Author: Apurva Sharma CS7641
GTID: 902490301

Assignment 4 : Markov Decision Processes

Introduction:

For this assignment, I choose two Markov Decision Processes – A simple maze navigation problem and a car race example (using Matlab's INRA MDP Toolbox). The first problem is relatively simple with number of states being determined by the size of the maze and the obstacles. The second problem is a little more complex with a large number of states (as many as 4149 when a 25X25 race track is chosen).

MDP Description:

Maze Navigation:

This is a very basic problem, yet with a lot of applications. The setup is simple: given a maze with walls and possible routes, the task is to navigate from a given origin to a destination.
Though simple, this path finding problem extends to a lot of practical problems: right from a robot trying to navigate through a hurdle course to intelligent packet forwarding in networks.
A typical maze looks like :

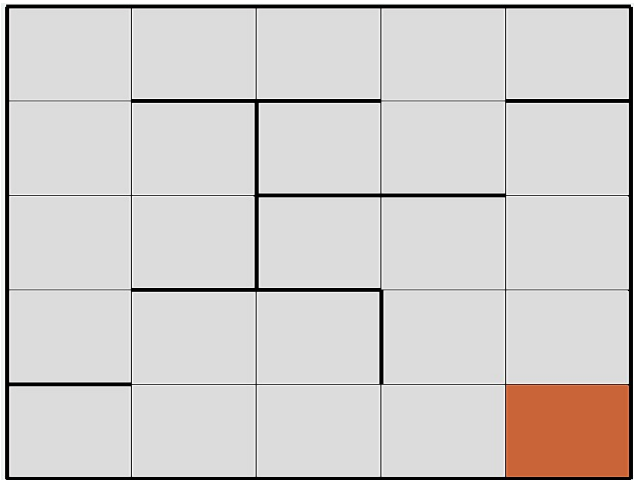


Fig 1(a). Simple 5x5 Maze

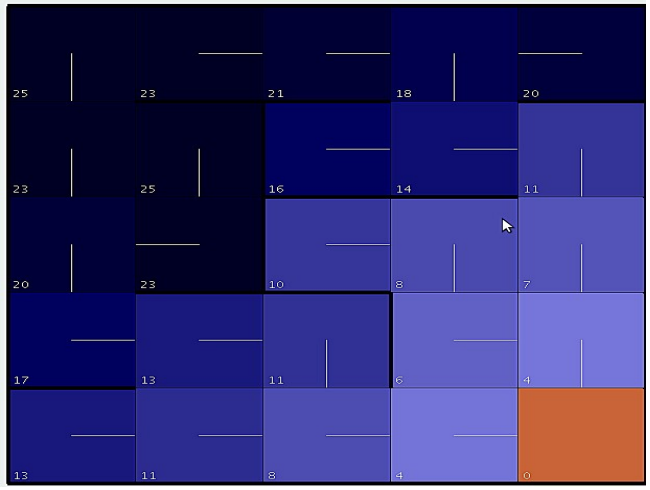


Fig 1(b) Sample policy for the maze in 1(a)

A brief problem description is as: Given a maze with hurdles (walls in dark black) and a destination (the cell in brown), find a policy to navigate to the destination cell from a start cell. The only possible movements from a cell are Up, Down, Left and Right.
In order to better simulate real world agents, we also model the noise in the environment, I.e, we relax the assumption that an agent always does what it is told to do. We assign a probability p with which an agent may make an error and land up in any of the neighboring squares to which transition is allowed. We also put a high penalty on bumping into a wall. This is justified even in the practical world where we try to avoid hitting the obstacles. Another justifiable assumption is that at a given instant, an agent may make only one move. The result of bumping into a wall is the penalty as well as the agent remaining in the same cell.
The objective is to come up with a policy which will always tell an agent which move to make, given a particular state. This should be done in a manner so that there is minimum wall bumping.
The numbers in the cells in Fig 1(b) indicate the values returned by the policy which signify the expected infinite discount reward as a result of executing the policy.

Car Race Problem:

This problem is a little more challenging. Given the map of a race track with a start and an end, the task is to control a car so that it can reach the end in the quickest time possible. There are two factors which influence the time taken – the path chosen (x & y

coords) as well as the velocity control in both x & y directions.

The car has an option of both accelerating as well as decelerating as well as holding constant speed. A human driver would typically accelerate in a simple stretch while decelerate in presence of hurdles and on turns in order to avoid bumping into them. We expect our agent to do the same.

