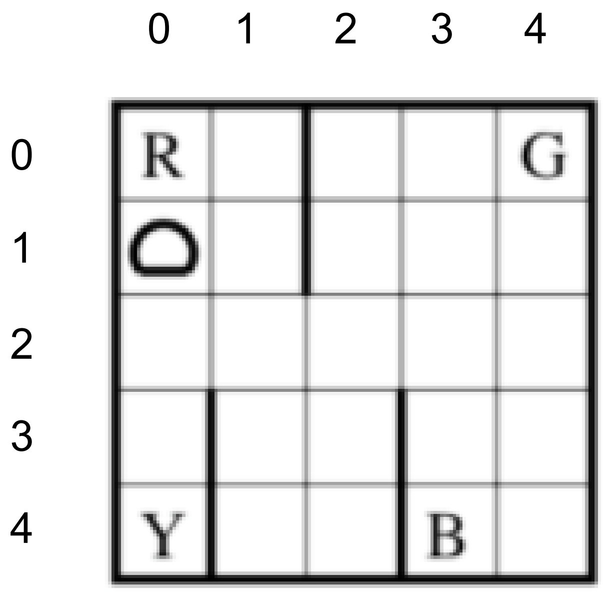
COL333 Artificial Intelligence

Aman Verma (2019CS50419)
Prakhar Aggarwal (2019CS50441)



Part A: Computing Policies

1.) Formulation of the taxi domain problem as MDP

State Space: Every state is a 5-tuple consisting of a binary variable (1 if the passenger is picked, 0 otherwise), taxi's and passenger's location (row and column). In total, we have R^2C^2 + RC states (R-rows, C-cols). If the passenger is not picked then RxCxRxC states are possible, and if picked then both are forced to be in the same cell, so RxC states, giving a net 625+25=650 states for the small grid. For the larger grid, 10100 states are created.

Action Space: There are six actions in the domain: (a) four navigation actions that move the taxi one grid cell to the North (N), South (S), East (E) or the West (W) directions, (b) a Pickup action (U), where the taxi attempts to pick up the passenger and (c) a Putdown action (D), where the taxi drops off the passenger. Each of the R^2C^2 + RC state has these 6 actions except for the terminal state.

Transition Model: For each transition, we have a certain probability assigned to it. For a navigation action; Each navigation action succeeds with a probability of 0.85 and moves in a random other direction with a probability of 0.15. Movements that attempt to cross the

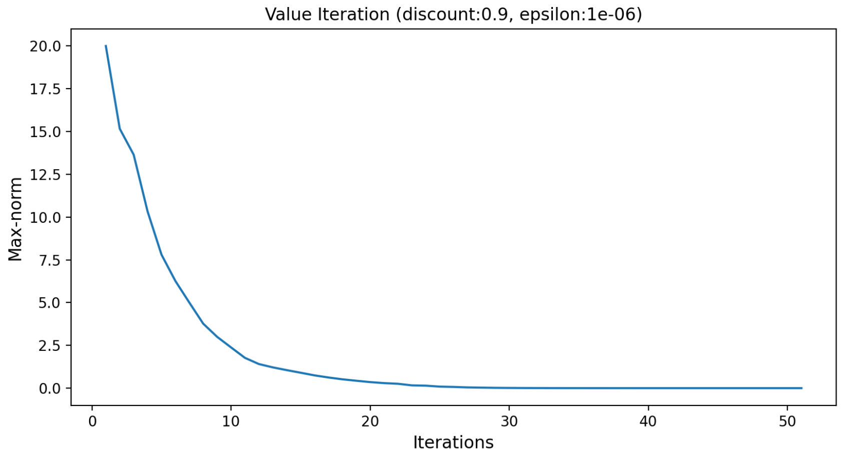
boundary of the grid or a wall result in no change in the state. The Pickup and the Putdown actions are deterministic and lead to their exact intended effects.
Reward Model: The taxi agent receives a reward of (-1) for performing each action. A reward of (+20) is received when the taxi agent successfully delivers the passenger to the destination grid cell. Further, a reward of (-10) is received if the taxi attempts to Pickup or Putdown a passenger when the taxi and the passenger are not located in the same grid cell.

The MDP class has a simulate function, which, given a state and action, generates the next state, taking into account the stochastic nature of the actions. It returns the next state and the transition reward.

The MDP class also takes as input the starting passenger position, taxi position, and destination depot position. In case these are not provided, these are generated randomly from the state space.

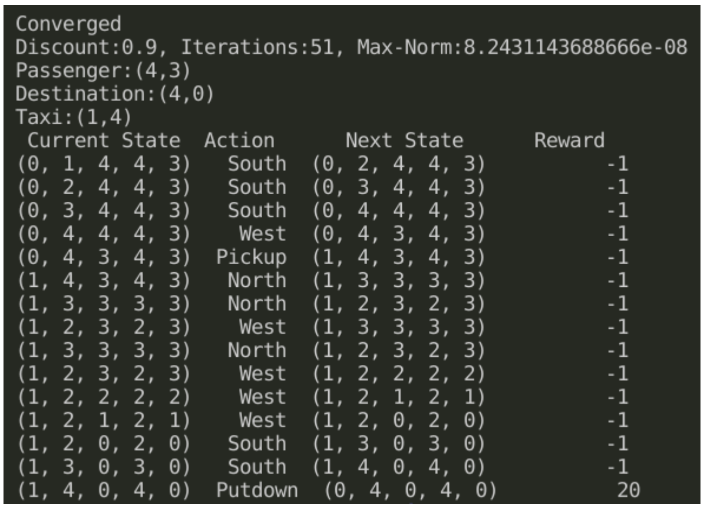
2.) Value Iteration for the taxi domain

(a) Discount: 0.9
Epsilon: 10^{-6} (maximum error allowed in the value of any state)
Iterations to converge: 51



We can observe that the max-norm loss decreases with an increase in iterations. This implies convergence in the utility values. On observing the optimal policy obtained, we can say that the model has learned optimal action.

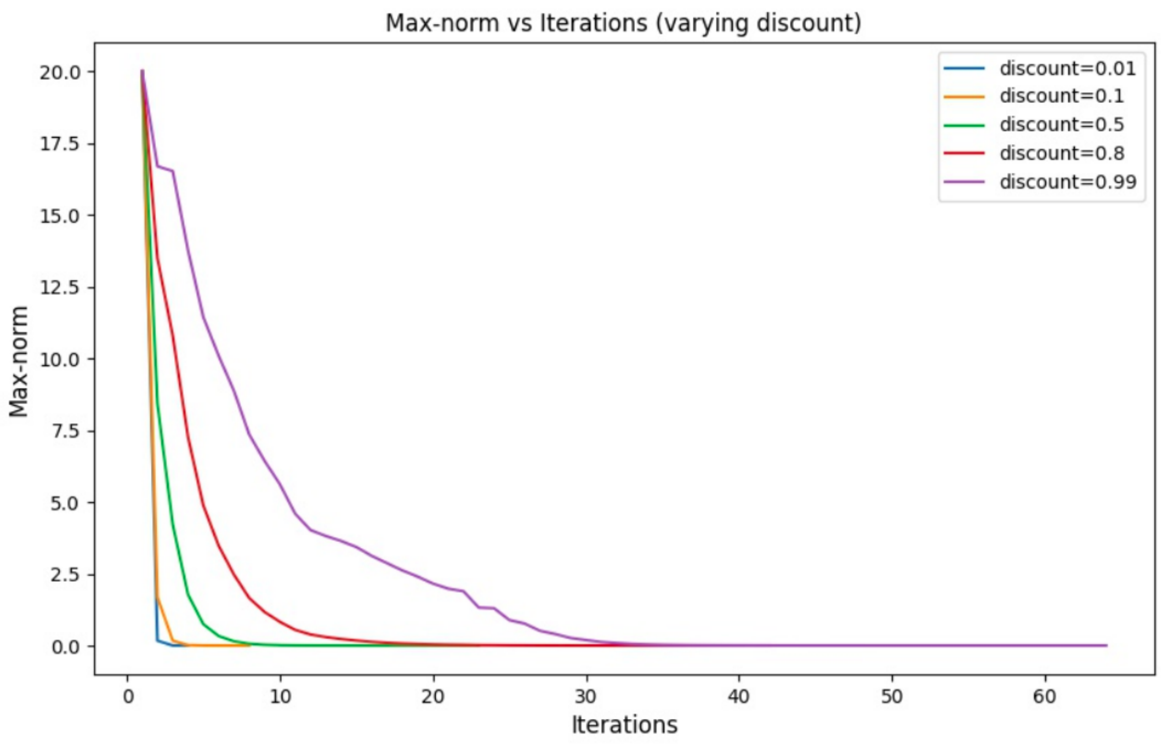
Eg.



```
Converged
Discount:0.9, Iterations:51, Max-Norm:8.2431143688666e-08
Passenger:(4,3)
Destination:(4,0)
Taxi:(1,4)
 Current State  Action      Next State      Reward
(0, 1, 4, 4, 3)   South  (0, 2, 4, 4, 3)          -1
(0, 2, 4, 4, 3)   South  (0, 3, 4, 4, 3)          -1
(0, 3, 4, 4, 3)   South  (0, 4, 4, 4, 3)          -1
(0, 4, 4, 4, 3)    West  (0, 4, 3, 4, 3)          -1
(0, 4, 3, 4, 3)  Pickup  (1, 4, 3, 4, 3)          -1
(1, 4, 3, 4, 3)   North  (1, 3, 3, 3, 3)          -1
(1, 3, 3, 3, 3)   North  (1, 2, 3, 2, 3)          -1
(1, 2, 3, 2, 3)    West  (1, 3, 3, 3, 3)          -1
(1, 3, 3, 3, 3)   North  (1, 2, 3, 2, 3)          -1
(1, 2, 3, 2, 3)    West  (1, 2, 2, 2, 2)          -1
(1, 2, 2, 2, 2)    West  (1, 2, 1, 2, 1)          -1
(1, 2, 1, 2, 1)    West  (1, 2, 0, 2, 0)          -1
(1, 2, 0, 2, 0)   South  (1, 3, 0, 3, 0)          -1
(1, 3, 0, 3, 0)   South  (1, 4, 0, 4, 0)          -1
(1, 4, 0, 4, 0)  Putdown  (0, 4, 0, 4, 0)          20
```

