AI基础

Lecture 4： Search in Complex Environments

Bin Yang
School of Data Science and Engineering
byang@dase.ecnu.edu.cn

Lecture 3 ILOs

$$\begin{array}{rcc}
\text{A* search:} & g(n)+h(n) & (W=1) \\
\text{Uniform-cost search:} & g(n) & (W=0) \\
\text{Greedy best-first search:} & h(n) & (W=\infty) \\
\text{Weighted A* search:} & g(n)+W\times h(n) & (1<W<\infty)
\end{array}$$

- Informed Search Strategies
  - Heuristic function
  - Greedy best-first search
  - A* search, cost optimality, admissibility
  - Memory bounded search, weighted A* search
  - Design heuristics functions
    - Generating heuristics from relaxed problems
    - Generating heuristics from subproblems, pattern databases
    - Generating heuristics from with landmarks
    - Learning heuristics from experience
- Search in complex environments
  - Local search
    - Hill climbing
    - Simulated annealing
    - Local beam search
    - Evolutionary search

Lecture 4 ILOs

- Search in complex environments
  - Local search in continuous spaces
  - Search with nondeterministic actions
  - Search in partially observable environments
  - Online search agents and unknown environments

Search in complex environment

- So far, the simplest environment:
  - Episodic, single agent, fully observable, deterministic, static, discrete, and known.
  - A solution is a sequence of actions.
- We relax the simplifying assumptions, to get closer to the real world.
  - Finding a good state without worrying about the path to get there, covering both discrete and continuous states.
    - Local search (in discrete and continuous states)
  - Relaxing determinism to nondeterministic states.
    - Conditional plan.
  - Partial observability.
    - Conditional plan.
  - Unknown environment.

Outline

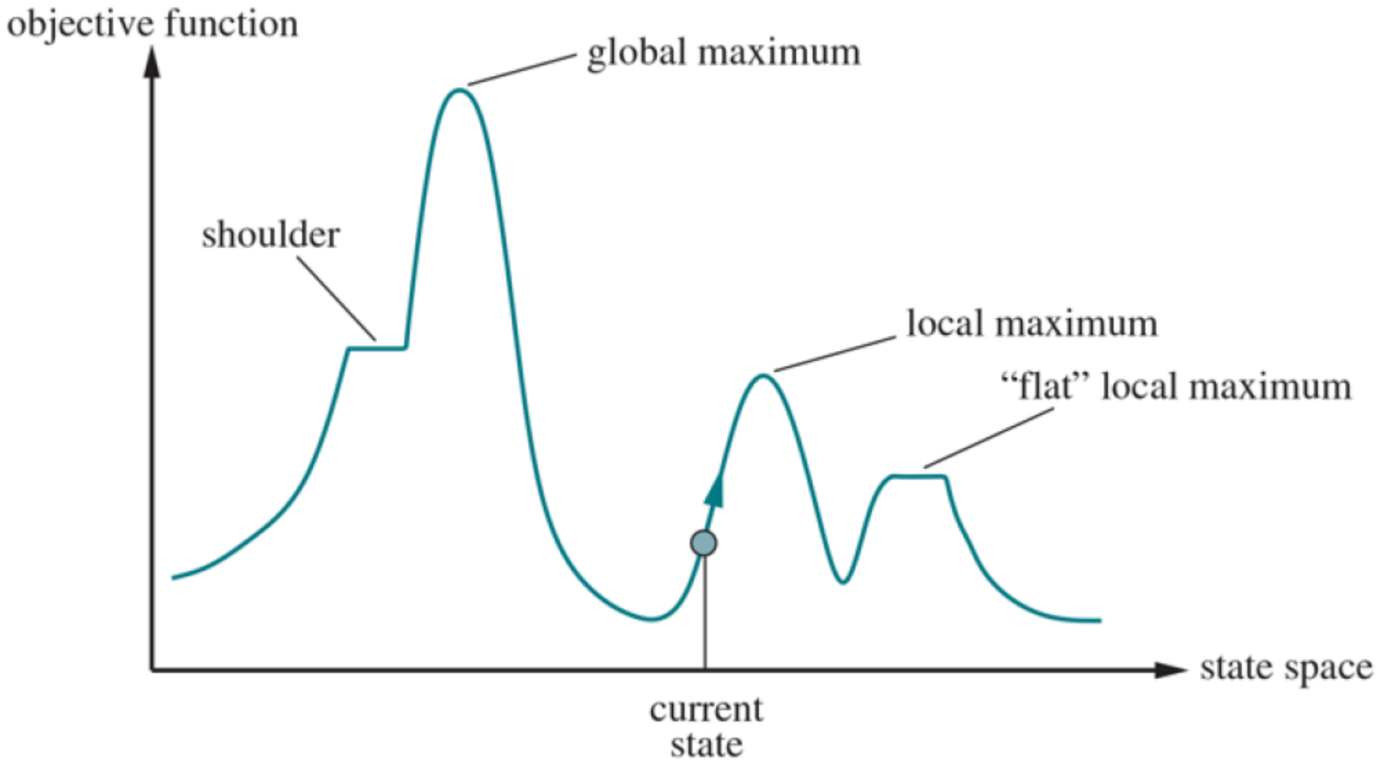
- Local search in continuous spaces
- Search with nondeterministic actions
- Search in partially observable environments
    - Sensorless problems
    - Partially observable environments
- Online search agents and unknown environments

