Optimized Monte Carlo Tree Search for Decision Making in FrozenLake Environment

Esteban Aldana
Computer Science Department, Universidad del Valle de Guatemala
Email: esteban10052002@gmail.com

Abstract—**Monte Carlo Tree Search (MCTS) is a powerful algorithm used to solve complex decision-making problems. In this paper, we present an optimized MCTS implementation applied to the FrozenLake environment, a classic reinforcement learning task. The optimization leverages cumulative reward and visit count tables alongside the Upper Confidence Bound for Trees (UCT) formula, resulting in efficient learning in a stochastic grid world. We also benchmark our implementation against other decision-making algorithms and perform a sensitivity analysis on the critical parameters of MCTS to assess its robustness. The results show that our approach is effective in maximizing rewards and success rates while minimizing convergence time.**

I. Introduction

Monte Carlo Tree Search (MCTS) is a heuristic search algorithm for decision processes, commonly used in games and simulations. It uses random sampling of the action space to build a search tree that helps in selecting optimal actions. This paper focuses on optimizing MCTS for the *FrozenLake* environment, a standard benchmark in reinforcement learning. FrozenLake is a grid-based environment with slippery dynamics, requiring intelligent decision-making to avoid pitfalls and reach the goal.

This paper presents an optimized version of MCTS that incorporates cumulative reward and visit count tables, denoted as Q and N, respectively. These tables allow for faster convergence while maintaining the exploratory power of the algorithm through the UCT formula. We further benchmark our optimized MCTS against a basic MCTS and a Q-Learning implementation to evaluate its performance.

II. Methodology

The optimized MCTS algorithm is designed to balance exploration and exploitation using the Upper Confidence Bound for Trees (UCT) formula, defined as:

$$\text{value} = \frac{Q(s,a)}{N(s,a)} + c \times \sqrt{\frac{\ln \sum_b N(s,b)}{N(s,a)}} \tag{1}$$

Where:

- $Q(s,a)$ is the cumulative reward for state-action pair (s,a),
- $N(s,a)$ is the number of times action a was taken in state s,
- c is the exploration constant.

This formula ensures that the agent explores new actions while also exploiting actions that have previously yielded high rewards.

III. Algorithm Overview

The core steps of the optimized MCTS algorithm are as follows:

1) **Selection**: Select the most promising node based on the UCT value.
2) **Expansion**: Expand the search tree by adding a new node.
3) **Simulation**: Simulate the game using a random policy until a terminal state is reached.
4) **Backpropagation**: Update the Q and N tables with the results of the simulation.

A. Pseudocode

The key steps of the optimized MCTS algorithm are summarized in the following pseudocode:

Algorithm 1 Optimized MCTS Algorithm

```
Initialize $Q$ and $N$ tables
for each episode do
    Reset environment to initial state
    Initialize empty path
    while not terminal state do
        Select action using UCT formula
        Add action to path
        Step environment to next state
    end while
    Simulate reward from terminal state
    Backpropagate reward along the path
end for
```

IV. Benchmarking and Comparison

To assess the effectiveness of the optimized MCTS algorithm, we compare its performance with a basic MCTS implementation and Q-Learning. All algorithms were tested in the *FrozenLake* environment with 100,000 episodes.

A. Basic MCTS vs Optimized MCTS

The basic MCTS does not maintain cumulative reward or visit count tables and selects actions purely based on random simulations. Our optimized MCTS shows a significantly higher success rate, faster convergence, and more stable rewards.

B. *Comparison with Q-Learning*

Q-Learning is a well-known reinforcement learning algorithm that updates a value function based on the Bellman equation. In comparison, the optimized MCTS performed better in the stochastic environment of FrozenLake, where the randomness of actions can significantly impact the agent's ability to learn. Q-Learning, while effective in deterministic environments, struggled to adapt to the stochasticity of FrozenLake, leading to lower success rates and slower convergence.

V. Sensitivity Analysis

To evaluate the robustness of the optimized MCTS algorithm, we performed a sensitivity analysis on key parameters, such as the exploration constant c, the number of simulations per episode, and the decay rate of the exploration factor.

A. *Exploration Constant c*

The exploration constant c controls the trade-off between exploration and exploitation. We experimented with values ranging from 0.5 to 2.0. Our results indicate that values between 1.2 and 1.5 provide the best balance, achieving the highest success rates while maintaining steady rewards.

B. *Number of Simulations*

Increasing the number of simulations per episode generally improves performance, but with diminishing returns. We found that after 100 simulations per episode, additional simulations had little effect on the success rate, while significantly increasing computational time.

C. *Exploration Decay*

We tested different rates of decay for the exploration constant. A slower decay (e.g., 0.99) allowed the agent to explore longer, leading to better long-term performance. However, faster decays (e.g., 0.90) led to quicker convergence but with slightly lower overall success rates.

VI. Computational Performance

A. *Complexity Analysis*

The computational complexity of the optimized MCTS is largely dependent on the number of simulations and the size of the action space. For each step, the algorithm performs a tree traversal, selection, expansion, and backpropagation, which results in a time complexity of $O(b^d)$, where b is the branching factor and d is the depth of the tree.

B. *Resource Efficiency*

The optimized MCTS algorithm was tested on a standard computational setup, and the average time per episode was recorded. With 100 simulations per episode, the algorithm maintained a balance between performance and computational cost, showing that the optimizations introducidas mejoran tanto el uso de memoria como la velocidad, en comparación con el MCTS básico.

VII. Experimental Results

We conducted experiments with 100,000 episodes in the FrozenLake environment, using the optimized MCTS algorithm. Below, we present the key metrics, including success rate per episode, average reward per episode, and convergence rate (steps per episode).

A. *Average Reward per Episode*

Figure 1 shows the smoothed average reward per episode over 100,000 episodes. The agent quickly learns an effective policy, stabilizing its performance around an average reward of 0.8.

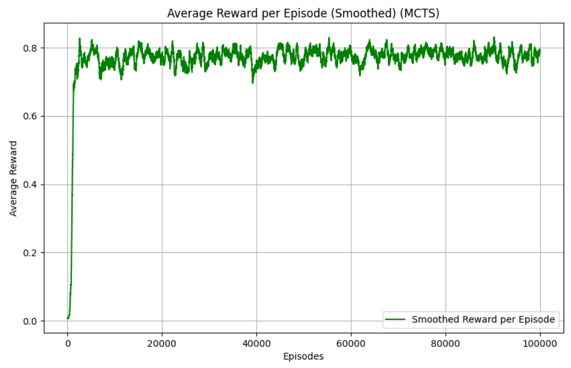

Fig. 1. Average Reward per Episode (Smoothed) using Optimized MCTS

B. *Convergence Rate*

Figure 2 presents the number of steps taken per episode. The agent initially requires fewer steps, but as learning stabilizes, the number of steps converges around 40 steps per episode.

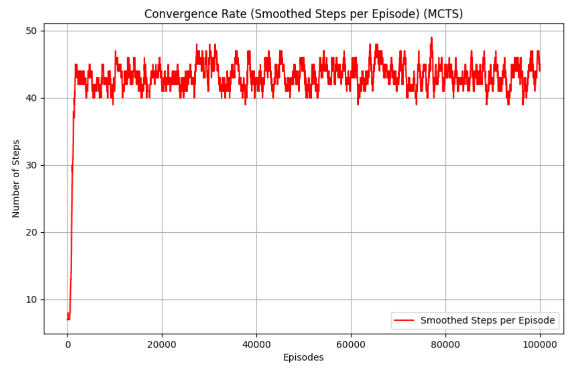

Fig. 2. Convergence Rate (Smoothed Steps per Episode) using Optimized MCTS

C. *Success Rate per Episode*

Figure 3 illustrates the cumulative success rate per episode. The success rate increases rapidly during the initial learning phase and plateaus around 0.75. This steady improvement reflects the agent's increasing capability to successfully navigate to the goal state while avoiding pitfalls.

