| |
|---|
| Experiment No.4 |
| Applying dynamic programming algorithms, such as policy evaluation and policy improvement, to solve a small-scale MDP problem. |
| Date of Performance: |
| Date of Submission: |

**Aim:** Applying dynamic programming algorithms, such as policy evaluation and policy improvement, to solve a small-scale MDP problem.

**Objective**: The objective is to employ dynamic programming algorithms, such as policy evaluation and policy improvement, to effectively address a small-scale Markov Decision Process (MDP) problem, aiming to iteratively refine and optimize policies, thereby gaining insights into foundational concepts of reinforcement learning and optimization.

**Theory:**

**Markov Decision Processes (MDPs)**: MDPs are mathematical frameworks used to model decision-making problems where outcomes are partly random and partly under the control of a decision-maker. They consist of states, actions, transition probabilities, rewards, and a discount factor.

**Policy Evaluation**: Policy evaluation is the process of determining the value function for a given policy. The value function represents the expected cumulative reward starting from a particular state and following a specific policy. The basic idea is to iteratively update the value of each state until convergence.

Algorithm for policy evaluation :

Input:

MDP: (S, A, P, R, $\gamma$), where
S is the set of states.
A is the set of actions.
P is the transition probability matrix, P(s' | s, a), representing the probability of transitioning to state s' from state s by taking action a.
R is the reward function, R(s, a, s'), representing the immediate reward received after transitioning from state s to state s' by taking action a.
$\gamma$ is the discount factor.
Output:

Value function V(s) for each state s.
**Algorithm**:

Initialize the value function arbitrarily: V(s) for all s in S.

Repeat until convergence:

Initialize $\Delta$ to be a very large value (e.g., infinity).

For each state s in S:

Let v be the current value of V(s).

Update V(s) using the Bellman expectation equation:

$V(s) = \Sigma [P(s' | s, a) * (R(s, a, s') + \gamma * V(s'))]$ over all possible next states s' and actions a.

Update $\Delta$ to be the maximum between $\Delta$ and $|v - V(s)|$.

If $\Delta$ is smaller than a predefined threshold $\varepsilon$, break.

Return the converged value function V(s).

**Explanation:**

The algorithm iteratively updates the value function V(s) for each state s using the Bellman expectation equation until convergence.

At each iteration, it computes the expected value of being in state s and taking action a, which is the sum of immediate reward R(s, a, s') and the discounted value of the next state V(s').

The process continues until the maximum change in the value function between iterations ($\Delta$) falls below a predefined threshold $\varepsilon$, indicating convergence.

The output is the converged value function V(s), which represents the expected cumulative reward from each state under the given policy.

This algorithm is known as "Iterative Policy Evaluation" and is a fundamental component of dynamic programming approaches for solving MDPs. It provides a way to estimate the value of each state under a given policy.

**Policy Improvement**: Policy improvement involves selecting better actions in each state to improve the current policy. It's based on the idea of greedily selecting actions that maximize the expected cumulative reward given the current value function.

**Algorithm for policy improvement:**

Input:

MDP: $(S, A, P, R, \gamma)$, where

S is the set of states.

A is the set of actions.

P is the transition probability matrix, P(s' | s, a), representing the probability of transitioning to state s' from state s by taking action a.

R is the reward function, R(s, a, s'), representing the immediate reward received after transitioning from state s to state s' by taking action a.

$\gamma$ is the discount factor.

Value function V(s) for each state s.

Output:

Improved policy $\pi'(s)$ for each state s.

**Algorithm:**

Initialize a boolean variable policy_stable to true.

For each state s in S, do:

Let old_action be the current action selected by the policy $\pi$ for state s.

Compute Q-value for each action a in A:

$Q(s, a) = \Sigma [P(s' \mid s, a) * (R(s, a, s') + \gamma * V(s'))]$ over all possible next states s'.

Select the action a_max that maximizes the Q-value: a_max = argmax(Q(s, a)).

Update the policy $\pi'(s)$ to select the action a_max.

Check for policy stability:

If the new policy $\pi'$ is different from the old policy $\pi$ for any state s, set policy_stable to false.

If policy_stable is true, return the improved policy $\pi'$.

Else, return to step 2 and repeat the process with the updated policy $\pi'$.

Explanation:

The algorithm iterates over each state in the state space and computes the Q-value for each action based on the current value function V(s).

It selects the action that maximizes the Q-value as the new action for the state.

The process continues until the policy stabilizes, i.e., the new policy is the same as the old policy for all states.

The output is the improved policy $\pi'$ that greedily selects actions to maximize the expected cumulative reward according to the current value function V(s).

This algorithm is known as "Policy Improvement" and is used in combination with policy evaluation to iteratively improve the policy until convergence to an optimal policy in dynamic programming approaches for solving MDPs.

1. **Iterative Policy Evaluation and Policy Improvement**: The process of policy evaluation and policy improvement is often interleaved. After evaluating a policy, we improve it by selecting

better actions based on the updated value function. Then, we re-evaluate the policy to refine the value estimates further.

2. **Convergence**: Both policy evaluation and policy improvement converge to the optimal value function and policy if executed iteratively until convergence.
3. **Implementation**: Dynamic programming algorithms can be implemented efficiently using programming languages like Python, where you can define MDPs, transition probabilities, rewards, value functions, and policies, and then iteratively update them until convergence.

This general approach can be applied to small-scale MDP problems to find the optimal policy efficiently. However, for larger problems, approximate methods like reinforcement learning techniques may be more practical due to the computational complexity of dynamic programming algorithms.

**Conclusion:**

1. Give one example of dynamic Programming
2. Explain how dynamic programming is utilized in reinforcement learning