Local search

- Local search algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, nor the set of states that have been reached.
  - Use very little memory
  - Find reasonable solutions in large or infinite state spaces
- Local search can also solve optimization problems, in which the aim is to find the best state according to an objective function.

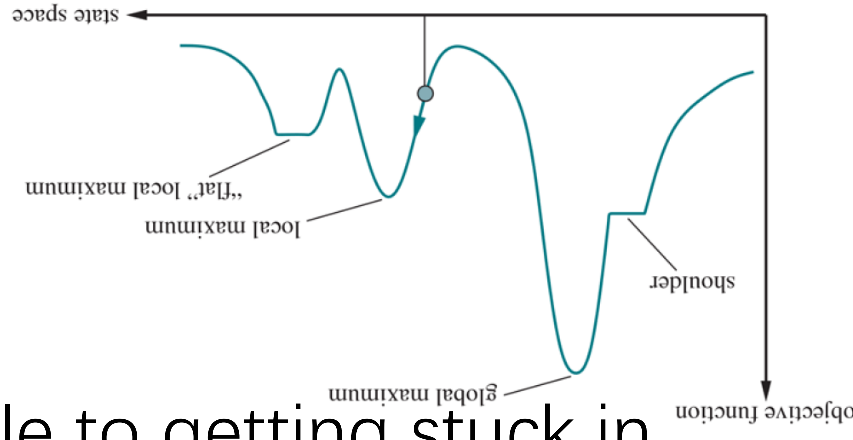
Local search

- Each state has an elevation, defined by the value of the objective function.
- The aim is to find the state with the highest elevation.
- Hill climbing, never climbs down.
- Problems:
  - Local maxima
  - Plateaus
  - Ridges
- Simulated annealing



A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum.

Simulated Annealing (SA)



- A HC neve makes downhill moves, is vulnerable to getting stuck in a local maximum.
- A purely random walk will eventually stumble upon the global maximum, but very inefficient
- Combining these two
  - Instead of picking the best move, SA picks a random move.
  - If the move improves the situation, it is always accepted.
  - Otherwise, SA accepts the move with some probability less than 1.
    - Badness of the move. Worse the move, smaller the probability.
    - Probability goes down as temperature goes down.

Local beam search

- Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations.
- The local beam search algorithm keeps track of k states rather than just one.
  - It begins with k randomly generated states. At each step, all the successors of all states are generated.
  - If any one is a goal, the algorithm halts.
  - Otherwise, it selects the best k successors from the complete list and repeats.

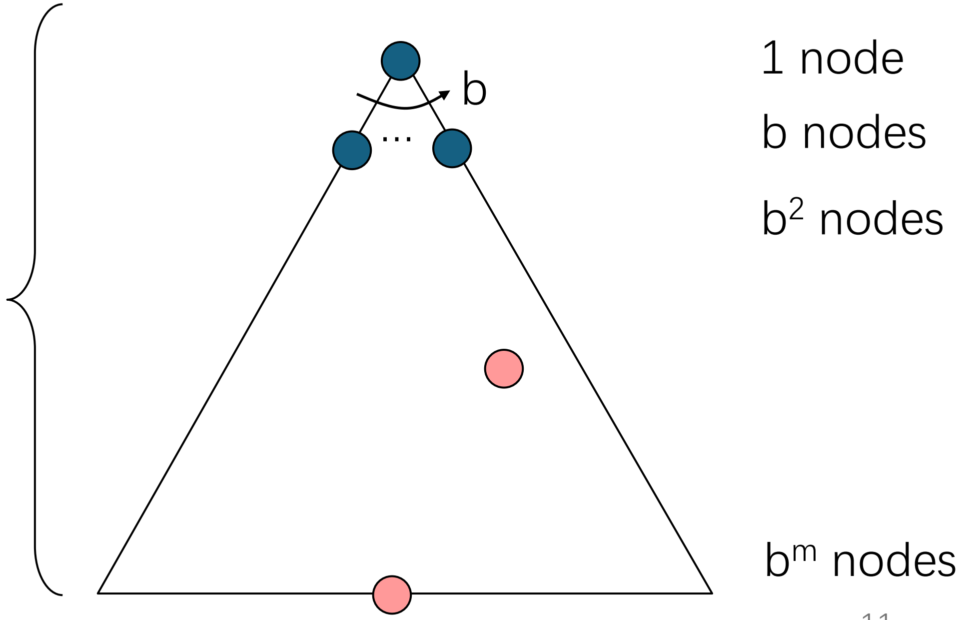
Evolutionary algorithms

- Motivated by the natural selection in biology
- There is a population of individuals (states)
- The fittest (highest value) individuals produce offspring (successor states) that populate the next generation

