INDIAN INSTITUTE OF TECHNOLOGY, DELHI

ASSIGNMENT REPORT

Planning in the Taxi Domain

APAR AHUJA | ENTRY NO. 2019CS10465

ARNAV TULI | ENTRY NO. 2019CS10424

Course - COL333 | Prof. Rohan Paul

Compiled on November 27, 2021

Contents

Chapter 1

Part A: Computing Policies

1.1 Taxi Domain as a Markov Decision Process

In this section, we formulate the taxi domain problem as an instance of sequential-decision making process in stochastic environment, more specifically, as a *Markov Decision Process*. Below, we give specifications for a general $n \times n$ Taxi domain with a pre-defined but arbitrary set of walls and boundary.

1.1.1 MDP Description

- **Agent:** Taxi (or Taxi Driver)
- **Real-World Setting:** $n \times n$ *grid world* with cell $(0, 0)$ in bottom-left corner and cell $(n-1, n-1)$ in top-right corner when viewed from the top (coordinate system as given in assignment)
- **Walls:** Set of walls that prevent movement from one grid cell to another *adjacent* grid cell (*adjacent*-cell refers to a grid-cell one-unit in North, South, East or West direction)
- **Start:** Refers to Taxi's initial grid-cell location $(= (i, j), 0 \leq i, j < n)$
- **Depots:** Set of designated grid-cell locations
- **Source:** Refers to Passenger's initial grid-cell location $(= (i, j) \in \textit{Depots})$
- **Destination:** Refers to Passenger's destination grid-cell location $(= (i, j) \in \textit{Depots}))$
- **State Space:** Set of *five*-tuples of the form *(taxi_x, taxi_y, pass_x, pass_y, isPicked)*, where *(taxi_x, taxi_y)* is Taxi's grid-cell location, *(pass_x, pass_y)* is Passenger's grid-cell location and *isPicked* is Passenger's pickup status (boolean-valued). If *isPicked* is *True (1)*, then the Passenger is already picked up by the *Taxi*, otherwise, *Taxi* is yet to pick up the passenger. We specify problem-specific constraints that limit the state space below:
  - **World Constraint:** $0 \leq$ *taxi_x, taxi_y, pass_x, pass_y* $< n$; *isPicked* $\in \{0, 1\}$
  - **Pickup Constraint:** If *isPickup* is *1*, then *(taxi_x, taxi_y) = (pass_x, pass_y)*
  - **Termination Constraint:** This constraint is based on following two-assumptions:
    - *Source* $\neq$ *Destination*
    - Episode ends as soon as Taxi drops Passenger at *Destination*

    Therefore, if *(pass_x, pass_y) = Destination*, then *(taxi_x, taxi_y) = Destination*

  Hence, *State Space (S)* = {(taxi_x, taxi_y, pass_x, pass_y, isPicked): all constraints are satisfied}. For a 5×5 grid, the total number of states is **626**. $(24 \times 25 + 1 + 25)$
  Notation: Given a *state*, *s = (taxi_x, taxi_y, pass_x, pass_y, isPicked)*; we define: *taxi(s) = (taxi_x, taxi_y)*, *pass(s) = (pass_x, pass_y)* and *isPicked(s) = isPicked*.

- **Start State:** This state indicates beginning of an episode, and is given by: $(x_T, y_T, x_S, y_S, 0)$ (assuming $Start = (x_T, y_T)$ and $Source = (x_S, y_S)$)
- **Goal State:** This state indicates termination of an episode, and is given by: $(x_D, y_D, x_D, y_D, 0)$ (assuming $Destination = (x_D, y_D)$)
- **Action Space:** All states except *Goal State* have *six*-legal actions, $A = \{$North (N), South (S), East (E), West (W), Pickup (U), Putdown (D)$\}$. These actions are as defined in assignment. *Goal State* has no legal actions ($A_{Goal} = \phi$)
- **Transition Model:** We define the transition probability model $T(s, a, s')$ as follows - **Note:** $s \neq$ *Goal State*
  - **Pickup** $(a = U)$:
    - If *taxi(s) = taxi(s')* **and** *pass(s) = pass(s')*, then,
      - If *taxi(s) = pass(s)*, then since, *pickup* is a **deterministic** action, the passenger must be picked up (*isPicked(s') = 1*) in the final state, i.e. transition to a not-picked state (*isPicked(s') = 0*) is not possible. Thus, the transition probability is, $T(s, a, s')$ = *isPicked(s')*
      - If *taxi(s)* $\neq$ *pass(s)*, then from *Pickup Constraint*, *isPicked(s) = isPicked(s')* $= 0$. Thus, $s = s'$. Hence, the transition probability is, $T(s, a, s') = 1$.
    - If *taxi(s)* $\neq$ *taxi(s')* **or** *pass(s)* $\neq$ *pass(s')*, then the transition probability is, $T(s, a, s') = 0$, as the *pickup* action does not perform translation. (taxi or passenger)
  - **Putdown** $(a = D)$:
    - If *taxi(s) = taxi(s')* **and** *pass(s) = pass(s')*, then,
      - If *taxi(s) = pass(s)*, then since, *putdown* is a **deterministic** action, the passenger must be not-picked (*isPicked(s') = 0*) in the final state, i.e. transition to a picked-up state (*isPicked(s') = 1*) is not possible. Thus, the transition probability is, $T(s, a, s')$ = **not** *isPicked(s')*
      - If *taxi(s)* $\neq$ *pass(s)*, then from *Pickup Constraint*, *isPicked(s) = isPicked(s')* $= 0$. Thus, $s = s'$. Hence, the transition probability is, $T(s, a, s') = 1$.
    - If *taxi(s)* $\neq$ *taxi(s')* **or** *pass(s)* $\neq$ *pass(s')*, then the transition probability is, $T(s, a, s') = 0$, as the *putdown* action does not perform translation. (taxi or passenger)
  - **North, South, East, West** $(a = N, S, E, W)$
    - If *isPicked(s)* $\neq$ *isPicked(s')*, then the transition probability is, $T(s, a, s') = 0$ as *N, S, E, W* actions do not change *picked-up* status
    - If *isPicked(s) = isPicked(s')* $= 0$ **and** *pass(s)* $\neq$ *pass(s')*, then the transition probability is, $T(s, a, s') = 0$, as *N, S, E, W* actions do not perform translation of passenger, if not *picked-up*
    - In all other situations, let *taxi_ N*, *taxi_ S*, *taxi_ E* and *taxi_ W* denote the coordinates of *taxi* if it had moved *North*, *South*, *East* and *West* respectively, **deterministically**. Also define, $P(K) = 0.85$ if $a = K$, else 0.05. Then, the transition probability is, $T(s, a, s') = P(N)\mathbb{1}\{taxi(s') = taxi\_N\} + P(S)\mathbb{1}\{taxi(s') = taxi\_S\} + P(E)\mathbb{1}\{taxi(s') = taxi\_E\} + P(W)\mathbb{1}\{taxi(s') = taxi\_W\}$. Here, we have assumed the probability to end-up in some other direction to be *uniformly* equal to $\frac{0.15}{3} = 0.05$

- **Reward Model:** We assume that rewards are associated with transition arcs, i.e., (s, a, s'). We define the reward model for *likely* $(T(s, a, s') \neq 0)$ transitions below:
  Note: $s \neq$ *Goal State*
  - **Pickup** $(a = U)$:
    - If *taxi(s)* = *pass(s)*, then the reward is, $R(s, a, s')$ = **-1**
    - If *taxi(s)* $\neq$ *pass(s)*, then the reward is, $R(s, a, s')$ = **-10**
  - **Putdown** $(a = D)$:
    - If *taxi(s)* = *pass(s)*, then,
      - If s' = *Goal State*, then the reward is, $R(s, a, s')$ = **+20**
      - If $s' \neq$ *Goal State*, then the reward is, $R(s, a, s')$ = **-1**
    - If *taxi(s)* $\neq$ *pass(s)*, then the reward is, $R(s, a, s')$ = **-10**
  - **North, South, East, West** $(a = N, S, E, W)$: The reward is, $R(s, a, s')$ = **-1**

1.1.2 MDP Simulator

The function *A1()* simulates a given policy for MDP in question. Refer code for appropriate documentation related to *input* parameters and usage. **Figure 1.1** displays a sample simulation output.

```
Taxi starting at location: (3, 0)
Passenger (source) at location: (0, 0)
Passenger (destination) at location: (4, 4)
--------------------------------------------------------
Starting simulation... (Max. updates = 50)
Update 1: (3, 0, 0, 0, 0) * North -> (3, 1, 0, 0, 0)
Update 2: (3, 1, 0, 0, 0) * North -> (3, 2, 0, 0, 0)
Update 3: (3, 2, 0, 0, 0) * West -> (2, 2, 0, 0, 0)
Update 4: (2, 2, 0, 0, 0) * West -> (1, 2, 0, 0, 0)
Update 5: (1, 2, 0, 0, 0) * West -> (1, 1, 0, 0, 0)
Update 6: (1, 1, 0, 0, 0) * North -> (1, 2, 0, 0, 0)
Update 7: (1, 2, 0, 0, 0) * West -> (0, 2, 0, 0, 0)
Update 8: (0, 2, 0, 0, 0) * South -> (0, 1, 0, 0, 0)
Update 9: (0, 1, 0, 0, 0) * South -> (0, 1, 0, 0, 0)
Update 10: (0, 1, 0, 0, 0) * South -> (0, 0, 0, 0, 0)
Update 11: (0, 0, 0, 0, 0) * Pickup -> (0, 0, 0, 0, 1)
Update 12: (0, 0, 0, 0, 1) * North -> (0, 1, 0, 1, 1)
Update 13: (0, 1, 0, 1, 1) * North -> (0, 2, 0, 2, 1)
Update 14: (0, 2, 0, 2, 1) * East -> (1, 2, 1, 2, 1)
Update 15: (1, 2, 1, 2, 1) * East -> (2, 2, 2, 2, 1)
Update 16: (2, 2, 2, 2, 1) * North -> (3, 2, 3, 2, 1)
Update 17: (3, 2, 3, 2, 1) * North -> (3, 3, 3, 3, 1)
Update 18: (3, 3, 3, 3, 1) * North -> (4, 3, 4, 3, 1)
Update 19: (4, 3, 4, 3, 1) * North -> (4, 4, 4, 4, 1)
Update 20: (4, 4, 4, 4, 1) * Putdown -> (4, 4, 4, 4, 0)
Stopping simulation... Destination reached.
--------------------------------------------------------
Discounted Reward: -5.947444846981023
```

Figure 1.1: Simulation Output

1.2 Value Iteration for the Taxi Domain

The function *A2()* is the driver for this part. Refer code for appropriate documentation related to *input* parameters and usage.

