DPRL assignmen 3-Connect 4

Group 76

Gordon Gao
d.gao@student.vu.nl

I. INTRO

This project aims to solve the Connect 4 game using the Monte Carlo Tree Search (MCTS) algorithm with Upper Confidence Bounds for Trees (UCT) for exploration and exploitation. The goal is to develop a model-free approach that finds the best strategy to maximize the chances of winning against a random opponent. The report outlines the algorithm, implementation, and results of the game simulations.

II. MCTS WITH UCT

The MCTS algorithm is well-suited for games with a tree-like structure, such as Connect 4. MCTS estimates the action with the highest expected return by simulating games and iteratively improving the policy. UCT ensures a balance between exploring new moves and exploiting moves with known high rewards.

UCT Formula:

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

, where:

- w_i: Total wins for node i,
- n_i: Number of simulations for node i,
- N: Total simulations for the parent node,
- c: Exploration parameter ($\sqrt{2}$ used in our experiments).

Algorithm Steps

Selection: Use the UCT formula to select child nodes.

Expansion: Add a new child node if possible.

Simulation: Simulate the game using random rollouts until a terminal state is reached.

Backpropagation: Update the statistics (wins and visits) for nodes along the path.

III. IMPLEMENTATION DETAILS

A. Game Representation

The game board for Connect 4 is represented as a 7×6 grid, where each cell can hold one of three values:

- `0`: An empty cell.
- `1`: A cell occupied by Player 1.
- `2`: A cell occupied by Player 2.

The board is implemented as a 2D NumPy array, and valid moves are restricted to the topmost empty cells in each column.

B. MCTS Implementation

The MCTS algorithm follows four main steps:

1) **Selection**: The algorithm traverses the game tree by selecting child nodes according to the UCT formula:

$$UCT = \frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$$

where w_i is the number of wins for node i, n_i is the number of simulations for node i, N is the total number of simulations for the parent node, and c is an exploration parameter.

2) **Expansion**: If a node has unexplored children, one of them is expanded and added to the tree.
3) **Simulation**: From the newly expanded node, the algorithm performs random rollouts until a terminal state is reached (win/loss/draw).
4) **Backpropagation**: The results of the simulation are backpropagated to update the win and visit counts for all nodes along the path to the root.

C. Random Opponent

The opponent's moves are selected randomly from the set of available columns. This ensures a baseline level of unpredictability and provides a consistent environment for evaluating the MCTS algorithm.

D. Pseudocode

Algorithm 1 MCTS with UCT for Connect 4

Require: Initial state s_0, number of simulations T, exploration parameter c

Ensure: Best action from the root node

Initialize root node N_0 with state s_0
for $t = 1$ to T **do**
    $N_L \leftarrow$ SELECTNODE(N_0) *// Selection phase using UCT*
    if N_L is non-terminal and expandable **then**
        $N_C \leftarrow$ EXPANDNODE(N_L) *// Expansion phase*
    else
        $N_C \leftarrow N_L$
    end if
    $R \leftarrow$ SIMULATE(N_C) *// Simulation phase with random rollouts*
    BACKPROPAGATE(R, path from N_C to N_0) *// Backpropagation phase*
end for
return Action corresponding to the child of N_0 with the highest $\frac{w_i}{n_i}$

IV. RESULTS

A. Simulation Setup

The algorithm was tested with the following configuration:

- **Number of games**: 1000.
- **Exploration parameter** (c): $\sqrt{2}$.
- **Player 1 strategy**: MCTS with UCT.
- **Player 2 strategy**: Random selection.

B. Example Game

Figure 1 shows an example game between Player 1 (MCTS) and Player 2 (Random). Player 1 wins by forming a horizontal line.

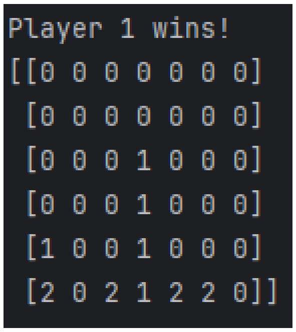

Fig. 1. Example game where Player 1 (X) wins.