(b) Relation between discount factor(γ) and the rate of convergence

Discount varies over {0.01, 0.1, 0.5, 0.8, 0.99} with epsilon = 10^{-6}

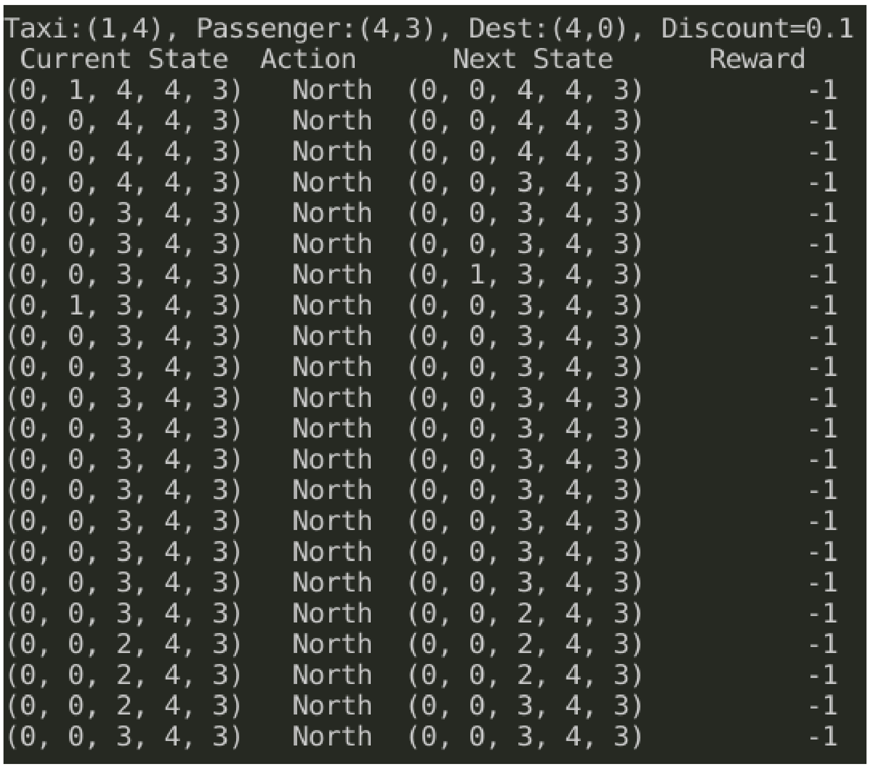


Observations: As we can see from the above graph, for lower discount, utility values converge very quickly. This is because, for lower γ, the reward at farther states is not back-propagated. Also, due to smaller γ, utility changes by a very small value. Hence, the model converges very quickly. But, as it doesn't look at the reward of farther states, the model doesn't converge optimally. Actions are much more impulsive and depend on best current reward for very low discount values.

Whereas for higher γ, the number of iterations required to converge increases but in return, it learns the optimal policy. (supporting example given in (c))

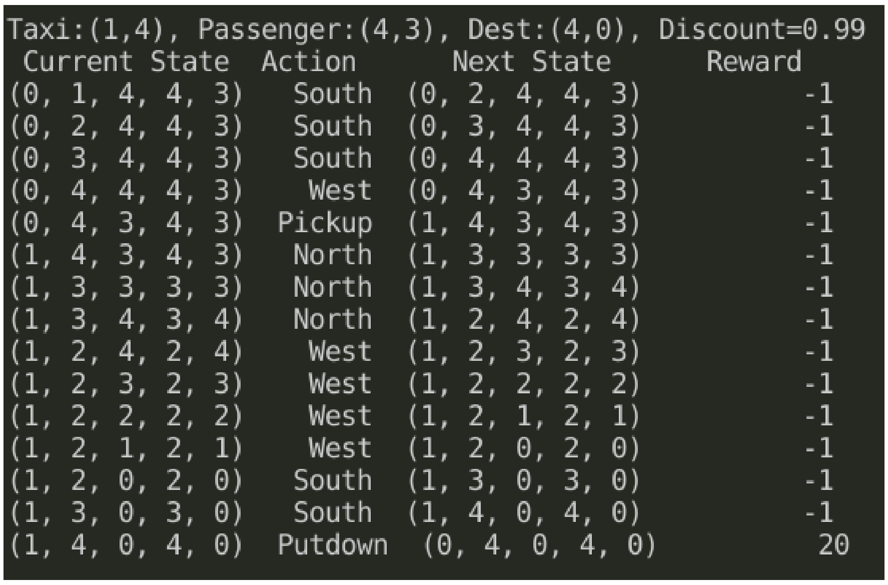
(c) Passenger Location: (4,3)
Passenger Destination: (4,0)
Taxi Location: (1,4)

Policy Simulation for γ = 0.1 and its state action sequence:



```
Taxi:(1,4), Passenger:(4,3), Dest:(4,0), Discount=0.1
 Current State  Action      Next State      Reward
(0, 1, 4, 4, 3)   North  (0, 0, 4, 4, 3)        -1
(0, 0, 4, 4, 3)   North  (0, 0, 4, 4, 3)        -1
(0, 0, 4, 4, 3)   North  (0, 0, 4, 4, 3)        -1
(0, 0, 4, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 1, 3, 4, 3)        -1
(0, 1, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 2, 4, 3)        -1
(0, 0, 2, 4, 3)   North  (0, 0, 2, 4, 3)        -1
(0, 0, 2, 4, 3)   North  (0, 0, 2, 4, 3)        -1
(0, 0, 2, 4, 3)   North  (0, 0, 3, 4, 3)        -1
(0, 0, 3, 4, 3)   North  (0, 0, 3, 4, 3)        -1
```

Policy Simulation for γ = 0.99 and its state action sequence:

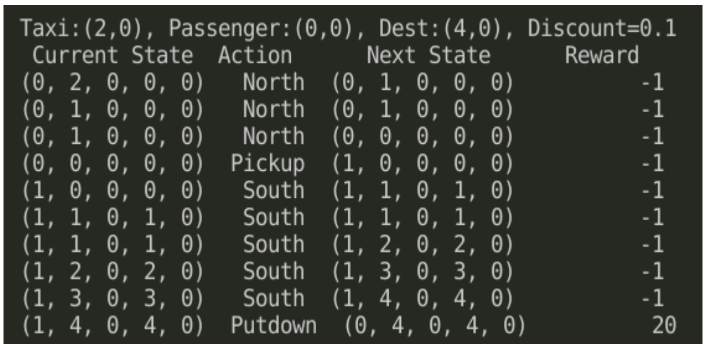


```
Taxi:(1,4), Passenger:(4,3), Dest:(4,0), Discount=0.99
 Current State  Action      Next State      Reward
(0, 1, 4, 4, 3)   South  (0, 2, 4, 4, 3)        -1
(0, 2, 4, 4, 3)   South  (0, 3, 4, 4, 3)        -1
(0, 3, 4, 4, 3)   South  (0, 4, 4, 4, 3)        -1
(0, 4, 4, 4, 3)    West  (0, 4, 3, 4, 3)        -1
(0, 4, 3, 4, 3)  Pickup  (1, 4, 3, 4, 3)        -1
(1, 4, 3, 4, 3)   North  (1, 3, 3, 3, 3)        -1
(1, 3, 3, 3, 3)   North  (1, 3, 4, 3, 4)        -1
(1, 3, 4, 3, 4)   North  (1, 2, 4, 2, 4)        -1
(1, 2, 4, 2, 4)    West  (1, 2, 3, 2, 3)        -1
(1, 2, 3, 2, 3)    West  (1, 2, 2, 2, 2)        -1
(1, 2, 2, 2, 2)    West  (1, 2, 1, 2, 1)        -1
(1, 2, 1, 2, 1)    West  (1, 2, 0, 2, 0)        -1
(1, 2, 0, 2, 0)   South  (1, 3, 0, 3, 0)        -1
(1, 3, 0, 3, 0)   South  (1, 4, 0, 4, 0)        -1
(1, 4, 0, 4, 0)  Putdown  (0, 4, 0, 4, 0)        20
```

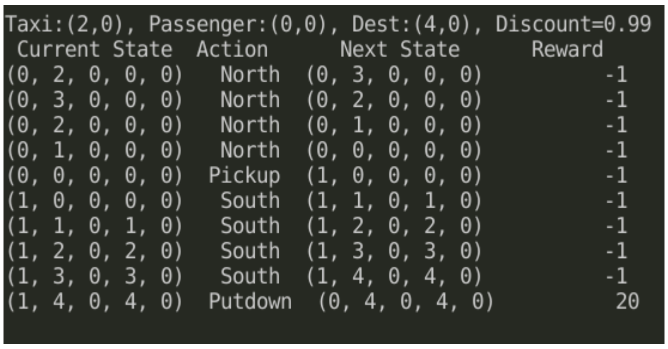
Observations: As can be seen from the above figures, the model with discount = 0.1 does converges whereas for discount = 0.99 converges. We can see that for discount=0.1, the agent performs **North** action most of the time as it doesn't know the optimal policy. The reason for this is explained in (b) (reward backpropagation problem). Simulation is also incorporated in this.

Policy Simulation for γ = 0.1 and γ = 0.99 by keeping the goal the same and varying the start states (for the taxi and the passenger).

Passenger Location: (0,0), Passenger Destination: (4,0), Taxi Location: (2,0)



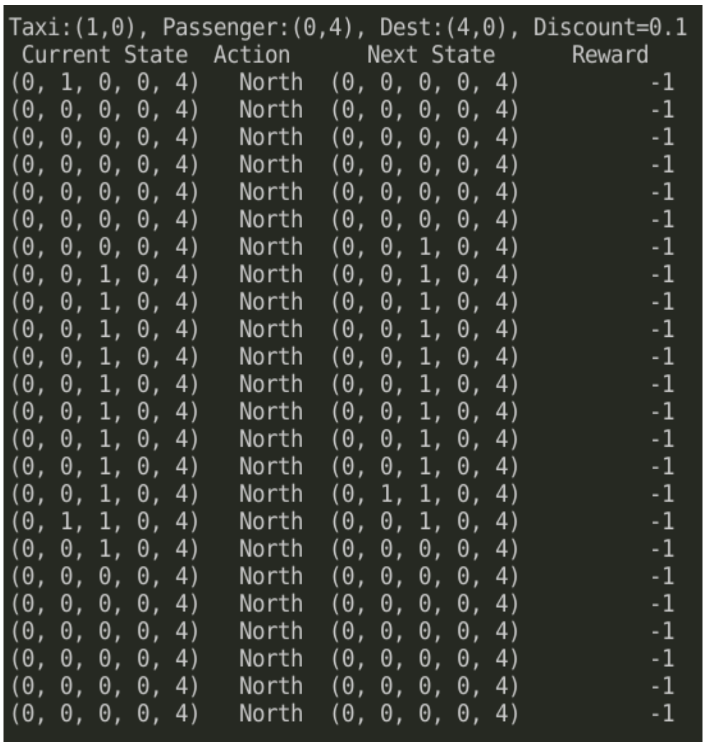
```
Taxi:(2,0), Passenger:(0,0), Dest:(4,0), Discount=0.1
 Current State  Action      Next State      Reward
(0, 2, 0, 0, 0)   North  (0, 1, 0, 0, 0)          -1
(0, 1, 0, 0, 0)   North  (0, 1, 0, 0, 0)          -1
(0, 1, 0, 0, 0)   North  (0, 0, 0, 0, 0)          -1
(0, 0, 0, 0, 0)  Pickup  (1, 0, 0, 0, 0)          -1
(1, 0, 0, 0, 0)   South  (1, 1, 0, 1, 0)          -1
(1, 1, 0, 1, 0)   South  (1, 1, 0, 1, 0)          -1
(1, 1, 0, 1, 0)   South  (1, 2, 0, 2, 0)          -1
(1, 2, 0, 2, 0)   South  (1, 3, 0, 3, 0)          -1
(1, 3, 0, 3, 0)   South  (1, 4, 0, 4, 0)          -1
(1, 4, 0, 4, 0)  Putdown  (0, 4, 0, 4, 0)         20
```



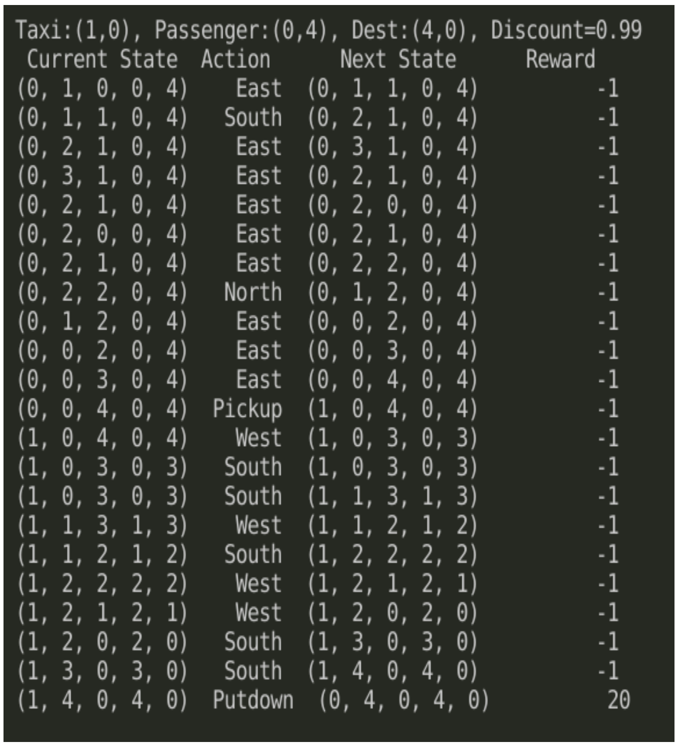
```
Taxi:(2,0), Passenger:(0,0), Dest:(4,0), Discount=0.99
 Current State  Action      Next State      Reward
(0, 2, 0, 0, 0)   North  (0, 3, 0, 0, 0)          -1
(0, 3, 0, 0, 0)   North  (0, 2, 0, 0, 0)          -1
(0, 2, 0, 0, 0)   North  (0, 1, 0, 0, 0)          -1
(0, 1, 0, 0, 0)   North  (0, 0, 0, 0, 0)          -1
(0, 0, 0, 0, 0)  Pickup  (1, 0, 0, 0, 0)          -1
(1, 0, 0, 0, 0)   South  (1, 1, 0, 1, 0)          -1
(1, 1, 0, 1, 0)   South  (1, 2, 0, 2, 0)          -1
(1, 2, 0, 2, 0)   South  (1, 3, 0, 3, 0)          -1
(1, 3, 0, 3, 0)   South  (1, 4, 0, 4, 0)          -1
(1, 4, 0, 4, 0)  Putdown  (0, 4, 0, 4, 0)         20
```

Observations: For this case, we can see that both models (for γ=0.1 and γ=0.99) follow the correct path. As the taxi -> passenger -> dest path is quite simple, γ=0.1 manages to learn this path. γ=0.99 anyway learns the optimal policy.

