta = 0.1, \gamma = 0.9, \epsilon = 0.99$

for SARSA is about three times lower than that for Q learning. That is understandable since Q-learning exploits the path found over and over again propagating higher rewards while SARSA tries to explore new paths. Even though both algorithms achieve satisfying performance, one can argue that SARSA does not enable its full potential on the current problem. In the present version of the Taxi problem, SARSA does not have to deal with dead ends which are harshly penalized. Moreover, for the initially undertaken values, SARSA struggled to reach good convergence values and instead fluctuated more than Q learning. Regardless, we say that after adopting the low valued learning rate, SARSA got closer to convergence than Q-learning itself. Increasing the levels of greedification leads to completely random solution as it was the case for Q learning too (see Figures 30 and 31). However, SARSA yet again obtained lower values than Q learning as shown on those figures. SARSA would be particularly good for the problem if there were many options which were beneficial in different ways for the agent. For example if there were different routes, suggesing more secure paths and if the maze was either bigger or had a 'cliff' or something with a big negative reward introduced instead of uniformly spread penalty values. Additionally, the off-policy

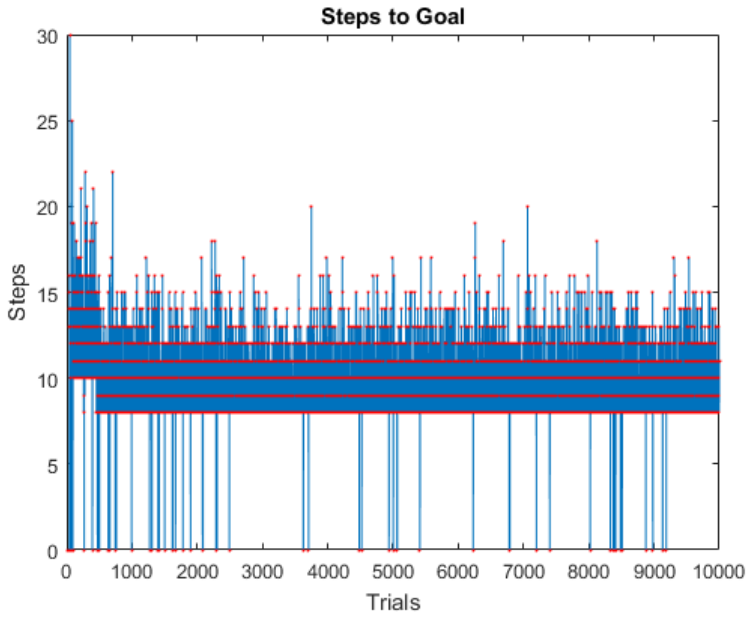

Figure 32: SARSA steps to goal.

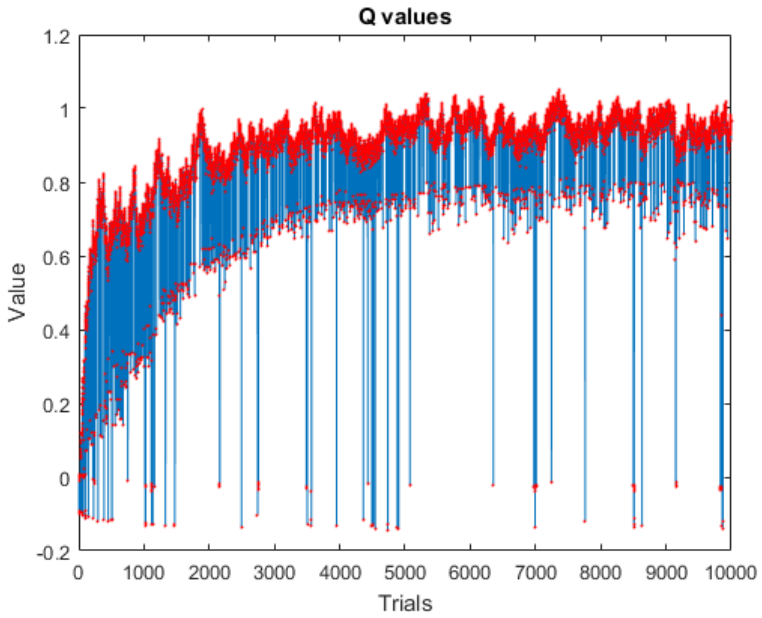

Figure 33: SARSA$_Qvalue$.

method is not that much slower than SARSA. In conclusion, SARSA is very good in exploiting a given good policy while Q-learning performs better if more exploration is necessary. That is why, SARSA receives values closer to zero.

