Introduction to Artificial Intelligence

Beyond Classical Search

Jianmin Li

Department of Computer Science and Technology
Tsinghua University

Spring, 2024

Review

- Assumptions about environment
  - Observable => the agent always knows the current state
  - Discrete => at any given state there are only finitely many actions to choose from
  - Known => the agent knows which states are reached by each action
  - Deterministic => each action has exactly one outcome

Review

- The solution to any problem is a fixed sequence of actions
- Tree search algorithm and graph search algorithm
- Search strategy
  - uninformed: BFS,UCS,DFS,IDS
  - informed: Greedy best-first, A*

Today

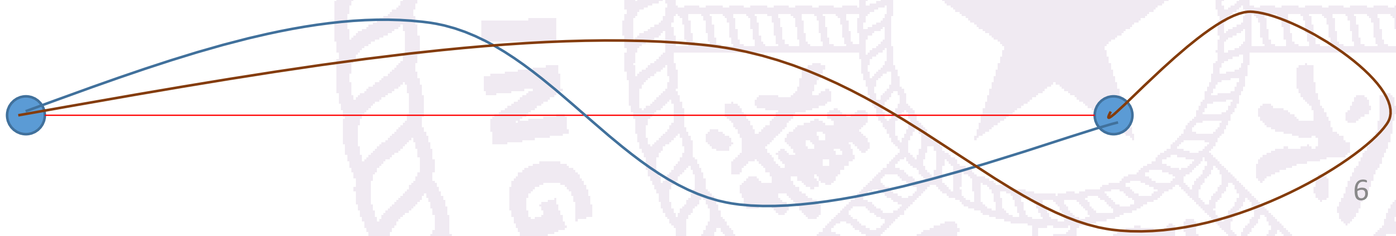
Relaxing the simplifying assumptions,
getting closer to the real world

- Local Search
- Search with Nondeterministic Actions
- Search with Partial Observations

Local Search

Path is irrelevant

- In many optimization problems, path is irrelevant, the goal state itself is the solution
  - *n*-queens problems
  - IC design
  - communications network optimization
- State space = set of "complete" configurations
  - find optimal configuration, e.g., Travelling Salesperson Problem (TSP)
  - find configuration satisfying constraints, e.g., timetable

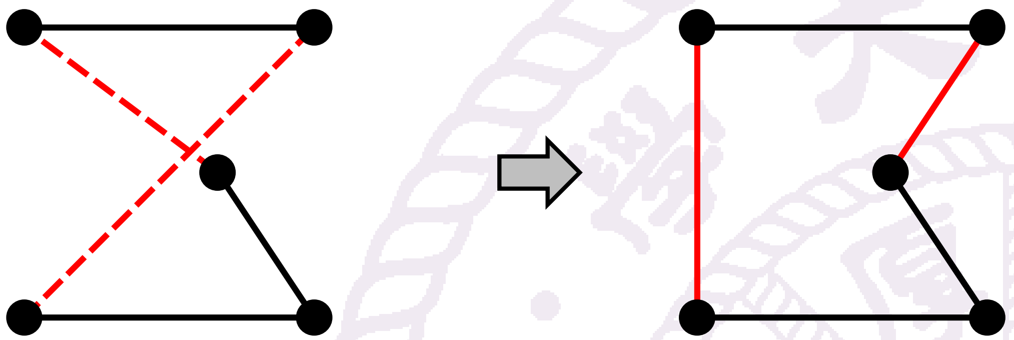


Iterative Improvement Algorithms

- Iterative improvement algorithms (local search algorithms)
  - keep a single “current” state
  - try to improve it by moving to neighbors
- Advantages
  - very little memory: usually constant space
  - reasonable solutions in large or infinitely state spaces
  - suitable for online as well as offline search

Example: Travelling Salesperson Problem

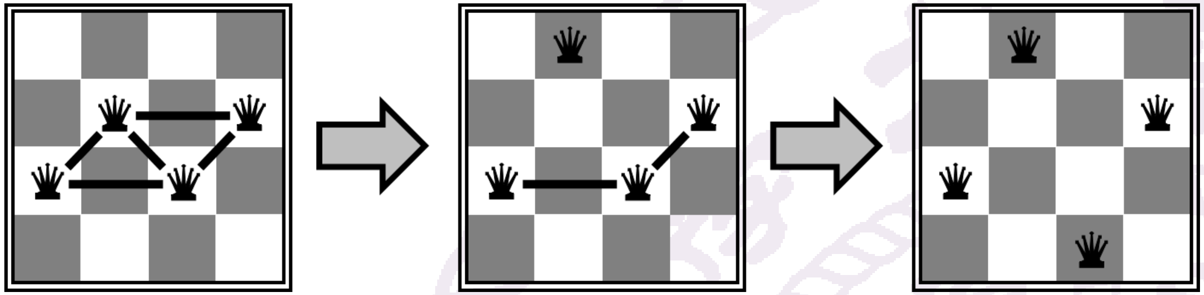
- Start with any complete tour, perform pairwise exchanges



- Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: *n*-queens

- Put *n* queens on an *n*x*n* board with no two queens on the same row, column, or diagonal
- Move a queen to reduce number of conflicts



***h* = 5** **_h_ = 2** **_h_ = 0**

- Almost always solves *n*-queens problems almost instantaneously
  - for very large *n*, e.g., *n* = 1million

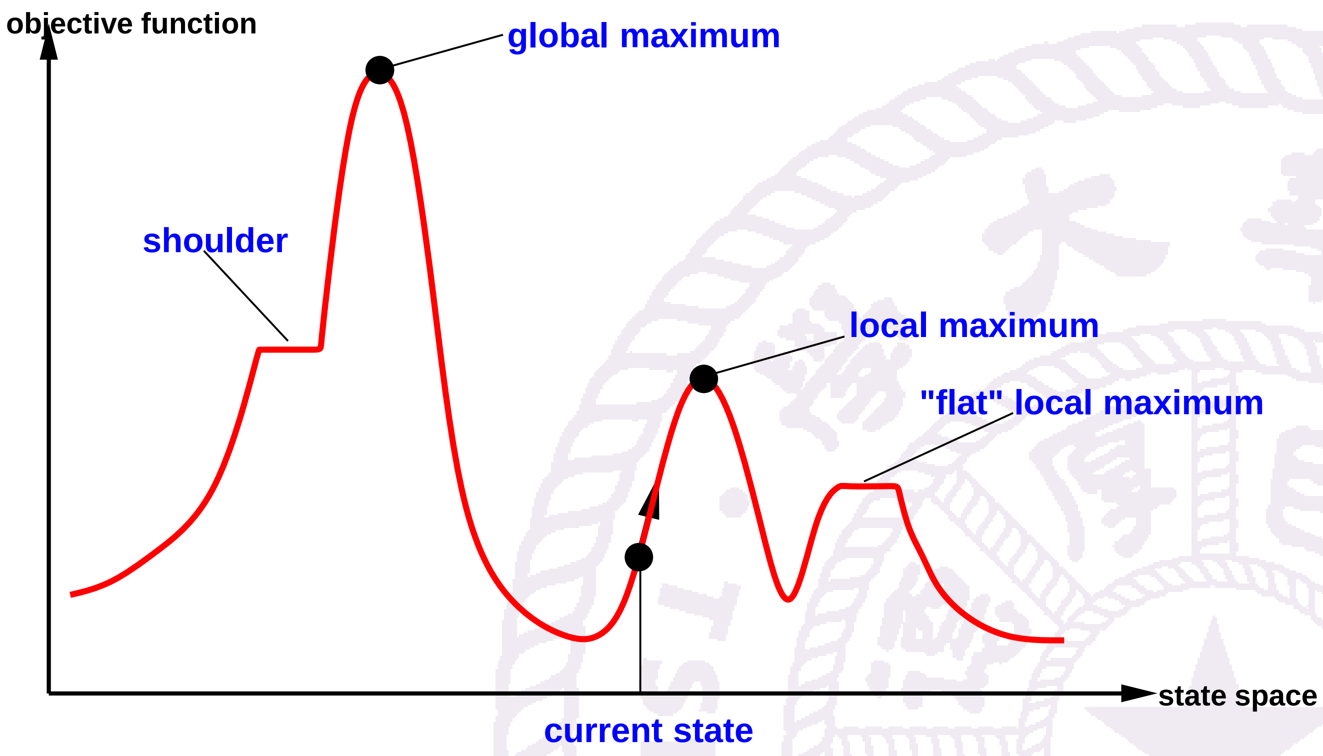
Hill-climbing

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

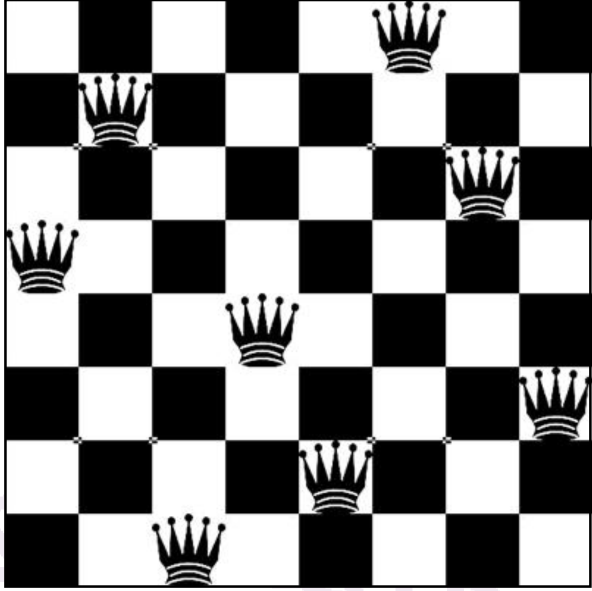
current ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
neighbor ← a highest-valued successor of *current*
if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
current ← *neighbor*

