CS747 - Foundations of Intelligent and Learning Agents

Assignment 2 Report

Kaishva Chintan Shah (200020066)

Department of Electrical Engineering
Indian Institute of Technology, Bombay
October 2023

1 Task 1: MDP Planning

1.1 Value Iteration Algorithm

1.1.1 Design Decisions

The Value Iteration algorithm is a fundamental approach for solving Markov Decision Processes (MDPs). Its design decisions aim to iteratively refine the value function and policy for an MDP by optimizing the Bellman equation. Here are the key design decisions:

1. **Convergence Threshold:** The algorithm uses a user-defined convergence threshold to determine when to stop iterating. When the maximum absolute difference between consecutive value functions falls below this threshold, the algorithm terminates.

2. **Vectorization:** Value Iteration efficiently computes Q-values for all states and actions by vectorizing the operations, enhancing performance for large state spaces.

3. **Policy and Value Function:** The algorithm maintains two arrays, the policy and the value function. The policy represents the best action to take in each state, and the value function stores the expected cumulative rewards from each state.

1.1.2 Assumptions

The Value Iteration algorithm relies on several assumptions:

1. **Convergence:** It assumes that the MDP has a finite number of states and actions and that the Bellman equation is guaranteed to converge within the provided threshold.

2. **Deterministic Transitions:** Value Iteration assumes deterministic transitions, where the next state solely depends on the current state and chosen action.

3. **Finite Horizon:** It assumes that the MDP has a finite time horizon. This ensures that the algorithm eventually reaches a state where the convergence criterion is met.

1.1.3 Observations

Based on the design and assumptions, several observations can be made about the Value Iteration algorithm:

1. **Convergence Rate:** Value Iteration typically converges relatively slowly, especially in large MDPs, because it exhaustively evaluates all states and actions at each iteration. The convergence threshold significantly influences the number of iterations required.

2. **Optimal Policy:** At the end of the algorithm, the policy represents the optimal action to take in each state, given the underlying MDP. The value function stores the expected rewards associated with this optimal policy.

3. **Policy Improvement:** Value Iteration follows a policy improvement strategy by iteratively refining the policy based on the current value function. It's guaranteed to find the optimal policy if sufficient iterations are allowed.

4. **Memory and Computational Complexity:** The algorithm's memory usage and computational complexity depend on the number of states, actions, and the chosen convergence threshold. It may become impractical for MDPs with extensive state spaces.

1.1.4 Conclusion

Value Iteration is a powerful algorithm for solving MDPs and finding optimal policies. While it has design decisions that lead to guaranteed convergence, the rate of convergence can be slow for large MDPs. Careful consideration of the convergence threshold and computational resources is necessary.

1.2 Howard's Policy Iteration Algorithm

1.2.1 Design Decisions

Howard's Policy Iteration is an iterative approach used to improve policies within Markov Decision Processes (MDPs). The algorithm's design decisions focus on policy evaluation and policy improvement. Here are the key design decisions:

1. **Random Initialization:** Howard's Policy Iteration starts with a randomly initialized policy. Actions are assigned randomly to states.

2. **Policy Evaluation:** The algorithm evaluates the current policy by solving the linear system of equations. This step computes the value function for the current policy.

3. **Policy Improvement:** Policy improvement is based on the calculated value function. Actions that maximize the Q-values are selected to create a new policy.

4. **Convergence Check:** The algorithm checks for convergence by comparing the new policy to the previous policy. If they match, the algorithm terminates.

5. **Maximum Iterations:** To avoid running indefinitely, the algorithm is equipped with a maximum number of iterations (e.g., 100,000) as a termination condition.

1.2.2 Assumptions

The Howard's Policy Iteration algorithm is based on several assumptions:

1. **Convergence:** It assumes that the MDP has a finite number of states and actions and that a solution within the chosen maximum iterations is attainable.
2. **Random Action Selection:** In cases where multiple actions lead to the same optimal value, the algorithm assumes that random action selection is acceptable.
3. **Exploration:** The algorithm introduces exploration by allowing random action selection among equally improving actions. It assumes that exploration enhances policy improvement.
4. **Linear Equation Solver:** The algorithm utilizes a linear equation solver to calculate the value function based on the current policy. This solver involves matrix operations to efficiently compute the value function.

