CS W4701
Artificial Intelligence

Fall 2013

Chapter 4:

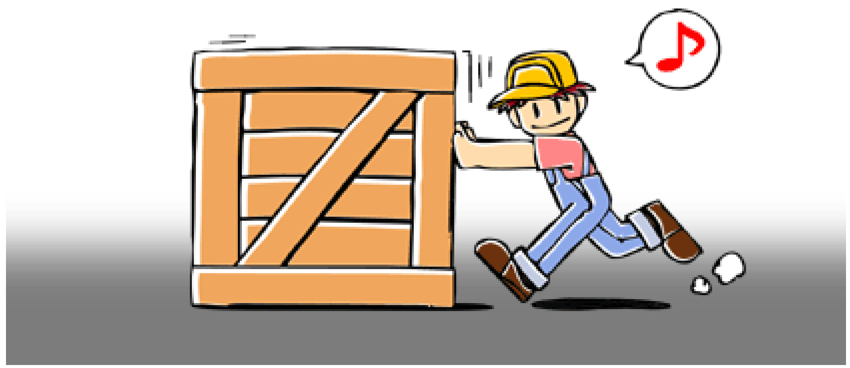
Beyond Classical Search

Jonathan Voris

(based on slides by Sal Stolfo)

Assignment 2

- Develop a Sokoban puzzle solving agent!



- Sokoban is Japanese for “warehouse keeper”
- Created by Hiroyuki Imabayashi in 1981
- First released as a game for Japanese home computers in 1982

Assignment 2

- Player who can move in cardinal directions
- Warehouse full of boxes and storage locations
- Boxes can be pushed by moving into them
- Goal: Push all boxes into a storage location

Assignment 2

- Input: Sokoban puzzle in ASCII
  - #	(hash)	Wall
  - .	(period)	Empty goal
  - @	(at)	Player on floor
  - +	(plus)	Player on goal
  - $	(dollar)	Box on floor
  - *	(asterisk)	Box on goal

Assignment 2

- Output: Sequence of moves to solve puzzle in CSV format
  - u up
  - d down
  - l left
  - r right

Assignment 2

- Example level

```
######
#    #
# #@ #
# $* #
# .* #
#    #
######
```

- Example solution
  - r, d, d, l, r, u, u, l, d, u, u, l, l, d, d, r

Assignment 2

- Your mission is to develop a Sokoban puzzle solving agent which utilizes a variety of search algorithms
  – BFS
  – DFS
  – UCS
  – Greedy Best-first search
  – A*
- Compare their performance!

Assignment 2

- Due in 2.5 weeks
  - Tuesday October 22nd @ 11:59:59 PM EDT
- Please follow submission instructions
  - https://www.cs.columbia.edu/~jvoris/AI/notes/Assignment%20submission%20guideline-Spring11.pdf
- Submit:
  - Code
  - Test Input/Output File
  - Readme Documentation File
- Submissions should run on CLIC machines

Sokoban Resources

- Animated game example
  - http://en.wikipedia.org/wiki/Sokoban
- Sokoban wiki
  - http://www.sokobano.de/wiki/index.php?title=Main_Page
- Sokoban puzzles
  - http://sneezingtiger.com/sokoban/levels.html
- Sokoban Java implementation
  - http://sourceforge.net/projects/jsokoapplet/

Outline

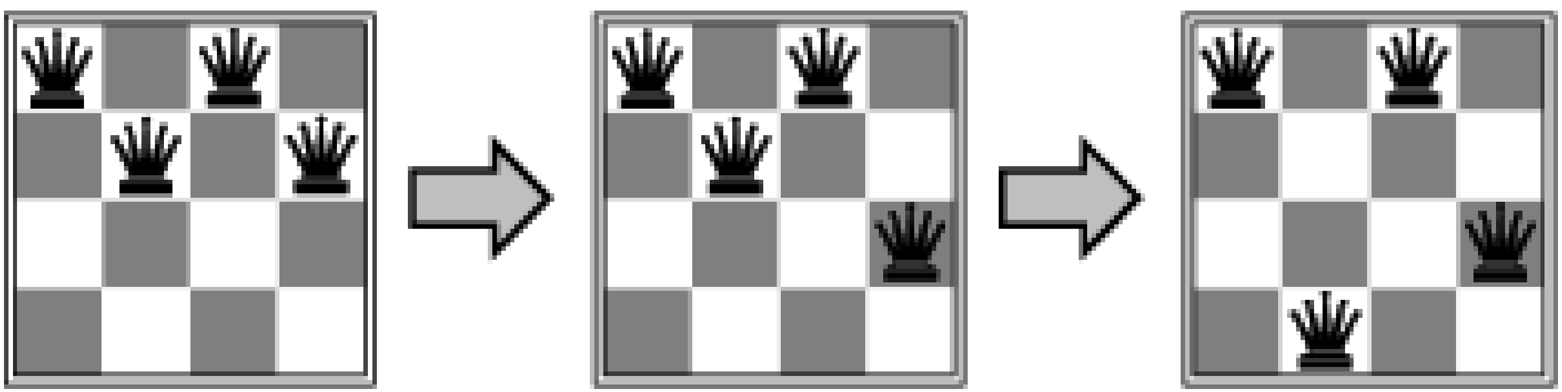
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
- Genetic algorithms