5 Taking into account the compositional structure of the problem, discuss how the learning speed can be improved?

A particularly interesting approach to improve learning by taking into account compositional structure is through partial policy recycling as defined in [Ramon et al., 2007]. For every other policy, they use things

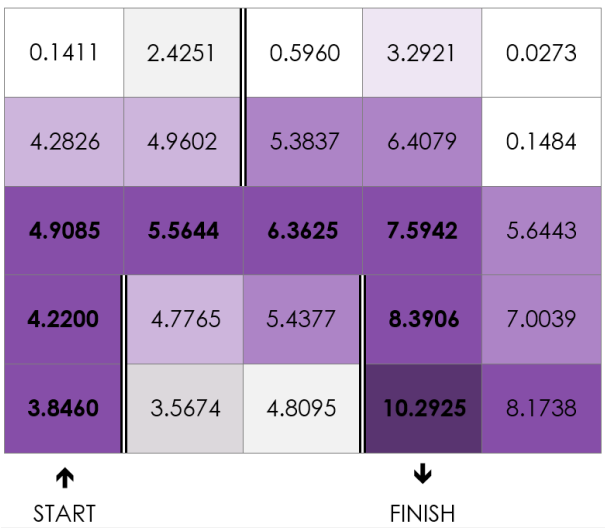

Figure 34: Spatial spread for Maximum Values for SARSA

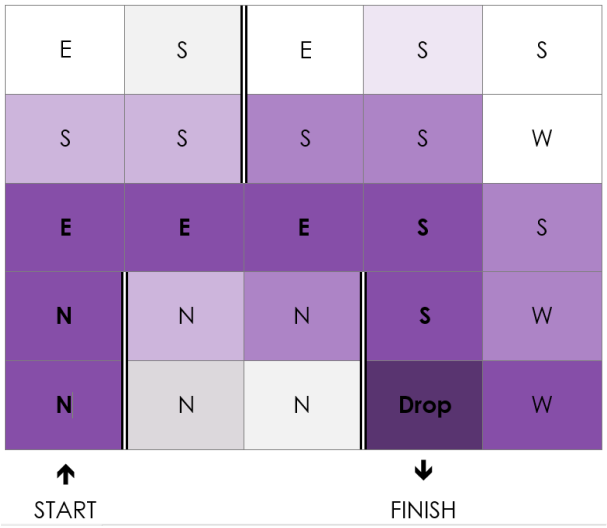

Figure 35: Spatial spread for best actions for SARSA

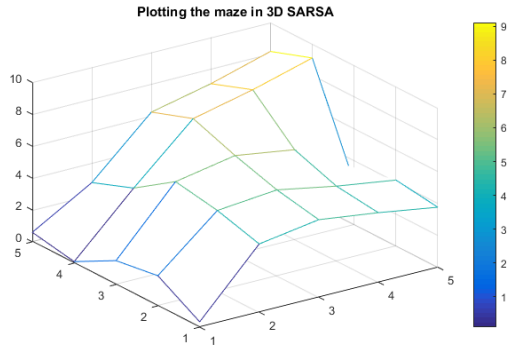

Figure 36: SARSA maze plot in 3D.

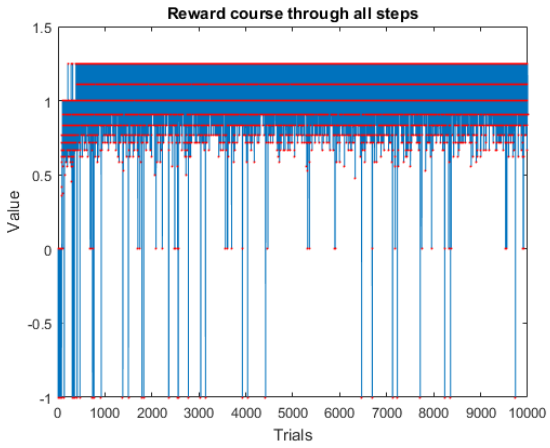

Figure 37: Reward course spread over tie

they've learnt in the previous one using a relational regression algorithm which is able to take advantage of the compositional structure of a set up. They propose an approach where we would store structures in the target function simulating an approach adopted from theory revision. To achieve this we might also want to make use of decision trees and introduce sever operations which will help us maintain a useful structure throughout learning. We would need to be able to split our leaves, prune our tree and revise an internal node, as suggested by Ramon et al.. Therefore, enabling to 'compare' our learning with the partial policy which has already learnt that a particular set of moves is good in given situations and vice versa. For example, regardless of the reason, when going towards state 'R' the paths around it can be exploited by the adoption of inductive transfers. Surely, going towards the goal will have block (2,2) for example explored and stored in the target as part of the structure. That way, when the agent gets to that place they will know that this has previously been a good move and will thus go there. In the same sense, state (2,3) would not be particularly good, given the wall standing in between that state and the goal.