1.2.1 Value Iteration Example

Instance Specifications:

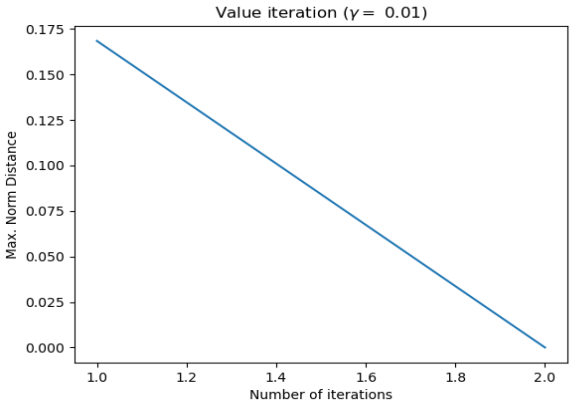
- $\epsilon = 0.01$
- Discount Factor, $\gamma = 0.9$
- Destination $= (4, 4)$
- Number of iterations required for convergence, $n = 35$

Figure 1.2 shows a sample output of value iteration in action. The output corresponds to the simulation displayed in **Figure 1.1**

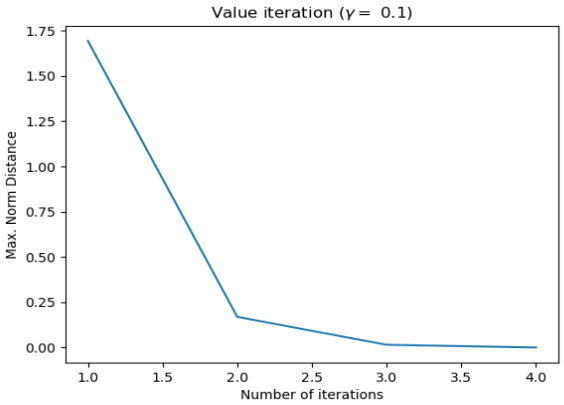
```
Iteration: 1, Max. Bellman Update: 20.0
Iteration: 2, Max. Bellman Update: 15.165
Iteration: 3, Max. Bellman Update: 13.6485
Iteration: 4, Max. Bellman Update: 10.982385
Iteration: 5, Max. Bellman Update: 8.8008433875
Iteration: 6, Max. Bellman Update: 7.069634143875
Iteration: 7, Max. Bellman Update: 5.585571293312812
Iteration: 8, Max. Bellman Update: 4.268299025752144
Iteration: 9, Max. Bellman Update: 3.402409000044856
Iteration: 10, Max. Bellman Update: 2.717209241136333
Iteration: 11, Max. Bellman Update: 2.1160334954752225
Iteration: 12, Max. Bellman Update: 1.6659692661607979
Iteration: 13, Max. Bellman Update: 1.2895109207731141
Iteration: 14, Max. Bellman Update: 1.0955970754256965
Iteration: 15, Max. Bellman Update: 0.9285043038155374
Iteration: 16, Max. Bellman Update: 0.7870364979715383
Iteration: 17, Max. Bellman Update: 0.6641553097714885
Iteration: 18, Max. Bellman Update: 0.5547716731165169
Iteration: 19, Max. Bellman Update: 0.4581155735393079
Iteration: 20, Max. Bellman Update: 0.3827726382549299
Iteration: 21, Max. Bellman Update: 0.2961077960698617
Iteration: 22, Max. Bellman Update: 0.25849588533024814
Iteration: 23, Max. Bellman Update: 0.1570729751344455
Iteration: 24, Max. Bellman Update: 0.14127440423727933
Iteration: 25, Max. Bellman Update: 0.08745590235566603
Iteration: 26, Max. Bellman Update: 0.06767380121472222
Iteration: 27, Max. Bellman Update: 0.0410448005843449
Iteration: 28, Max. Bellman Update: 0.028827178872758097
Iteration: 29, Max. Bellman Update: 0.017168337430181424
Iteration: 30, Max. Bellman Update: 0.011296590949078755
Iteration: 31, Max. Bellman Update: 0.006621083059635602
Iteration: 32, Max. Bellman Update: 0.0041589346542796335
Iteration: 33, Max. Bellman Update: 0.002403812577445973
Iteration: 34, Max. Bellman Update: 0.0014589027354556805
Iteration: 35, Max. Bellman Update: 0.0008330170557719896
Number of iterations: 35
```

Figure 1.2: Value Iteration Output

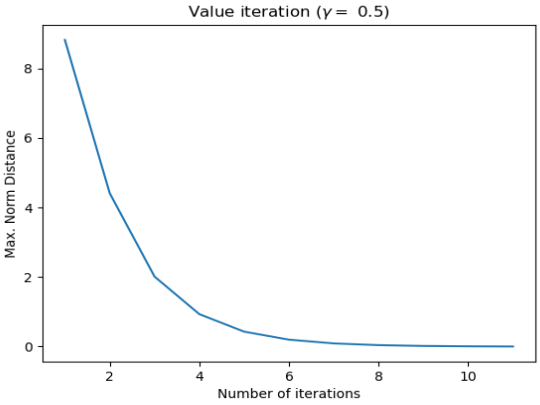
1.2.2 Value Iteration for different values of γ



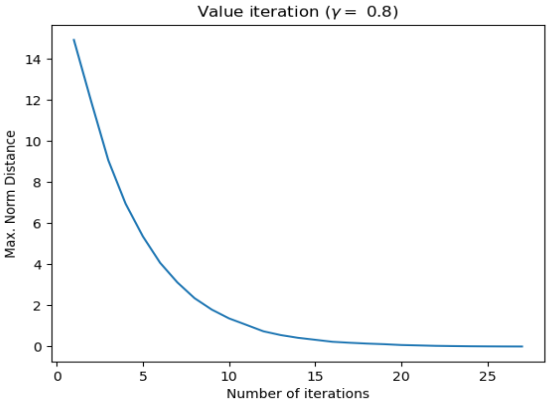
(a)



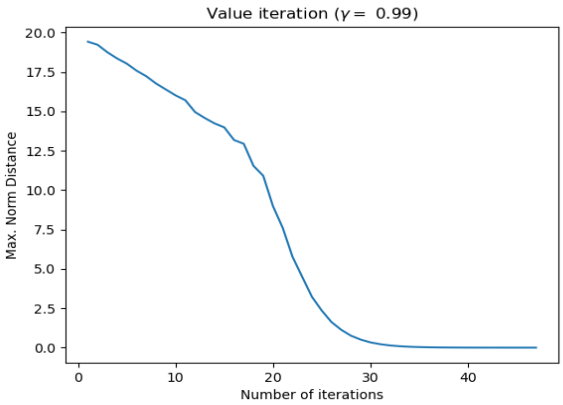
(b)



(c)



(d)



(e)

Figure 1.3: (a) $\gamma = 0.01$, (b) $\gamma = 0.1$, (c) $\gamma = 0.5$, (d) $\gamma = 0.8$, (e) $\gamma = 0.99$

Figure 1.3 shows variation of *max-norm* distance vs the number of iterations in *Value-Iteration* algorithm for different values of discount factor, γ. Here, *max-norm* at i^{th} iteration is defined to be $||V_i - V^*||_{max}$, where V_i is the vector of state values after iteration i, and V^* is the vector of optimal state values ($i \to \infty$). Since, taking i to ∞ is infeasible, we use the *converged* (on basis of ϵ) set of state values to be a proxy for V^*, for plotting purposes.

Instance Specifications:

- $\epsilon = 0.01$
- Destination $= (4, 4)$

Discount Factor (γ)	Number of Iterations (n)
0.01	2
0.1	4
0.5	11
0.8	27
0.99	47

Table 1.1: Rate of convergence of Value-Iteration Algorithm

Observations:

- The initial *max-norm* distance increases with increase in discount factor (γ).
  Plausible Reason: Higher discount factor means that the final reward of +20 is going to be propagated significantly to a greater number of states in the state space. Hence, final (or *optimal*) values of states is going to be a lot different than the initial immediate reward received by the agent ($|\frac{R}{1-\gamma} - R|$ increases with increasing γ). This in turn implies that there is a larger gap between initial and optimal values indicated by a larger initial *max-norm* distance.