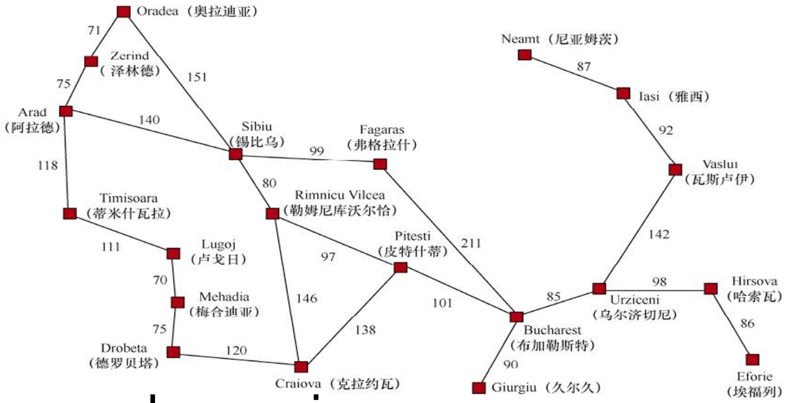
- **The mixing number**, ρ, which is the number of parents that come together to form offspring.
- **Selection**: selecting the individuals who will become the parents of the next generation
- **Recombination**: randomly select a crossover point to split each of the parent strings and recombine the parts to form children.
- **Mutation rate:** how often offspring have random mutations.

Local search in continuous spaces

- A continuous action space has an infinite branching factor, and thus cannot be handled by most of the algorithms we have covered so far.
  - Branching factor is an important parameter in search
  - Cartoon of search tree:
    - b is the branching factor
    - m is the maximum depth
    - solutions at various depths
  - Number of nodes in entire tree?
    - $1 + b + b^2 + \cdots + b^m = O(b^m)$
  - If b is too large, not good
- Mini quiz:
  - Is Hill Climbing good for continuous spaces?



Example



- Suppose we want to place three new airports anywhere in Romania, such that the sum of squared straight-line distances from each city on the map to its nearest airport is minimized.
- The state space: the locations of the three new airports
  - $(x_1, y_1), (x_2, y_2), (x_3, y_3)$
  - Six variables $\mathbf{x}=< x_1, y_1, x_2, y_2, x_3, y_3 >$
- Objective function $f(\mathbf{x})=f(x_1, y_1, x_2, y_2, x_3, y_3)$
  - C_i is the set of cites whose closest airport is airport i.

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

Discretize

- Instead of allowing the (x_i, y_i) locations to be any point in continuous two-dimensional space, we could limit them to fixed points on a rectangular grid with spacing of size σ
  - Each state would have only 12 successors
  - $x_i+\sigma$, $x_i-\sigma$, $y_i+\sigma$, $y_i-\sigma$, where $i = 1, 2, 3$
  - Any local search algorithm can be applied in this discrete space.

Using the gradients

- We often have an objective function expressed in a mathematical form such that we can use calculus to solve the problem analytically rather than empirically.
- Many methods attempt to use the gradient of the landscape to find a maximum.
- The gradient of the objective function is a vector ∇f that gives the magnitude and direction of the steepest slope

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right) .$$

- Mini quiz: placing only one airport (x, y), and we have N cities with coordinates (a_i, b_i) where i=1⋯N.
  - Sum of the squared straight-line distances from each city on the map to the new airport is minimized
  - What is the optimal location of the airport

Steepest ascent hill climbing

- No solution in closed form
- Three airports: gradient depends on what cities are closest to each airport
- Given a locally correct gradient, we use steepest ascent hill climbing
- Compute the gradient

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right) .$$

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c) .$$

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x}) ,$$

- Use a small constant step size α to climb the hill
  - Adjusting α
  - Too small: too many steps are needed, slow
  - Too large: overshoot the maximum