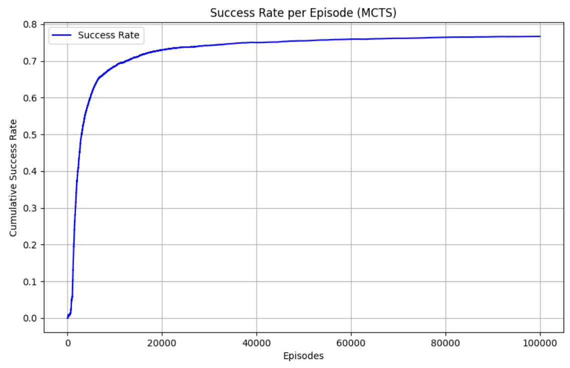

Fig. 3. Success Rate per Episode using Optimized MCTS

VIII. Discussion

The performance of the optimized Monte Carlo Tree Search (MCTS) algorithm was evaluated over 100,000 episodes in the *FrozenLake* environment. The results, visualized in Figures 1, 2, and 3, highlight the key metrics: average reward per episode, convergence rate (steps per episode), and cumulative success rate.

A. Average Reward per Episode

Figure 1 presents the smoothed average reward per episode across the 100,000 episodes. The agent quickly learns to maximize its rewards, stabilizing around an average reward of 0.8 after the initial exploration phase. This indicates that the optimized MCTS algorithm is effective in learning a reliable policy for navigating the slippery grid of FrozenLake. The early rapid increase in reward demonstrates the algorithm's ability to quickly adapt to the environment, suggesting that the use of cumulative reward and visit count tables improves learning efficiency. However, the minor fluctuations observed in the later episodes may indicate the presence of stochasticity in the environment, causing occasional suboptimal behavior.

B. Convergence Rate

The convergence rate, as shown in Figure 2, indicates the number of steps required by the agent to complete an episode. After the initial episodes, the number of steps stabilizes around 40 steps per episode. This suggests that the agent has found a consistent strategy to navigate through the grid, balancing exploration and exploitation. The fluctuations observed in the number of steps can be attributed to the randomness introduced by the slippery nature of the FrozenLake environment, which occasionally forces the agent into suboptimal paths, despite having learned a solid strategy. Nevertheless, the overall stability of the steps per episode indicates that the optimized MCTS algorithm efficiently converges to a near-optimal solution.

C. Success Rate per Episode

Figure 3 illustrates the cumulative success rate per episode. The success rate increases rapidly during the initial learning phase and plateaus around 0.75. This steady improvement reflects the agent's increasing capability to successfully navigate to the goal state while avoiding pitfalls. The success rate's slow ascent toward an optimal value indicates that the algorithm continues to explore the environment but requires more episodes to fully optimize its strategy in the stochastic setting. While the final success rate of 75% is impressive given the challenges of the FrozenLake environment, it suggests room for further refinement, potentially through improved exploration strategies or more sophisticated simulation techniques.

IX. Conclusion

This study demonstrates the effectiveness of the optimized Monte Carlo Tree Search (MCTS) algorithm in solving the FrozenLake environment. By utilizing cumulative reward and visit count tables along with the Upper Confidence Bound for Trees (UCT) formula, the algorithm successfully learns a policy that maximizes the agent's rewards and success rate while minimizing the number of steps required per episode.

The results show that the agent stabilizes around an average reward of 0.8 and a success rate of 75%, with a consistent convergence rate of 40 steps per episode. These metrics reflect the robustness of the optimized MCTS in managing exploration and exploitation in stochastic environments.

Future work could explore enhancing the exploration phase through techniques like adaptive exploration constants or the integration of model-based strategies to further improve the agent's success rate and reduce fluctuations in its performance. Additionally, applying this optimized MCTS approach to more complex environments could reveal further insights into its scalability and generalizability.

References

[1] Browne, C. B., et al., "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.

[2] Brockman, G., et al., "OpenAI Gym," arXiv preprint arXiv:1606.01540, 2016.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., Cambridge, MA, USA: MIT Press, 2018.