Agents that Search Together: Stochastic Search

Russell and Norvig: Chapter 5

CSE 240: Winter 2023

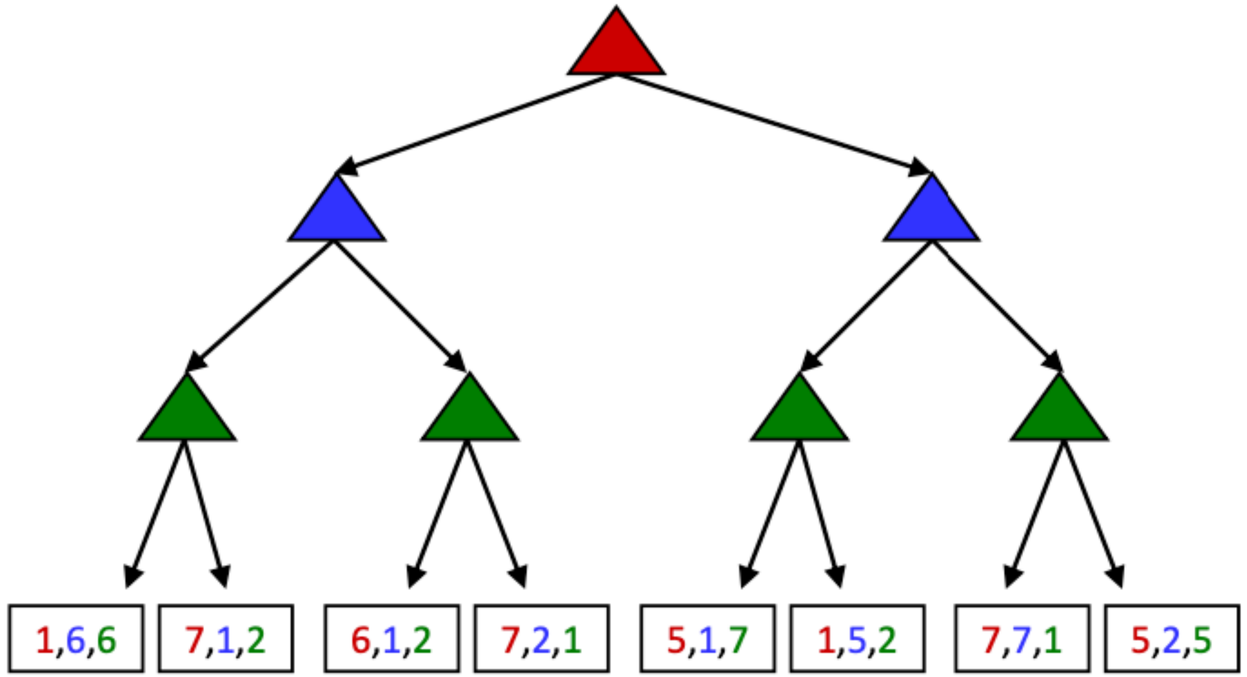
Lecture 7

Agenda and Topics

- Game agents in stochastic environments
  - Expectimax algorithm
- Local search and optimization algorithms

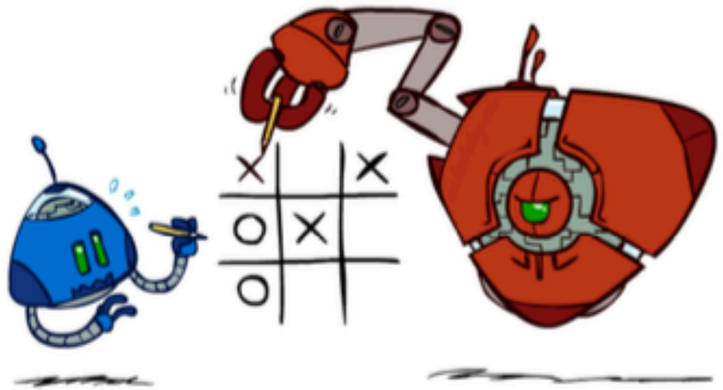
Multi-Agent Utilities

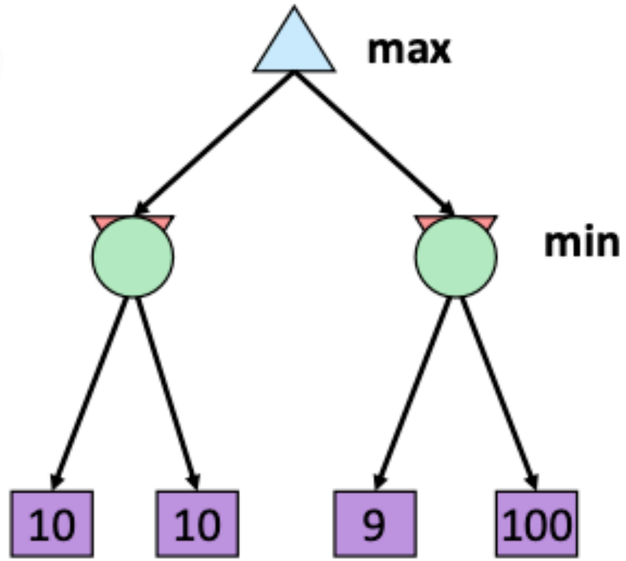
- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and competition dynamically...