"Like climbing Everest in thick fog with amnesia"

Hill-climbing



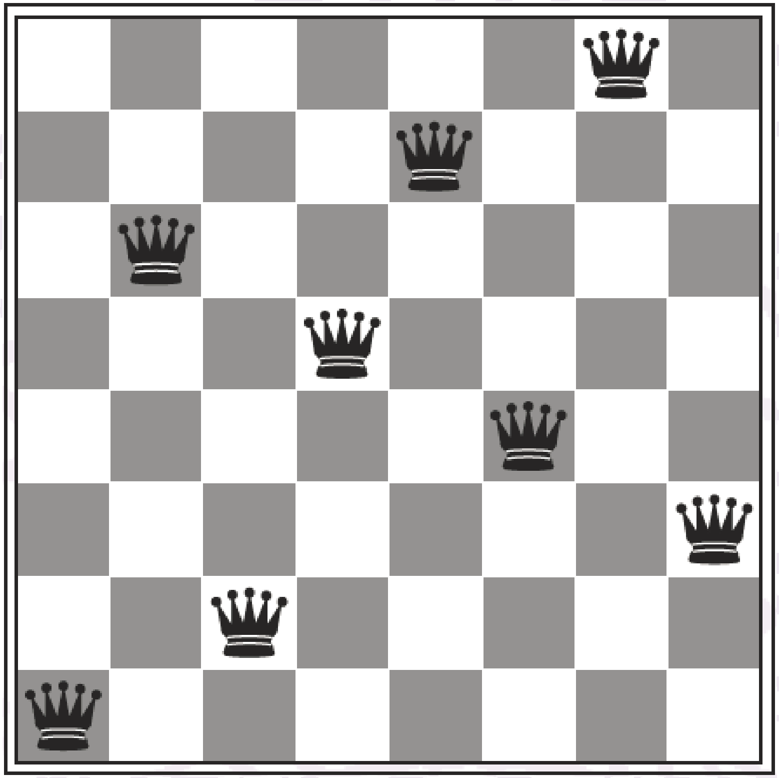
Hill-climbing: *n*-queens



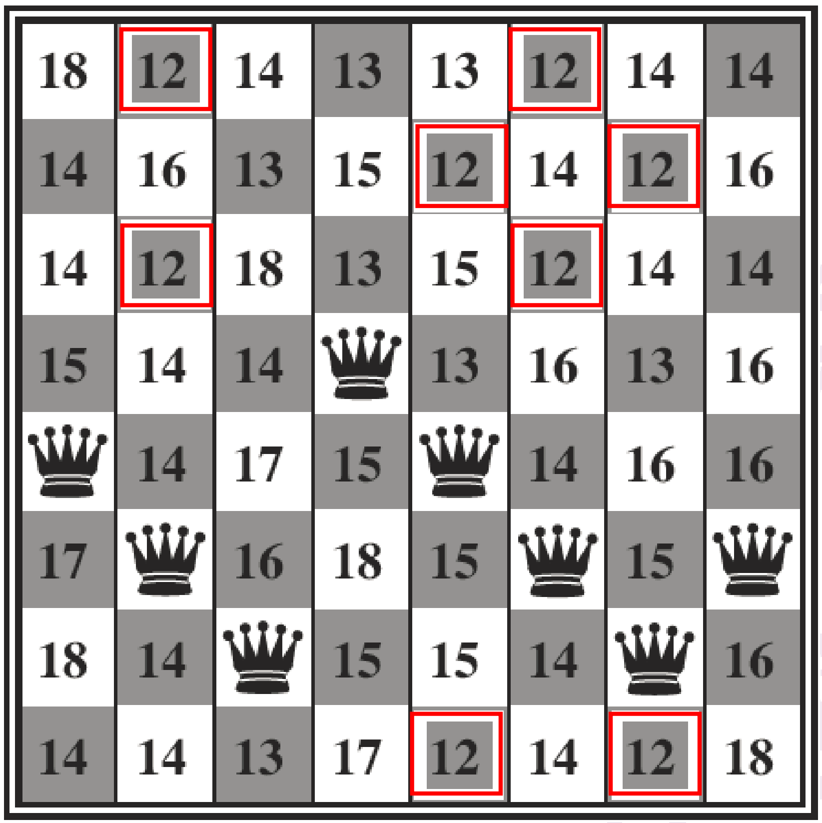
- Complete-state formulation
  - State
    - *n* queens on the board, one per column
  - Action
    - moving a single queen to another square in the same column
    - each state has 8*7 = 56 successors.
- Heuristic cost function *h*
  - the number of pairs of queens that are attacking each other, either directly or indirectly

Question 1

- An 8-queens state. heuristic cost estimate h = ?
    A. 0
    B. 1
    C. 2
    D. 3



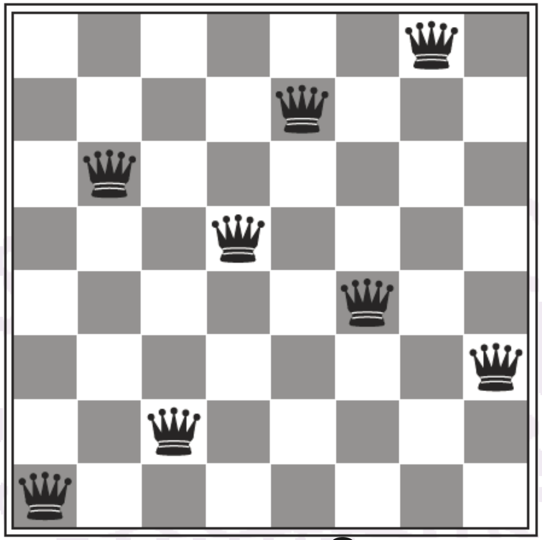
Hill-climbing: *n*-queens

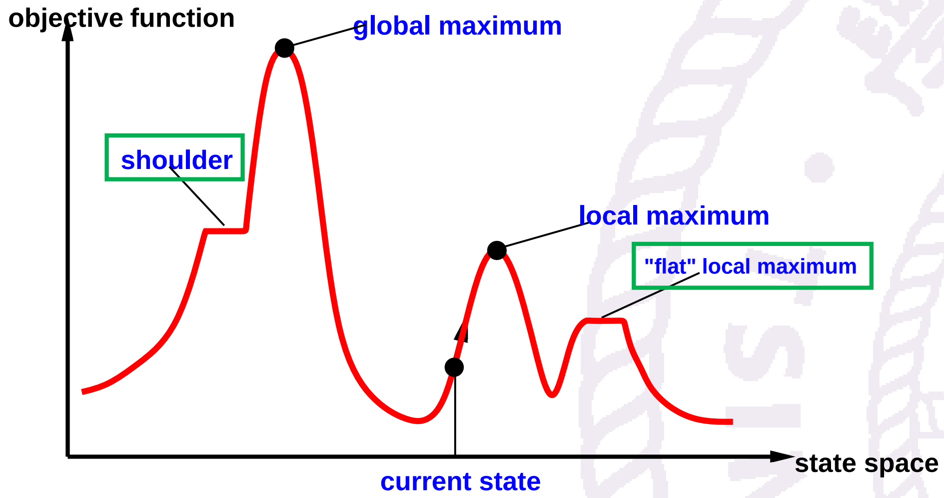


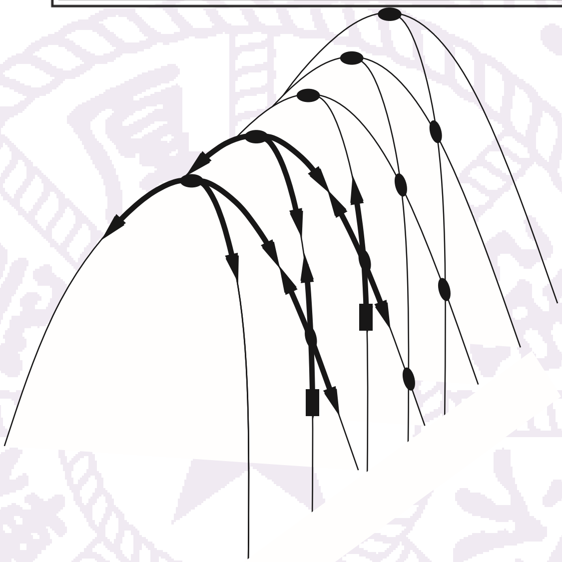
- An 8-queens state
- heuristic cost estimate $h = 17$
- Best successor, $h = 12$