- Number of iterations required for convergence of *Value-Iteration* algorithm increases with increase in discount factor (γ).
  Plausible Reason: Lower discount factor (γ) makes the agent act as a *reflex*-agent with more focus on *short-term* rewards only. Higher discount factor (γ) makes the agent more conscious of its action choices that are sequential in nature. Therefore, the agent acts in a way so as to maximise the *long-term* reward (sequence) over short term gains. Further, with higher discount factor (γ), the effect of change in value of any state propagates to more number of states in the state space. This is because of the exponentially decaying factor (γ) which is used to discount the reward sequence. Hence, more iterations are required to update the effect of any change on other states, and consequently, Value-Iteration takes more iterations to converge to the optimal values.

1.2.3 Simulation of Optimal Policies

In this section, we simulated the optimal policies for $\gamma = 0.1$ and $\gamma = 0.99$ on various *start-state* configurations. One set of simulation is displayed in **Figure 1.4**. Since, similar patterns were observed for different configurations, the outputs for other runs are omitted from report. They can be found in *output.txt* file.

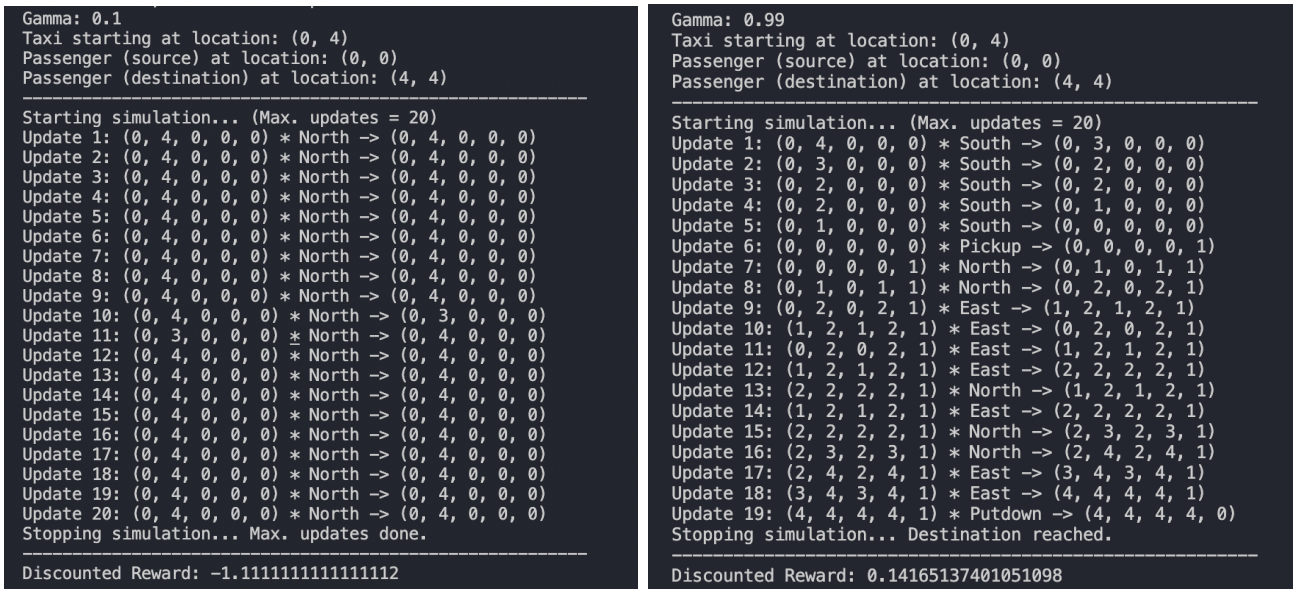


Figure 1.4: Taxi Start = 'R', Passenger Source = 'Y', Passenger Destination = 'G'

Observations:
These observations are general and apply to all 12 *start*-states on which we ran our simulation. These 12 states comprise of all possible start states such that *start, source* $\in$ *Depots* and *source* $\neq$ *destination*. Refer *output.txt* for corresponding state-action sequences.

- Only *one*-action is repeated by the agent with $\gamma = 0.1$. This is because for low discount factor, the agent acts more like a *reflex*-agent than a *goal*-based agent and tries to maximize short term rewards. As 0.1^n decreases exponentially with n, the rewards of reaching the destination don't propagate back to predecessor states significantly. Thus, the agent is blinded and chooses the first action (*North*) it finds and executes it. Note that the reward for executing any action is **-1** (some exceptions, refer *section 1.1.1*), and hence, the agent chooses the first action irrespective of actual source and destination locations.
- Agent with $\gamma = 0.99$ is more *goal*-based and acknowledges the *source* and *destination* locations. It first moves to (0, 0) to pick-up the passenger (Update 6). Then, it moves to (4, 4) (destination), where the passenger is put-down by the taxi (Update 19). Hence, the destination is reached. Therefore, the agent is learning well in the given domain. A plausible reason for this is that for high discount factors, the reward of completing the task (+20) propagates to a large number of states significantly, and consequently, the agent learns how to maximize long term sequential rewards, i.e. drop passenger at the destination.

- Particular Observation: $\gamma = 0.99$ has a positive discounted reward whereas $\gamma = 0.1$ has a negative discounted reward. This is because in the former case, the agent can see the long term reward (+20) in a significant amount. Hence, it learns that it is better to reach the destination and drop the passenger than simply moving *North*.

1.3 Policy Iteration for the Taxi domain

The function *A3()* is the driver for this part. Refer code for appropriate documentation related to *input* parameters and usage.

1.3.1 Implementation of Policy Iteration

Policy-Iteration algorithm consists of two parts: *Policy-Evaluation* and *Policy-Update*. Policy-Update is simple and can be directly performed using the obtained state utility values. However, there are two common ways in which Policy-Evaluation can be implemented. We mention these two methods below:

Iterative Algorithm

For a given policy, π, the modified Bellman Equations give a solution to the optimal values of state utilities, i.e., for all $s \in S$,

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma * V^{\pi}(s'))$$

This formulation gives way to a straight-forward dynamic-programming solution (iterative):

Algorithm 1 Policy Evaluation − Iterative

while maxBellmanUpdate $\geq \frac{\epsilon(1-\gamma)}{\gamma}$ **do**

  for all $s \in S$ **do**

    $\mathrm{V}^{\pi}_{new}(s) \leftarrow \sum_{s'} T(s, a, s')[R(s, a, s') \ + \ \gamma V^{\pi}_{old}(s')]$

Policy Evaluation using Linear Algebra

Define vectors/matrices as follows:
$V \in \mathbb{R}^{n \times 1}$: $V[i]$ = value of state s_i ($|S| = n$)
$P \in \mathbb{R}^{n \times n}$: $P[i][j] = \mathrm{T}(\mathrm{s}_i, \pi(\mathrm{a}), \mathrm{s}_j)$
$R \in \mathbb{R}^{n \times n}$: $R[i][j] = \mathrm{R}(\mathrm{s}_i, \pi(\mathrm{a}), \mathrm{s}_j)$
γ: Discount Factor
Using these matrices, the modified Bellman Equations (set of linear equations) can be expressed as follows:

$$V = diag(PR^T) + \gamma PV \implies (I - \gamma P)V = diag(PR^T) \implies \boldsymbol{V = (I - \gamma P)^{-1} diag(PR^T)}$$

Here, $I \in \mathbb{R}^{n \times n}$ is the identity matrix.

Algorithm 2 Policy Evaluation − Analytical

$V^{\pi} \leftarrow (I - \gamma P)^{-1} \text{diag}(PR^{T})$

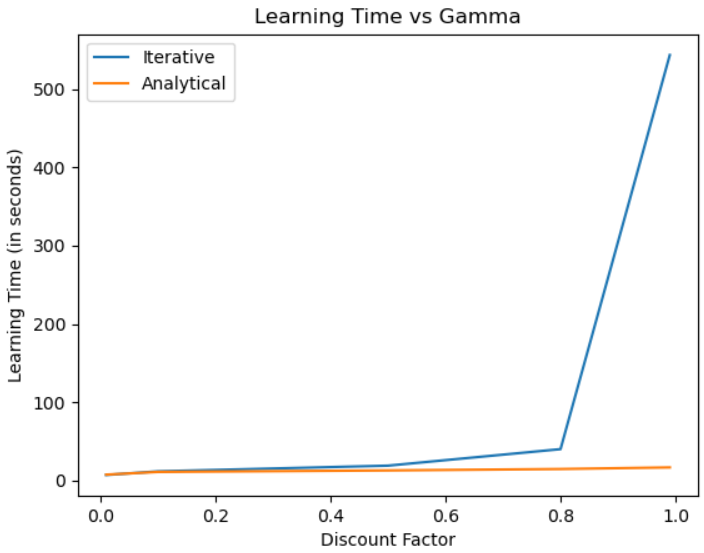
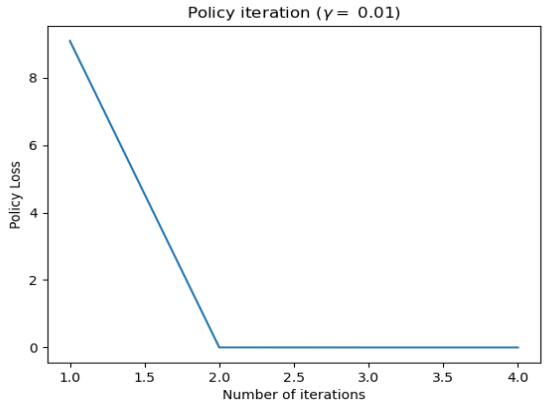