Uncertain Outcomes

Worst Case Versus Average Case





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

Evaluation function

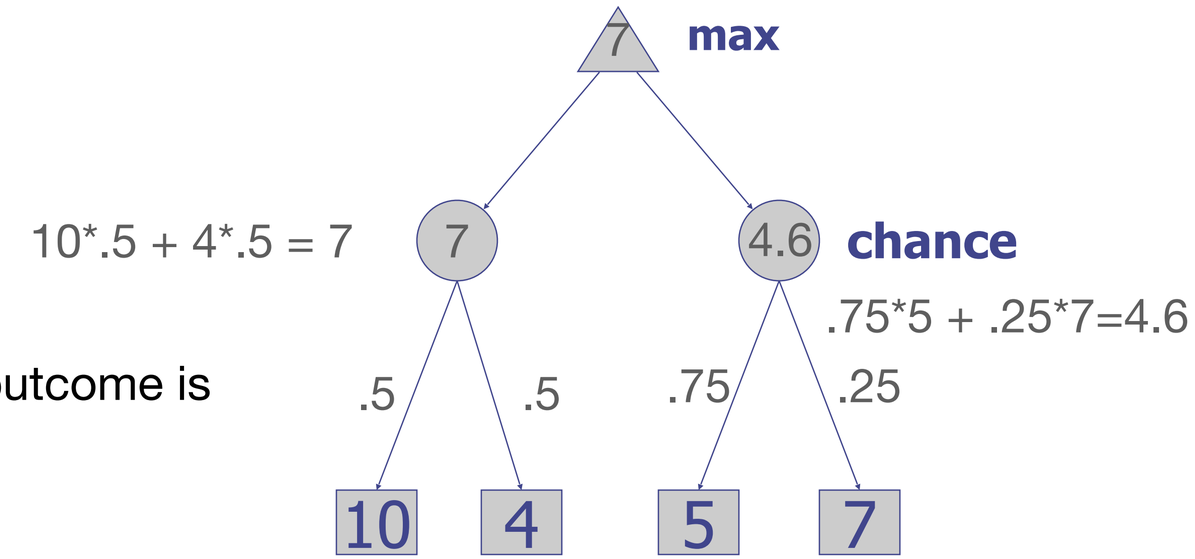
- **Evaluation function** or **static evaluator** is used to evaluate the "goodness" of a game position.
  - Contrast with heuristic search where the evaluation function was a non-negative estimate of the cost from the start node to a goal and passing through the given node
- The zero-sum assumption allows us to use a single evaluation function to describe the goodness of a board with respect to both players.
  - **f(n) >> 0**: position n good for me and bad for you
  - **f(n) << 0**: position n bad for me and good for you
  - **f(n) near 0**: position n is a neutral position
  - **f(n) = +infinity**: win for me
  - **f(n) = -infinity**: win for you

Evaluation function examples

- Example of an evaluation function for Tic-Tac-Toe:
  - f(n) = [# of 3-lengths open for me] - [# of 3-lengths open for you]
  - where a 3-length is a complete row, column, or diagonal
- Alan Turing's function for chess
  - **f(n) = w(n)/b(n)** where w(n) = sum of the point value of white's pieces and b(n) = sum of black's
- Most evaluation functions are specified as a weighted sum of position features:
  - $f(n) = w_1*feat_1(n) + w_2*feat_2(n) + ... + w_n*feat_k(n)$
- Example features for chess are piece count, piece placement, squares controlled, etc.
- Deep Blue has about 6000 features in its evaluation function

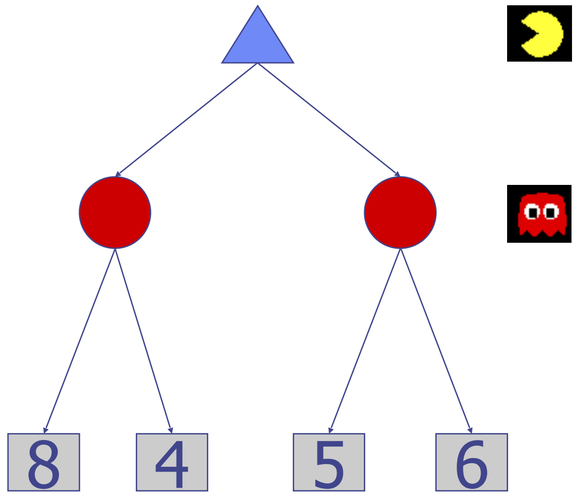
Expectimax Search Trees

- What if we don't know what the result of an action will be? E.g.,
  - In solitaire, next card is unknown
  - In minesweeper, mine locations
  - In pacman, the ghosts act randomly
- Can do **expectimax search**
  - Chance nodes, like min nodes, except the outcome is uncertain
  - Calculate expected utilities
  - Max nodes as in minimax search
  - Chance nodes take average (expectation) of value of children
- Later, we'll learn how to formalize the underlying problem as a **Markov Decision Process**



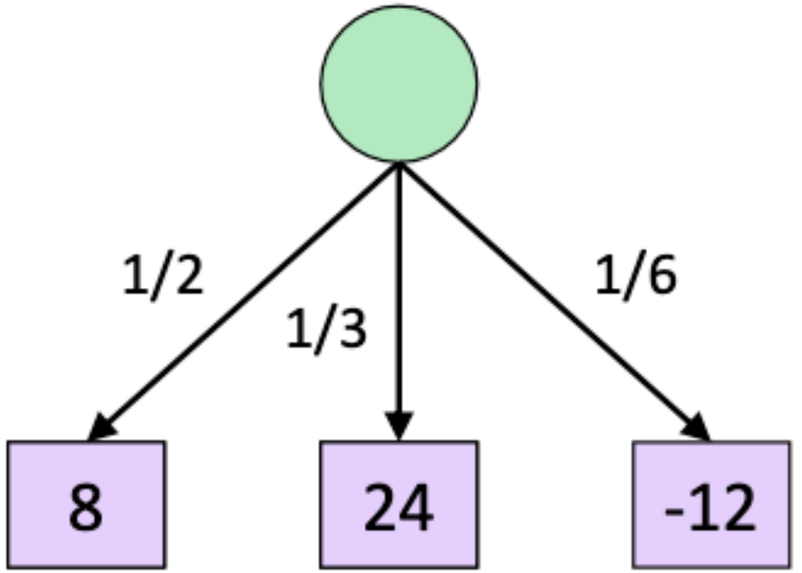
Expectimax Pseudocode

```
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)
```