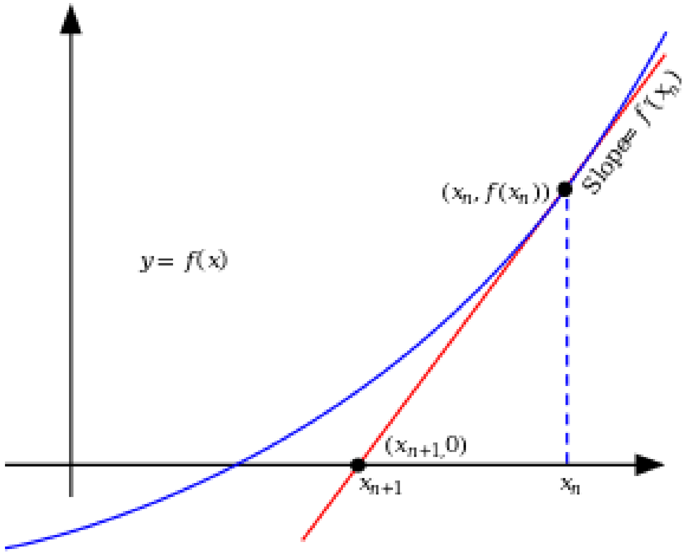
Line search

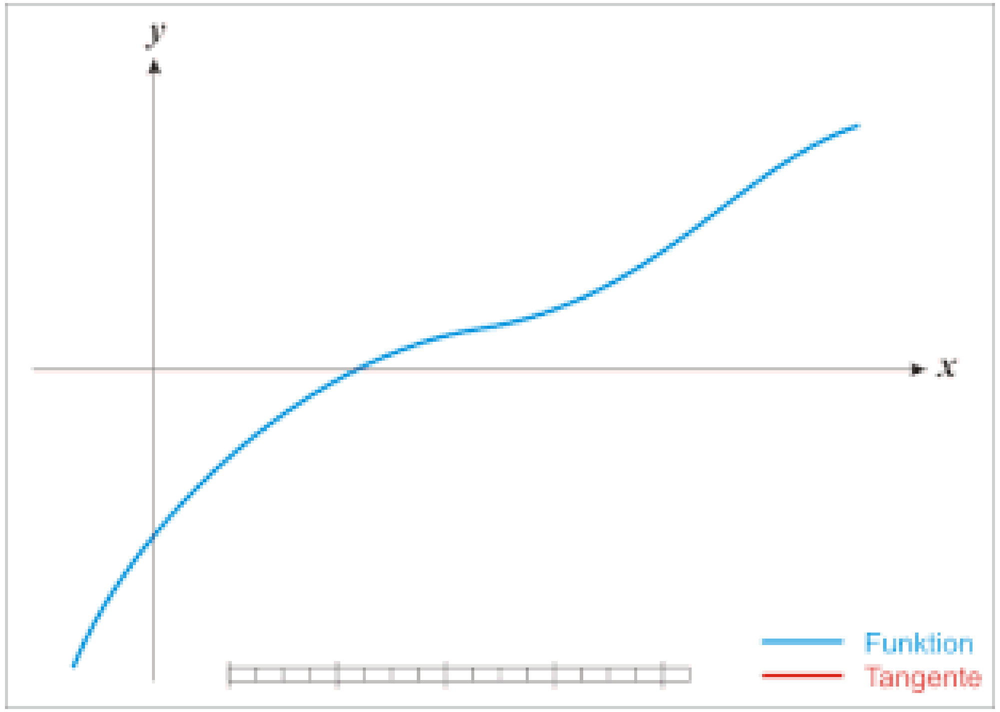
- To overcome this dilemma by extending the current gradient direction, usually by repeatedly doubling α, until f starts to decrease again.
- The point at which this occurs becomes the new current state.

Newton-Raphson method

- A general technique to find roots of functions
  - f(x)=0

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$





Newton-Raphson method

- To find a maximum or minimum of f, we need to find **x** such that the gradient of f is a zero vector.
  - $\nabla f(\mathbf{x}) = 0$

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x})\nabla f(\mathbf{x}),$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

- Hessian matrix
  - n^2 elements
  - inversion computation is expensive

Open-loop and closed-loop system

- Fully observable, deterministic, and known environment
  - The solution to any problem is a **fixed sequence of actions**
  - **Open-loop**, ignore percepts while executing breaks the loop between agent and environment.
  - Example: shortest path

