Game Tree

[From the Handbook of AI (Barr & Feigenbaum 1981)]

• Basic features of many games, e.g. chess, checkers, Go and tic-tac-toe:
  – Two players
  – Players alternate in making moves
  – The possible legal moves are known in advance (there is no element of chance)
  – Complete information (no hidden pieces)
  – Specified beginning state
  – Ends in win/lose/draw

• A complete game tree is a representation of all possible plays of such a game
Game Tree

- A **game tree** has the following components:
  - **root node**: initial state, first player’s turn to move
  - **successor nodes**: states reachable in one move starting from the root node for the first player, then for the 2nd player, and so on...
  - **terminal nodes**: a terminal state represents a win/loss or draw
  - **Note**: tree does not have duplicate nodes (each possible position represented once)

- Each path from the root to a terminal node gives a different complete play of the game

- **Complexity**
  - **State space complexity**: # of legal positions reachable from the root
    - tic-tac-toe: 5,748
  - **Game tree complexity**: # of possible games = # terminal nodes
    - tic-tac-toe: 255,168
  - **game tree >> state space**
    - *more than one way of reaching the same legal position*
# Game Tree

- from wikipedia:

<table>
<thead>
<tr>
<th>Game</th>
<th>Board size (cells)</th>
<th>State-space complexity (as (\log) to base 10)</th>
<th>Game-tree complexity (as (\log) to base 10)</th>
<th>Average game length (plies)</th>
<th>Complexity class of suitable generalized game</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>PSPACE-complete(^2)</td>
</tr>
<tr>
<td>Pentominoes</td>
<td>64</td>
<td>12</td>
<td>18</td>
<td>10 [(^3)]</td>
<td>?, but in PSPACE</td>
</tr>
<tr>
<td>Connect Four</td>
<td>42</td>
<td>14(^1)</td>
<td>21(^1)</td>
<td>36(^1)</td>
<td>?, but in PSPACE</td>
</tr>
<tr>
<td>English draughts (8x8)</td>
<td>32</td>
<td>20(^4) or 18(^1)</td>
<td>31(^1)</td>
<td>70(^1)</td>
<td>EXPTIME-complete(^5)</td>
</tr>
<tr>
<td>Oware(^6)</td>
<td>12</td>
<td>12(^1)</td>
<td>32(^1)</td>
<td>60(^1)</td>
<td>? Generalization is not clear</td>
</tr>
<tr>
<td>Qubic</td>
<td>64</td>
<td>30(^1)</td>
<td>34(^1)</td>
<td>20(^1)</td>
<td>PSPACE-complete(^2)</td>
</tr>
<tr>
<td>Fanorona</td>
<td>45</td>
<td>21</td>
<td>46</td>
<td>44</td>
<td>?, but in EXPTIME</td>
</tr>
<tr>
<td>Nine Men's Morris</td>
<td>24</td>
<td>10(^1)</td>
<td>50(^1)</td>
<td>?</td>
<td>?, but in EXPTIME</td>
</tr>
<tr>
<td>International draughts (10x10)</td>
<td>50</td>
<td>30(^?)</td>
<td>54(^1)</td>
<td>90(^1)</td>
<td>EXPTIME-complete(^5)</td>
</tr>
<tr>
<td>Chinese checkers (2 sets)</td>
<td>121</td>
<td>27</td>
<td>?</td>
<td>?</td>
<td>?, but in EXPTIME</td>
</tr>
<tr>
<td>Chinese checkers (6 sets)</td>
<td>121</td>
<td>77</td>
<td>?</td>
<td>?</td>
<td>?, but in EXPTIME</td>
</tr>
</tbody>
</table>
Game Tree

chess:

- state space complexity
  - $10^{50}$
- game tree complexity
  - $10^{123}$

<table>
<thead>
<tr>
<th>Lines of Action</th>
<th>64</th>
<th>24</th>
<th>56</th>
<th>?</th>
<th>?, but in EXPTIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reversi</td>
<td>64</td>
<td>28</td>
<td>58</td>
<td>58</td>
<td>PSPACE-complete⁷</td>
</tr>
<tr>
<td>Hex (11x11)</td>
<td>121</td>
<td>56</td>
<td>?</td>
<td>40</td>
<td>PSPACE-complete⁸</td>
</tr>
<tr>
<td>Go-Moku, freestyle, 15 x 15</td>
<td>225</td>
<td>105²⁹</td>
<td>70¹⁰</td>
<td>30¹¹</td>
<td>PSPACE-complete²</td>
</tr>
<tr>
<td>Chess</td>
<td>64</td>
<td>50</td>
<td>123</td>
<td>80</td>
<td>EXPTIME-complete¹⁰</td>
</tr>
<tr>
<td>Connect6</td>
<td>361</td>
<td>172</td>
<td>140</td>
<td>200</td>
<td>?, but in PSPACE</td>
</tr>
<tr>
<td>Backgammon</td>
<td>28</td>
<td>20</td>
<td>144</td>
<td>?</td>
<td>? Generalization is not clear</td>
</tr>
<tr>
<td>Xiangqi</td>
<td>90</td>
<td>48</td>
<td>150</td>
<td>80</td>
<td>?, believed to be EXPTIME-complete</td>
</tr>
<tr>
<td>Quoridor</td>
<td>81</td>
<td>42</td>
<td>162</td>
<td>?</td>
<td>?, but in PSPACE</td>
</tr>
<tr>
<td>Shogi</td>
<td>81</td>
<td>71¹¹</td>
<td>226¹¹</td>
<td>110²</td>
<td>EXPTIME-complete¹²</td>
</tr>
<tr>
<td>Arimaa</td>
<td>64</td>
<td>42</td>
<td>296¹³</td>
<td>70</td>
<td>?, but in EXPTIME</td>
</tr>
<tr>
<td>Irensei</td>
<td>361</td>
<td>?171</td>
<td>360</td>
<td>80</td>
<td>?, but in EXPTIME</td>
</tr>
<tr>
<td>Go (19x19)</td>
<td>361</td>
<td>171¹⁴</td>
<td>360¹⁴</td>
<td>150¹⁵</td>
<td>EXPTIME-complete¹⁵</td>
</tr>
</tbody>
</table>
Game Tree

• Obviously (?) we can’t compute the complete game tree for chess

  – means the game is still interesting!

Checkers 'solved' after years of number crunching

• 19:00 19 July 2007 · NewScientist.com news service · Justin Mullins
• The ancient game of checkers (or draughts) has been pronounced dead. The game was killed by the publication of a mathematical proof showing that draughts always results in a draw when neither player makes a mistake. For computer-game aficionados, the game is now "solved".
• Draughts is merely the latest in a steady stream of games to have been solved using computers, following games such as Connect Four, which was solved more than 10 years ago.
• The computer proof took Jonathan Schaeffer, a computer-games expert at the University of Alberta in Canada, 18 years to complete and is one of the longest running computations in history.
Game Tree

- The crucial part of Schaeffer's computer proof involved playing out every possible endgame involving fewer than 10 pieces.
- The result is an endgame database of 39 trillion positions.
- By contrast, there are only 19 different opening moves in draughts.
- Schaeffer's proof shows that each of these leads to a draw in the endgame database, providing neither player makes a mistake.
- Schaeffer was able to get his result by searching only a subset of board positions rather than all of them, since some of them can be considered equivalent.
- He carried out a mere $10^{14}$ calculations to complete the proof in under two decades.
- At its peak, Schaeffer had 200 desktop computers working on the problem full time, although in later years he reduced this to 50 or so.
- http://www.cs.ualberta.ca/~chinook/
Game Tree

- For chess

<table>
<thead>
<tr>
<th>Ply</th>
<th>Without Check nor Checkmate</th>
<th>In Check</th>
<th>In Checkmate</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ply 0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ply 1</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>ply 2</td>
<td>400</td>
<td>0</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>ply 3</td>
<td>8890</td>
<td>12</td>
<td>0</td>
<td>8902</td>
</tr>
<tr>
<td>ply 4</td>
<td>196812</td>
<td>461</td>
<td>8</td>
<td>197281</td>
</tr>
<tr>
<td>ply 5</td>
<td>4838258</td>
<td>27004</td>
<td>347</td>
<td>4865609</td>
</tr>
<tr>
<td>ply 6</td>
<td>118251225</td>
<td>798271</td>
<td>10828</td>
<td>119060324</td>
</tr>
<tr>
<td>ply 7</td>
<td>3162798012</td>
<td>32668081</td>
<td>435767</td>
<td>3195901860</td>
</tr>
<tr>
<td>ply 8</td>
<td>84029997363</td>
<td>959129557</td>
<td>9852036</td>
<td>84998978956</td>
</tr>
<tr>
<td>ply 9</td>
<td>2403434332264</td>
<td>35695709940</td>
<td>400191963</td>
<td>2439530234167</td>
</tr>
<tr>
<td>ply 10</td>
<td>68265214423776</td>
<td>1078854669486</td>
<td>8790619155</td>
<td>69352859712417</td>
</tr>
<tr>
<td>ply 11</td>
<td>2058141026024096</td>
<td>39147687661803</td>
<td>362290010907</td>
<td>2097651003696806</td>
</tr>
<tr>
<td>ply 12</td>
<td>(*) 62854969236701747</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Minimax Search

• **Situation**
  - two players: A and B
  - A to move (pick 2 or 3)
  - What should A pick?
Minimax Search

- Min/Max computation
  - bottom-up
  - values:
    - win > draw > lose
    - A chooses the maximum value move available
    - B chooses the minimum value move available

