CSE 573: Artificial Intelligence
Autumn 2012

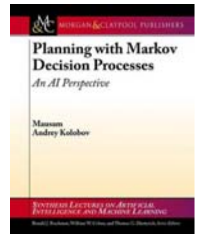
Adversarial Search

Dan Weld

Based on slides from
Dan Klein, Stuart Russell, Andrew Moore and Luke Zettlemoyer

Logistics 1

- Dan in Boston (UIST) on Wed 10/10
- Guest lecture by Mausam



Logistics 2

- PS 1 due ~~Tues 10/9~~ Thurs 10/11
- PS 2 due Tues 10/16
- PS 3 due Tues 10/23

Outline

- Adversarial Search
  - Minimax search
  - α-β search
  - Evaluation functions
  - Expectimax

Types of Games

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	stratego	bridge, poker, scrabble, nuclear war

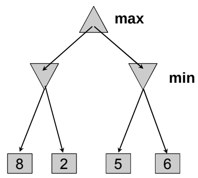
Number of Players? 1, 2, …?

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

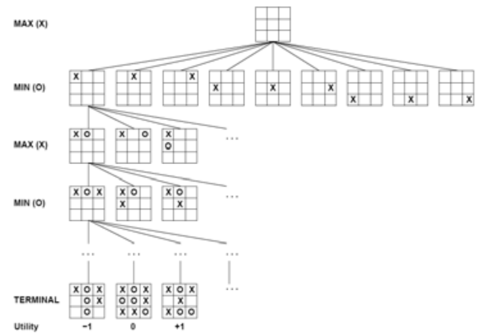
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Zero-sum games
  - One player maximizes result
  - The other minimizes result



- **Minimax search**
  - A state-space search tree
  - Players alternate
  - Choose move to position with highest minimax value = best achievable utility against best play