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
- Advantages:
  - Better space efficiency
  - Work with larger state spaces

Example: *n*-queens

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

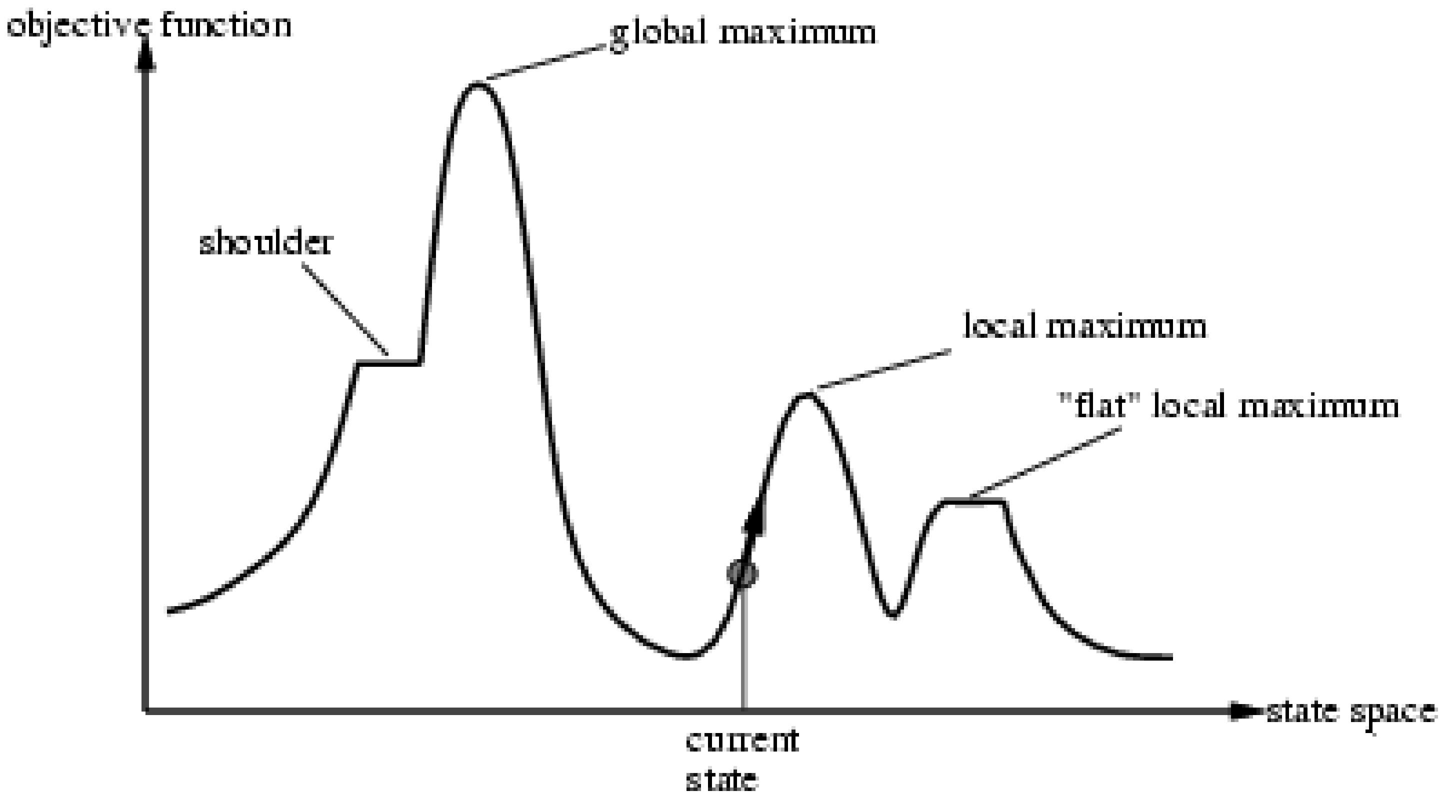


Hill-Climbing Search

- "Like climbing Everest in thick fog with amnesia"