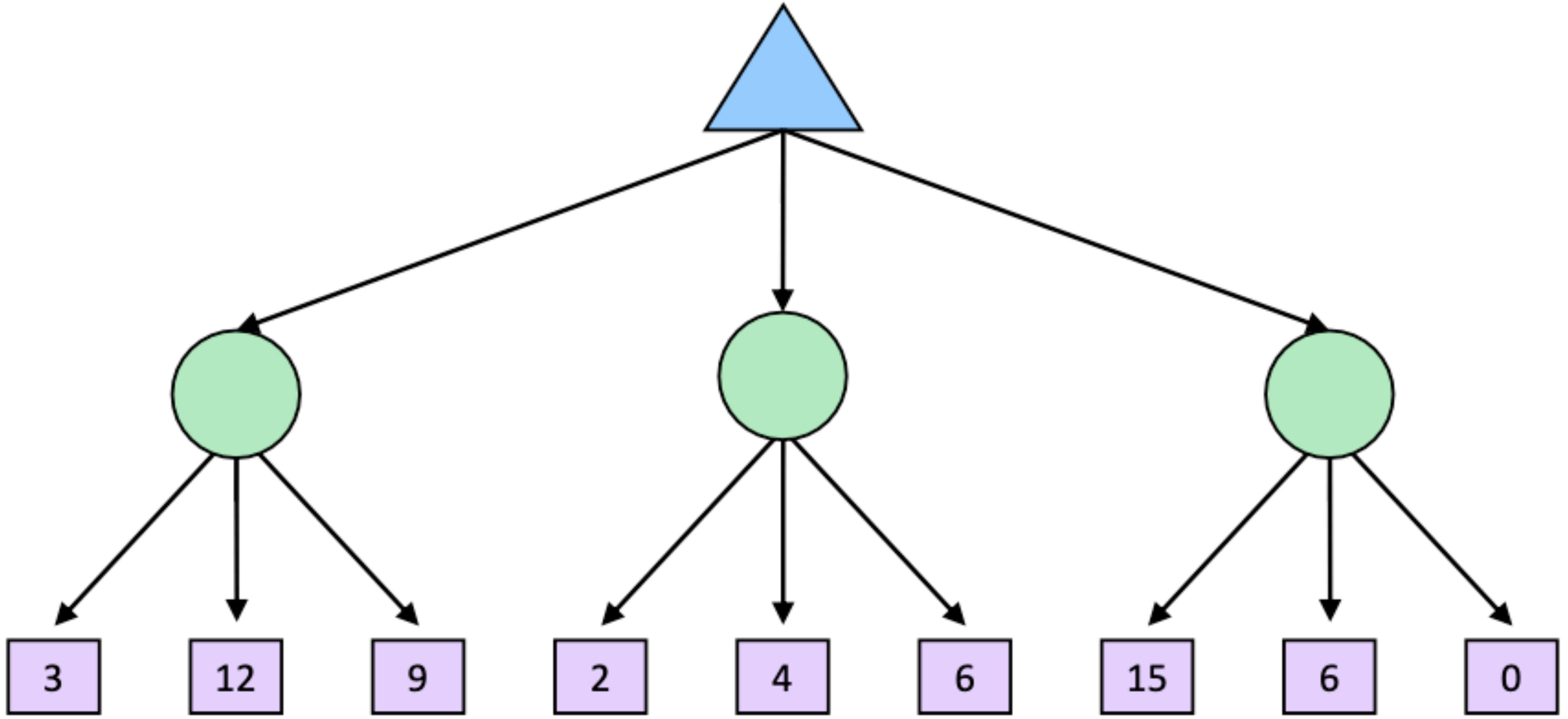
Expectimax Pseudocode

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

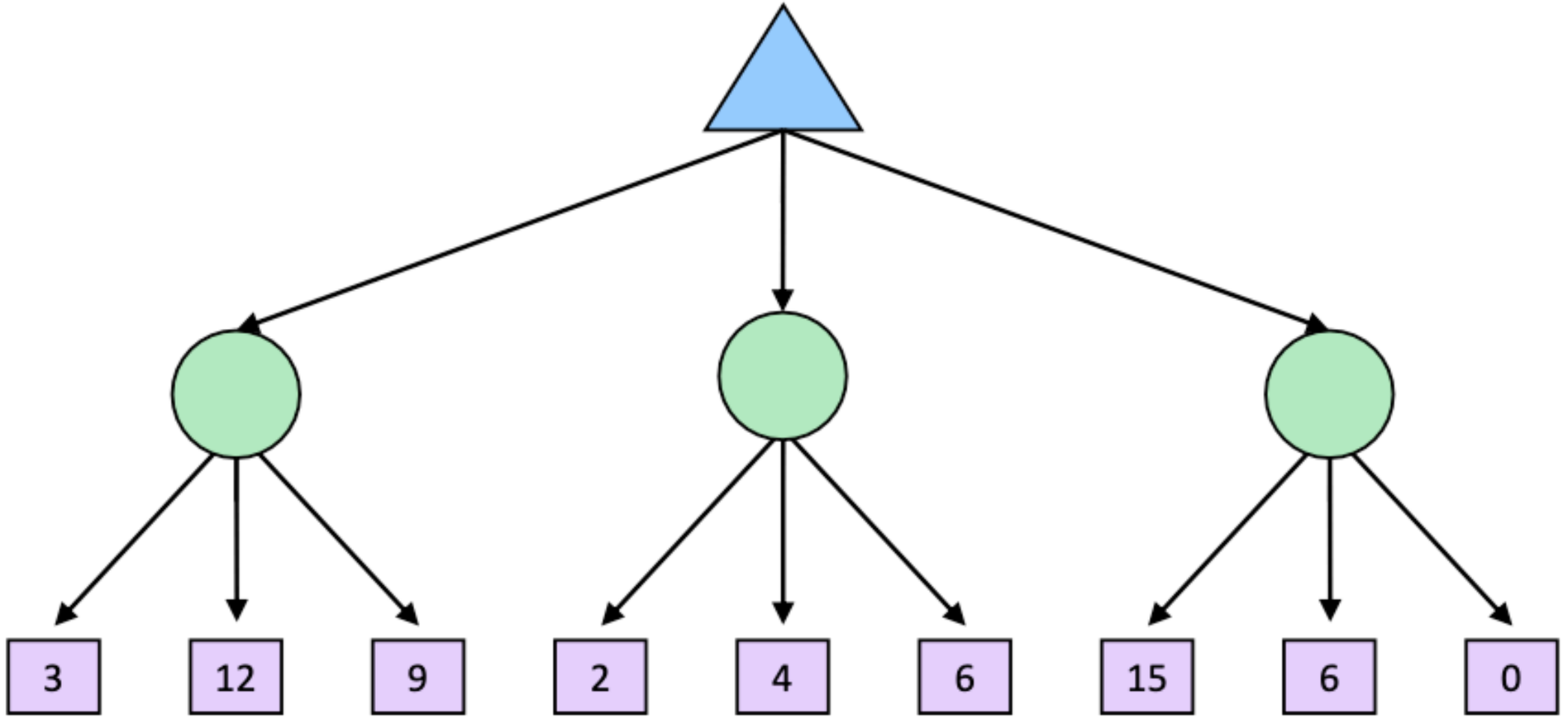


