Chess Algorithms
Timo Poranen
Department of Computer Sciences, University of Tampere

Figure 1: Position after black’s 12th move in game Kasparov vs. X3D Fritz.

- The classic answer to the question “How far do you see ahead?”, given by a master is “One move further than my opponent!”
Presentation structure (18.2)

- Motivation
- Short history of chess and other chess-like games
- Very short introduction to the rules of chess
- History of chess computers / algorithms
- A sample game against gnuchess program
- Designing chess algorithms
  - Searching the game tree (Minimax -algorithm)
  - Position evaluation
  - Pruning the game tree (Alpha-Beta pruning)
- Internet resources and how to start your own chess programming project!
- (26.2) Chess algorithms continues + GO and other board games
Motivation: why it is important to study game algorithms?

● Theoretical approach
  – Games have mathematical formalization and it is always worth to study the properties of mathematical structures!

● Practical approach
  – Game algorithms can help players to become even better players
  – To solve open problems: When it is possible to win a game with Rook+Bishop+King against Rook+King

● Commercial approach
  – Firms can make money with their products (sell chess games)
  – Chess programs can be used to advertise the products

● “Mental” approach
  – It is fun to play chess, study chess, and program chess algorithms
Short history of chess and other chess-like games

- Chess is originated from the Indian game Chaturanga, about 1400 years ago
- Chess spread westwards to Europe and eastwards as far as Japan, spawning variants as it went
- Game was introduced into Spain in the 10th century
- By the end of the 15th century, the modern rules for the basic moves established (western chess)
- Currently there exists many variations of chess: western chess, Chinese chess ("xiangqi"), Japanese chess ("shogi"), Korean chess, Burmese chess, Cambodian chess, Thai chess, Malaysian chess, Indonesian chess, Turkish chess...
- Western chess and the other variations of chess have millions of players
**Very short introduction to the rules of chess**

- Chess is a two-player game, played on a board with 64 squares.
- Both sides have 8 pawns, 2 knights, 2 bishops, 2 rooks, a queen and a king
- The aim of each player, proceeding by alternate moves (1 half-move = 1 ply), is to checkmate his opponent’s king (also draw is possible)
- Chess is a perfect information game, it is based on tactics and strategy

![Chessboard](image)

*Figure 2: Initial position and two checkmate positions.*
Computational complexity of chess

- Chess is a complex game:
  - Typically a position has thirty to forty possible moves (varies between 0 and 218, average is 37)
  - After 2 ply, $\geq 1000$ pos., after 4 ply, $\geq 1000000$ pos.
  - The number of legal positions is estimated to be between $10^{43}$ and $10^{50}$
  - The game-tree complexity is approximately $10^{123}$
  - Best programs can search on average to depth 15 in the game-tree

- Complexity results for other games
  - Tic-Tac-Toe: positions: $3.6 \times 10^6$
  - Connect four: game-tree complexity $10^{13}$
  - Xiangqi: complexity: $10^{150}$
  - Go: positions: $10^{170}$, complexity: $10^{397}$ (?)
History of chess algorithms I/II

- ELO rating system is a method for calculating the relative strength of chess players. ELO varies (in practice) between 900 and 2850
- 1945 Alan Turing used chess-playing as an example of what a computer could do
- 1957 First program that could play a full game of chess were developed (4 ply search in 8 minutes) by Alex Bernstein at MIT
- 1958 The alpha-beta pruning algorithm for chess was discovered
- 1970 CHESS 3.0 program, ELO 1200. 1977 CHESS 4.6 program, ELO 1700
- 1981 CRAY BLITZ (ELO 2258) won Mississippi state Championship. First won from a chess master (ELO > 2200)
History of chess algorithms II/II

- 1985 HITECH, ELO 2400 (beat a grandmaster 1988)
- 1989 DEEP THOUGHT, ELO 2550 loses two exhibition matches to Gary Kasparov (world champion)
- 1994 Kasparov loses a blitz game to Fritz 3 computer
- 1996 Deep Blue ($50 \times 10^6$ pos. / 3 min) wins one game in a tournament against Gary Kasparov
- 1997 Deep Blue won Kasparov in a 6 game match held in new york (3.5 - 2.5)
- 2003 Best computers have ELO app. 2800. Programs can evaluate over $3 \times 10^6$ pos. / 1 sec. (down to depth 15 in the game-tree)
A sample game against GNU Chess program

