CS-7641 Machine Learning: Assignment 4

Markov Decision Processes
Chenzi Wang
cwang493@gatech.edu
November 29, 2015

1 Introduction

Markov Decision Processes (MDPs) consist of dealing with problems where outputs are in part random and in part related to some form of rule. In this way, they are stochastic control processes. Stochastic meaning the output is directly controlled by some random variable. Two MDPs are investigated in the present assignment as well as the functionality of reinforcement learning algorithm in solving them. They are solved using value iteration, policy iteration, and a reinforcement learning algorithm (Gauss-Seidel).

2 Two Interesting Markov Decision Processes (MDPs)

2.1 Grid-World

The grid-world problem consists of a grid of locations. One of these locations contains a prize or goal and some locations consist of obstacles. The problem is performed by a searcher starting at a random location on this grid, traveling to adjacent grid locations without being able to see what is in them beforehand, the obstacles blocking a route are located always in the same locations, and the searcher attempts to reach the goal location in the quickest route possible, consequently earning optimal points. Such a problem mirrors the task of a robot attempting trying to remove a bomb amongst rubble. Solving such a problem could also be used by companies such as Uber to route cars on optimal routes where they are most likely to find customers (prize) while also avoiding traffic jams (obstacles). The searcher can move up, down, left, or right. If the searcher chooses to go into an adjacent square and it is empty, it is awarded no points. If it moves into a cell with an obstacle or a cell outside the limits of its grid world, it is penalized a point and remains in its original cell. If it moves into the cell that is the goal, it is awarded one point. The fastest route is desirable since the larger the number of moves, the larger the chance of hitting an obstacle and being penalized points. From the grid model used, the optimal choice of move is determined starting from any location given discovery of empty cells and cells with obstacles. There is a set probability (p) the searcher will move according to these optimal choice in moves but there is also a probability (1-p) that the searcher will choose to take a move in a random direction. For this assignment, the discount rate is set to 0.9. The two different grid worlds used in this assignment can be found in Figure 1. This problem has a relatively lower number of possible states. Figure 1a should be an easier problem to solve because there is a smaller probability of choosing the wrong direction given a random starting location. On the other hand, for almost all cells, there is only one direction that can be taken at any location that would bring the searcher closer to the goal and not move into an obstacle.

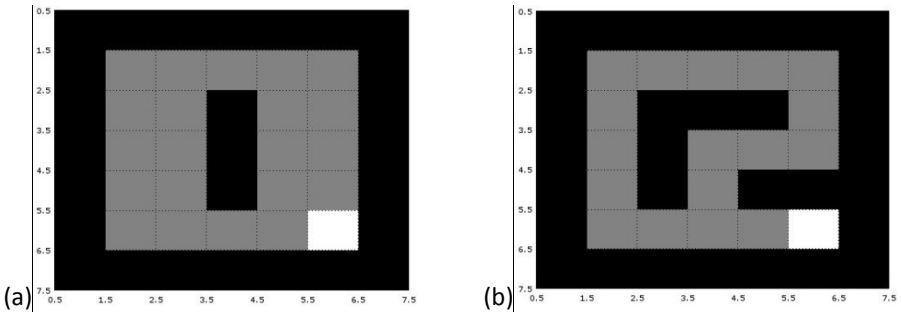
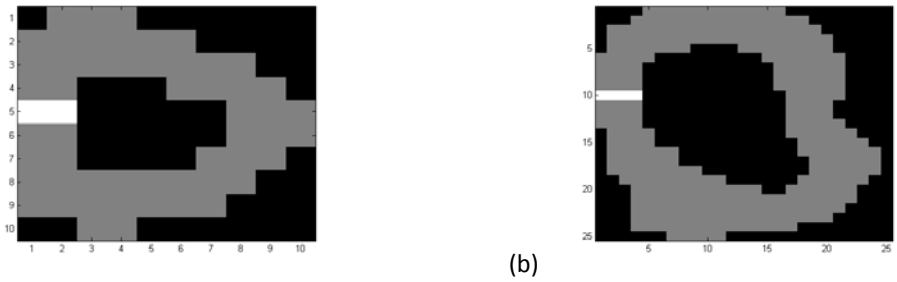


Figure 1: Grid worlds used for this assignment: (a) easier and (b) harder problem to solve due to the number of wrong choices possible at any location. The white square is the location of the goal/prize.

2.2 Racetrack Problem

This problem concerns the control of a race car as it travels around a user defines racetrack such that it can traverse the racetrack in the shortest time possible. The relevance of this problem with real life situations is