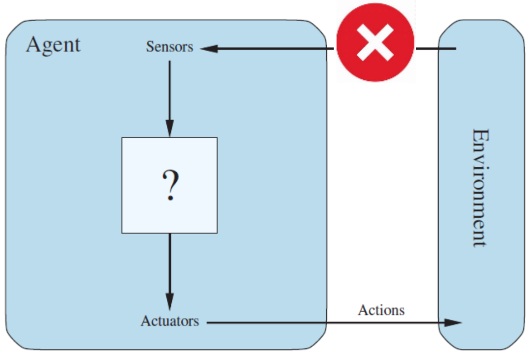
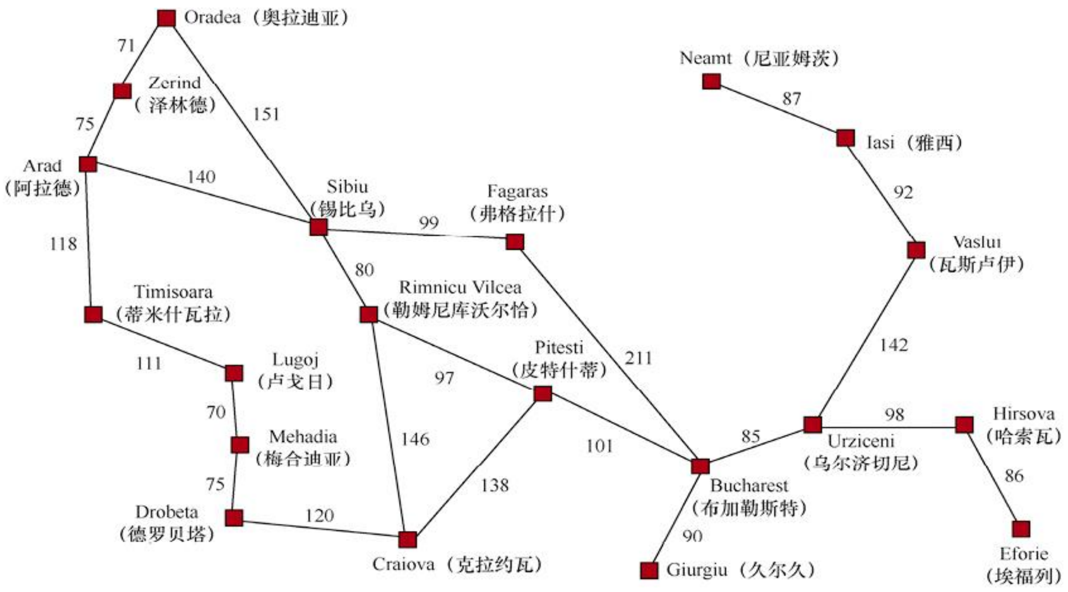


Figure 2.1 Agents interact with environments through sensors and actuators.



- Partially observable, non-deterministic
  - Actions depends on what percepts arrive
  - Example: traffic based fastest path
  - The solution is a **strategy/conditional plan /contingency plan**
    - Different future actions based on percepts
  - **Closed-loop**

Search with Nondeterminism

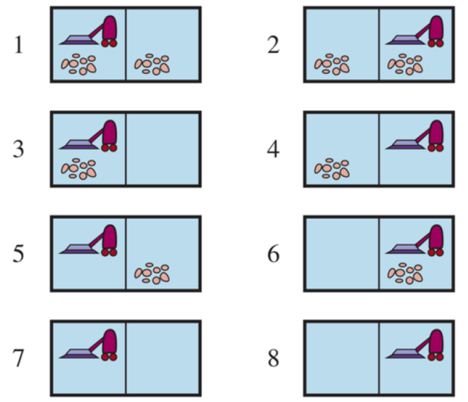
- When environment is partially observable
  - The agent doesn't know for sure which state it is in
- When the environment is nondeterministic
  - The agent doesn't know what state it transitions to after taking an action

- I'm in state s_1 and if I do action a, I'll end up in state s_1
- I'm either in state s_1 or s_2, and if I do action a, I'll end up in state s_3 , s_4 or s_5.

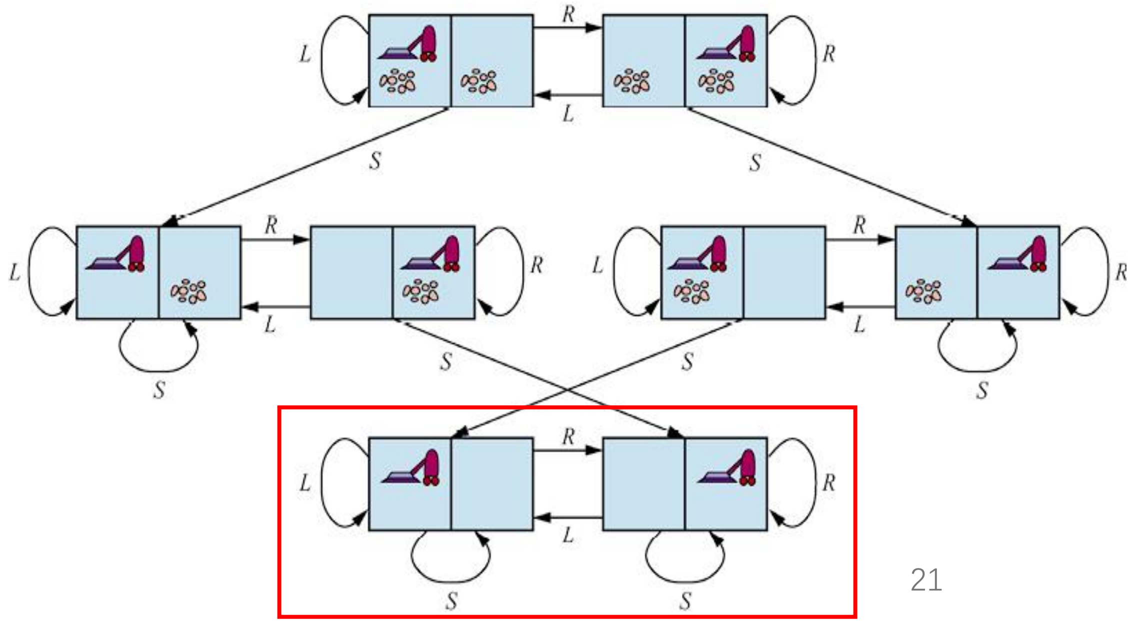
- A set of physical states that the agent believes are possible a **belief state**.