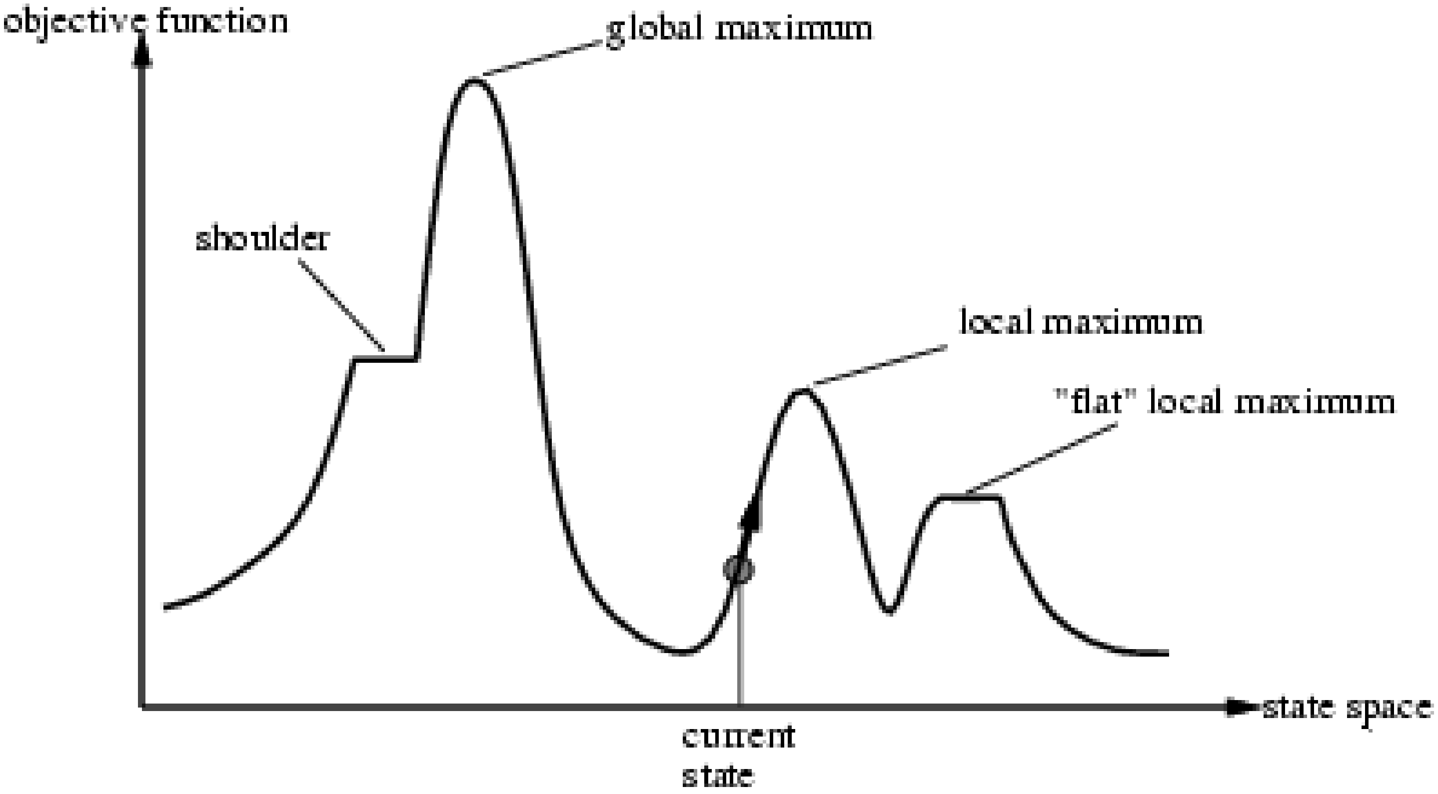
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    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



