Reinforcement Learning

Final Exam Problem 1, 2, 4, and Project Comprehensive Report

Hartaj Oberoi, Pragya Thakur

Rutgers Business School

Final Exam Problem 1, 2, 4

Problem 1: Basic Theory Understanding
(a) Definitions and Distinctions
(i) Explain and compare these terms:
1. Discounted vs. Average Reward MDPs: While average-reward MDPs aim to maximize the long-run average reward per time step, the discounted MDPs prioritize immediate rewards using a discount factor $\gamma \in (0, 1)$. The average is appropriate for ongoing tasks, while discounted settings favor finite horizons.

- Discounted: Prioritizes instant rewards using a discount factor $\gamma \in (0,1)$.
- Average: Perfect for ongoing tasks, it focuses on the long-run average reward each time step.

2. Model-Based vs. Model-Free RL: Model-Based RL (e.g., Policy Iteration) requires established transition models, whereas Model-Free RL (such as Q-Learning) takes up knowledge from its surroundings. Exchanges without being aware of P or R. Model-free is flexible in unknown territory.

- Model-Based: assumes that transition/reward models are accessible (such as Policy Iteration).
- Model-Free: gains knowledge from experience without being aware of the dynamics of MDP(such as Q-learning).

3. Q-Learning vs. R-Learning: R-Learning is used for usual reward settings, while Q-Learning is used for discounted rewards. R-Learning is better suited for continuous tasks without a natural γ, and records both relative value h(x) and average reward g.

Q-Learning: Q-values are directly estimated for discounted MDPs.

R-Learning: Gain, and relative Q-values are estimated in this average reward design.

(ii) Importance of the Unichain Assumption: All policies are guaranteed to generate a single recurrent class by the Unichain assumption. This enables algorithms like R-Learning or RVI to converge because it ensures that the average reward g^* remains constant throughout all states.

- Ensures that a Markov chain with a single recurrent class is the outcome of each policy.
- This makes sure algorithms such as Relative Value Iteration or R-learning converge to the right average reward.

(b) Dynamic Programming (DP) Equations
(i) Discounted Reward (Q-form):

$$Q^*(x,a) = r(x,a) + \gamma \sum_{x'} P(x'|x,a) \max_{a'} Q^*(x',a')$$

(ii) Average Reward DP Equation:

$$g^* + h^*(x) = \max_a \left[r(x,a) + \sum_{x'} P(x'|x,a) h^*(x') \right]$$

(c) Algorithmic Steps
Relative Value Iteration (RVI): *g,* and h(x) are iteratively changed using a fixed bias reference. Every h(x) update depends on transition values, and *g*.

Policy Iteration: alternates between implementing V^π to enhance π and evaluating V^π for the existing policy π. Discounted MDPs show fast convergence.

LP for First Passage MDPs: LP utilizes constraints that represent the value of each state to solve for displayed cost or reward until a desired state is achieved.

- Relative Value Iteration (RVI): helps with convergence in average-reward problems by subtracting the value of a reference state at each iteration.
- Policy Iteration: changes between assessing, and enhancing a policy.
- LP for First Passage: determines the estimated cost or benefit of arriving at a particular state using linear programming.

Problem 2: Basic Supply Chain MDP
(a) MDP Formulation
States: S = {0, 1, 2} (inventory levels)
Actions: A = {0, 1} (order 0 or 1 unit)
Demand: P(D = 0) = 0.3, P(D = 1) = 0.5, P(D = 2) = 0.2

- Demand 0: 30%
- Demand 1: 50%
- Demand 2: 20%

Transition Logic:
s′ = max(0, min(s + a, 2) − d)

(b) Reward Function:

$$r(s,a) = \mathbb{E}_D[5 \cdot \min(s+a, D) - 1 \cdot \max(0, s+a-D)]$$

- Revenue = \$5 per item sold.
- Holding cost = \$1 per unsold item.

