RL CIA-2

Create a 100x100 grid with obstacles in between 2 random points. Build an MDP based RL agent to optimise both policies and actions at every state. Benchmark DP method with other RL solutions for the same problem.

A Reinforcement Learning (RL) agent is designed to navigate a grid environment while avoiding obstacles and finding a path to a target (goal) point. The grid is of size 100x100, with random obstacles placed between the starting and goal points. Three algorithms are used to train the agent: Q-Learning, SARSA, and Value Iteration.

We compare the performance and convergence rates of these algorithms based on metrics such as average rewards, average steps to reach the goal, and computation time.

Problem Setup

1. **Grid Environment**:
    - The environment is a 100x100 grid with randomly placed obstacles.
    - A start point and a goal point are randomly generated.
    - The agent receives a reward of +10 for reaching the goal, -5 for hitting an obstacle, and -0.1 for each move.
2. **Actions**:
    - The agent can move up, down, left, or right.
3. **Metrics**:
    - **Average Reward**: Measures the effectiveness of the agent's policy.
    - **Average Steps**: Tracks efficiency in reaching the goal.
    - **Computation Time**: Measures the training time for each algorithm.

Reinforcement Learning Algorithms

1. **Q-Learning**:
    - **Q-Learning** is an off-policy RL algorithm that updates Q-values based on the maximum expected reward of future actions. It balances exploration and exploitation using an epsilon-greedy approach.
    - **Hyperparameters**:
        - Learning Rate (α): 0.1
        - Discount Factor (γ): 0.9
        - Exploration Rate (ε): 0.1
2. **SARSA**:

- **SARSA** (State-Action-Reward-State-Action) is an on-policy algorithm that updates Q-values based on the next action chosen by the policy.
- **Hyperparameters**:
  - Similar to Q-Learning.

3. **Value Iteration**:
   - **Value Iteration** is a Dynamic Programming method used to compute an optimal policy by iteratively updating value estimates until they converge.
   - **Convergence Threshold (θ)**: 1e-4

Results

1. **Convergence Plot**:
   - **Q-Learning and SARSA**: The rewards increase over episodes, indicating policy improvement as agents learn to avoid obstacles and move efficiently toward the goal.
   - **Value Iteration**: The delta changes decrease over iterations, showing that the value estimates are stabilizing.
2. **Performance Comparison**:
   - **Q-Learning** generally learns faster but may not always converge as smoothly as SARSA.
   - **SARSA** provides more stable updates but might converge more slowly in exploration-heavy environments.
   - **Value Iteration** is efficient for solving smaller, well-defined grids but is computationally expensive on large grids.

Conclusion

This assignment demonstrated how different RL methods—Q-Learning, SARSA, and Value Iteration—approach the problem of navigating a grid environment with obstacles. The agents were trained and evaluated based on reward, steps taken, and time, and their performance was visualized through convergence plots.

This experiment shows that Q-Learning and SARSA are effective for learning in large environments but vary in exploration strategies, while Value Iteration is best for smaller problems due to its computational demands.