Getting Stuck

- Local maxima
- Ridges
- Plateaux

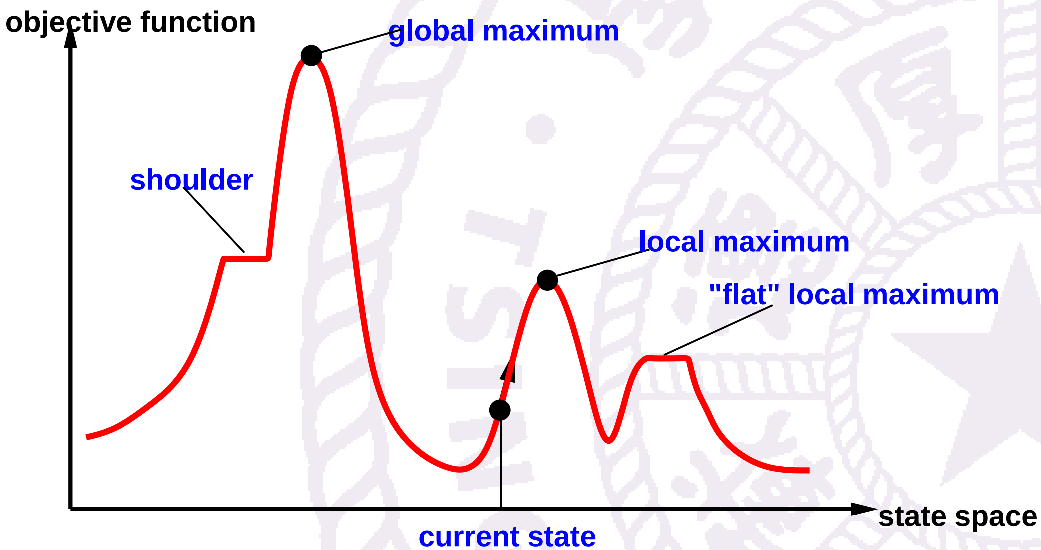






Improvements

- Random sideways moves
  - escape from shoulders
  - loop on flat maxima
- Random-restart hill climbing
  - overcomes local maxima - trivially complete



Simulated Annealing

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

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  inputs: $problem$, a problem
    $schedule$, a mapping from time to "temperature"

  $current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
  for t = 1 **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow next$.VALUE – $current$.VALUE
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

Properties of Simulated Annealing

- If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Devised by Metropolis et al. (1953) for physical process modeling
- Widely used in VLSI layout, airline scheduling, etc

Local Beam Search

- Keep track of (top) *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.
- Question: *k* searches run in parallel?

Local Beam Search

- Problem: all *k* states end up on same local hill
- Solution idea: choose *k* successors randomly, biased towards good ones (Stochastic Beam Search).
- Close analogy to natural selection

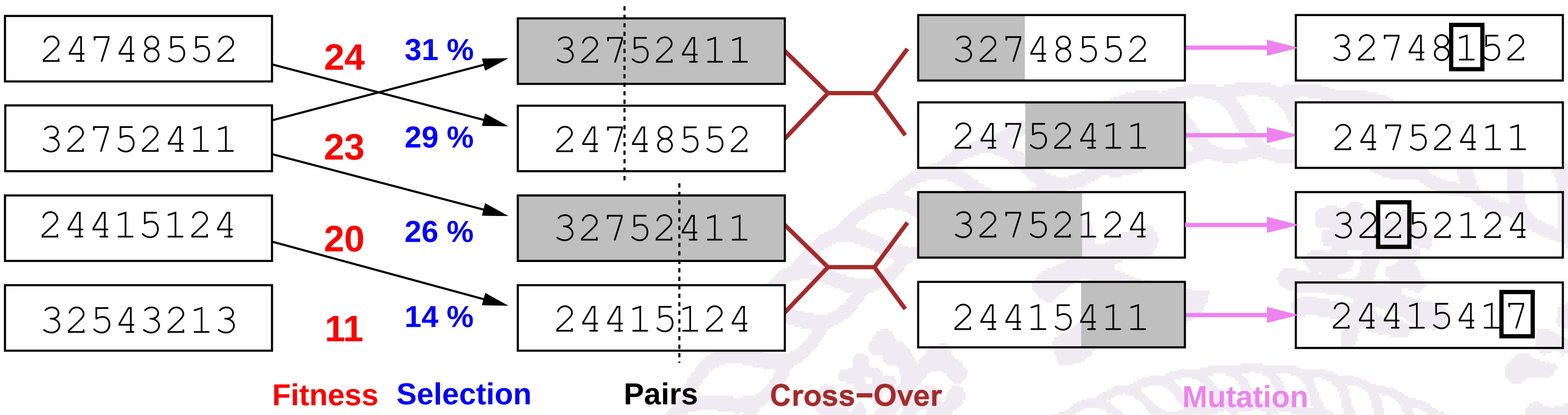
Genetic Algorithms

- Individual (i.e. state): each is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Population: Start with *k* randomly generated individuals
- Fitness function: evaluation of the “goodness” of a given state
- Produce the next generation of states by selection, crossover, and mutation

Genetic Algorithms

- A successor is generated by combining two parents from the current population
- stochastic local beam search + generate successor from pairs of states

Genetic Algorithms: *n*-queens



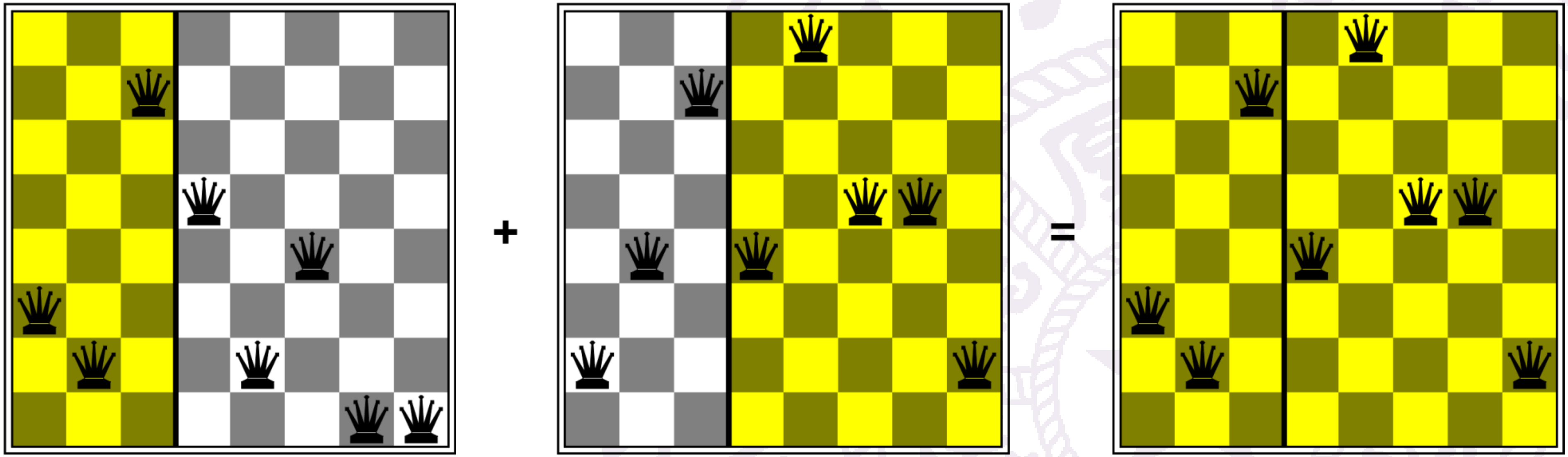
- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 x 7/2 = 28)

24/(24+23+20+11) = 31%
23/(24+23+20+11) = 29%

20/(24+23+20+11) = 26%
11/(24+23+20+11) = 14%

Encoding

- GAs require states encoded as strings
- Crossover helps iff substrings are meaningful components



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
    for i = 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

