Connect 4: Principles and Techniques

Amy Shen, Pranav Sriram, & Catherina Xu | CS 221 | Fall 2017

PROBLEM

Connect Four is a popular board game. Players take turns dropping colored tiles into a grid, with the goal of occupying four slots in a row. This project aims to explore various AI techniques to play Connect Four.

MOTIVATION

Connect Four is a solved game; the starting player has a winning strategy under optimal play.

However, playing the optimal strategy is not trivial. There are 10^{13} possible board positions in a standard 6 x 7 board, making it infeasible to store a move tree in memory.

As avid Connect 4 players, we're interested in using AI to create a gameplay agent (in other words, a worthy opponent!)

CHALLENGES

Limited compute power. We don't have the ability to explore most of the game tree.

Heuristic development. It's not always obvious how to translate high-level strategies into concrete heuristic functions to evaluate a board.

Evaluation and validation. It's difficult to evaluate the effectiveness of an agent, other than to play the game many times against an opponent. An additional metric we utilize is the total # of moves played.

APPROACHES, ALGORITHMS & METHODS

Setup

Our testing harness accommodates five types of "players" -- a human, a naive player, a Minimax agent, a Monte Carlo Tree Search agent, and a TD Learning agent -- and pits them against each other in a end-to-end gameplay GUI.

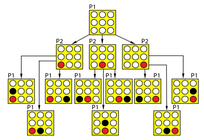
```
Computer's reward for the move: 105
Displaying board:
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 B 0 0 0 0 0
0 R 0 0 0 0 0
0 R 0 0 0 0 0
0 R R B B B 0
```

Agent 0: Naive

The agent randomly places a piece to the left, right, or above the last-played piece by the opponent.

Agent 1: Minimax

The minimax tree alternates between maximum and minimum layers corresponding to P1 and P2 (opponent), respectively.



Depth limitation and alpha-beta pruning: We utilized both methods to decrease the runtime of the algorithm.

Evaluation function: The value function v1 is based on previous work. We created another value function, v2, with additional features such as the number of spaces around "streaks."

$$v1(state) = \begin{cases} -100000 & \text{if opponent has four in a row} \\ 100000 * (\#\text{-4-in-a-rows}) + 100 * (\#\text{-3-in-a- rows}) + (\#\text{-2-in-a-rows}) & \text{else} \end{cases}$$

Agent 2: Monte Carlo Tree Search (MCTS)

MCTS builds up a search tree of possible move sequences and simulates each one multiple times to determine the expected outcome. We implemented the **UCT (Upper Confidence Bounds for Trees)** variant of MCTS, which selects the move for which $\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}}$ is maximized, where w_i = number of wins for the node after move i, n_i = number of simulations for the node after move i, N_i = total simulations after move i, and c is an adjustable parameter. MCTS has been proven to converge to minimax, but it converges slowly and has the advantage of not needing an evaluation function.

Agent 3: Temporal Difference (TD Learning)

We implemented TD Learning using a linear evaluation function on the raw board state. TD learning works by using reflex policies based on an evaluation function learned by playing against itself repeatedly. The board is represented as a vector in 42-dimensional space, where red pieces are represented as 1, black pieces by -1, and empty squares by 0.

Acknowledgements Thank you Percy, Stefano, Bardia, and the rest of the CS221 staff!

GAMEPLAY RESULTS

Minimax vs Naive

	Wins	Number of moves before win
Minimax depth 3	100	12.84
Naive	0	n/a

Minimax vs. TD Learning

TD vs Minimax (depth 1)	Win percentage
TD-Learning	58.5
Minimax (depth 1)	39.5

TD vs Minimax (depth 2)	Win percentage
TD-Learning	2
Minimax (depth 2)	98

All win percentages are calculated over 100 trials (with draws discarded).

Minimax vs. Minimax (with different depths)

	Win percentage	Avg number of moves before win
Minimax depth 5	64	24.28
Minimax depth 3	23	34.52

Minimax (depth 4) vs. Monte Carlo Tree Search (budget 1000)

	Red win percentage	Number of moves before win (Red)	Black win percentage	Number of moves before win (Black)
Minimax depth 4	71.4	16.17	50.5	19.92
Monte Carlo Tree Search, budget 1000	49.5	23.65	28.6	24.82

Minimax (depth 3) vs. Monte Carlo Tree Search (budget 1000)

	Red win percentage	Number of moves before win (Red)	Black win percentage	Number of moves before win (Black)
Minimax depth 3	69	16.17	57.14	18.91
Monte Carlo Tree Search, budget 1000	42.86	23.60	31	24.82

ANALYSIS

Minimax performance increases with depth, as expected. Higher depths allow for more "lookahead" in selecting the optimal move.

MCTS generally performs worse than Minimax with the number of simulations we allowed the MCTS algorithm to perform. MCTS may perform better with more simulations, but that would require significantly more time and computational power.

TD-Learning outperforms minimax with depth 1, which implies that the evaluation function learned by the TD-Learning agent is better than the hardcoded evaluation used by minimax.

Future Work

TD-Learning with a deep convolutional neural network. CNNs seem to be a good fit for the value-function approximation task due to the central role of spatially contiguous patterns in Connect Four strategies. The learned parameters can then be used to improve the minimax evaluation function.