Chapter 6
Games and Game Tree Search

Game Tree for Tic-Tac-Toe

MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
Optimal Strategy: Minimax Search

• Find the contingent strategy for MAX assuming an infallible MIN opponent
• Assumption: Both players play optimally!
• Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

\[
\text{MINIMAX-VALUE}(n) = \begin{cases} 
\text{UTILITY}(n) & \text{If } n \text{ is a terminal} \\
\max_s \in \text{succ}(n) \text{ MINIMAX-VALUE}(s) & \text{If } n \text{ is a MAX node} \\
\min_s \in \text{succ}(n) \text{ MINIMAX-VALUE}(s) & \text{If } n \text{ is a MIN node} 
\end{cases}
\]
Choose this move

Minimax Algorithm

function MINMAX-DECISION(state) returns an action
    \( v \leftarrow \text{MAX-VALUE}(\text{state}) \)
    return the action in \( \text{SUCCESSORS}(\text{state}) \) with value \( v \)

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

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

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)

Good enough?

- **Chess:**
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^m \approx 35^{100} \approx 10^{154}$

- **The Universe:**
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{21}$ milliseconds

Can we search more efficiently?
Two-Ply Game Tree

Minimax algorithm explores depth-first

Pruning trees
Pruning trees

MAX

MIN

3
12
8
2

≠ 2

X X

No need to look at or expand these nodes!!
Pruning trees

MAX

MIN

3 12 8 2 14 5

X X

Pruning trees

MAX

MIN

3 12 8 2 14 5

X X
Prune this tree!
Do we need to check this node?
No, because $\max(-29, -37) = -29$ and other children of min can only lower min's value of -37 (because of the min operation)

Another pruning opportunity!
Pruning can eliminate entire subtrees!
This form of tree pruning is known as alpha-beta pruning

alpha = the highest (best) value for MAX along path
beta = the lowest (best) value for MIN along path
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
- Define β similarly for min

The α-β algorithm
(minimax with four lines of added code)

function ALPHA-BETA-SEARCH(state) returns an action
inputs: state, current state in game
v ← MAX-VALUE(state, −∞, +∞)
return the action in SUCCESSORS(state) with value v

function MAX-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 s, v in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
if v ≥ β then return v
α ← MAX(α, v)
return v
The $\alpha$-$\beta$ algorithm (cont.)

```
function 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 Terminal-Test(state) then return Utility(state)
    $v \leftarrow +\infty$
    for $a, s$ in Successors(state) do
        $v \leftarrow \text{Min}(v, \text{Max-Value}(s, \alpha, \beta))$
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow \text{Min}(\beta, v)$
    return $v$
```

Properties of $\alpha$-$\beta$

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  allows us to search deeper - doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)
Good enough?

$\text{Chess:}
\begin{align*}
&\text{branching factor } b \approx 35 \\
&\text{game length } m \approx 100 \\
&\text{a-}\beta \text{ search space } b^{m/2} \approx 35^{50} \approx 10^{77}
\end{align*}

$\text{The Universe:}
\begin{align*}
&\text{number of atoms } \approx 10^{78} \\
&\text{age } \approx 10^{21} \text{ milliseconds}
\end{align*}

Can we do better?

- Strategies:
  - search to a fixed depth (cut off search)
  - iterative deepening (most common)
  - stop only at 'quiescent' nodes
  - Unlikely to change wildly in value in near future
Evaluation Function

огда search space is too large, create game tree up to a certain depth only.

Art is to estimate utilities of positions that are not terminal states.

Example of simple evaluation criteria in chess:

- Material worth: pawn=1, knight=3, rook=5, queen=9.
- Other: king safety, good pawn structure
- Rule of thumb: 3-point advantage = certain victory

\[
\text{eval}(s) = w_1 \times \text{material}(s) + w_2 \times \text{mobility}(s) + w_3 \times \text{king safety}(s) + w_4 \times \text{center control}(s) + \ldots
\]
Cutting off search

§ Does it work in practice?
   If \( b^m = 10^6 \) and \( b=35 \Rightarrow m=4 \)

§ 4-ply lookahead is a hopeless chess player!
   § 4-ply \( \approx \) human novice
   § 8-ply \( \approx \) typical PC, human master
   § 12-ply \( \approx \) Deep Blue, Kasparov

Transposition Tables

• Game trees contain repeated states

• In chess, e.g., the game tree may have \( 35^{100} \) nodes, but there are only \( 10^{40} \) different board positions

• Similar to closed list in graph-search, maintain a transposition table

Got its name from the fact that the same state is reached by a transposition of moves.
Games that Include an Element of Chance

White has just rolled 6-5 and has 4 legal moves.

Game Tree for Games with an Element of Chance

§ In addition to MIN- and MAX nodes, we need chance nodes (e.g., for rolling dice).

Expectiminimax Algorithm:
For chance nodes, compute expected value over successors

§ Search costs increase: Instead of \( O(b^d) \), we get \( O((bn)^d) \), where \( n \) is the number of chance outcomes.
Imperfect Information

• E.g. card games, where opponents' initial cards are unknown

• Idea: For all deals consistent with what you can see
  $\text{compute the minimax value of available actions for each of possible deals}$
  $\text{compute the expected value over all deals}$

Game Playing in Practice

$\text{Checkers:}$ Chinook ended 40 year reign of human world champion Marion Tinsley in 1994; used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions (!)

$\text{Chess:}$ Deep Blue defeated human world champion Gary Kasparov in a 6 game match in 1997; Deep Blue searches 200 million positions per second, uses very sophisticated evaluation functions, and undisclosed methods for extending some lines of search up to 40 ply

$\text{Othello:}$ human champions refuse to play against computers because software is too good

$\text{Go:}$ human champions refuse to play against computers because software is too bad
Summary of Game Tree Search

- Basic idea: minimax -- too slow for most games
- Alpha-Beta pruning can increase max depth by factor up to 2
- Limited depth search may be necessary
- Static evaluation functions necessary for limited depth search and help alpha-beta
- Opening game and End game databases can help
- Computers can beat humans in some games (checkers, chess, othello) but not in others (Go)