

Beyond Classical Search

Search Algorithms So Far

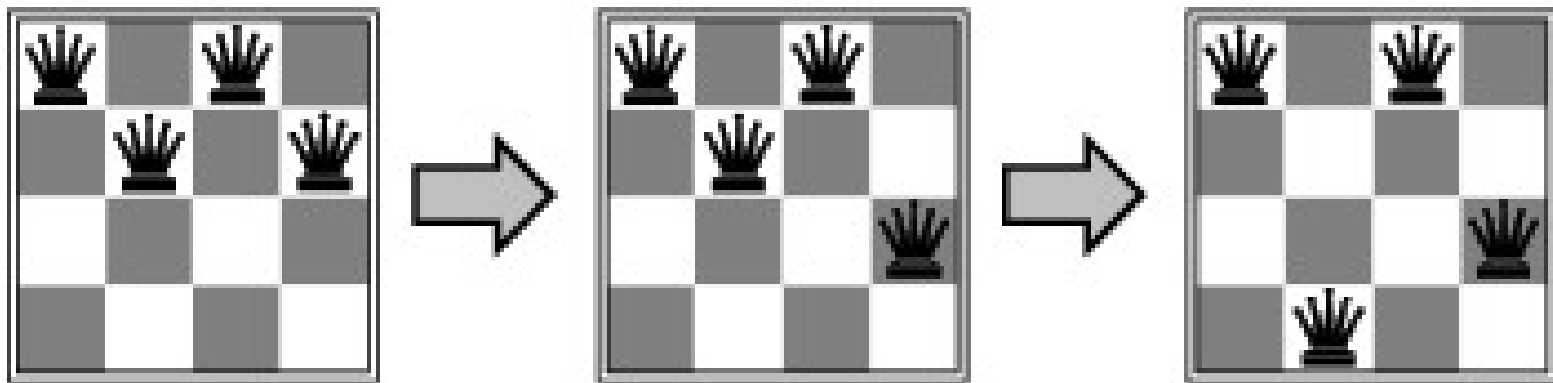
- Designed to explore search space systematically:
 - keep one or more paths in memory
 - record which have been explored and which have not
 - a path to goal represents the solution

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
 - use very little memory – usually a constant amount
 - find reasonable solutions in large or infinite state spaces for which systematic solutions are unsuitable
 - useful for solving optimization problems, e.g. Darwinian evolution, no “goal test” or “path cost”