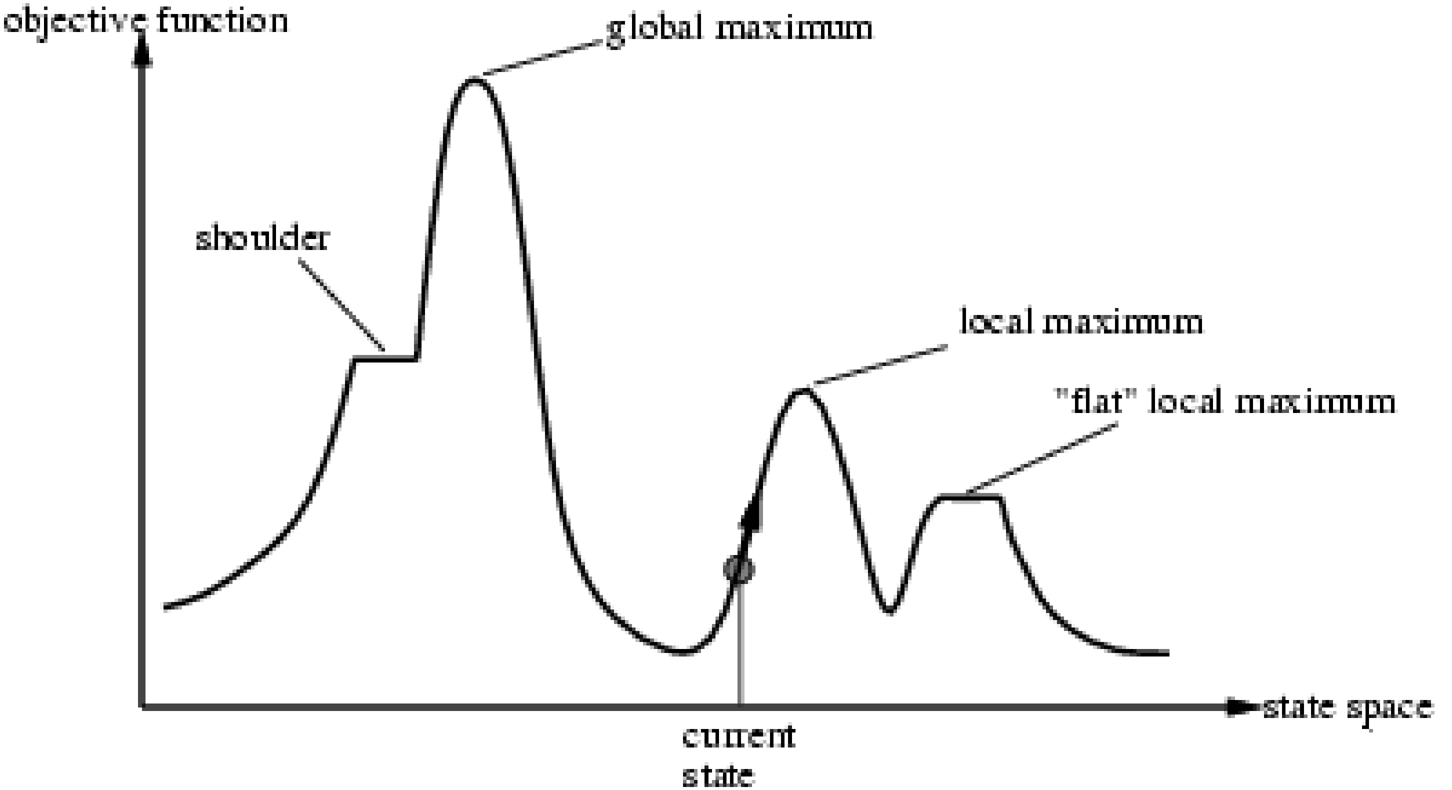
Hill-Climbing Search

- Problem: depending on initial state, can get stuck



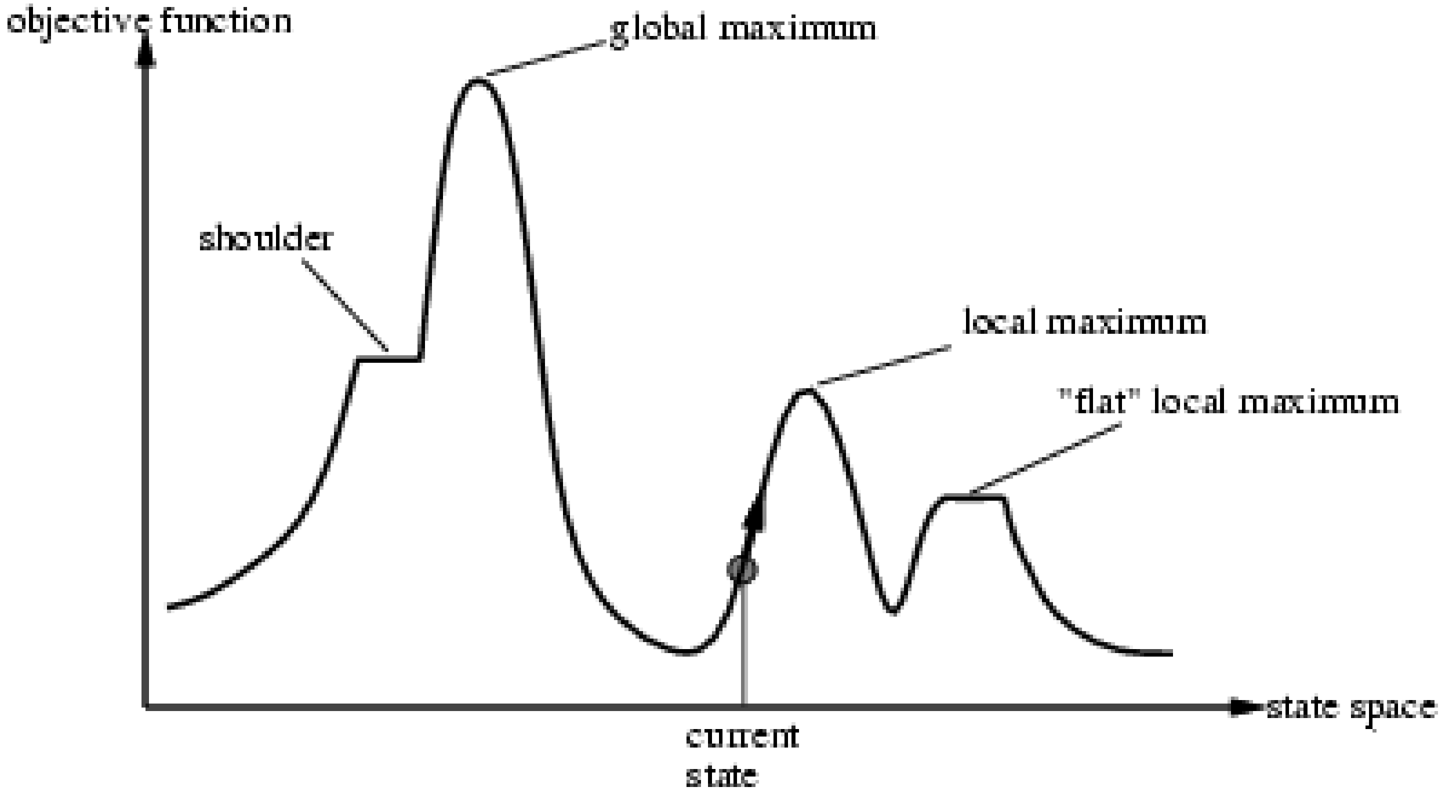
Hill-Climbing Search

