Search

Dr. Melanie Martin
CS 4480
September 21, 2012

Beyond Classical Search

- Chapter 4
  - Hill Climbing
  - Simulated Annealing
  - Beam Search
  - Genetic Algorithms

Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
  - Find configuration satisfying constraints, e.g., n-queens
- In such cases, we can use local search algorithms
  - keep a single "current" state, try to improve it

Local search algorithms

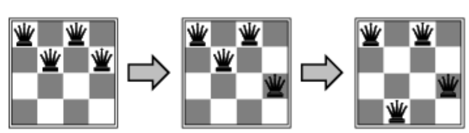
- Does path matter?
  - Chess
  - Robot
  - 8 queens
  - Circuit design
  - Job scheduling
- Optimization Problems
  - Goal is best state according to some "objective" function
    - No goal test or path cost
  - "Reproductive fitness" in nature
  - Local search may work well

Local search algorithms

- Idea: in current state
  - Expand
  - Move to a neighbor
- Pros:
  - Usually Constant Memory
  - Can often find reasonable solution in infinite or continuous state spaces

Example: *n*-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal



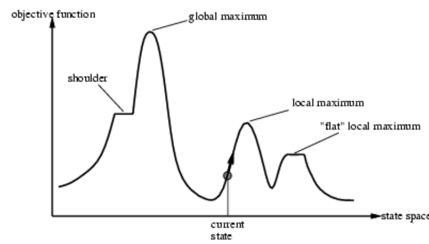
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

```
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
```

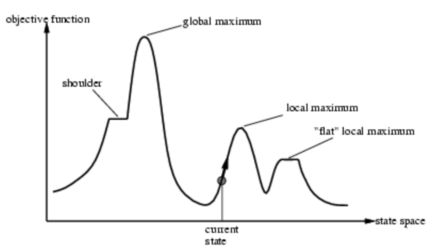
Hill-Climbing Search

- Problem: depending on initial state, can get stuck in local maxima



Hill-Climbing Search

- If using cost function will want global minimum



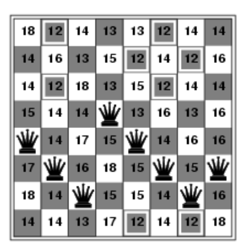
Hill-Climbing Search

- What to do when stuck?
- Stochastic Hill-Climbing
  - Choose successor at random
  - Probability based on steepness
- First Choice Hill-Climbing
  - Generate random successors until one is better

Hill-Climbing Search

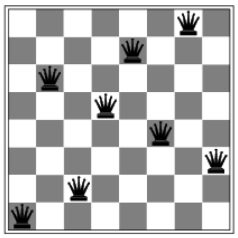
- What to do when stuck?
- Random-Restart Hill-Climbing
  - Series of hill-climbing searches from randomly generated initial states
  - Complete
  - Random sideways moves escape from shoulders
  - But loop on flat maxima

Hill-climbing search: 8-queens problem



- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

Hill-climbing search: 8-queens problem



- A local minimum with $h = 1$

Simulated Annealing Search

- Anneal from

http://www.merriam-webster.com/dictionary/anneal

to heat and then cool (as steel or glass) usually for softening and making less brittle; *also* **:** to cool slowly usually in a furnace

Simulated Annealing Search

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
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
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```

Properties of simulated annealing search

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Widely used in VLSI layout, airline scheduling, etc
  - VLSI: very large scale integration for creating integrated circuits

Local beam search

- Keep track of k states rather than just one
- Start with k randomly generated states
- At each iteration, all the successors of all k states are generated
  - If any one is a goal state, stop; else select the k best successors from the complete list and repeat.
- Not the same as k searches in parallel
- Problem: all k states may end up on same local hill
  - Choose the k successors randomly, biased toward good ones

Genetic algorithms

- A successor state is generated by combining two parent states
- Start with k randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation

Genetic algorithms

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms