Chess Algorithms
Theory and Practice

Rune Djurhuus
Chess Grandmaster
runed@ifi.uio.no / runedj@microsoft.com
September 20, 2017
Content

- **Complexity** of a chess game
- **Solving chess**, is it a myth?
- **History** of computer chess
- **Chess** compared to **Go**
- **Search trees** and **position evaluation**
- **Minimax**: The basic search algorithm
- **Negamax**: «Simplified» minimax
- **Node explosion**
- **Pruning** techniques:
  - **Alpha-Beta** pruning
  - Analyze the **best move** first
  - **Killer-move** heuristics
  - **Zero-move** heuristics
- **Iterative deeper** depth-first search (IDDFS)
- Search tree **extensions**
- **Transposition** tables (position cache)
- Other challenges
- Endgame **tablebases**
- Demo
Complexity of a Chess Game

- **20** possible start moves, **20** possible replies, etc.
- **400** possible positions after **2** ply (half moves)
- **197,281** positions after **4** ply
- **7^{13}** positions after **10** ply (5 White moves and 5 Black moves)
- **Exponential explosion!**
- Approximately **40 legal moves** in a typical position
- There exists about **10^{120}** possible chess games
Solving Chess, is it a myth?

Chess Complexity Space

- The estimated number of possible chess games is $10^{120}$
  - Claude E. Shannon
  - 1 followed by 120 zeroes!!!
- The estimated number of reachable chess positions is $10^{47}$
  - Shirish Chinchalkar, 1996
- Modern GPU’s performs $10^{13}$ flops
- If we assume one million GPUs with 10 flops per position we can calculate $10^{18}$ positions per second
- It will take us $1\ 600\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000$ years to solve chess

Assuming Moore’s law works in the future

- Todays top supercomputers delivers $10^{16}$ flops
- Assuming 100 operations per position yields $10^{14}$ positions per second
- Doing retrograde analysis on supercomputers for 4 months we can calculate $10^{21}$ positions.
- When will Moore’s law allow us to reach $10^{47}$ positions?
- Answer: in 128 years, or around year 2142!

http://chessgpgpu.blogspot.no/2013/06/solving-chess-facts-and-fiction.html
History of Computer Chess

• Chess was a good fit for computers:
  – Clearly defined rules
  – Game of complete information
  – Easy to evaluate (judge) positions
  – Search tree is not too small or too big

• 1950: Programming a Computer for Playing Chess (Claude Shannon)
• 1951: First chess playing program (on paper) (Alan Turing)
• 1958: First computer program that can play a complete chess game
• 1981: Cray Blitz wins a tournament in Mississippi and achieves master rating
• 1989: Deep Thought loses 0-2 against World Champion Garry Kasparov
• 1996: Deep Blue wins a game against Kasparov, but loses match 2-4
• 1997: Upgraded Dee Blue wins 3.5-2.5 against Kasparov
• 2005: Hydra destroys GM Michael Adams 5.5-0.5
• 2006: World Champion Vladimir Kramnik looses 2-4 against Deep Fritz (PC chess engine)
• 2014: Magnus Carlsen launches “Play Magnus“ app on iOS where anyone can play against a chess engine that emulates the World Champion’s play at 21 different ages (5 to 25 years).
Chess Compared to Go

- Go is played on a 19x19 square board where a new stone is placed on any free square each move (and never moved around)
- Go has a much higher branching factor (starting with 361 and slowly descending) and much more complicated leaf node evaluation
- For many years the best Go programs had amateur rating only
- In 2016 Alpha Go surprisingly beat Lee Sedol (9-dan profession) 4-1 using a combination of machine learning (deep neural network) and Monte Carlo tree search algorithm.
- Alpha Go beat Ke Jie (ranked no. 1 in the world) 3-0 in 2017 and retired afterwards.
Search Trees and Position Evaluation

- Search trees (nodes are positions, edges are legal chess moves)
- Leaf nodes are end positions which needs to be evaluated (judged)
- A simple judger: Check mate? If not, count material
- Nodes are marked with a numeric evaluation value
Minimax: The Basic Search Algorithm