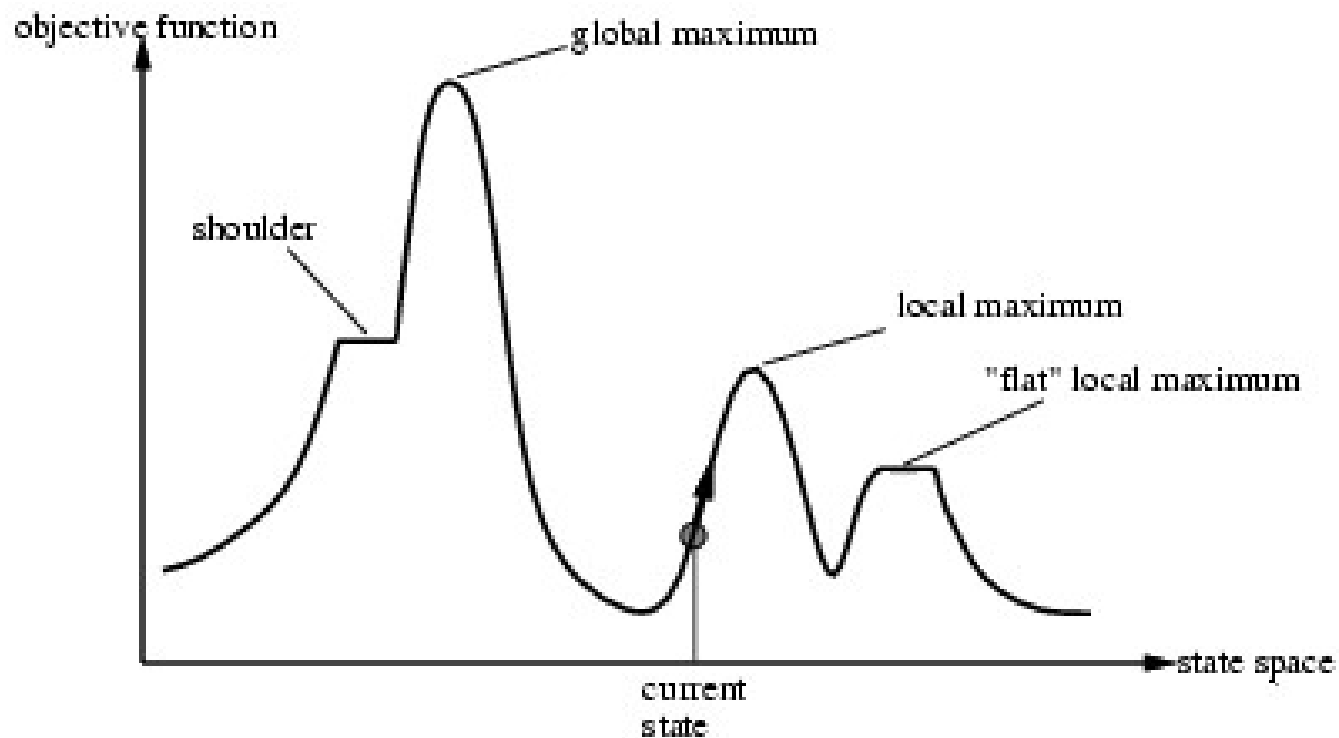
Example: n-Queen Problem

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

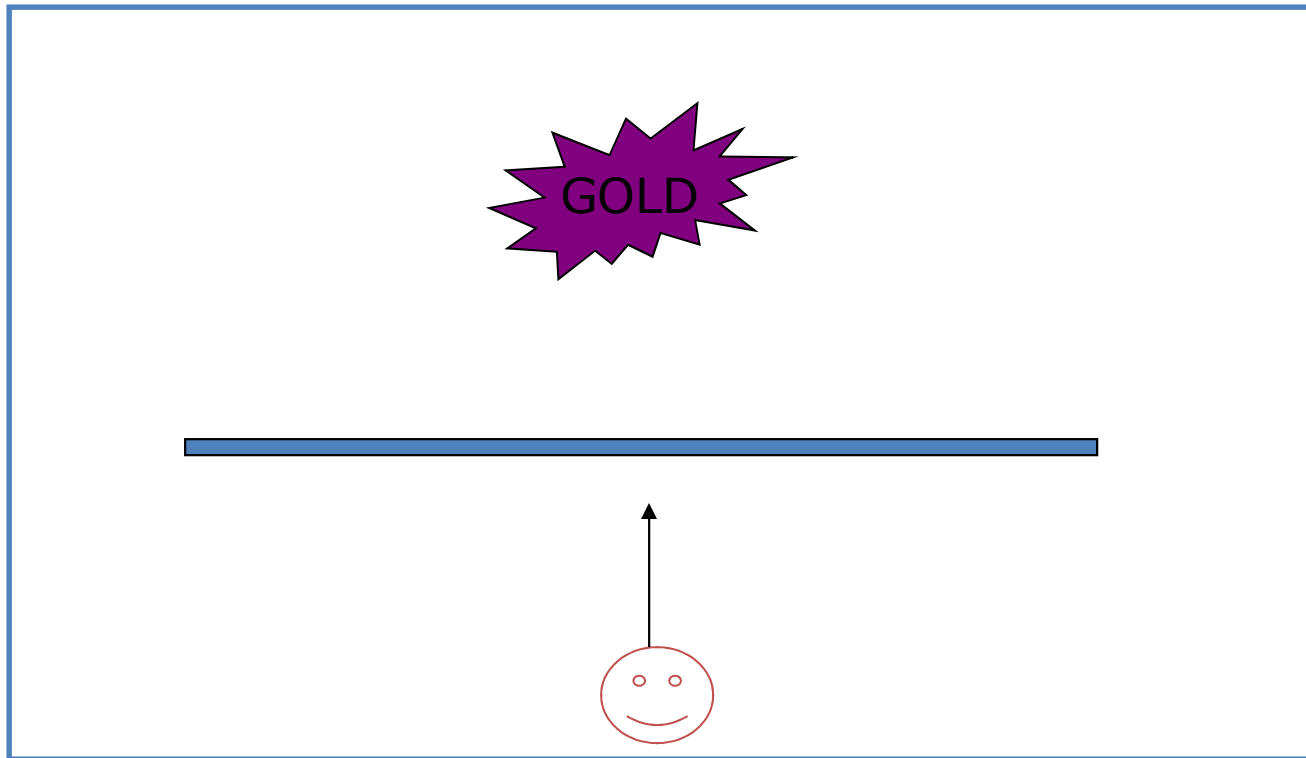


State Space Landscape

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

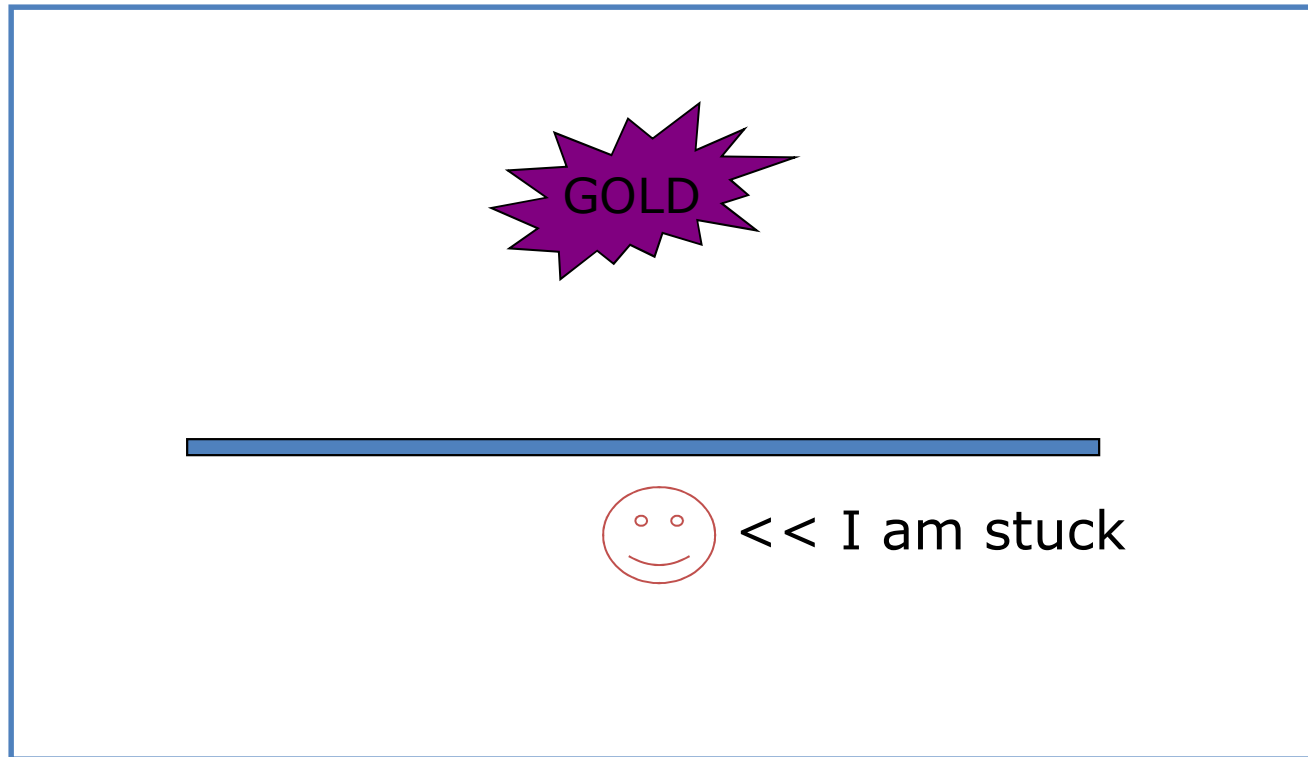


Example: Initial State



Assume the objective function measures the straight-line distance

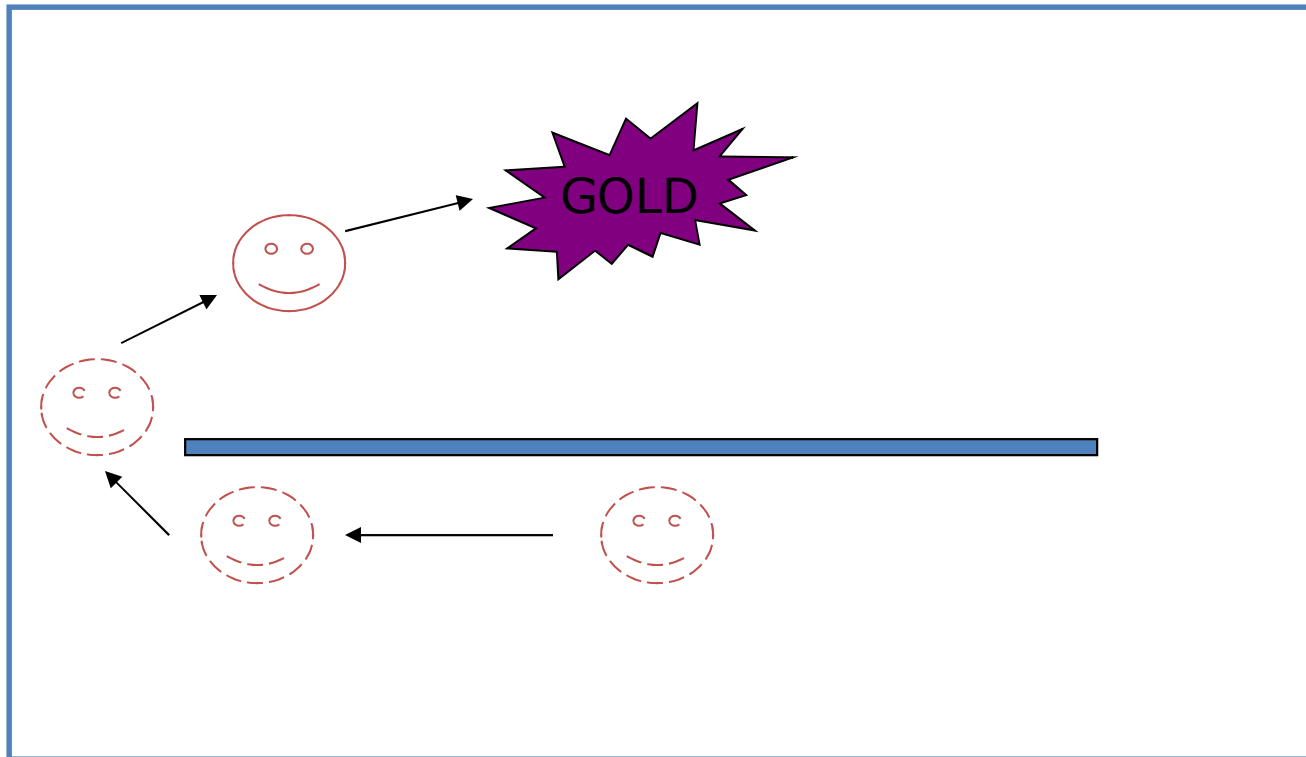
Example: Local Minima



Assume the objective function measures the straight-line distance

Example: A Plausible Solution

Making some “bad” choices is actually not that bad



Assume the objective function measure the straight-line distance

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

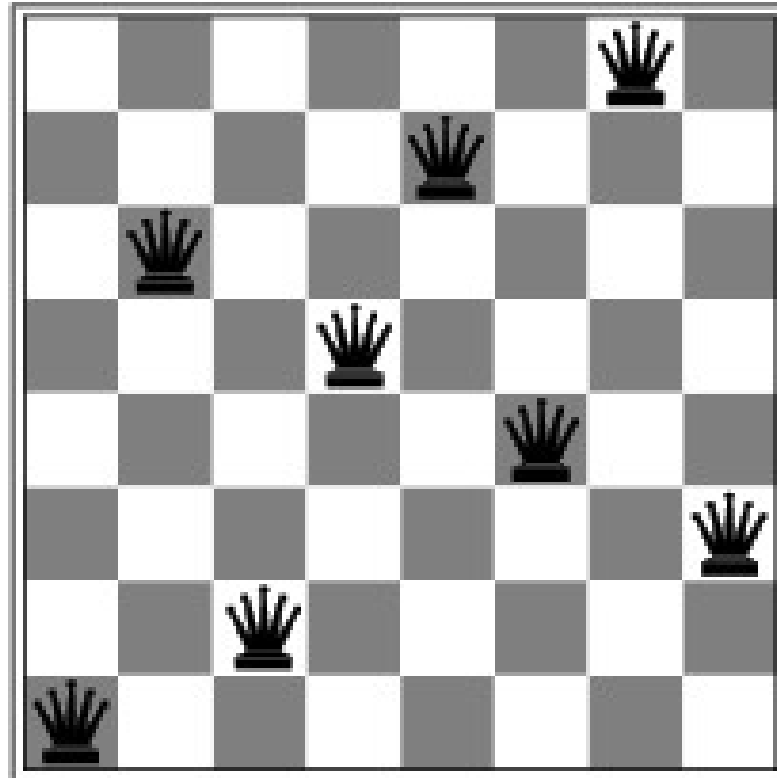
  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```

Example: 8-Queen

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♙	13	16	13	16
