Fall 2021
COMS W4995: Parallel Functional Programming
Project Proposal

Gomokururu: A Gomoku Solver

Kevin Xue, Andreas Cheng
{kx2154, hc3142}@columbia.edu

November 23, 2021

1 Background

Gomoku is a chess game where two players place black and white chess in turn on a board. The game board is formed by $N \times N$ horizontal lines. N is traditionally 15 or 19, where N=19 is an older standard that people Gomoku on a Go Board. Players should place chess on the intersection of these lines. The player assigned to the black chess plays first. The goal is to form a consecutive shape of 5 chess, which could be vertical, horizontal, or diagonal.

2 Goals

The major goal of this project is to design a Gomoku AI. AI of this type usually uses **minimax and alpha-beta pruning**. The size of the chessboard is big (the choice space of each move), and the number of moves is also at least 20-30 (one has to have at least 5 chess on the board to win). Therefore, the board states needed for the AI algorithm to work require a great amount of computation, making parallelism important. Without parallelism, players will have to wait much longer than if they play with human players.

We have found several sequential Gomoku AI implementations in various languages.

1. https://github.com/sowakarol/gomoku-haskell. This is an AI written in sequential Haskell. Although this AI is fast (2-3 seconds response time), it is ill-designed. We ran the algorithm, and a player can win in 5 steps. Even if the player places 4 consecutive chess, the AI will not attempt to stop the expansion.
2. https://github.com/lukesalamone/gomoku-2049. This is an AI written in JavaScript. Although it is not ill-designed, it is not responsive. After around 10 moves, the player must wait a very long time for the computation to be done.

Our goal is to do better than all of the algorithms above and create an AI that could play well with the player. That is, the player faces similar difficulty and wait a similar amount of time as if they play with a human player. We will use ThreadScope and timers to determine whether our implementation is better.

3 Design

Depending on time, we plan to do the following:

1. A non-interactive sequential Haskell AI, in which two AI's play in turn, implement the algorithm used in the JavaScript implementation. In this stage, we will understand the AI algorithm and ideally identify which parts can be implemented with parallelism.
2. Use parallelism to improve the above AI, keeping it non-interactive. We could submit the work of this stage as a complete project. The metrics can be tested out by replicating the moves performed by one of the AI as a player in the baseline implementations.
3. Make the AI interactive – add a player.
4. Improve the algorithm and optimize the parallelism so that it could be faster and smarter.

4 Benchmarking

To benchmark the performances of different Gomoku solvers, we would measure the explored nodes and depths within a limited time, like 5 seconds.

We have several ideas on how to automate the benchmarking process as follow:

1. Let the solvers play against each other, bounding their decision time for like 2 seconds, and see how many nodes/depth they can explore on average.
2. Let the solvers solve several designated boards and compare their performance on a single move. In this case, we only need one solver.