Adversarial Search

Minimax, pruning, Expectimax

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

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.
- 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 s₀)
  - 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
**Single-Agent Trees**

- Non-Terminal States:
  - The best achievable outcome (utility) from that state

**Value of a State**

- Terminal States: $V(s) = \text{likes}$
- Non-Terminal States: $V(s) = \max_{s' \in \text{successors}(s)} V(s')$

**Adversarial Game Trees**

- States Under Agent’s Control:
  - Terminal States: $V(s) = \text{likes}$
  - Minimax Values: computed recursively

**Minimax Values**

- States Under Opponent’s Control:
  - Terminal States: $V(s) = \text{likes}$
  - Minimax values: part of the game

**Tic-tac-toe Game Tree**

- 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

**Adversarial Search (Minimax)**

- Minimax values: computed recursively
- Terminal values: part of the game
Minimax Implementation

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

V(s) = \max_{s' \in successors(s)} V(s')
```

```python
def min_value(state):
    initialize v = +\infty
    for each successor of state:
        v = min(v, max_value(successor))
    return v

V(s') = \min_{s' \in successors(s')} V(s')
```

---

Concrete Minimax Example

```
max

\[ \begin{array}{c}
\text{min} \\
\text{A}_1 \\
3 \quad 12 \quad 8 \quad 2 \\
A_2 \\
4 \quad 6 \quad 14 \quad 5 \\
\end{array} \]
```

---

Minimax Properties

- **Optimal?**
  - Yes, against perfect player. Otherwise?

- **Time complexity**
  - O(b^m)

- **Space complexity?**
  - O(bm)

- For chess, b \approx 35, m \approx 100
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?

---

Pruning Example

```
A_1

[\sim \infty, 2]
```

```
A_2

3 \quad 12 \quad 8 \quad 2
```

---

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

```
Player
\[ \begin{array}{c}
\text{Opponent} \\
A_1 \\
\text{A}_2 \\
\text{A}_3 \\
? \\
\end{array} \]
```
**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 $O(b^{m/2})$
  - Full search of, e.g. chess, is still hopeless...

**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

**Alpha-Beta Implementation**

\[
\text{def min-value}(\text{state, } \alpha, \beta):
\]

initialize $v = +\infty$

for each successor of state:

$v = \min(v, \text{value(successor, } \alpha, \beta))$

if $v \leq \alpha$ return $v$

$\beta = \min(\beta, v)$

return $v$

\[
\text{def max-value}(\text{state, } \alpha, \beta):
\]

initialize $v = -\infty$

for each successor of state:

$v = \max(v, \text{value(successor, } \alpha, \beta))$

if $v \geq \beta$ return $v$

$\alpha = \max(\alpha, v)$

return $v$

\[\alpha: \text{MAX's best option on path to root}\]
\[\beta: \text{MIN's best option on path to root}\]

**Which algorithm?**

- $\alpha-\beta$, depth 4, better eval fun
Worst-Case vs. Average Case

Idea: Uncertain outcomes controlled by chance, not an adversary!

Expectimax Search

- Why wouldn’t we know what the result of an action will be?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip
- Values should now reflect average-case (expectimax) outcomes, not worst-case (minimax) outcomes
- Expectimax search: compute the average score under optimal play
  - Chance nodes are like min nodes but the outcome is uncertain
  - Calculate their expected utilities
  - I.e. take weighted average (expectation) of children
- Later, we’ll learn how to formalize the underlying uncertain-result problems as Markov Decision Processes

Expectimax Pseudocode

```python
def value(state):
    if the state is a terminal state: return the state’s utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

Minimax vs Expectimax

3 ply look ahead, ghosts move randomly

Expectimax Pseudocode

```python
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```
Expectimax Example

Expectimax Pruning?

Depth-Limited Expectimax

Estimate of true expectimax value (which would require a lot of work to compute)