**First-Visit Monte Carlo Prediction for State-Value Estimation**

**1. Introduction** Reinforcement Learning (RL) is a powerful branch of machine learning focused on training agents to make sequential decisions in an environment to maximize a cumulative reward signal. Applications range from game playing (AlphaGo) and robotics to resource management and recommendation systems. A core task within RL is policy evaluation: determining how good a particular strategy (policy) is. This is typically done by estimating the state-value function, denoted V\_π(s), which quantifies the expected long-term return (cumulative discounted reward) starting from a state s and following policy π thereafter. Monte Carlo (MC) methods offer a way to learn value functions directly from experience, gathered by running complete episodes of interaction with the environment. They are considered model-free because they don't need prior knowledge of the environment's dynamics (i.e., the probabilities of transitioning between states and receiving rewards). This report provides a comprehensive overview of the First-Visit Monte Carlo Prediction algorithm, a fundamental MC technique for estimating V\_π(s). We will explore its theoretical basis, analyze a Python implementation, discuss its properties, and compare it briefly with related concepts.

**2. Foundational Concepts in RL** To understand First-Visit MC, let's clarify some essential RL terms:

* **Agent:** The learner or decision-maker.
* **Environment:** The external system the agent interacts with.
* **State (s):** A representation of the current situation of the environment. The set of all possible states is denoted by S.
* **Action (a):** A choice the agent can make in a given state. The set of actions available in state s is A(s).
* **Policy (π):** The agent's strategy or behavior. It maps states to actions (deterministic policy, π(s) = a) or probabilities of taking actions (stochastic policy, π(a|s)). The goal of policy evaluation is to assess a given, fixed policy π.
* **Reward (R):** A scalar feedback signal received from the environment after taking an action in a state. R\_{t+1} is the reward received after taking action A\_t in state S\_t and transitioning to state S\_{t+1}.
* **Episode:** A sequence of interactions starting from an initial state and ending at a terminal state. It represents one complete "play" or "trial".

Example: S\_0, A\_0, R\_1, S\_1, A\_1, R\_2, ..., S\_{T-1}, A\_{T-1}, R\_T, S\_T.

* **Discount Factor (γ):** A value between 0 and 1 (0 ≤ γ ≤ 1) that determines the present value of future rewards. Rewards received k steps in the future are discounted by γ^k. It prevents infinite returns in cyclic tasks and allows trading off immediate vs. future rewards.
* **Return (G\_t):** The total discounted reward from time step t onwards in an episode:

G\_t = R\_{t+1} + γ R\_{t+2} + γ² R\_{t+3} + ... + γ^{T-t-1} R\_T = Σ\_{k=0}^{T-t-1} γ^k R\_{t+k+1}.

* **State-Value Function (V\_π(s)):** The expected return when starting in state s and following policy

π: V\_π(s) = E\_π[G\_t | S\_t = s].

The expectation E\_π[·] accounts for randomness in the environment's transitions and/or the policy itself.

**3. Monte Carlo Methods for Prediction** MC methods learn value functions by averaging the returns observed in sample episodes.

* **Model-Free Nature:** They learn directly from raw experience (state, action, reward sequences) without needing a model p(s', r | s, a). This makes them applicable even when the environment's rules are unknown or too complex to model.
* **Episodic Requirement:** Standard MC methods require tasks to be episodic, meaning they must eventually terminate. Updates are performed only at the end of each episode.
* **Learning from Experience:** They estimate the expectation E\_π[·] by averaging sample returns G\_t obtained from running episodes using policy π.

**3.1. First-Visit vs. Every-Visit MC**

* **First-Visit MC:** Averages the returns following *only the first* visit to state s in each episode. Subsequent visits in the same episode are ignored for updating V(s) based on that episode's return.
* **Every-Visit MC:** Averages the returns following *every* visit to state s in each episode.
* Both methods converge to the true V\_π(s) as the number of visits approaches infinity. First-Visit MC is often preferred for analysis. Every-Visit MC can sometimes be easier to implement and may converge slightly faster. The algorithm presented focuses on First-Visit MC.

**4. The First-Visit MC Prediction Algorithm** This algorithm estimates V\_π(s) by collecting returns from the first time each state s is visited in many simulated episodes.

* **Algorithm Steps:**
  + **Input:** A policy π to be evaluated.
  + **Initialize:**
    - V(s) arbitrarily (e.g., 0) for all s ∈ S.
    - Returns(s) as an empty list for all s ∈ S.
  + **Loop (Repeat for many episodes):**
    - a. Generate Episode using policy π. Store the sequence.
    - b. Initialize Return: G ← 0.
    - c. Initialize Visited Set for Episode: states\_visited\_in\_episode ← empty set.
    - d. Process Episode Backwards (t = T-1, T-2, ..., 0):
      * i. Update Return: G ← γG + R\_{t+1}.
      * ii. Check First Visit: If state S\_t is not in states\_visited\_in\_episode:
        + Append G to Returns(S\_t).
        + Update value estimate: V(S\_t) ← average(Returns(S\_t)).
        + Add S\_t to states\_visited\_in\_episode.

* **Incremental Implementation:** More efficient updates can be done using:
  + V(S\_t) ← V(S\_t) + (1/N(S\_t)) \* (G - V(S\_t))
  + Or using a constant step-size α: V(S\_t) ← V(S\_t) + α \* (G - V(S\_t)).
  + (Note: The document mentions the provided code uses a specific averaging method that needs care across episodes ).

**5. Analysis of the Python Implementation**

* **5.1. Environment and Policy:** Uses create\_standard\_grid() for the environment. A deterministic policy dictionary is used. gamma = 0.9 discounts future rewards.
* **5.2. Simulating an Episode (play\_game):** The play\_game function simulates a trajectory and collects (state, reward) pairs. An example output is shown.
* **5.3. Calculating Returns (compute\_G / Loop Logic):** The backward loop correctly calculates discounted returns G\_t. Example returns are shown.
* **5.4. Value Updates and First-Visit Implementation:** Logic uses a seen\_states set within an episode for the first-visit check. **Important Note on Averaging:** The document points out a potential issue in code snippets (In [15], In [17]) where returns = {} might be re-initialized inside the episode loop, causing the value to reflect only the most recent episode's return instead of an average across all episodes. A corrected structure accumulating returns across episodes is suggested.
* **5.5. The Need for Exploration:** With a deterministic policy and fixed start, the agent follows the same path, missing other states. Using random starting states (**Exploring Starts**) helps ensure all relevant states are eventually visited and evaluated.

**6. Properties, Advantages, and Disadvantages of First-Visit MC**

* **Advantages:**
  + Model-Free.
  + Simple Concept.
  + Unbiased Estimate.
  + Effective in Simulation.
* **Disadvantages:**
  + High Variance.
  + Episodic Requirement.
  + Offline Updates (updates at end of episode).
  + Exploration Dependence.

**7. Conclusion** First-Visit Monte Carlo prediction is a fundamental, model-free algorithm for policy evaluation in RL. It averages returns from the first visit to each state over episodes. While simple and unbiased, it requires handling variance and ensuring exploration. The Python code analysis highlights its mechanics and the importance of exploring starts. It provides a foundation for more advanced RL techniques.

**8. References**

* Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press. (Chapter 5).
* Various online resources.
* Provided Python code snippets and image.