CS 221 Section 5: Games

Jerry Qu & Will Deaderick

Section Contents

- Minimax Search, w/ example
- Evaluation Functions
- Alpha-Beta Pruning, w/ example
- Alpha-Beta practice problem
- Game Theory practice problem
- MCTS & AlphaZero (time permitting)

Minimax

- Applicable to two-player zero-sum games (and other adversarial games)
- Characteristics:
    - Players take turns
    - Utility comes from end state, no intermediate utility
- Overall goal is to model outcomes when both players play optimally
- When state transitions are not deterministic, we use *expectiminimax*.
- Complexity: O(d) space, O(b^(2d)) time, for depth d, branching factor b

Evaluation Functions

- Goal is to provide some estimate of $V_{minimax}(s)$ for state s.
- Oftentimes requires domain knowledge since we can't make additional moves.
- Because of large time complexity for searching games, evaluation functions are often necessary.
- Pitfalls: in some cases where utility can fluctuate (e.g. a piece capturing sequence in chess), evaluation functions may become inaccurate.

Alpha-Beta Pruning: Interpretation

- At each **“Max”** state, the “**Max**” player keeps track of **alpha**, a lower bound on her value for that state.

Alpha-Beta Pruning: Interpretation

- At each **“Max”** state, the “**Max**” player keeps track of **alpha**, a lower bound on her value for that state.
    - Values lower than **alpha** don’t interest **“Max”**

Alpha-Beta Pruning: Interpretation

- At each **“Max”** state, the “**Max**” player keeps track of **alpha**, a lower bound on her value for that state.
    - Values lower than **alpha** don’t interest **“Max”**

- At each **“Min”** state, the “**Min**” player keeps track of **beta**, an upper bound on her value for that state.

Alpha-Beta Pruning: Interpretation

- At each **“Max”** state, the “**Max**” player keeps track of **alpha**, a lower bound on her value for that state.
  - Values lower than **alpha** don’t interest **“Max”**
- At each **“Min”** state, the “**Min**” player keeps track of **beta**, an upper bound on her value for that state.
  - Values larger than **beta** don’t interest **“Min”**

Alpha-Beta Pruning: Rules

- Start with **alpha** as negative infinity, and **beta** as infinity
- Propagate **alpha** and **beta** down the search tree
- At each **“Max”** node, update **alpha** if we find a <u>larger</u> (than **alpha**) child leaf value, child alpha, or child beta
- At each **“Min”** node, update **beta** if we find a <u>smaller</u> (than **beta)** child leaf value, child alpha, and child beta
- If **alpha >= beta**, prune!

Alpha-Beta Pruning: Intuition

- We can prune beneath a **“Max”** node if that **“Max”** node’s parent is already guaranteed a better (i.e. smaller) value than any of the **“Max”** node’s children.

Alpha-Beta Pruning: Intuition

- We can prune beneath a **“Max”** node if that **“Max”** node’s parent is already guaranteed a better (i.e. smaller) value than any of the **“Max”** node’s children.

- We can prune beneath a **“Min”** node if that **“Min”** node’s parent is already guaranteed a better (i.e. larger) value than any of the **“Min”** node’s children.

Extensions

- <u>Beam search (“forward pruning”)</u> - use an evaluation function to only expand the most promising child nodes, effectively lowering the branching factor

Extensions

- Beam search ("forward pruning") - use an evaluation function to only expand the most promising child nodes, effectively lowering the branching factor
    - Pros: similar to human intelligence, computational efficiency

Extensions

- <u>Beam search ("forward pruning")</u> - use an evaluation function to only expand the most promising child nodes, effectively lowering the branching factor
    - Pros: similar to human intelligence, computational efficiency
    - Cons: similar to human intelligence, no more guarantees

Extensions

- Beam search ("forward pruning") - use an evaluation function to only expand the most promising child nodes, effectively lowering the branching factor

    - Pros: similar to human intelligence, computational efficiency

    - Cons: similar to human intelligence, no more guarantees

- Dynamic move ordering - Manually specify which moves to expand first, or use an evaluation function, or keep track of which moves have been most successful in the past

Game Theory

- Applicable to single-move simultaneous games (covered already) and more.
- Defined by a set of players, actions, and utility matrix.
- We can have *pure strategies* or *mixed strategies*.
- Nash Equilibrium: a (pure or mixed) strategy for all players such that no player can gain utility by changing their strategy.
    - In any finite-player game with finite actions, there's ALWAYS at least one Nash Equilibrium.