Assignment 4: CS 7641

Gandharv Kashinath

April 19, 2015

Introduction

In this assignment Markov's Decision Processes (MDP) algorithms were explored. Performance and characteristics of the policy iteration and value iteration algorithms were explored. A reinforcement learning algorithm, namely, Q-learning was also compared and the results are presented. In order to study these algorithms two common MDPs namely; the Grid World and the Race Track problem was studied. These problems are particularly interesting due to its applications in real world scenarios which are discussed in the description. They also are two different type of problems that will help understand the nuances of these algorithms. For instance, the Grid World problem is simple in its setup and is a modification of the MDP presented as part of the class lectures. The Race Track problem is more challenging and hence will test the boundaries of these algorithms.

The paper starts by giving a detailed explanation of the two problems and continues to present the results obtained from each of these algorithms. These results are analyzed and the some meaningful conclusions are drawn to complete the paper.

Problem Descriptions

Grid World Problem

A simple grid world problem which is a modification of the MDP discussed in the lectures was used for one of the "simpler" MDPs studied for this assignment. The model consists of an m × n world that contains obstacles, terminals, a starting initial state, and where each state in the world has a reward value that can be positive or negative value. The following figure shows the grid-worlds used for this assignment along with the rules valid in this world.

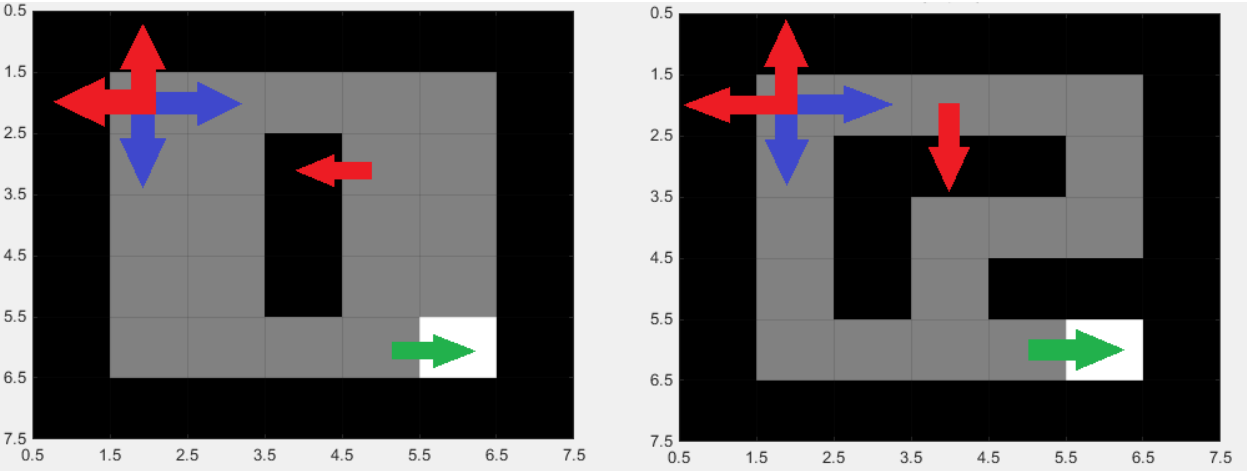

Figure 1: Grid world setup and rules. Red indicates a reward of -1, green indicates a reward of 1 and blue indicates 0 reward.

The agents can start anywhere on a grey tile, i.e. unblocked tile and tries to reach the white tile or the goal. These agents can execute 4 different actions at each time step: left, right, up and down. The agent does not get any reward when moving between unblocked tiles. It gets a reward of 1 when moving from an unblocked tile to the goal tile. When the agent tries to move outside the world or into an obstacle it stays in the same state but gets a reward of -1. So the agent will try to avoid hitting any obstacles and will try to find the shortest way to goal. To make the setting more realistic and robust to errors, the agent executes with a probability p the action it intends to execute, and will move in another direction with probability 1−p. The discount rate is set to 0.9. It is interesting to investigate which paths for the optimal policy will be computed with respect to different choices of p. The above two grid worlds were used to examine how the iterations of each algorithms change. The world on the right (intermediate) is more constrained since only 2 of the 4 actions are valid in each tile, and hence should be computed faster than the world on the left and this will be used to study how the iterations change for each of the algorithms. The world on the left (simple) is less constrained and has more than 2 actions per tile and hence should take longer than the intermediate case.

Although these grid worlds are simple and have limited states, several practical applications can be imagined with these MDPs. Taxi routes in big cities, robot movements in factories and manufacturing and several similar real world problems can be modeled using these simple grid world MDPs. This MDP also gives the flexibility to understand the effects of various parameters in studying MDP algorithms without making the world too complex.