straightforward in that the goal of finishing the course of a racetrack in the fastest time is the goal in any race and can be useful for a driver determining his or her best decisions on how to drive the course or programming a computer controlled driver to do so. At any point on the track, the race car is at velocity and has the ability to accelerate up to a maximum velocity, decelerate and the problem being solved is the optimal sequence of accelerating and braking to get around the course. A car must accelerate in one dimension and decelerate in the other to perform a turn. The car is required to stay on the track. Each step in time, the car is penalized one point, and the car is penalized 100 points each time it hits or traverses the border of the racecourse. Because there is a probability (p) in this problem that the car may make non-optimal decision on whether to accelerate or apply brakes, this mirror real life by accounting for human driver error (a driver may also have a limiting response time). At any point of time on the 2-dimensional track, the race car has two position variables [x(t), y(t)], such that location on the track can be known, and two velocity variables (Vx(t), Vy(t)], one velocity in each dimension. To limit the possible states of these variables, they are constrained to integers in a pre-defined range. At each point in time, the race car can keep the same speed, accelerate, or decelerate in either dimension. The control variable are thus change in velocity in each dimension [ax(t), ay(t)]. The capability of a racecar to accelerate and decelerate is limited by its components; as such, in the problem used presently, the racecar is restricted to being able to accelerate or decelerate by one integer of velocity in each direction per integer of distance[ax(t), ay(t) = {-1, 0, 1}]. Because the two components of velocity has three different choices at each instant of time, they can be in nine different combinations of actions of acceleration, deceleration, and maintaining speed. The maximum velocity that the race car in this problem can reach is set to 3 total in any direction (positive 3 in the positive x and y direction and negative three in the negative x and negative y direction). This gives maximum speed in one direction when [Vx(t), Vy(t)] = [±3, 0] or [0, ±3] or when they are combined direction to [±2, ±2]. This leads to 29 different possible velocity states at any location. The racecourses used for this assignment can be found in Figure 2. The 10 X 10 racecourse in Figure 2a has 55 cells within the track. With the 29 different possible velocity states, this gives 29*55 = 1595 different variable states. This gives 1595*9 = 14355 in a reward matrix and 1595*1595*9 = 2.29E7 in a transition matrix. The values in a The 25 X 25 racecourse in Figure 2b has 319 cells within the track, giving 29*319 = 9251 different variable states. This gives 9251*9 = 83259 values in its reward matrix and 7.70E7 values in its transition matrix. A probability of success in executing the optimal action is set to p = 0.9 and the discount rate for this problem is set to 0.99.[1]



(a) (b)

Figure 2: Racecourses used for the racetrack problem in the present assignment. The area of white cells is the start/finish line, the grey is the racecourse, and black cells are off track. The course on the left (a) is made up of 10 X 10 cells with 55 cells in the track and the course on the right (b) is 25 X 25 cells with 319 cells in the track.

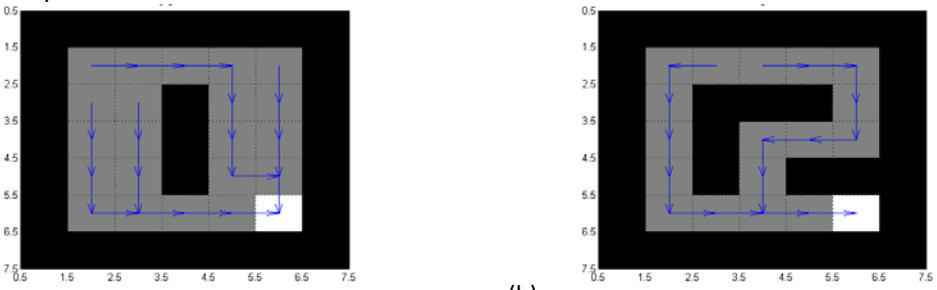
3 Results

3.1 Grid-World Results

For both grid-world examples, the value, policy, and Gauss-Seidel iteration algorithms find the same policy for optimal movements to get to the goal location (see Figure 3). The optimal policies found do visually appear to be the best policies to get to the goal from any start location. How the algorithms performed in solving these problems can be understood from the parameters found in Table 1. The value algorithm must perform many more iterations (around 7-11 times more) on both grid world problems to determine the optimal policy in comparison with the policy algorithm and takes more computational time (around twice as much). The reason why

[1] http://cs.jhu.edu/~vmohan3/document/ai_rl.pdf

the computational time of the value algorithm is not substantially longer than the policy algorithm when it has so many more iterations is that iterations in the policy algorithm take up more computional time because it solves a linear system. The grid world thought to be the more difficult problem did take more iterations for each algorithm but, unexpectedly, these additional iterations did not cause a significant increase in computational time. In fact, it took more computational time for the value algorithm to solve the easier grid world problem than the more difficult one. The Gauss-Seidel is designed to word as a variation of the value algorithm but is supposed to converge faster to the optimal policy by updating calculations performed for visited stated during each iteration.[2] Gauss-Seidel is seen here to effectively reduce the number of iterations needed from those needed for the value algorithm, particularly for the more difficult grid-world. The computation time is about the same for the easy grid but increases for the difficult grid. This increase in computation time with decrease in iterations are strong evidence that iterations of Gauss-Seidel are more computationally demanding, likely due to the value updates performed in each cycle.



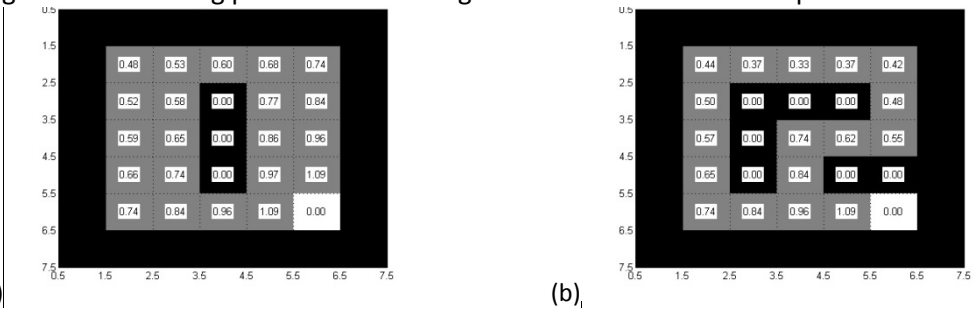
(a) (b)

Figure 3: Optimal routes to the goal from any random starting cell in both the grid worlds evaluated in the present assignment seen in Figure 1.