♙	14	17	15	♙	14	16	16
17	♙	16	18	15	♙	15	♙
18	14	♙	15	15	14	♙	16
14	14	13	17	12	14	12	18

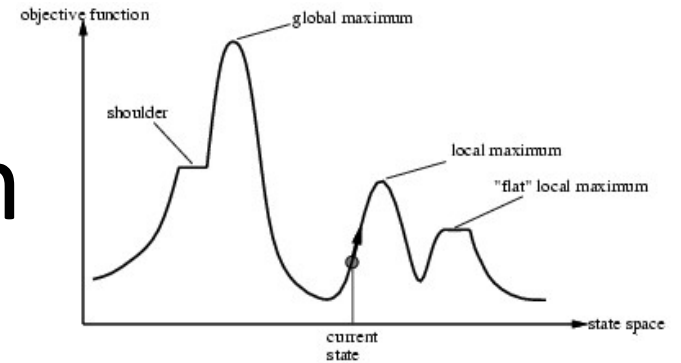
- h = number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state

Example: 8-Queen



- A local minimum with $h = 1$

More on Hill Clim



- Complete? Optimal?
- Hill climbing is sometimes called **greedy local search**
- Although greedy algorithms often perform well, hill climbing gets stuck when:
 - Local maxima/minima
 - Ridges
 - Plateau (shoulder or flat local maxima/minima)
- The steepest-ascent hill climbing solves only 14% of the randomly-generated 8-queen problems with an avg. of 4 steps
- **Allowing sideways move** raises the success rate to 94% with an avg. of 21 steps, and 64 steps for each failure

Variants of Hill Climbing

- **Stochastic hill climbing:**
 - chooses at random from among uphill moves
 - converges more slowly, but finds better solutions in some landscapes
- **First-choice hill climbing:**
 - generate successors randomly until one is better than the current
 - good when a state has many successors
- **Random-restart hill climbing:**
 - conducts a series of hill climbing searches from randomly generated initial states, stops when a goal is found
 - It's complete with probability approaching 1

More on Random-Restart Hill Climbing

- Assume each hill climbing search has a probability p of success, then the expected number of restarts required is $1/p$
- For 8-queen problem, $p = 14\%$, so we need roughly 7 iterations to find a goal
- Expected # of steps = $\text{cost_to_success} + (1-p)/p * \text{cost_to_failure}$
- Random-restart hill climbing is very effective for n-queen problem
- 3 million queens can be solved < 1 min