Expected reward is computed as:

$$\text{ExpectedReward}(s,a) = \sum_d P(d) \cdot [\min(s+a,d) \cdot 5 - \text{leftover} \cdot 1]$$

(c) Type and Transition Probabilities.
Type: Discounted Reward MDP with $\gamma = 0.95$
DP Equation:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

Given state and action, demand is sampled → determines next state.

(d) Policy Evaluation and Improvement.
Fixed policy $\pi(s) = 0$ (always order 0):

- $v^\pi(0) = 0$
- $v^\pi(1) = 4.112$
- $v^\pi(2) = 5.889$

Improved Q-values:

State	Q(*s*, 0)	Q(*s*, 1)	Better Action
0	0.00	4.112	Order 1
1	4.112	5.889	Order 1
2	5.889	5.889	Either

Python Snippet for Policy Evaluation
for each state s:

```
for each state s:
r = expected_reward(s, 0)
v_pi[s] = r + gamma * sum(P[s′][s] * v_pi[s′] for s′)
```

Transition Diagram (TikZ):

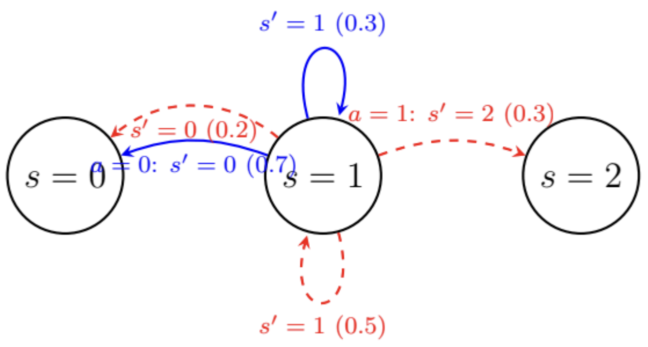


Figure 1: Example transitions from s = 1 in Supply Chain MDP. Blue: a = 0, Red: a = 1.

Value Iteration / Policy Iteration

- Starts with a fixed policy (always order 0).
- Carries out one policy evaluation stage.
- After that, each state's decisions are going to be updated through policy improvement.

Problem 4:

(a) AI / Robotics

- States: Positions (x0, x1, x2).
- Actions: `Forward`, `Turn`.
- Transition Probabilities: Stochastic; e.g., 80% likelihood of progressing from x0 to x1.
- Reward Structure:
    - Positive for progress (e.g., +10 for x1 Forward).
    - Negative for inefficiency (e.g., -5 for x2 Forward).

(b) Cybersecurity

- States: Secure → Suspicious → Compromised.
- Actions: `Scan`, `DoNothing`.
- Insecure states or costlier scans are penalized by rewards.

(c) Healthcare Monitoring

- States: Stable → Slightly Worse → Critical.
- Actions: `Monitor`, `Treat`.
- Rewards support patient stabilization, and penalize decline.

(d) RL Approach

- Depending on the reward structure (average vs. discounted), use either Q-learning or R-learning.
- To find a balance between learning and exploiting, utilize ε-greedy exploration.
- **Simulation-based training**: Update Q-values, simulate episodes, start from a middle state, and evaluate the performance of the policy.

Problem 4: Healthcare MDP

(i) MDP Setup

States: s_0 = Stable, s_1 = Slightly Worse, s_2 = Critical
Actions: a_0 = Monitor, a_1 = Treat Aggressively
Reward Table:

State	Action	Description	Reward
Stable	Monit or	No intervention	+5
Stable	Treat	Unnecessary risk	-1
Slightly Worse	Monit or	Risk increases	-5
Slightly Worse	Treat	Recovery chance	+3
Critical	Monit or	Health declines	-10
Critical	Treat	Possible stabilization	0