Table 1: Factors showing the performance of the algorithms on solving the grid world problems (p = 0.9).

Algorithm	Problem	Comp. Time (s)	Iterations
Value	Easy Grid	0.05	18
Value	Difficult Grid	0.02	23
Policy	Easy Grid	0.01	2
Policy	Difficult Grid	0.03	3
Gauss-Seidel	Easy Grid	0.05	13
Gauss-Seidel	Difficult Grid	0.05	14

V-values are the expected reward in each cell if the optimal route is taken. The V-values for the two grid world problems can be found in Figure 4. The closer to the goal, the higher the V-values. This corresponds with the fact that the closer to the goal, the higher the chance for the highest score by getting the goal without being penalized for hitting an obstacle or the outside perimeter.



(a) (b)

Figure 4: V values for the (a) easy and (b) more difficult grid world problems.

[2] http://castlelab.princeton.edu/ORF569papers/Powell_ADP_2ndEdition_Chapter%203.pdf

It was also chosen to investigate the effect of the probability chosen on the optimal path found by the algorithms. The probability, p, being referred to is the probability that the optimal directional choice will be made at each location. The optimal routes with probability choices p = 0.4, p = 0.6, and p = 1.0 can be seen in Figure 5 for the harder grid world problem and Figure 6 for the easier grid world. The solution for optimal policy for p = 1.0 is shifted over one tile to the right in comparison to its solution with p = 0.9. This is because, with a p = 1.0, the probability of taking a wrong turn is not an issue (the path on the right takes more turns to reach the goal). With p = 0.6 in the harder grid world, the policy is shifted to make the route with less turns shorter, again to avoid additional penalties. The optimal policy for the easier grid world is the same at p = 0.6, 0.9, and 1.0. As probability of taking a non-optimal direction increases, it is clear that the algorithms produce policies that try to compensate for wrong moves to best avoid penalty points.

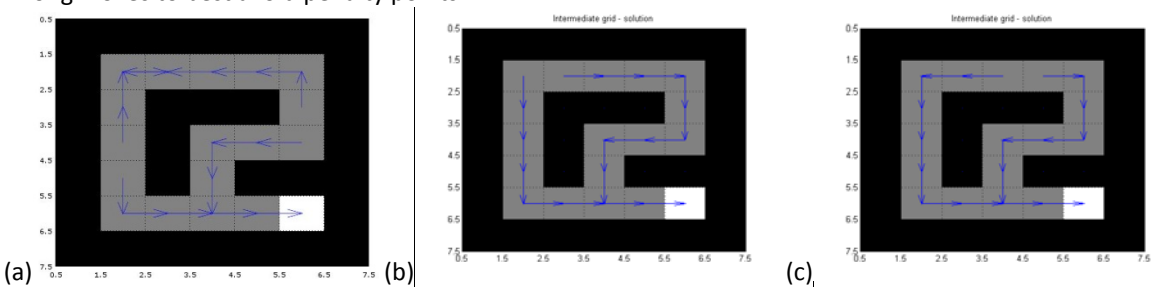


Figure 5: Optimal policies determined for the harder grid world problem with probability of choosing a non-optimal direction set to (a) p = 0.4, (b) p = 0.6, and (c) p = 1.0.

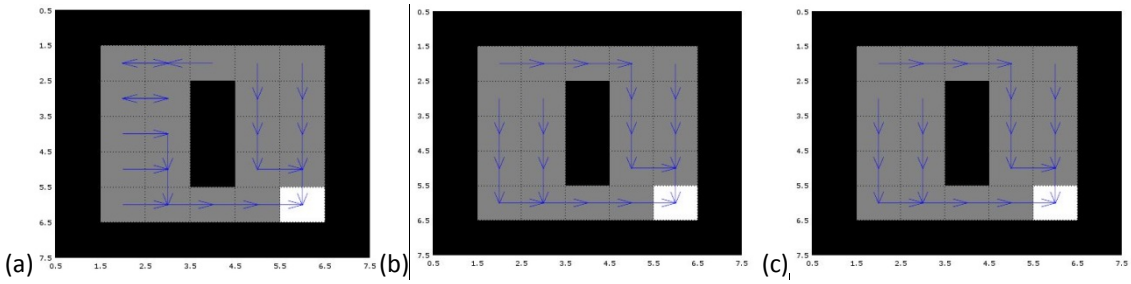


Figure 6: Optimal policies determined for the easier grid world problem with probability of choosing a non-optimal direction set to (a) p = 0.4, (b) p = 0.6, and (c) p = 1.0.

The easy grid world example was expanded to 10 X 10 and 20 X 20 versions to investigate the effect of increasing the number of possible states. These can be found in Figure 7. The performance parameters of these larger grid worlds are graphically compared with each other and with the original grid world in Figure 8. Both computation time and iteration size increase with increasing grid size. There appears to be a linear correlation between grid size and iterations but this is not the case with computational time. Computational time appears to increase more with increasing grid sizes. This likely relates to each iteration becoming more complex and time consuming. The higher number of possible states has the effect of increasing the computation time for value algorithm results more so than in the policy algorithm. It can thus be said that, for this problem, the policy algorithm outperforms the value algorithm for increased number of states.

