Total nodes expanded by minimax

\[
N(b, d) = 1 + b + b^2 + \ldots + b^{d-1} + b^d \\
b \cdot N(b, d) = b + b^2 + \ldots + b^{d-1} + b^d + b^{d+1} \\
b \cdot N(b, d) - N(b, d) = b^{d+1} - 1 \\
N(b, d) \cdot (b - 1) = b^{d+1} - 1 \\
N(b, d) = \frac{b^{d+1} - 1}{b - 1} \\
N(b, d) \approx \frac{b^{d+1}}{b - 1} = b^d \frac{b}{b - 1}
\]

Nodes expanded by iterative deepening

- Minimax work at depth d: \( b^d - \frac{b}{b-1} \)

\[
\frac{b^d}{b-1} + b^{d-1} \frac{b}{b-1} + b^{d-2} \frac{b}{b-1} + \ldots + b \frac{b}{b-1} + \frac{b}{b-1} \\
\frac{b}{b-1} (b^d + b^{d-1} + b^{d-2} + \ldots + b + 1) \\
\frac{b}{b-1} b^d = b^d \left( \frac{b}{b-1} \right)^2
\]

Advantages of iterative deepening

- How do I order nodes?
  - Hand analysis/heuristics
  - Use information from the previous iterations
- Start with principal variation
- History Heuristic
Move ordering

- Can we learn a good move ordering?
- Killer Heuristic:
  - Find the move that is causing the most cutoffs at each depth of the tree
  - Try it first

Move Ordering

- Killer heuristic just finds a single move
  - Can we generalize to all moves?
  - Simplistic “learning”

History Heuristic

- Each move has a score
  - Increment score whenever a move is the best move in a state (or causes a cutoff)
  - Score is $2^d$, where $d$ is the depth of the tree analyzed below the move
- Sort moves by score

Null move

- If one player is in a strong position, they could skip their turn and still win
  - Perform “null” move
  - Search to depth 2/3 ply shallower than required
    - If the value of the game is still better on previous branches, it’s a win
    - Otherwise re-search with the full tree
- Hop Step?
- zugzwang -- sometimes it’s better not to move
Quiescence search

- Quiescence = quiet
  - Searching to a fixed depth may not be advantageous
  - eg if a capture has just been made, and the capture response hasn’t
  - Extend search until position is quiet
    - eg no captures and no check

Horizon Effect

- A result of limited depth knowledge
- Something bad is about to happen, but find a way to delay it until it happens after the search depth
- May turn a minor problem into a catastrophic one
  - Make a bad move now to avoid a worse state by the horizon effect
  - The worse state still happens later

Using alpha-beta bounds

- When performing alpha-beta pruning, don’t have to start with alpha = -∞, beta = ∞
  - If you have evidence that the value is in a smaller range, can search with a tighter bound
    - Produces smaller tree
    - But, if the value returned is at the bounds of your range you have to re-search

Graphs versus Trees

- A graph is not a tree
  - Turning a graph into a tree results in exponential blow-up
- Transposition (hash) Table
  - Key: Minimax uses (almost) no memory
    - Use some memory to detect tree
    - When two paths transpose into the same state
Breakthrough example

• When is the first transposition in Breakthrough?
• Assume you are searching 8 ply
  • 3-ply into the game
  • Save 5-ply ($20^5 \approx 3.2$ million states)

Transpositions

• When should we look for transpositions?
  • Near the top of the tree
    • Large savings
    • Likely to find transpositions
  • When shouldn’t we look for transpositions?
    • Near the bottom of the tree
      • Minimal savings
      • Unlikely to find transpositions

Transpositions

• What is needed to test for transpositions?
  • Naïve - list of states, and a linear search
  • Better - tree of states log(s) search
  • Best - hash table

• What hash function should we use?