Project 3 Report

郭庭佑

10th January, 2023

1 CODE REWRITE

Thanks to TA provided code :). It is much clean so I rewrite my code of the part of MCTS completely, so that the implementation of the following is not hard to do.

2 MCTS

For each tree node s, Q_s is the win rate of who takes turn at s, so in opponents nodes, MCTS selection prefers actions beneficial to opponents. This makes sense (like min-max search) and is mentioned in one random teacher's slide, while it's not implemented in TA provided sample code, nor mentioned in main part of slides.

3 RAVE

$k = 100$

4 BITBOARD

A board is stored in 4 `uint128`. The first 2 is black pieces and white pieces. The last two are legal moves of black and white, so when we waat to place a piece on a board, we check the masks easily, and when a piece is placed, we update the masks. I think this can do faster when simulation, since we only check legal moves instead of all moves and check liberty again and again. But I don't compare them anyways.

5 PARALLELIZATION

I implement lock-free tree parallelization. In the selection stage, add the virtual loss, i.e., visit count and rave visit count.

Pre-allocated space for nodes is set to 10^6. That is, `std::vector::reserve`.

Note that expend fails when the node is a terminal or the space is full. It is important to prevent thinking-too-long attack. I mean if the opponent takes to much time in one move on purpose, my MCTS may memory exceed and explodes.

Also, my MCTS will still proceed in opponent's round.

Lock-free tree parallelization maybe a pit! A little bit too large or too small c dramatically reduce the performance, and finding the best c is a waste of time.

6 Tree Recycle

For each move, I don't release the MCTS tree, instead, when the next time taking action. I set the root be the new state.

7 Thinking when opponent thinking

As the title said, after taking action $s \xrightarrow{a} s'$, MCTS search continues to search s'. Adding this with tree recycling, sometimes the saved tree size can be over millions.

8 Time management

I use enhanced time management, where $c = 20$, $maxply = 20$. The approximated MCTS per millisecond is 200.

Also, I use EARLY-C. It is easy to implement but in mock contest, I found nobody do this, because they think too long in the endgame.

9 Simulation Balancing

I attempt to do simulation balancing. The score of a move is the score of s' minus the score of s. Anyways my conclusion is that my design of the score may be too complicated, so the simulation is too slow, and the performance is super bad. I was thinking a new method that takes advantage of bitboard, which will be much faster. But I win the mock contest, so.... But I got second place today :(, maybe I should finish this.