Optimizing Rubik's Cube Game Using State Space Search

Shreya Manyam[1,2], Vibhansh Goel[2], Battu Ujwal Reddy[2] and Erramshetti Abhilash[2]

[1] **G**roup Name: Agile
[1] IIT Vadodara, ICD

Abstract

This paper explores three key reinforcement learning concepts through practical implementations: the Matchbox Educable Noughts and Crosses Engine (MENACE), the binary bandit problem with stationary rewards, and the 10-armed bandit with non-stationary rewards. MENACE utilizes a simple trial-and-error learning mechanism to play Tic-Tac-Toe, where game states are represented as matchboxes and moves are chosen probabilistically based on bead counts. The binary bandit problem is tackled using an epsilon-greedy algorithm to balance exploration and exploitation, optimizing decisions for stationary stochastic rewards. For the 10-armed bandit problem, the algorithm is adapted to handle non-stationary rewards by employing an exponential moving average, allowing the agent to track evolving reward distributions over time. Each approach demonstrates core principles of reinforcement learning, with practical insights into the challenges of non-stationary environments.

Keywords: Reinforcement Learning (RL), Probabilistic Decision Making, Tic-Tac-Toe, 10-Armed Bandit Problem, Epsilon-Greedy Algorithm

I. MENACE (Matchbox Educable Noughts and Crosses Engine)

A. Initialize MENACE

```
- Create a set of all possible game states.
- For each game state:
    - Create a matchbox for that state.
    - For each possible move in the state:
        - Add an initial number of beads
        (e.g., 2 beads per move).
```

B. Play a Game

```
- Set the initial state to the empty
Tic-Tac-Toe board.
- Initialize an empty list to track states and
moves for this game.

Repeat until the game ends (win, lose, or draw):
    1. Retrieve the current game state.
    2. Find the corresponding matchbox.
    3. Select a move:
        - Shake the matchbox and select a move
        (based on bead quantity).
        - Record the selected move and
        current state.
    4. Apply the move and update the game board.
    5. Check if the game has ended
    (win, lose, or draw).
```

C. After the Game

```
If MENACE won:
    - For each recorded state and move in the game:
        - Reward the selected move by adding beads
        to the corresponding matchbox.
If MENACE lost:
    - For each recorded state and move in the game:
        - Penalize the selected move by removing
        beads from the corresponding matchbox.
If the game was a draw:
    - Optionally, provide a smaller reward
    (e.g., 1 bead per move).
```

II. Binary Bandit Problem (Stationary Rewards)

This pseudocode follows the epsilon-greedy algorithm to decide which action to take and update the reward estimates.

A. Initialize the Binary Bandit

```
- Set the true probability of success for Action 1
(p1) and Action 2 (p2).
- Set initial estimates for both actions
(Q1 = 0, Q2 = 0).
- Set counters for both actions (N1 = 0, N2 = 0).
- Set the exploration rate epsilon.
```

B. Repeat for Each Time Step

```
1. Choose an action:
    - With probability epsilon,
```

```
    randomly select an action (exploration).
    - With probability (1 - epsilon),
    select the action with the highest
    estimated reward (exploitation).

2. Observe the reward:
    - If Action 1 was selected,
    generate a reward of 1 (success)
    with probability p1, else 0 (failure).
    - If Action 2 was selected,
    generate a reward of 1 (success)
    with probability p2, else 0 (failure).

3. Update the reward estimate for the
selected action:
    - If Action 1 was chosen:
        Increment N1 (the count of times
        Action 1 has been selected).
        Update Q1 using:
        Q1 = Q1 + (1/N1) * (reward - Q1).
    - If Action 2 was chosen:
        Increment N2 (the count of times
        Action 2 has been selected).
        Update Q2 using:
        Q2 = Q2 + (1/N2) * (reward - Q2).

4. Continue until the stopping condition is
met (e.g., after 1000 steps).
```

III. 10-Armed Bandit with Non-Stationary Rewards

In this section, we modify the epsilon-greedy algorithm to account for non-stationary rewards.

A. Initialize the 10-Armed Bandit

```
- Set the true reward for each of the
10 actions to an initial value (e.g., 0).
- Set initial reward estimates for all actions
(Q[i] = 0 for i = 1 to 10).
- Set counters for each action
(N[i] = 0 for i = 1 to 10).
- Set the exploration rate epsilon.
- Set the step-size parameter alpha
(for the exponential moving average).
```

B. Repeat for Each Time Step

```
1. For each action i (1 to 10):
    - Update the true reward for action i by
    adding a normally distributed increment
    with mean 0 and standard deviation 0.01.

2. Choose an action:
    - With probability epsilon,
    randomly select an action (exploration).
    - With probability (1 - epsilon),
    select the action with the
    highest estimated reward (exploitation).

3. Observe the reward for the chosen action:
    - The reward is stochastic based on the
    true reward of the chosen action.

4.Update the reward estimate for the chosen action:
    - Use an exponential moving average to update
    the estimate:
    Q[action]=Q[action]+alpha*(reward - Q[action])

5. Continue for at least 10,000 steps.
```

C. After Completing the Time Steps

```
- Analyze whether the agent was able to adapt to
the changing rewards.
- Plot the performance(e.g,total rewards over time,
frequency of choosing each action).
```

IV. Modified Epsilon-Greedy Algorithm for Non-Stationary Rewards

A. Algorithm Steps

1. Initialize Action Values and Step-Size:
   - Instead of using a standard sample average for updating the value estimates, we use a constant step-size parameter (alpha) that applies more weight to recent rewards.
2. Action Selection (Epsilon-Greedy):
   - At each time step, the agent selects an action using the epsilon-greedy method:
     - With probability epsilon, the agent explores by selecting a random action.
     - With probability $1-\epsilon$, the agent exploits by selecting the action with the highest estimated value.
3. Non-Stationary Rewards:
   - The true reward for each action changes over time following a random walk (normally distributed increment with mean 0 and standard deviation 0.01).
4. Update Action-Value Estimates:
   - After selecting an action and observing the reward, the agent updates the estimated value of the chosen action using an exponential moving average:

$$Q[action] = Q[action] + \alpha \cdot (reward - Q[action])$$

Here, alpha is a constant step-size parameter that gives more weight to recent rewards.

5. Repeat for Many Time Steps:
    - The agent interacts with the bandit for at least 10,000 time steps to allow sufficient learning and adaptation to the non-stationary environment.
6. Analysis:
    - After 10,000 time steps, evaluate whether the agent has been able to track the changing rewards and latch onto the best actions.