Passenger Location: (0,4), Passenger Destination: (4,0), Taxi Location: (1,0)



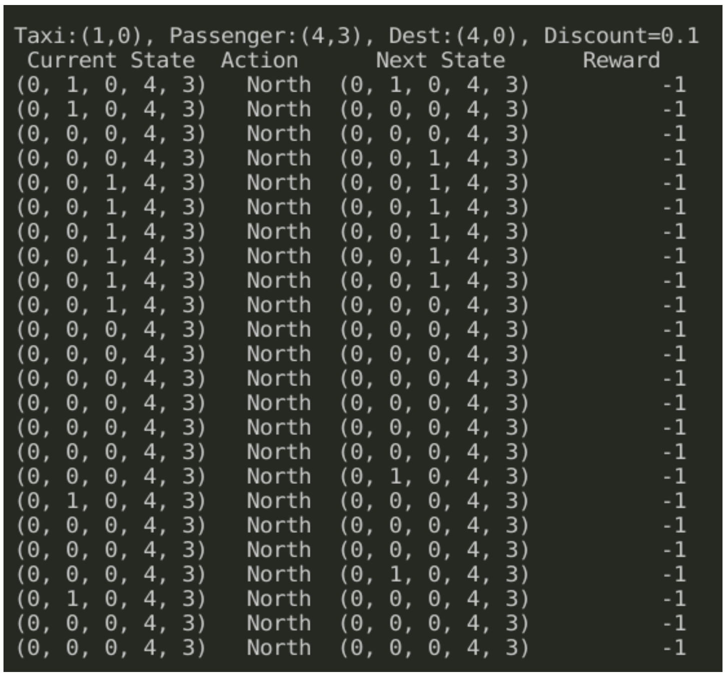
```
Taxi:(1,0), Passenger:(0,4), Dest:(4,0), Discount=0.1
 Current State  Action      Next State      Reward
(0, 1, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 1, 1, 0, 4)          -1
(0, 1, 1, 0, 4)   North  (0, 0, 1, 0, 4)          -1
(0, 0, 1, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
(0, 0, 0, 0, 4)   North  (0, 0, 0, 0, 4)          -1
```



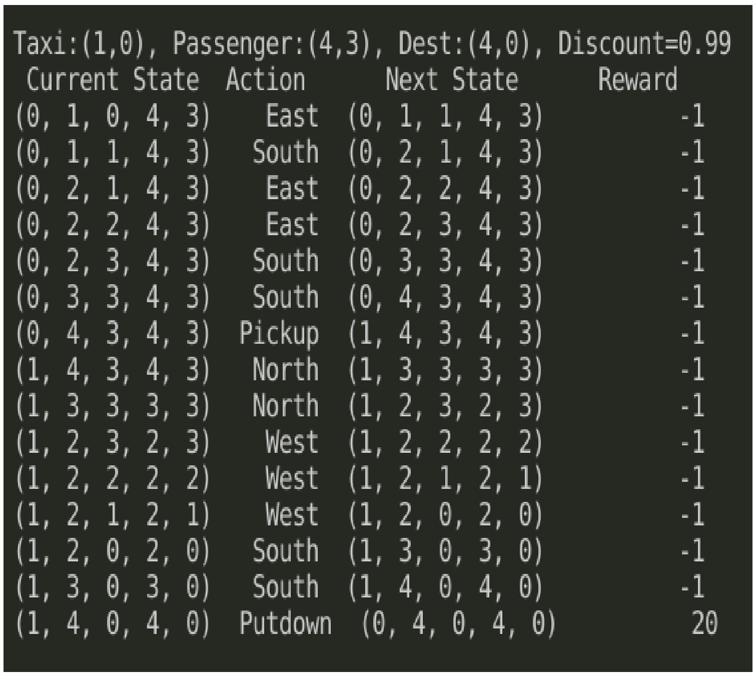
```
Taxi:(1,0), Passenger:(0,4), Dest:(4,0), Discount=0.99
 Current State  Action      Next State      Reward
(0, 1, 0, 0, 4)    East  (0, 1, 1, 0, 4)          -1
(0, 1, 1, 0, 4)   South  (0, 2, 1, 0, 4)          -1
(0, 2, 1, 0, 4)    East  (0, 3, 1, 0, 4)          -1
(0, 3, 1, 0, 4)    East  (0, 2, 1, 0, 4)          -1
(0, 2, 1, 0, 4)    East  (0, 2, 0, 0, 4)          -1
(0, 2, 0, 0, 4)    East  (0, 2, 1, 0, 4)          -1
(0, 2, 1, 0, 4)    East  (0, 2, 2, 0, 4)          -1
(0, 2, 2, 0, 4)   North  (0, 1, 2, 0, 4)          -1
(0, 1, 2, 0, 4)    East  (0, 0, 2, 0, 4)          -1
(0, 0, 2, 0, 4)    East  (0, 0, 3, 0, 4)          -1
(0, 0, 3, 0, 4)    East  (0, 0, 4, 0, 4)          -1
(0, 0, 4, 0, 4)  Pickup  (1, 0, 4, 0, 4)          -1
(1, 0, 4, 0, 4)    West  (1, 0, 3, 0, 3)          -1
(1, 0, 3, 0, 3)   South  (1, 0, 3, 0, 3)          -1
(1, 0, 3, 0, 3)   South  (1, 1, 3, 1, 3)          -1
(1, 1, 3, 1, 3)    West  (1, 1, 2, 1, 2)          -1
(1, 1, 2, 1, 2)   South  (1, 2, 2, 2, 2)          -1
(1, 2, 2, 2, 2)    West  (1, 2, 1, 2, 1)          -1
(1, 2, 1, 2, 1)    West  (1, 2, 0, 2, 0)          -1
(1, 2, 0, 2, 0)   South  (1, 3, 0, 3, 0)          -1
(1, 3, 0, 3, 0)   South  (1, 4, 0, 4, 0)          -1
(1, 4, 0, 4, 0)  Putdown  (0, 4, 0, 4, 0)         20
```

Observations: In this case, γ=0.1 is unable to learn the optimal policy (explained in (b)) and performs **North** action most of the time, whereas γ=0.99 learns the optimal policy.

Passenger Location: (4,3), Passenger Destination: (4,0), Taxi Location: (1,0)



```
Taxi:(1,0), Passenger:(4,3), Dest:(4,0), Discount=0.1
 Current State  Action      Next State       Reward
(0, 1, 0, 4, 3)   North  (0, 1, 0, 4, 3)          -1
(0, 1, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 1, 4, 3)          -1
(0, 0, 1, 4, 3)   North  (0, 0, 1, 4, 3)          -1
(0, 0, 1, 4, 3)   North  (0, 0, 1, 4, 3)          -1
(0, 0, 1, 4, 3)   North  (0, 0, 1, 4, 3)          -1
(0, 0, 1, 4, 3)   North  (0, 0, 1, 4, 3)          -1
(0, 0, 1, 4, 3)   North  (0, 0, 1, 4, 3)          -1
(0, 0, 1, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 1, 0, 4, 3)          -1
(0, 1, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 1, 0, 4, 3)          -1
(0, 1, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
(0, 0, 0, 4, 3)   North  (0, 0, 0, 4, 3)          -1
```



```
Taxi:(1,0), Passenger:(4,3), Dest:(4,0), Discount=0.99
 Current State  Action      Next State      Reward
(0, 1, 0, 4, 3)    East  (0, 1, 1, 4, 3)         -1
(0, 1, 1, 4, 3)   South  (0, 2, 1, 4, 3)         -1
(0, 2, 1, 4, 3)    East  (0, 2, 2, 4, 3)         -1
(0, 2, 2, 4, 3)    East  (0, 2, 3, 4, 3)         -1
(0, 2, 3, 4, 3)   South  (0, 3, 3, 4, 3)         -1
(0, 3, 3, 4, 3)   South  (0, 4, 3, 4, 3)         -1
(0, 4, 3, 4, 3)  Pickup  (1, 4, 3, 4, 3)         -1
(1, 4, 3, 4, 3)   North  (1, 3, 3, 3, 3)         -1
(1, 3, 3, 3, 3)   North  (1, 2, 3, 2, 3)         -1
(1, 2, 3, 2, 3)    West  (1, 2, 2, 2, 2)         -1
(1, 2, 2, 2, 2)    West  (1, 2, 1, 2, 1)         -1
(1, 2, 1, 2, 1)    West  (1, 2, 0, 2, 0)         -1
(1, 2, 0, 2, 0)   South  (1, 3, 0, 3, 0)         -1
(1, 3, 0, 3, 0)   South  (1, 4, 0, 4, 0)         -1
(1, 4, 0, 4, 0)  Putdown  (0, 4, 0, 4, 0)         20
```

For this case too, the γ=0.99 learns the optimal policy but γ=0.1 doesn’t.