The Race Track Problem

The Race Track problem was one of the first few examples used to study and develop dynamic programming models. A detailed description of the problem can be found in [1]. A race track is constructed using a starting line at one end and a finish line at the other consisting of designated squares. Each square within the boundary of the track is a possible location for the race car. The objective is to learn to control the car so that it crosses the finish line in as few moves as possible. Acceleration and deceleration rules and other considerations are specified in [1].

This problem is an example of an abstract stochastic shortest path problem and has applications in robot motion, navigation problems, car race analysis and others. This problem is also very interesting because it models a human driver on a circuit that is not perfect, i.e. with a probability *p* the intended action will not be executed. Because a human driver has a specific response time to recognize events and act accordingly, he may fail to act correct in a specific moment. Furthermore, even when the circuit is represented by a small and coarse grid, the state space will be so huge that some algorithms cannot be applied. Figure 2 shows the tracks used for this assignment. The gray pixels represent the road, the white pixels represent the start/finish line and black pixels represent the border. A driver always starts at the start/finish line and has to reach the finish line, which is an absorbing state. The car state is modeled by a combination of its position and its current velocity. More specifically its state is represented by a quadruple (u_x, u_y, $\bar{u}_x$, $\bar{u}_y$) where (u_x, u_y) represents its current position and ($\bar{u}_x$, $\bar{u}_y$) its velocity in x and y direction. All values are constrained to be integers.

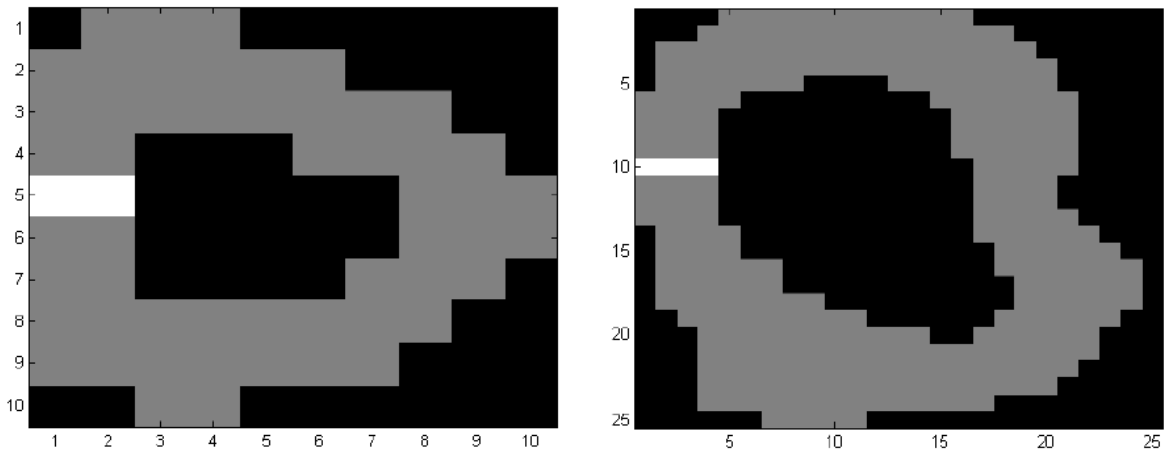

Figure 2: Race Track problem domain; left: 10 x 10, right 25 x25.

At each time step the car can accelerate, brake or keep its velocity constant. To simplify the problem the car is only allowed to change the velocity by 1 unit in each direction independently, i.e. the acceleration must be in {-1, 0, 1}. This leads to a total of 9 different possible actions. The question arises how big the state space is? For example, a car on the left track has 10 × 10 = 100 possible positions, but only 55 of these are inside the border. The velocity magnitude is limited to a maximum of $\bar{u}_{max}$= 3. That means that the highest possible discrete velocities are of the form (±3, 0) or (±2, ±2). There are four directions, so four possible velocities of the form (3, 0) and in the range of −2 to 2 are 5 integers, leading to 25 possible velocities, and therefore a total of 29 discrete velocities with a magnitude less than or equal to $\bar{u}_{max}$. The state space has 29 × 59 = 1595 different states and the transition matrix has 1595 × 1595 × 9 = 22.89 million entries, the reward matrix has 1595 × 9 = 14355 entries. Although these matrices are sparse, we can conclude that the policy iteration is hard to apply to this problem, because solving linear equations with more than 10,000 unknowns is very memory intensive and takes a lot of computational time.

To complete the track as fast as possible a solution with the fewest time steps was preferred over the distance. Hence at each time step, the driver receives a “penalty reward” of -1. If the driver hits the border or uses a shortcut crossing the border he receives a penalty reward of -100. The discount rate is set to 0.99, because the focus was on how many time steps was needed to reach the finish line. Maximizing the reward was expected to lead to the fastest and optimal track time.

Approach

In order to solve these two MDPs the MATLAB MDP Toolbox was used [2]. The value iteration and policy iteration algorithms available as part of the toolbox was used to solve the 2 MDPs. The Q-learning algorithm, which is a part of this toolkit was also used to demonstrate the capabilities of the reinforcement learning algorithms on the two MDPs described above.

Results & Discussion

Grid World Problem

The Grid World problem was solved using value iteration and policy iteration algorithms. Both the algorithms gave the same optimal policy for the simple and the intermediate grid worlds. The optimal

policy for both the grid worlds are presented in Figure 3 and from this it can be concluded that the optimal policies obtained by these algorithms are as expected.

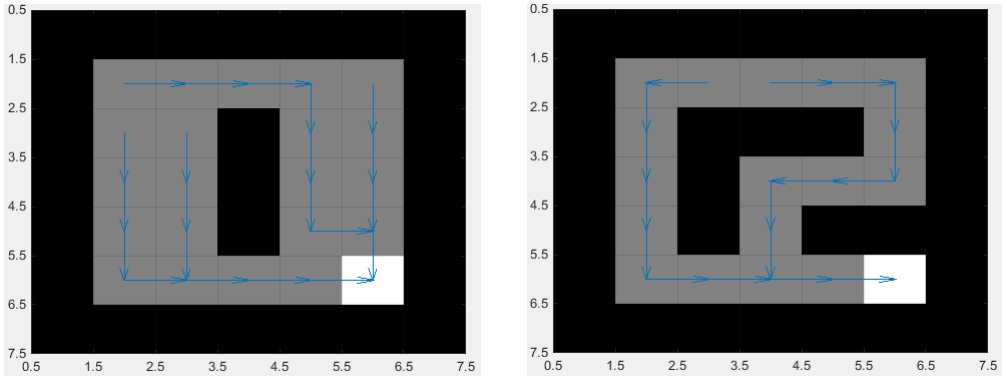

Figure 3: Optimal solutions for the two; left - easy and right - intermediate Grid Worlds.

The following table summarizes the iterations taken for the combination of algorithms and grid worlds.

Algorithm	Grid World	Iterations	Time (s)
Value	Simple	17	0.03
Value	Intermediate	21	0.007
Policy	Simple	3	0.006
Policy	Intermediate	2	0.02

Table 1: Runtime and iterations for the two Grid Worlds.

From the above table it can be concluded that the value iteration takes 5-10 times more iterations and 3-4 times more execution time. Since policy iteration solves a linear system and does not refine the policy in a greedy manner, it uses more time per iteration step, and hence there is a discrepancy between the factors of iterations and execution time.

Figure 4 shows the v-values i.e. the expected reward of each state following the optimal policy. It can be seen that the closer the state to the goal the higher the v-value. This is as expected because, faster the agent reaches the goal, and higher is its reward. It is interesting that the v-value of tiles that are adjacent to the goal is 1.09 instead of 1. This is due to the fact that the expected reward following the optimal policy, there is a 10% chance not to execute the action the agent wants and hence leading the agent away from the goal. Setting the error probability to 0 leads to a v-value of 1, as expected.

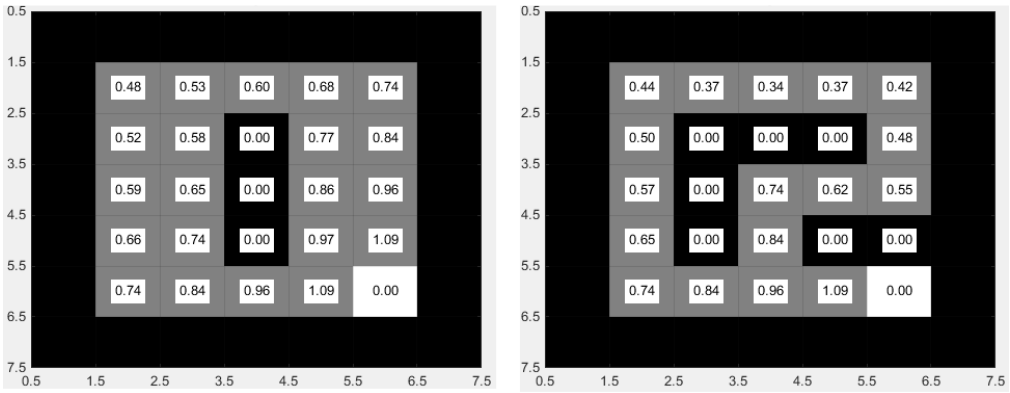

Figure 4: V-values for the two Grid Worlds.

In order to understand the effect of varying the probability *p*, i.e. the probability to execute the intended action, two cases where *p* was set to 0.6 and 1.0 was run and the solution is presented in Figure 5.