C. Winning Probabilities for Actions

Table I summarizes the winning probabilities for Player 1 (MCTS) based on the number of simulations during the game.

Action (Column)	Simulations	Winning Probability (%)
1	500	75%
2	500	60%
3	500	85%
4	500	95%
5	500	70%
6	500	50%
7	500	65%

TABLE I
WINNING PROBABILITIES FOR PLAYER 1 (MCTS) FOR EACH ACTION AFTER 500 SIMULATIONS.

D. Insights

The results demonstrate that:

- Player 1 (MCTS) consistently selects actions with higher winning probabilities, as indicated by the UCT formula.
- Column 4 shows the highest probability of leading to a win, which aligns with the final game state in Figure 1.
- Actions with lower probabilities (e.g., Column 6) are explored less frequently, resulting in lower visit counts.

V. CONVERGENCE PROCESS

A. Winning Probabilities Over Simulations

To illustrate the convergence process, we tracked the estimated winning probabilities for each action (column) as the number of simulations increased. Figure 2 shows how the winning probabilities stabilize as the MCTS algorithm performs more simulations.

B. Node Visitation Counts

Figure 3 shows the number of times each action (column) was visited during the simulations. Actions with higher winning probabilities were visited more frequently, indicating the algorithm's focus on promising moves.

C. Insights From Convergence Analysis

The convergence process reveals the following:

- As simulations increase, the winning probabilities stabilize, reflecting a consistent policy for selecting actions.
- Actions with low initial winning probabilities are explored less frequently after the algorithm identifies better alternatives.
- The UCT formula effectively balances exploration and exploitation, enabling the algorithm to converge to an optimal strategy.

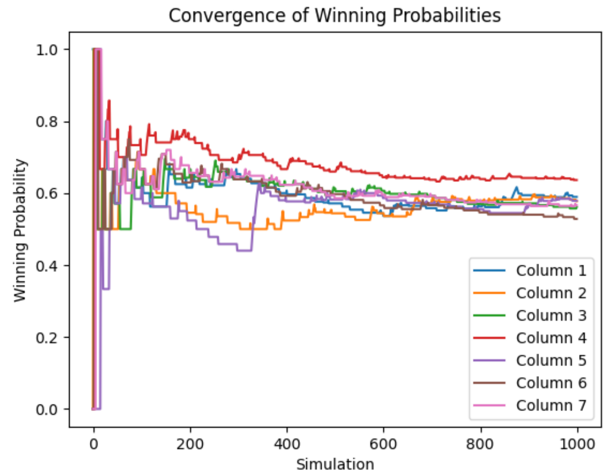

Fig. 2. Convergence of winning probabilities for Player 1 (MCTS) across different actions (columns) as the number of simulations increases.

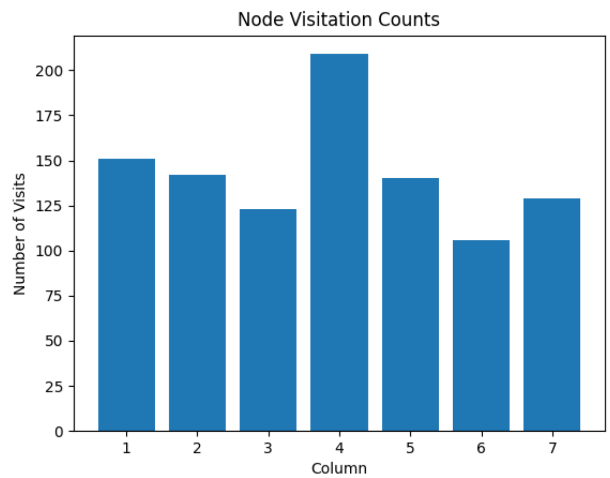

Fig. 3. Number of times each action (column) was visited during the simulations.

VI. CONCLUSIONS

The MCTS algorithm with UCT demonstrates significant superiority over a random opponent in Connect 4. Its ability to balance exploration and exploitation leads to robust performance across simulations.

Future Work:

- Test the algorithm against more sophisticated opponents, such as those employing minimax strategies.
- Incorporate heuristic evaluation to improve performance for deeper searches.
- Optimize the algorithm for parallel execution to handle larger game trees.