Figure 1.5: Policy Iteration − Iterative v/s Analytical

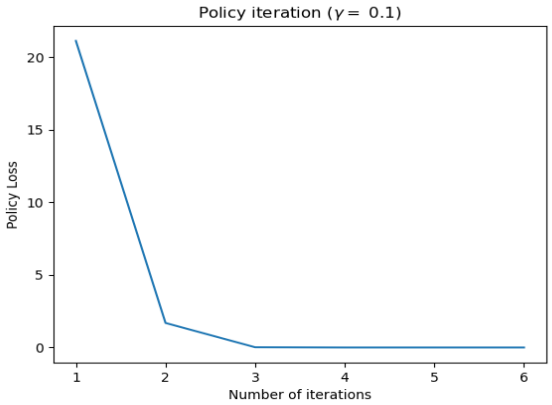
Analysis:

- For small state spaces (small n), analytical solution is faster due to the small matrix size. The computation cost is mostly because of the matrix inverse and multiplication ($O(n^3)$) which is small for small state spaces.
- For large state spaces (large n), the iterative algorithm takes the lead. The matrix inverse calculation becomes expensive ($O(n^3)$), whereas the iterative algorithm can still converge to a close approximation (of utility value) faster. Since, exact values of optimal utility are not required for determining optimal policy, the iterative version can be a good choice as far as *correctness* and *efficiency* are considered.
- The *Analytical*-algorithm's learning time doesn't change with the value of discount factor (γ). This is because the computation cost of matrix-inverse (and multiplication) is independent of γ. The learning time for *Iterative*-algorithm increases with increase in discount factor (γ). This is because for high discount factor the effect of change in value of any state propagates significantly to more number of states in the state space. This in turn is due to the exponential decay (γ^n) present in reward sequence. This means more number of iterations are required to update the effect of any change on other states. Hence, iterative method takes more time to converge. This effect on learning time can also be directly observed from **Figure 1.5**. Large n favours *Iterative*. Large γ favours *Analytical*.

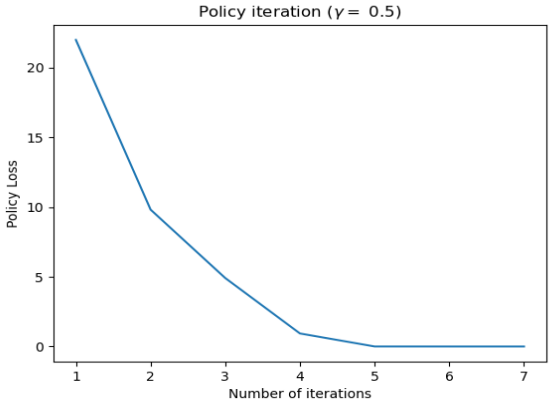
1.3.2 Policy Iteration for different values of γ



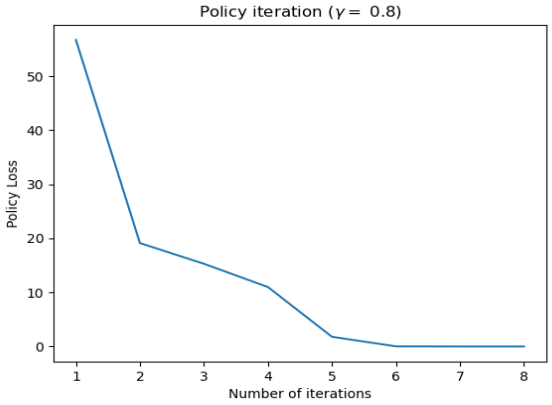
(a)



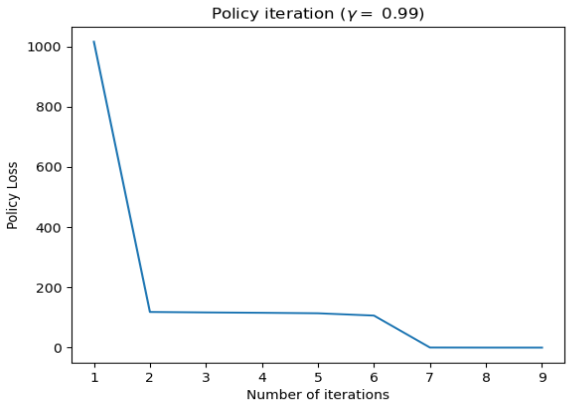
(b)



(c)



(d)



(e)

Figure 1.6: (a) $\gamma = 0.01$, (b) $\gamma = 0.1$, (c) $\gamma = 0.5$, (d) $\gamma = 0.8$, (e) $\gamma = 0.99$

Figure 1.6 shows variation of *policy-loss* vs the number of iterations in *Policy-Iteration* algorithm for different values of discount factor, γ. Here, *policy-loss* at i^{th} iteration is defined to be $||V^{\pi_i} - V^*||_{max}$, where V^{π_i} is the vector of state values after iteration i (with policy π_i), and V^* is the vector of optimal state values ($i \to \infty$). Since, taking i to ∞ is infeasible, we use the *converged* (on basis of ϵ) set of state values to be a proxy for V^*, for plotting purposes.

Instance Specifications:

- $\epsilon = 0.01$
- Destination $= (0, 4)$

Discount Factor (γ)	Number of Iterations (n)
0.01	3
0.1	5
0.5	6
0.8	7
0.99	8

Table 1.2: Rate of convergence of Policy-Iteration Algorithm

Observations:

- The initial *policy-loss* increases with increase in discount factor (γ).
  Plausible Reason: Higher discount factor means that the final reward of $+20$ is going to be propagated significantly to a greater number of states in the state space. Hence, final (or *optimal*) values of states is going to be a lot more different than the initial immediate reward received by the agent ($|\frac{R}{1-\gamma} - R|$ increases with increasing γ). This in turn implies that there is a larger gap between initial and optimal values indicated by a larger initial *policy-loss*.

- The number of iterations required for convergence of Policy-Iteration algorithm increases slightly with increase in value of discount factor. This can be directly observed from **Table 1.2**. A plausible reason is that with higher γ, the effect of change in value of any state propagates significantly to a larger number of states in the state space. This explains the slight increase in number of iterations of *Policy-Iteration* (updating policy of a state may affect many more states for high γ), and also explains the increase in learning time (associated with value iteration), which can be seen in **Figure 1.6**. This means more iterations are required to update the effect of any change on other states. Lower γ values make the agent act more as a *reflex*-agent with focus on short term rewards only. Higher γ values make the agent more conscious of its action choices, that are sequential in nature. Hence, it learns to acts in a way so as to maximize long term rewards over short term gains.

Chapter 2

Part B: Incorporating Learning

2.1 RL Approaches for the Taxi Domain

The functions *qLearning()* and *SARSA()* in the *squareGrid* class implement the corresponding algorithms. Refer code for appropriate documentation related to *input* parameters and usage.

2.2 Performance of different RL-Algorithms

Instance Specifications:

- Discount Factor, $\gamma = 0.99$; Initial Exploration Rate, $\epsilon = 0.1$
- Learning Rate, $\alpha = 0.25$; Destination $= (0, 4)$
- Number of Training Episodes $= 2000$ (max. 500 steps per episode)

2.2.1 Methodology

We trained *four* RL-algorithms for a maximum of 2000 training episodes. Every training episode starts from a randomly selected start state (refer *section 1.1.1*) and terminates when agent reaches the goal state or **500** steps have been executed by the agent. There are a total of **75** start states under our model (*3* possible depot locations for *source* and *25* possible locations for *start*). Note that the *destination* is fixed throughout the training process, so as to fix the optimal policy. After every 20 training episodes, we determine the optimal policy based on Q-values learnt till that point, and then evaluate the policy by means of *averaging*. The procedure used for averaging is described below:

- There are **75** start states in total. Create *80* episodes corresponding to each start state (a total of $75 * 80 =$ *6000* episodes). Note that *destination* is fixed throughout the episode generation process and is the same as that used during learning.
- Execute the optimal policy (learnt till that point) on each of these *6000* episodes (max. 50 steps) and determine the sum of discounted rewards (individually).
- Take the **mean** of the individual sum of discounted rewards (*6000* of them). This averaging acts as a proxy for $\mathbb{E}_{SS}[V(s)]$ ($s \in SS$), where SS is the set of all start states.
- We then plot this **mean** value against the number of training episodes as we train our agent further.

Note that the set of start states (infact state space) is the same for all the learning algorithms in our experiment. Hence, we can *expect* them to converge to the same $\mathbb{E}_{SS}[V(s)]$ value as training proceeds. However, since, our averaging process is over a finite set of episodes and the environment is *stochastic*, we can *expect* some minor deviations from the true optimum nevertheless.

2.2.2 Convergence of Algorithms

In this section, we display and analyse the convergence of the *mean* value for different RL-algorithms.