- Local maxima
  - Might get stuck on a short hill



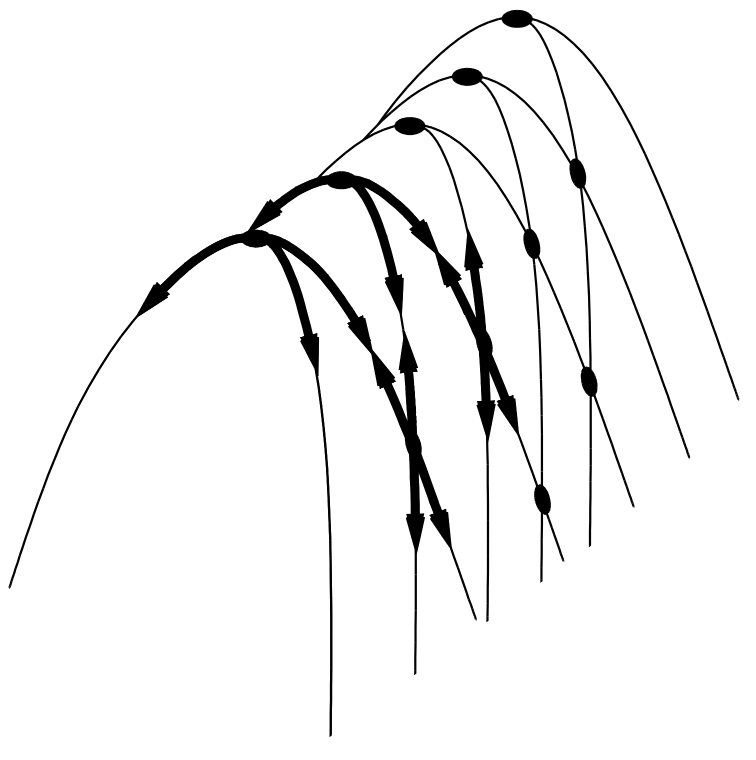
Hill-Climbing Search

- Plateaux
  - Which way when all successors are equal?

