Adversarial Search

School of EECS
Washington State University
Games

- Classic AI challenge
  - Easy to represent
  - Difficult to solve
- Zero-sum games
  - Total final reward to all players is constant
- Perfect information (e.g., Chess, Checkers)
  - Fully observable and deterministic
- Imperfect information (e.g., Poker)
- Chance (e.g., Backgammon)
Tic–Tac–Toe

- Average branching factor about 2
- Average game length about 8
- Search tree has about $2^8 = 256$ nodes
- State space (search graph) has about $3^9 = 19,683$ nodes
Game Tree

- MAX wants to maximize its outcome
- MIN wants to minimize its outcome
- Search tree refers to the search for a player’s next move
- Terminal node
- Utility
Chess

- Average branching factor about 35
- Average game length about 100 (50 moves per player)
- Search tree has about $35^{100} = 10^{154}$ nodes
- State space (search graph) about $10^{40}$ nodes

Garry Kasparov vs. IBM's Deep Blue (1997)
Optimal Play

MAX

MIN

Artificial Intelligence 6
Optimal Play

- **Minimax value**
  - Best player can achieve assuming all players play optimally

  \[
  \text{Minimax}(s) = \begin{cases} 
  \text{Utility}(s) & \text{if } \text{TerminalTest}(s) \\
  \max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a)) & \text{if } \text{Player}(s) = \text{MAX} \\
  \min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a)) & \text{if } \text{Player}(s) = \text{MIN}
  \end{cases}
  \]

- **Minimax decision**
  - Action that leads to minimax value
Minimax Algorithm

function MINIMAX-DECISION \((state)\) returns an action

\[
\text{return arg max}_{a \in \text{ACTIONS}(state)} \text{MIN-VALUE(RESULT}(state,a))
\]

function MAX-VALUE \((state)\) returns a utility value

\[
\text{if TERMINAL-TEST}(state) \text{ then return UTILITY}(state) \\
v \leftarrow -\infty \\
\text{for each } a \text{ in ACTIONS}(state) \text{ do} \\
v \leftarrow \text{MAX}(v, \text{MIN-VALUE(RESULT}(state,a))) \\
\text{return } v
\]

function MIN-VALUE \((state)\) returns a utility value

\[
\text{if TERMINAL-TEST}(state) \text{ then return UTILITY}(state) \\
v \leftarrow \infty \\
\text{for each } a \text{ in ACTIONS}(state) \text{ do} \\
v \leftarrow \text{MIN}(v, \text{MAX-VALUE(RESULT}(state,a))) \\
\text{return } v
\]
Minimax Demo

www.yosenspace.com/posts/computer-science-game-trees.html
Minimax Algorithm

- Essentially depth-first search of game tree
- Time complexity: $O(b^m)$
  - $m = \text{maximum tree depth}$
  - $b = \text{legal moves at each state}$
- Space complexity
  - Generates all actions: $O(bm)$
  - Generates one action: $O(m)$
- Practical?
Pruning Search Tree

(a) \([-\infty, +\infty]\)

[b] \([-\infty, 3]\)

3

(b) \([-\infty, +\infty]\)

[b] \([-\infty, 3]\)

3 12

(c) \([3, +\infty]\)

[b] \([3, 3]\)

3 12 8

(d) \([3, +\infty]\)

[b] \([3, 3]\)

3 12 8 2

(e) \([3, 14]\)

[b] \([3, 3]\)

3 12 8 2 14

(f) \([3, 3]\)

[b] \([3, 3]\)

3 12 8 2 14 5 2
Alpha–Beta Pruning

- Prune parts of the search tree that MAX and MIN would never choose

- \( \alpha = \) value of best choice for MAX so far (highest value)

- \( \beta = \) value of best choice for MIN so far (lowest value)

- Keep track of alpha \( \alpha \) and beta \( \beta \) during search

If \( m > n \), Player will never move to \( n \).
function **Alpha-Beta-Search** (*state*) returns an action

\[ v \leftarrow \text{Max-Value}(state, -\infty, +\infty) \]

return the *action* in \text{ACTIONS}(state) with value \( v \)

function **Max-Value** (*state*, \( \alpha \), \( \beta \)) returns a utility value

if **Terminal-Test**(*state*) then return **Utility**(*state*)

\[ v \leftarrow -\infty \]

for each *a* in \text{ACTIONS}(state) do

\[ v \leftarrow \text{Max}(v, \text{Min-Value}(*\text{Result}(state,a)*, \alpha, \beta)) \]

if \( v \geq \beta \) then return \( v \)

\( \alpha \leftarrow \text{Max}(\alpha, v) \)

return \( v \)

function **Min-Value** (*state*, \( \alpha \), \( \beta \)) returns a utility value

if **Terminal-Test**(*state*) then return **Utility**(*state*)

\[ v \leftarrow +\infty \]

for each *a* in \text{ACTIONS}(state) do

\[ v \leftarrow \text{Min}(v, \text{Max-Value}(*\text{Result}(state,a)*, \alpha, \beta)) \]

if \( v \leq \alpha \) then return \( v \)

\( \beta \leftarrow \text{Min}(\beta, v) \)

return \( v \)
Alpha–Beta Pruning Demo

- [inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice](http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice)
Move Ordering

