Lecture 4 – Adversarial search

CS/CNS/EE 154
Andreas Krause
Projects

- Recitations: Thursday 4:30pm – 5:30pm, Annenberg 107
  - Details about projects
  - Will also be posted on webpage

- By Monday 10/11
  - Form team of 3 students
  - Need to select project (Doodle link will be sent today)
  - For independent projects: need to submit proposal

- If you don’t have a team, send email to TAs

- Homework 1 out on Friday
### Types of games

<table>
<thead>
<tr>
<th></th>
<th>Chess</th>
<th>Backgammon</th>
<th>Poker</th>
<th>Rock Paper Scissors</th>
<th>WoW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observable?</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Determ.?</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Simultan.?</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Zero-sum?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Discrete?</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td># Players?</td>
<td>2</td>
<td>2</td>
<td>≥2</td>
<td>2</td>
<td>~8·10⁶</td>
</tr>
</tbody>
</table>

In this class, focus on two-player, sequential, zero-sum, discrete (mostly deterministic)
Games vs. search

- In games, actions are **nondeterministic**
  - Opponent can affect state of the environment
- Optimal solution no longer sequence of actions, instead a **strategy** (policy, conditional plan)
  - If you X I’ll do Y, else if you do Y I’ll do Z, ....
Game tree

MAX (x)

MIN (o)

MAX (x)

MIN (o)

TERMINAL

Utility

-1  0  +1
Minimax game tree

- Search for optimal move no matter what opponent does
- minimax value = best achievable payoff against best play

MAX

MIN

Can evaluate using DFS
Solving deterministic games

- MiniMax used to calculate optimal move:
- Inductive definition:

  If $n$ is terminal node:
  - Value is utility($n$.state)

  If $n$ is MAX node:
  - Value is highest value of all successor node values

  If $n$ is MIN node
  - Value is lowest value of all successor node values
Proper)es \(\text{of minimax search}\)

- Complete? \(\text{Yes if tree is finite}\)
- Time complexity? \(O(b^m)\)
- Space complexity? \(O(b \cdot m)\)
- Optimal? \(\text{Yes if finite and if opponent plays optimally}\)

For Tic Tac Toe, \(b \leq 9, m \leq 9, q = \text{dim symmetries} \Rightarrow \leq 30000\) nodes

For Chess, \(b \approx 35, m \approx 50, 35 \cdot 50 = 1750\) hopeless intractable in
α-β-pruning

(a)
α-β-pruning

(b) 

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

[\(-\infty, 3\)]

3 12
α-β-pruning

(c)
\( \alpha-\beta\) pruning

\[(d)\]

\[
\begin{align*}
\text{A} & \quad [3, +\infty] \\
\text{B} & \quad [3, 3] \\
\text{C} & \quad [-\infty, 2] \\
3 & \\
12 & \\
8 & \\
2 & \\
\end{align*}
\]
\( \alpha-\beta\)-pruning
α-β-pruning

(f)
α-β-pruning

Key idea: For each node n in minimax tree keep track of

- α: Best value for MAX player if n is reached
- β: Best value for MIN player if n is reached

Never need to explore consequences of actions for which β<α

Avoid exploring “provably suboptimal” parts of minimax tree
α-β-pruning algorithm

function `ALPHA-BETA-DECISION(state)` returns an action
return the `a` in `ACTIONS(state)` maximizing `MIN-VALUE(RESULT(a, state))`

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 `a, s` in `SUCCESSORS(state)` do
    `v ← MAX(v, MIN-VALUE(s, α, β))`
if `v ≥ β` then return `v`
`α ← MAX(α, v)`
return `v`

function `MIN-VALUE(state, α, β)` returns a utility value
same as `MAX-VALUE` but with roles of `α, β` reversed
Does move ordering matter?

MAX

MIN

\[
\begin{array}{c}
\text{A} \\
3 \\
a_1 \\
\text{B} \\
3 \\
b_1 \\
b_2 \\
b_3 \\
3 \\
\text{C} \\
2 \\
c_1 \\
c_2 \\
c_3 \\
2,2 \\
\text{D} \\
2 \\
d_1 \\
d_2 \\
d_3 \\
5 \\
14 \\
6 \\
4 \\
\end{array}
\]
Move ordering matters a lot

- Worst case: No improvement \( O(b^m) \)
- Best case (ideal ordering): \( O(b^{m/2}) \) $\leq$ can get fairly close in practice
- Random ordering: \( O((b/\log b)^m) \)

How to find a good ordering?

Use IDS, "remember" best moves from shallower trees
Large state spaces

- Typical branching factor in chess: 35
- Computing the complete minimax tree is intractable
- Instead: Cut off search, and replace utility(s) with eval(s)
  - eval(s) is heuristic value of state s

Materia ( & black/white pawns, rooks, ...)
Developing evaluation functions

This is where expert knowledge comes in

Typical approach:

- Select features \( f_1, \ldots, f_n \) that may be useful, e.g., value of pieces on board, positions of pieces, ...
- Learn weights from examples

\[
eval(s) = \sum_{i=1}^{n} w_i f_i(s)
\]

- Deep Blue used \( \sim 6,000 \) different features!
- Often, reinforcement learning is very useful here (e.g., TD-gammon beats world champion in backgammon)
Problems with cutoff search

Black to move
Taming the horizon effect

- Quiescence search
  - Evaluation function also evaluates “stability” (e.g., strong captures, etc.)
  - Cutoff postponed if position is unstable
  - Search time no longer constant

- Singular extension
  - Search deeper if a node’s value is much better than its siblings’
  - Reduces effective branching factor
  - Can search much longer sequences (even 30-40ply)
Playing world class chess

- Current PCs can evaluate ~200 million nodes / 3 min
- Minimax search: ~5 ply lookahead
- With α-β pruning: ~10 ply

Further improvements:
- Quiescence search: Only evaluate “stable” positions
- Transposition tables: Remember states evaluated before
- Singular extensions: Expand tree if there is singular best move
- Null move heuristic: Get lower bound by letting opp. move 2x
- Precompute endgames (all 5, some 6 piece positions)
- Opening library (up to ~30ply in first couple moves)
- Hydra: 18 ply lookahead (on 64 processor cluster)
Two “types” of uncertainty

Adversarial and stochastic
Solving stochastic games

- ExpectiMiniMax used to calculate optimal move
- Defined inductively:

  If $n$ is terminal node (or cutoff):
  - Value is $\text{utility}(n.\text{state})$ (or $\text{eval}(n.\text{state})$)

  If $n$ is MAX node:
  - Value is highest value of all successor node values

  If $n$ is MIN node
  - Value is lowest value of all successor node values

  If $n$ is CHANCE node
  - Value is (weighted) average of all successor node values
Dealing with large state spaces

- Backgammon:
  - 21 possible roles with 2 die; ~20 legal moves
  - \#nodes for depth 4 tree:
    \[21 \cdot (20, 21)^3\]

- As depth increases, reaching any particular node becomes exponentially unlikely
  - Lookahead becomes less valuable
  - \(\alpha-\beta\)-pruning much less useful: world just won’t play along!

- TD-gammon competitive with best human players:
  - Uses only 2 ply lookahead!
  - But very carefully trained evaluation function