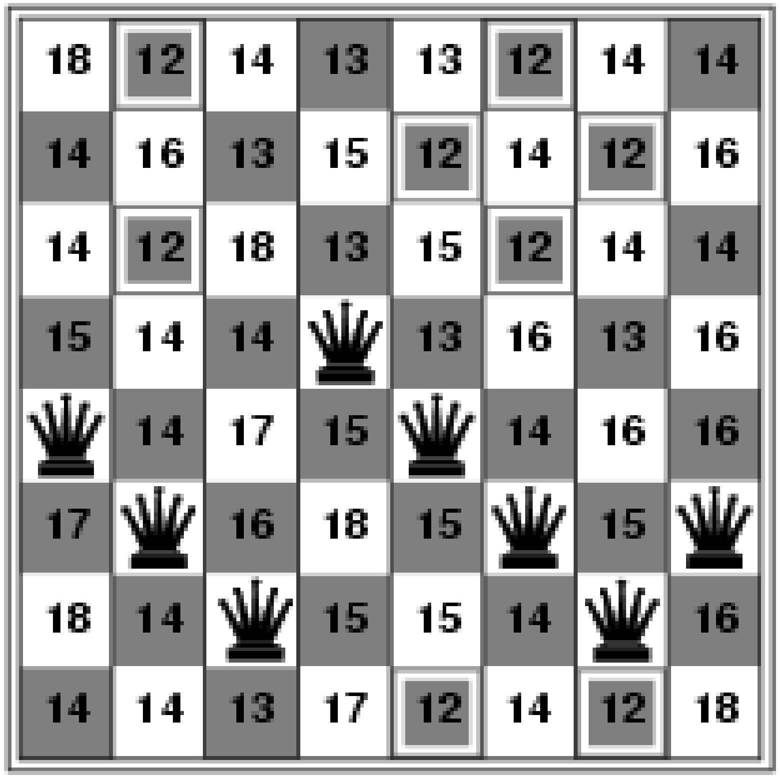


Problems with Hill-Climbing

- Ridges
  - Can’t “back up” and choose higher path

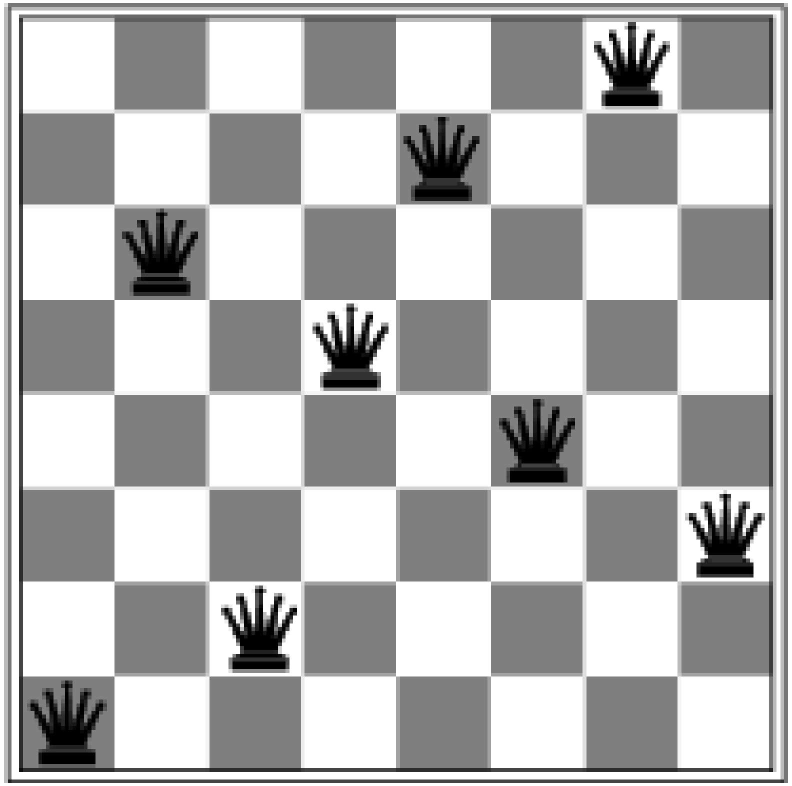


Hill-Climbing Search: 8-queens Problem



- *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 $h = 1$

Hill Climbing Tweaks

- Sideways moves?
- Stochastic hill climbing
  - Pick move with probability based on steepness
- Random restart

Random Walk

- What if you just select a random action?
- Efficiency?
- Completeness?

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^(Δ E/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.

Local Beam Search

- Keep track of *k* states rather than just one
- *k* is called the **beam width**
- 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.

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
  - Mutation

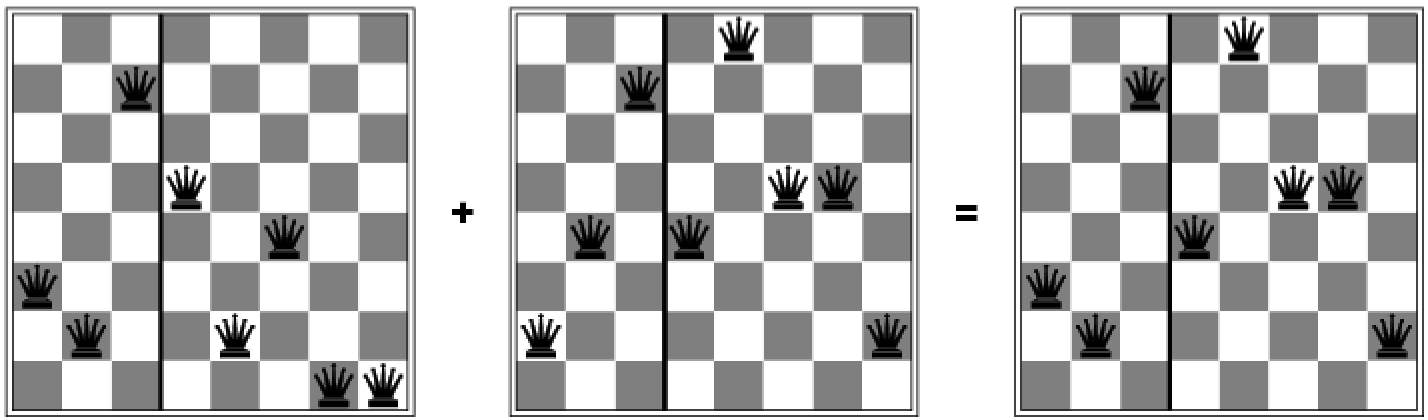
Genetic Algorithms

- What is the fitness function?
- How is an individual represented?
  – Using a string over a finite alphabet
  – Each element of the string is a *gene*
- How are individuals selected?
  – Randomly, with probability of selection proportional to fitness
  – Usually, selection is done with *replacement*
- How do individuals reproduce?
  – Through crossover and mutation