- **ALPHA–BETA–SEARCH** still $O(b^m)$ worst case
- If order moves by value, then could prune maximally (always choose best move next)
  - Achieve $O(b^{m/2})$ time
  - Effective branching factor $b^{1/2}$
  - Chess: $35 \rightarrow 6$
  - But not practical
- Choosing moves randomly
  - Achieve $O(b^{3m/4})$ average case
- Choosing moves based on impact
  - E.g., chess: captures, threats, forward, backward
  - Closer to $O(b^{m/2})$
Real-Time Game Play

- Minimax and Alpha–Beta search to terminal nodes
- Impractical for most games due to time limits
- Employ cutoff test to treat nodes as terminal nodes
- Heuristic evaluation function at these nodes to estimate utility
- \( d = \text{depth} \)

\[
H - \text{Minimax}(s, d) = \\
\begin{cases} 
\text{Eval}(s) & \text{if CutoffTest}(s, d) \\
\max_{a \in \text{Actions}(s)} H - \text{Minimax}(\text{Result}(s, a), d + 1) & \text{if Player}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} H - \text{Minimax}(\text{Result}(s, a), d + 1) & \text{if Player}(s) = \text{MIN}
\end{cases}
\]
Real-Time Game Play

- Heuristic evaluation function $\text{EVAL}(s)$
  - Weighted linear combination of features
    \[ \text{Eval}(s) = \sum_{i=1}^{n} w_i f_i(s) \]
    - E.g., chess
      - $f_1(s) =$ #pawns, $w_1 = 1$
      - $f_4(s) =$ #bishops, $w_4 = 3$
  - Weighted non-linear combination of features
  - Learn weights
  - Learn features
Cutoff test
- Cutoff at a fixed depth limit
- Iterative deepening until time runs out
- Cutoff only at quiescent states
  - No eminent large changes in evaluation function
  - E.g., captures in chess
- Horizon effect pushes inevitable bad outcomes beyond cutoff depth
  - Singular extension continues search along moves that look clearly better than others
Other Speedups

- Transposition table
  - States can be reached from different paths
  - Hash table keeps track of explored states and their values

- Opening and ending move databases
  - Fewer choices at opening and end of game
  - Memorize optimal strategies
Go

- Average branching factor about 250
- Average game length about 200 (100 moves per player)
- Search tree has about $250^{200} = 10^{480}$ nodes
- State space (search graph) about $10^{170}$ nodes


Lee Sedol vs. Google DeepMind’s AlphaGo (2016)

www.wired.com/2016/03/sadness-beauty-watching-googles-ai-play-go
Stochastic Games

- Element of chance (e.g., dice roll)
- Include chance nodes in game tree
  - Branch to possible outcomes with their probabilities
Stochastic Games

- Can’t compute minimax values
- Can compute expected minimax values

\[
\text{ExpectiMinimax}(s) = \begin{cases} 
\text{Utility}(s) & \text{if} \quad \text{TerminalTest}(s) \\
\max_{a \in \text{Actions}(s)} \text{ExpectiMinimax}(\text{Result}(s, a)) & \text{if} \quad \text{Player}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{ExpectiMinimax}(\text{Result}(s, a)) & \text{if} \quad \text{Player}(s) = \text{MIN} \\
\sum_r P(r) \text{ExpectiMinimax}(\text{Result}(s, r)) & \text{if} \quad \text{Player}(s) = \text{CHANCE}
\end{cases}
\]

- $r$ represents possible chance event (e.g., dice roll)
- $\text{Result}(s, r) = \text{state } s \text{ with a particular outcome } r$
Stochastic Games

- Chance nodes increase branching factor
- Search time complexity $O(b^{mn^m})$
  - Where $n$ is the number of chance outcomes
  - E.g., backgammon: $n = 21$, $b \approx 20$ (can be large)
  - Can only search a few moves ahead
- Estimate ExpectiMinimax values
Can reason about all possible states of unknown information

If $P(s)$ represents probability of each unknown state $s$, then best move is:

$$\arg \max_a \sum_s P(s) \text{Minimax}(\text{Result}(s, a))$$

If $|s|$ too large, take a random sample

- Monte Carlo method
State of the Art

- **Chess**
  - Komodo ([komodochess.com](http://komodochess.com)) – proprietary
  - Stockfish ([stockfishchess.org](http://stockfishchess.org)) – open source

- **Checkers (solved, perfect play)**
  - Chinook ([webdocs.cs.ualberta.ca/~chinook](http://webdocs.cs.ualberta.ca/~chinook))
  - Open/close database plus brute-force search

- **Backgammon**
  - Extreme Gammon ([www.extremegammon.com](http://www.extremegammon.com))
  - GNU Backgammon ([www.gnubg.org](http://www.gnubg.org))
  - Neural network based evaluation function

- **Poker**
  - DeepStack ([poker.cs.ualberta.ca](http://poker.cs.ualberta.ca))
  - Libratus ([en.wikipedia.org/wiki/Libratus](http://en.wikipedia.org/wiki/Libratus))

- **Go (Hard: 19x19 board, b >200)**
  - AlphaGo ([deepmind.com/research/alphago](http://deepmind.com/research/alphago))
  - Zen ([senseis.xmp.net/?ZenGoProgram](http://senseis.xmp.net/?ZenGoProgram))
Summary

- Adversarial search and games
- Minimax search
- Alpha–beta pruning
- Real–time issues
- Stochastic and partially observable games
- State of the art ...

In what games can humans still beat computers?