1.2.3 Linear Equation Solver

Howard's Policy Iteration employs a linear equation solver to calculate the value function for the current policy. The solver uses the Bellman equation and matrix operations to efficiently find the expected cumulative rewards for each state. The linear equation solver consists of the following components:

- **Matrix A:** A matrix representing the coefficients for the value function equation. It incorporates transition probabilities and discount factors.
- **Vector b:** A vector representing the right-hand side of the equation, which includes the expected rewards.
- **Diagonal Elements:** The diagonal elements of matrix A correspond to self-transitions. They are computed as $1 - \text{discount} \times \text{transition probability}$.
- **Off-diagonal Elements:** The off-diagonal elements of matrix A are computed as $-\text{discount} \times \text{transition probability}$.

The linear system is solved using standard numerical methods, such as the NumPy function `np.linalg.solve`.

1.2.4 Observations

Based on the design and assumptions, several observations about Howard's Policy Iteration can be made:

1. **Convergence Guarantee:** Howard's Policy Iteration guarantees convergence to the optimal policy when at least one improving action exists for each state.

2. **Exploration:** The algorithm introduces exploration by randomly choosing between multiple improving actions. This random selection can improve convergence.

3. **Maximum Iterations:** To prevent indefinite execution, the algorithm includes a maximum number of iterations. If convergence is not reached within this limit, the algorithm terminates.

4. **Efficient Value Function Calculation:** The use of a linear equation solver allows for the efficient calculation of the value function. It avoids the need for exhaustive iteration over states, making it suitable for large MDPs.

1.2.5 Conclusion

Howard's Policy Iteration is a robust algorithm for solving MDPs. It iteratively refines policies through policy evaluation and improvement, and its convergence is ensured as long as there is at least one improving action at each state. The introduction of exploration through random action selection enhances its performance.

1.3 Linear Programming (LP) Algorithm

1.3.1 Design Decisions

The Linear Programming (LP) algorithm is a versatile approach for solving Markov Decision Processes (MDPs) by formulating the problem as a linear program. The design decisions focus on creating an LP problem to optimize the value function and policy. Here are the key design decisions:

1. **Linear Program Formulation:** The algorithm formulates the MDP problem as an LP problem by defining LP variables for state values and creating an objective function to maximize the sum of state values.

2. **Constraints:** Constraints are defined based on the Bellman equation. They ensure that the value function respects the expected cumulative rewards and transition probabilities while considering the discount factor.

3. **Linear Program Solver:** The algorithm relies on LP solvers, such as PuLP, to solve the formulated LP problem and obtain the optimal values and policy.

1.3.2 Assumptions

The LP algorithm is based on several assumptions:

1. **LP Formulation Applicability:** It assumes that the MDP problem can be accurately represented as a linear program. In some cases, LP might not be suitable for highly nonlinear or non-convex problems.

2. **Availability of LP Solver:** The algorithm assumes access to an LP solver, such as PuLP, which can efficiently handle linear programs. The solver should be able to find the optimal solution.

1.3.3 Observations

Based on the design and assumptions, several observations about the LP algorithm can be made:

1. **Optimal Solution:** The LP algorithm guarantees an optimal solution by formulating the MDP as a linear program. It finds the value function and policy that maximize expected cumulative rewards.

2. **Suitability for Large MDPs:** LP is a viable choice for large MDPs with numerous states and actions. The use of LP solvers allows efficient solution even for complex problems.

3. **Flexibility:** The algorithm is flexible and can be used for various types of MDPs, including those with non-standard reward structures and transition dynamics.

1.3.4 Conclusion

The Linear Programming (LP) algorithm is a powerful technique for solving MDPs. It leverages LP solvers to efficiently find the optimal value function and policy while accommodating a wide range of MDP complexities. The LP formulation allows for mathematical precision in solving MDPs.

2 Task 2

Disclaimer: While running the task on the autograder, I faced the error for memory issues multiple times, "Unable to allocate 5.00 GiB for an array with shape (8194, 10, 8194) and data type float64". Please stop all the unnecessary background processes to run the planner without any errors.

