

# CS 188: Artificial Intelligence

## Fall 2007

### Lecture 7: Adversarial Search

9/18/2007

Dan Klein – UC Berkeley

Many slides over the course adapted from either Stuart Russell or Andrew Moore

## Announcements

---

- Project 2 is up (Multi-Agent Pacman)
- SVN groups coming, watch web page
- Dan's office hours Tuesday moving to Friday 1-2pm

## Local Search

---

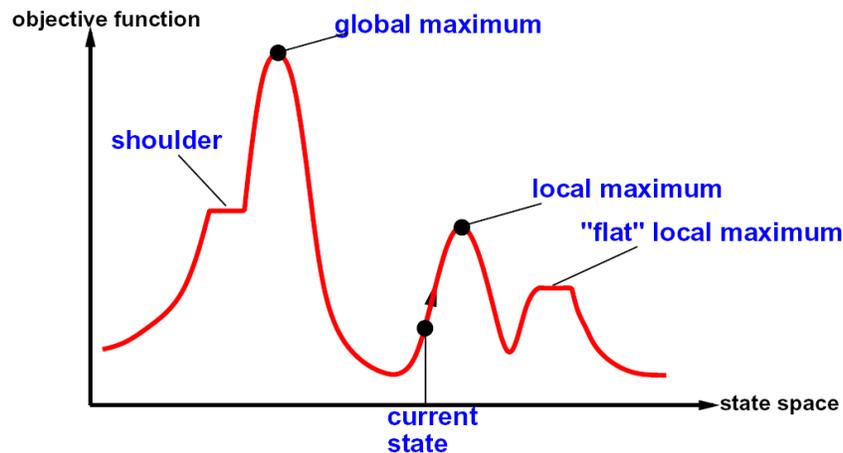
- Queue-based algorithms keep fallback options (backtracking)
- Local search: improve what you have until you can't make it better
- Generally much more efficient (but incomplete)

## Hill Climbing

---

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What's good about it?

## Hill Climbing Diagram



- Random restarts?
- Random sideways steps?

## Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

**function** SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

**inputs:** *problem*, a problem

*schedule*, a mapping from time to "temperature"

**local variables:** *current*, a node

*next*, a node

*T*, a "temperature" controlling prob. of downward steps

*current* ← MAKE-NODE(INITIAL-STATE[*problem*])

**for** *t* ← 1 **to** ∞ **do**

*T* ← *schedule*[*t*]

**if** *T* = 0 **then return** *current*

*next* ← a randomly selected successor of *current*

$\Delta E$  ← VALUE[*next*] − VALUE[*current*]

**if**  $\Delta E > 0$  **then** *current* ← *next*

**else** *current* ← *next* only with probability  $e^{\Delta E/T}$

# Simulated Annealing

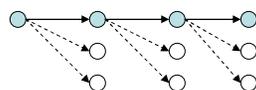
---

- Theoretical guarantee:
  - Stationary distribution:  $p(x) \propto e^{-\frac{E(x)}{kT}}$
  - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape, the less likely you are to every make them all in a row
  - People think hard about *ridge operators* which let you jump around the space in better ways

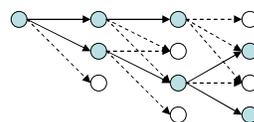
# Beam Search

---

- Like hill-climbing search, but keep K states at all times:



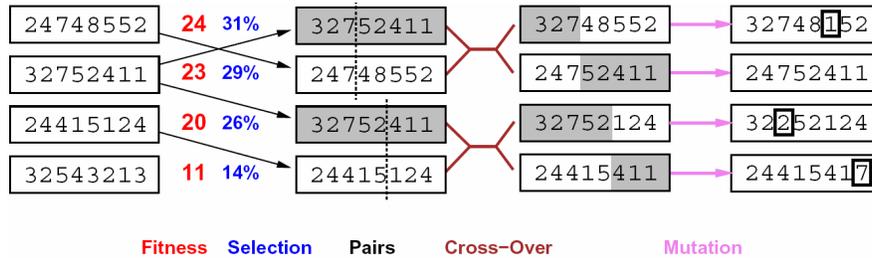
Greedy Search



Beam Search

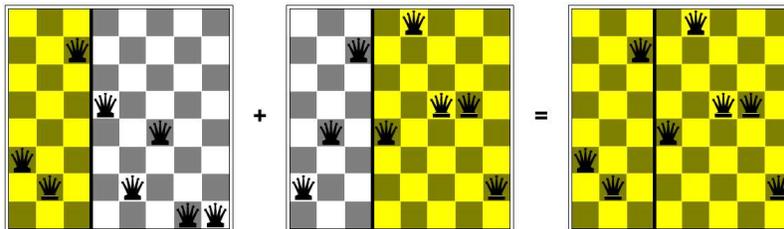
- Variables: beam size, encourage diversity?
- The best choice in MANY practical settings
- Complete? Optimal?
- Why do we still need optimal methods?

# Genetic Algorithms



- Genetic algorithms use a natural selection metaphor
- Like beam search (selection), but also have pairwise crossover operators, with optional mutation
- Probably the most misunderstood, misapplied (and even maligned) technique around!

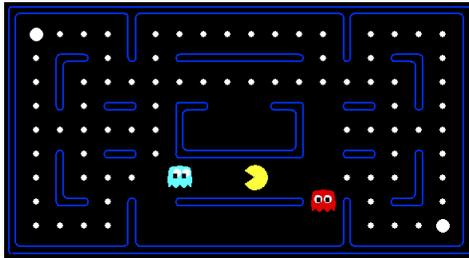
## Example: N-Queens



- Why does crossover make sense here?
- When wouldn't it make sense?
- What would mutation be?
- What would a good fitness function be?

# Adversarial Search

---



[DEMO 1]

## Game Playing State-of-the-Art

---

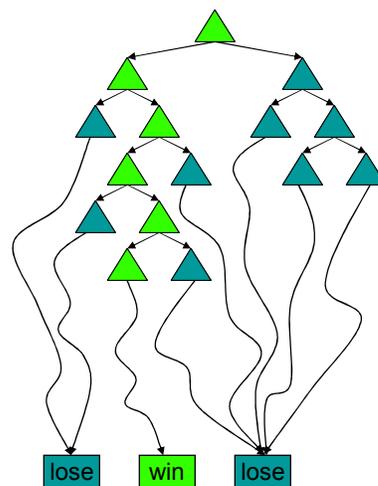
- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!
- **Chess:** Deep Blue defeated human world 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

# Game Playing

- **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

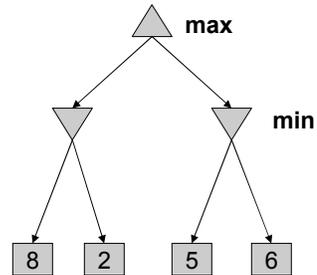
## Deterministic Single-Player?

- **Deterministic, single player, perfect information:**
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik's cube
- ... it's just search!
- **Slight reinterpretation:**
  - Each node stores the best outcome it can reach
  - This is the maximal outcome of its children
  - Note that we don't store path sums as before
- After search, can pick move that leads to best node