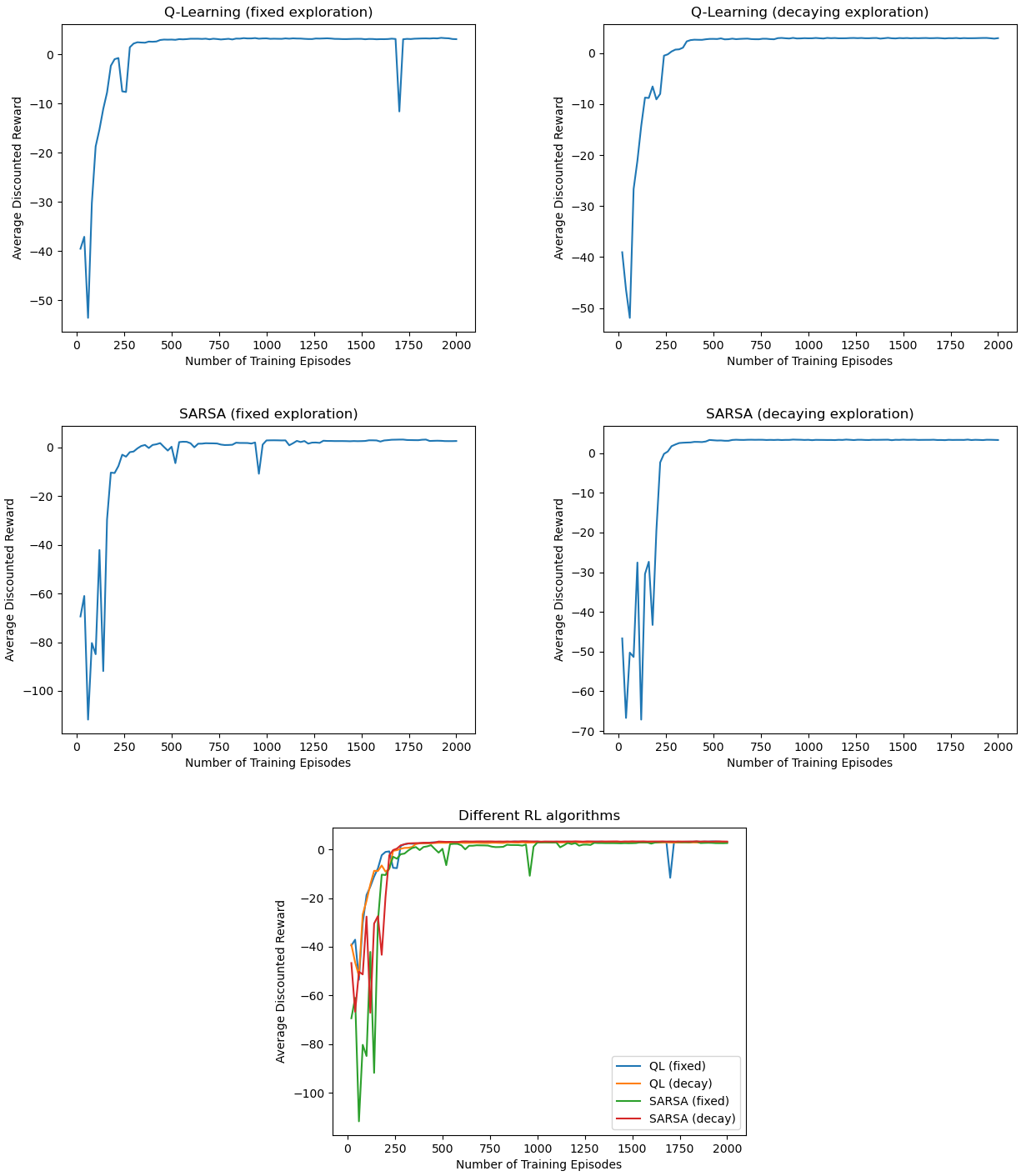


Figure 2.1: Average Discounted Reward ($\mathbb{E}_{SS}[V(s)]$) v/s Number of Training Episodes

Statistical Data: From **Figure 2.1**

Algorithm	Maximum Average Reward	Index
QL (fixed)	3.35557	95
QL (decay)	2.98121	70
SARSA (fixed)	3.25371	85
SARSA (decay)	3.39728	45

Here *Index* indicates the training episode when the maximum was obtained. More precisely, *Index* $= i$, corresponds to $20 * i$ training episodes.

Observations:

- All algorithms converge (optimal policy found) within 2000 epochs to an average discounted reward of around **3**.
- For the first *250* training episodes, the curves for all the RL-algorithms are rough. This indicates the *exploration*-phase of the algorithms. *SARSA* has more roughness possibly due to the fact that it updates its Q-values from actual data (on-policy) rather than greedy data (off-policy). Hence, it is more prone to making bad decisions, thereby, leading to roughness. After *250* episodes, QL (decay) and SARSA (decay) effectively enter the *exploitation*-phase (as exploration rate becomes very small). QL (fixed) and SARSA (fixed) have fixed exploration rate and keep on exploring even after they have determined the optimal policy. These sub-optimal decisions when combined with large learning rate (α) can lead to *spikes* in the graph, which can be observed from the graphs (**Figure 2.1**). Below, we will try to solve this issue by lowering down the learning rate for these two algorithms.
- QL (decay) has the smallest peak out of the four (2.98121). This can be due to two reasons: *stochasticity* and *insufficient exploration*. Since, the value is very close to the peak value of other algorithms (+ 0.3), it is unlikely that the agent only suffers from insufficient exploration. Rather, it is a combination of both factors with higher weight to *stochasticity*.
- SARSA (decay) on the other hand has the highest peak (3.39). Even though the exploration rate is decaying with each Q-value update, SARSA is able to explore well in the first *250* training episodes as it learns from more practical data, when compared to Q-Learning. This can be seen in the form of roughness in its graph (**Figure 2.1**)
- SARSA (decay) the highest maximum average discounted reward (3.39). It also reaches its peak value the fastest (Index=45) when compared to other algorithms. Hence, SARSA (decay) will be our choice for *best* RL-algorithm given the evidence.

Remedy for Fixed Exploration Algorithms: We reduced the learning rate for QL (fixed) and SARSA (fixed) to 0.1 (from 0.25), and trained them for 2000 episodes (other instance specifications are the same as before). We display plots obtained on the following page (**Figure 2.2**).
Observations:

- Low learning rate makes the learning process slower and *exploration*-phase exists for a longer duration (750-1000 episodes as opposed to 250 episodes before).
- However, lower learning effect reduces the effect of sub-optimal actions after the optimal policy has been found, thereby, smoothing the curve in the *exploitation*-phase. Previously, due to large learning rate, there were some *spikes* even after the policy had converged.

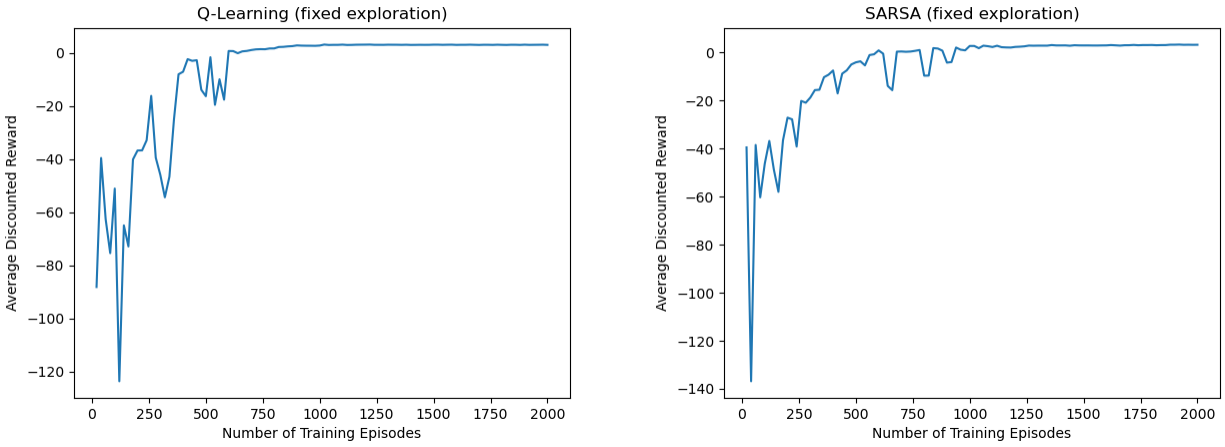


Figure 2.2: Average Discounted Reward ($\mathbb{E}_{SS}[V(s)]$) v/s Number of Training Episodes

2.3 Optimal Policy and Start State

From *section 2.2*, we concluded that SARSA (decay) was the best RL-algorithm among the four given the evidence (highest and quickest). Hence, we chose to run SARSA (decay) on 5 randomly selected instances by varying the Taxi *start* and Passenger *source* location. *Destination* location remains fixed. Refer **Figure 2.3** for simulation output.