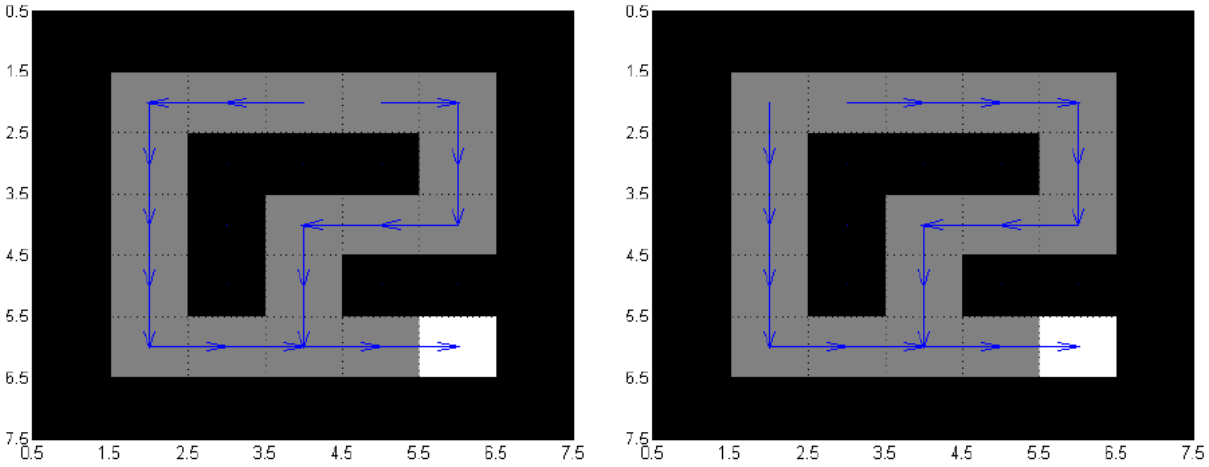

Figure 5: Solution for p = 1.0 (left) and p = 0.6 (right).

From the above figure, for *p* = 1 the solution shifts at the top one tile to right, but is still optimal, the distance of the top tile to the goal is same whether the agent goes first right or left. For *p* = 0.6, which means the probability to execute the intended actions is just slightly above average, the solution shifts one tile to the left compared to *p* = 0.9. This case does not provide the optimal solution for the case of starting from the second tile on the top row. It would be optimum to go left instead of right and hence by reducing *p* = 0.6 the optimal route was missed. This is also as expected since the probability of picking the optimal path is slightly more than chance. Because both algorithms select the starting position to calculate the optimal policy randomly, each algorithm was run several times. However the problem was so simple that the results never changed. Also, the iterations have been found be constant for the problem.

The Race Track Problem

The Race Track problem shows that policy iteration is not always the preferred algorithm. If the state space is large and the number of actions are small compared to that, the transition matrix will be very sparse. Solving the linear system for the Q or respectively V values can take significantly long. Also, as noted above the matrix may become very big and iterative solvers need to be used. Table 2 gives the result for both algorithms for the 10 × 10 case.

Algorithm	Iterations	Time (s)
Value	27	0.156001000000288
Policy	8	0.421202700000322

Table 2: Runtime and iterations for Race Track problem for 10 x 10 grid.

The value iteration algorithm is around 2-3 times faster than policy iteration, although it needs more iterations. Figure 6 shows the results from the 10 x 10 case for both the value and policy algorithms and the optimal routes match in both cases. The directional vectors (green) and the acceleration vectors (red) are exactly the same for both algorithms. The algorithm of choice for this case would be the value algorithm due to the shorter run time.