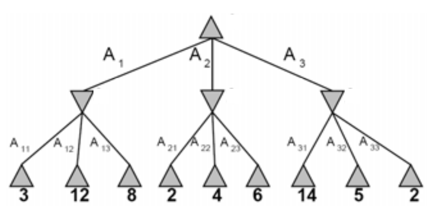
Tic-tac-toe Game Tree



Minimax Example

max

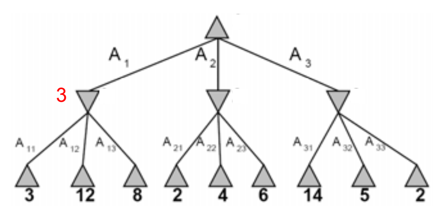
min



Minimax Example

max

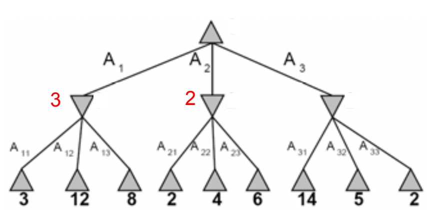
min



Minimax Example

max

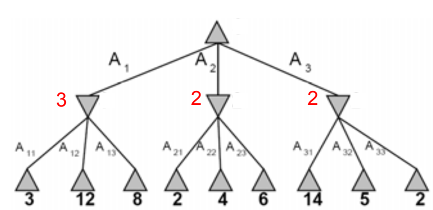
min



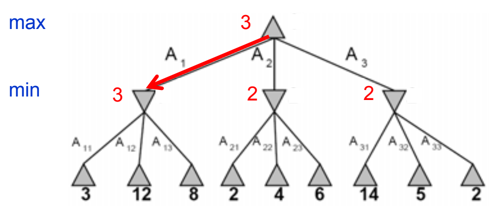
Minimax Example

max

min



Minimax Example

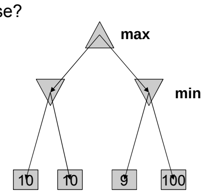


Minimax Search

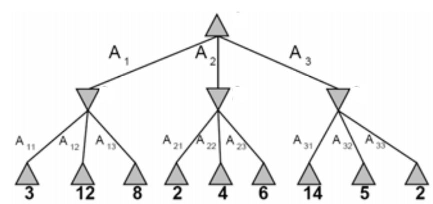
```
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return v

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

Minimax Properties

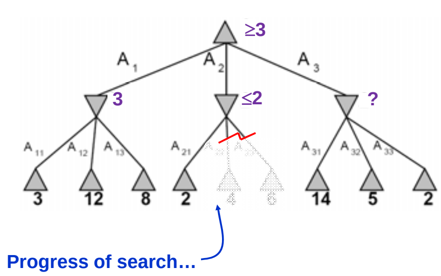
- Optimal?
  - Yes, against perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm)
- For chess, b ~ 35, m ~ 100
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



Do We Need to Evaluate Every Node?

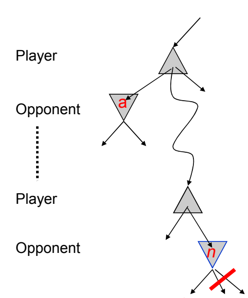


α-β Pruning Example



α-β Pruning

- 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 Pseudocode

inputs: *state*, current game state
  α, value of best alternative for MAX on path to *state*
  β, value of best alternative for MIN on path to *state*
returns: *a utility value*

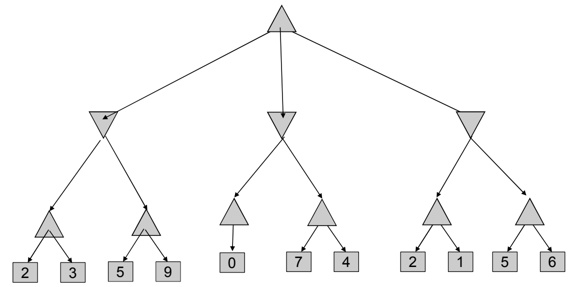
```
function MAX-VALUE(state,α,β)
  if TERMINAL-TEST(state) then
      return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
      v ← MAX(v, MIN-VALUE(s,α,β))
      if v ≥ β then return v
      α ← MAX(α,v)
  return v
```