$$v = (1/2)\ (8) + (1/3)\ (24) + (1/6)\ (-12) = 10$$

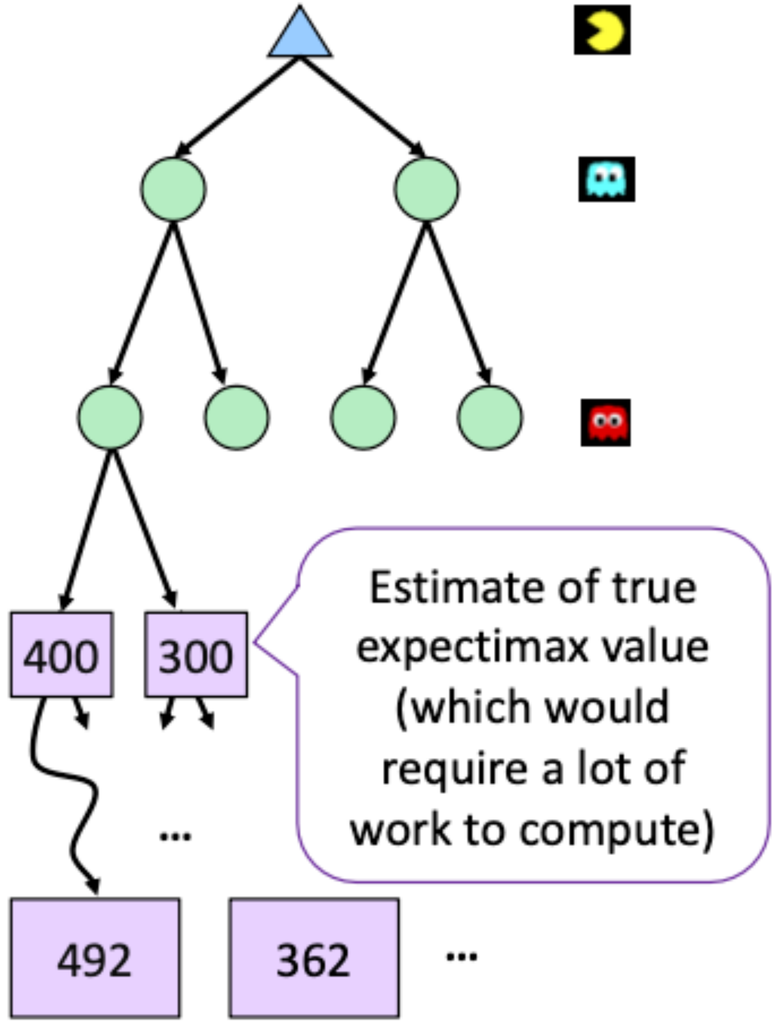
Expectimax Example



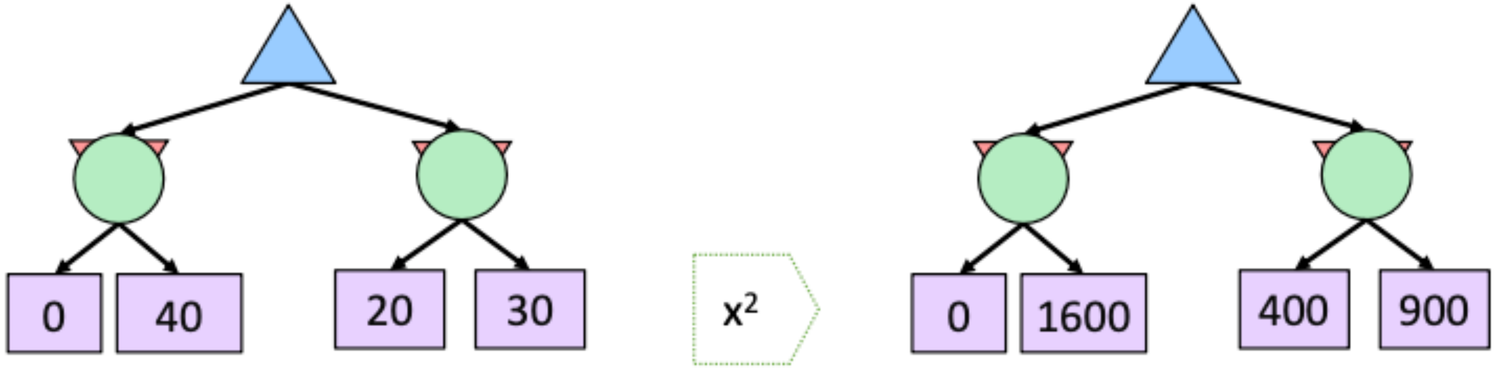
Expectimax Pruning?