Observations: We can say that γ=0.1 doesn’t learn the optimal policy but if the start state is close to the dest state (with taxi too), then it manages to follow the optimal policy in the (illustrated by example 1).

3.) Policy Iteration for the taxi domain

(a) The Policy Evaluation step (2 ways):

- Linear Algebra method
  - As the Bellman equations in policy iteration are a linear system of equations without the max operator, evaluating utility values is the same as solving a set of linear equations.
  - A matrix "**A**" is calculated (corresponding to Bellman eqs. formed, A: n*n (n:states))
  - Matrix B is computed (equation(**A[i]**) = **B[i]**)
  - So, we can formulate it as **A*x = B** (x: utility values)
  - **x** can be obtained as **x = A^{-1}B** (in case A is singular, we use pseudo inverse)

- Iterative method
  - $$V_0^\pi(s) = 0$$
    $$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^\pi(s')]$$
  - This method computes the approximate utility values using the Bellman equation.

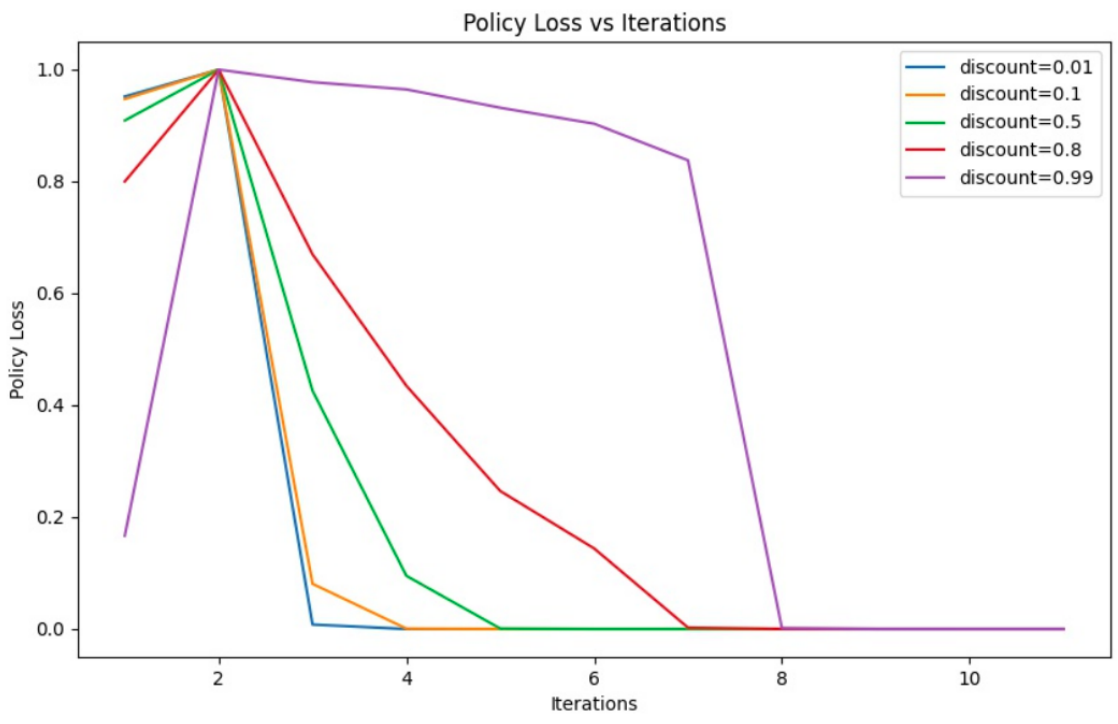
- The linear algebra method takes O(n^3) time for 1 update which is computationally expensive and gives the exact utility values. Whereas, the iterative method is significantly faster (O(n^2) time for 1 update) but gives approximate utility values. Also, the number of iterations required to converge in the linear algebra method is quite less than those required in the iterative method.

- Note that the policy iteration iterative method takes O(n^2) time in the worst case but, according to our implementation (for the given MDP), we know for sure that we have at max 5 different states to go to from the current state. So the extra factor of n gets reduced to 5 i.e. O(5n) or O(n)

- So, if the number of states(n) is less, then the linear algebra method is better, but if the number of states(n) is large, then the iterative method will give promising results.

<u>(b) Policy Loss</u>

Passenger Location: (4,3)
Passenger Destination: (4,0)
Taxi Location: (1,4)
Discount varies over: {0.01, 0.1, 0.5, 0.8, 0.99}



Observations: As we can see from the above graph, the optimal policy is computed in a significantly lesser number of iterations than those required in the value iteration algorithm. Using the same explanation of the backpropagation problem (given in (b)), we can say that our model converges quickly for lower values of γ (though it doesn't learn the optimal policy) whereas, for higher γ, iterations required are more but it learns the optimal policy.

Policy iteration iterative turns out to be much more efficient in terms of run time because the number of iterations required for convergence increase by a small constant factor(around 5-6) whereas the time for one iteration is O(n) for the iterative one whereas it is O(n^3) for the linear algebra method. Given a large number of states(650), the iterative policy iteration runs significantly faster).

Part B: Incorporating Learning

1.) Model-Free Learning

a.) Q-learning

```
q_learning(params, episodes, learning_rate, discount,
epsilon_exploration=0.1, decay=False)
```

We use the following update for Q-learning:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\,[sample]$$

Here for generating a sample we have 2 possibilities:

1. With probability 1-epsilon, we select the best action

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

2. With probability epsilon, we select a random action and choose the Reward + discounted Q-value of the next state for the sample value.

In Q-learning, only the state information is passed between steps.

b.) Q-learning with decay in exploration

```
q_learning(params, episodes, learning_rate, discount,
epsilon_exploration=0.1, decay=True)
```

Similar to part a) but with decayed exploration rate as described below.

c.) SARSA

```
sarsa(params, episodes, learning_rate, discount,
epsilon_exploration=0.1, decay=False)
```

Here, we use the SARSA algorithm for learning. Rather than using only the state, action, reward, and new state, we also include the next state action in our update. Thus we use the selected action from the previous iteration and observe the next state and reward, then use the epsilon greedy strategy to select the action of the next state and use this to update the current states value using the selected action.
In SARSA, the state and action, both are passed between the steps(since the next state action is determined in the current step itself).

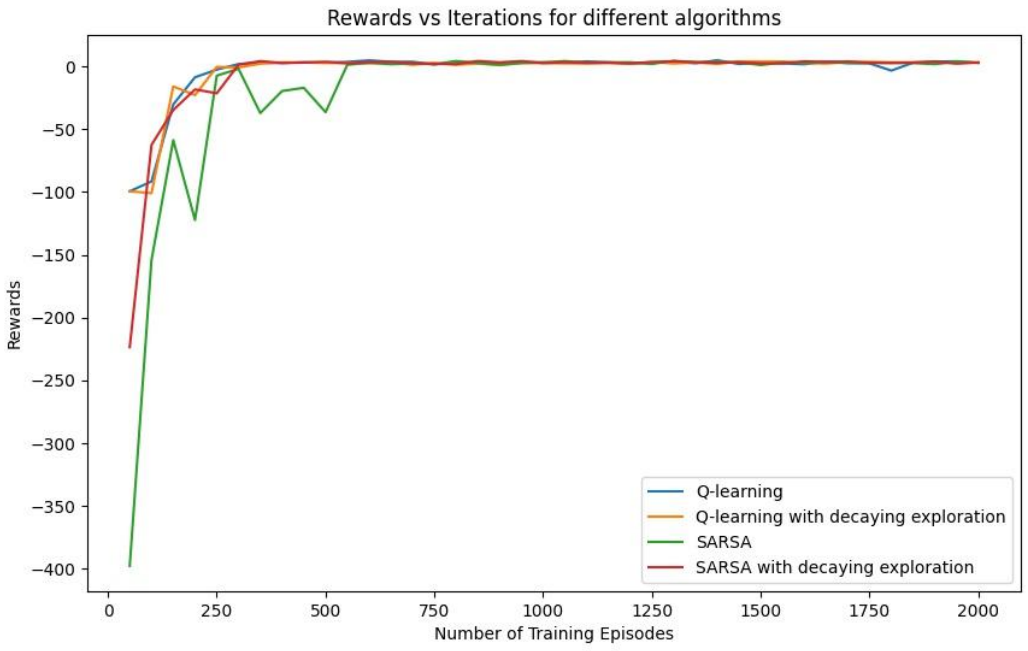
d.) SARSA with decay in exploration

```
sarsa(params, episodes, learning_rate, discount,
epsilon_exploration=0.1, decay=True)
```

Similar to part c) but with a decayed exploration rate as described below.