• Minimax: Assume that both White and Black plays the best moves. We maximizes White’s score
• Perform a depth-first search and evaluate the leaf nodes
• Choose child node with highest value if it is White to move
• Choose child node with lowest value if it is Black to move
• Branching factor is 40 in a typical chess position

White
Black
White
Black
White

ply = 0
ply = 1
ply = 2
ply = 3
ply = 4
NegaMax – “Simplified” Minimax

**Minimax**

```c
int maxi( int depth ) {
    if ( depth == 0 )
        return evaluate();
    int max = -\infty;
    for ( all moves ) {
        score = mini( depth - 1 );
        if ( score > max )
            max = score;
    }
    return max;
}
```

```c
int mini( int depth ) {
    if ( depth == 0 )
        return -evaluate();
    int min = +\infty;
    for ( all moves ) {
        score = maxi( depth - 1 );
        if ( score < min )
            min = score;
    }
    return min;
}
```

**NegaMax**

```c
int negaMax( int depth ) {
    if ( depth == 0 ) return evaluate();
    int max = -\infty;
    for ( all moves ) {
        score = -negaMax( depth - 1 );
        if ( score > max )
            max = score;
    }
    return max;
}
max(a, b) == -\text{min}(-a, -b)
```
Node explosion

A typical middle-game position has 40 legal moves.

- 10 M nodes per second (nps) is realistic for modern chess engines
- Modern engines routinely reach depths 25-35 ply at tournament play
- But they only have a few minutes per move, so they should only be able to go 5-6 ply deep
- How do they then get to depth 25 so easily?

<table>
<thead>
<tr>
<th>Depth</th>
<th>Node count</th>
<th>Time at 10M nodes/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>0.000004 s</td>
</tr>
<tr>
<td>2</td>
<td>1 600</td>
<td>0.00016 s</td>
</tr>
<tr>
<td>3</td>
<td>64 000</td>
<td>0.0064 s</td>
</tr>
<tr>
<td>4</td>
<td>2 560 000</td>
<td>0.256 s</td>
</tr>
<tr>
<td>5</td>
<td>102 400 000</td>
<td>10.24 s</td>
</tr>
<tr>
<td>6</td>
<td>4 096 000 000</td>
<td>6 min 49,6 s</td>
</tr>
<tr>
<td>7</td>
<td>163 840 000 000</td>
<td>4 h 33 min 4 s</td>
</tr>
<tr>
<td>8</td>
<td>6 553 600 000 000</td>
<td>7 d 14 h 2 min 40 s</td>
</tr>
</tbody>
</table>
Pruning Techniques

- The complexity of searching \( d \) ply ahead is \( O(b^*b^*...^*b) = O(b^d) \)
- With a branching factor \( (b) \) of 40 it is crucial to be able to prune the search tree
Alpha-Beta Pruning

“Position is so good for White (or Black) that the opponent with best play will not enter the variation that gives the position.”

- Use previous known max and min values to limit the search tree
- Alpha value: White is guaranteed this score or better (start value: -∞)
- Beta value: Black is guaranteed this score or less (start value: +∞)
- If Alpha is higher than Beta, then the position will never occur assuming best play
- If search tree below is evaluated left to right, then we can skip the greyed-out sub trees
- Regardless of what values we get for the grey nodes, they will not influence the root node score

```
White
Black
White
Black
White
```

```
ply = 0
ply = 1
ply = 2
ply = 3
ply = 4
```
Analyze the Best Move First

- Even with alpha-beta pruning, if we always start with the worst move, we still get $O(b^*b^*..*b) = O(b^d)$
- If we always start with the best move (also recursive) it can be shown that complexity is $O(b^1*b^1*b^1...b^1...) = O(b^{d/2}) = O(\sqrt{b^d})$
- We can **double** the **search depth** without using more resources
- Conclusion: It is very important to try to **start** with the **strongest moves first**
Killer-Move Heuristics

• Killer-move heuristics is based on the assumption that a strong move which gave a large pruning of a sub tree, might also be a strong move in other nodes in the search tree.

• Therefore we start with the killer moves in order to maximize search tree pruning.
Zero-Move Heuristics

- Alpha-Beta cutoff: “The position is so good for White (or Black) that the opponent with best play will avoid the variation resulting in that position”
- Zero-Move heuristics is based on the fact that in most positions it is an advantage to be the first player to move
- Let the player (e.g. White) who has just made a move, play another move (two moves in a row), and perform a shallower (2-3 ply less) and therefore cheaper search from that position
- If the shallower search gives a cutoff value (e.g. bad score for White), it means that most likely the search tree can be pruned at this position without performing a deeper search, since two moves in a row did not help
- Very effective pruning technique!
- Cavecats: Check and endgames (where a player can be in “trekktvang” – every move worsens the position)
Iterative Deeper Depth-First Search (IDDFS)

• Since it is so important to evaluate the best move first, it might be worthwhile to execute a shallower search first and then use the resulting alpha/beta cutoff values as start values for a deeper search.

• Since the majority of search nodes are on the lowest level in a balanced search tree, it is relatively cheap to do an extra shallower search.
Search Tree Extensions

• PC programs today can compute **25-35 ply ahead** (Deep Blue computed 12 ply against Kasparov in 1997, Hydra (64 nodes with FPGAs) computed at least 18 ply)

• It is important to **extend** the search in leaf nodes that are “**unstable**”

• Good **search extensions** includes all moves that gives **check** or **captures** a piece

• The longest search extensions are typically **double** the average length of the search tree!
Transposition Table

• **Same position** will commonly occur from different move orders
• All chess engines therefore has a **transposition table** (position cache)
• Implemented using a **hash table** with chess position as key
• Doesn’t have to evaluate large sub trees over and over again
• Chess engines typically uses half of available memory to hash table – proves how important it is
Other challenges

• Move generator (hardware / software)
  – Hydra (64 nodes Xeon cluster, FPGA chips) computed 200 millions positions per second, approximately the same as Deep Blue (on older ASIC chip sets)
  – Hydra computed 18+ ply ahead while Deep Blue only managed 12 (Hydra prunes search tree better)
  – Komodo 10 chess engine calculates 3-4 mill moves/second on my Surface Book (Intel i7 @ 2.6 GHz with 3 cores) and computes 20+ ply in less than 5 seconds and 25+ ply in less than 30 seconds

• Efficient data structure for a chess board (0x88, bitboards)

• Opening library suited for a chess computer

• Position evaluation:
  • Traditionally chess computers has done deep searches with a simple evaluation function
  • But one of the best PC chess engines today, Rybka, sacrifices search depth for a complex position evaluation and better search heuristics
Endgame Tablebases

- Chess engines play endgames with 3-7 pieces left on the board perfectly by looking up best move in huge tables.
- These endgame databases are called **Tablebases**.
- Retrograde analyses: Tablebases are generated by starting with **final positions** (check mate, steal mate or insufficient mating material (e.g. king vs. king)) and then **compute backwards** until all nodes in search tree are marked as win, draw or loose.
- Using complex **compression** algorithms (Nalimov, Syzygy).
- The newer Syzygy compression format uses less than 200 GB for all endgames with up to 6 piezes (compared to over 1 TB for Nalimov tablebases).
Lomonosov Tablebases

• All 7 piece endgames (except 6 pieces vs a lone king) calculated for the first time in 2013 on the Lomonosov supercomputer in Moscow State University.
• Took 6 months to generate
• Needed 140 TB of storage
• Longest forced mate: White to mate in 545 moves!

Demo

• Demo: ChessBase with chess engine Komodo 10 and Stockfish 7
• Best open source UCI chess engine (and may be best overall):
  – Stockfish (stockfishchess.org)
Thank you

Presenter: Rune Djurhuus
Contact:
  runed@ifi.uio.no
  runedj@microsoft.com
Version: Autumn 2017