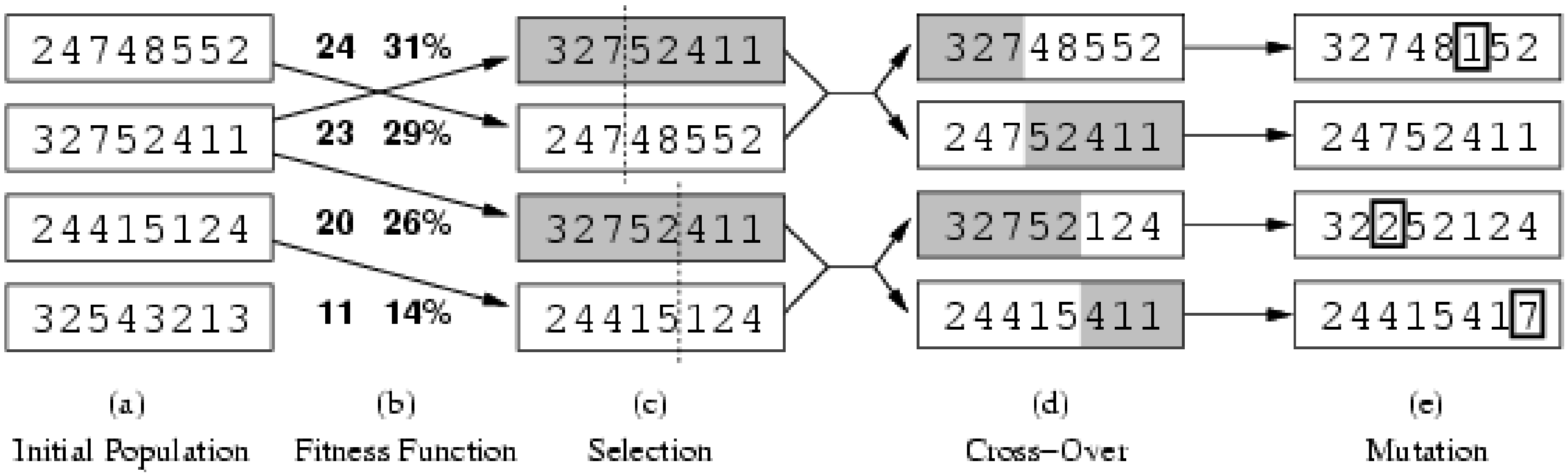
Genetic Algorithm Pseudocode

- Choose initial population (usually random)
- Repeat (until terminated)
  - Evaluate each individual's fitness
  - Select pairs to mate
  - Replenish population (next-generation)
    - Apply crossover
    - Apply mutation
  - Check for termination criteria

Genetic Algorithms



Genetic Algorithms



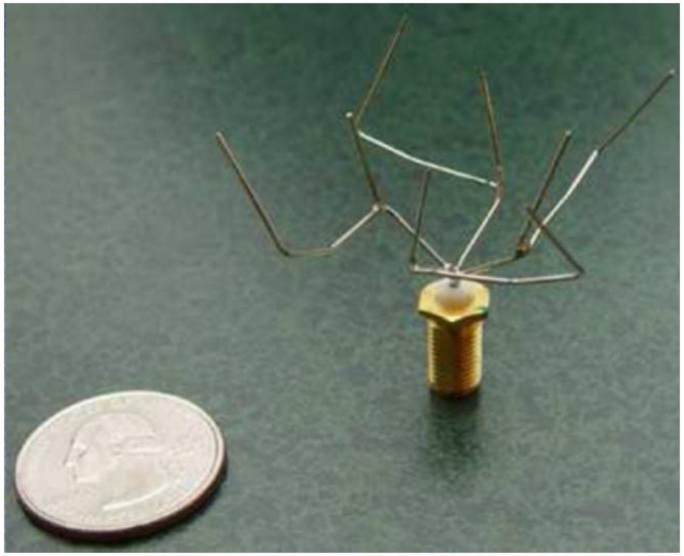
- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc.

Replacement

- Simple or generational GAs replace entire population
- Steady state or online GAs use different replacement schemes:
  - Replace worst
  - Replace best
  - Replace parent
  - Replace random
  - Replace most similar

Does This Actually Work?

- Genetic algorithms have seen success in a variety of areas
  - Data modeling
  - Signal processing
  - Economic modeling
  - Computer games
- Generally good at optimizations



Does This Actually Work?

