Artificial Intelligence

Adversarial Search

Ch. 5
Games vs. search problems

• "Unpredictable" opponent → specifying a move for every possible opponent reply

• Time limits → 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**

```plaintext
function MINIMAX-DECISION(state) returns an action
    \( v \leftarrow \text{MAX-VALUE}(state) \)
    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)
    \( v \leftarrow -\infty \)
    for a, s in SUCCESSORS(state) do
        \( v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s)) \)
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    \( v \leftarrow \infty \)
    for a, s in SUCCESSORS(state) do
        \( v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s)) \)
    return v
```
Properties of minimax

• Complete?  
  \( \text{Yes} \)

• Optimal?  
  \( \text{Yes} \)

• Time complexity?  
  \( \text{Exp} = \text{size of game tree} \)

• Space complexity?
α-β pruning example

MAX

MIN

3

12

8

≥ 3
α-β pruning example
α-β pruning example
α-β pruning example
α-β pruning example
Properties of $\alpha$-$\beta$

- 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 α-β?

• α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for max

• If \( v \) is worse than α, max will avoid it
  → prune that branch

• Define β similarly for min

• What if α\( >= \)β? Prune!
The α-β algorithm

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

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

function $\text{MIN-VALUE}(state, \alpha, \beta)$ returns a utility value

inputs: $state$, current state in game

$\alpha$, the value of the best alternative for $\text{MAX}$ along the path to $state$

$\beta$, the value of the best alternative for $\text{MIN}$ along the path to $state$

if $\text{TERMINAL-TEST}(state)$ then return $\text{UTILITY}(state)$

$v \leftarrow +\infty$

for $a, s$ in $\text{SUCCESSORS}(state)$ do

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$

if $v \leq \alpha$ then return $v$

$\beta \leftarrow \text{MIN}(\beta, v)$

return $v$
Resource limits

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
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

- e.g., \( w_1 = 9 \) with
  \[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{etc.} \]
**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) =**
- \( \text{Utility}(n) \) if \( n \) is a terminal state
- \( \max_s \text{Expectiminmax}(s) \) if \( n \) is a max node
- \( \min_s \text{Expectiminmax}(s) \) if \( n \) is a min node
- \( \sum_s P(s) * \text{Expectiminmax}(s) \) if \( n \) is a chance node