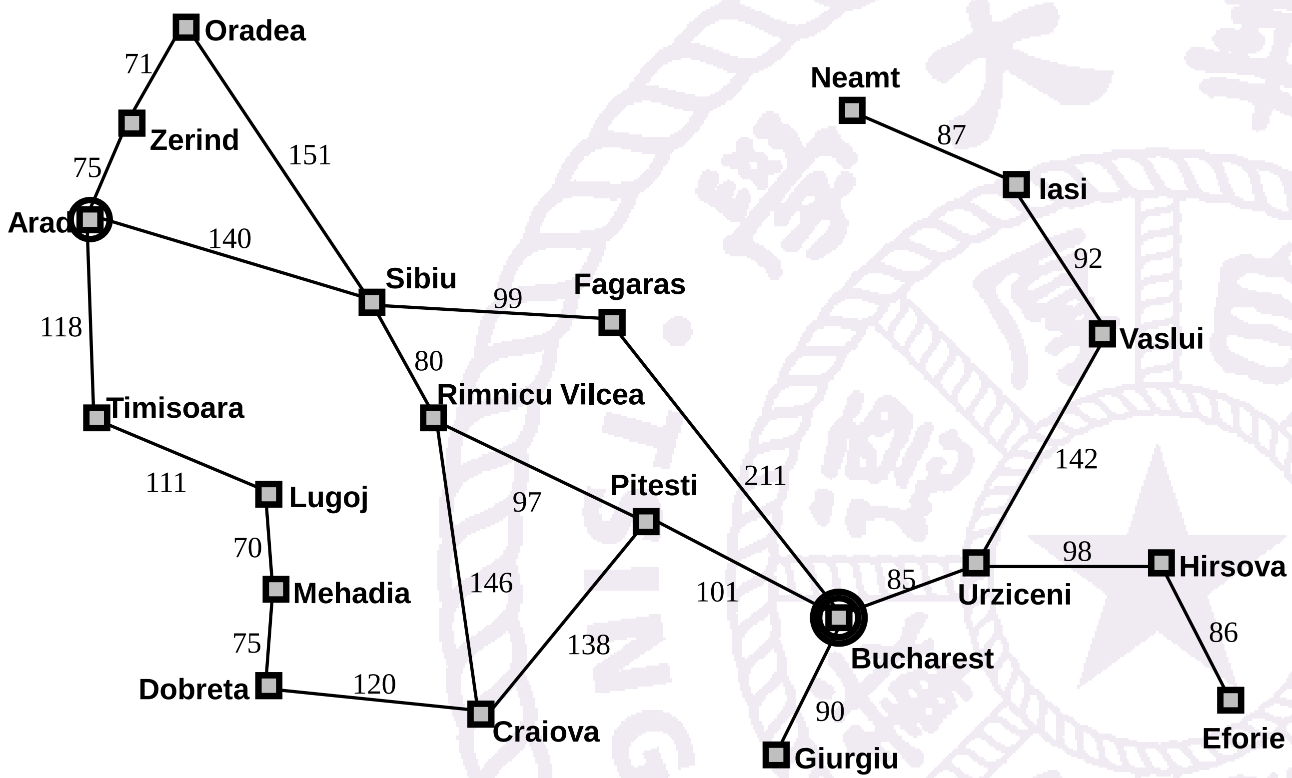
---

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($y, c + 1, n$))

Continuous Search Space

- Three new airports in Romania, such that the sum of squared distances from each city on the map to its nearest airport is minimized



Continuous Search Space

- 6-D state space

$$(x_1, y_1, x_2, y_2, x_3, y_3)$$

- Objective function
  - sum of squared distances from each city on the map to its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

  - C_i is the set of cities whose closest airport (in the current state) is airport i

Continuous Search Space

- Most real-world environments are continuous
  - a continuous state space

$$S = \{ \left(x_1, x_2 \cdots, x_N\right) \middle| x_i \in R \}$$

  - a continuous object function

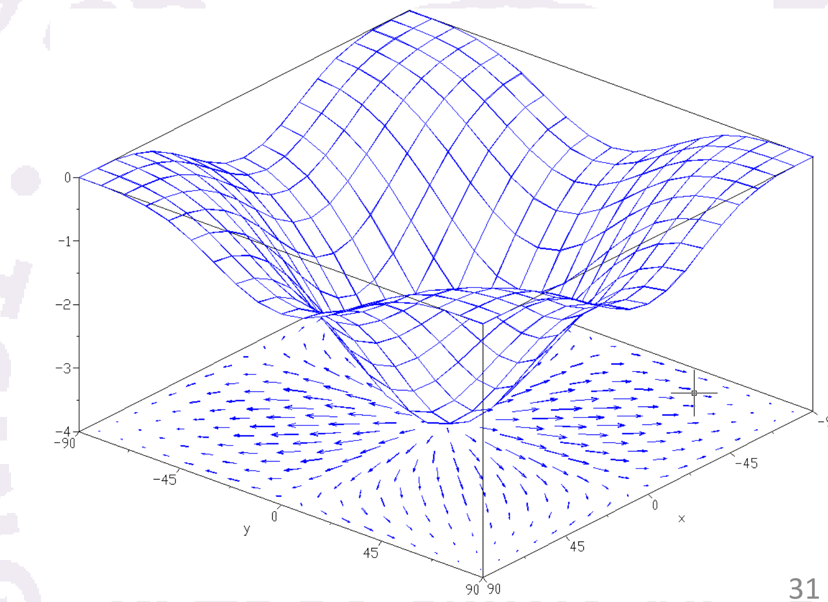
$$f(x_1, x_2 \cdots, x_N)$$

- Successor function would return infinitely many states!
- Discretization
  - Turns continuous space into discrete space

Gradient

- The gradient of the objective function is a vector
- The gradient gives the magnitude and direction of the steepest slope at a point
- empirical gradient considers $\pm\delta$ change in each coordinate

$$f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_N}\right)$$



the gradient of the function $f(x, y) = (\cos^2 x + \cos^2 y)^2$

Steepest-ascent Hill-climbing

$$x_{k+1} = x_k + \alpha \nabla f$$

- *a* is step size
  - if *a* is too small, too many steps are needed
  - if *a* is too large, the search could overshoot the maximum
- Line search
  - Extending the current gradient direction - usually by repeatedly
  - doubling *a* - until *f* starts to decrease again

Newton-Raphson Method

- To find a maximum or minimum of f , we need to find *x* such that the gradient is zero, i.e. $\nabla f(x) = 0$
- Newton-Raphson method
  - a general technique for finding roots of function, i.e. solving equations of the form $g(x) = 0$
  - computing a new estimate for the root *x* according to Newton's formula

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$

  - $g(x) = \nabla f(x)$

$$x_{k+1} = x_k - H_f^{-1}(x_k)\nabla f(x_k) \qquad H_{\{ij\}} = \frac{\partial^{\{2\}} f}{\partial x_i \partial x_j}$$

Search with Nondeterministic Actions

Percepts

- In previous slides we assume that
  - the environment is fully observable and deterministic
  - the agent knows what the effects of each action are
- Therefore, the agent
  - can calculate exactly which state results from any sequence of actions
  - and always knows which state it is in
- Its percepts provide no new information after each action

Percepts

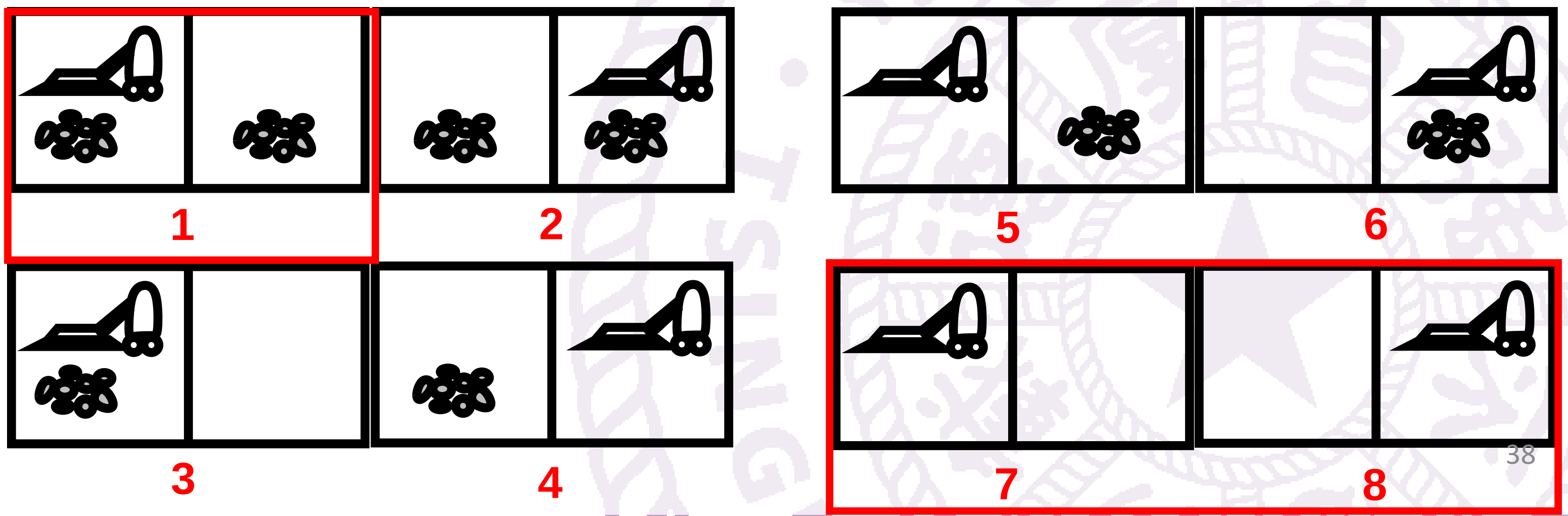
- Percepts become useful in partially observable or nondeterministic environment
  - In a partially observable environment, every percept helps narrow down the set of possible states the agent might be in, thus making it easier for the agent to achieve its goals
  - When the environment is nondeterministic, percepts tell the agent which of the possible outcomes of its actions has actually occurred