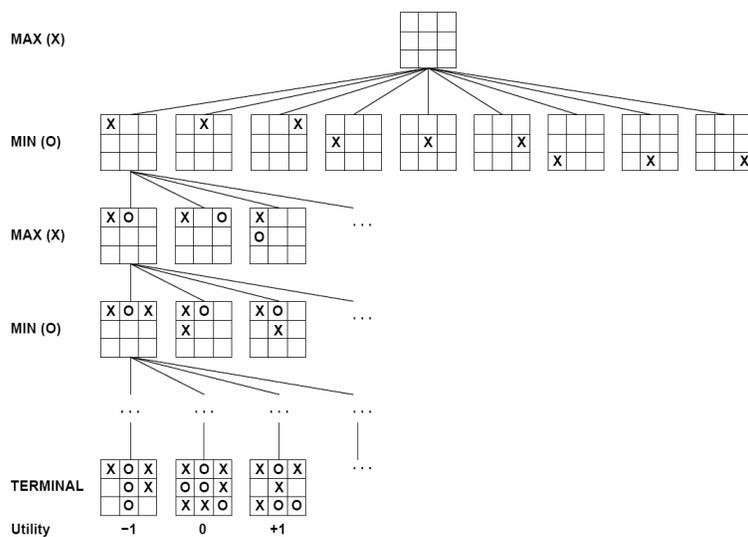


# Deterministic Two-Player

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

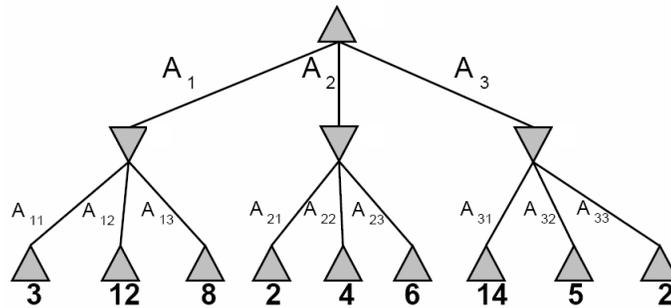


# Tic-tac-toe Game Tree



## 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  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$   
  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  $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$   
  return  $v$ 
```

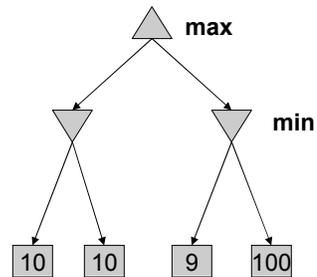
# Minimax Properties

- Optimal against a perfect player. Otherwise?

- Time complexity?
  - $O(b^m)$

- Space complexity?
  - $O(bm)$

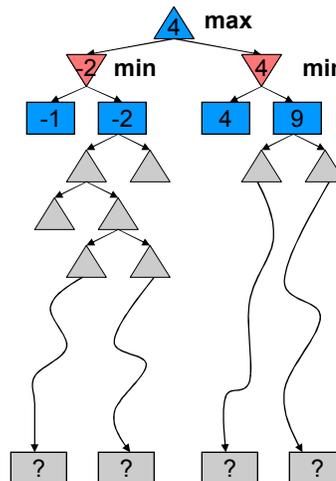
- For chess,  $b \approx 35$ ,  $m \approx 100$ 
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree?



[DEMO2:  
minVsExp]

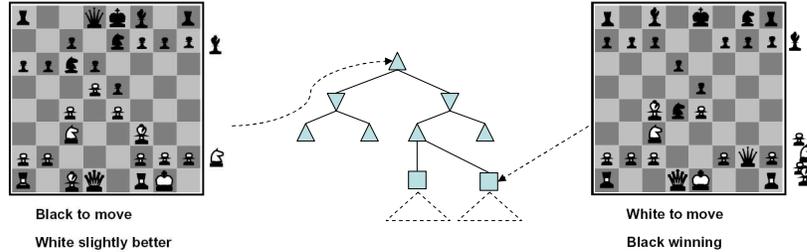
# Resource Limits

- Cannot search to leaves
- Limited search
  - Instead, search a limited depth of the 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 3: limitedDepth]
- 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

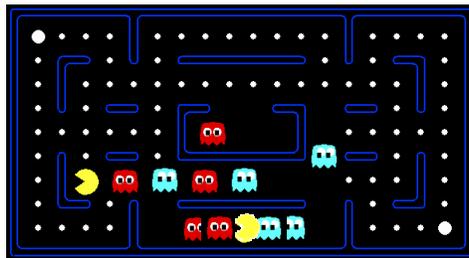


- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:

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

- e.g.  $f_1(s) = (\text{num white queens} - \text{num black queens})$ , etc.

# Evaluation for Pacman



[DEMO4: evalFunction,  
thrashing]

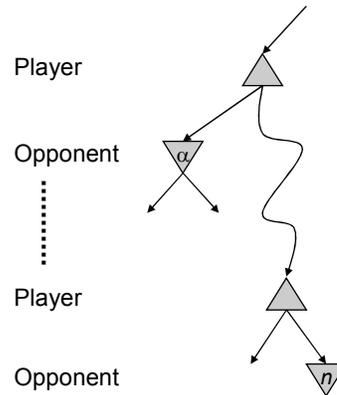
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$



# $\alpha$ - $\beta$ Pruning

## General configuration

- $\alpha$  is the best value the MAX can get at any choice point along the current path
- If  $n$  is worse than  $\alpha$ , MAX will avoid it, so prune  $n$ 's branch
- Define  $\beta$  similarly for MIN



# $\alpha$ - $\beta$ Pruning Pseudocode