Search with nondeterministic actions

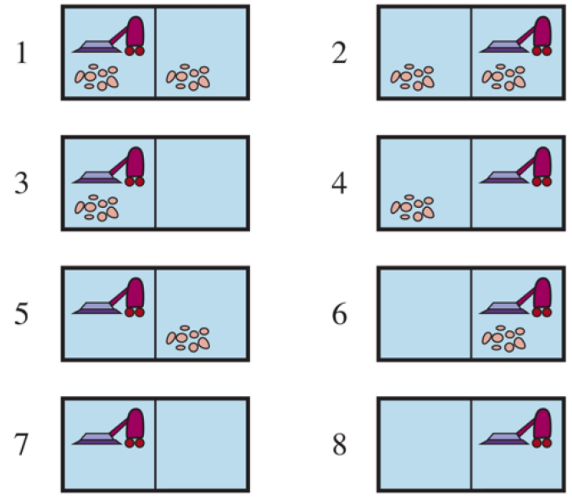
- The vacuum world
  - From state 1:
    - Solution: an action sequence: S, R, S
- The erratic vacuum world, with a nondeterministic suck action
  - When applied to a dirty square, the action cleans the square and sometimes cleans up dirt in an adjacent square, too.
  - When applied to a clean square the action sometimes deposits dirt on the carpet.
  - Results(1, suck)={5, 7}
  - Results(4, suck)={4, 2}



The eight possible states of the vacuum world; states 7 and 8 are goal states.



Conditional plan



The eight possible states of the vacuum world; states 7 and 8 are goal states.

Results(1, suck)={5, 7}

- If we start in state 1, no single sequence of actions solves the problem, but the following conditional plan does.

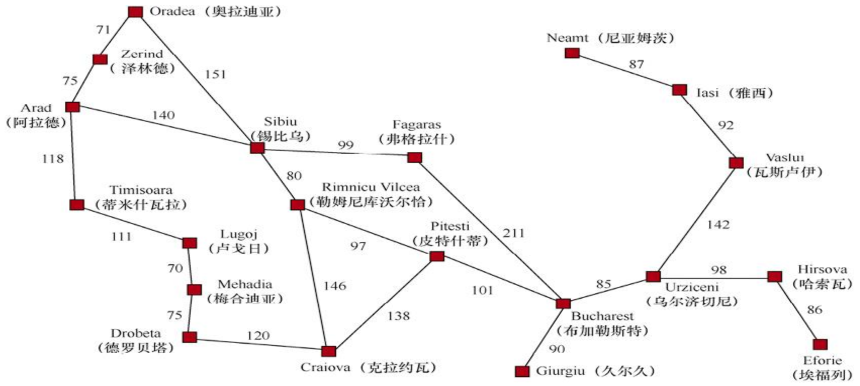
$$[Suck, \textbf{if } State = 5 \textbf{ then } [Right, Suck] \textbf{ else } []] \,.$$

- Solutions are trees rather than sequences
- If-then-else
  - If statement tests to see what the current **state** is; this is something the agent will be able to observe at runtime, but doesn't know at planning time.
  - Alternatively, tests the **percept** rather than state
  - Closed-loop system
- How can we find a conditional plan?

AND-OR search trees

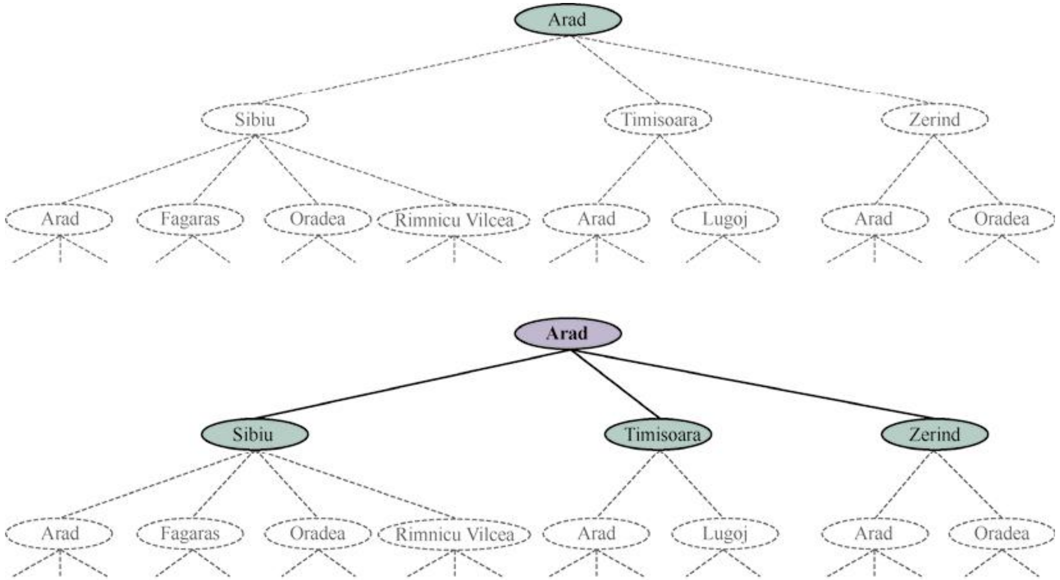
- OR nodes
  - The nodes that we have seen in search trees in deterministic environments.
  - Branching is introduced by the agent's own choices in each state.

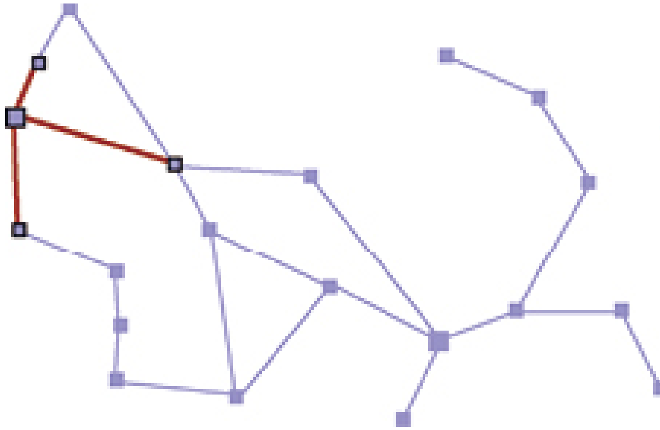
Search algorithms



- Input: a search problem
- Output: a solution, or an indication of failure (no solution exists)

- Algorithms that superimpose a **search tree** over the state-space graph
  - Nodes: states
  - Edges: actions
  - Root: initial state