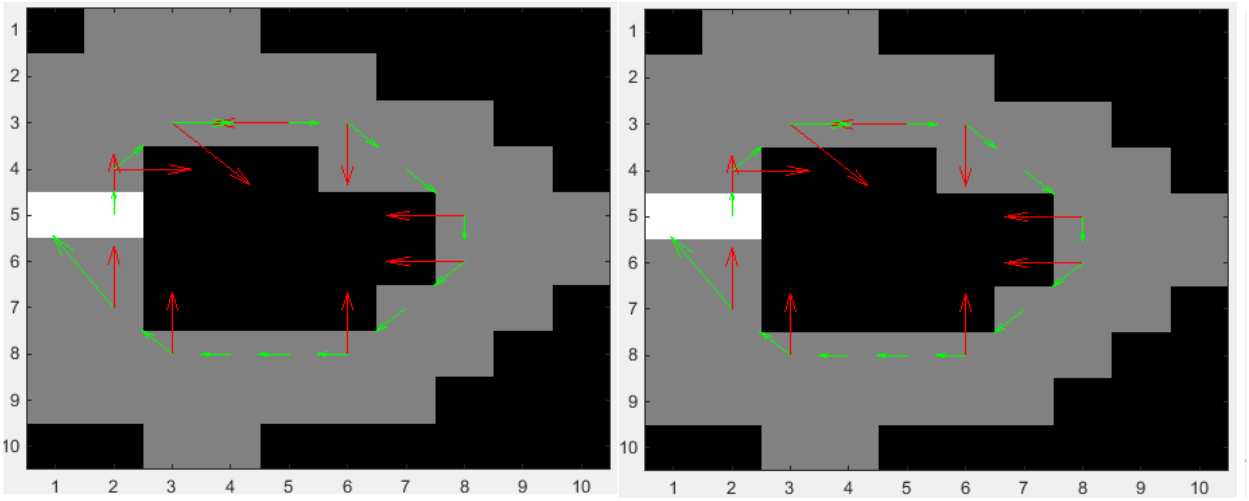

Figure 6: Optimal routes for value (left) and policy (right) algorithms for the Race Track problem (10 x 10).

Table 3 gives the result for the value and policy algorithms for the 25 × 25 case.

Algorithm	Iterations	Time (s)
Value	36	1.107607099999768
Policy	11	1.014006499999596

Table 3: Runtime and iterations for Race Track problem for 25 x 25 grid.

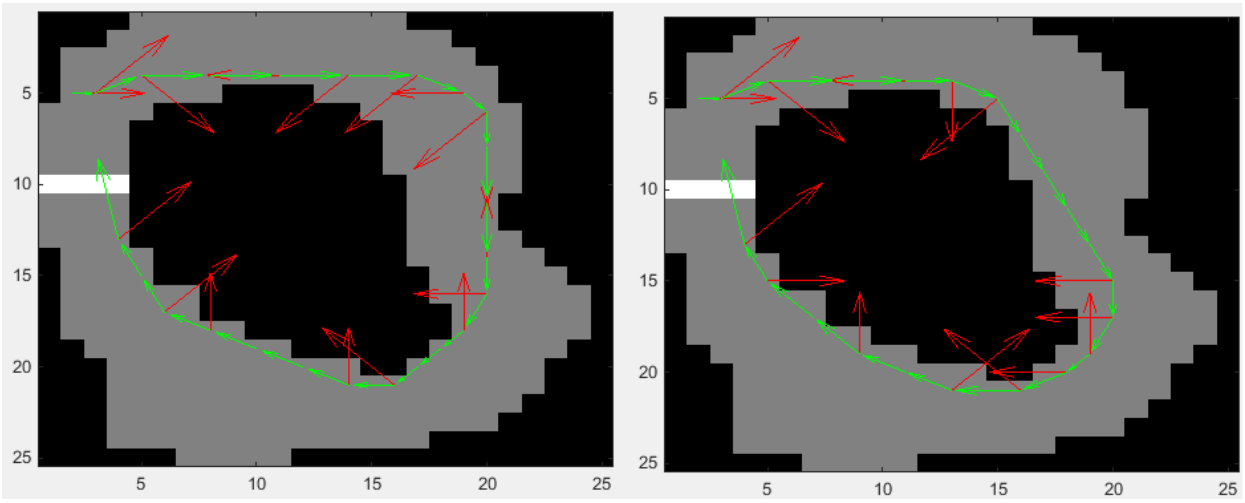

Figure 7: Optimal routes for value (left) and policy (right) algorithms for the Race Track problem (25 x 25).

Figure 7 shows the optimal route calculated by both the algorithms for the 25 × 25 case and from this figure it can be concluded that the optimal route calculated by the policy algorithm is better than the value algorithm. The optimal route predicted by the policy algorithm has the shorter distance and this algorithm also took less time compared to the value algorithm. Due to these reasons the algorithm of choice for this setup is the policy algorithm.

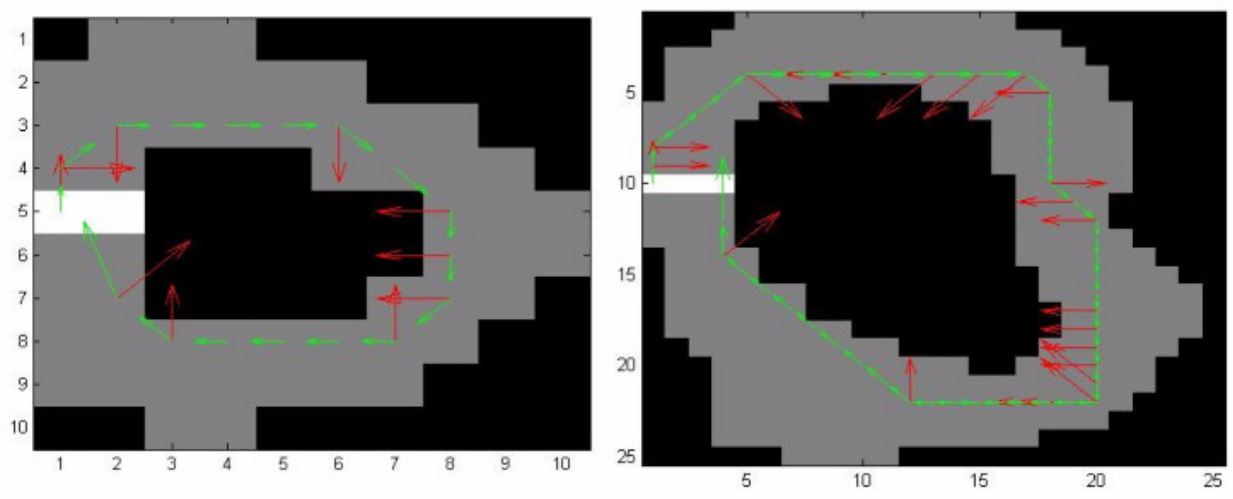

Figure 8: Optimal routes for 10 x 10 (left) and 25 x 25 (right) track problem with *p* = 0.6.

In order to study the effect of varying the error probability both the cases were run with $p = 0.6$, slightly better than chance. The policy iterations were robust to changing p but the value iterations took too long to converge and hence couldn't be studied. From Figure 8, it can be concluded that the optimum route has increased in distance from Figure 6 and 7. More studies needs to be conducted to pick between the two algorithms when the error probability is lower than 0.9.

Reinforcement Learning for MDPs

Although the above two algorithms perform well, in more realistic settings these algorithms would prove to be computationally expensive and reinforcement learning algorithms like, Q-learning may become more relevant. In this section the Q-learning algorithm will be used to study the two problems part of this assignment and the results will be evaluated and compared against the value and policy algorithms.

Q-learning Algorithm

The Q-learning algorithm comes under the category of active reinforcement learning and hence decides what action to take in each step of the MDP. The difference between this algorithm and value iteration is that this algorithm compares the expected utility of the available actions and hence does not require a model of the environment. The Q-learning algorithm available as part of the MDP Toolbox was used to run the two problems and the results are presented below.

Grid World Problem

The Grid World problem was run using the Q-learning algorithm. Figure 9 shows the results of the optimal route and the V-values for these runs.