Depth-Limited Expectimax



What Utilities to Use?



- For worst-case minimax reasoning, terminal function scale doesn't matter
- We just want better states to have higher evaluations (get the ordering right)
- We call this insensitivity to monotonic transformations
- For average-case expectimax reasoning, we need *magnitudes* to be meaningful.

Modeling Assumptions

The Dangers of Optimism and Pessimism

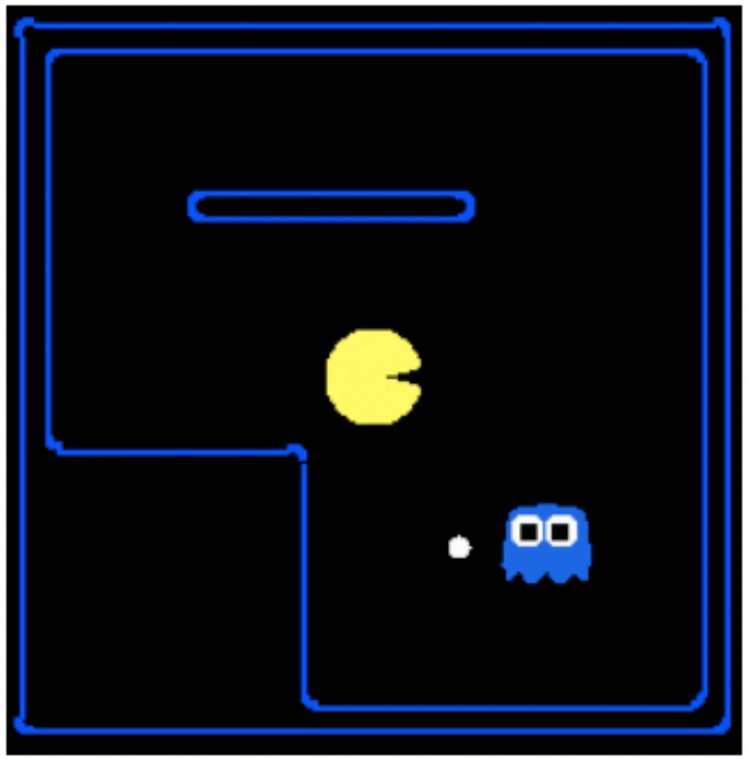
Dangerous Optimism

Assuming chance when the world is adversarial

Dangerous Pessimism

Assuming the worst case when it's not likely

Assumption versus Reality



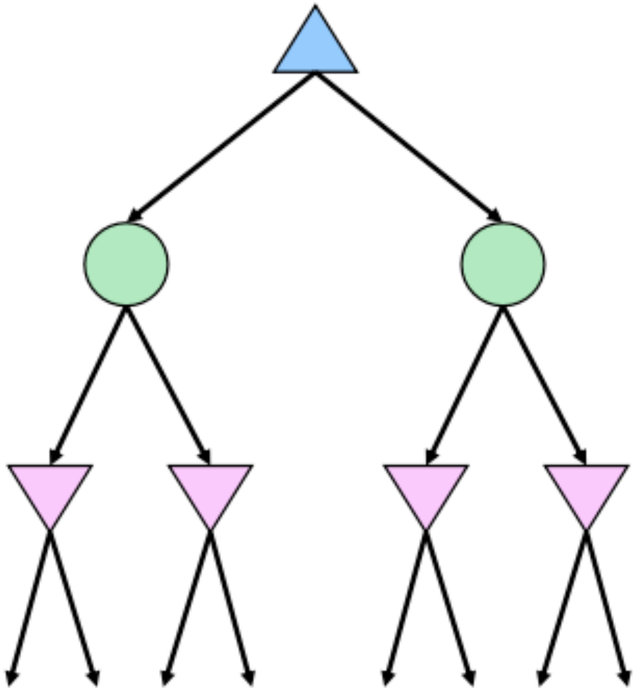
	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

Mixed Layer Types

- E.g. Backgammon
- Expecti-minimax
  - Environment is an extra “random agent” player that moves after each min/max agent
  - Each node computes the appropriate combination of its children



CE 7: Feedback

- https://forms.gle/ghXVUmjMJjpdcMH6A

Local Search Algorithms

Local Search Algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
  - Local search: widely used for very big problems
  - Returns good but not optimal solutions
- Local search algorithms
  - Keep a single "current" state, or small set of states
  - Iteratively try to improve it / them
  - Very memory efficient
    - Keeps only one or a few states
    - You control how much memory you use