Observations:

- The agent reaches the goal-state (passenger reaches the destination) within 23 steps in all 5 configurations. This shows that the agent's training was successful (optimal or *near*-optimal policy found).
- Since the start location of the Taxi was varied in the configurations, it shows that the learnt policy is independent of the Taxi start location.
- Similarly the source location of the passenger was varied in the configurations, which shows that the learnt policy is independent of the passenger source location.
- From the above two points, we can conclude that the learnt policy (optimal-policy) is independent of the start-state, which is defined by the Taxi start and Passenger source locations. Hence, the optimal-policy only depends on the goal-state. (given a Taxi domain and discount factor)

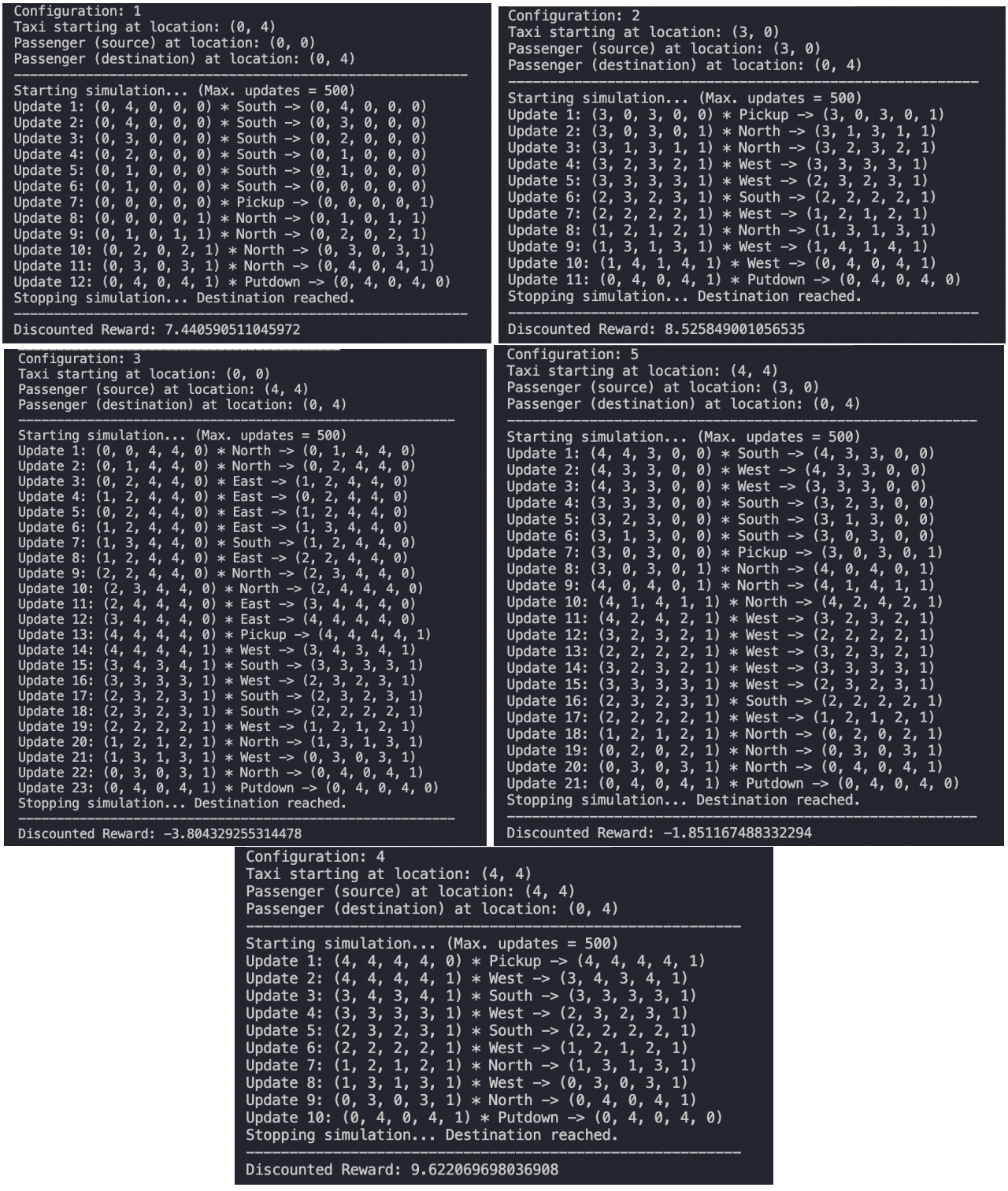


Figure 2.3: Simulation of learnt policy (SARSA (decay)) on five instances

2.4 Effect of Exploration and Learning Rate

Methodology used for averaging and determining Average Discounted Reward is same as that described in *section 2.2.1*. The only difference being that the averaging is performed after every **50** episodes rather than after every *20*.

2.4.1 Q-Learning for different values of ϵ

Instance Specifications:

- Algorithm: *QL (fixed)*
- Discount Factor, $\gamma = 0.99$
- Learning Rate, $\alpha = 0.1$
- Destination = $(0, 0)$
- Number of Training Episodes = 4000 (max. 500 steps per episode)

Statistical Data: From **Figure 2.4**

Exploration Rate	Maximum Average Discounted Reward	Index
0	3.34540	69
0.05	3.27300	73
0.1	3.28109	48
0.5	3.42735	51
0.9	3.41368	44

Here, *Index* is as defined in *section 2.2.2*

2.4.2 Q-Learning for different values of α

Instance Specifications:

- Algorithm: *QL (fixed)*
- Discount Factor, $\gamma = 0.99$
- Exploration Rate, $\epsilon = 0.1$
- Destination = $(0, 0)$
- Number of Training Episodes = 4000 (max. 500 steps per episode)

Statistical Data: From **Figure 2.5**

Learning Rate	Maximum Average Discounted Reward	Index
0.1	3.36063	54
0.2	3.41635	43
0.3	3.25809	62
0.4	3.34546	36
0.5	3.32564	39

Here, *Index* is as defined in *section 2.2.2*

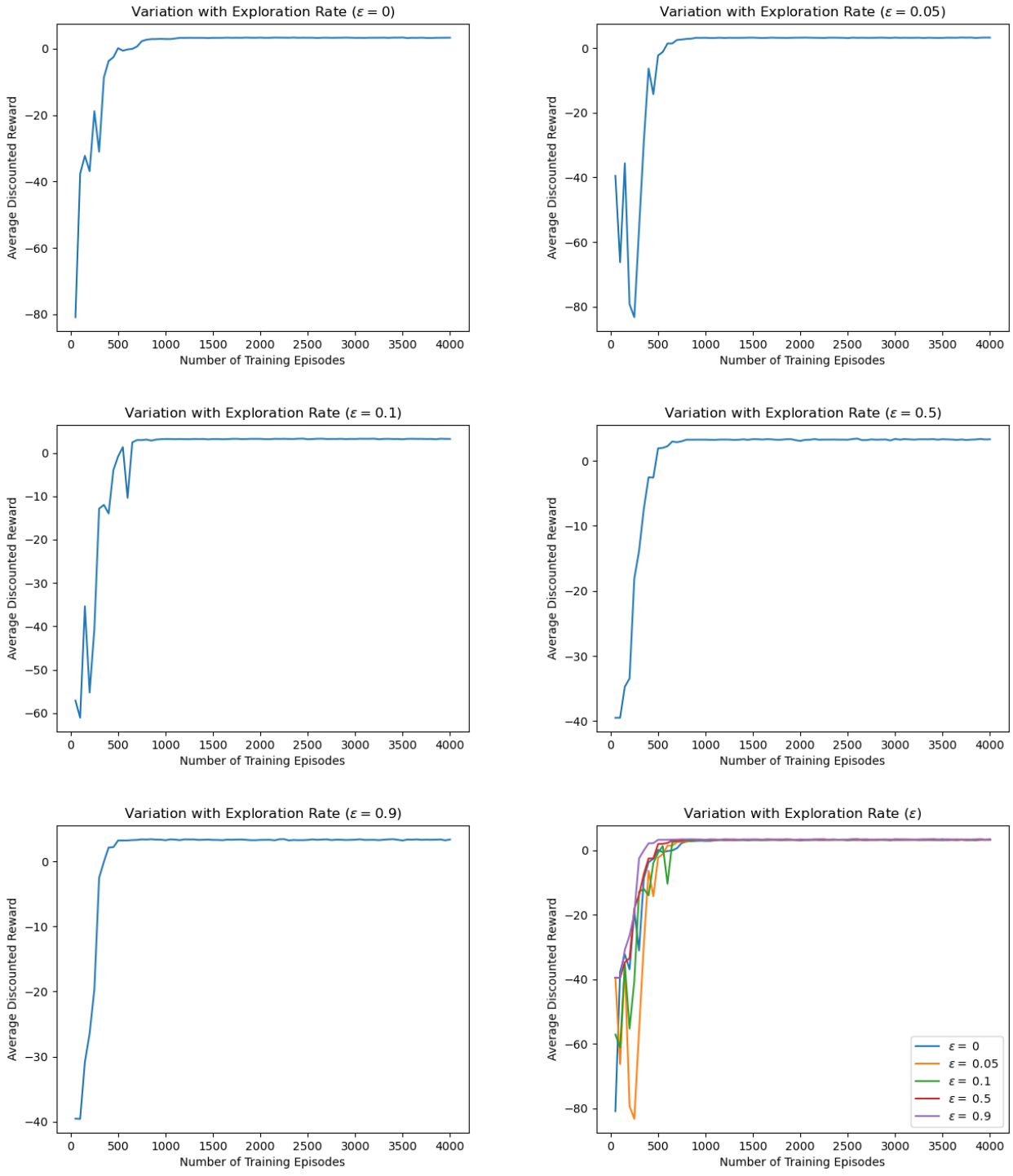


Figure 2.4: Average Discounted Reward ($\mathbb{E}_{SS}[V(s)]$) v/s Number of Training Episodes