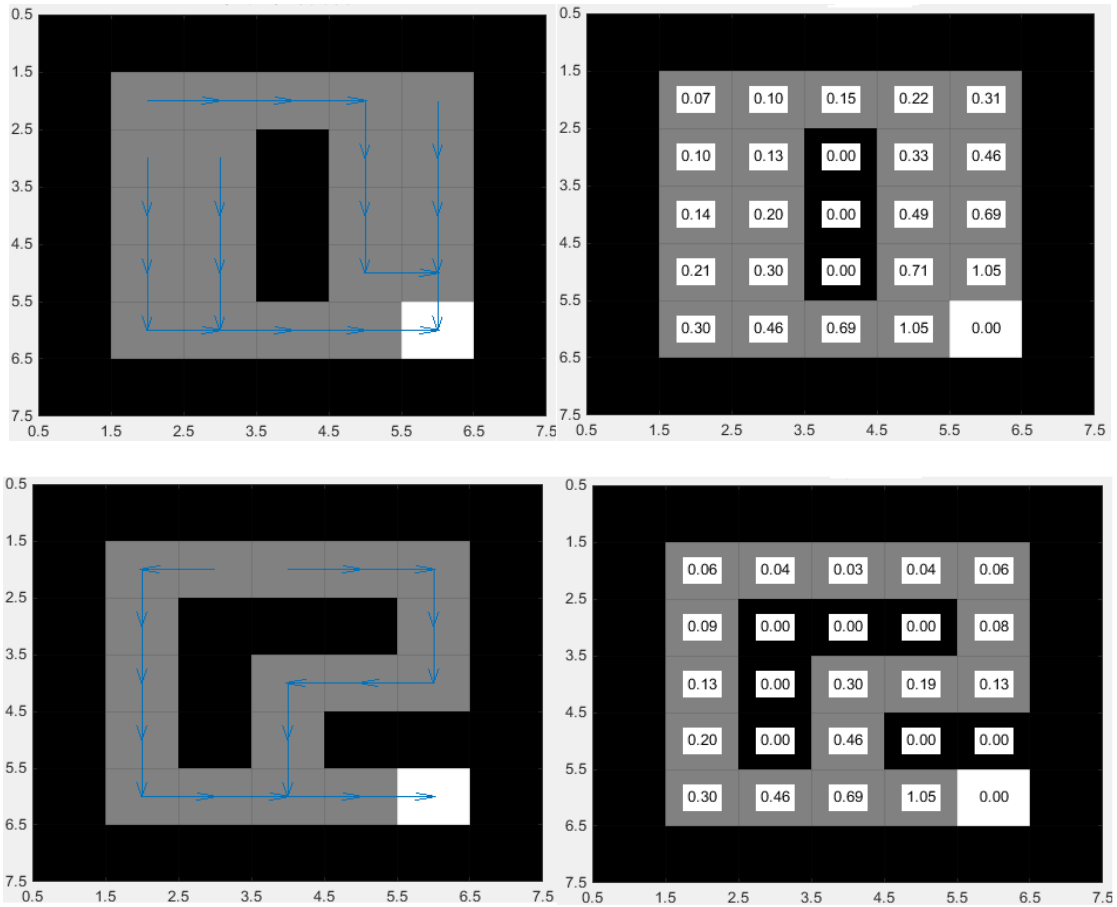

Figure 9: optimal solutions (left) and V-Values (right) for the simple (top) and intermediate (bottom) Grid Worlds.

From the above plots, it can be concluded that the optimal routes match the previous results obtained from the value and policy algorithms. The Q-learning algorithm execution time was more than the other two algorithms and ran in 0.1 s for the simple and 0.08s for the intermediate case. This result was as somewhat expected since this is an iterative algorithm and the Grid World problem is too small to see the benefits of the iterative solver. For larger more realistic problems this algorithm will do better than the value and policy algorithms.

Figure 10, shows the mean discrepancy as a function of iterations and converges to the right solution very quickly.

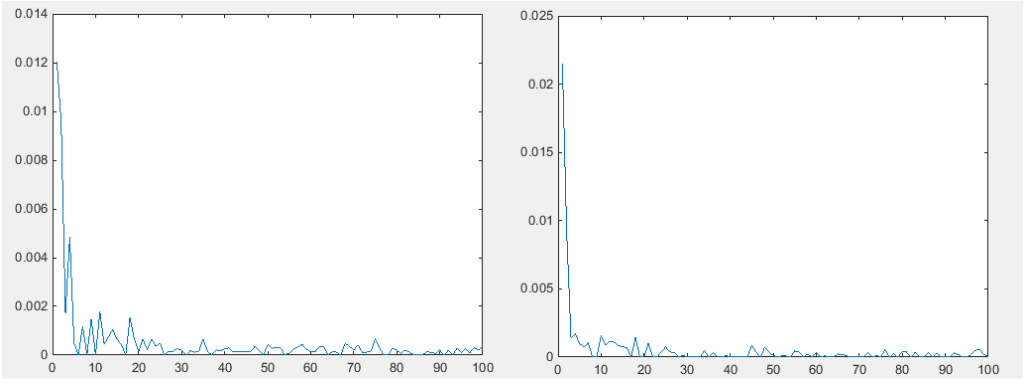

Figure 10: Mean discrepancies plot with iterations for Q-learning; simple (left) and intermediate (right) Grid Worlds.

The Q-learning algorithm available in MDP Toolbox uses a linear classifier in order to approximate the Q-function and since this problem is relatively simple this works well. The performance of this algorithm is worth evaluating for more complex problems where a linear classifier might limit the quality of the solution. This is explored in the next problem.

The Race Track Problem

The Race Track problem was run for the 10 × 10 and 25 × 25 case and the results were analyzed. Even though these problems are complex the execution time was significantly higher than the value and policy iteration algorithms. This can be attributed to the classifier used in the Q-learning algorithm of the MDP Toolkit. Due to its linear nature it might have taken more computational effort to converge to a solution. The inherent non-linear nature of the problem was limiting in gaining significant performance benefits over the value and policy iteration algorithms. As opposed to value iteration, the Q-learning algorithm learns an action-utility representation instead of learning utilities. This involves more computations as opposed to a simple linear algebra solution generated by the value iteration algorithm.

The following plot, figure 11 shows the mean-discrepancy obtained by the Q-learning algorithm for the 10 × 10 and 25 × 25 Race Track problems and from this plot it can be concluded that the 25 × 25 case took more time to converge to an optimal solution. This was expected due to the complexity of the domain.

