

Haskell Parallel Chess Engine

Nikolaus Holzer

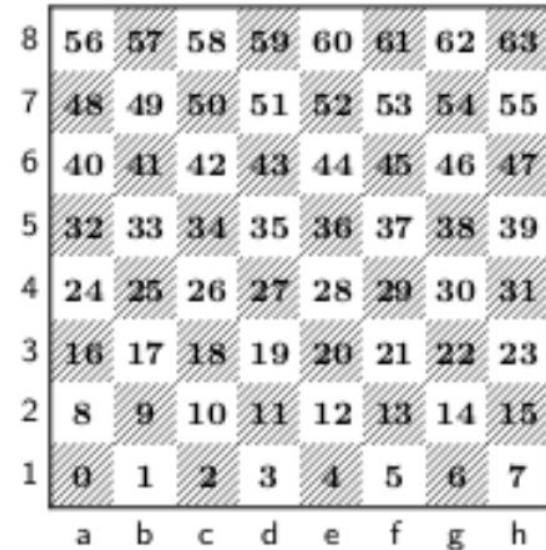
Introduction

Minimax based chess engine

Bitboards

Parallelization

Live demo



A diagram of an 8x8 chessboard with squares numbered from 0 to 63. The numbering starts at the bottom-left corner (a1) as 0 and proceeds row by row from left to right. The top-right corner (h8) is 63. The board is shaded in a checkerboard pattern, with squares where the sum of the row and column indices is even being shaded. The columns are labeled 'a' through 'h' at the bottom, and the rows are labeled '1' through '8' on the left side.

| | | | | | | | | |
|---|----|----|----|----|----|----|----|----|
| 8 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 7 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 6 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 5 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 4 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 3 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 2 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | a | b | c | d | e | f | g | h |

Figure 1: Chess board squares with the corresponding number

Bitboards

- Each square corresponds to a bit in a 64 bit word.
- We can easily map bits to squares
- Each piece can be shown as a 64 bit word

| | | | | | | | | |
|---|----|----|----|----|----|----|----|----|
| 8 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 7 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| 6 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| 5 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 |
| 4 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 3 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 2 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | a | b | c | d | e | f | g | h |

Bitboards

```
startpos :: Board
startpos =
  Board
  { pawnsWhite = 0x000000000000FF00 -- [a,h]2 entire row
  , pawnsBlack = 0x00FF000000000000 -- [a,h]7 entire row
  , knightsWhite = 0x0000000000000042 -- b1 (bit 1) and g1 (bit 6)
  , knightsBlack = 0x4200000000000000 -- b8 (bit 57) and g8 (bit 62)
  , bishopsWhite = 0x0000000000000024 -- c1 (bit 2) and f1 (bit 5)
  , bishopsBlack = 0x2400000000000000 -- c8 (bit 58) and f8 (bit 61)
  , rooksWhite = 0x0000000000000081 -- a1 (bit 0) and h1 (bit 7)
  , rooksBlack = 0x8100000000000000 -- a8 (bit 56) and h8 (bit 63)
  , queensWhite = 0x0000000000000008 -- d1 (bit 3)
  , queensBlack = 0x0800000000000000 -- d8 (bit 59)
  , kingsWhite = 0x0000000000000010 -- e1 (bit 4)
  , kingsBlack = 0x1000000000000000 -- e8 (bit 60)
  }
```

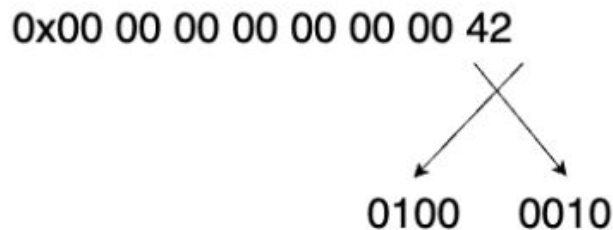


Figure 3: Hex to bitboard mapping

Bitboards

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
```

(a) Occupancy (stop before)

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
```