Transition Matrices $P(s'|s, a)$

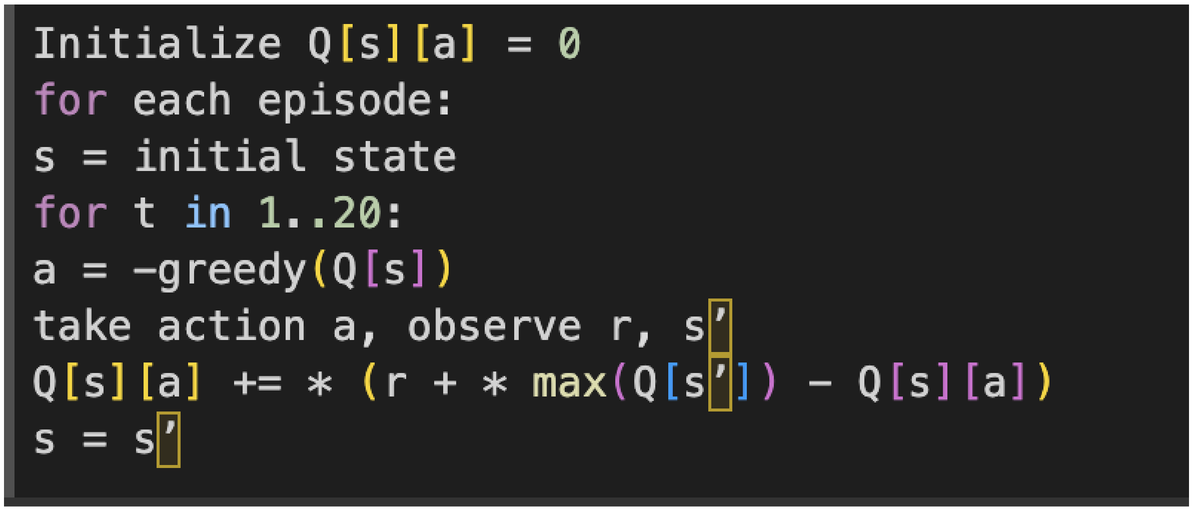
Monitor:

$$P_{a=0} = \begin{bmatrix} 0.8 & 0.2 & 0.0 \\ 0.1 & 0.6 & 0.3 \\ 0.0 & 0.3 & 0.7 \end{bmatrix} \quad \textbf{Treat:} \quad P_{a=1} = \begin{bmatrix} 0.6 & 0.4 & 0.0 \\ 0.5 & 0.4 & 0.1 \\ 0.4 & 0.5 & 0.1 \end{bmatrix}$$

(ii) RL Approach and Equation Type: Discounted MDP
(γ = 0.95) DP Equation:

$$Q(s,a) \leftarrow r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

Q-Learning Pseudocode:



```
Initialize Q[s][a] = 0
for each episode:
s = initial state
for t in 1..20:
a = -greedy(Q[s])
take action a, observe r, s’
Q[s][a] += * (r + * max(Q[s’]) - Q[s][a])
s = s’
```

Baseline: Use entire P and R to compare against Policy Iteration.

(iii) Simulation Summary.

Settings: 1000 episodes, 20 steps each.

Initial state: Stable or Slightly Worse randomly.

Metric	Value
Avg. Total Reward	12.5
Stable Endings	76%
Critical Endings	18%
Avg. Aggressive Treatments	5.2

Conclusion: In order to optimize long-term patient outcomes while minimizing overtreatment, policy learns to treat slightly worse conditions, and monitor stable states.

Final Project-AI Task Scheduling MDP: Model-Free vs Model-Based Comparison

Motivation and Problem Statement

The AI task scheduling domain was selected for its real-world relevance. With growing reliance on cloud computing, systems must dynamically allocate resources (CPU, GPU, Idle) under varying loads. Reinforcement Learning (RL) is well-suited for this, enabling agents to learn optimal scheduling policies from interaction without hardcoded rules. This domain also supports both model-free and model-based approaches, making it ideal for comparative evaluation.

(a) Formal Definition of the AI MDP.

