CS 188: Artificial Intelligence Spring 2009

Lecture 6: Adversarial Search
2/5/2009

John DeNero – UC Berkeley
Slides adapted from Dan Klein, Stuart Russell or Andrew Moore

Announcements

- Written Assignment 1:
  - Due Tuesday in lecture!
  - No late days for written assignments
  - Printed copies will be here after class
- Countdown to math:
  - Markov decision processes are 3 lectures away
- Project 2:
  - Posted tonight; due Wednesday, 2/18
  - Material from today and next Tuesday
- Midterm on Thursday, 3/19, at 6pm in 10 Evans

Game Playing

- Many different kinds of games!
- Axes:
  - 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

Example: Peg Game

Jump each tee and remove it:

- Leave only one -- you're genius
- Leave two and you're purty smart
- Leave three and you're just plain dumb
- Leave four or mor'n you're an EG-NO-RA-MOOSE

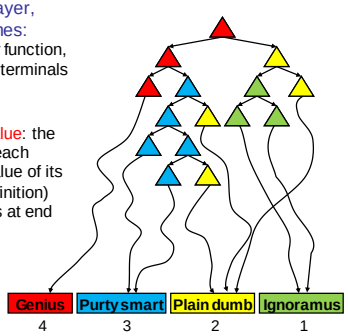
Looks like a search problem:

- Has a start state, goal test, successor function
- But the goal cost is not the sum of step costs!
- Are all of our search algorithms useless here?

Instructions from Cracker Barrel Old Country Store

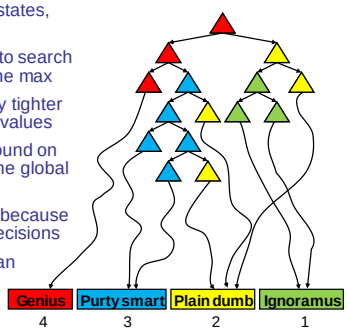
Deterministic Single-Player

- Deterministic, single player, perfect information games:
  - Start state, successor function, terminal test, utility of terminals
- Max search:
  - Each node stores a value: the best outcome it can reach
  - This is the maximal value of its children (recursive definition)
  - No path sums; utilities at end
- After search, can pick move that leads to the best outcome



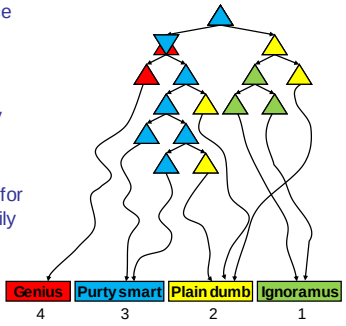
Properties of Max Search

- Terminology: terminal states, node values, policies
- Without bounds, need to search the entire tree to find the max
- Computes successively tighter lower bounds on node values
- With a known upper bound on utility, can stop when the global max is attained
- Nodes are max nodes because one agent is making decisions
- Caching max values can speed up computation

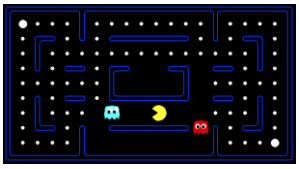


Uses of a Max Tree

- Can select a sequence of moves that maximizes utility
- Can recover optimally from bad moves
- Can compute values for certain scenarios easily



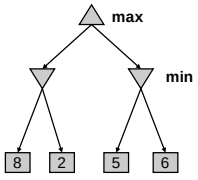
Adversarial Search



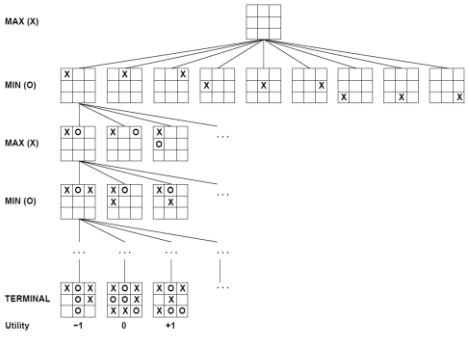
[DEMO: mystery pacman]

Deterministic Two-Player

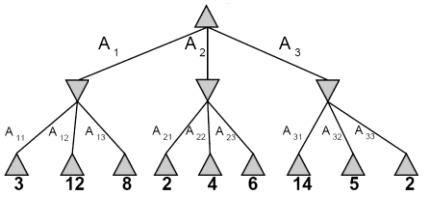
- Deterministic, zero-sum games:
  - tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest minimax value: best achievable utility against a rational adversary