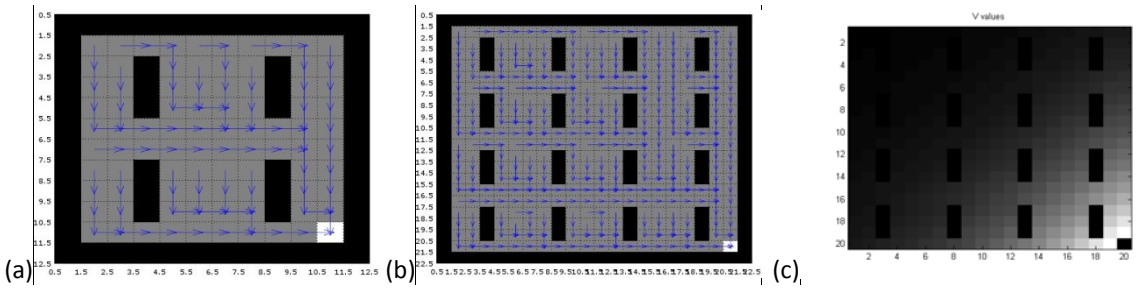
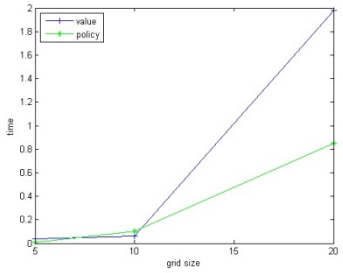
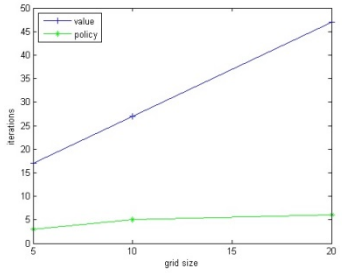


Figure 7: (a) Optimal policy found for a 10 X 10 version of the easy grid world problem, (b) a 20 X 20 version, and (c) its V values where lighter cells have higher values than darker cells.



(a)



(b)

Figure 8: Trends of (a) computation time and (b) iterations to find the optimal policy with increasing grid size. Results for the value algorithm are blue and the policy algorithm are green.

3.2 Racetrack Problem Results

Results for optimal policy with a probability (p) of choosing an optimal action set to 0.9 can be seen for the 10X10 and 25X25 racetrack problems in Table 2 and Table 3 respectively, including results using the value, policy, and a Gauss-Seidel algorithm. Graphical representations of the sample runs can be found for the 10X10 racetrack problems using the value and policy iteration algorithms in Figure 9 and Figure 10 respectively and those for the 25X25 racetrack problem using the value and policy iteration algorithms in Figure 11 and Figure 12 respectively. Unsurprisingly, more iterations and substantially more computational time is needed to determine the optimal policy for the 25X25 racetrack problem due to the substantially larger number of possible states. For each, the value algorithms need a good deal more iterations to produce an answer than the policy algorithm. However, the computational times to run each algorithm are quite close. This is similar behavior to what was seen in the grid world problems and can be explained by each iteration of the policy algorithm being more computationally demanding because it solves a linear system. The optimal policies found using Gauss-Seidel are comparable to the other algorithms. It is seen that it does effectively reduce the number of iterations from the number needed for the value algorithm, but vastly more computation time is needed for each iteration (likely due to updating calculated values each cycle) that it is the poorest performing algorithm.

The effect of changing the probability (p) of choosing an optimal action was investigated. Additional values chosen were p = 0.6 and p = 1.0. Graphical representations of sample runs at p = 0.6 for the 10X10 racetrack using the value and policy algorithms can be found in Figure 13 and Figure 15 respectively and those for the 25X25 racetrack using the value and policy algorithms in Figure 14 and Figure 16 respectively. Policies found with p = 1.0 were the same using the value and policy algorithms in both problems; their graphical representations can be found in Figure 17. This can be explained because the policies found using p = 1.0 are the true ideal policies for the racetracks since no non-optimal actions are permitted. Graphical representations of the track time (Figure 18), iteration count (Figure 19), and computational time (Figure 20) are provided to have a better understanding of any differences in performance between the value and policy algorithms as well as the effect of changing p. The policies are found to be consistently more optimal with increasing probability p of choosing an optimal action. At each value for p, the value and policy algorithms produce analogous results for both problems. The iteration count for the value algorithm is seen to drop with increasing values of p whereas the iteration count for the policy algorithm is seen to increase with increasing values of p. This shows that decreasing the probability that non-optimal actions will be chosen makes it easier for the value algorithm to attain an optimal policy whereas the policy algorithm must try somewhat harder to find the optimal policy when action choosing capability is closer to ideal. This may be due to the method which the algorithms choose to stop iterating an accept a policy is optimal when the decision making capability is closer to ideal. Yet, as before, the iterations needed to produce an optimal policy are greater for the value algorithm than for the policy algorithm for all values of p. However, also as seen before, the increase in iterations in for the value algorithm do not correlate to an equally scaled up value in computation time. The computation time cannot be said to be affected by change in p and it can be seen that the

its values for both algorithm are quite close due to the increased time for each iteration in the policy algorithm (the difference in computation time is exaggerated in Figure 20 due to the scale used). Though there is little difference between the two, it can be said that the value algorithm takes consistently more computational time than the policy algorithm.