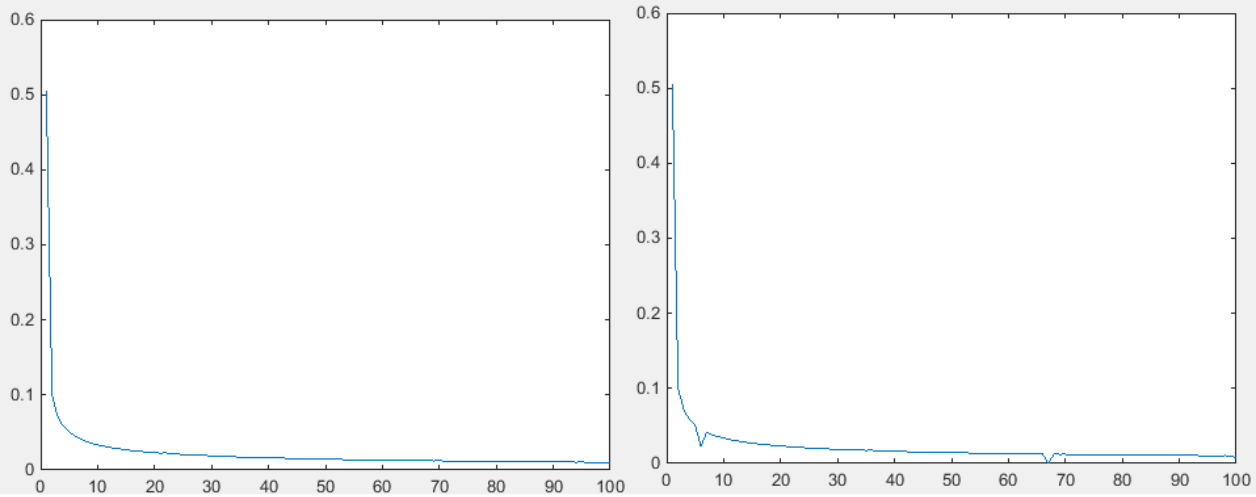

Figure 11: Mean discrepancies plot with iterations for Q-learning; 10 x 10 (left) and 25 x 25 (right) Race Track problem.

Figure 12, shows the optimal path for the 10 × 10 case. 25 × 25 couldn't be obtained due to the very long runtimes for the Q-learning algorithm. The optimal route obtained by the Q-learning is very similar to the policy iteration and value iteration results. Although the optimal results were obtained, the simulation took unreasonably long to run and would not be the algorithm of choice for this problem.

The Q-learning algorithm for 10 × 10 case took 2hrs to run. Other Q-learning algorithms which use different function approximator is available in literature and should be explored to find the best algorithms for Q-learning [3, 4].

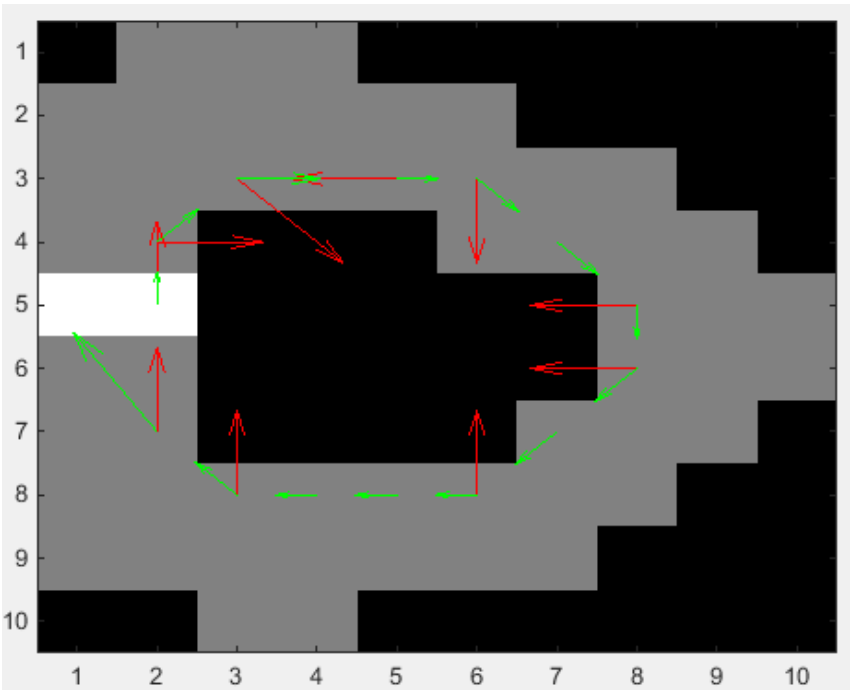

Figure 12: Optimal solution 10 x 10 Race Track problem for Q-learning.

Conclusion

In this assignment three MDP solution algorithms were studied on two MDPs and the results were presented and analyzed. The best algorithm for the two problems was the policy iteration algorithm and this was due to the low execution time and the optimal solutions obtained in all cases. The value iteration algorithm worked well for both the problems but had issues with finding the optimal solution and took more time than the policy iteration algorithm. The Q-learning algorithm resulted in optimal solutions all the time but took unreasonable time to execute. This could be attributed to the function approximator used in the MDP Toolbox which was a linear approximator and took a long time to converge to a solution. Other function approximators could be used to improve performance and detailed analysis with literature review is necessary to understand this algorithm. Perhaps a neural-network can be used as a function approximator? This is a question worth exploring in the future.

Reference

[1] A. G. Barto et. al. *Learning to Act using Real-Time Dynamic Programming, 1993.*
[2] MDP Toolbox in MATLAB
[3] T.M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
[4] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 3rd Ed.* Prentice Hall, 2009.