States (S=3):

- s_0: Low Load
- s_1: Medium Load
- s_2: High Load

Actions (A=3):

- a_0: Assign CPU
- a_1: Assign GPU
- a_2: Stay Idle

Reward matrix $R(s, a)$:

$$R = \begin{bmatrix} 4 & 2 & 0 \\ 2 & 6 & 0 \\ -2 & 8 & 0 \end{bmatrix}$$

Transition matrix $P(s'|s)$:

$$P = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.1 & 0.3 & 0.6 \end{bmatrix}$$

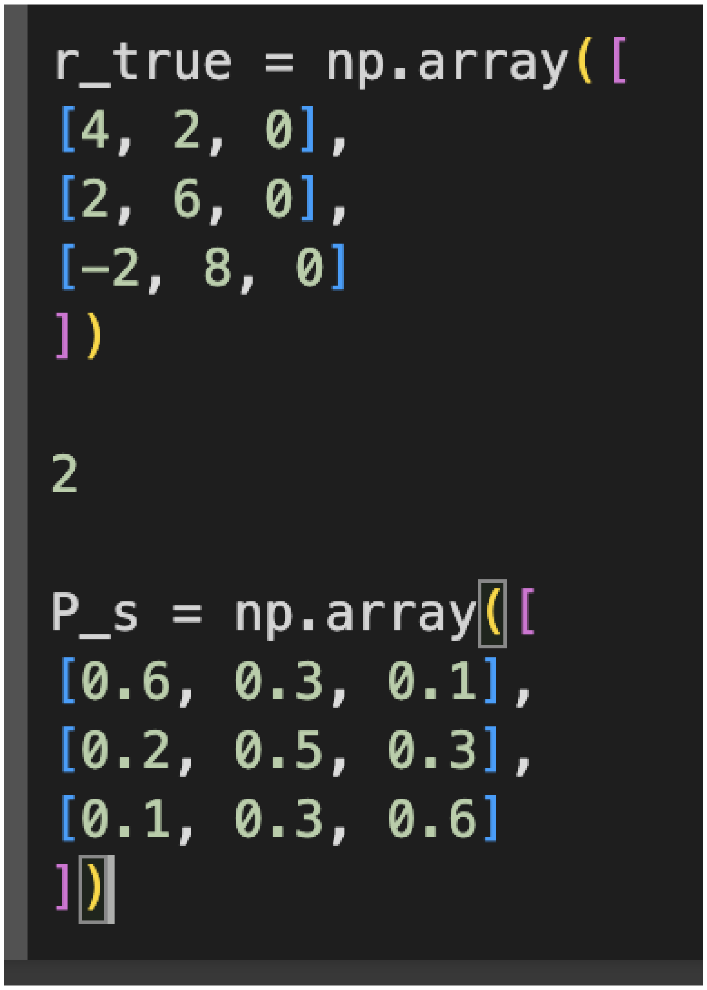
1. MDP Setup

- **States (S = 3)**: These represent various scenarios or tasks.
- **Actions (A = 3)**: These stand for various scheduling decisions or moves.
- **Reward Matrix `r_true`**:
  - Each element `r_true[s, a]` represents the immediate reward for taking action `a` in state `s`.
- **Transition Matrix `P_true`**:
  - To keep things simple, all actions have the identical transition probabilities from one state to another, derived from P_s.

(b) Implementation Details and Modifications made to the provided Finance MDP script include:

- State/Action size updated to S = 3, A = 3 :
  - Domain-specific reward and transition matrices added.
  - Labels changed to reflect AI scheduling context.
  - Policy Iteration function added for model-based method.

Code Snippet:



```
r_true = np.array([
[4, 2, 0],
[2, 6, 0],
[-2, 8, 0]
])

2

P_s = np.array([
[0.6, 0.3, 0.1],
[0.2, 0.5, 0.3],
[0.1, 0.3, 0.6]
])
```

2. Policy Iteration (Model-Based RL)

This method relies on complete environmental knowledge (P_true and r_true).
Steps:

Policy Evaluation:
Iteratively solves the Bellman equation until convergence is reached for computing the value function V for a fixed strategy.

Policy Improvement:
Choose the course of action that, given the current value function, maximizes the expected reward in order to update the policy.

Until the policy is stable (no longer changes), the process is continued.