Percepts

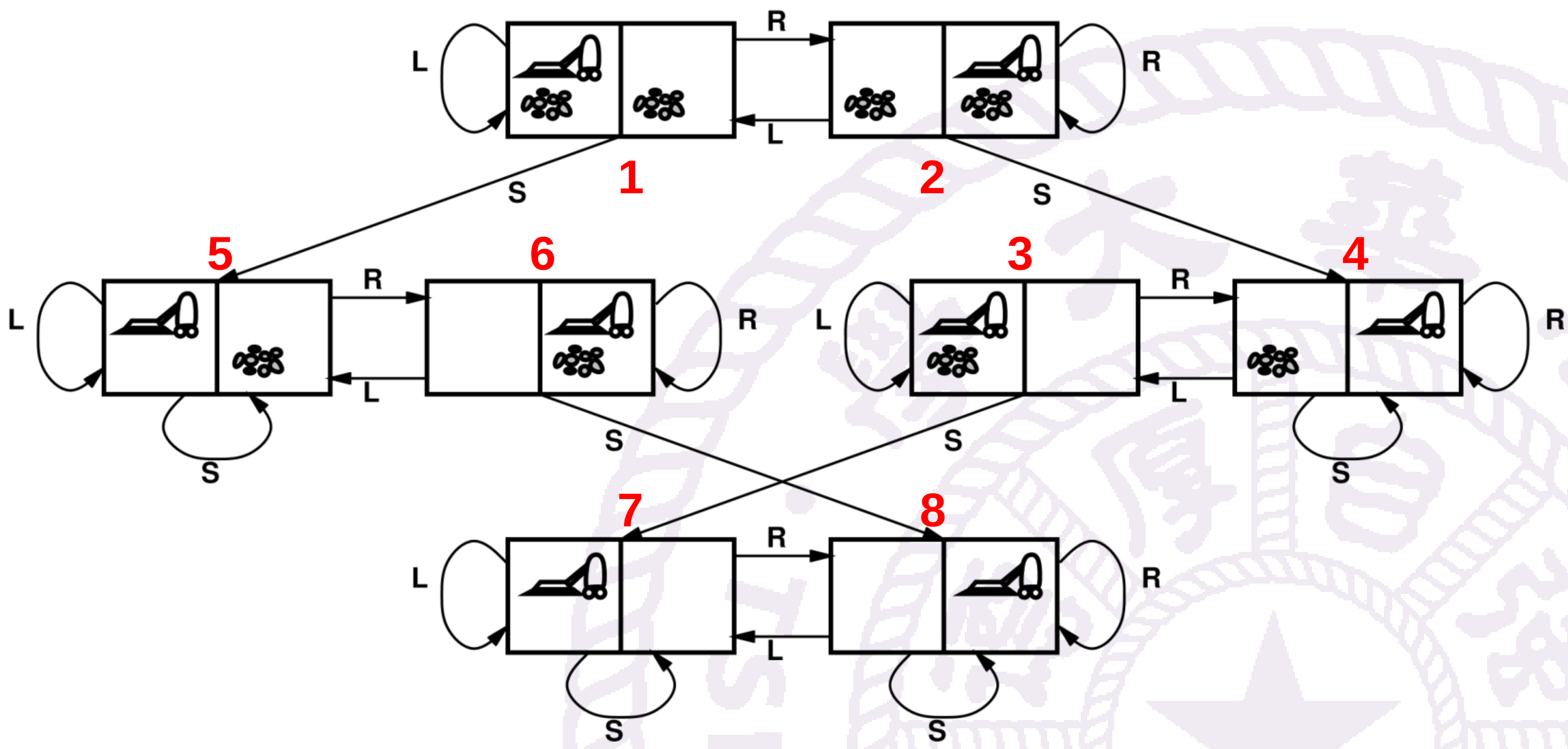
- The future percepts cannot be determined in advance and the agent's future actions will depend on those future percepts
- Solution to a problem is a contingency plan (or strategy) rather than a sequence of actions
  - specifies what to do depending on what percepts are received

Deterministic Vacuum World

- actions = {left, right, suck}
- environment is observable, deterministic, and completely known