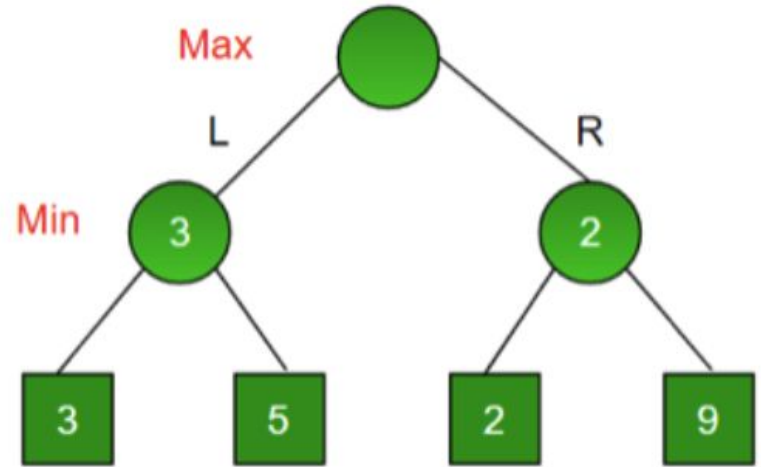
(b) Captureable (stop on)

```
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
1 0 0 1 0 0 0 0
1 0 0 0 1 0 0 1
1 0 0 0 0 1 0 1
1 1 0 0 0 0 1 1
0 1 1 0 1 1 1 0
```

(c) Queen moves from a1, h1

Minimax Algorithm with optimizations

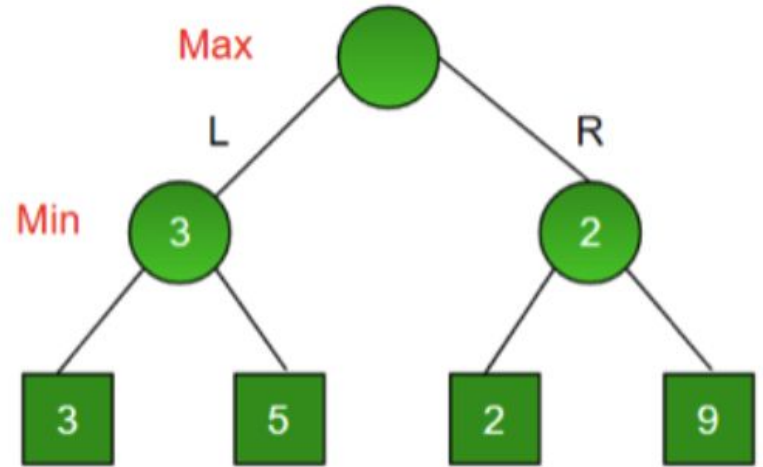
- Each turn the maximizing and min player switch roles and choose the most optimal branch
- Assumes each player plays optimally
- Space complexity 2^n



Minimax Algorithm with optimizations

Caching

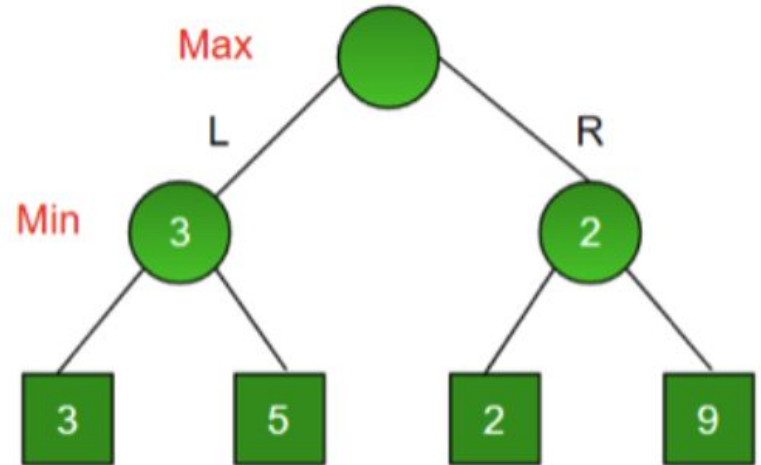
- Avoid recomputing expensive bitboard operations and comparisons
- Significant speedup at higher depths