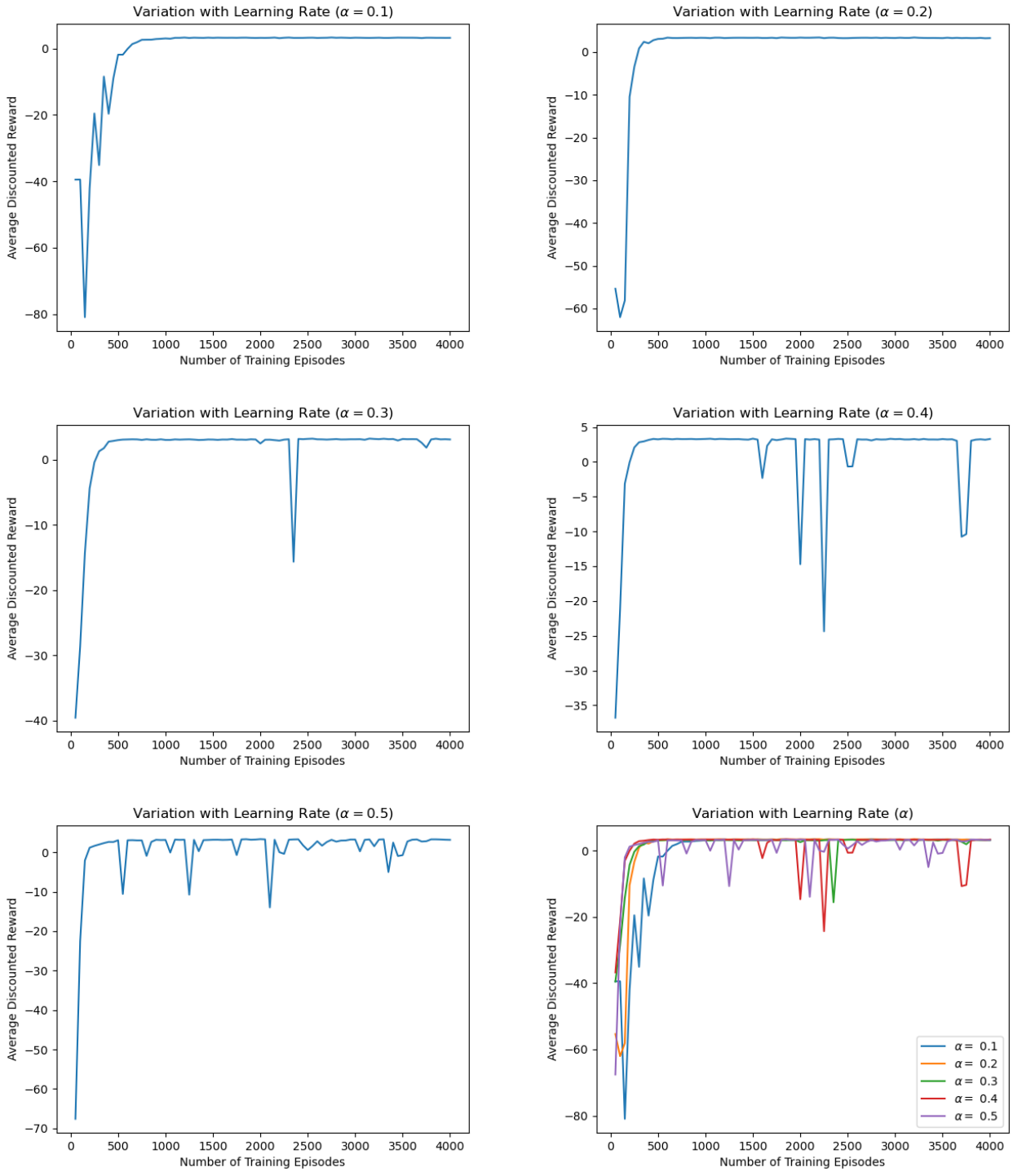


Figure 2.5: Average Discounted Reward ($\mathbb{E}_{SS}[V(s)]$) v/s Number of Training Episodes

Observations:

- As the exploration rate is increased, the average discounted reward rises to a much higher value (3.42 ($\epsilon = 0.5$) and 3.41 ($\epsilon = 0.9$)). This is possibly because, more exploration allows the agent the learn more about the environment. Hence, the agent is able to better optimize its average utility value. Also, the peak value is observed much faster for high exploration rates (Index=51 ($\epsilon = 0.5$), Index=44 ($\epsilon = 0.9$)) as compared to lower ones. Again, this can be explained using the increased tendency of agent to explore the environment. This allows the agent to learn about the environment quicker as compared to the case, where its not exploring much.
- We conclude that the best exploration rate is $\boldsymbol{\epsilon = 0.5}$ (given the evidence), as it reaches the highest average reward (3.42) and that too sufficiently fast (Index=51 (min. = 44)). Hence, there is a good exploration-exploitation trade-off when $\epsilon = 0.5$
- As the learning rate is increased, the agent learns more from every experience that it receives (more weight). Hence, $\alpha = 0.4$ and $\alpha = 0.5$ reach their maximum value at Index=36 and Index=39 respectively, which is a lot less than the indices for other learning rate values. Hence, learning is faster for a given exploration rate. However, as the agent is using fixed exploration rate, this increased learning rate is actually bad during the *exploitation*-phase. This leads to significant *spikes* in the graphs of $\alpha = 0.4$ and $\alpha = 0.5$. This issue has also been discussed in *section 2.2.2*. Thus, this spiking can reduce the algorithms tendency to reach high average utility values (*overshooting*). Hence, there is a trade-off between high and low learning rates: High Learning Rate $\implies$ Fast Exploration but *Spiky* exploitation; Low Learning Rate $\implies$ Slow Exploration ($\alpha = 0.1$) but *Smooth* exploitation.
- We conclude that the best value of learning rate is $\boldsymbol{\alpha = 0.2}$, which provides the highest average utility (3.41) and decent rate of convergence (Index=43).

2.5 10 × 10 Taxi Domain Problem

We modelled the 10 × 10 grid as an MDP using our formulation (*section 1.1.1*). The list of depots and wall locations is as specified in the assignment. The state space now consists of **10001** (99 × 100 + 1 + 100) states (including 1 goal-state). Also, the number of start states (for a given destination) is **700** (7 × 100).
We chose **5** destination locations randomly and learnt optimal policies for each one of them.
Algorithm Specifications:

- Algorithm: SARSA (decay)
- Discount Factor, $\gamma = 0.99$
- Exploration Rate, $\epsilon = 0.5$
- Learning Rate, $\alpha = 0.2$
- Number of Training Episodes = 10000 (max. 2000 steps per episode)

We then evaluated our policy (for each destination) using the averaging method described in *section 2.2.1*, with suitable modifications to account for 10 × 10 grid. Rather than setting the max. step limit to *50*, we set it to **200** (larger grid). Number of episodes for a given start state was still the same, i.e. **80**, but the number of start states themselves, was much larger (around 700).

2.5.1 Destination Cell = (0, 9)

Average Utility ($\mathbb{E}_{SS}[V(s)]$) = −11.30146

```
Taxi starting at location: (0, 0)
Passenger (source) at location: (0, 1)
Passenger (destination) at location: (0, 9)
------------------------------------------------------------
Starting simulation... (Max. updates = 200)
Update 1: (0, 0, 0, 1, 0) * North -> (0, 0, 0, 1, 0)
Update 2: (0, 0, 0, 1, 0) * North -> (0, 1, 0, 1, 0)
Update 3: (0, 1, 0, 1, 0) * Pickup -> (0, 1, 0, 1, 1)
Update 4: (0, 1, 0, 1, 1) * North -> (0, 2, 0, 2, 1)
Update 5: (0, 2, 0, 2, 1) * North -> (0, 3, 0, 3, 1)
Update 6: (0, 3, 0, 3, 1) * North -> (0, 4, 0, 4, 1)
Update 7: (0, 4, 0, 4, 1) * North -> (0, 5, 0, 5, 1)
Update 8: (0, 5, 0, 5, 1) * North -> (0, 6, 0, 6, 1)
Update 9: (0, 6, 0, 6, 1) * North -> (0, 7, 0, 7, 1)
Update 10: (0, 7, 0, 7, 1) * North -> (0, 8, 0, 8, 1)
Update 11: (0, 8, 0, 8, 1) * North -> (0, 9, 0, 9, 1)
Update 12: (0, 9, 0, 9, 1) * Putdown -> (0, 9, 0, 9, 0)
Stopping simulation... Destination reached.
------------------------------------------------------------
Discounted Reward: 7.440590511045972
Average Utility: -11.301455769696686
```

Figure 2.6: Simulation Output Example

2.5.2 Destination Cell = (4, 0)

Average Utility ($\mathbb{E}_{SS}[V(s)]$) = −8.42864

```
Taxi starting at location: (0, 0)
Passenger (source) at location: (0, 1)
Passenger (destination) at location: (4, 0)
------------------------------------------------------------
Starting simulation... (Max. updates = 200)
Update 1: (0, 0, 0, 1, 0) * North -> (0, 1, 0, 1, 0)
Update 2: (0, 1, 0, 1, 0) * Pickup -> (0, 1, 0, 1, 1)
Update 3: (0, 1, 0, 1, 1) * North -> (0, 2, 0, 2, 1)
Update 4: (0, 2, 0, 2, 1) * North -> (0, 3, 0, 3, 1)
Update 5: (0, 3, 0, 3, 1) * North -> (0, 4, 0, 4, 1)
Update 6: (0, 4, 0, 4, 1) * East -> (1, 4, 1, 4, 1)
Update 7: (1, 4, 1, 4, 1) * East -> (2, 4, 2, 4, 1)
Update 8: (2, 4, 2, 4, 1) * East -> (3, 4, 3, 4, 1)
Update 9: (3, 4, 3, 4, 1) * East -> (4, 4, 4, 4, 1)
Update 10: (4, 4, 4, 4, 1) * South -> (4, 3, 4, 3, 1)
Update 11: (4, 3, 4, 3, 1) * South -> (4, 2, 4, 2, 1)
Update 12: (4, 2, 4, 2, 1) * South -> (4, 1, 4, 1, 1)
Update 13: (4, 1, 4, 1, 1) * South -> (4, 0, 4, 0, 1)
Update 14: (4, 0, 4, 0, 1) * Putdown -> (4, 0, 4, 0, 0)
Stopping simulation... Destination reached.
------------------------------------------------------------
Discounted Reward: 5.302522759876155
Average Utility: -8.428644016716525
```