2.1 Movement Function

The 'movement' function plays a crucial role in modelling the movement dynamics of player b1, player b2, and defender R within the Football problem's Markov Decision Process (MDP). It computes transition probabilities and rewards based on the current state, action, and MDP parameters.

2.1.1 Parameters

- `state` - Represents the game's current state, including the positions of player b1, player b2, defender R, and possession information.

- `action` - An integer representing the movement action.

- `transitions` - Contains transition probabilities from one state to another in the MDP.

- `rewards` - Stores the reward values associated with different state-action-state transitions.
- `end_state` - Indicates the end state where the game concludes.
- `p` - An MDP parameter affecting transition probabilities.
- `q` - Another MDP parameter affecting transition probabilities.
- `R_mdp_data` - Contains additional information about the state of the game, including probabilities of new positions for the defender R.

2.1.2 Position Updates

The function updates the positions of players b1 and b2 based on the chosen movement action, considering grid bounds and potential out-of-bounds scenarios that would lead to the end of an episode.

2.1.3 Possession Handling

The function distinguishes between two scenarios: when player 1 (b1) has possession (poss = 1) and when player 2 (b2) has possession (poss = 2). It manages the transition probabilities and rewards accordingly.

2.1.4 End of Episode

In cases where an episode ends, such as when a player moves out of bounds, the function triggers transitions leading to the end state with associated probabilities and rewards.

2.2 Passing Function

The ‘passing‘ function is a critical component of the Football problem, modeling the passing dynamics of the ball between players b1 and b2 while considering the position of the defender R within the Markov Decision Process (MDP). The function calculates transition probabilities and rewards based on the current state, action, and MDP parameters.

2.2.1 Parameters

- `state` - Represents the current state of the game, including the positions of players b1, b2, the defender R, and possession information.
- `action` - An integer representing the passing action.
- `transitions` - Contains transition probabilities from one state to another in the MDP.
- `rewards` - Stores the reward values associated with different state-action-state transitions.

- `end_state` - Indicates the end state where the game concludes.
- `p` - An MDP parameter affecting transition probabilities in the passing dynamics.
- `q` - Another MDP parameter influencing transition probabilities.
- `R_mdp_data` - Contains additional information about the state of the game, including probabilities of new positions for the defender R.

2.2.2 Interception Probability

The function calculates the probability of a successful pass, which depends on the action taken, possession, and other MDP parameters. The interception probability is adjusted based on the values of `p` and `q`.

2.2.3 Interception Detection

To account for potential interceptions by defender R, the function checks if the ball's path aligns with the defender's position. If an interception is detected, the interception probability is reduced accordingly.

2.2.4 State Transition and Rewards

The 'passing' function then updates the state, considering the new positions of players b1, b2, and the defender R, as well as the change in possession. It calculates the probability of a successful pass and updates the transition probabilities and rewards accordingly.

2.2.5 Losing Possession

In cases where a pass is intercepted, the function triggers a transition leading to the loss of possession by the passing team and initiates an episode ending. This transition is determined by the probability of a failed pass, calculated as one minus the probability of a successful pass.

2.3 Shooting Function

The 'shooting' function represents a critical aspect of the Football problem's Markov Decision Process (MDP), simulating the shooting dynamics of player b1 and player b2 with the goal of scoring. This function computes transition probabilities and rewards based on the current state, action, and MDP parameters, considering the positions of players b1 and b2 and the defender R.

2.3.1 Parameters

- `state` - Denotes the game's current state, which includes the positions of players b1, b2, the defender R, and possession information.

- `action` - An integer representing the shooting action.
- `transitions` - Stores transition probabilities from one state to another within the MDP.
- `rewards` - Manages the reward values associated with different state-action-state transitions.
- `end_state` - Identifies the end state where the game concludes.
- `p` - An MDP parameter influencing transition probabilities during the shooting phase.
- `q` - Another MDP parameter affecting transition probabilities during shooting.
- `R_mdp_data` - Contains supplemental information about the game's state, including probabilities of new positions for the defender R.
- `score_state` - Represents the state where a goal is scored and the game ends.

2.3.2 Interception Probability

The 'shooting' function calculates the probability of a successful shot on the goal. This probability is influenced by the possession state (poss), and other MDP parameters, specifically `p` and `q`. The `probability_shoot_succ` variable is adjusted accordingly.