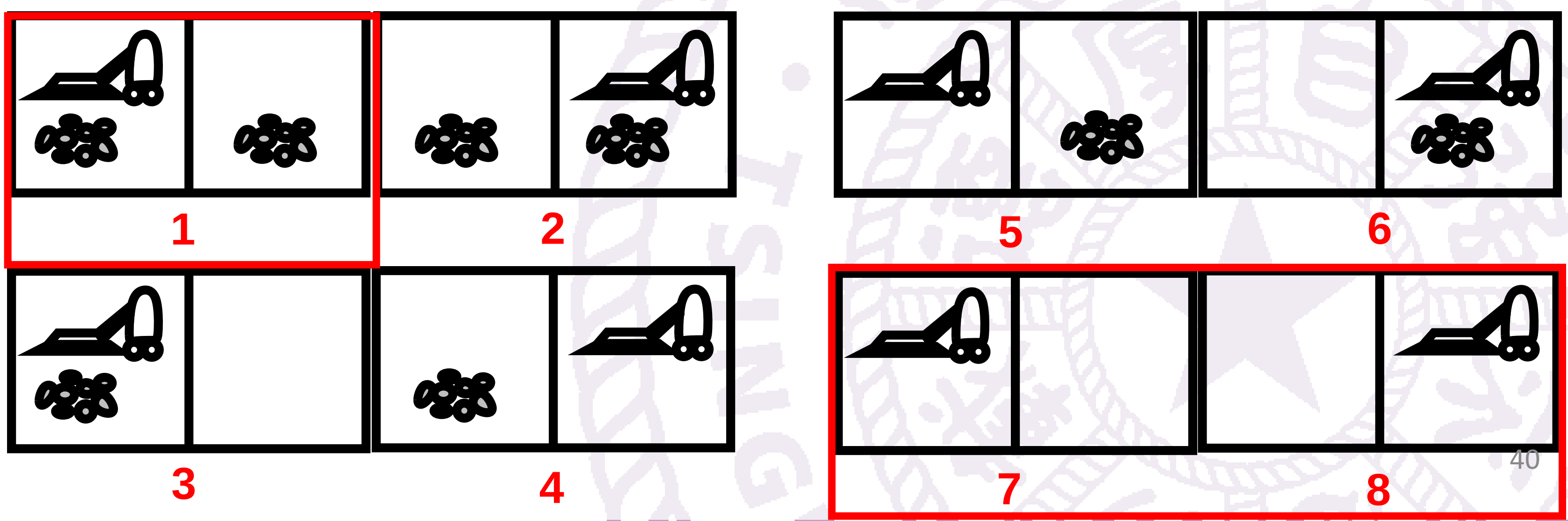
Deterministic Vacuum World



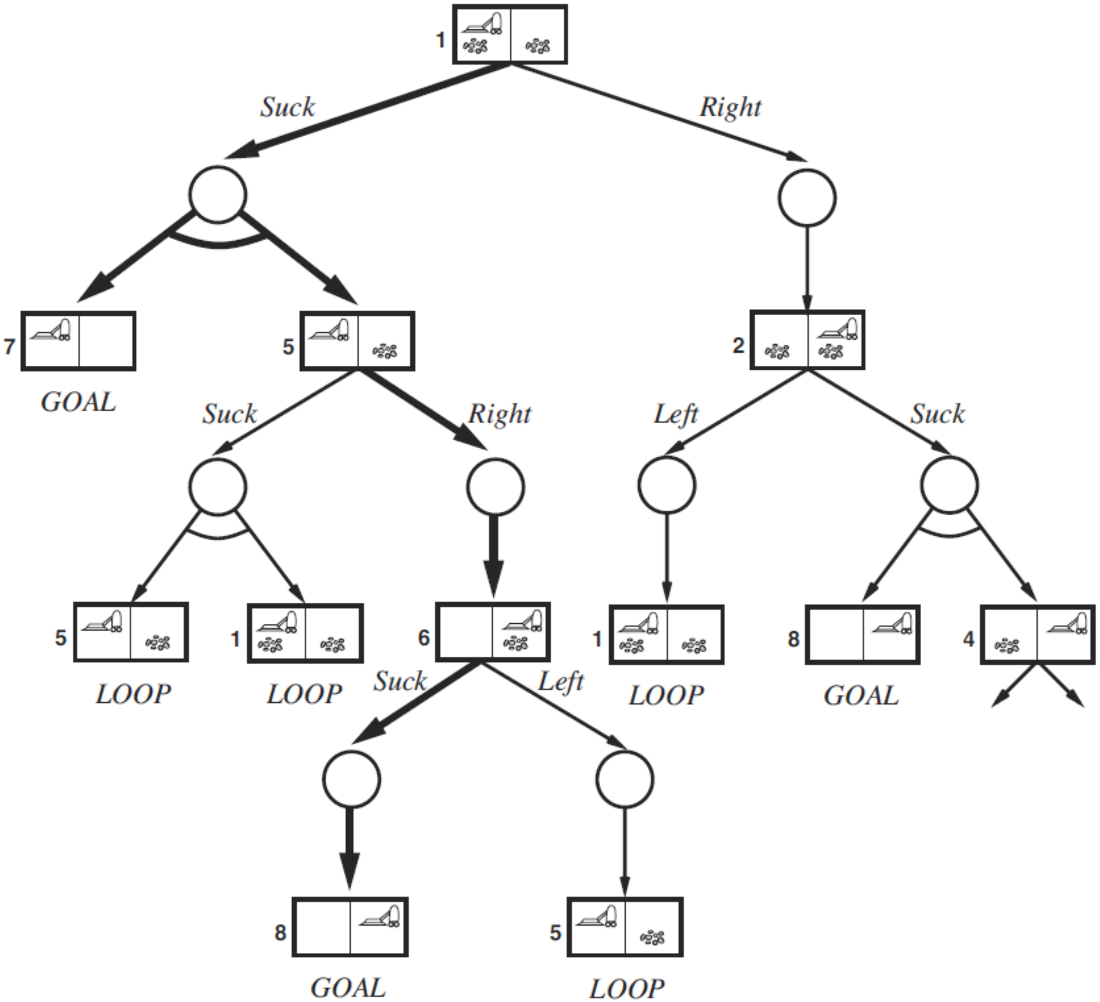
[Suck, Right, Suck]

Erratic Vacuum World

- actions = {left, right, suck}
- When sucking a dirty square, it cleans it and sometimes cleans up dirt in an adjacent square
- When sucking a clean square, it sometimes deposits dirt on the carpet



AND-OR Search Tree



OR nodes

- search possible actions

AND nodes

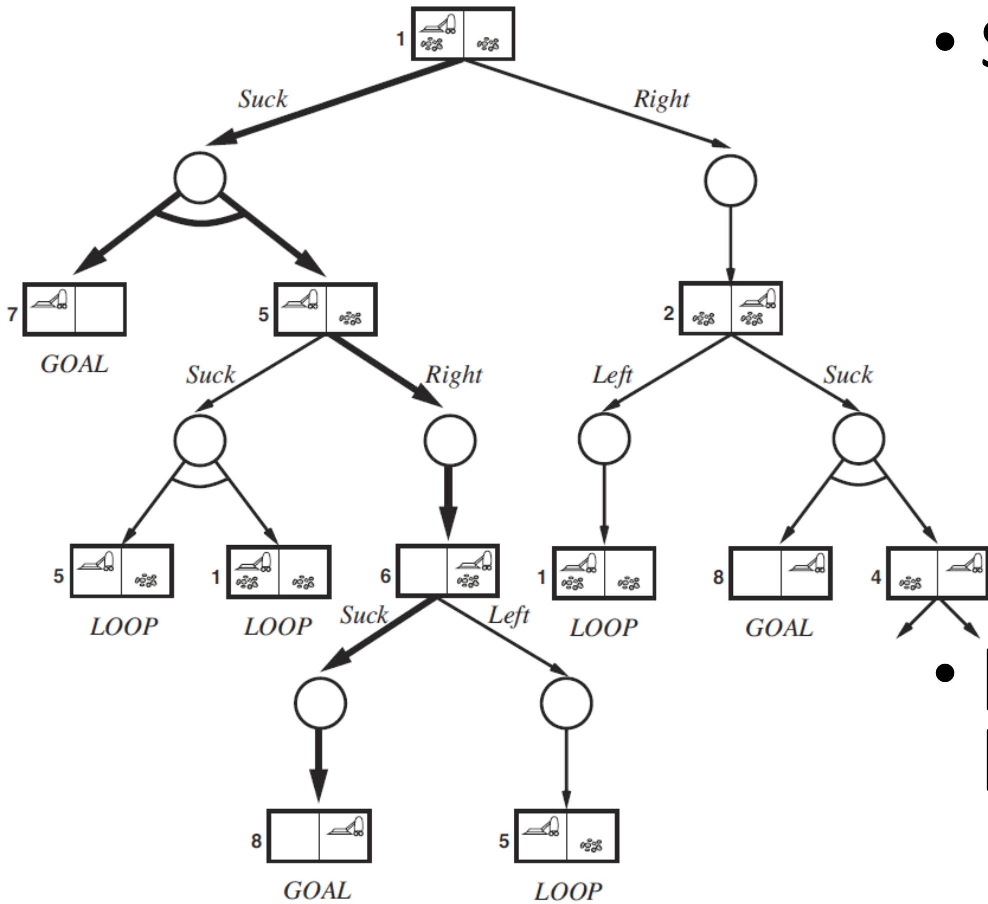
- consider all outcome states

Loops

- terminate search (on that path)

Suck action in state l leads to a state in the set {5, 7}, so the agent would need to find a plan for state 5 **and** for state 7.

AND-OR Search Tree



- Solution
  - a goal node at every non-loop leaf
  - specify one action at each of its OR nodes
  - includes every outcome branch at each of its AND nodes
- [Suck, if State = 5 then [Right, Suck] else [ ]]

A Recursive, Depth-first Solution

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
   OR-SEARCH(problem.INITIAL-STATE, problem, [])
```

```
function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
   if problem.GOAL-TEST(state) then return the empty plan
   if state is on path then return failure
   for each action in problem.ACTIONS(state) do
       plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
       if plan ≠ failure then return [action | plan]
   return failure
```

```
function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
   for each s_i in states do
       plan_i ← OR-SEARCH(s_i, problem, path)
       if plan_i = failure then return failure
   return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

- Save path to avoid loops
- Search over all the possibilities for an uncertain action outcome
- Solution is contingency plan, dealing with each possible outcome

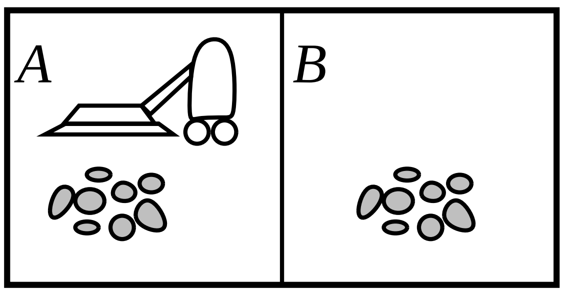
Search with Partial Observations

Uncertainty Strikes Twice

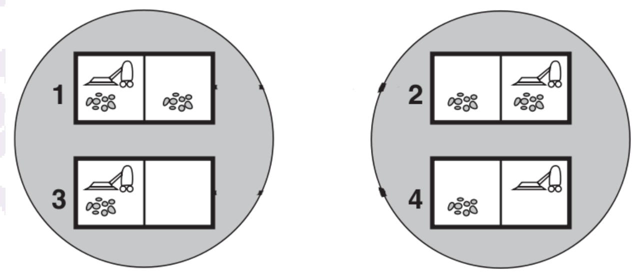
- Uncertainty in action
  - as above
- Uncertainty in sensing
  - no observation
  - partial observations

Belief State

- Belief state
  - representing the agent's current belief about the possible physical states it might be in



physical states for vacuum



belief states for sensorless vacuum

- To plan a sequence of actions, the agent searches a space of belief states, instead of a space of states

Searching with No Observation

- Sensorless problem
  - agent's percepts provide no information at all
- Is it possible for a sensorless agent to solve a problem if has no idea what state it's in?
  - Sensorless problems are quite often solvable

Searching with No Observation

- Sensorless agent can be surprisingly useful
- they don’t rely on sensors working properly
    - many ingenious methods in manufacturing systems
- the cost of sensing is too high
    - doctors often prescribe a broadspectrum antibiotic

Search in Belief State Space

- Search in space of belief states instead of physical states
- In belief-state space, the problem is fully observable
  - the agent always knows its own belief state
- Furthermore, the solution (if any) is always a sequence of actions
  - the percepts received after each action are completely predictable - they're always empty!
  - there are no contingencies to plan for
  - This is true even if the environment is nondeterministic

Sensorless Problem Definition

- Suppose the underlying physical problem P is defined by
  - $ACTIONS_P$, $RESULT_P$, $GOAL\text{-}TEST_P$ and $STEP\text{-}COST_P$
- How to define corresponding sensorless problem?