Example Problems for Local and Systematic Search

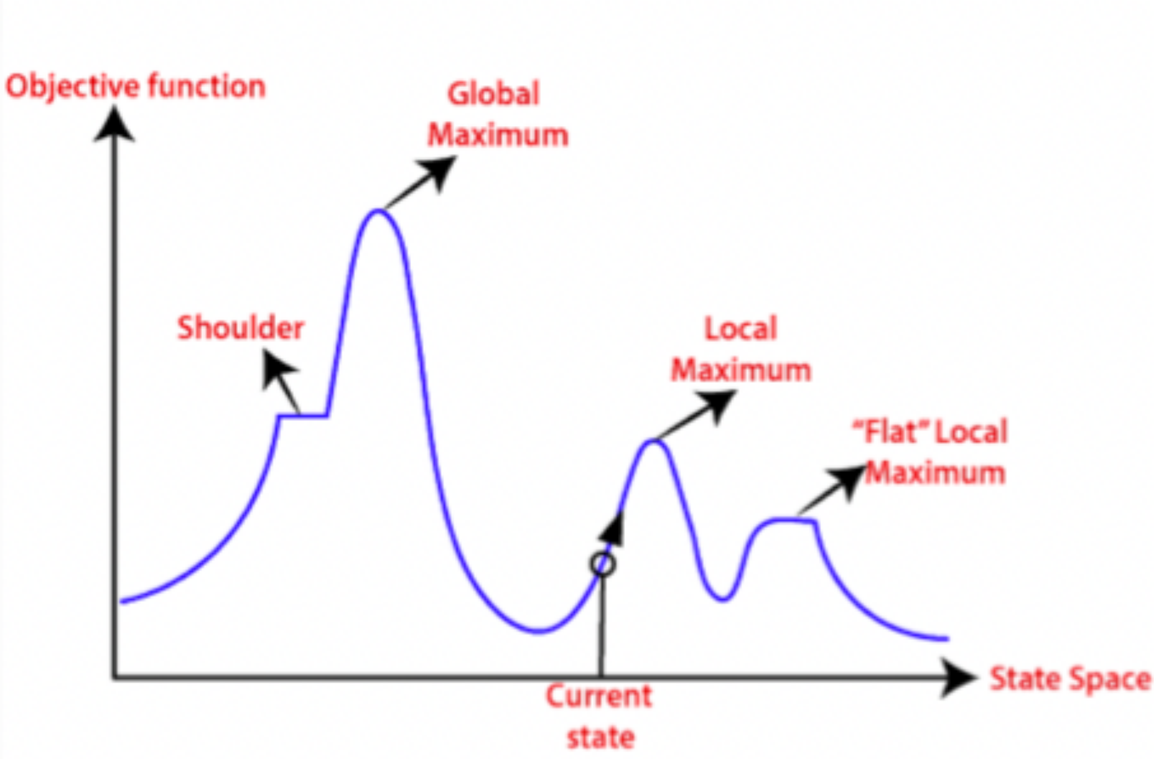
- Systematic search: path to goal is important
  - Theorem solving
  - Route finding
  - 8-puzzle
  - Chess
- Local search: goal state itself is important
  - 8-Queen
  - Traveling Salesman Problem
  - Job Scheduling

State-space Landscape

- Local search algorithms explore the landscape

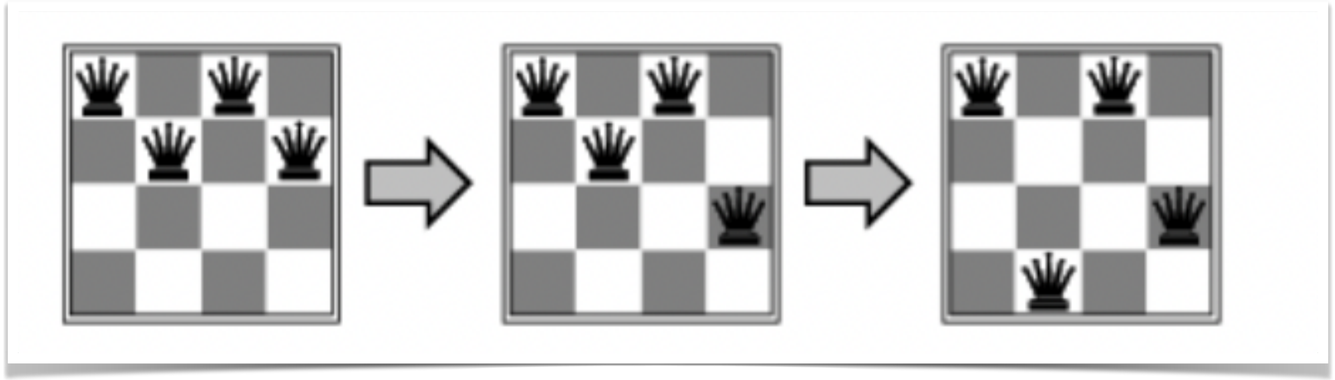
- A landscape has both
  - "location" (defined by the state)
  - "elevation" (defined by value of the heuristic cost function or objective function)

- Solution: a state with the optimal value of the objective function
  - Note that a minimization problem can turn into a maximization problem by adding a minus sign



A one-dimensional state-space landscape in which elevation corresponds to the objective function

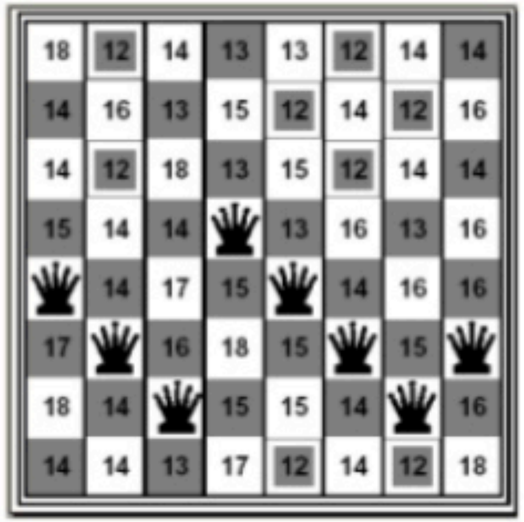
Example: n-queen



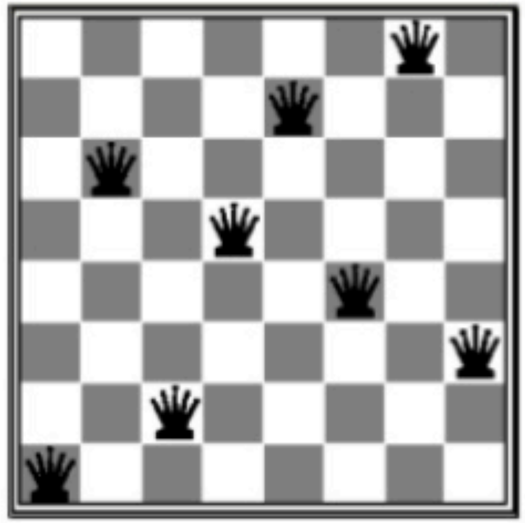
- Put n queens on an nxn board with no two queens on the same row, column or diagonal.
- What is the state space?
- What is the objective function?

