CSE 473: Artificial Intelligence
Spring 2017

Adversarial Search

Dieter Fox
Based on slides adapted Luke Zettlemoyer, Dan Klein, Pieter Abbeel, Dan Weld, Stuart Russell or Andrew Moore

Adversarial Search

Game Playing State-of-the-Art 2017

- **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. Checkers is now solved!
- **Chess**: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Othello**: Human champions refuse to compete against computers, which are too good.
- **Go**: In March 2016, AlphaGo beats 9-dan master Lee Sedol (3 wins, 1 loss, 1 win). Combines Monte-Carlo tree search with deep reinforcement learning.
- **Poker**: In December 2016, computer beats professional players at no-limit Texas hold ‘em

Adversarial Search

- Many different kinds of games!
- **Choices**:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move in each state

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 x A → S
  - Terminal Test: S → {t,f}
  - Terminal Utilities: S x P → R
- Solution for a player is a policy: S → A

Zero-Sum Games

- **Zero-Sum Games**
  - Agents have opposite utilities (values on outcomes)
  - Lets us think of a single value that one maximizes and the other minimizes
  - Adversarial, pure competition

- **General Games**
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, & more are possible

Game Playing

- Many different kinds of games!
- **Choices**:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move in each state
**Single-Agent Trees**

![Single-Agent Trees Diagram]

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

---

**Value of a State**

![Value of a State Diagram]

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

---

**Adversarial Game Trees**

![Adversarial Game Trees Diagram]

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

---

**Minimax Values**

![Minimax Values Diagram]

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

---

**Tic-tac-toe Game Tree**

![Tic-tac-toe Game Tree Diagram]

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu

---

**Adversarial Search (Minimax)**

- **Deterministic, zero-sum games:**
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result

- **Minimax search:**
  - A state-space search tree
  - Players alternate turns
  - Compute each node’s minimax value: the best achievable utility against a rational (optimal) adversary

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu
Minimax Implementation

```python
def max_value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min_value(successor))
    return v

V(s) = max s' ∈SUCCESSORS(s) V(s')

V(s) = max s' ∈SUCCESSORS(s) V(s')
```

---

Concrete Minimax Example

```
max

min

max

A_1 A_2 A_3

A_1 A_2 A_3

A_1 A_2 A_3

3 12 8 2 4 6 14 5 2
```

---

Minimax Properties

- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity
  - O(b^m)
- Space complexity?
  - O(bm)
- For chess, b ≈ 35, m ≈ 100
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

---

α-β Pruning

- General configuration
  - α is the best value that
    MAX can get at any choice
    point along the current path
  - If n becomes worse than α,
    MAX will avoid it, so can
    stop considering n’s other
    children
  - Define β similarly for MIN

```
```
Alpha-Beta Pruning
Alpha-Beta Pruning Properties

- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong!
  - but, they are bounds
- Good child ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…

Alpha-Beta Implementation

α: MAX’s best option on path to root
β: MIN’s best option on path to root

```
def max_value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v,
            value(successor, α, β))
        if v ≥ β return v
    α = max(α, v)
    return v
```

```
def min_value(state, α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v,
            value(successor, α, β))
        if v ≤ α return v
    β = min(β, v)
    return v
```

Slide from Dan Klein & Pieter Abbeel - ai.berkeley.edu
**Resource Limits**

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - $\alpha - \beta$ reaches about depth 8 – decent chess program

**Evaluation Functions**

- Function which scores non-terminals

$$Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$$

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. $f_i(s) = (\text{num white queens} - \text{num black queens})$, etc.

**Which algorithm?**

- $\alpha - \beta$, depth 4, simple eval fun

**Which algorithm?**

- $\alpha - \beta$, depth 4, better eval fun