Table 2: Factors showing the performance of the algorithms on solving the 10X10 racetrack problem (p = 0.9).

Algorithm	Track Time	Comp. Time (s)	Iterations	Run #
Value	16	1.388	27	1
Value	14	1.404	27	2
Value	15	1.466	27	3
Value	15	1.342	27	4
Value	14	1.42	27	5
Value	16	1.591	27	6
Policy	14	1.404	8	1
Policy	14	1.373	8	2
Policy	15	1.404	8	3
Policy	15	1.42	8	4
Policy	15	1.388	8	5
Policy	14	1.45	8	6
Gauss-Seidel	15	23.463	23	1
Gauss-Seidel	14	23.961	23	2
Gauss-Seidel	16	23.416	23	3

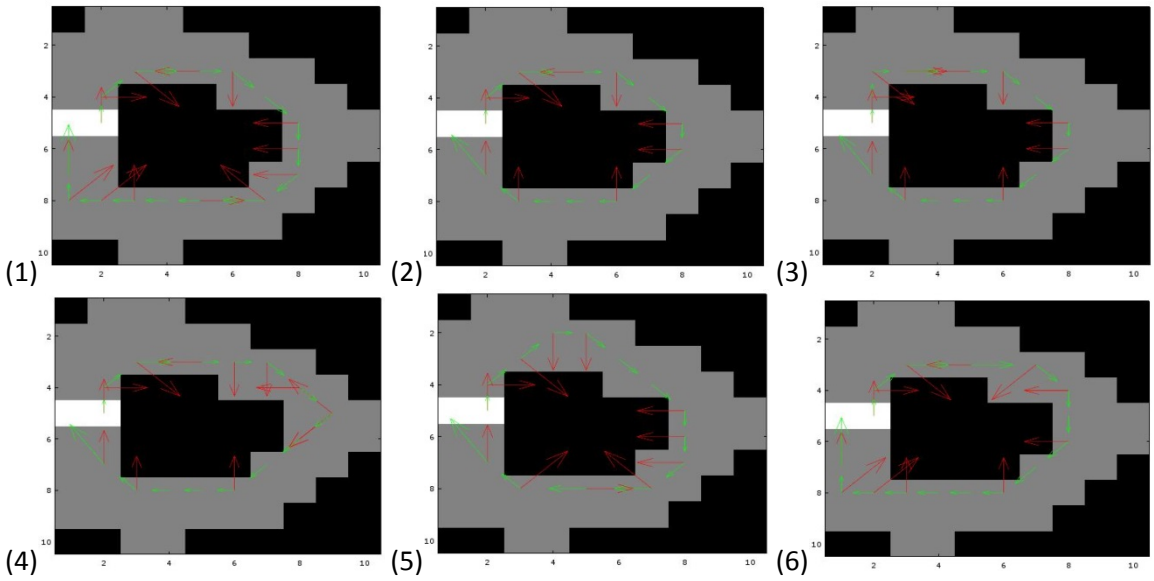


Figure 9: Sample runs of policies determined for 10X10 racetrack problem using value iteration algorithm (p = 0.9)

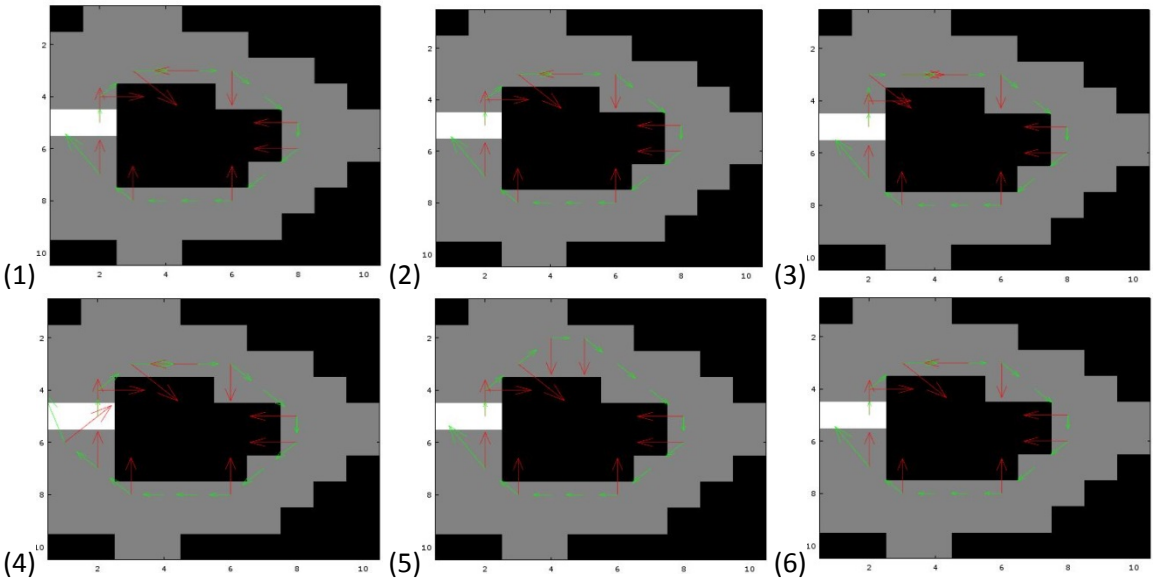


Figure 10: Sample runs of policies determined for 10X10 racetrack problem using policy iteration algorithm (p = 0.9)

Table 3: Factors showing the performance of the algorithms on solving the 25X25 racetrack problem (p = 0.9).

