Adversarial Search

David Cline
A game consists of:

- \( S_0 \) – the initial state
- Player(s) – defines which player moves in state \( s \)
- Result(s,a) – result of an action
- TerminalTest(s) – whether endgame state reached
- Utility(s,p) – utility function for terminal states

- In Chess: (0, 1), (1, 0), (½, ½)
Game characteristics

• Zero-sum
  – Sum of utilities is 0, or more generally, players compete for limited # of points

• Adversarial game
  – Players have competing goals
Game Tree

- A game tree maps out the state space of game play
- Tree levels alternate between players
- Branches are different actions taken by players
- Each turn called a "ply"
Example Game Tree

- Tic-Tac-Toe game tree
  - (Q) How big is this game tree, and is that tractable?
MiniMax Tree

- Players at different levels want to maximize or minimize utility value.
  - Assume both players will play optimally
  - MIN player wants to minimize value
  - MAX player wants to maximize value

- (Q) Why assume other player is play optimally?
MiniMax Tree

\[
\text{MINIMAX}(s) = \\
\begin{cases} 
\text{UTILITY}(s) & \text{if \ TERMINAL-TEST}(s) \\
\max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if \ PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if \ PLAYER}(s) = \text{MIN} 
\end{cases}
\]

- (Q) Show min/max value at each node and highlight actions for optimal gameplay.
MiniMax Pseudocode

function MINIMAX-DECISION(state) returns an action
    return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s, a)))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(s, a)))
    return v

(Q) Can we avoid going down to the leaf nodes to get min and max values?
Alpha-Beta pruning

• If at some point, the opponent has a better choice than our current best, no need to evaluate the rest of the opponent node.

Figure 5.6  FILES: figures/alpha-beta-general.eps (Tue Nov 3 16:22:20 2009). The general case for alpha–beta pruning. If $m$ is better than $n$ for Player, we will never get to $n$ in play.
Alpha-Beta Search

function \textsc{Alpha-Beta-Search}(state) returns an action
\begin{align*}
v & \leftarrow \textsc{Max-Value}(state, -\infty, +\infty) \\
\text{return the action in ACTIONS(state)} & \text{ with value } v
\end{align*}

function \textsc{Max-Value}(state, \alpha, \beta) returns a utility value
\begin{align*}
& \text{if TERMINAL-TEST(state) then return UTILITY(state)} \\
& v \leftarrow -\infty \\
& \text{for each } a \text{ in ACTIONS(state) do} \\
& \hspace{1em} v \leftarrow \textsc{Max}(v, \textsc{Min-Value}(RESULT(s, a), \alpha, \beta)) \\
& \hspace{1em} \text{if } v \geq \beta \text{ then return } v \\
& \hspace{1em} \alpha \leftarrow \textsc{Max}(\alpha, v) \\
& \text{return } v
\end{align*}

function \textsc{Min-Value}(state, \alpha, \beta) returns a utility value
\begin{align*}
& \text{if TERMINAL-TEST(state) then return UTILITY(state)} \\
& v \leftarrow +\infty \\
& \text{for each } a \text{ in ACTIONS(state) do} \\
& \hspace{1em} v \leftarrow \textsc{Min}(v, \textsc{Max-Value}(RESULT(s, a), \alpha, \beta)) \\
& \hspace{1em} \text{if } v \leq \alpha \text{ then return } v \\
& \hspace{1em} \beta \leftarrow \textsc{Min}(\beta, v) \\
& \text{return } v
Alpha-Beta pruning

- (Q) Perform alpha-beta pruning. Show alpha, beta, and v values

  - Alpha and beta get passed down to child nodes
  - v gets initialized to worst possible value for node
  - Maximizer updates alpha, Minimizer updates beta
Heuristic Evaluation functions

- Even with Alpha-Beta pruning, the search tree may be too big.
- We can apply a heuristic function on non-terminal nodes to stop the recursion:

\[
\text{Standard:} \quad \text{MINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\
\max_{a \in \text{Actions}(s)} \text{MINIMAX}((\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{MINIMAX}((\text{RESULT}(s, a)) & \text{if PLAYER}(s) = \text{MIN}
\end{cases}
\]

\[
\text{Heuristic:} \quad \text{H-MINIMAX}(s, d) = \begin{cases} 
\text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\
\max_{a \in \text{Actions}(s)} \text{H-MINIMAX}((\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\
\min_{a \in \text{Actions}(s)} \text{H-MINIMAX}((\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MIN}.
\end{cases}
\]
Move Ordering

- Alpha-Beta pruning can be improved by move reordering.
  - Use a heuristic to decide which moves to try first.
  - Use iterative deepening search to help decide ordering.
Deep Blue

- First chess program to beat world champion (1997)
  - 30 IBM RS/6000 processors
  - Alpha-Beta search
  - Searched up to 30 billion positions per move
  - Reached depth of 14 routinely
  - As many as 40 plies on some queries by selective tree extension.
  - Opening book of 4000 positions
  - 700,000 grand master game database
  - Large end game database
Stochastic games

• Games with chance element (such as rolling dice) add "chance nodes" to the minimax tree:
• (Q) What are the implications for the search?
Stochastic Minimax

- Stochastic minimax uses expected values (weighted averages) for min and max:

\[
\text{EXPECTIMINIMAX}(s) = \begin{cases} 
\text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\
\max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\
\min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\
\sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE}
\end{cases}
\]
State of the art game programs

• Chess - Deep Blue, Hydra
• Checkers – Chinook
• Othello – Logistello
• Backgammon - Gerry Tesauro
• Scrabble – Quackle
• Go – MoGO (Still not that good)
Checkers

• Chinook
  – Runs on PC and uses alpha-beta search
  – Beat human champion in 1990
  – Since 2007 plays perfectly
    • Uses database of 39 trillion endgame positions
Othello (Reversi)

• Has smaller search space than chess
  – Usually 5-10 legal moves
• Logistello defeated world champion
Backgammon

- Gerry Tesauro (1992)
  - Reinforcement learning + neural networks
  - Searches only to depth 2 or 3
  - Competetive with top human players
Scrabble

• Easy to write a program that will determine best move on board.
• This is not the whole story, however.
• In 2006 a program called “Quackle” defeated the former world champion, David Boys.
Go

- Played on 19 x 19 board
- Very high branching factor (starts at 361)
- Difficult to write evaluation functions for Go
- Current Go programs
  - Master level for 9 x 9 small board
  - Only amateur level for 19 x 19 board