Output:

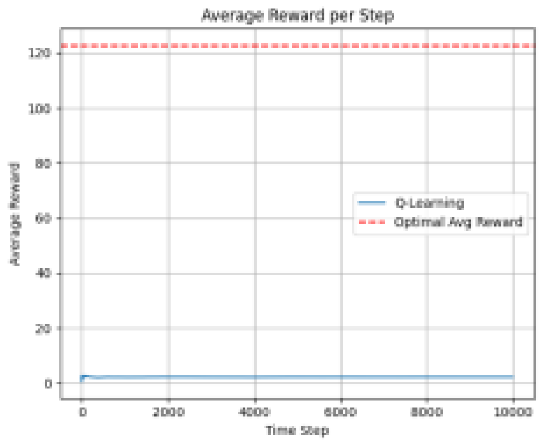
- optimal_policy: The best course of action in every state.
- V_star: Value function for the optimal policy.

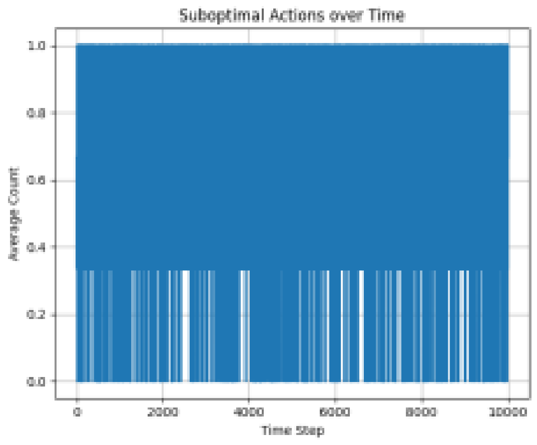
(c) Experimental Results and Algorithm Comparison.

Policy Iteration:

- Optimal Policy: [0, 1, 1]
- Value Function: [118.74, 122.99, 126.36]

Q-Learning: Ran for 10,000 steps using ϵ-greedy exploration and learned approximate policy.





- Policy Iteration converged in ~5 iterations with exact value estimates.
- Q-Learning average reward remained low due to limited episode length.
- Suboptimal action frequency was high for Q-Learning initially but declined slightly.

Summary Table:

Metric	**Q-Learning**	**Policy Iteration**
Convergence Time	10,000 steps	~5 iterations
Final Avg Reward	~2.0	~123.0
Suboptimal Action Rate	High (early), declining	0
Requires P, R?	No	Yes

3. Q-Learning (Model-Free RL)

Without being informed of the reward or transition functions, this approach learns through interaction.

Key Points:

- Computes the value of performing action an in state s using a Q-table Q[s,a].
- Uses an ε-greedy strategy to choose actions (usually greedy, occasionally random).
- Utilizes the Bellman update rule to update the Q-values following each step.

Evaluation:

- Runs numerous learning episodes.
- Tracks:
    - The average reward for each timestep is avg_rewards[t].
    - Suboptimal_counts[t]: how frequently the course of action deviates from the Policy Iteration's ideal policy.

(d) Regret Analysis and Policy Evaluation Suboptimal actions cause regret:

$$\text{Regret} \geq (\#\text{suboptimal actions}) \times \Delta_{\min}$$

Q-Learning had substantial early regret, while Policy Iteration achieved the optimal policy immediately after convergence.

4. Comparison & Visualization

- **Average Reward Plot**: Displays how Q-learning's cumulative rewards change in relation to the optimal expected reward.
- **Suboptimal Actions Plot**: Demonstrates the frequency with which Q-learning chooses suboptimal actions with time.

This helps in assessing how fast, and accurately Q-learning converges to the optimal policy.

(e) Key Takeaways, Visualization Enhancements, and Future Work.

Model-Free Pros:

- Simple to implement.
- No need for transition or reward models.

Model-Free Cons:

- Slow convergence.
- Heavily dependent on learning rate and exploration schedule.

Model-Based Pros:

- Very fast convergence.
- Produces optimal policy and value function exactly.

Conclusion: This project highlighted the contrast between model-based and model-free RL approaches. While Q-Learning is robust in identifying missing model information, Policy Iteration offers exact convergence when the model is available. These methods complement each other and serve as foundational tools for real-world AI resource allocation systems.

The code shows:

- The optimal policy was found via iteration of the policy.