Algorithm	Track Time	Comp. Time (s)	Iterations	Run #
Value	23	7.83	36	1
Value	24	7.972	36	2
Value	24	7.753	36	3
Value	23	7.956	36	4
Value	23	7.987	36	5
Value	25	7.738	36	6
Policy	23	7.94	13	1
Policy	23	7.816	13	2
Policy	22	7.753	13	3
Policy	22	7.956	13	4
Policy	23	7.784	13	5
Policy	23	7.753	13	6
Gauss-Seidel	23	690.82	33	1
Gauss-Seidel	23	683.706	33	2
Gauss-Seidel	23	697.012	33	3

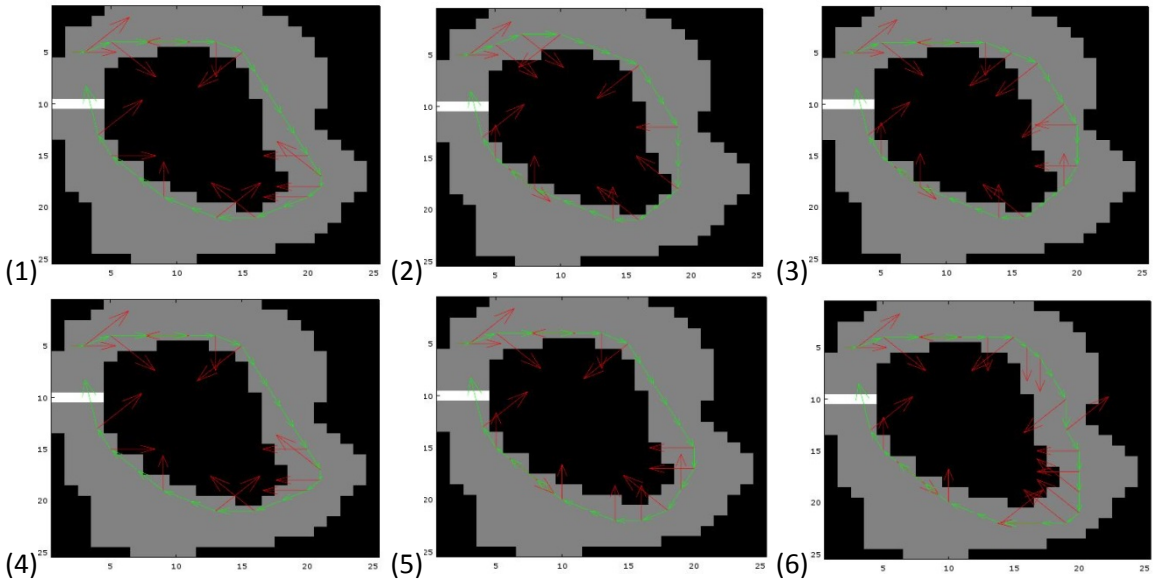


Figure 11: Sample runs of policies determined for 25X25 racetrack problem using value iteration algorithm (p = 0.9)

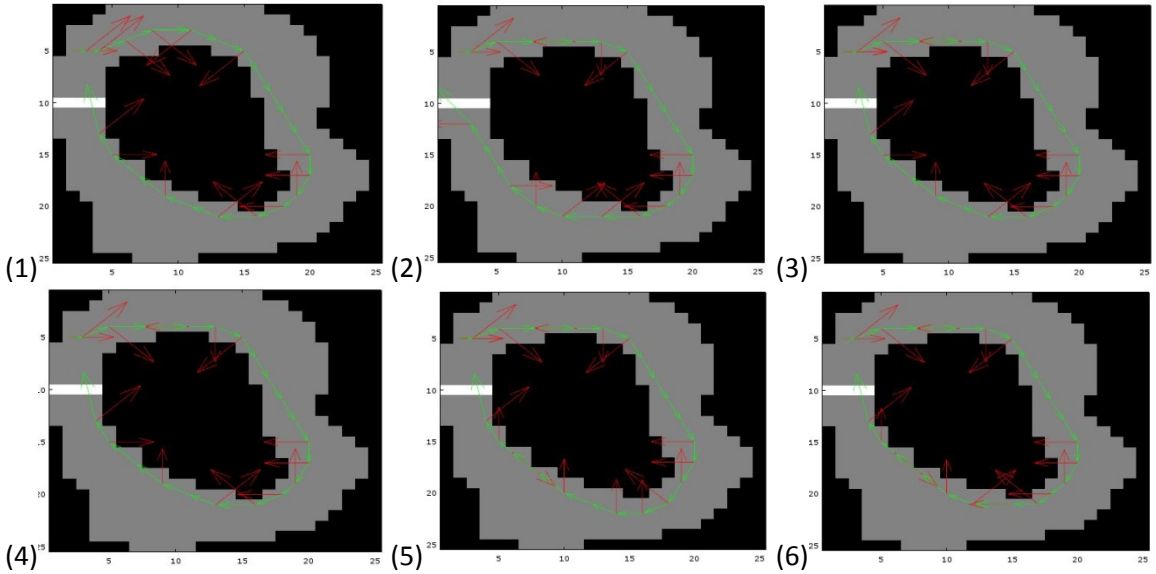


Figure 12: Sample runs of policies determined for 25X25 racetrack problem using policy iteration algorithm (p = 0.9)