- GNU Chess is a free chess-playing program
  (http://www.gnu.org/software/chess/)
- GNU Chess program has ELO $\geq 2100$
- During the game, look carefully the “computer’s opinion” on positions and thinking (in the upper left corner of the game board)
- If the value is positive, GNU Chess stands better, if the value is negative, lecturer stands better
- Lecturer’s ELO is app. 1800, so the probability of that GNU Chess wins the game is app. 0.9
- Don’t make any bets!
Designing chess algorithms

- In order to be able to play chess well, a computer program:
  - need only know the rules of the game
  - how to evaluate any position which may arise

- A game board presentation and the rules are not difficult to program

- It is easy to recognize terminal positions (checkmate or stalemate on the board)

- For simple games, it is possible to search whole search-tree to find the winner

- The game-tree of chess is too large for a full search. The search depth needs a limit (to achieve any reasonable running time)

- If no pruning methods are applied, the maximum search depth is between 5 and 7 ($37^5 = 69343957$, $37^6 \approx 2.5 \times 10^9$ and $37^7 \approx 9.5 \times 10^{10}$)

- Difficulty is to evaluate non-terminal positions and how to prune the search tree!
Tree searching and minimax algorithm

- The minimax algorithm is a recursive algorithm for choosing the next move in a two-player game.

- A value is associated with each position of the game (if the value is negative, it is good for black and if it is positive, it is good for white).

- If it is white's turn, he chooses always such a move that maximizes the value of the next position, and black chooses always a move that minimizes the value of the next position.
Position Evaluation

- The evaluation function should return $+\infty$ for white’s winning positions, $-\infty$ for black’s winning positions and 0, if the game is a draw.

- The evaluation function should take account many factors:
  - The most important feature in chess is material
  - Mobility: the total number of moves that can be made
  - Development: how many pieces are moved away from their original positions
  - Castling: is it still possible to castle the king?
  - Centre control: from central squares pieces can attack more squares than from edge or corner squares
  - King attack and king safety
  - Pawn structure, piece attack and piece defence
  - Endgame evaluation, endgame and opening databases
Position Evaluation - material

- Other things being equal, the player who is ahead material will usually win the game

- Beginners are told that the pieces have different values which can be measured in multiples of a pawn
  - pawn = 1
  - knight, bishop = 3 (for open positions, bishop = 3.3)
  - rook = 5
  - queen = 9
  - king = above all value, say 1000

- In the beginning material is $39 +$ the value of king for both sides

- This simple material based evaluation function leads to a program that really plays chess!
Tree searching and minimax algorithm
Position Evaluation - mobility

- Mobility is the total number of moves that can be made by one’s pieces
- One possibility to combine material and mobility:
  \[ \text{score} = \text{material} + (0.1 \times \text{mobility}) \]
- In the beginning the mobility is 20

Figure 3: Positions with mobility 30, 19 and 22 for white.
Position Evaluation - Development and castling I/III

- If your pieces are all “developed” (i.e. off their original squares), then they will be ready for the ensuing battle which comes in the middle game.

- If you are badly behind in development, then you will be not ready for the struggle.

- The concept is relative simply, but quantifying is not an easy matter.

- Three extra development moves are worth of approximately one pawn.

- A normal game strategy: develop first bishops and knights, then castle king, then develop rooks and queen.

- Castling is almost essential when queens are still on the board.