Decay has been considered reciprocal in nature:
epsilon_decayed = epsilon/number_of_learning_updates
The above 4 methods have been implemented.

2.) Training



We run 2000 training episodes. For evaluation, we run 50 evaluation episodes after every 50 episodes of training. The averaged reward over these 50 episodes is plotted.

Algorithm	Discounted Reward(averaged over 50 runs)
Q-learning	2.8778704205779166
Q-learning with decay	3.4307976620414684
SARSA	2.9704180850751953
SARSA with decay	3.159710537188779

All of the algorithms learn the optimal policy equally well, but the rewards vary due to randomness in the instance sampling process.

Observations

- It can be observed from the above plot that Q-learning with decaying exploration is the first to converge along with Q-learning and SARSA with decayed exploration. SARSA takes the largest number of training episodes to converge.
- Since SARSA is an on policy algorithm, due to exploration steps which may be non-optimal, it takes longer to converge. Decayed SARSA still converges soon enough due to the exploration factor reducing substantially during training.
- By the end, the discounted rewards become slightly positive values as expected for all 4 of the algorithms as the optimal policy is learnt by each of the algorithms.

Thus, we conclude that **Q-learning with decayed exploration** is the best algorithm for the given MDP.

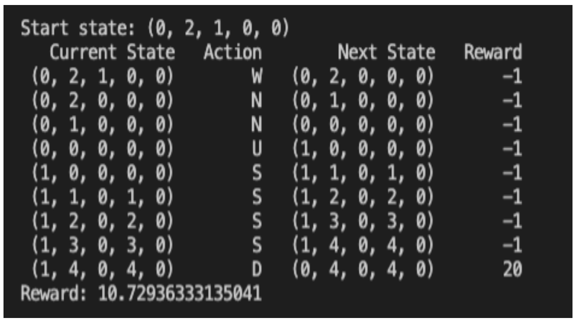
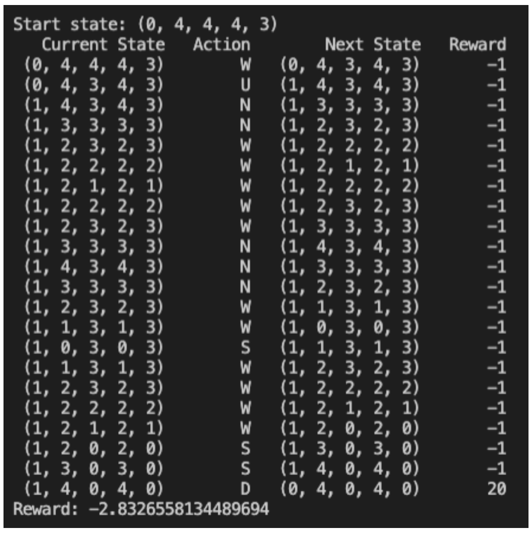
For the purpose of the report (4,0) was chosen as the destination depot(for the MDP). Implementation is generic and can be used with MDP with any parameter.

3.) Simulation

Since Q-learning with decay accumulated to the highest reward, we run 5 simulations on the policy generated on training. The results can be seen as follows (training was with destination depot (4,0):

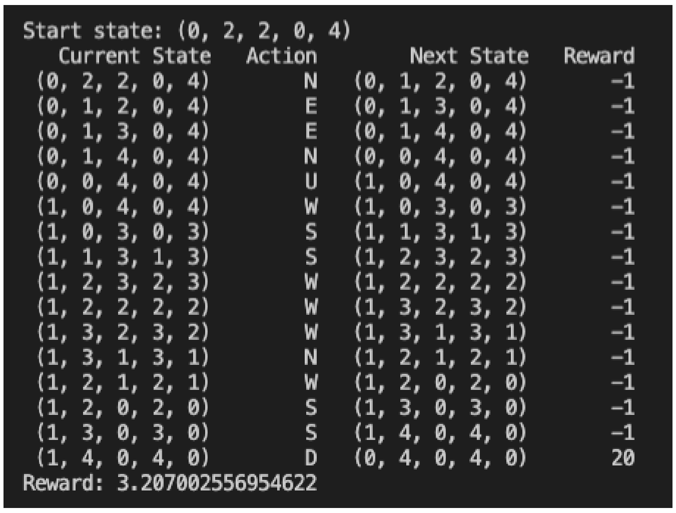
Actions in the below figure are mapped as follows:

{N-North, E-East, S-South, W-West, U-Pickup, D-Putdown}

Simulation	Comments
	Taxi starts at (2,1) Client is at (0,0) Drop is at (4,0) It can be clearly seen that the taxi directly seeks the client at first and then goes on to reach the destination directly. A very high reward is obtained in this case.
	Taxi starts at (4,4) Client is at (4,3) Drop is at (4,0) This case clearly shows that the taxi has learned the location of the walls. Due to the stochastic nature of the actions, a lot of delay is caused in this particular instance and the overall discounted reward turns out to be negative. However, at all times, the taxi acts optimally.

```
Start state: (0, 2, 1, 0, 0)
   Current State   Action       Next State   Reward
 (0, 2, 1, 0, 0)        W   (0, 2, 0, 0, 0)      -1
 (0, 2, 0, 0, 0)        N   (0, 1, 0, 0, 0)      -1
 (0, 1, 0, 0, 0)        N   (0, 0, 0, 0, 0)      -1
 (0, 0, 0, 0, 0)        U   (1, 0, 0, 0, 0)      -1
 (1, 0, 0, 0, 0)        S   (1, 1, 0, 1, 0)      -1
 (1, 1, 0, 1, 0)        S   (1, 2, 0, 2, 0)      -1
 (1, 2, 0, 2, 0)        S   (1, 3, 0, 3, 0)      -1
 (1, 3, 0, 3, 0)        S   (1, 4, 0, 4, 0)      -1
 (1, 4, 0, 4, 0)        D   (0, 4, 0, 4, 0)      20
Reward: 10.72936333135041
```

```
Start state: (0, 4, 4, 4, 3)
   Current State   Action       Next State   Reward
 (0, 4, 4, 4, 3)        W   (0, 4, 3, 4, 3)      -1
 (0, 4, 3, 4, 3)        U   (1, 4, 3, 4, 3)      -1
 (1, 4, 3, 4, 3)        N   (1, 3, 3, 3, 3)      -1
 (1, 3, 3, 3, 3)        N   (1, 2, 3, 2, 3)      -1
 (1, 2, 3, 2, 3)        W   (1, 2, 2, 2, 2)      -1
 (1, 2, 2, 2, 2)        W   (1, 2, 1, 2, 1)      -1
 (1, 2, 1, 2, 1)        W   (1, 2, 2, 2, 2)      -1
 (1, 2, 2, 2, 2)        W   (1, 2, 3, 2, 3)      -1
 (1, 2, 3, 2, 3)        W   (1, 3, 3, 3, 3)      -1
 (1, 3, 3, 3, 3)        N   (1, 4, 3, 4, 3)      -1
 (1, 4, 3, 4, 3)        N   (1, 3, 3, 3, 3)      -1
 (1, 3, 3, 3, 3)        N   (1, 2, 3, 2, 3)      -1
 (1, 2, 3, 2, 3)        W   (1, 1, 3, 1, 3)      -1
 (1, 1, 3, 1, 3)        W   (1, 0, 3, 0, 3)      -1
 (1, 0, 3, 0, 3)        S   (1, 1, 3, 1, 3)      -1
 (1, 1, 3, 1, 3)        W   (1, 2, 3, 2, 3)      -1
 (1, 2, 3, 2, 3)        W   (1, 2, 2, 2, 2)      -1
 (1, 2, 2, 2, 2)        W   (1, 2, 1, 2, 1)      -1
 (1, 2, 1, 2, 1)        W   (1, 2, 0, 2, 0)      -1
 (1, 2, 0, 2, 0)        S   (1, 3, 0, 3, 0)      -1
 (1, 3, 0, 3, 0)        S   (1, 4, 0, 4, 0)      -1
 (1, 4, 0, 4, 0)        D   (0, 4, 0, 4, 0)      20
Reward: -2.8326558134489694
```