Sensorless Problem Definition

- Belief states
  - The entire belief-state space contains every possible set of physical states
  - Bad news
    - If P has N states, then the sensorless problem N has up to 2^N states
  - Good news
    - Many may be unreachable from the initial state
- Initial state
  - Typically the set of all states in P
  - In some cases the agent will have more knowledge

Sensorless Problem Definition

- Actions
  - The agent is in belief state $b = \{s_1, s_2\}$, but
    $$ACTIONS_P(s_1) \neq ACTIONS_P(s_2)$$
  - The agent is unsure of what actions are legal
    - If illegal actions have no effect on the environment
      $$ACTIONS(b) = \bigcup_{s \in b} ACTIONS_p(s)$$
    - If an illegal action might be the end of the world
      $$ACTIONS(b) = \bigcap_{s \in b} ACTIONS_p(s)$$

Sensorless Problem Definition

- Transition model
  - For deterministic actions, the set of states that might be reached is

$$b' = RESULT(b, a)$$
$$= \{s' : s' = RESULT_P(s, a), and\ s \in b\}$$

  - For nondeterministic actions, the set of states that might be reached is

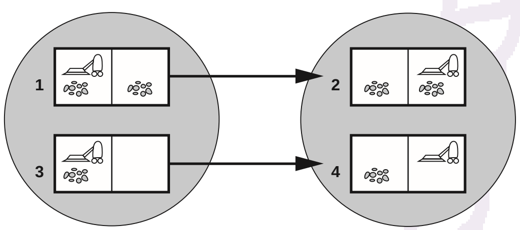
$$b' = RESULT(b, a)$$
$$= \{s' : s' \in RESULTS_P(s, a), and\ s \in b\}$$
$$= \bigcup_{s \in b} RESULTS_P(s, a)$$

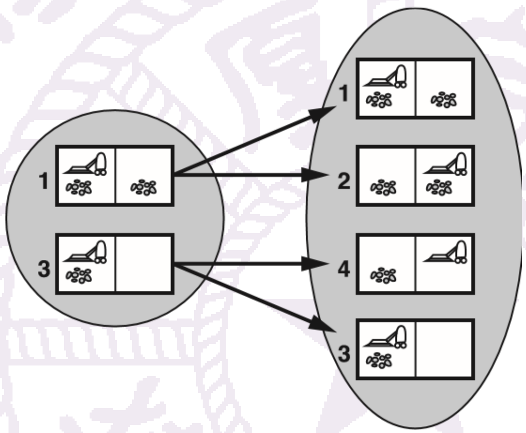
Sensorless Problem Definition

- Transition model
  - Prediction step
    - generating the new prediction belief state after the action

$$b' = PREDICT(b, a)$$

  - With deterministic actions, b' is never larger than b
  - With nondeterministic actions, b' may be larger than b





Sensorless Problem Definition

- Goal test
  - a belief state satisfies the goal only if all the physical states in it satisfy *GOAL-TEST*$_P$
  - The agent may accidentally achieve the goal earlier, but it won't know that it has done so

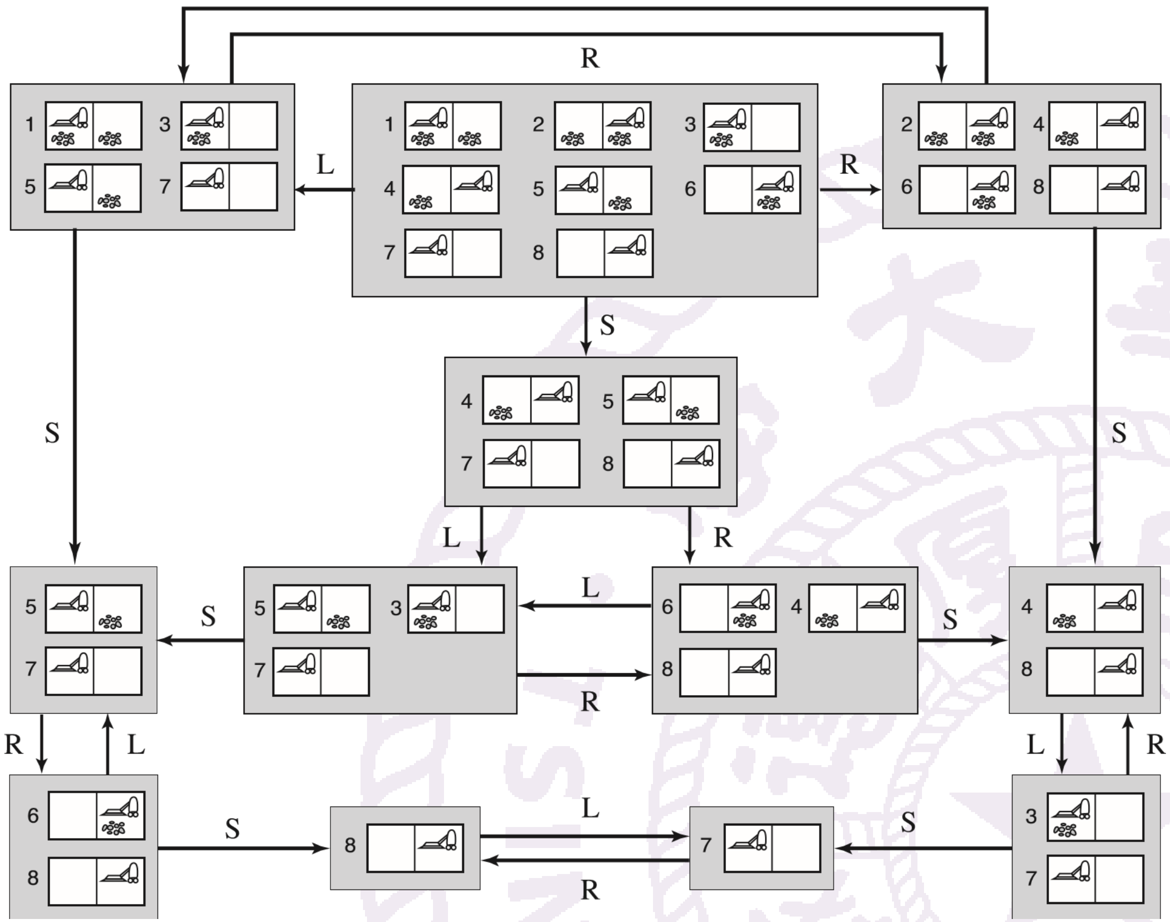
Sensorless Problem Definition

- Path cost
  - the cost of taking an action in a given belief state could be one of several values
  - A simple case
    - the cost of an action is the same in all states
    - can be transferred directly from the underlying physical problem