As with the previous problem, here too we try to model some of the real life conditions. Namely, we do not want the car to hit the obstacles. For this purpose, we induce a penalty for every time the car hits the wall. Also, we relax the assumption of a perfect driver- the one who always does the right thing by the notion of probabilities. We define the probability of non transmission as the probability that the driver wont do what it's told. Since the idea is to shorten the trip duration, we award -1 for every time unit that passes by, thus motivating the agent to do take the faster approach (thus inducing the finite horizon criteria). Also since hitting the obstacles may be potentially dangerous, I assign a serious penalty of -100 every time an obstacle is hit.
We use discounted rewards as when we are close to the end, rewards in distant future seem not to provide any incentive.
I used race tracks of dimensions 10X10 and 25X25 (Greater sized tracks like 50X50 were constantly throwing “Not enough Memory” error). A pictorial view of these tracks is visualized in much detail while analyzing the results.

Value Iteration:

Experiment Setup:

For Value Iteration, I plan to make vary the size of the mazes and make the following observations:

1. The number of iterations taken for convergence.
2. The execution time.

Besides this, I also plan to compare these with Policy Iteration.

Observations:

I ran value Iteration for mazes with the following configurations:

1. 3X3 Simple – 1 Goal State
2. 10X6 – Multiple Goal States
3. 10X10 – Multiple Goal States
4. 45X45 – 1 Goal State

I got the following policies:

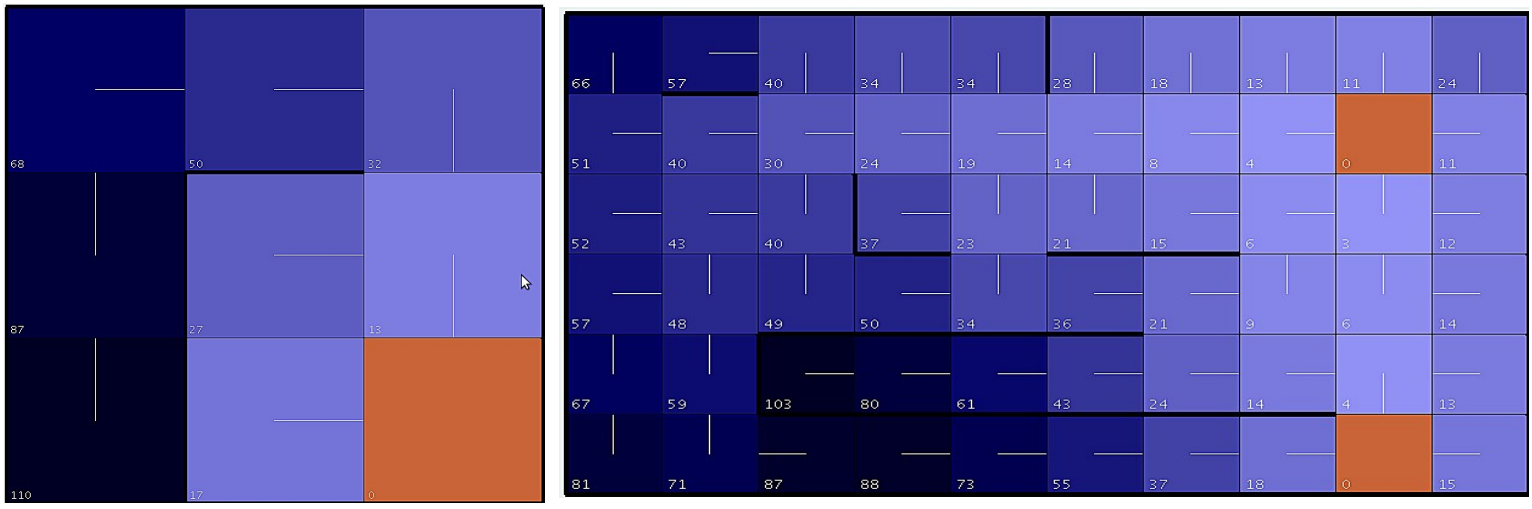


Fig 2 a-b Policies for configs 1& 2 from value iteration

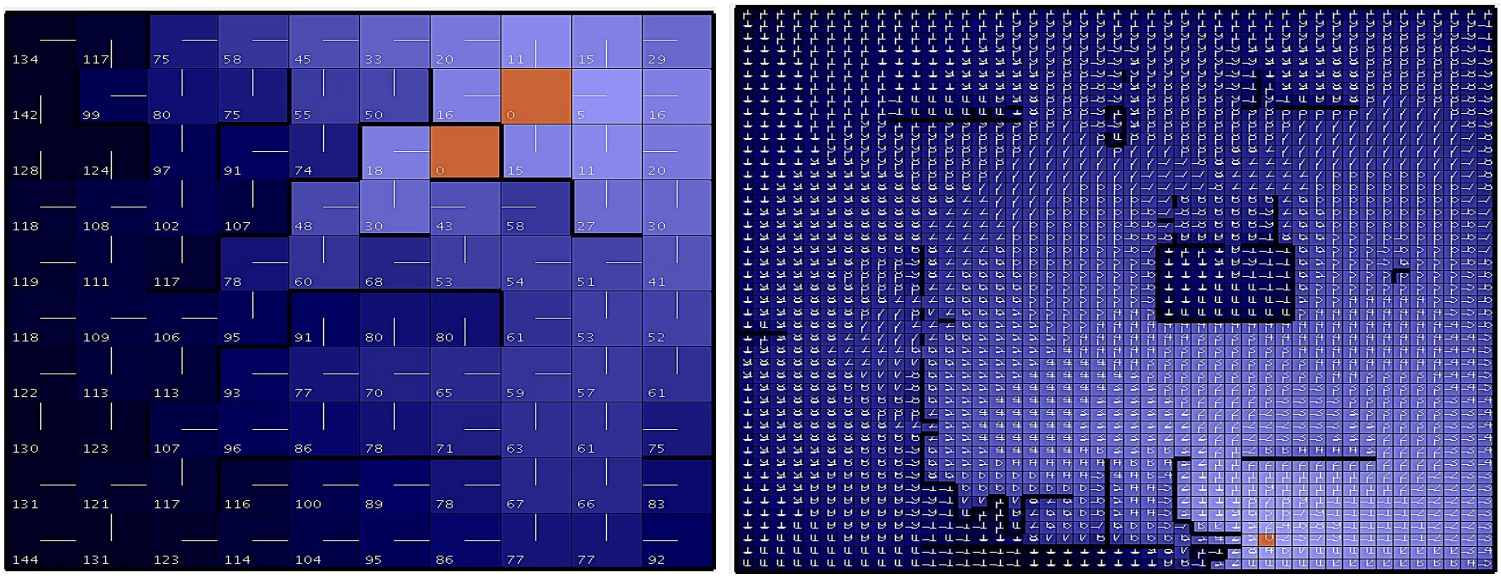


Fig 2 a-b Policies for configs 3&4 from value iteration

One observation made was that number of iterations taken for convergence for value iteration seemed pretty high. But conclusions about this can be made only after observations from Policy Iteration.
In the most complex scenario, the algorithm took 3692 ms and 181 iterations.

Another interesting experiment can be varying the penalty for bumping into a wall. For easy observation, I assign no penalty to hitting a wall for the simplest scenario (3X3 maze). The result is expected and interesting. I get the following policy:

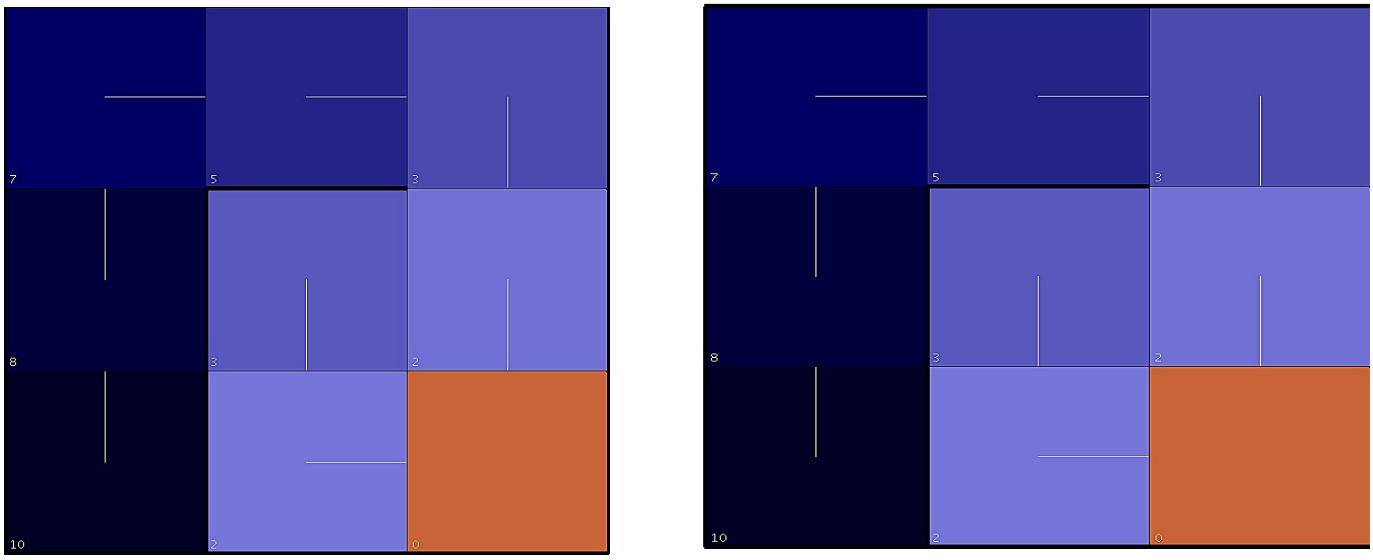


Fig 3 Behavior changes as penalties change for both algorithm

As expected, the agent moves around much more freely now, in particular, if we look at cell (2,2), it now moves down as it does not have to worry about hitting the adjacent wall. This reduces the number of iterations to 24.

Policy iteration gives the same result with just 2 iterations!

For the race car scenario, I take tracks of 2 variations : a 10X10 track and a 25X25 track.

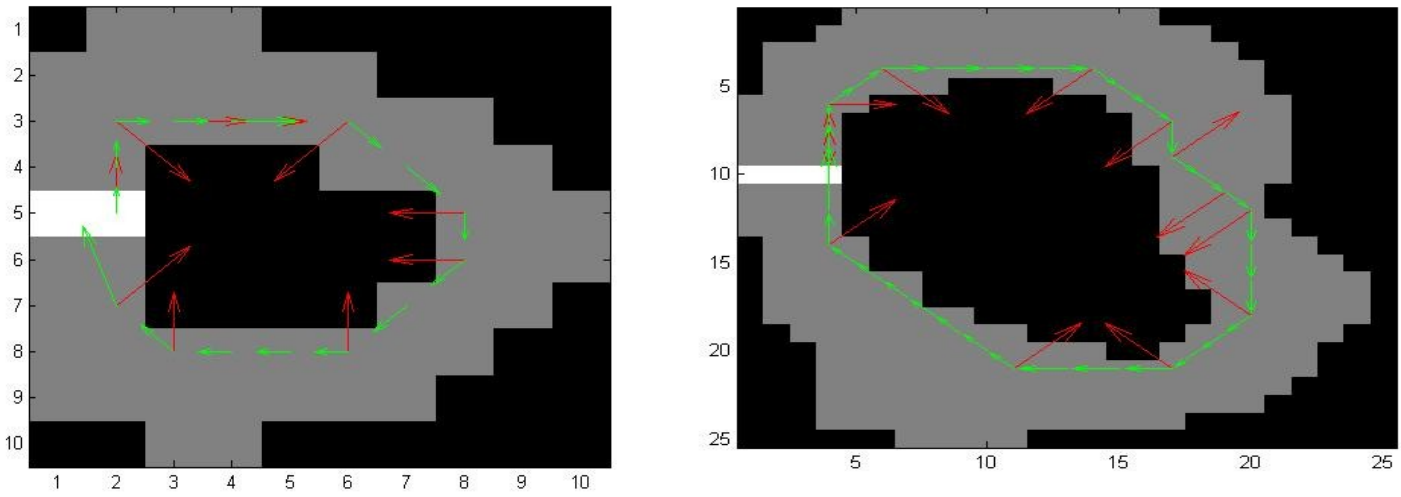


Fig 4 Configurations for the car race problem with Value Iteration Solutions

For car race problem, the number of states is much higher. For a 10X10 track, there are 717 states while for the 25X25 track, there are as many as 4149 states! It is interesting to see how well does value iteration scale up.

In the diagram, the green arrows indicate the position vectors, while the red arrows indicate the acceleration vectors.