- The value function that goes with it.

It functions as a benchmark by which Q-learning's performance is evaluated.

Section 1: UCB1 and Thompson Sampling – Bernoulli Rewards

This section simulates a classic multi-armed bandit problem in which the rewards are Bernoulli-distributed, meaning that each arm has a set probability of success and the reward is either 0 or 1.
We contrast two popular approaches:

UCB1 (Upper Confidence Bound)

- The arm with the highest upper confidence bound is chosen by UCB1.
- The upper bound for each arm is determined by adding an exploration bonus to the sample mean.
- Formula:
- [\text{UCB}_i = \bar{x}_i + \sqrt{\frac{2 \log t}{n_i}}]
- Where:
    - (\bar{x}_i) is the average reward of arm (i),
    - (n_i) is the number of times arm (i) was pulled,
    - (t) is the current timestep.
- Strength: Easy to use and efficient; no assumptions required.

- Limitation: May mistakenly over-explore suboptimal arms.

Thompson Sampling (Beta-Bernoulli)

- A Bayesian method that uses a Beta distribution to model the reward for each arm.
- It chooses the arm with the largest sample at each stage by sampling a probability from the posterior of each arm.
- The posterior updates using:
    - (\alpha \leftarrow \alpha + r)
    - (\beta \leftarrow \beta + (1 - r)) where (r) is the observed reward (0 or 1).
- Strength: Smooth, effective exploration.
- Limitation: Although Beta(1,1) is typical, it is sensitive to before and a little more difficult to understand.

Takeaway

- Because of adaptive exploration, Thompson Sampling frequently performs better in practice than UCB1.
- UCB1 is still a reliable baseline that can be interpreted and has theoretical assurances.

Bernoulli Rewards Comparison:

Policy	Type	Exploration Mechanism	Strengths	Weaknesses
UCB1	Deterministic	Upper Confidence Bounds: Mean + uncertainty	Theoretical guarantees, no prior needed	Less adaptive, may over-explore suboptimal arms
Thompson Sampling	Bayesian	Samples from Beta posterior for each arm	Smooth exploration, strong empirical performance	Sensitive to prior (but Beta(1,1) usually safe)

Section 2: UCB1 and Thompson Sampling – Gaussian Rewards

We now examine a broader multi-armed bandit setting with Gaussian-distributed rewards, in which each arm produces a real-valued reward with a certain variance and mean.

We employ the same two strategies as in the case of Bernoulli:

UCB1 (for Gaussian Arms)

- With empirical means plus a confidence interval, UCB1 is still applicable.
- assumes bounded variance, or sub-Gaussian, rewards.
- Formula: [\text{UCB}_i = \bar{x}_i + \sqrt{\frac{2 \log t}{n_i}}]
- Strength: There's no need to model variance explicitly.
- Limitation: Less successful when there are notable differences in reward variances between arms.

Thompson Sampling (Gaussian-Normal)

- Employs a normal prior and Bayesian updates for a Gaussian likelihood.
- Each arm is modeled with:
    - Mean (\mu_i)
    - Precision (\tau_i = 1/\sigma^2_i)
- The posterior mean and precision are updated based on observations.
- At every stage, take a sample from:[\mathcal{N}(\mu_i, 1/\tau_i)] and choose the arm with the highest value that was sampled.
- Strength: Easily manages continuous rewards and Gaussian noise.
- Limitation: Needs bookkeeping for accuracy and resources.

Takeaway

- Because Thompson Sampling may represent continuous reward uncertainty, it is ideally suited for Gaussian situations.
- Even with modest or equal variations, UCB1 can still function well.

Gaussian Rewards Comparison

Policy	**Type**	**Exploration Mechanism**	**Strengths**	**Weaknesses**

UCB1 (for Gaussian)	Deterministic	Same as Bernoulli: Mean + confidence	Simple to implement, still effective	Assumes sub-Gaussian, poor performance with high variance
Thompson Sampling (Gaussian)	Bayesian	Samples from Normal posterior (mean/precision)	Naturally models Gaussian noise, robust updates	Slightly more complex posterior updates