Tic-tac-toe Game Tree



Minimax Example



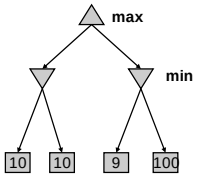
Minimax Search

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return *v*

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(s))
  return *v*

Minimax Properties

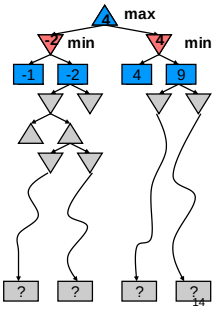
- Optimal against a perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm)
- For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - Lots of approximations and pruning



[DEMO: minVsExp]

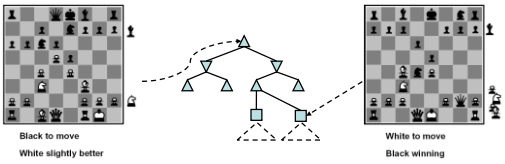
Resource Limits

- 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
- More plies makes a BIG difference
  - [DEMO: limitedDepth]
- 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
  - Deep Blue sometimes reached depth 40+



Evaluation Functions

- Function which scores non-terminals

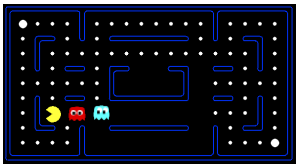


- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation for Pacman

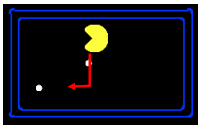


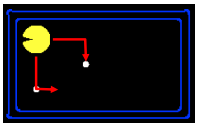
[DEMO: thrashing, smart ghosts]

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

Why Pacman Starves

- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating





Iterative Deepening

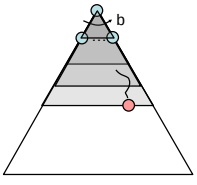
Iterative deepening uses DFS as a subroutine:

1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
2. If "1" failed, do a DFS which only searches paths of length 2 or less.
3. If "2" failed, do a DFS which only searches paths of length 3 or less.

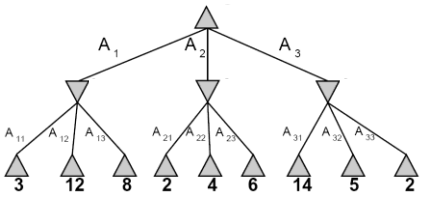
   ….and so on.

This works for single-agent search as well!

Why do we want to do this for multiplayer games?

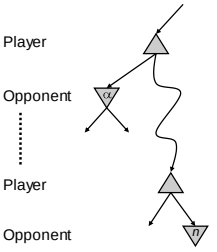


α-β Pruning Example



α-β Pruning

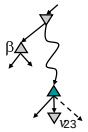
- General configuration
  - α is the best value that MAX can get at any choice point along the current path
  - If *n* becomes worse than α, MAX will avoid it, so can stop considering *n*'s other children
  - Define β similarly for MIN



α-β Pruning Pseudocode

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
  return v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

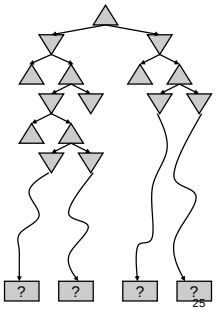


α-β Pruning Properties

- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong
- Good move 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

More Metareasoning Ideas

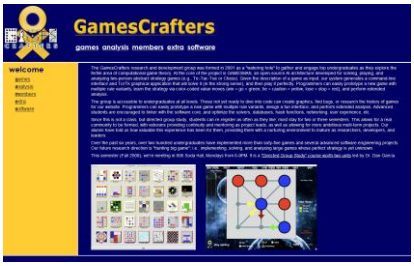
- Forward pruning – prune a node immediately without recursive evaluation
- Singular extensions – explore only one action that is clearly better than others. Can alleviate horizon effects
- Cutoff test – a decision function about when to apply evaluation
- Quiescence search – expand the tree until positions are reached that are quiescent (i.e., not volatile)



Game Playing State-of-the-Art

- **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.
- **Othello:** human champions refuse to compete against computers, which are too good.
- **Go:** human champions refuse to compete against computers, which are too bad. In go, b > 300, so most programs use pattern knowledge bases to suggest plausible moves.
- **Pacman:** unknown

GamesCrafters



http://gamescrafters.berkeley.edu/

27