For the worse case (25X25), value iteration took 1980ms and 42 iterations for convergence.
Another observation can be made by tweaking the probability of an error. I expect the paths to differ and hence, the number of iterations taken too.
In our original case, I had taken error probabilities to be 0.1 . In order to get a better understanding of the impact of this parameter, I used an exaggerated value of 0.5 to obtain paths as in Fig 4.
Fig 4(a) is particularly interesting(10X10) as in this case, because of such high error probability, the car does not even finish the race but continues to drive into an obstacle.

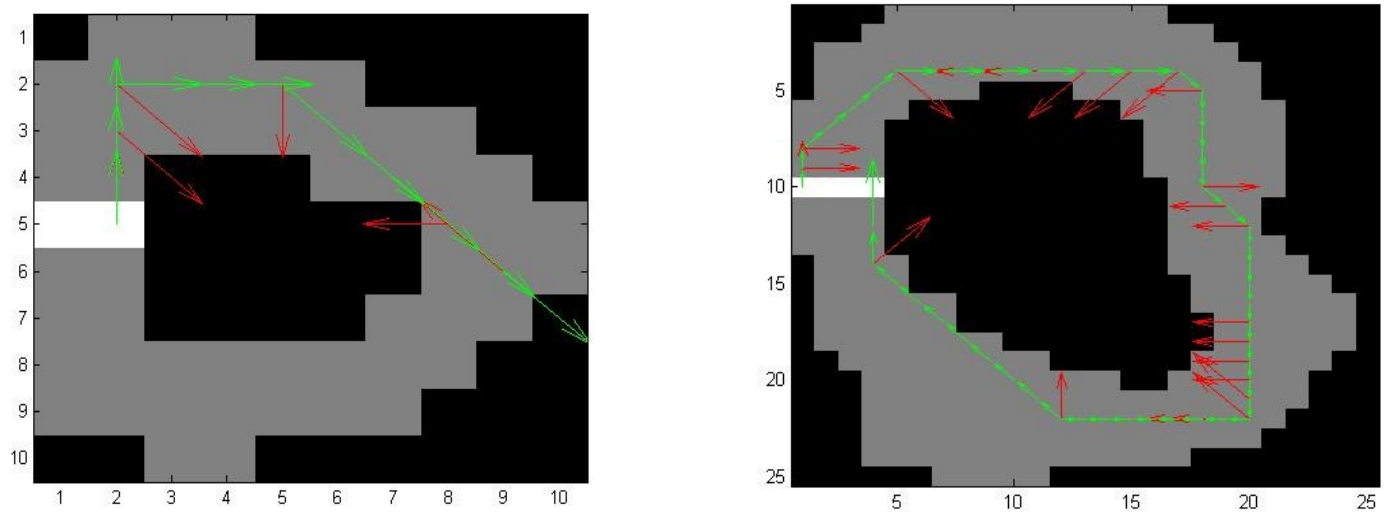


Fig 5. Paths when error probability = 0.5

Policy Iteration:

For the maze problem, I ran the policy iteration algorithm to find the exact same policies as those by value iteration. This is expected for problems with relatively less number of states. The values (expected infinite discounted reward) though differ.
But what is really interesting is the number of iterations and execution time as compared to value iteration to reach to the same policy.

The following graphs give a neat comparison between the two as there is a definite trend.