- If your opponent no longer has his queen on the board, then it is usually no longer so essential to castle.
Position Evaluation - Development and castling II/III

development $= \frac{D}{3} - \frac{U}{4} - (k \times C)$, where

- $D =$ number of minor pieces not on their original squares
- $U = 0$, if the queen has not been moved (or has been captured)
- $U =$ the number of undeveloped pieces if the queen has been moved but not captured
- $C = 2$, if the opponents queen is still on the board
- $C = (1-P/4)$, if the opponent’s queen is captured, where $P$ is the number of opposing rooks and minor pieces which have been captured
- $k = 0$, if the player has castled, $1/3$ if the player has lost queen side castling right, $2/3$ if player has lost king side castling rights, $1$ if player has lost all castling rights
- A suitable factor for the development (in pawn units) is between $0.5$ and $1$
Position Evaluation - Development and castling III/III

- The position below is a critical one arising from king’s gambit (black’s move)

<table>
<thead>
<tr>
<th></th>
<th>material</th>
<th>mobility</th>
<th>development</th>
<th>eval. function</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>34</td>
<td>46</td>
<td>3/3-0/4-(0*2)=1</td>
<td>39.6 . . 39.1</td>
</tr>
<tr>
<td>Black</td>
<td>38</td>
<td>34</td>
<td>3/3-3/4-(1*2)=-1.75</td>
<td>39.65 . . 40.525</td>
</tr>
</tbody>
</table>

- eval: $-1.875 . . -0.05$. If the pawn structure is taken account, eval. $\approx 0$
Position Evaluation - Weightings for the evaluation function

- When it is decided which features are included in the evaluation function, it is necessary to determine reasonable weightings for the features.

- Evaluation function takes form

\[
(W_1 \ast F_1) + (W_2 \ast F_2) + \ldots + (W_k \ast F_k),
\]

where \( F_1, F_2, \ldots, F_k \) are various features and \( W_1, W_2, \ldots, W_k \) are the relative weights for these features.

- How to adjust weightings?

- For example, is a pawn worth of 8, 10 or 12 units of mobility?
  - Intelligent guess, and improve on the quesswork in the light of experience
  - Make the program play with both sides with different parameters. Try to recognize a good parameter setting by changing the weights
  - A fully automatic method by Arthur Samuel
  - There is no method that guarantee optimal parameter setting
Horizontal effect

- The horizontal effect happens in positions, in which a program cannot see a crucial move because that move is beyond the horizon of its look-ahead analysis. Suppose that black makes a full width search to a depth of 2 ply.

<table>
<thead>
<tr>
<th>Black’s 1st ply move:</th>
<th>... h5 b2</th>
<th>... h6 c3</th>
<th>any other</th>
</tr>
</thead>
<tbody>
<tr>
<td>White’s 2st ply move:</td>
<td>g6 b2</td>
<td>g5 c3</td>
<td>capture black’s queen</td>
</tr>
<tr>
<td>Net material loss:</td>
<td>3 pawns</td>
<td>3 pawns</td>
<td>at least 6 pawns</td>
</tr>
</tbody>
</table>
The concept “never”

- Typically, the strong human player assumes that computers will “never” be able to ...

- White is ahead 4 pawns in the following two positions

  - Left position is black’s win (needs a 14-ply search)
  - Right position is white’s win (verification needs a 14-ply search)
Alpha-beta pruning I/II

- A human chess master knows that certain moves “just cannot be right”. This reduces the size of the game-tree to something between 50 and 100 positions.

- For a computer program, anything which helps to reduce the number of moves in the game-tree is helpful.

- Alpha-beta pruning is a technique to reduce the number of nodes evaluated by the minimax algorithm.

- It prunes out parts of the search tree that are so good for one player that the opponent will never allow them to be reached.

- Under optimal conditions, 99.8% of nodes of the game-tree can be pruned out (normally 99.5%).

- Alpha-beta pruning can double the search depth.
Alpha-beta pruning II/II

- The game-tree below illustrates the efficiency of the alpha-beta pruning

- The larger the game-tree becomes, the bigger are cuts

- In the worst-case alpha-beta can make the same number of calls for the evaluation function than minimax, but this could happen only when the values of all nodes are very similar
Computer chess resources

- A good collection of computer chess links can be found from
  http://www.tim-mann.org/chess.html

- Free chess engines: GNU Chess, Crafty
  (http://www.gnu.org/software/chess/)
  (http://www.limunltd.com/crafty/)

- Free game viewers: Xboard (linux), Winboard (windows)

- More computer chess and chess-like game links and information:
  http://en.wikipedia.org/wiki/Chess