CS 4804: Intro to AI

Game

Virginia Tech CS 4804 Fall 2021
Instructor: Yinlin Chen
Today’s Topics

• Game
• Adversarial search
Project & Homework Reminder

• Homework 1: Due 09/12 11:59pm

• Project 1: Due 09/17 11:59pm
Informed Search Recap

1. Is A* graph search guaranteed to return an optimal solution?
2. Is A* graph search guaranteed to expand no more nodes than uniform-cost graph search?
3. Is Greedy graph search guaranteed to return an optimal solution?
4. Let $h_1(s)$ be an admissible A* heuristic. Let $h_2(s) = 2h_1(s)$
   a. Is the solution found by A* tree search with $h_2$ guaranteed to have a cost at most twice as much as the optimal path?
   b. Is the solution found by A* graph search with $h_2$ guaranteed to be an optimal solution.
Agent has adversaries

Image from: battle vs chess
Game

- Chess: 1997 - Deep Blue defeated Gary Kasparov
- Checkers: 2007 Checkers was solved.
- Go: 2016 ALPHAGOGO
- StarCraft II: 2019 ALPHASTAR
- ...
Types of Games

• Deterministic or stochastic?
• One, two, or more players?
• Zero Sum?
• Perfect information?
  – (fully observable or not)
• Example:
  – Chess is a deterministic, two-player, turn-taking, perfect information, and zero-sum game.
  – StarCraft II is a deterministic, multi-player, real-time, partial observable, and zero-sum game
Deterministic Games

• Many possible formalizations, one is:
  – States: \( S \) (start at \( s_0 \))
  – Players: \( P={1...N} \) (usually take turns)
  – Actions: \( A \) (may depend on player / state)
  – Transition Function: \( S \times A \rightarrow S' \)
  – Terminal Test: \( T(S) \)
  – Terminal Utilities: \( S \times P \rightarrow R \)

• Solution for a player is a policy:
  \( S \rightarrow A \)
Zero-Sum Games

• Agents have opposite utilities
• Agent MAX vs Agent MIN
  – MAX: Maximize the (value, points, scores)
  – MIN: Minimize the (value, points, scores)
• Pure competition
• Adversarial
• Example: Chess, GO, Clash Royale, etc.
Definition of Two-player Zero-Sum Game

- States: S. Start at $S_0$ (initial state)
- To-Move(s): Move in state $s$
- Action(s): Set of legal moves in state $s$
- Result($s$, $a$): The transition model. The state resulting from taking action $a$ in state $s$
- Is-Terminal(s): A terminal test. Game over?
  - Terminal states: States where the game has ended
- Utility($s$, $p$): A utility function which defines the final number value to player $p$ when the game ends in terminal state $s$
General Games

- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
Adversarial Game Tree

• Deterministic, zero-sum games:
  – Tic-tac-toe, chess, go, etc.
  – One player maximizes result
  – The other player minimizes result

• Example: Two-ply game tree:
  – A state-space search tree
  – Players (MAX, MIN) alternate turns (ply)
  – Compute \textit{minimax value} of each node in the tree and select best (achievable) utility against adversary
Tic-Tac-Toe Game Tree

MAX (x)

MIN (o)

MAX (x)

MIN (o)

TERMINAL

Utility: -1, 0, +1
Two-ply Game Tree

MAX

MIN

b₁ b₂ b₃ c₁ c₂ c₃ d₁ d₂ d₃

3 12 8 2 4 6 14 5 2
Minimax

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

- DFS
- Time complexity: $O(b^m)$
- Space complexity: $O(bm)$

function \textsc{Minimax-Search}(game, state) returns an action
player \leftarrow \text{game.To-Move}(state)
value, move \leftarrow \textsc{Max-Value}(game, state)
return move

function \textsc{Max-Value}(game, state) returns a \text{(utility, move)} pair
if \text{game.Is-Terminal}(state) then return \text{game.Utility}(state, player), \text{null}
v \leftarrow -\infty
for each a in \text{game.Actions}(state) do
  v2, a2 \leftarrow \textsc{Min-Value}(game, \text{game.Result}(state, a))
  if $v2 > v$ then
    v, move \leftarrow v2, a
return v, move

function \textsc{Min-Value}(game, state) returns a \text{(utility, move)} pair
if \text{game.Is-Terminal}(state) then return \text{game.Utility}(state, player), \text{null}
v \leftarrow +\infty
for each a in \text{game.Actions}(state) do
  v2, a2 \leftarrow \textsc{Max-Value}(game, \text{game.Result}(state, a))
  if $v2 < v$ then
    v, move \leftarrow v2, a
return v, move
Game Tree Pruning

- Chess: branching factor $b \approx 35$, ply $\approx 80$. $35^{80} \approx 10^{123}$
- We don’t need to explore the whole game tree!
Alpha-Beta Pruning

• General principle:
  – A player has a choice of moving to n
  – However, m’ is a better choice for player. Player choice m’
  – Or m is a better choice for player. Player choice m
  – Player will never move to n. So, we can prune it.
Alpha-Beta Pruning

- Can be applied to trees of any depth
- Prune entire subtrees or leaves
- $\alpha$ : MAX’s best choice we have found along the path
- $\beta$ : MIN’s best choice we have found along the path
function ALPHA-BETA-SEARCH(game, state) returns an action
player ← game.To-MOVE(state)
value, move ← MAX-VALUE(game, state, −∞, +∞)
return move

function MAX-VALUE(game, state, \(\alpha, \beta\)) returns a (utility, move) pair
if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
\(v \leftarrow -\infty\)
for each \(a\) in game.ACTIONS(state) do
\(v_2, a_2 \leftarrow\) MIN-VALUE(game, game.RESULT(state, a), \(\alpha, \beta\))
if \(v_2 > v\) then
\(v, move \leftarrow v_2, a\)
\(\alpha \leftarrow\) MAX(\(\alpha, v\))
if \(v \geq \beta\) then return \(v, move\)
return \(v, move\)
function MIN-VALUE(game, state, α, β) returns a (utility, move) pair
    if game.IS-TERMINAL(state) then return game.UTILITY(state, player), null
    v ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
        if v2 < v then
            v, move ← v2, a
            β ← MIN(β, v)
        if v ≤ α then return v, move
    return v, move
Alpha-Beta Pruning Properties

• State can be pruned because it makes no difference to the outcome (minimax value for the root)
• Effectiveness is highly dependent on the child ordering
• With **perfect** ordering:
  – Time complexity drop from $O(b^m)$ to $O(b^{m/2})$
  – Double solvable depth
• Random move ordering is about $O(b^{3m/4})$
Demo

- Minimax
- Alpha-Beta pruning
- Iterative Deepening
- Demo
Resource Limits

• Resource is Limit, game tree is way too big!
• Solution:
  – Use a heuristic evaluation function
  – Replace the UTILITY function with EVAL
  – Terminal test ➔ Cutoff test
  – Search only a limited depth in the tree (Depth-limited search)
• Cutoff test
  – Return true for terminal states
  – Decide to cut off the search
Shannon’s Strategies

• Type A strategy
  – Consider only a certain depth in the search tree
  – Use EVAL to estimate the utility
  – Explores a **wide** but **shallow** portion of the tree

• Type B strategy
  – Ignores moves that look bad
  – Follows promising lines “as far as possible”
  – Explores a **deep** but **narrow** portion of the tree

• Examples: Chess is Type A and Go is Type B
Heuristic Alpha-Beta Tree Search

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

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

• Less computation time (Not too long!)
• Should be strongly correlated with the actual **chances** of winning. $\text{Eval}(s) > \text{Eval}(s')$
• Return an **estimated** value about outcome (Expected utility of state $s$ to player $p$)
• A linear combination of **features**. (Weighted linear function)

$$\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$
Evaluation Functions

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

- \( f_i(s) \) is a feature extracted from the input state \( s \)
- \( w_i \) is a weight assigned to a feature
- Example: Checker
  - Features: # of agent pawns, # of agent kings, # of opponent pawns, and # of opponent kings
  - Select positive weights for agent and negative weights for opponents
  - Kings has more weights than pawns

\[ \text{Eval}(s) = 2 \cdot \text{agent_kings}(s) + \text{agent_pawns}(s) - 2 \cdot \text{opponent_kings}(s) - \text{opponent_pawns}(s) \]

Features and weights come from human experience or machine learning, a domain-specific and approximate estimate of the value \( V_{\text{minimax}}(s) \).

Fine-tuning  Experimenting
Heuristic Evaluation Functions

• Cutting off Search
  If game.Is-Cutoff(state, depth) the return game.Eval(state, player), null
  – Set a fixed depth limit
  – Iterative deepening
  – Use a transposition table

• Forward pruning
  – Prunes possible poor moves
  – Saves computation time at the risk
    • Could possibly prune good moves
Heuristic Evaluation Functions

• Beam search
  – Consider only a “beam” of the n best moves
  – Also could possibly prune good moves

• PROBCUT: probabilistic cut
  – Use statistics gained from prior experience
  – Prune nodes that are probably no need to be considered
Search vs Lookup

• Create a table and lookup
• Opening sequences most often lead to a win
• Policy: Map every possible state to the best move in that state
Multiplayer (3) Game Tree

to move
A
B
C
A

(1, 2, 6)  (1, 2, 6)  (1, 2, 6)  (1, 2, 6)  (1, 2, 6)  (1, 2, 6)  (1, 2, 6)
Reading and Next Class

• Adversarial Search and Games: AIMA 5.1-5.3
• Next:
  – Expectimax: AIMA 5.5
  – Utilities: AIMA 16.1-16.3