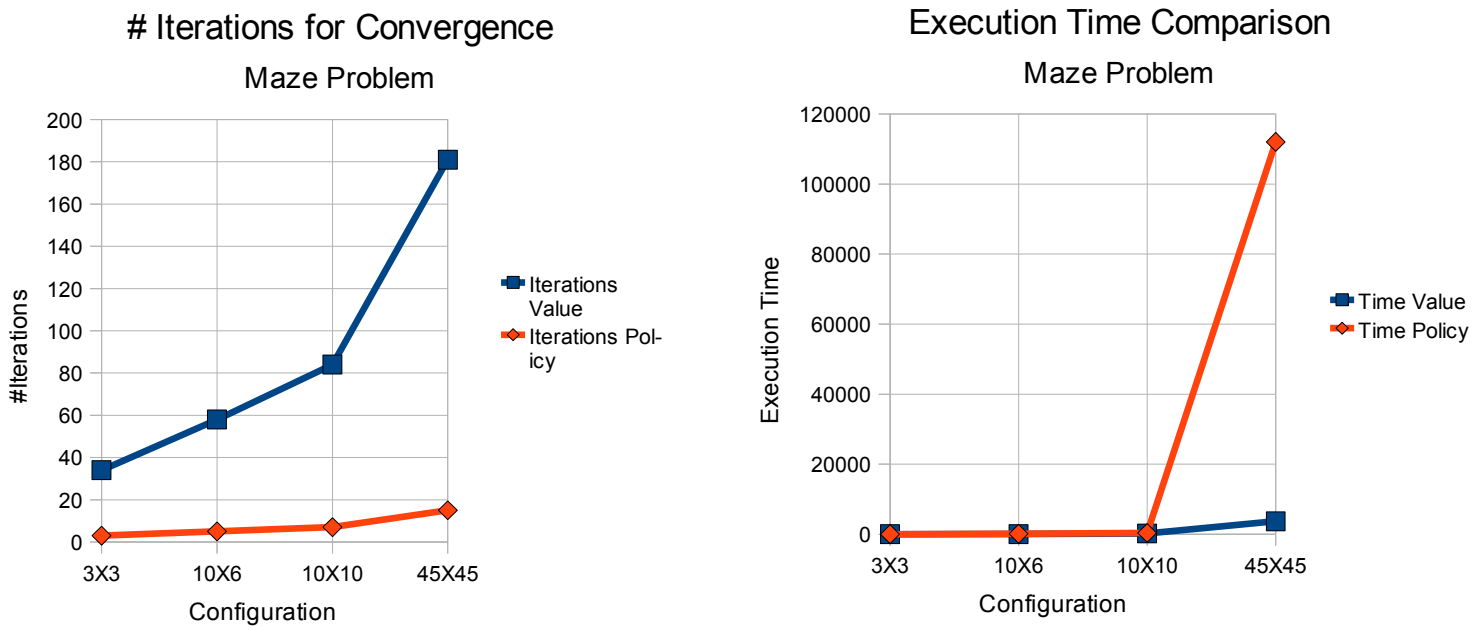
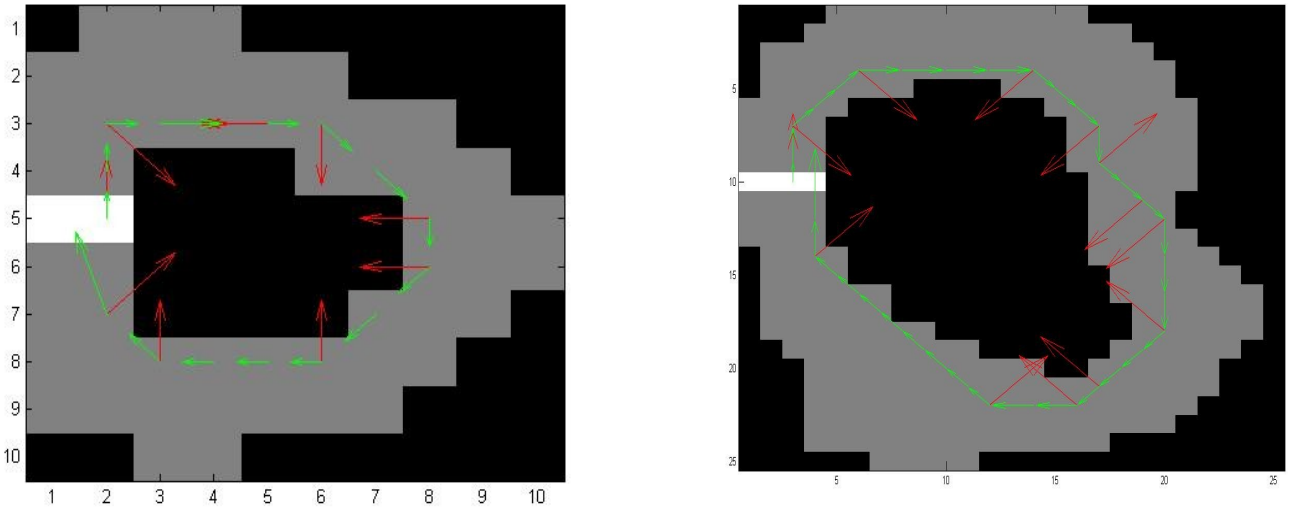


Fig 6 : Comparisons of Value Iteration and Policy Iteration w.r.t iterations and execution time for Maze Problem

It is clear that Value Iteration takes considerably more number of iterations for convergence but this should not be taken a reason for choosing policy iteration over it. Closer inspection reveals, that in spite of a deceptively low number of iterations, Policy iteration takes considerably more time, at least for the maze problem.