2.3.3 Interception Detection

To account for potential interceptions by defender R, the function checks whether the defender is in a position to intercept the shot. If an interception is detected, the `probability_shoot_succ` is reduced by 50%.

2.3.4 Scoring a Goal

If a shot succeeds, the function triggers a transition to the `score_state`, representing scoring a goal. This transition is associated with a reward of 1.

2.3.5 Missing the Shot

In cases where the shot is unsuccessful, the function initiates a transition leading to the loss of possession to defender R, marking the end of the episode. The transition probability for this scenario is calculated as `probability_shoot_fail`, which is complementary to the probability of a successful shot.

2.4 Results

2.4.1 Probability of Winning

The probability of winning, starting from the specified position $[05, 09, 08, 1]$ against Policy-1, is closely tied to the expected number of goals scored. We can justify using the optimal value function to represent the probability of scoring goals based on the nature of the rewards in the problem. In this scenario, a reward of 1 is only granted for scoring goals, while all other state action pairs receive a reward of 0. Since the discount factor is set to 1, the value function can be expressed as an expectation of the summation of rewards. Therefore, we seek to calculate $E[1|s]$, which can be interpreted as the product of 1 and the probability of scoring 1 goal in state s. In this context, this approach directly corresponds to estimating the probability of scoring a goal.

- For different values of p (ranging from 0 to 0.5) with a fixed q value of 0.7 (Graph 1):
  - As p increases, the probability of goal scoring typically decreases. This is because higher values of p make it more challenging to execute actions, such as movement, successfully, leading to goal scoring. Being unsuccessful at any action causes the episode to end. Since p has direct control over the movement of the players, its value is critical in deciding the probability of scoring a goal.
- For different values of q (ranging from 0.6 to 1) with a fixed p value of 0.3 (Graph 2):
  - As q increases, the probability of goal scoring is expected to increase. Higher q values represent a greater probability of successful goal-scoring actions, such as passing and scoring, leading to more goals scored. Again, we know that being unsuccessful at any action will lead to an episode end; increasing q increases the possibility of performing actions that get you to a stage where you can score a goal.