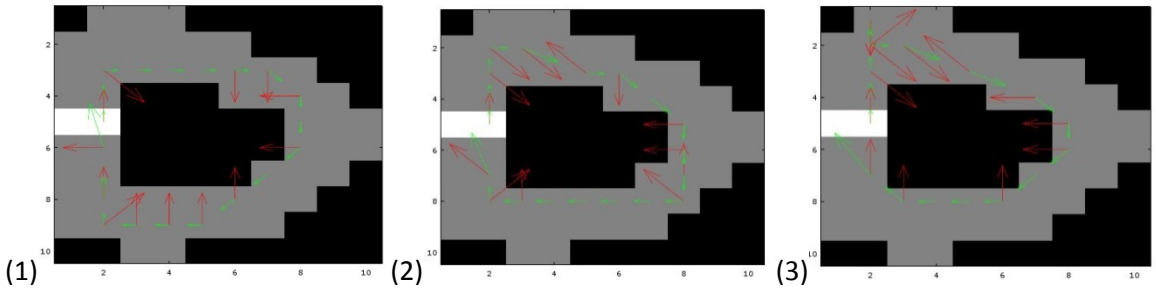


Figure 13: Sample runs of policies determined for 10X10 racetrack problem using value iteration algorithm (p = 0.6)

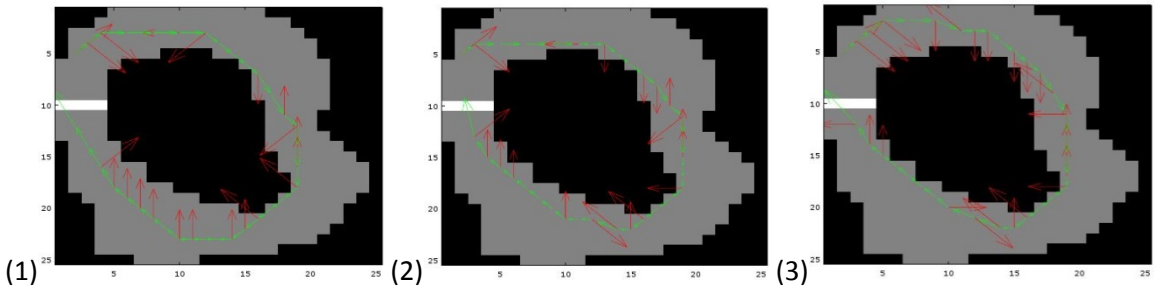


Figure 14: Sample runs of policies determined for 25X25 racetrack problem using value iteration algorithm (p = 0.6)

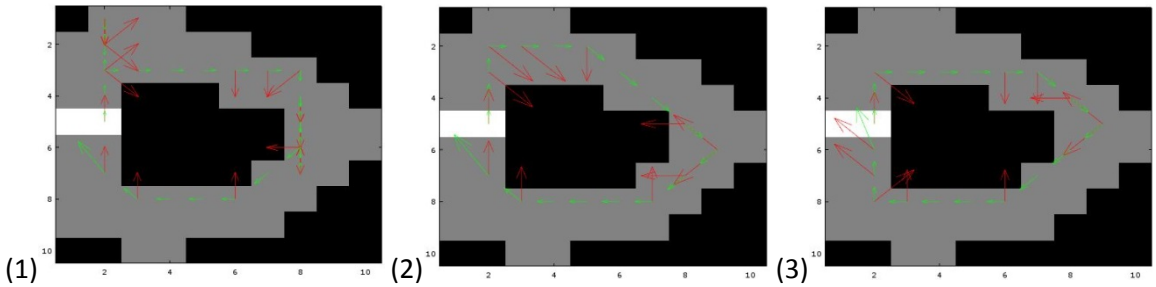


Figure 15: Sample runs of policies determined for 10X10 racetrack problem using policy iteration algorithm (p = 0.6)

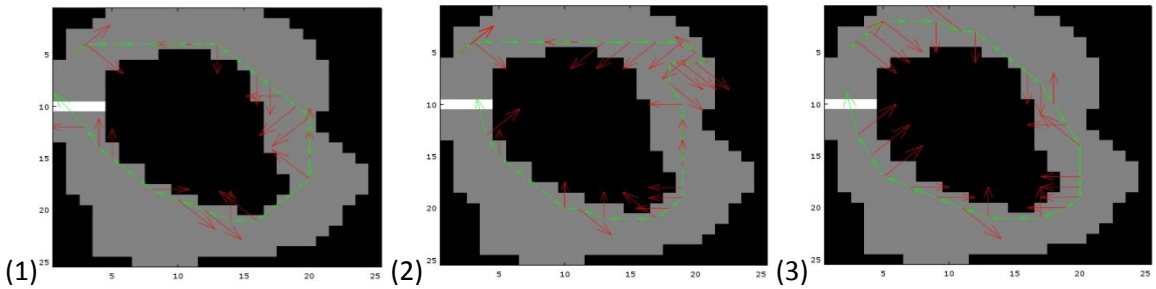


Figure 16: Sample runs of policies determined for 25X25 racetrack problem using policy iteration algorithm (p = 0.6)