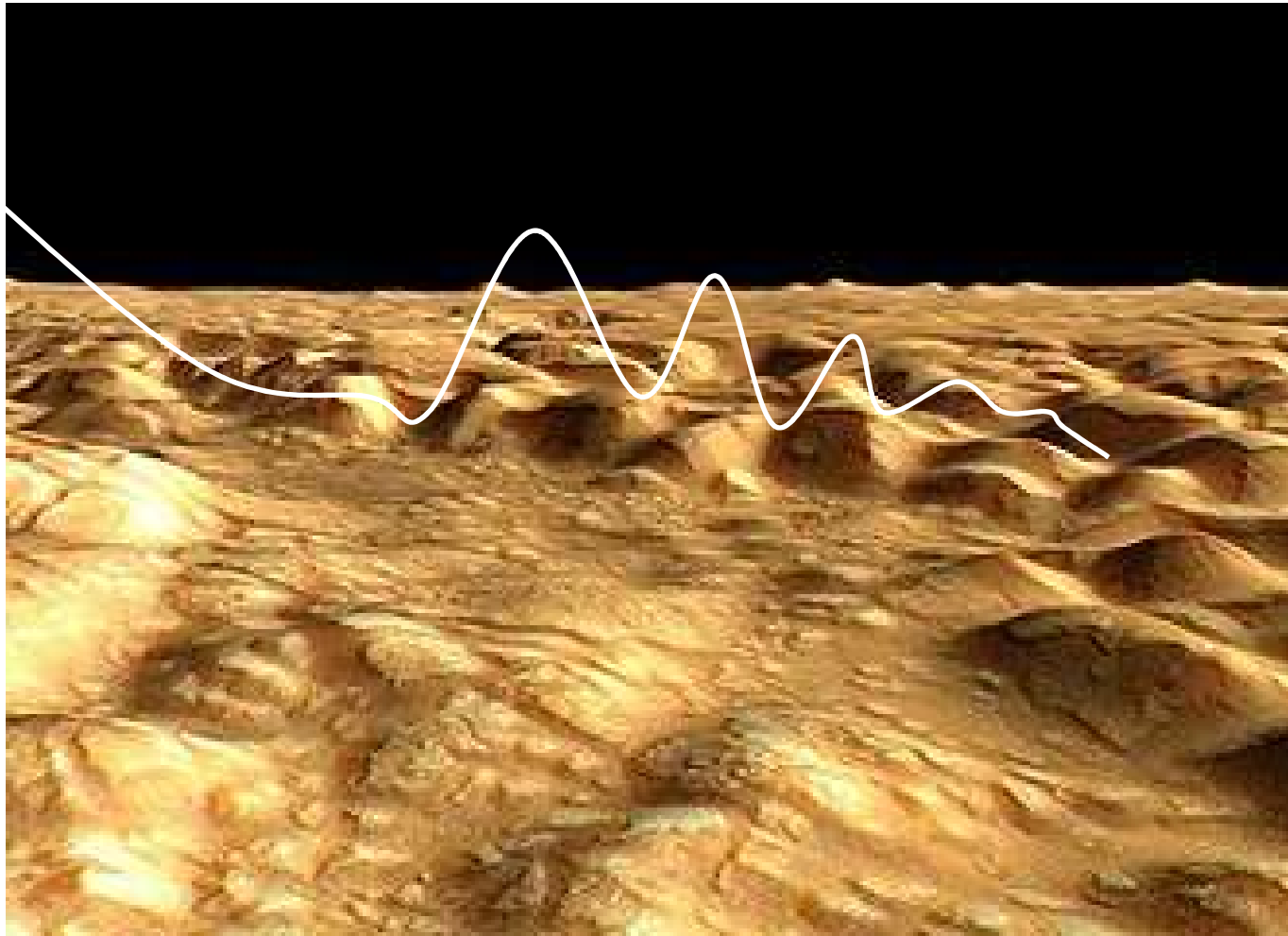
Some Thoughts

- NP-hard problems typically have an exponential number of local maxima/minima to get stuck on
- A hill climbing algorithm that never makes “downhill” (or “uphill”) moves is guaranteed to be incomplete
- A purely random walk – moving to a successor chosen uniformly at random – is complete, but extremely inefficient
- What should we do?
- Simulated annealing (P115)

What is Simulated Annealing?

- The process used to harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state

Ping-Pong Ball Example



Simulated Annealing

- 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
     $\Delta E \leftarrow \text{VALUE}[\textit{next}] - \text{VALUE}[\textit{current}]$ 
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E / T}$ 
```

Analysis of Simulated Annealing

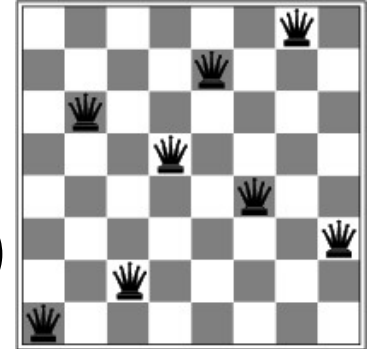
- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Widely used in VLSI layout, airline scheduling, etc.
- An example:
<http://foghorn.cadlab.lafayette.edu/fp/fpIntro.html>

Local Beam Search

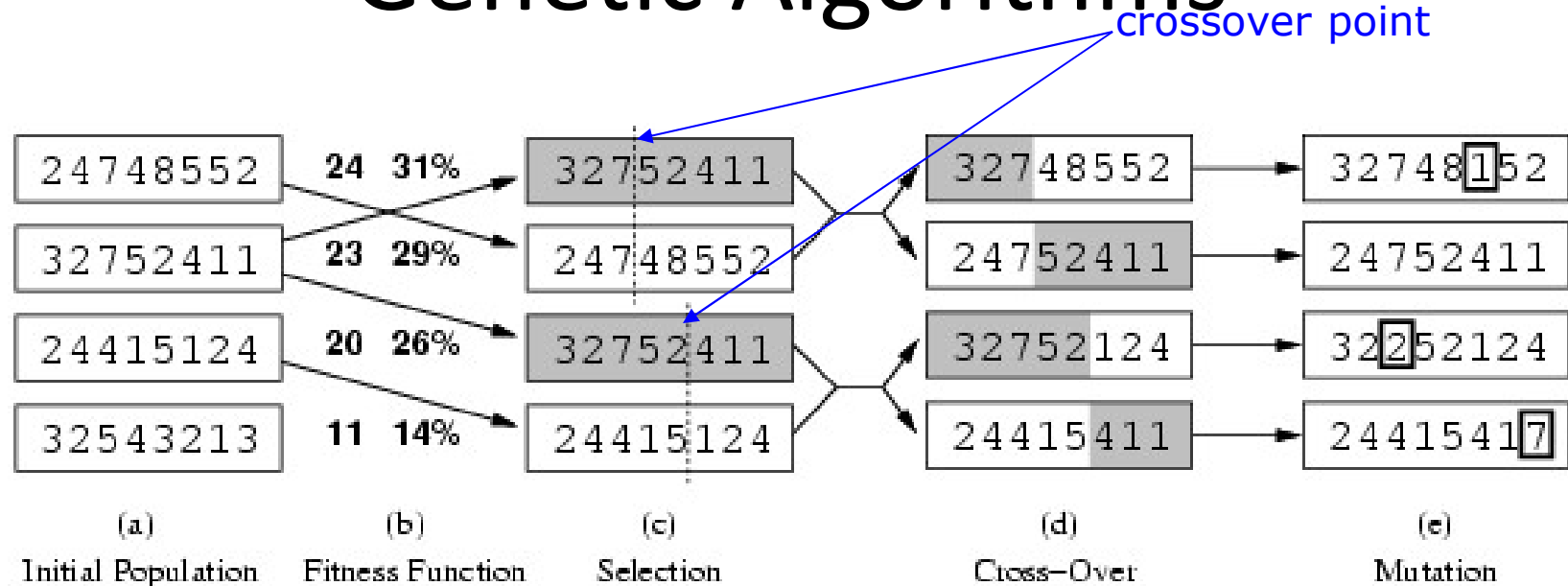
- Idea:
 - Keep track of k states rather than just one
 - Start with k randomly generated states
 - At each iteration, all the successors of all k states are generated
 - If any one is a goal state, stop; else select the k best successors from the complete list and repeat
- Is it the same as running k random-restart searches?
- Useful information is passed among the k parallel search threads
- **Stochastic beam search**: similar to natural selection, offspring of a organism populate the next generation according to its fitness

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 **16257483** (often a string of 0s and 1s or digits)
- Evaluation function (**fitness function**). Higher values for better states
- Produce the next generation of states by **selection**, **crossover**, and **mutation**

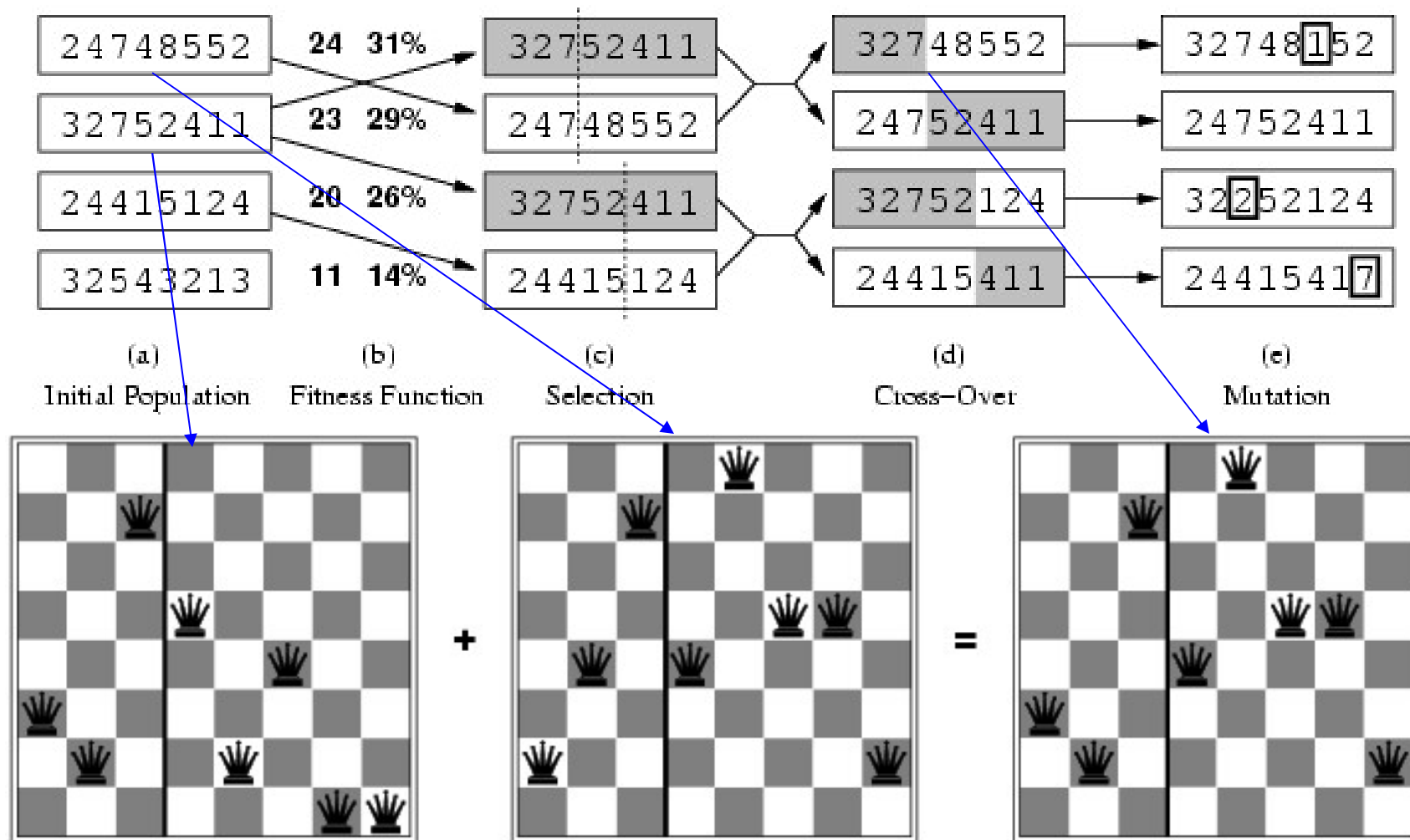


Genetic Algorithms



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

Example: 8-Queen



Example: TSA

- <http://www.obitko.com/tutorials/genetic-algorithms/>

More on Genetic Algorithms

- Genetic algorithms combine an uphill tendency with random exploration and exchange of information among parallel search threads
