AlphaGambit

David Zhang (dwz2107) Anthony Zhou (az2681) Abhishek Chaudhary (ac5003)

Background

People love playing chess, but don't always have someone to play with. Fortunately, people can now play chess with an algorithm, called a chess solver. A chess solver takes in the position of pieces on a chessboard and returns the best next move, making it the core component of any simulated gameplay.

Garry Kasparov plays IBM's Deep Blue. In 1997, the chess engine defeated the world chess champion.

Parallelism Rationale

Chess is a complex game demanding sophisticated approaches to evaluate and predict optimal moves. The game presents the ideal challenge with its sheer number of possible moves and positions making exhaustive evaluation infeasible without leveraging parallel processing. Traditional sequential evaluation methods become inefficient as the complexity of the game grows; as the move-tree in chess is vast, parallelism allows for faster and more efficient exploration of the tree leading to quicker decision-making processes.

Overview

In this project, we are implementing one such chess solver based on the minimax algorithm, using parallelism and alpha-beta pruning to speed up results. As part of the chess solver, we will develop the following functions:

1. **Position generation:** Given a chess board, determine the possible next moves up to a certain depth.
2. **Scoring function:** Given a chess board, determine the "quality" of that position using heuristics
3. Use alpha-beta pruning to obtain a reasonably sized tree of possible moves which is evaluated using minimax to determine the optimal move.

Extension (tournament of heuristics): we parameterize certain heuristics (that is, give different weights to each heuristic or each piece) and make the scoring functions compete against each other to find the best one with a bracket system.

Algorithm Description

Leveraging a Chess Move Generator we determine the total set of possible moves up to a constant depth *d*. This gives us a tree of depth *d*, which we use as input for the following algorithms. In a sequential setup, these possible moves are removed using Alpha-beta pruning to determine which moves we want to build our tree on. In the parallel version, we use a combination of sequential and parallel design to first sequentially determine the alpha-beta pruning values for a subset of the move-tree and then run the evaluation of all other possible board positions using that set alpha-beta values in parallel to find the best next move.

We will define the scoring function using Stockfish's evaluation. This evaluation, also known as Main evaluation, uses a combination of factors not limited to material balance, piece mobility, pawn structure, king safety, etc. Using this evaluation gives a benchmark for comparing board states for min-max.

Benchmarking

We will implement and measure runtime for three versions of the game solver:

1. Sequential minimax
2. Sequential minimax with alpha-beta pruning
3. Parallel minimax with alpha-beta pruning

The main levers to change are tree depth and number of positions to evaluate. The determination of speed increases is by computation time to determine the best move from a given state for the three versions.

Tournament of Heuristics [Extension]

After all this chess solving, we might be left wondering: how good are these heuristics, really? Will our chess solver be able to defeat 7-year-old Magnus Carlsen? One way to test the solver quality would be to play it ourselves, but this would be limited by our own chess-playing ability.

Instead, we're going to have our chess solvers play each other. First, we define a set of heuristics, like the ones mentioned above. Each will have a weight assigned to it, as well as parameters inside of it, at first randomly assigned. Then, we make 1028 different chess solvers using random variations on the parameters, and have them play against each other in a bracket-style tournament. The solver that wins this tournament will represent the best parameters for the heuristics.