```

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  $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$ 
  return  $v$ 

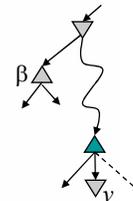
```

```

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return  $v$ 
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return  $v$ 

```



## $\alpha$ - $\beta$ Pruning Properties

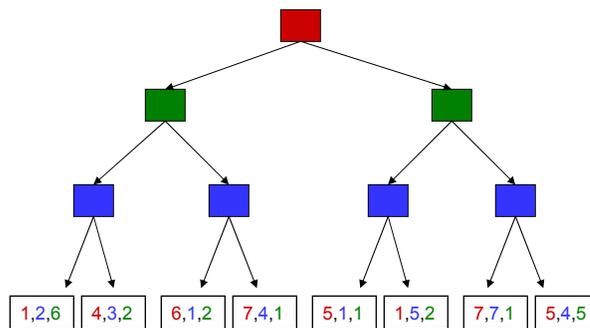
---

- Pruning has **no effect** on final result
- 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!
- A simple example of **metareasoning**, here reasoning about which computations are relevant

## Non-Zero-Sum Games

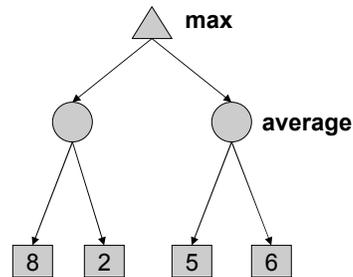
---

- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children



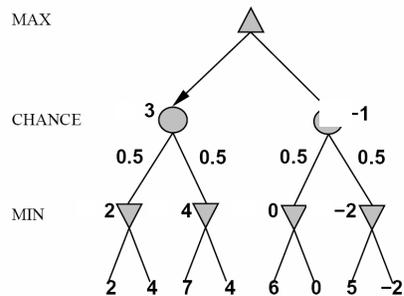
# Stochastic Single-Player

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, don't know where the mines are
- Can do **expectimax search**
  - Chance nodes, like actions except the environment controls the action chosen
  - Calculate utility for each node
  - Max nodes as in search
  - Chance nodes take average (expectation) of value of children
- Later, we'll learn how to formalize this as a **Markov Decision Process**



# Stochastic Two-Player

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

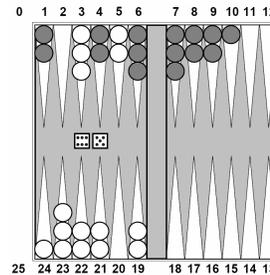


if *state* is a MAX node then  
 return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)  
 if *state* is a MIN node then  
 return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)  
 if *state* is a chance node then  
 return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)

## Stochastic Two-Player

---

- Dice rolls increase  $b$ : 21 possible rolls with 2 dice
  - Backgammon  $\approx 20$  legal moves
  - Depth 4 =  $20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$
- 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



## What's Next?

---

- Make sure you know what:
  - Probabilities are
  - Expectations are
- Next topics:
  - Dealing with uncertainty
  - How to learn evaluation functions
  - Markov Decision Processes