Local Search: 8-queen Problem

- States: 8 queens on the board, one per column ($8^8 \approx 17M$)
- Successors(s): all states resulted from state s by moving a single queen to another square of the same column ($8 \times 7 = 56$ successors per state)
- Cost function $h(s)$: number of queen pairs that are attacking each other (directly or indirectly)
- Global minimum: $h(s) = 0$



best moves reduce $h = 17$ to $h = 12$



local minimum with $h = 1$

Hill Climbing Search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

- Search strategy: steepest ascent among immediate neighbors until reaching a peak.
- “...like trying to find the top of Mount Everest in a thick fog while suffering from amnesia”

Example: Hill Climbing, 8 queens

h = # of pairs of queens that are attacking each other, either directly or indirectly

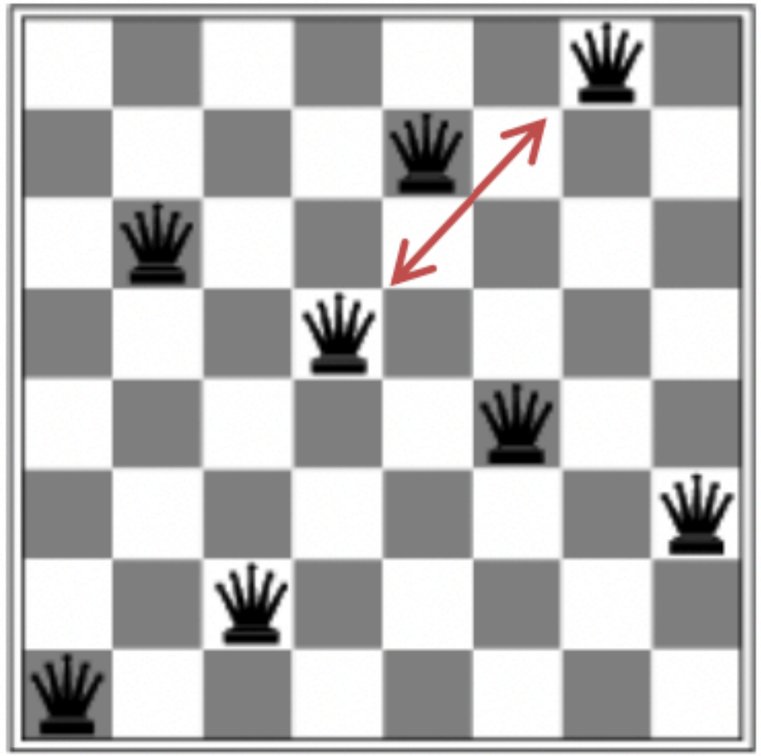
$h = 17$ for this state

Each number indicates h if we move a queen in its column to that square

12 (boxed) = best h among all neighbors; select one randomly

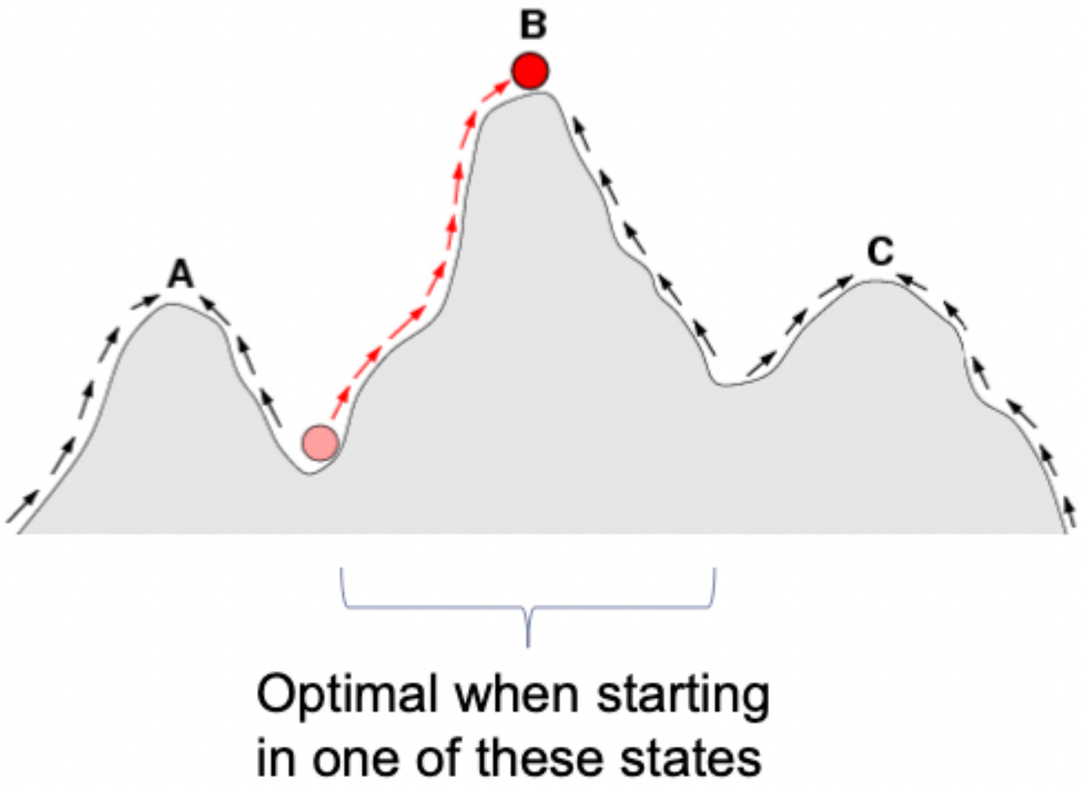
Example: Hill Climbing, 8 queens

- A local minimum with $h = 1$
- All one-step neighbors have higher h values
- What can you do to get out of this local minimum?