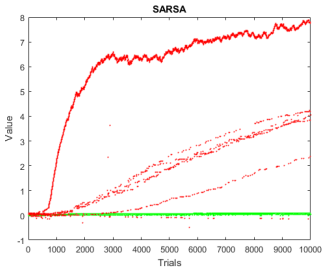

Figure 38: $\eta = 0.01, \gamma = 0.9, \epsilon = 0.1$

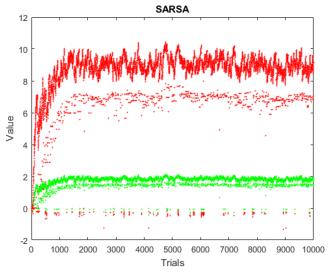

Figure 39: $\eta = 0.2, \gamma = 0.9, \epsilon = 0.1$

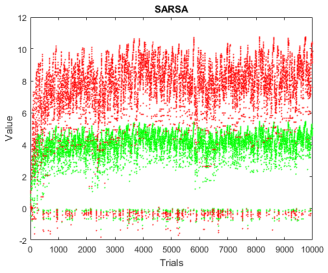

Figure 40: $\eta = 0.5, \gamma = 0.9, \epsilon = 0.1$

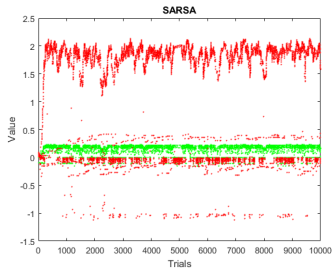

Figure 41: $\eta = 0.1, \gamma = 0.5, \epsilon = 0.1$

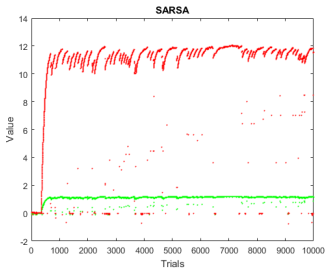

Figure 42: $\eta = 0.1, \gamma = 0.9, \epsilon = 0.01$

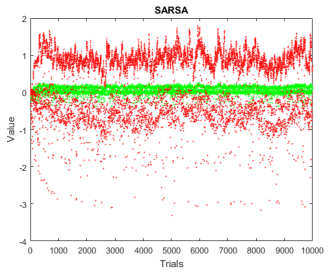

Figure 43: $\eta = 0.1, \gamma = 0.9, \epsilon = 0.5$

Although tricky, this can be adopted by making use of revision and pruning. If, for example a node turns out to be useful in an external policy it will be revised in both places aiming at the same goal the node turned useful for. Similarly, if the node ends up being a bad choice, it will be pruned from the tree stored. Although it is hard to state how much this approach will improve the learning process due to the fact that there are state and action selections involved as well as an ϵ-greedy heuristic, it has been proven that it significantly increases the learning speed. This approach can potentially improve the learning speed by changing the so far adopted ϵ-greedy search with a tree search which is roughly O(nlogn). Experiments shown in [Ramon et al., 2007] prove this to be very efficient and useful in a variety of domains. Thus, we expect this approach to reduce the learning time to 2-3 from 200-300 (i.e. log200).

6 Conclusion

The solutions of the problems presented in this assignment showed us how to tackle basic reinforcement learning problems. We introduced a variety of techniques for testing the validity and correctness of our solutions as well as observed our algorithms' behaviour with different parameter values. We inferred that depending on the task, we will need to adapt those parameters accordingly. In addition, we observed how we can use two different algorithms to tackle a problem in different ways and thus achieve optimal solutions with different settings and approaches depending on the problem.

References

Jan Ramon, Kurt Driessens, and Tom Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Machine Learning: ECML 2007*, pages 699–707. Springer, 2007.