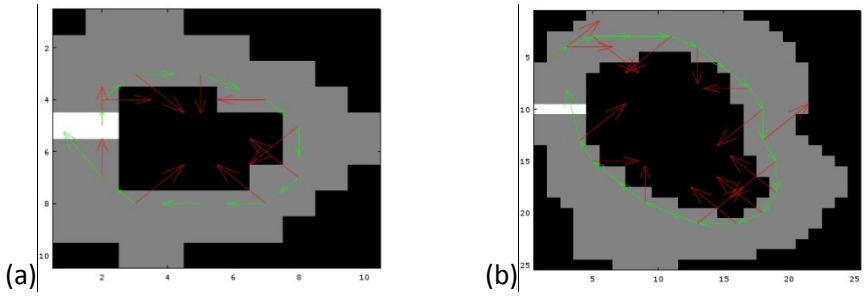


Figure 17: Policies determined for the (a)10X10 and (b) 25X25 racetrack problems (p = 1.0). Results were the same using value and policy algorithms for all sample runs.

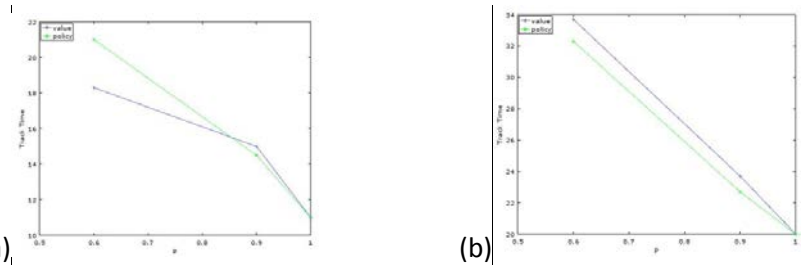


Figure 18: Trends of track time per change in probability of choosing a non-optimal action, p, for the (a) 10X10 and (b) 25X25 racetrack problems. The lower the time steps, the more optimal the policy. Results for the value algorithm are blue and the policy algorithm are green.

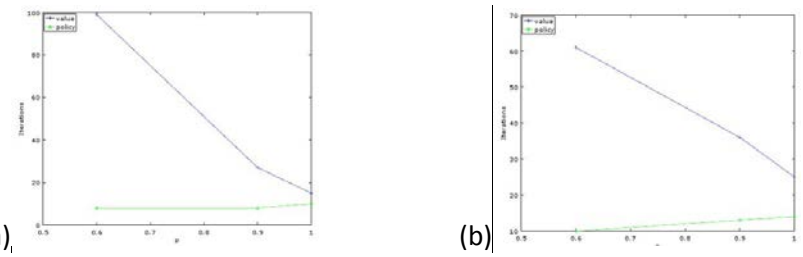


Figure 19: Trends of iteration count per change in probability of choosing a non-optimal action, p, for the (a) 10X10 and (b) 25X25 racetrack problems. Results for the value algorithm are blue and the policy algorithm are green.

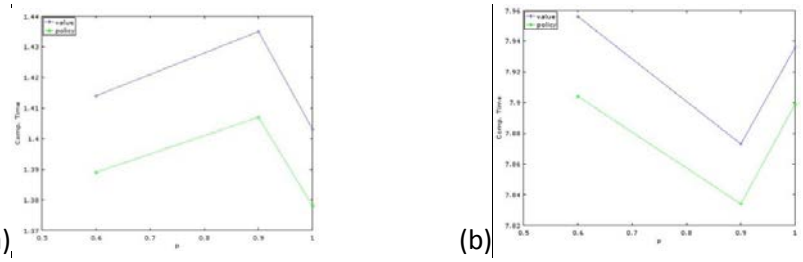


Figure 20: Trends of computational time per change in probability of choosing a non-optimal action, p, for the (a) 10X10 and (b) 25X25 racetrack problems. Note that the variation of time with p is exaggerated due to the scale. Results for the value algorithm are blue and the policy algorithm are green.

4 Conclusions

For both the grid world problems and the racetrack problems, it was found that the value and policy algorithms produced comparable results for optimal policies. For problems with smaller number of possible states, such as in the grid world problems, it was found that computational time for both the value and policy algorithms were comparable but with somewhat increased number of states the computational time for the value algorithm escalates such that the policy algorithm outperforms it. However, with highly increased number of states such as in the racetrack problem, it was seen that the computation time seems to approach comparable values between the two algorithms, though the value algorithm was found to produce results with slightly higher computation times. Consequently, it can be said that the policy algorithm outperforms the value algorithm for lower state problems but the two algorithms are comparable for high state problems. It was found for all problems that the value algorithm consistently needed more iterations to find an optimal policy than the policy algorithm but that this increase in iterations outweighed any increase in computational time (not scaled equivocally). This shows that each iteration in the policy algorithm is substantially more computationally demanding than an iteration in the value algorithm. This is likely because each iteration of the policy algorithm works to solve a linear system. The Gauss-Seidel algorithm produces comparable optimal policies and does effectively reduce the number of iterations below that of the value algorithm but not so much as to be below the policy algorithm. Each iteration is also more computationally demanding such that its computation is the longest of the algorithms, increasing with more states, making it the poorest performer.