For the Car Race problem, the results are a little different for policy iteration. If we look carefully,



We see that the directional vectors are the same in both the cases (green) while the acceleration vectors (in red) are subtly different. This is probably expected in problems like these with huge number of states. I explain this because of the fact that there are more than one possible optimal policies which are being discovered in different runs.

Comparison for iteration count and execution time for the two policies for this problem are as:

Fig 8 Iterations and Execution Time Comparison for Value and Policy Iteration (Car Race Problem)

The trend holds exactly like it did in the maze problem which is interesting because in the literature I read (Andrew Moore's slides) it was mentioned that for lots of actions, Policy Iteration is preferred. Though not directly obvious, looking carefully shows that the time execution gap between the two algorithms is a lot less here than the maze problem. This leads me to believe that as the problem space will get larger and larger, this gap will close even further, thus possibly justifying the author's claim.

One more experiment worth repeating for policy iteration is the variation in the error probability. Once again, I increase this to 0.5 to get the following results:

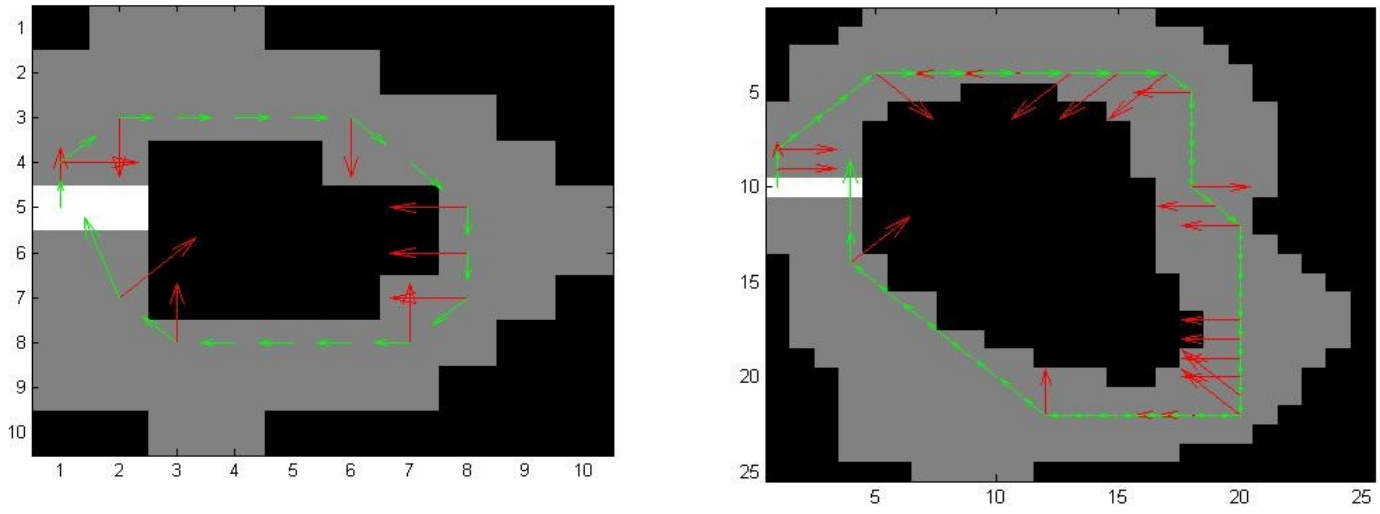


Fig 9. Paths when error probability = 0.5

For some reason, policy iteration seems to give a pretty robust policy. Even with error probability 0.5, we still see cohesive paths as opposed to Fig 5. However, I am still not convinced enough to make a claim as bold as “Policy Iteration is more robust to error probability”. This is an interesting observation that will require further reading up.

Conclusion:

There were two very clear conclusions from this assignment:

1. Value iteration in general requires a greater number of iterations for convergence.
2. These iterations, however are simpler to calculate as the execution time in most cases was still lower than Policy Iteration.

Another possible conclusion is about the stability of the solutions provided by Policy iteration, but this requires further investigation.