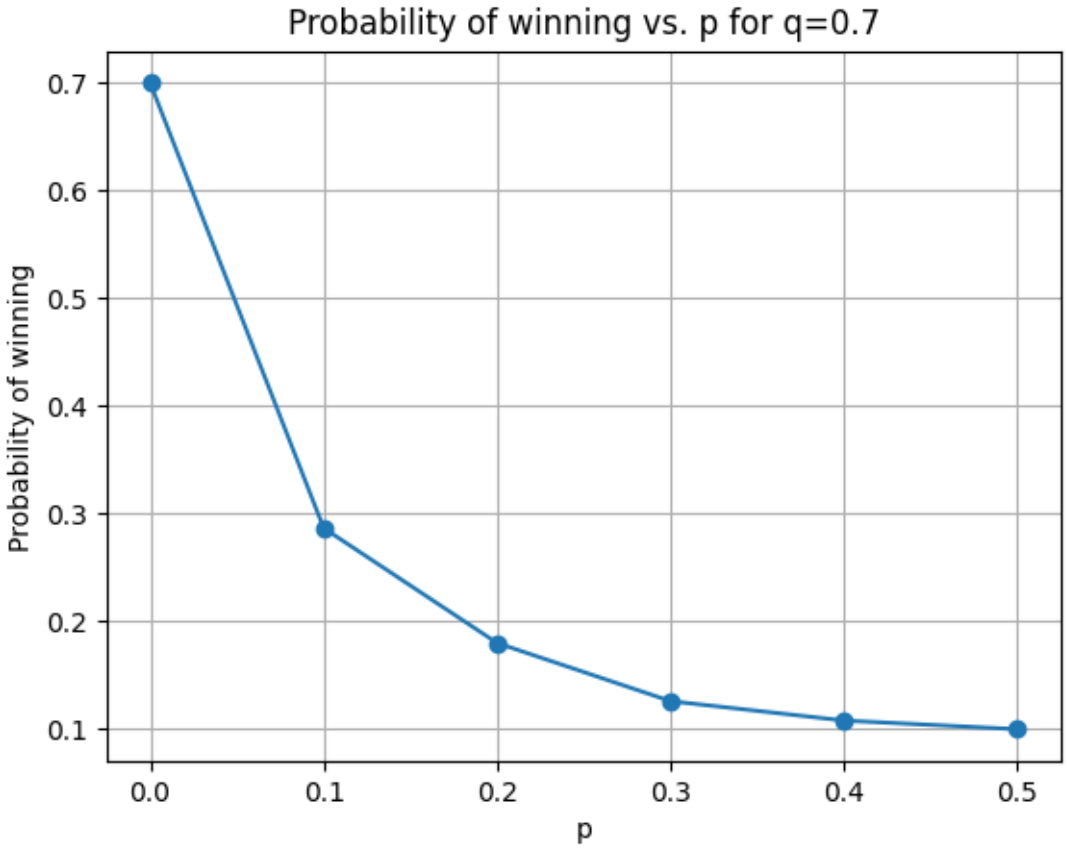

Figure 2.1: For p in 0, 0.1, 0.2, 0.3, 0.4, 0.5 and q = 0.7

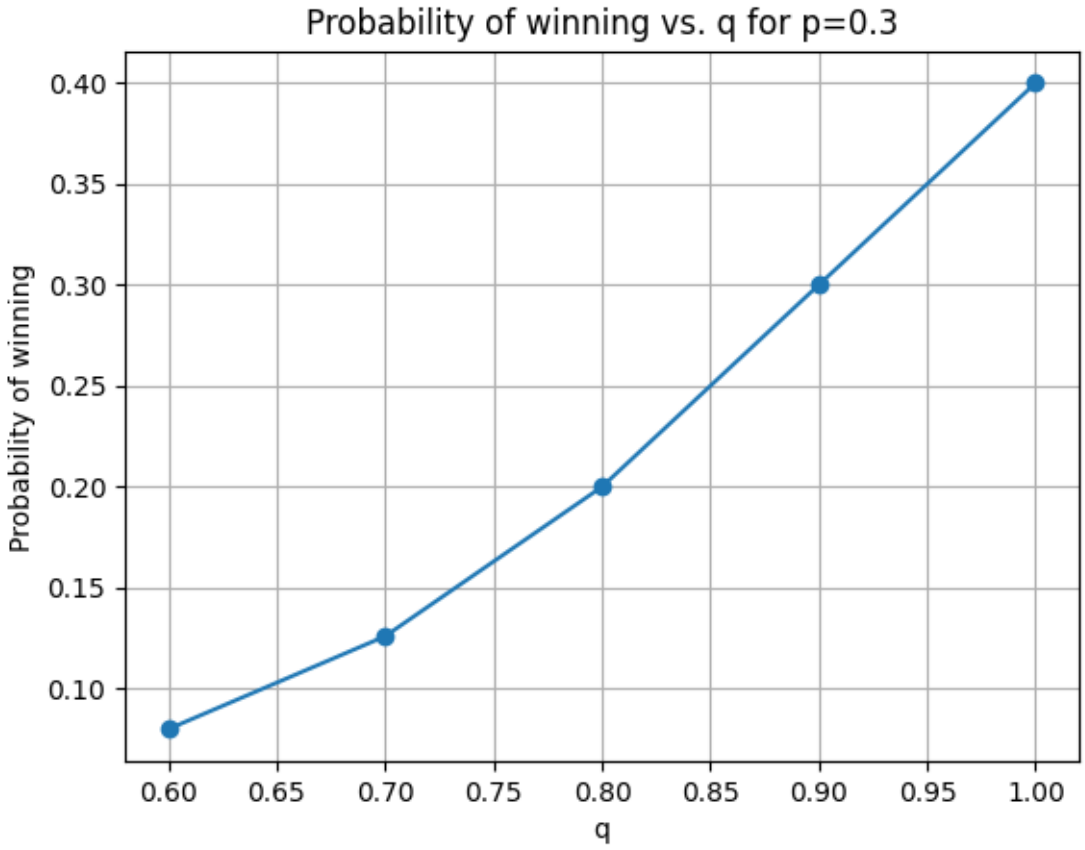

Figure 2.2: For q in 0.6, 0.7, 0.8, 0.9, 1 and p = 0.3