Adversarial Search


Chess: 1997: Deep Blue defeats human champion Gary Kasparov in a six-game match. Deep Blue examined 200 M positions per second, used very sophisticated evaluation and unorthodox methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.

Go: Human champions are now starting to be challenged by machines. In 2016, Google programs used pattern knowledge bases, but big recent advances use Monte Carlo (randomized) expansion methods and deep learning. Match with World Champion Lee Sedol in March!

Video of Demo Mystery Pacman

Adversarial Games

Many different kinds of games!

Axes:
- Deterministic or stochastic?
- One, two, or more players?
- Zero sum?
- Perfect information (can you see the state)?

Want algorithms for calculating a strategy (policy) which recommends a move from 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: SxA → S
- Terminal Test: S → {t,f}
- Terminal Utilities: SxA → R

Solution for a player is a policy: S → A

Zero-Sum Games

- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - Let's 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, and more are all possible
- More later on non-zero-sum games

Adversarial Search

Single-Agent Trees

Types of Games

- Many different kinds of games!

- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Zero sum?
  - Perfect information (can you see the state)?

- Want algorithms for calculating a strategy (policy) which recommends a move from each state
Value of a State

Value of a state: The best achievable outcome (utility) from that state

Non-Terminal States:

Terminal States:

Adversarial Game Trees

States Under Agent's Control:

States Under Opponent’s Control:

Minimax Values

Minimax Implementation

Minimax Efficiency

Tic-Tac-Toe Game Tree

Adversarial Search (Minimax)

Minimax Example

Minimax Efficiency

Minimax Implementation (Dispatch)
Minimax Properties

Optimal against a perfect player. Otherwise?

Minimax Properties

Resource Limits

▪ Problem: In realistic games, cannot search to leaves!
▪ Solution: Depth-limited search
  + Instead, search only to a limited depth in the tree
  + Replace terminal utilities with an evaluation function for non-terminal positions
▪ Example:
  + Suppose we have 100 seconds, can explore 10K nodes/sec
  + So can check 1M nodes per move
  + α-β reaches about depth 8 – decent chess program
▪ Guarantee of optimal play is gone
▪ More plies makes a BIG difference
▪ Use iterative deepening for an anytime algorithm

Minimax Properties

Resource Limits

Depth Matters

▪ Evaluation functions are always imperfect
▪ The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function matters
▪ An important example of the tradeoff between complexity of features and complexity of computation

Minimax Properties

Resource Limits

Depth Matters

Evaluation Functions
Evaluation Functions

- Evaluation functions score non-terminals in depth-limited search
- Ideal function: returns the actual minimax value of the position
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = \omega_1 f_1(s) + \omega_2 f_2(s) + \ldots + \omega_n f_n(s) \]
  e.g. \[ f_1(s) = \text{num white queens} - \text{num black queens} \]

Why Pacman Starves

- A danger of replanning agents!
  - He knows his score will go up by eating the dot now (west, east)
  - He knows his score will go up just as much by eating the dot later (east, west)
  - There are no point-scoring opportunities after eating the dot (within the horizon, two here)
  - Therefore, waiting seems just as good as eating: he may go east, then back west in the next round of replanning!

Video of Demo Thrashing (d=2)

Evaluation for Pacman

Video of Demo Thrashing -- Fixed (d=2)

Video of Demo Smart Ghosts (Coordination)

Video of Demo Smart Ghosts (Coordination) – Zoomed In

Game Tree Pruning

Minimax Example
Minimax Pruning

- General configuration (MIN version)
  - We're computing the MIN-VALUE at some node \( n \)
  - We're looping over \( n \)'s children
  - \( n \)'s estimate of the children's min is dropping
  - Who cares about \( n \)'s value? MAX
  - Let \( a \) be the best value that MAX can get at any choice point along the current path from the root
  - If \( n \) becomes worse than \( a \), MAX will avoid it, so we can stop considering \( n \)'s other children (it's already bad enough that it won't be played)
  - MAX version is symmetric

Alpha-Beta Pruning

- ▪ General configuration (MIN version)
- ▪ We're computing the MIN-VALUE at some node \( n \)
- ▪ We're looping over \( n \)'s children
- ▪ \( n \)'s estimate of the children's min is dropping
- ▪ Who cares about \( n \)'s value? MAX
- ▪ Let \( a \) be the best value that MAX can get at any choice point along the current path from the root
- ▪ If \( n \) becomes worse than \( a \), MAX will avoid it, so we can stop considering \( n \)'s other children (it's already bad enough that it won't be played)
- ▪ MAX version is symmetric

Alpha-Beta Implementation

```python
def min_value(state, \( \alpha \), \( \beta \)):
    initialize v = \( -\infty \)
    for each successor of state:
        v = min(v, value(successor, \( \alpha \), \( \beta \)))
    if v \( \leq \) \( \alpha \)
        return v
    \( \beta \) = min(\( \beta \), v)
    return v

def max_value(state, \( \alpha \), \( \beta \)):
    initialize v = \( +\infty \)
    for each successor of state:
        v = max(v, value(successor, \( \alpha \), \( \beta \)))
    if v \( \geq \) \( \beta \)
        return v
    \( \alpha \) = max(\( \alpha \), v)
    return v
```

Alpha-Beta Pruning Properties

- This pruning has no effect on minimax value computed for the root!
- Values of intermediate nodes might be wrong
  - Important: children of the root may have the wrong value
  - So the most naive version won't let you do action selection
- 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...
- This is a simple example of metareasoning (computing about what to compute)

Alpha-Beta Quiz

```python
def max_value(state, \( \alpha \), \( \beta \)):
    initialize v = \( -\infty \)
    for each successor of state:
        v = max(v, value(successor, \( \alpha \), \( \beta \)))
    if v \( \geq \) \( \beta \)
        return v
    \( \alpha \) = max(\( \alpha \), v)
    return v

def min_value(state, \( \alpha \), \( \beta \)):
    initialize v = \( +\infty \)
    for each successor of state:
        v = min(v, value(successor, \( \alpha \), \( \beta \)))
    if v \( \leq \) \( \alpha \)
        return v
    \( \beta \) = min(\( \beta \), v)
    return v
```

Alpha-Beta Quiz 2

```python
def max_value(state, \( \alpha \), \( \beta \)):
    initialize v = \( -\infty \)
    for each successor of state:
        v = max(v, value(successor, \( \alpha \), \( \beta \)))
    if v \( \geq \) \( \beta \)
        return v
    \( \alpha \) = max(\( \alpha \), v)
    return v

def min_value(state, \( \alpha \), \( \beta \)):
    initialize v = \( +\infty \)
    for each successor of state:
        v = min(v, value(successor, \( \alpha \), \( \beta \)))
    if v \( \leq \) \( \alpha \)
        return v
    \( \beta \) = min(\( \beta \), v)
    return v
```