Artificial Intelligence

Adversarial Search

Ch. 5
Games vs. search problems

• "Unpredictable" opponent $\rightarrow$ specifying a move for every possible opponent reply

• Time limits $\rightarrow$ unlikely to find goal, must approximate
Game tree (2-player, deterministic, turns)
Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:
Minimax algorithm

function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
Properties of minimax

- Complete?
- Optimal?
- Time complexity?
- Space complexity?
α-β pruning example

MAX

MIN

3 12 8

≥ 3
α-β pruning example

```
MAX

MIN

3
12
8
2

≥ 3

≤ 2

X  X
```
α-β pruning example
α-β pruning example
α-β pruning example
Properties of α-β

• Pruning does not affect final result

• Good move ordering improves effectiveness of pruning

• With "perfect ordering," time complexity = $O(b^{m/2})$
  $\rightarrow$ doubles depth of search

• A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
Why is it called $\alpha$-$\beta$?

- $\alpha$ is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for $\text{max}$
- If $v$ is worse than $\alpha$, $\text{max}$ will avoid it → prune that branch

- Define $\beta$ similarly for $\text{min}$

- What if $\alpha \geq \beta$? Prune!
The $\alpha$-$\beta$ algorithm

\begin{align*}
\textbf{function} & \quad \text{Alpha-Beta-Search}(state) \quad \textbf{returns} \quad \text{an action} \\
\text{inputs:} & \quad \text{state, current state in game} \\
& \quad v \leftarrow \text{Max-Value}(state, -\infty, +\infty) \\
& \quad \text{return the action in Successors(state) with value } v
\end{align*}

\begin{align*}
\textbf{function} & \quad \text{Max-Value}(state, \alpha, \beta) \quad \textbf{returns} \quad \text{a utility value} \\
\text{inputs:} & \quad \text{state, current state in game} \\
& \quad \alpha, \text{the value of the best alternative for } \text{MAX along the path to } state \\
& \quad \beta, \text{the value of the best alternative for } \text{MIN along the path to } state \\
& \quad \text{if Terminal-Test}(state) \text{ then return Utility}(state) \\
& \quad v \leftarrow -\infty \\
& \quad \text{for } a, s \text{ in Successors(state) do} \\
& \quad \quad v \leftarrow \text{Max}(v, \text{Min-Value}(s, \alpha, \beta)) \\
& \quad \quad \text{if } v \geq \beta \text{ then return } v \\
& \quad \quad \alpha \leftarrow \text{Max}(\alpha, v) \\
& \quad \text{return } v
\end{align*}
The α-β algorithm

function Min-Value(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if Terminal-Test(state) then return Utility(state)
    v ← +∞
    for a, s in Successors(state) do
        v ← Min(v, Max-Value(s, α, β))
        if v ≤ α then return v
        β ← Min(β, v)
    return v
Suppose we have 100 secs, explore $10^4$ nodes/sec \( \rightarrow 10^6 \) nodes per move

Real game trees are often too large to evaluate completely!

Standard approach:

• **cutoff test:**
  
  e.g., depth limit

• **evaluation function**
  
  = estimated desirability of position
Evaluation functions

- For chess, typically **linear weighted sum of features**
  
  $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

- e.g., $w_1 = 9$ with
  
  $f_1(s) = \text{(number of white queens)} - \text{(number of black queens)}$, etc.
**Cutting off search**

*MinimaxCutoff* is identical to *MinimaxValue* except

1. *Terminal* is replaced by *Cutoff*
2. *Utility* is replaced by *Eval*

Does it work in practice?

\[ b^m = 10^6, \ b=35 \rightarrow m=4 \]

4-ply look ahead is a hopeless chess player!

- 4-ply \( \approx \) human novice
- 8-ply \( \approx \) typical PC, human master
- 12-ply \( \approx \) Deep Blue, Kasparov
Deterministic games in practice

- **Checkers**: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

- **Chess**: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.

- **Othello**: human champions refuse to compete against computers, who are too good.

- **Go**: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
Nondeterministic games: backgammon
Nondeterministic games in general

- In nondeterministic games, chance introduced by dice, card-shuffling
- Simplified example with coin-flipping:
Games of chance

• Expectiminmax – consider the probabilities of all possible moves and compute a utility value for each node

• Expectiminmax(n) =
  – Utility(n) if n is a terminal state
  – max_s Expectiminmax(s) if n is a max node
  – min_s Expectiminmax(s) if n is a min node
  – \( \sum_s P(s) \times \text{Expectiminmax}(s) \) if n is a chance node