```
function MIN-VALUE(state,α,β)
  if TERMINAL-TEST(state) then
      return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
      v ← MIN(v, MAX-VALUE(s,α,β))
      if v ≤ α then return v
      β ← MIN(β,v)
  return v
```

At max node:
Prune if $v \geq \beta$;
Update α

At min node:
Prune if $\alpha \leq v$;
Update β

Alpha-Beta Pruning Example



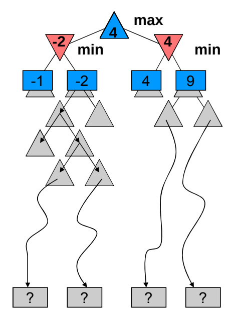
α is MAX's best alternative here or above
β is MIN's best alternative here or above

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…

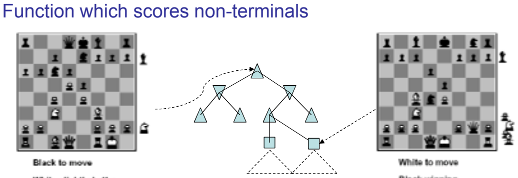
Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with heuristic 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



Heuristic Evaluation Function

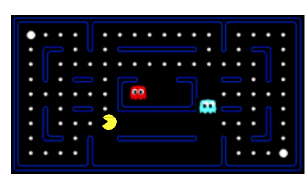
- Function which scores non-terminals



- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.

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

Evaluation for Pacman



What features would be good for Pacman?

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

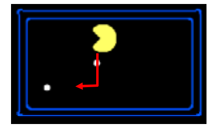
Which algorithm?

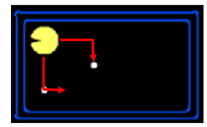
α-β, depth 4, simple eval fun

QuickTime™ and a
GIF decompressor
are needed to see this picture.

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





Which algorithm?

α-β, depth 4, better eval fun

QuickTime™ and a
GIF decompressor
are needed to see this picture.

Which Algorithm?

Minimax: no point in trying

QuickTime™ and a
GIF decompressor
are needed to see this picture.

3 ply look ahead, ghosts move randomly

Which Algorithm?

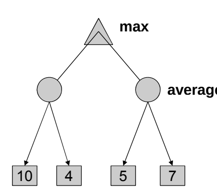
Expectimax: wins some of the time

QuickTime™ and a
GIF decompressor
are needed to see this picture.

3 ply look ahead, ghosts move randomly

Stochastic Single-Player

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, mine locations
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Max nodes as before
  - Chance nodes take average (expectation) of value of children



Soon, we'll generalize this problem to a Markov Decision Process

Maximum Expected Utility

- Why should we average utilities? Why not minimax?
- Principle of maximum expected utility: an agent should chose the action which maximizes its expected utility, given its knowledge
  - General principle for decision making
  - Often taken as the definition of rationality
  - We'll see this idea over and over in this course!
- Let's decompress this definition…

Reminder: Probabilities

- A random variable models an event with unknown outcome
- A probability distribution assigns weights to outcomes
- Example: traffic on freeway?
  - Random variable: T = whether there's traffic
  - Outcomes: T in {none, light, heavy}
  - Distribution: P(T=none) = 0.25, P(T=light) = 0.55, P(T=heavy) = 0.20
- Some laws of probability (read ch 13):
  - Probabilities are always non-negative
  - Probabilities over all possible outcomes sum to one
- As we get more evidence, probabilities may change:
  - P(T=heavy) = 0.20, P(T=heavy | Hour=5pm) = 0.60
  - We'll talk about methods for reasoning and updating probabilities later

What are Probabilities?

- Objectivist / frequentist answer:
  - Averages over repeated *experiments*
  - E.g. empirically estimating P(rain) from historical observation
  - E.g. pacman's estimate of what the ghost will do, given what it has done in the past
  - Assertion about how future experiments will go (in the limit)
  - Makes one think of *inherently random* events, like rolling dice
- Subjectivist / Bayesian answer:
  - Degrees of belief about unobserved variables
  - E.g. an agent's belief that it's raining, given the temperature
  - E.g. pacman's belief that the ghost will turn left, given the state
  - Often *learn* probabilities from past experiences (more later)
  - New evidence *updates beliefs* (more later)

Uncertainty Everywhere

- Not just for games of chance!
  - I'm sick: will I sneeze this minute?
  - Email contains "FREE!": is it spam?
  - Tummy hurts: have appendicitis?
  - Robot rotated wheel three times: how far did it advance?
- Sources of uncertainty in random variables:
  - Inherently random process (dice, opponent, etc)
  - Insufficient or weak evidence
  - Ignorance of underlying processes
  - Unmodeled variables
  - The world's just noisy – it doesn't behave according to plan!

Review: Expectations

- Real valued functions of random variables:

$$f : X \rightarrow R$$

- Expectation of a function of a random variable

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x)$$

- Example: Expected value of a fair die roll

X	P	f
1	1/6	1
2	1/6	2
3	1/6	3
4	1/6	4
5	1/6	5
6	1/6	6

$$1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6}$$

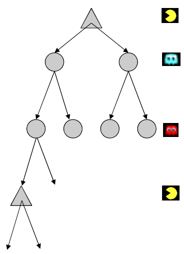
$$= 3.5$$

Utilities

- Utilities are functions from outcomes (states of the world) to real numbers that describe an agent's preferences
- Where do utilities come from?
  - In a game, may be simple (+1/-1)
  - Utilities summarize the agent's goals
  - Theorem: any set of preferences between outcomes can be summarized as a utility function (provided the preferences meet certain conditions)
- In general, we hard-wire utilities and let actions emerge (why don't we let agents decide their own utilities?)
- More on utilities soon…

Expectimax Search

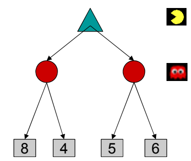
- In expectimax search, we have a probabilistic model of how the opponent (or environment) will behave in any state
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a node for every outcome out of our control: opponent or environment
  - The model might say that adversarial actions are likely!



- For now, assume for any state we magically have a distribution to assign probabilities to opponent actions / environment outcomes

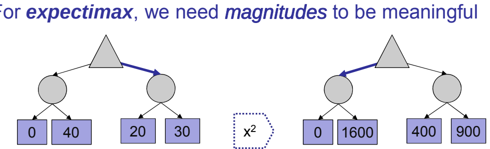
Expectimax Pseudocode