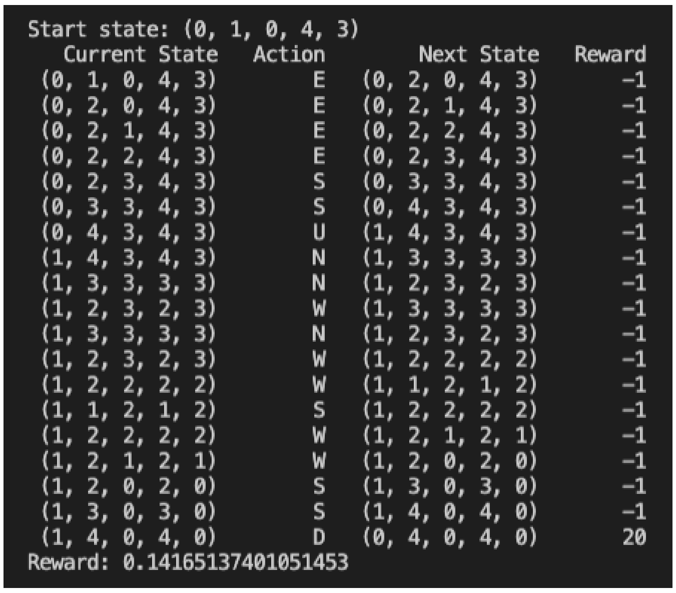


```
Start state: (0, 2, 2, 0, 4)
   Current State   Action       Next State   Reward
  (0, 2, 2, 0, 4)       N   (0, 1, 2, 0, 4)      -1
  (0, 1, 2, 0, 4)       E   (0, 1, 3, 0, 4)      -1
  (0, 1, 3, 0, 4)       E   (0, 1, 4, 0, 4)      -1
  (0, 1, 4, 0, 4)       N   (0, 0, 4, 0, 4)      -1
  (0, 0, 4, 0, 4)       U   (1, 0, 4, 0, 4)      -1
  (1, 0, 4, 0, 4)       W   (1, 0, 3, 0, 3)      -1
  (1, 0, 3, 0, 3)       S   (1, 1, 3, 1, 3)      -1
  (1, 1, 3, 1, 3)       S   (1, 2, 3, 2, 3)      -1
  (1, 2, 3, 2, 3)       W   (1, 2, 2, 2, 2)      -1
  (1, 2, 2, 2, 2)       W   (1, 3, 2, 3, 2)      -1
  (1, 3, 2, 3, 2)       W   (1, 3, 1, 3, 1)      -1
  (1, 3, 1, 3, 1)       N   (1, 2, 1, 2, 1)      -1
  (1, 2, 1, 2, 1)       W   (1, 2, 0, 2, 0)      -1
  (1, 2, 0, 2, 0)       S   (1, 3, 0, 3, 0)      -1
  (1, 3, 0, 3, 0)       S   (1, 4, 0, 4, 0)      -1
  (1, 4, 0, 4, 0)       D   (0, 4, 0, 4, 0)      20
Reward: 3.207002556954622
```

Taxi starts at (2,2)
Client is at (0,4)
Drop is at (4,0)

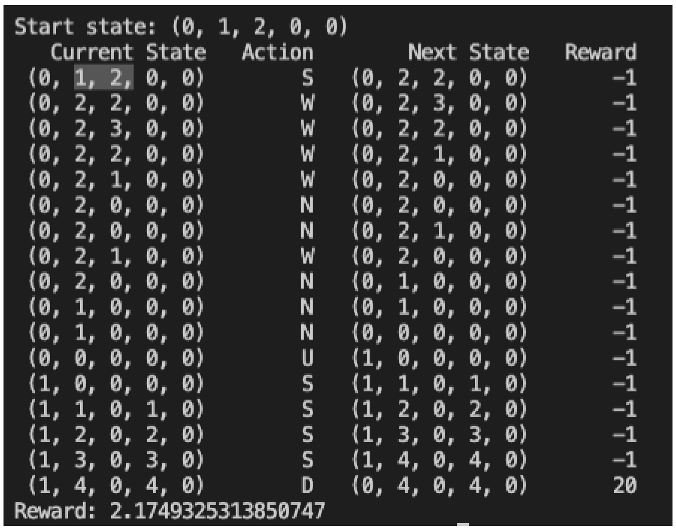
Another instance to show that the taxi has learnt to go “around” the walls.

It can also be seen that no redundant pickups and dropoffs are performed.



```
Start state: (0, 1, 0, 4, 3)
   Current State   Action       Next State   Reward
  (0, 1, 0, 4, 3)       E   (0, 2, 0, 4, 3)      -1
  (0, 2, 0, 4, 3)       E   (0, 2, 1, 4, 3)      -1
  (0, 2, 1, 4, 3)       E   (0, 2, 2, 4, 3)      -1
  (0, 2, 2, 4, 3)       E   (0, 2, 3, 4, 3)      -1
  (0, 2, 3, 4, 3)       S   (0, 3, 3, 4, 3)      -1
  (0, 3, 3, 4, 3)       S   (0, 4, 3, 4, 3)      -1
  (0, 4, 3, 4, 3)       U   (1, 4, 3, 4, 3)      -1
  (1, 4, 3, 4, 3)       N   (1, 3, 3, 3, 3)      -1
  (1, 3, 3, 3, 3)       N   (1, 2, 3, 2, 3)      -1
  (1, 2, 3, 2, 3)       W   (1, 3, 3, 3, 3)      -1
  (1, 3, 3, 3, 3)       N   (1, 2, 3, 2, 3)      -1
  (1, 2, 3, 2, 3)       W   (1, 2, 2, 2, 2)      -1
  (1, 2, 2, 2, 2)       W   (1, 1, 2, 1, 2)      -1
  (1, 1, 2, 1, 2)       S   (1, 2, 2, 2, 2)      -1
  (1, 2, 2, 2, 2)       W   (1, 2, 1, 2, 1)      -1
  (1, 2, 1, 2, 1)       W   (1, 2, 0, 2, 0)      -1
  (1, 2, 0, 2, 0)       S   (1, 3, 0, 3, 0)      -1
  (1, 3, 0, 3, 0)       S   (1, 4, 0, 4, 0)      -1
  (1, 4, 0, 4, 0)       D   (0, 4, 0, 4, 0)      20
Reward: 0.14165137401051453
```

Taxi starts at (1,0)
Client is at (4,3)
Drop is at (4,0)



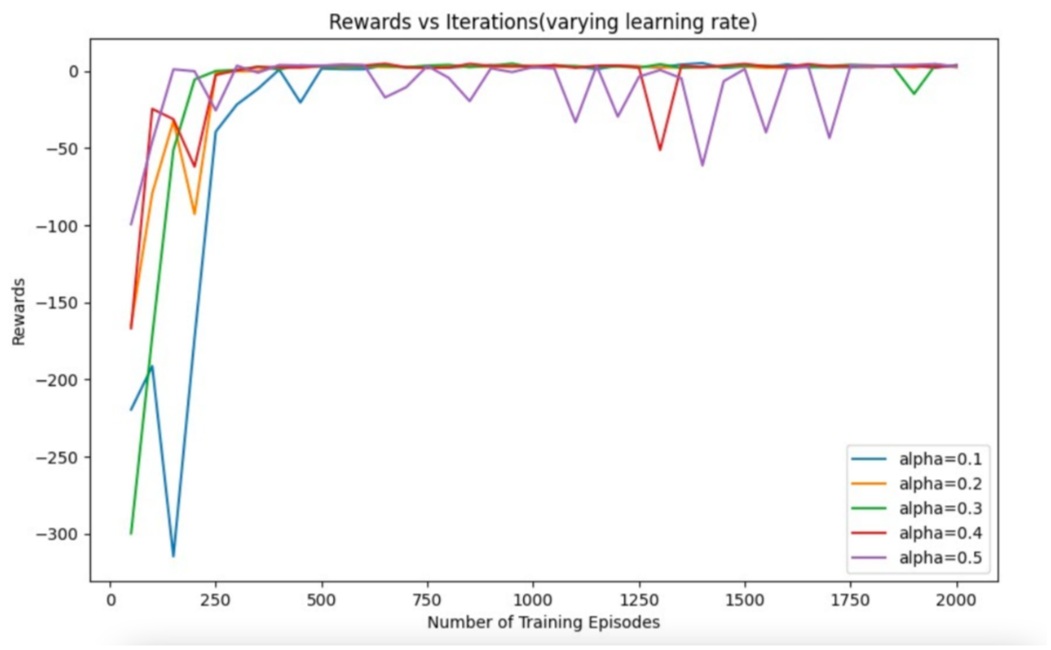
```
Start state: (0, 1, 2, 0, 0)
   Current State   Action       Next State   Reward
  (0, 1, 2, 0, 0)       S   (0, 2, 2, 0, 0)      -1
  (0, 2, 2, 0, 0)       W   (0, 2, 3, 0, 0)      -1
  (0, 2, 3, 0, 0)       W   (0, 2, 2, 0, 0)      -1
  (0, 2, 2, 0, 0)       W   (0, 2, 1, 0, 0)      -1
  (0, 2, 1, 0, 0)       W   (0, 2, 0, 0, 0)      -1
  (0, 2, 0, 0, 0)       N   (0, 2, 0, 0, 0)      -1
  (0, 2, 0, 0, 0)       N   (0, 2, 1, 0, 0)      -1
  (0, 2, 1, 0, 0)       W   (0, 2, 0, 0, 0)      -1
  (0, 2, 0, 0, 0)       N   (0, 1, 0, 0, 0)      -1
  (0, 1, 0, 0, 0)       N   (0, 1, 0, 0, 0)      -1
  (0, 1, 0, 0, 0)       N   (0, 0, 0, 0, 0)      -1
  (0, 0, 0, 0, 0)       U   (1, 0, 0, 0, 0)      -1
  (1, 0, 0, 0, 0)       S   (1, 1, 0, 1, 0)      -1
  (1, 1, 0, 1, 0)       S   (1, 2, 0, 2, 0)      -1
  (1, 2, 0, 2, 0)       S   (1, 3, 0, 3, 0)      -1
  (1, 3, 0, 3, 0)       S   (1, 4, 0, 4, 0)      -1
  (1, 4, 0, 4, 0)       D   (0, 4, 0, 4, 0)      20
Reward: 2.1749325313850747
```