- Advantages come from “crossover”, which raise the level of granularity

A Genetic Algorithm

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual

inputs: *population*, a set of individuals

FITNESS-FN, a function that measures the fitness of an individual

repeat

new_population \leftarrow empty set

loop for *i* **from** 1 **to** SIZE(*population*) **do**

x \leftarrow RANDOM-SELECTION(*population*, FITNESS-FN)

y \leftarrow RANDOM-SELECTION(*population*, FITNESS-FN)

child \leftarrow REPRODUCE(*x*, *y*)

if (small random probability) **then** *child* \leftarrow MUTATE(*child*)

 add *child* to *new_population*

population \leftarrow *new_population*

until some individual is fit enough, or enough time has elapsed

return the best individual in *population*, according to FITNESS-FN

function REPRODUCE(*x*, *y*) **returns** an individual

inputs: *x*, *y*, parent individuals

n \leftarrow LENGTH(*x*)

c \leftarrow random number from 1 to *n*

return APPEND(SUBSTRING(*x*, 1, *c*), SUBSTRING(*y*, *c* + 1, *n*))

Exercise

- 4.11 Give the name of the algorithm that results from each of the following special cases:
 - Local beam search with $k = 1$
 - Local beam search with one initial state and no limit on the number of states retained
 - Simulated annealing with $T = 0$ at all times (and omitting the termination test)
 - Genetic algorithm with population size $N = 1$

Searching With Partial Information

- We have covered: **Deterministic, fully observable** → **single-state problem**
 - agent knows exactly which state it will be in
 - solution is a sequence
- **Deterministic, non-observable** → **multi-state problem**
 - Also called **sensorless problems** (**conformant problems**)