```
def value(s)
    if s is a max node return maxValue(s)
    if s is an exp node return expValue(s)
    if s is a terminal node return evaluation(s)

def maxValue(s)
    values = [value(s') for s' in successors(s)]
    return max(values)

def expValue(s)
    values = [value(s') for s' in successors(s)]
    weights = [probability(s, s') for s' in successors(s)]
    return expectation(values, weights)
```

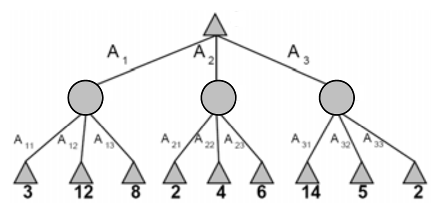


Expectimax Evaluation

- Evaluation functions quickly return an estimate for a node's true value (which value, expectimax or minimax?)
- For ***minimax***, evaluation function ***scale*** doesn't matter
  - We just want better states to have higher evaluations (ie, get the ordering right)
  - We call this insensitivity to monotonic transformations
- For ***expectimax***, we need ***magnitudes*** to be meaningful



Expectimax Pruning?

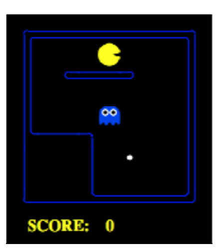


- Not easy
  - exact: need bounds on possible values
  - approximate: sample high-probability branches

Expectimax for Pacman

Results from playing 5 games

	Minimizing Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 493	Won 5/5 Avg. Score: 483
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503



Pacman does depth 4 search with an eval function that avoids trouble
Minimizing ghost does depth 2 search with an eval function that seeks Pacman

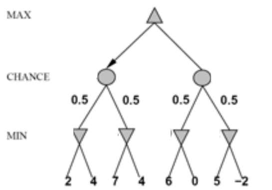
Expectimax for Pacman

- Notice that we've gotten away from thinking that the ghosts are trying to minimize pacman's score
- Instead, they are now a part of the environment
- Pacman has a belief (distribution) over how they will act

- Quiz: Can we see minimax as a special case of expectimax?
- Quiz: what would pacman's computation look like if we assumed that the ghosts were doing 1-ply minimax and taking the result 80% of the time, otherwise moving randomly?

Stochastic Two-Player

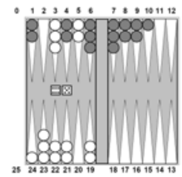
- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax



if *state* is a MAX node **then**
  return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a MIN node **then**
  return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
if *state* is a chance node **then**
  return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

Stochastic Two-Player

- Dice rolls increase *b*: 21 possible rolls with 2 dice
  - Backgammon: 20 legal moves
  - Depth 4 = 20 x (21 x 20)^3 = 1.2 x 10^9
- As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible…



- TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play

Multi-player Non-Zero-Sum Games

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (aka "back up") nodes from children
  - Can give rise to cooperation and competition dynamically…