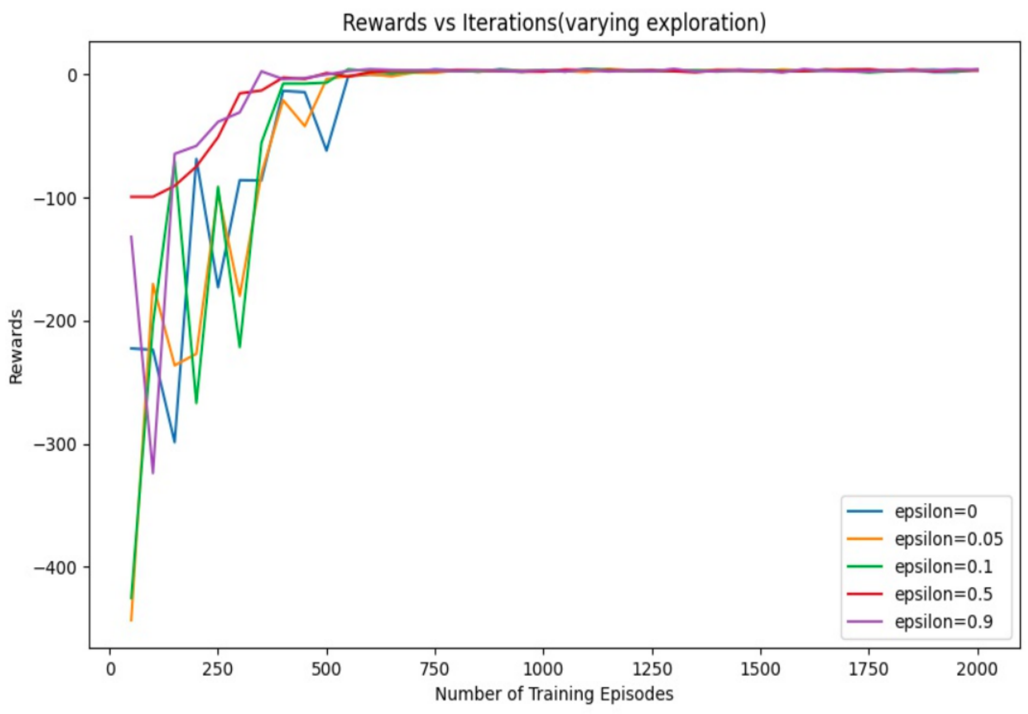
Taxi starts at (1,2)
Client is at (0,0)
Drop is at (4,0)

In each of the cases above, at each state, the action prescribed by the policy generated on learning is optimal.

4.) Varying exploration and learning rate

The following plots depict the effect of exploration rate and learning rate on the training process.





Observations
Varying learning rate: As the learning rate increases, initially convergence rate is increased. However, when the learning rate becomes too large, the Q values start oscillating as changes are very large and may be unfavorable due to the stochastic nature of transition which may cause the policy to deviate from optimal and will take a longer time to converge. For alpha=0.5, the policy learning is not complete till the end.

For alpha=0.3, convergence occurs faster than alpha=0.2, which in turn converges faster than alpha=0.1. alpha=0.4 converges really fast, but a disruption can be seen by one kink in the middle. alpha=0.5 fails to converge completely.

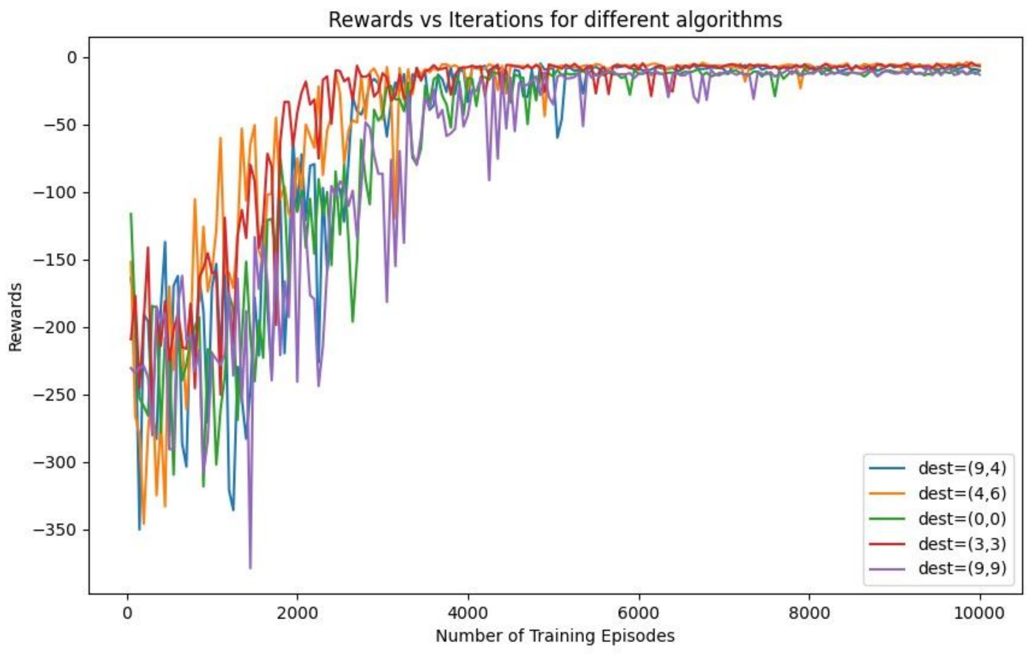
Varying exploration rate:
The general trend is that higher exploration rates cause faster policy convergence, however finding the exact Q-value for optimal action takes longer. This is because due to large exploration factor, all actions are explored evenly. However, the optimal action isn't used much during learning so Q-value convergence may take more time. Policy converges fastest for epsilon = 0.9 and 0.5. For the smaller exploration rates, some more training episodes are required before they arrive at the optimal policy(around 20-40% more training episodes).

5.) 10x10 extension

We use the algorithm written in part B.1.a) to solve a new MDP with 10x10 domain and 8 depots.

Here, we create 5 MDPs(with different destinations) and train each with 10000 episodes.

The discounted reward vs training can be seen in the plot below:



Destination	Average Discounted Reward(50 runs)
(9,4)	-10.103663858034825
(4,6)	-5.9108766953840215
(0,0)	-10.212668959551378
(3,3)	-7.119375314537936
(9,9)	-13.36873183091748

It can also be seen in the table as well as the plot that the rewards corresponding to some destination states are lower as compared to others. This can be explained in the following manner:
On average, corner depots would take larger number of steps to reach to, thus centrally located depots have higher rewards as compared to the depots located at the corners. (4,6) is centrally located thus on average would have lower number of steps to reach to, where as (9,9) is at the bottom right corner and has average reward = -13.37.

Also, convergence for each instance of the MDP(varying destination depot) takes roughly the same number of iterations but centrally located destinations tend to converge a little bit sooner than the edges and corners.