Figure 2.7: Simulation Output Example

2.5.3 Destination Cell = (3, 6)

Average Utility ($\mathbb{E}_{SS}[V(s)]$) = −6.83443

```
Taxi starting at location: (0, 0)
Passenger (source) at location: (0, 1)
Passenger (destination) at location: (3, 6)
------------------------------------------------------------
Starting simulation... (Max. updates = 200)
Update 1: (0, 0, 0, 1, 0) * North -> (0, 1, 0, 1, 0)
Update 2: (0, 1, 0, 1, 0) * Pickup -> (0, 1, 0, 1, 1)
Update 3: (0, 1, 0, 1, 1) * North -> (0, 2, 0, 2, 1)
Update 4: (0, 2, 0, 2, 1) * North -> (0, 3, 0, 3, 1)
Update 5: (0, 3, 0, 3, 1) * North -> (0, 4, 0, 4, 1)
Update 6: (0, 4, 0, 4, 1) * East -> (1, 4, 1, 4, 1)
Update 7: (1, 4, 1, 4, 1) * East -> (2, 4, 2, 4, 1)
Update 8: (2, 4, 2, 4, 1) * East -> (3, 4, 3, 4, 1)
Update 9: (3, 4, 3, 4, 1) * North -> (3, 5, 3, 5, 1)
Update 10: (3, 5, 3, 5, 1) * North -> (3, 6, 3, 6, 1)
Update 11: (3, 6, 3, 6, 1) * Putdown -> (3, 6, 3, 6, 0)
Stopping simulation... Destination reached.
------------------------------------------------------------
Discounted Reward: 8.525849001056535
Average Utility: -6.834432034955708
```

Figure 2.8: Simulation Output Example

2.5.4 Destination Cell = (0, 1)

Average Utility ($\mathbb{E}_{SS}[V(s)]$) = −10.57963

```
Taxi starting at location: (0, 0)
Passenger (source) at location: (0, 9)
Passenger (destination) at location: (0, 1)
--------------------------------------------------------------------------------
Starting simulation... (Max. updates = 200)
Update 1: (0, 0, 0, 9, 0) * North -> (0, 1, 0, 9, 0)
Update 2: (0, 1, 0, 9, 0) * North -> (0, 2, 0, 9, 0)
Update 3: (0, 2, 0, 9, 0) * North -> (0, 3, 0, 9, 0)
Update 4: (0, 3, 0, 9, 0) * North -> (0, 4, 0, 9, 0)
Update 5: (0, 4, 0, 9, 0) * North -> (0, 5, 0, 9, 0)
Update 6: (0, 5, 0, 9, 0) * North -> (0, 6, 0, 9, 0)
Update 7: (0, 6, 0, 9, 0) * North -> (0, 7, 0, 9, 0)
Update 8: (0, 7, 0, 9, 0) * North -> (0, 8, 0, 9, 0)
Update 9: (0, 8, 0, 9, 0) * North -> (0, 9, 0, 9, 0)
Update 10: (0, 9, 0, 9, 0) * Pickup -> (0, 9, 0, 9, 1)
Update 11: (0, 9, 0, 9, 1) * South -> (0, 8, 0, 8, 1)
Update 12: (0, 8, 0, 8, 1) * South -> (0, 7, 0, 7, 1)
Update 13: (0, 7, 0, 7, 1) * South -> (0, 6, 0, 6, 1)
Update 14: (0, 6, 0, 6, 1) * South -> (0, 5, 0, 5, 1)
Update 15: (0, 5, 0, 5, 1) * South -> (0, 4, 0, 4, 1)
Update 16: (0, 4, 0, 4, 1) * South -> (0, 3, 0, 3, 1)
Update 17: (0, 3, 0, 3, 1) * South -> (0, 3, 0, 3, 1)
Update 18: (0, 3, 0, 3, 1) * South -> (0, 2, 0, 2, 1)
Update 19: (0, 2, 0, 2, 1) * South -> (0, 1, 0, 1, 1)
Update 20: (0, 1, 0, 1, 1) * Putdown -> (0, 1, 0, 1, 0)
Stopping simulation... Destination reached.
--------------------------------------------------------------------------------
Discounted Reward: -0.8597651397295927
Average Utility: -10.579632132173698
```

Figure 2.9: Simulation Output Example

2.5.5 Destination Cell = (8, 9)

Average Utility ($\mathbb{E}_{SS}[V(s)]$) = −13.35729

```
Taxi starting at location: (0, 0)
Passenger (source) at location: (0, 1)
Passenger (destination) at location: (8, 9)
--------------------------------------------------------------------------------
Starting simulation... (Max. updates = 200)
Update 1: (0, 0, 0, 1, 0) * North -> (0, 1, 0, 1, 0)
Update 2: (0, 1, 0, 1, 0) * Pickup -> (0, 1, 0, 1, 1)
Update 3: (0, 1, 0, 1, 1) * North -> (0, 2, 0, 2, 1)
Update 4: (0, 2, 0, 2, 1) * North -> (0, 3, 0, 3, 1)
Update 5: (0, 3, 0, 3, 1) * North -> (0, 4, 0, 4, 1)
Update 6: (0, 4, 0, 4, 1) * East -> (0, 4, 0, 4, 1)
Update 7: (0, 4, 0, 4, 1) * East -> (1, 4, 1, 4, 1)
Update 8: (1, 4, 1, 4, 1) * East -> (0, 4, 0, 4, 1)
Update 9: (0, 4, 0, 4, 1) * East -> (1, 4, 1, 4, 1)
Update 10: (1, 4, 1, 4, 1) * East -> (2, 4, 2, 4, 1)
Update 11: (2, 4, 2, 4, 1) * East -> (3, 4, 3, 4, 1)
Update 12: (3, 4, 3, 4, 1) * East -> (4, 4, 4, 4, 1)
Update 13: (4, 4, 4, 4, 1) * East -> (5, 4, 5, 4, 1)
Update 14: (5, 4, 5, 4, 1) * South -> (5, 3, 5, 3, 1)
Update 15: (5, 3, 5, 3, 1) * East -> (6, 3, 6, 3, 1)
Update 16: (6, 3, 6, 3, 1) * East -> (7, 3, 7, 3, 1)
Update 17: (7, 3, 7, 3, 1) * North -> (7, 4, 7, 4, 1)
Update 18: (7, 4, 7, 4, 1) * North -> (6, 4, 6, 4, 1)
Update 19: (6, 4, 6, 4, 1) * East -> (7, 4, 7, 4, 1)
Update 20: (7, 4, 7, 4, 1) * North -> (7, 3, 7, 3, 1)
Update 21: (7, 3, 7, 3, 1) * North -> (7, 4, 7, 4, 1)
Update 22: (7, 4, 7, 4, 1) * North -> (7, 5, 7, 5, 1)
Update 23: (7, 5, 7, 5, 1) * East -> (8, 5, 8, 5, 1)
Update 24: (8, 5, 8, 5, 1) * North -> (8, 6, 8, 6, 1)
Update 25: (8, 6, 8, 6, 1) * North -> (8, 5, 8, 5, 1)
Update 26: (8, 5, 8, 5, 1) * North -> (9, 5, 9, 5, 1)
Update 27: (9, 5, 9, 5, 1) * West -> (8, 5, 8, 5, 1)
Update 28: (8, 5, 8, 5, 1) * North -> (8, 6, 8, 6, 1)
Update 29: (8, 6, 8, 6, 1) * North -> (8, 5, 8, 5, 1)
Update 30: (8, 5, 8, 5, 1) * North -> (8, 6, 8, 6, 1)
Update 31: (8, 6, 8, 6, 1) * North -> (8, 7, 8, 7, 1)
Update 32: (8, 7, 8, 7, 1) * North -> (8, 8, 8, 8, 1)
Update 33: (8, 8, 8, 8, 1) * North -> (8, 9, 8, 9, 1)
Update 34: (8, 9, 8, 9, 1) * Putdown -> (8, 9, 8, 9, 0)
Stopping simulation... Destination reached.
--------------------------------------------------------------------------------
Discounted Reward: -13.872336088206978
Average Utility: -13.357289910492675
```

Figure 2.10: Simulation Output Example

Observations:

- Global Average Utility (average of 5 utilities) = -10.10029
- The agent reached the goal-state in all instances (**Figure 2.6** to **2.10**), and hence, the learning algorithm (SARSA (decay)) was successful.
- The average utility was negative in all the cases (and hence, global average). This is probably because of the increased distances that the Taxi has to travel before reaching the goal-state (breathing cost is negative, i.e., -1)