**assumption:** B is always going to make his best move
Negmax Search

- Alternative but equivalent formulation to minimax
- (Knuth & Moore 1975):
  - \(+1 = \text{win, } -1 = \text{lose, } 0 = \text{draw}\)
  - value at each node is the value to the player whose turn it is at that node
  - \(F(n) = \max \{-F(n_1), \ldots, -F(n_k)\}\)
  - assuming node \(n\) has children \(n_1, \ldots, n_k\)
Iterative Deepening

• since chess game tree is too large and time is a factor in a practical game, **iterative deepening** is used

• **Idea:**
  • generate each **ply**, i.e. level, using your legal move generator
  • and apply static evaluation function to compute best possible move
  • **Benefit:** you always have the best move (given ply depth completed so far) ready

• **Node expansion:** expand nodes at each ply, and replace static value at each node with **backed-up**, i.e. bottom-up computed values, via minimax/negmax computation

• **Assumption:** backed-up values from game tree expansion are more accurate than static evaluation function values
Alpha-Beta (α/β) Pruning

• There are situations where it’s not necessary to expand certain nodes during minimax/negmax search

• **Idea:** at node 1 (looking to maximize)
  - \( \alpha \) represents a lower bound cutoff
  - initially \( \alpha = -\infty \)
  - evaluate node 2, \( \alpha = 15 \)
  - evaluate node 4, value of node 3 is \( \leq 10 \)
  - (prune) cut off search

• **Example:** alpha (\( \alpha \)) cutoff
  - minimax
  - negmax

\[
\begin{align*}
1 & \quad \text{MAX} \\
2 & \quad \text{MIN} \\
& \quad F(2)=15 \\
3 & \quad \text{MIN} \\
& \quad F(3) \leq 10 \\
4 & \quad \text{MAX} \\
& \quad F(4)=10 \\
5 & \quad \text{MAX} \\
\end{align*}
\]

no point in expanding node 5
Alpha-Beta (α/β) Pruning

- **Idea:** at node 2 (looking to minimize)
  - β represents an upper bound cutoff
  - initially, set β = +∞
  - evaluate node 4, β = 20
  - evaluate node 6, value of node 35 is ≥ 25
  - (prune) cut off search

- **Example:** beta (β) cutoff

- **minimax**

- **negmax**

**Note:** effective α/β-pruning relies on setting a good threshold for α and β early – this means ordering, i.e. trying, the best moves first
Alpha-Beta (α/β) Pruning

• Pascal-style pseudo code (negmax formulation):

```pascal
integer procedure VALUE (position p, integer alpha, integer beta):
begin
    integer m, i, t, d;
    determine the successor positions \( p_1, p_2, \ldots, p_d \) of position \( p \);
    if \( d = 0 \) then VALUE := \( f(p) \) else
    begin m := alpha;
        for i := 1 step 1 until d do
            begin
                t := -VALUE \( (p_i, \text{-} beta, \text{-} m) \);
                if \( t > m \) then m := t;
                if m \( \geq \) beta then go to done;
            end;
    end;
    done: VALUE := m;
end;
```

basically the same code is also given in http://www.frayn.net/beowulf/theory.html#analysis
Move Ordering

• According to http://www.frayn.net/beowulf/theory.html#analysis
  – Alpha-beta search effectively reduces the branching factor at each node from 30-40 to about 7 or 8 provided that you have a reasonably good initial move ordering routine.
  – One usually considers captures first, followed by pawn pushes and checks, moves towards the centre, and then the rest. This tends to put the best move at or near the top of the list in the majority of cases.
Transposition Table

• **Dynamic programming**
  – Savings through memoization
  – Simply use a hash table of evaluated positions (and their values), use the bitboard for fast position comparison (same position = same bitboard)
  – Since there may be multiple paths to the same position, each new position is checked against the transposition table
Depth and Playing Strength

• There’s an interesting discussion in http://people.csail.mit.edu/heinz/dt/node49.html

• **Basic idea:**
  – Have different versions of the same chess program set for different search depths play each other
  – Assuming enough games played (for results to be statistically significant)

• webpage indicates that not enough games were played in the studies.
  – However, last study cited was **1997: Junghanns et al.**
  – “The gain in playing strength averaged around 200 rating points per ply of search.”
The Basics

• We’ve covered the basics of game tree search (there is much more out there...)

• Your job (Task 5)
  – Implement minimax/negmax game tree search with alpha/beta pruning
  – demos in 2 weeks?
Roadmap

• **Task 6**
  – your term project
  – go beyond the lectures in some aspect of your chess program
  – e.g. opening book, endgame, better search, parallel search, quiescent search, beam width manipulation, etc.
  – implement the improved algorithm
  – write up a report (5–10 pages)

• **End of semester**
  – submit your chess program
  – include instructions on how to run
  – submit your term project report
HOMEWORK
Lecture 10

• Read the References cited below.
• Be ready for a “5 Minute Quiz” for the next time that will extract questions from those references.
CHEATING IN CHESS

References.

5. http://mywebpages.comcast.net/danheisman/Articles/ZCheating_Chess.htm