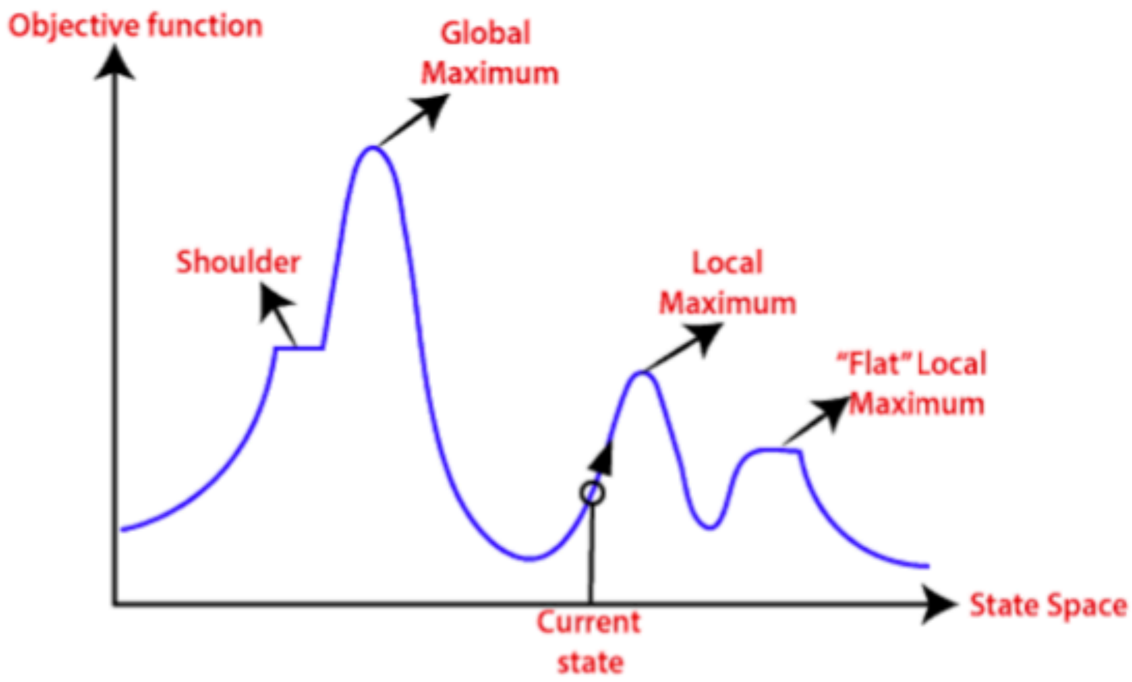
Hill Climbing Strategy is Greedy

- Greedy local search: considering only one step ahead and select the best successor state (steepest ascent).
- Can provide rapid progress towards a solution
  - Usually quite easy to improve a bad initial solution.



Hill Climbing Difficulties

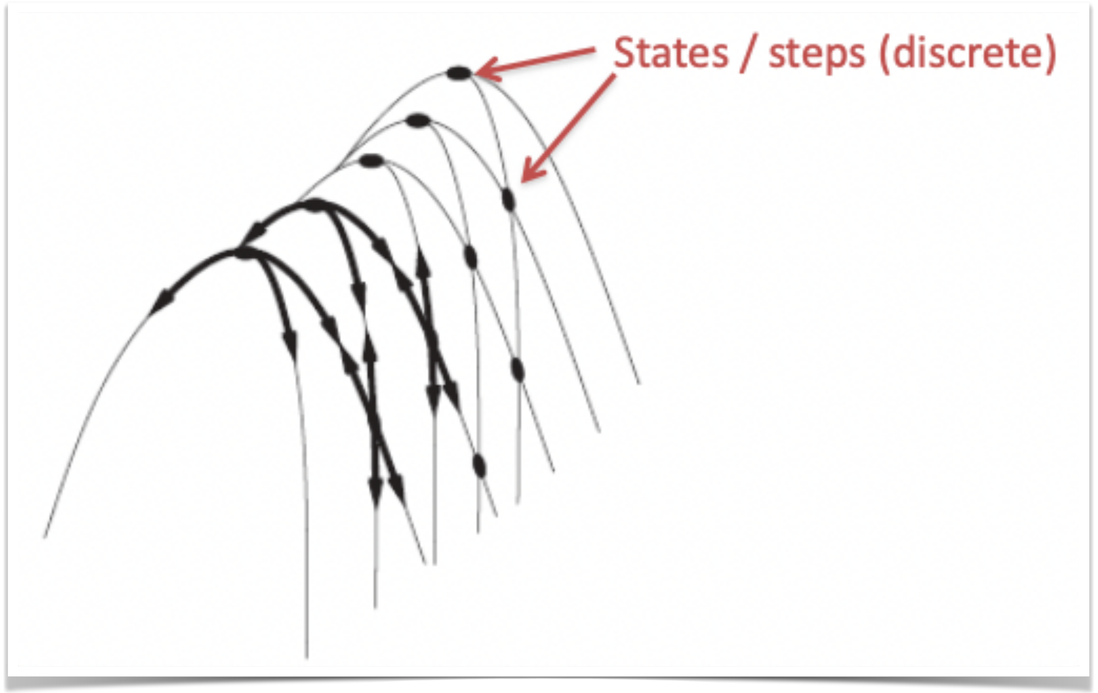
- Depending on initial state, algorithm can get stuck in local optimum
- Local maxima: a peak that's not a global maximum
- Plateau: a flat area (shoulders or flat local maximum).



A one-dimensional state-space landscape in which elevation corresponds to the objective function

Hill Climbing Difficulties (Cont.)

- Ridge problem: every neighbor appears to be downhill
  - But, search space has an uphill (just not in neighbors(
- Ridge:
  - Fold a piece of paper and hold it tilted up at an unfavorable angle to every possible search space step.
  - Every step leads downhill; but the ridge leads uphill



Hill Climbing Properties

- Completeness: Not complete
- Time complexity: Worst-case exponential time
- Space complexity: Simple O(1) space complexity

Summary

- Game agents in stochastic environments
  - Expectimax algorithm
- Local search and optimization algorithms

Next time:

- Continue: Local search and optimization algorithms