Monte Carlo Tree Search Agent in Reversi

Khang Vu, 301255281, khangv

CMPT 310 Spring 2020

SFU

I. **Source Code Files:**

The source code contains these files:

- *main.cpp :* contains logic that prints out the game's user interface, takes and checks user input, etc.

- *reversi.cpp* and *reversi.h* : contains game logic of reversi that includes logic for calculating available moves each turn, updating the board with each move by flipping tiles, etc.

- *mcts.cpp* and *mcts.h* : contains logic for Monte Carlo Tree Search Agent that includes logic for tree data structure, functions for playouts, updating tree with a playout's result, 2 versions of MCTS, etc.

Compiling these files should give an executable that can be run to play the game.

II. **Implementation Overview:**

The logic contained in *reversi.cpp* and *.h* is based on the rules of Reversi. The functions implemented have brief descriptions in the header file and they are quite self-explanatory.

The files *mcts.cpp* and *mcts.h* contain the logic for Monte Carlo Tree Search. There is a tree data structure with each node storing a pointer to its parent node, list of pointers to children nodes, and data of playouts that include number of wins, losses, plays and draws.

There are 2 versions of MCTS implemented in my code. The first is contained in functions *randomPlayout()* and *randomMCTS()* that uses random move at each step of a playout. The best move after a certain number of playouts is chosen by having the highest average of (wins/plays).

The second version of MCTS does not use random playouts but a version of MCTS described in *Analysis of Monte Carlo Techniques in Othello,* Section 3.3 by Ryan Archer[1]. The algorithm for playouts is as follows:

Step 1. Play each move once.

Step 2. Loop: Play move j that maximizes $X_j + \sqrt{\frac{2 \log n}{T_j(n)}}$ where X_j is win average (wins/plays), n is the total number of plays, $T_j(n)$ is the number of plays of a move.

[1] Can be downloaded from
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.468.8160&rep=rep1&type=pdf

This function, named UCB1, has two properties:

- if X_j is large, it means that the move has potential and should be explored more.

- if $\sqrt{\frac{2 \log n}{T_j(n)}}$ is large, it means that the move has not been explored for a while, and should be explored more.

In general, this version of MCTS explores good options more often, but still explores adequately all options.

III. **Performance:**

Checking for performance is rather difficult since I am a complete beginner at Reversi and do not know how to evaluate performance of each MCTS agent. However, below are results of each agent using 500 playouts per move against myself, and easy and medium AI opponents on site mathisfun.com[2]. The data is limited due to the time it takes me to manually input each move on both the program and the website.

Opponent	pureMCTS (wins./plays)	ucbMCTS (wins./plays)
Myself	4/5	5/5
Easy difficulty on site mathisfun.com	2/3	3/3
Medium difficulty on site mathisfun.com	0/3	2/3

Overall, pureMCTS performs adequately, but worse than ucbMCTS. This is confirmed as I simulate games between the 2 agents. Agent ucbMCTS won all 40 games against pureMCTS.

[2] https://www.mathsisfun.com/games/reversi.html