- Genetic algorithm drawbacks
  – Expensive fitness functions
  – Scalability
  – Suboptimal solutions

Nondeterminism & Search

- What if effects of agent’s actions are unknown?
- Good idea to keep your eyes open while acting
- Vacuuum world example:
  – Sucking a dirty tile might clean one tile or multiple tile
  – Sucking a clean tile dirties it

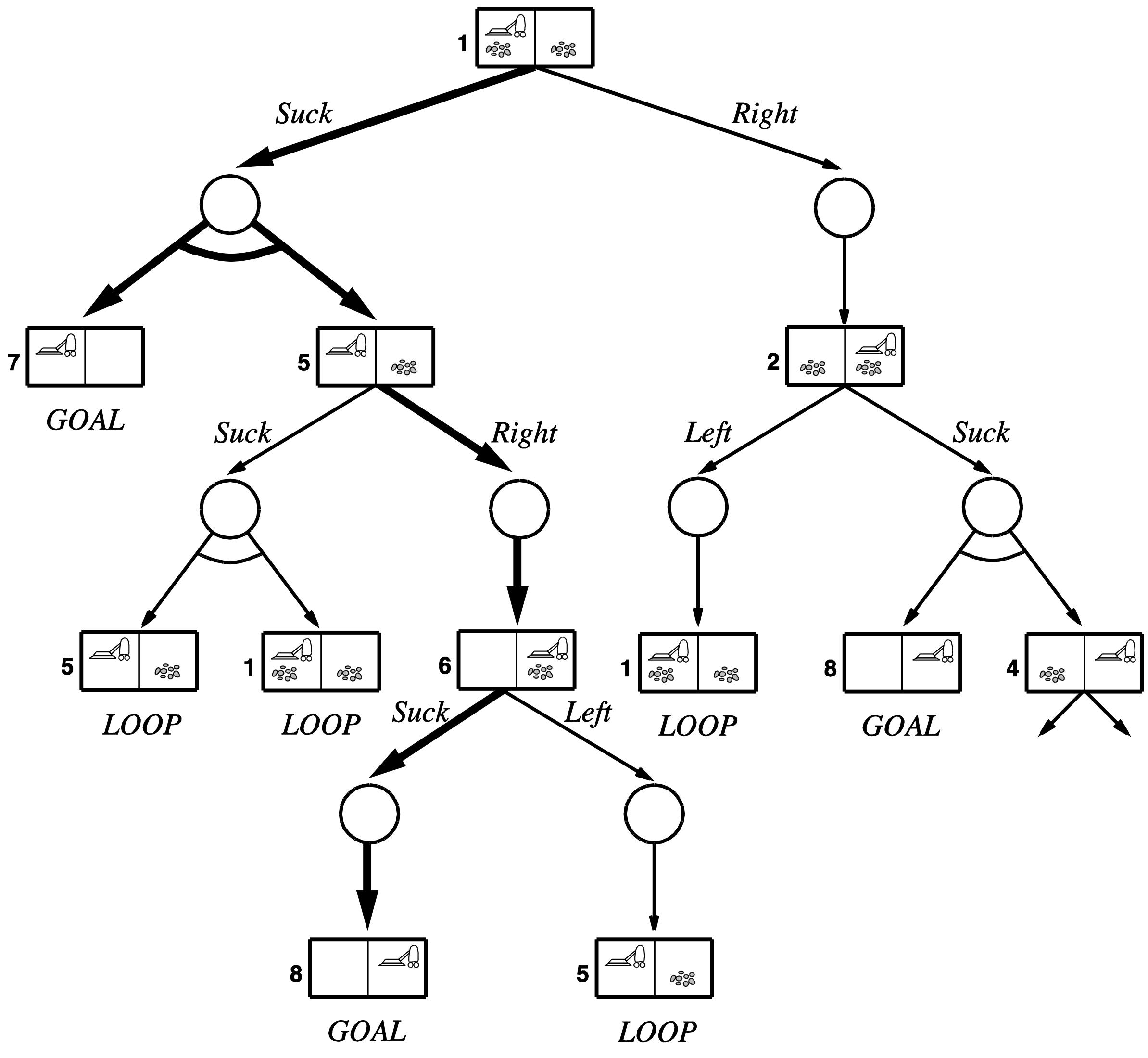
Nondeterminism & Search

- Problem changes
  - Transition yields a set of states
- Solution changes
  - Requires control flow
  - if condition(state) {action y} else {action x}
- What would be an easy way to represent this?

Nondeterminism & Search

- Agent is in control of itself
  - Knows it will perform one action **or** another
- Agent doesn't control environment
  - Needs to plan for first outcome **and** second **and** third etc
- Model this with an **and-or tree**
  - Search tree consisting of alternating layers of choices and contingencies

Nondeterministic Search Tree



Nondeterminism & Search

- Solution is an and-or subtree with
  - A goal at each leaf
  - An action at each or
  - Includes all and branches

Search Types

- Offline
  - Agent searches for solution, then acts
- Online
  - Deal with contingencies as they occur
  - Agent must *interleave* planning with action
  - We’ll see more of this shortly!