Chess Algorithms Theory and Practice

Rune Djurhuus

Chess Grandmaster

runed@ifi.uio.no / runedj@microsoft.com

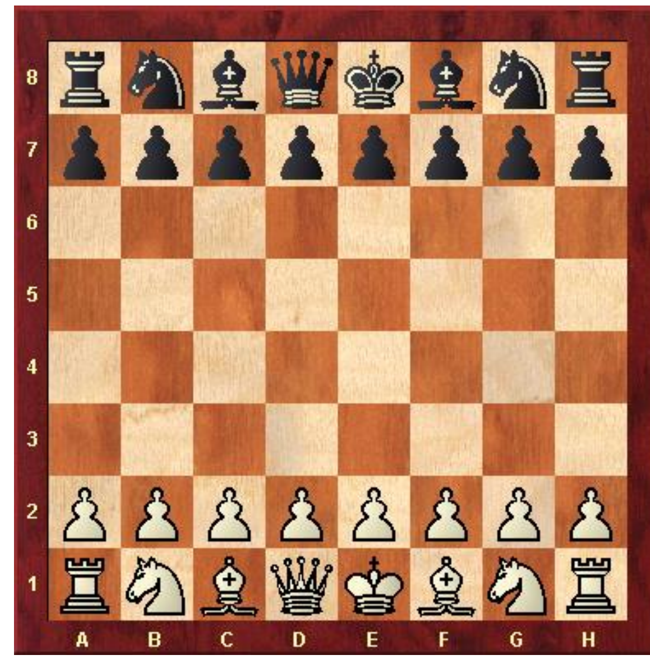
September 23, 2013

Content

- **Complexity** of a chess game
- **History** of computer chess
- **Search trees** and **position evaluation**
- **Minimax**: The basic search algorithm
- **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

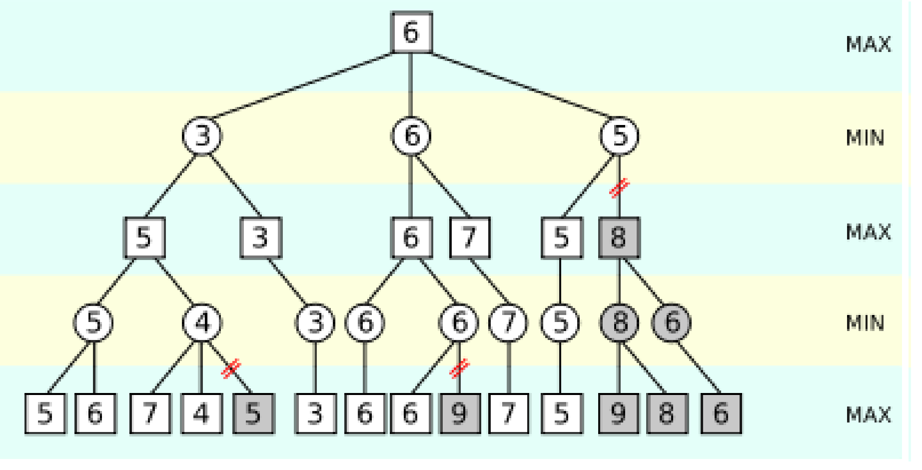


History of Computer Chess

- Chess is 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)

