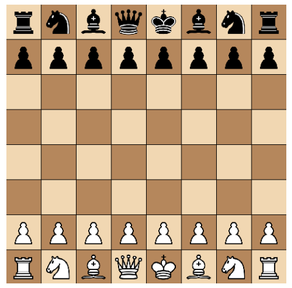
Generating Chess Tactics and Quantifying Perceived Difficulty

Robbie Selwyn

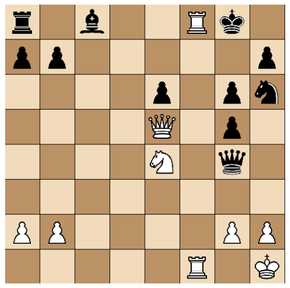
Stanford
Computer Science

Chess

- Chess is a 2-player game, played on an 8x8 board where each sides' goal is to checkmate the opponent (make it so that the opponent's king can't move)
- Each player begins with 16 pieces on the first two rows of the board and has to move them in the most efficient order possible to try and attack the opponent's king
- Players must maintain defense for one's own king as to not get checkmated first.



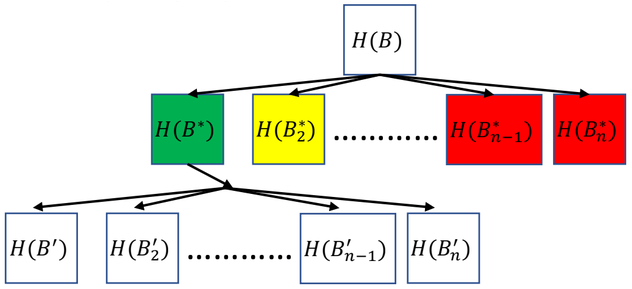
Starting Position



Checkmate

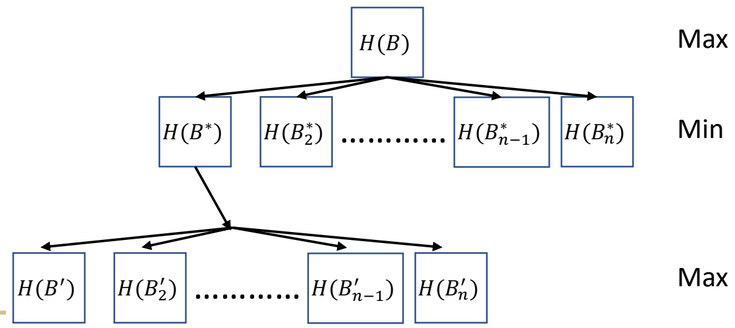
Recursively Quantifying Problem Difficulty

- Tactics are difficult because of **breadth** and **depth** (number of moves to consider and how deep to look)
- Algorithm Idea:
  - Compute $E^{MAX}[b]$ (the evaluation at **maximum depth** of the board at position b)
  - Count the number of moves that are close to the best move (differ by some constant)
  - Perform this evaluation at **every other depth**
  - Multiply the number of "close" moves at each depth by a constant, and sum up the numbers across the depths to get a difficulty score
- **Example Below**: The first node is the best node, the second node is good at lower depths, and the last 2 nodes are not good at any depth.
- The other two parts took longer, so this part is not completely implemented in code.



Developing the Engine

- The engine is dependent on minimax, a search algorithm for two-player games
- Steps of Minimax:
  - Build up a tree of all board states (each child of a node is the board with a possible move applied to it)
  - Fill in the **leaf** nodes with their associated board value (using a heuristic function)
  - Starting from the bottom of the tree and working the way up, set each node to be either the **maximum** of all the children, or the **minimum** of all the children, depending on the node.



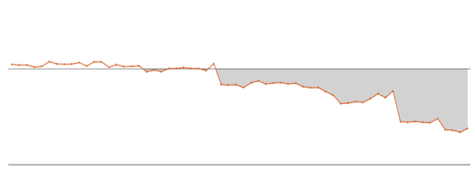
- Heuristic
  - Evaluates the quality of the position depending on factors including king safety, piece position, and which pieces are still on the board.
  - Returns a positive value if white is winning, negative value if black is winning
- Pruning and Efficiency
  - After just 5 steps with 30 possible moves at each step, the search tree is at $\mathbf{30^5 = 24.3}$ **Million**
  - Using **αβ-Pruning**, we can speed this up immensely
  - αβ-Pruning creates a feasibility region

    (α, β)

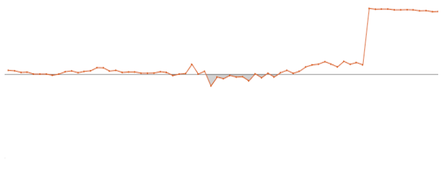
    in which we know nodes will be valid (allowing a substantial amount of the nodes to be removed)
- Further Time-Complexity Improvements
  - All of the node processing is done in a **multi-threaded** manner in order to speed up evaluation
  - Transposition tables (essentially memorization for minimax values) into the evaluation by using a custom Zobrist Hash function
  - A Zobrist function $Z(B)$ allows for efficient hashing of a full chess board.

Discovering Tactics

- In general, the game of chess has two different styles of play: **positional** and **tactical**.
- Playing **positional** chess involves placing your pieces on squares that tend to be stronger, and slowly fighting for an advantage.
- A **tactic** involves finding a sequences of moves that lead to a quick advantage (for example, allowing you to win a Knight)



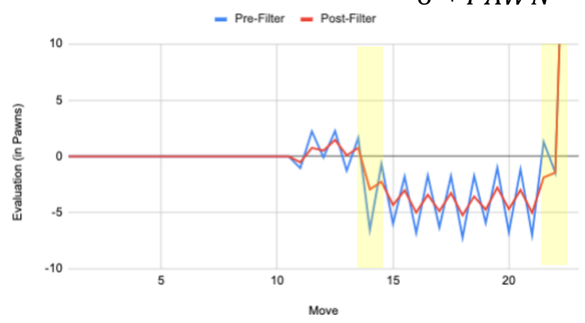
Positional Game (Kovalev vs. Nakamura)
(Analysis with Stockfish)



Tactical Game (Carlsen vs. Aronian)
(Analysis with Stockfish)

- To find tactics, we can look at the change in engine evaluation over time.
- Data Transformations: We **need to** smooth out and change the units of the engine evaluation.
- To do this, we use a moving Average, and divide by the value of a pawn to get the evaluation in terms of pawns

$$X_i = \frac{(3X_i + 2X_{i-1} + X_{i-2})}{6 * PAWN}$$



The pre-filtered and post-filtered evaluation for a sample game. Tactics highlighted

- Selection is done by looking at any instance where there is a **change** in evaluation (either way) over a certain margin **(when the position drastically improves)**

$$|X_i - X_{i-1}| > C_{DROP}$$

- However, we must also figure out where the tactic is "complete" (as in, all the moves to gain material have been played)
- To do this, it is possible to look at not just the best move from the chess engine, but also the second best move.
- By looking at the difference in the quality of the best move and the second best move, it is easy to figure out **if all the good moves have been played**.
- If we let $E[b]_i$ be the ith best result of the engine for a board then we just need to find the point at which

$$|E[b]_1 - E[b]_2| < C_{NOMINAL}$$