Minimax Algorithm with optimizations

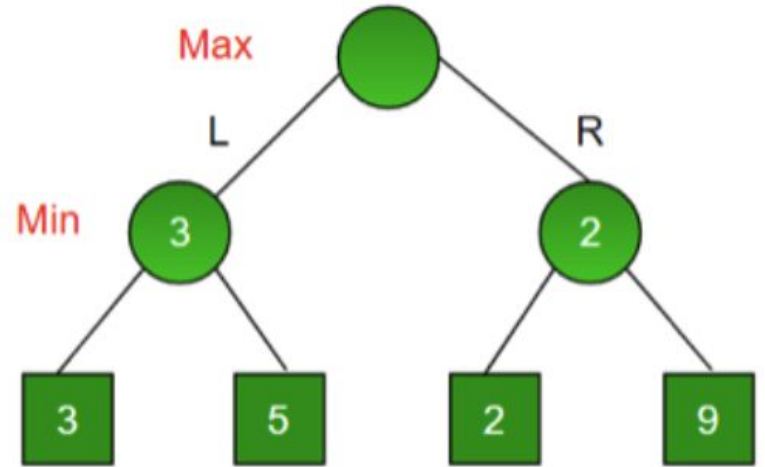
Pruning

- Avoid computing branches that we know the algorithm will never reach to save computational resources
- Has much more overhead than just caching but it takes runtime down even more aggressively

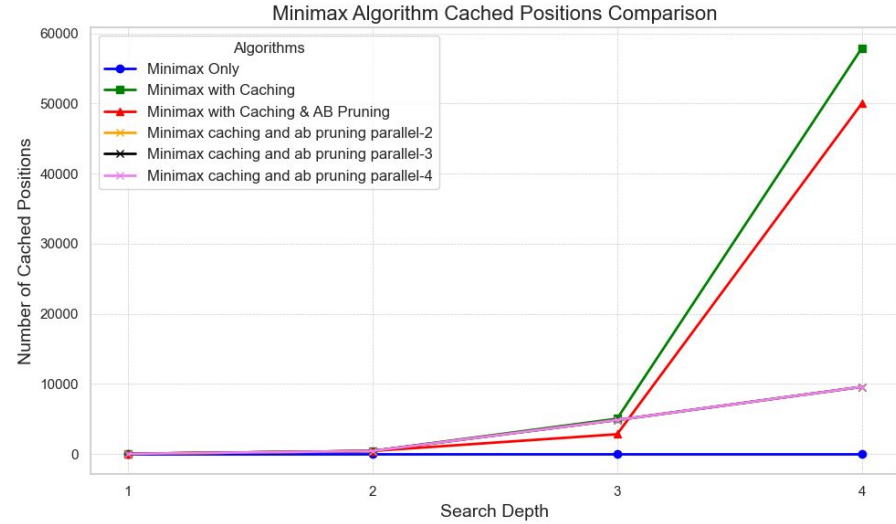
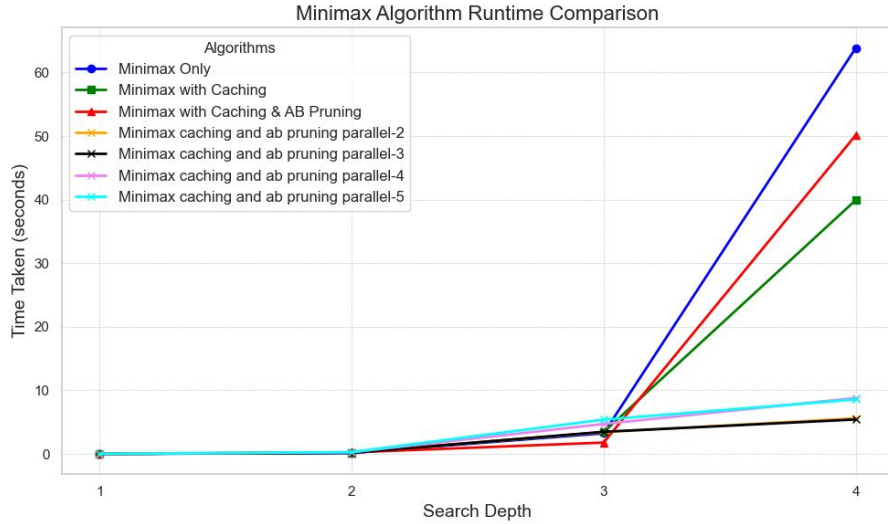


Parallelization

- Parallelize each top level minimax operation
- Leaves enough single threaded work, and breaks down the big work into sizeable chunks to take advantage of overhead
- Danger of exhausting system memory (24gb M3)



Preliminary Results



Preliminary Results - potential issues

- Exhausting resources
- Timing may not be fully accurate