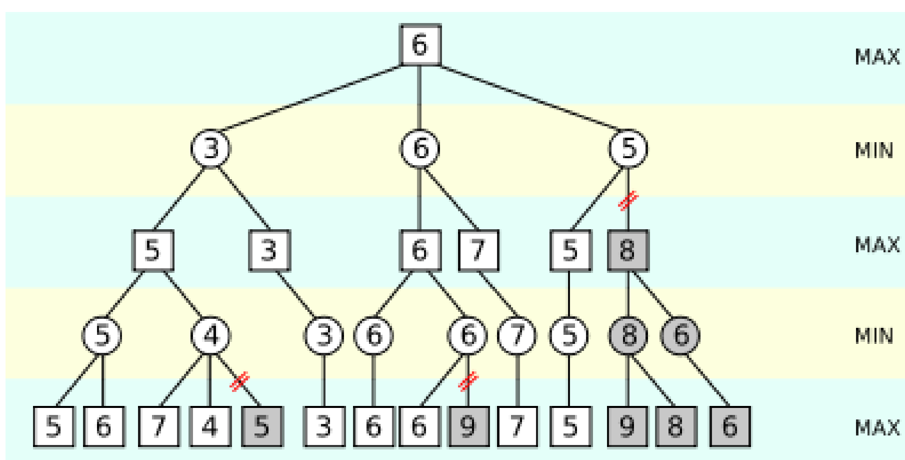
Search Trees and Position Evaluation

- 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	ply = 0
Black	ply = 1
White	ply = 2
Black	ply = 3
White	ply = 4

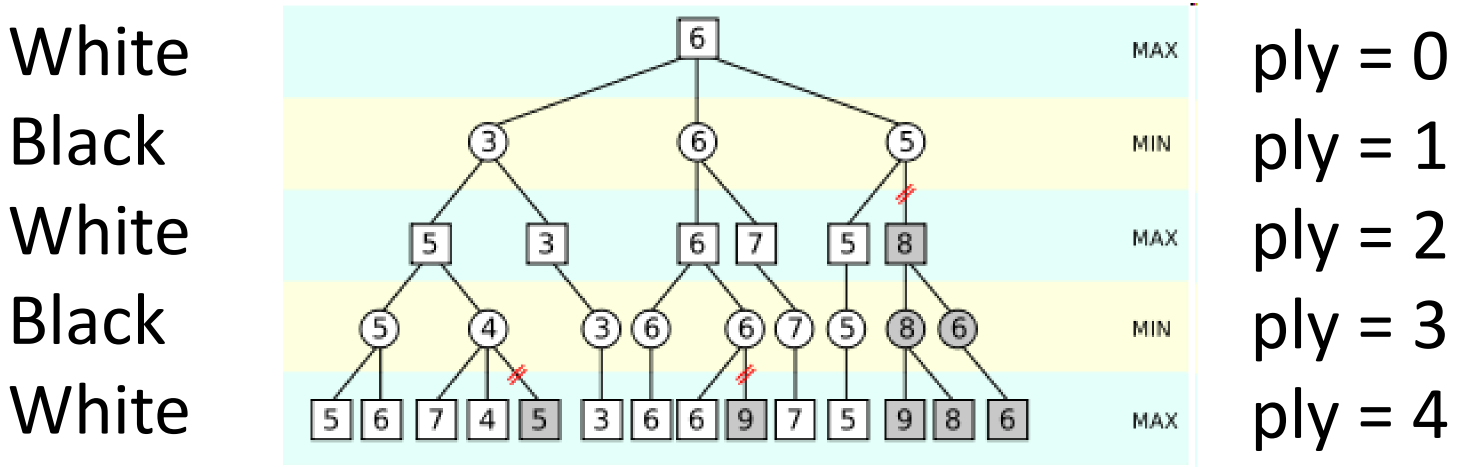
Pruning Techniques

- The complexity of searching *d* ply ahead is $O(b*b*\ldots*b) = O(b^d)$
- 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 Beta is less than Alpha, 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



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…) = $O(b^{d/2}) = O(\sqrt{b^d})$
- 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 **14-17 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 computes 18 ply ahead while Deep Blue only managed 12 (Hydra prunes search tree better)
  - Fritz 13 chess engine (single processor/core) manages 2.3 mill moves/second on my laptop and computes 15+ ply
  - Efficient data structure for a chess board
  - 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 plays 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 (Eugene Nalimov)
- All 3-5 piece endgames and some 6 piece endgames are stored in just 21 GB
- http://en.wikipedia.org/wiki/Nalimov_tablebase

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!

- See http://chessok.com/?page_id=27966

Demo

- Demo: ChessBase with chess engine Deep Rybka 4, Houdini 1.5 and Fritz 13

Thank you

Presenter: Rune Djurhuus

Contact:

runed@ifi.uio.no

runedj@microsoft.com

Version: Autumn 2013