Deterministic, Sensorless Vacuum World

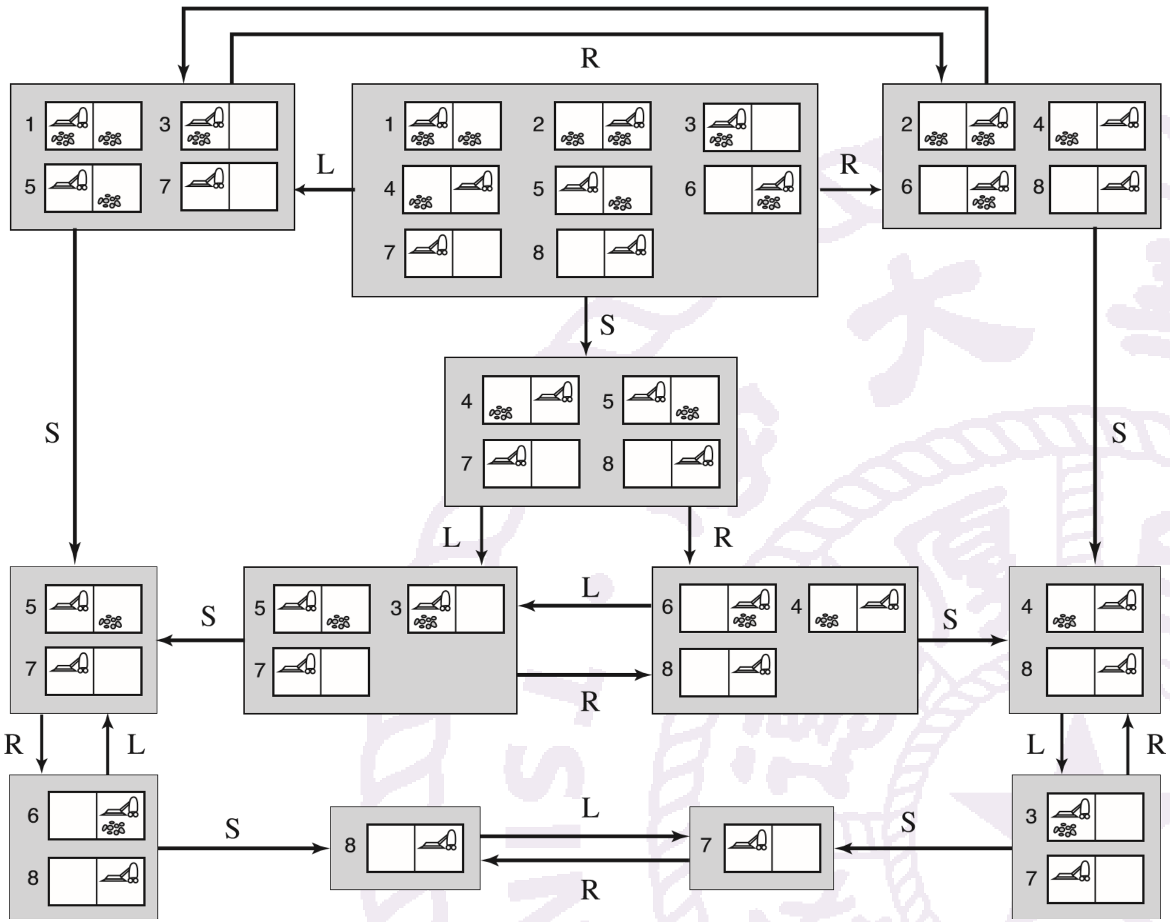
- Assume that the agent knows the geography of its world, but doesn’t know its location and the distribution of the dirt.
- Its initial state could be any element of the set of all states

Deterministic, Sensorless Vacuum World



How to reach the goal state (7)?
[right, suck, left, suck]

Sensorless Problem Solving



Apply any of the search algorithm we have known

Searching with Partial Observations

- How the environment generates percepts for the agent
  - Local-sensing vacuum world: a position sensor and a local dirt sensor, no global dirt sensor
- If sensing is deterministic, a *PERCEPT*(*s*) function that returns the percept received in a given state
  - Fully observable problem: *PERCEPT*(*s*)=*s* for every state *s*
  - Sensorless problem: *PERCEPT*(*s*)=null
- If sensing is nondeterministic, then function *PERCEPTS* that returns a set of possible percepts

Partially Observable Problem Definition

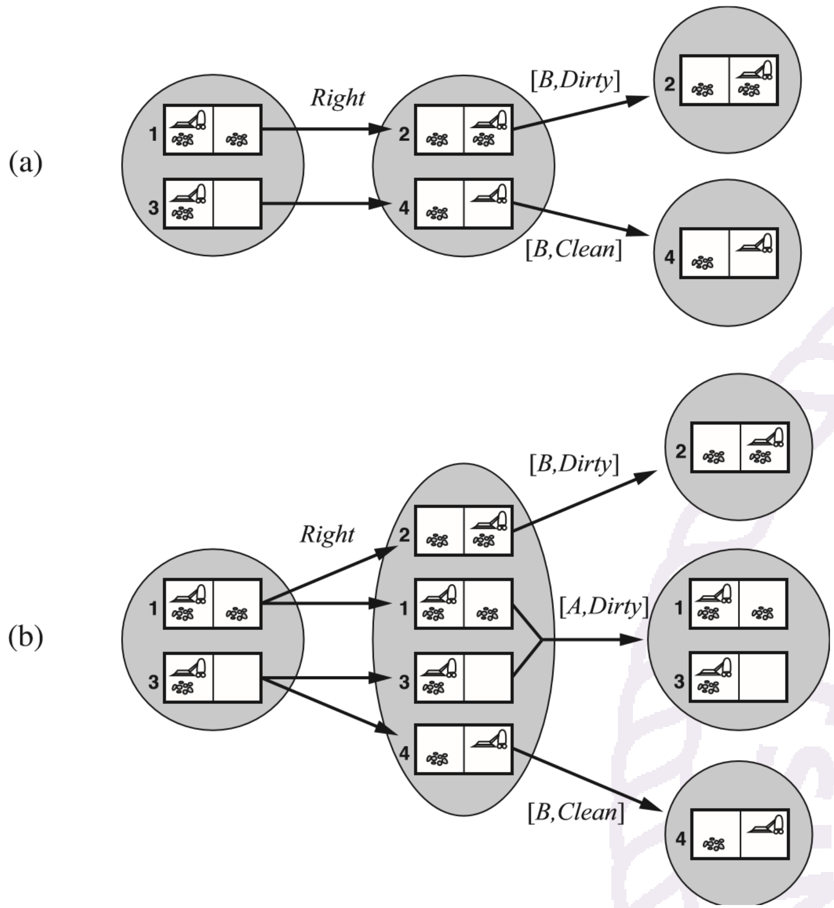
- *ACTIONS*, *STEP-COST*, and *GOAL-TEST* are the same as for sensorless problems
- The transition model is a bit more complicated
  - The prediction stage
    $$\hat{b} = PREDICT(b, a)$$
  - The observation prediction stage
    $$POSSIBLE\text{-}PERCEPTS(\hat{b}) = \{o : o = PERCEPT(s) \text{ and } s \in \hat{b}\}$$
  - The update stage
    $$b_o = UPDATE(\hat{b}, o) = \{s : o = PERCEPT(s) \text{ and } s \in \hat{b}\}$$

Partially Observable Problem Definition

- Possible belief states resulting from a given action and the subsequent possible percepts:

$$RESULTS(b, a)$$
$$= \{b_o : b_o = UPDATE(PREDICT(b, a), o) \text{ and } o$$

Partially Observable Problem Definition

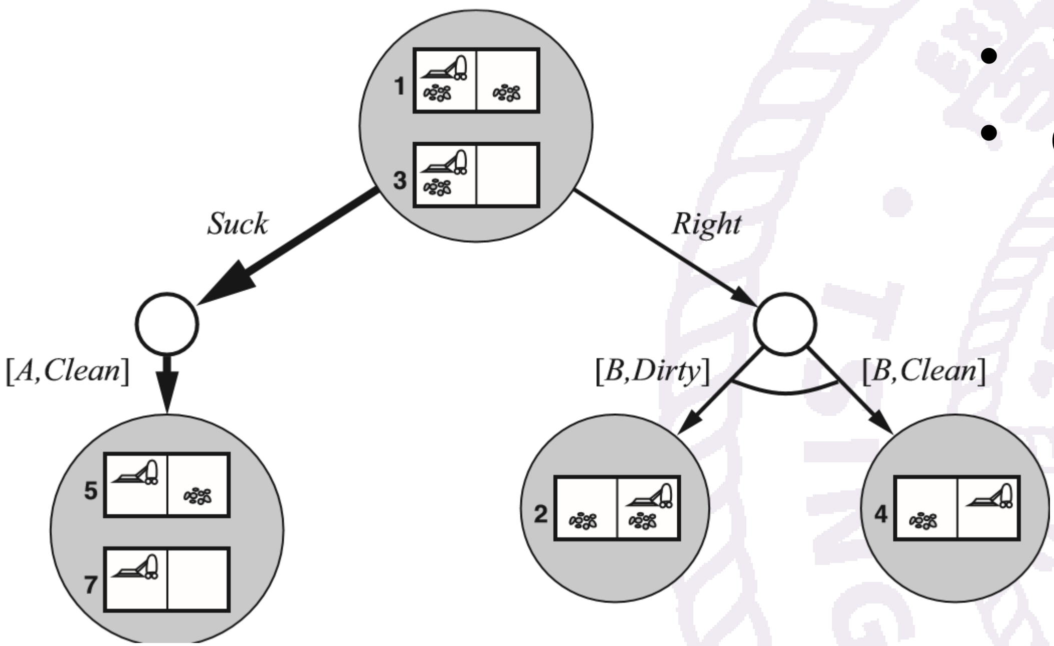


local-sensing vacuum worlds

- a. Deterministic world
- b. Slippery world

Solving Partially Observable Problems

- A nondeterministic belief-state problem with
  - *ACTIONS*, *RESULTS*, *GOAL-TEST*, *STEP-COSTS*
- AND-OR search algorithm can be applied to derive a solution



- initial percept [A, Dirty]
- conditional plan

```
[
    Suck,
    Right,
    if Bstate = {6}
    then Suck
    else []
]
```

Summary

- Local Search
  - Hill climbing
  - Simulated annealing
  - Genetic algorithms
  - Continuous space
- Search with Uncertainty
  - Nondeterministic Actions
  - Partial Observations

谢谢！