 - agent may have no idea where it is
 - solution is a sequence
- **Nondeterministic and/or partially observable** → **contingency problem**
 - percepts provide **new** information about current state
 - often **interleave** search, execution
 - solution is a tree or policy
- **Unknown state space** → **exploration problem (“online”)**
 - states and actions of the environment are unknown

Example: Vacuum World

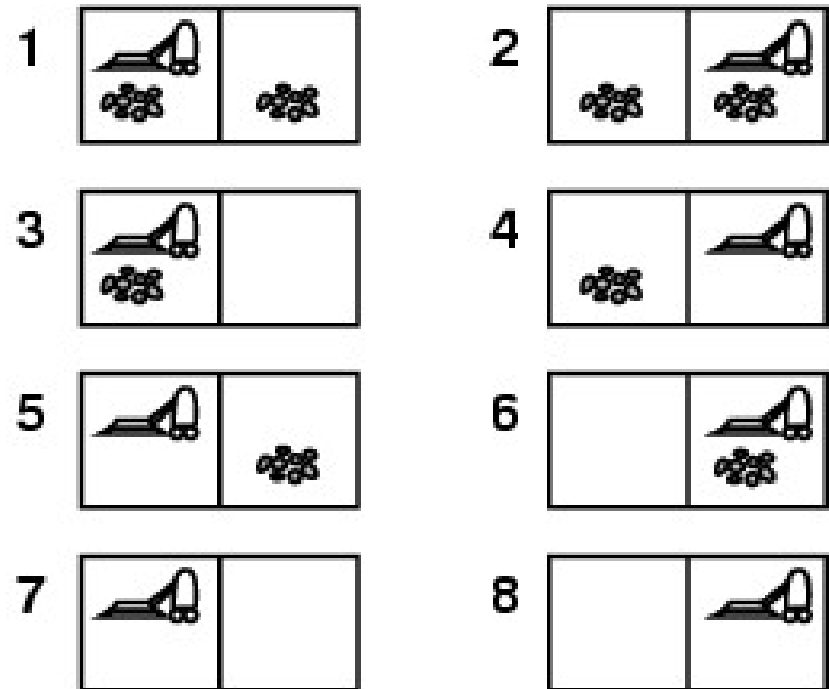
- **Single-state**, start in #5.

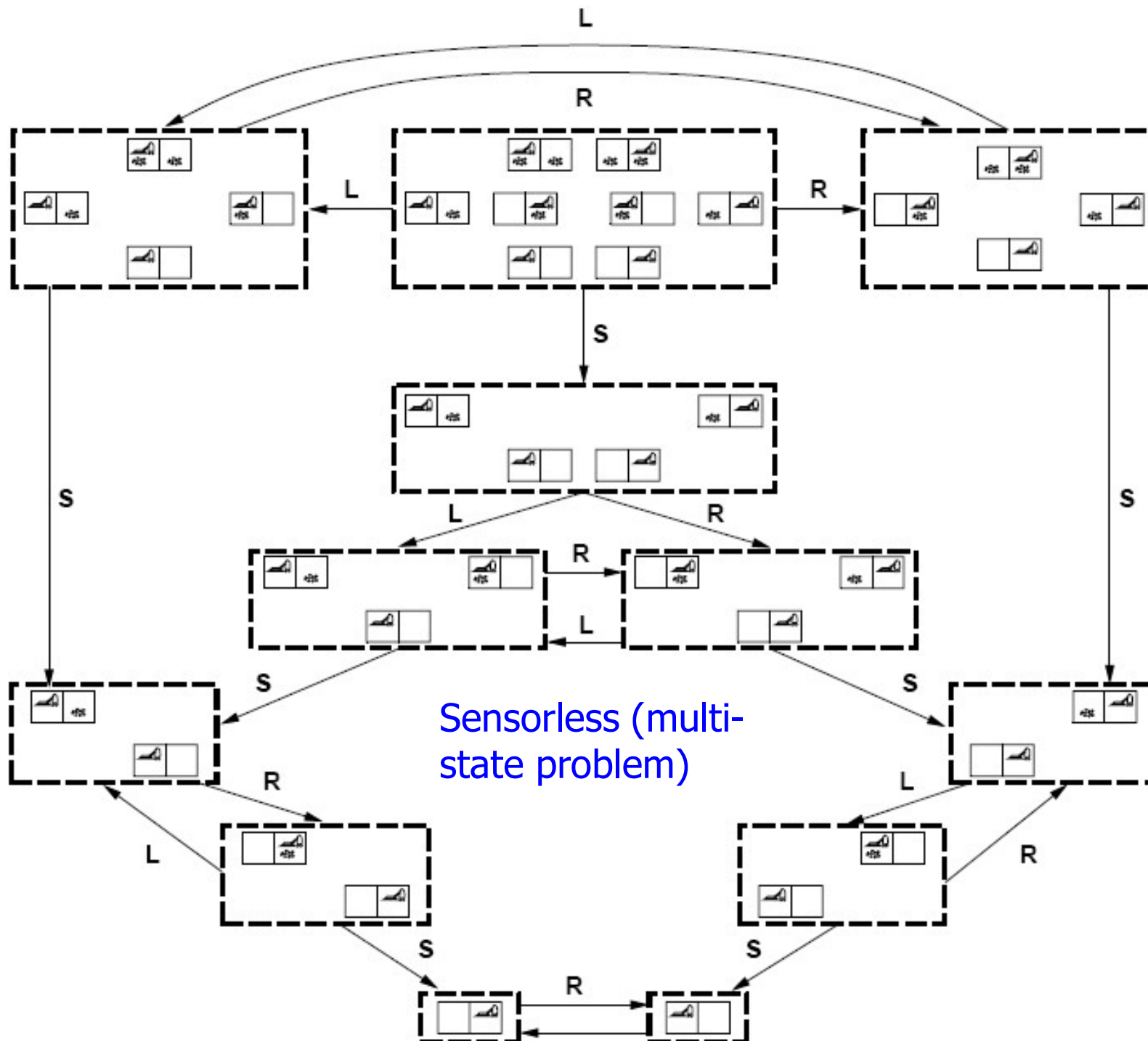
Solution?

– [Right, Suck]

- **Multi-state**, start in #[1, 2, ..., 8]. Solution?

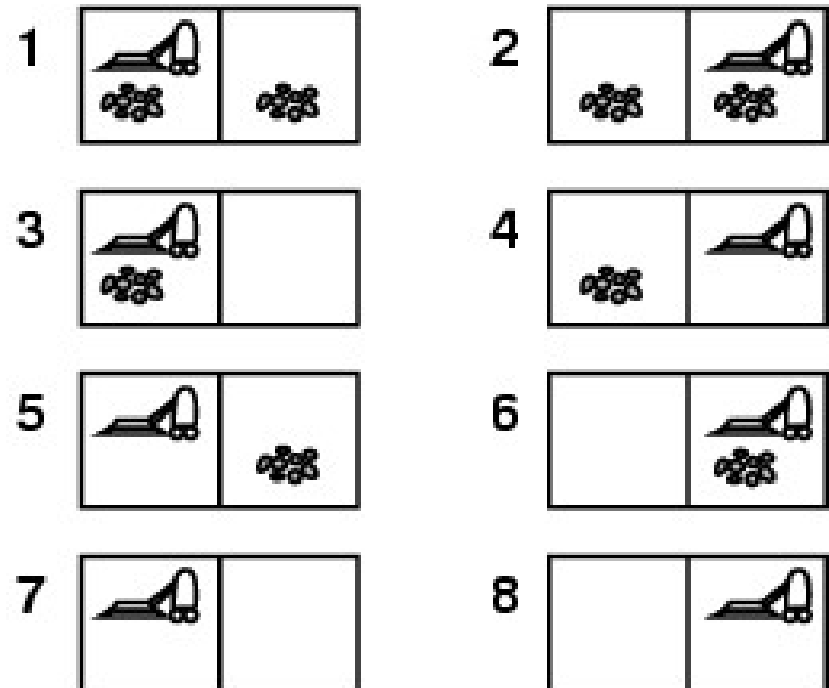
– [Right, Suck, Left, Suck]





Contingency Problem

- **Contingency**, start in #5 & 7.
 - *Nondeterministic*: suck may dirty a clean carpet
 - *local sensing*: dirt, location only at current location
 - Solution?
 - Percept: [Left, Clean] → [Right, **if** dirty **then** Suck]



Exam 2

- Closed book and closed notes
- Coverage:
 - Informed Search
 - Local Search
- Format:
 - True/False
 - Short-Answer
 - Problem Solving

Informed Search

- Understand what is informed search
- Understand best-first, greedy, and A^*
- Understand the optimality and completeness of each search
- Be able to draw search tree
- Understand the heuristic function in A^* , under what condition will A^* be optimal?
 - Admissible heuristic
 - Consistent heuristic

Local Search

- Hill climbing (HC)
 - Steepest HC, stochastic HC, 1st choice HC, random-restart HC
 - HC with side-moves allowed
 - Simulated annealing
 - Local beam search
 - Genetic algorithms
-
- Understand each search strategy
 - Understand how to define the objective function for local search