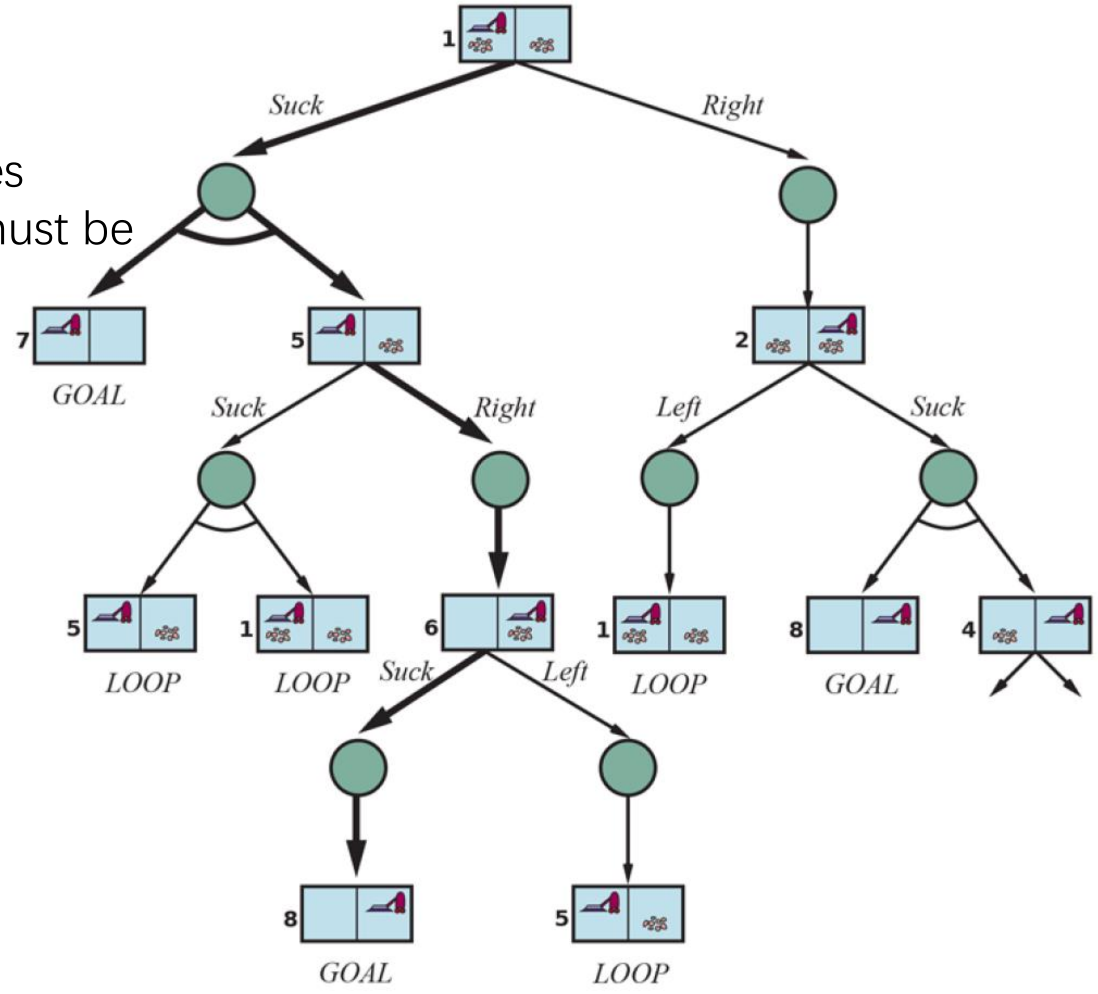


AND-OR search trees

- OR nodes
  - The nodes that we have seen in search trees in deterministic environments.
  - Branching is introduced by the agent's own choices in each state.
- AND nodes
  - Branching is also introduced by the environment's choice of outcome for each action
- Two kinds of nodes alternate, leading to an AND-OR tree

OR Node, rectangles
Some action must be chosen

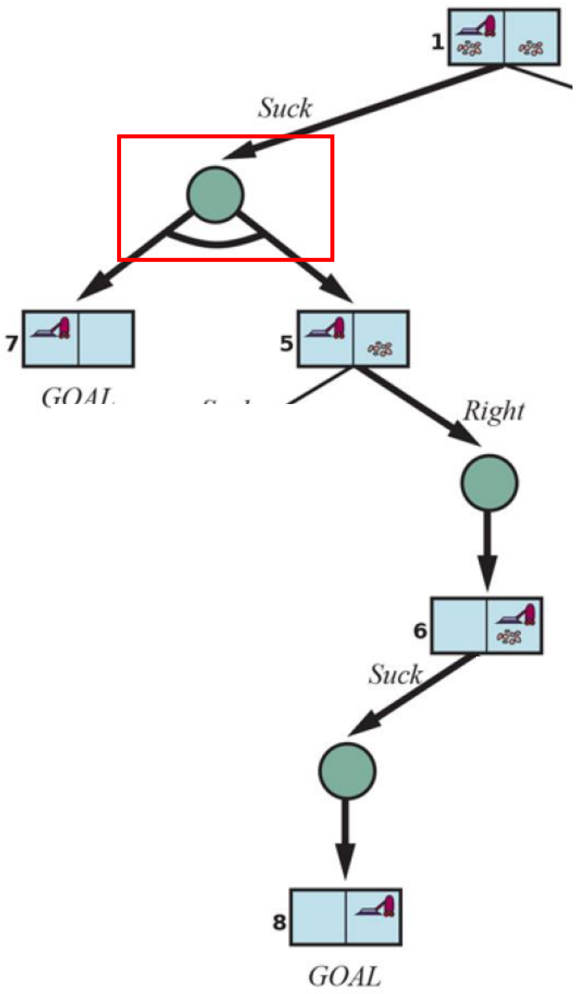
AND Node, circles
Every outcome must be handled



The first two levels of the search tree for the erratic vacuum world. State nodes are OR nodes where some action must be chosen. At the AND nodes, shown as circles, every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution found is shown in bold lines.

AND-OR search trees

- A solution for an AND-OR search problem is a subtree of the complete search tree that
  - Has a goal node at every leaf
  - Specifies one action at each of its OR nodes
  - Includes every outcome branch at each of its AND nodes



The first two levels of the search tree for the erratic vacuum world. State nodes are OR nodes where some action must be chosen. At the AND nodes, shown as circles, every outcome must be handled, as indicated by the arc linking the outgoing branches. The solution found is shown in bold lines.

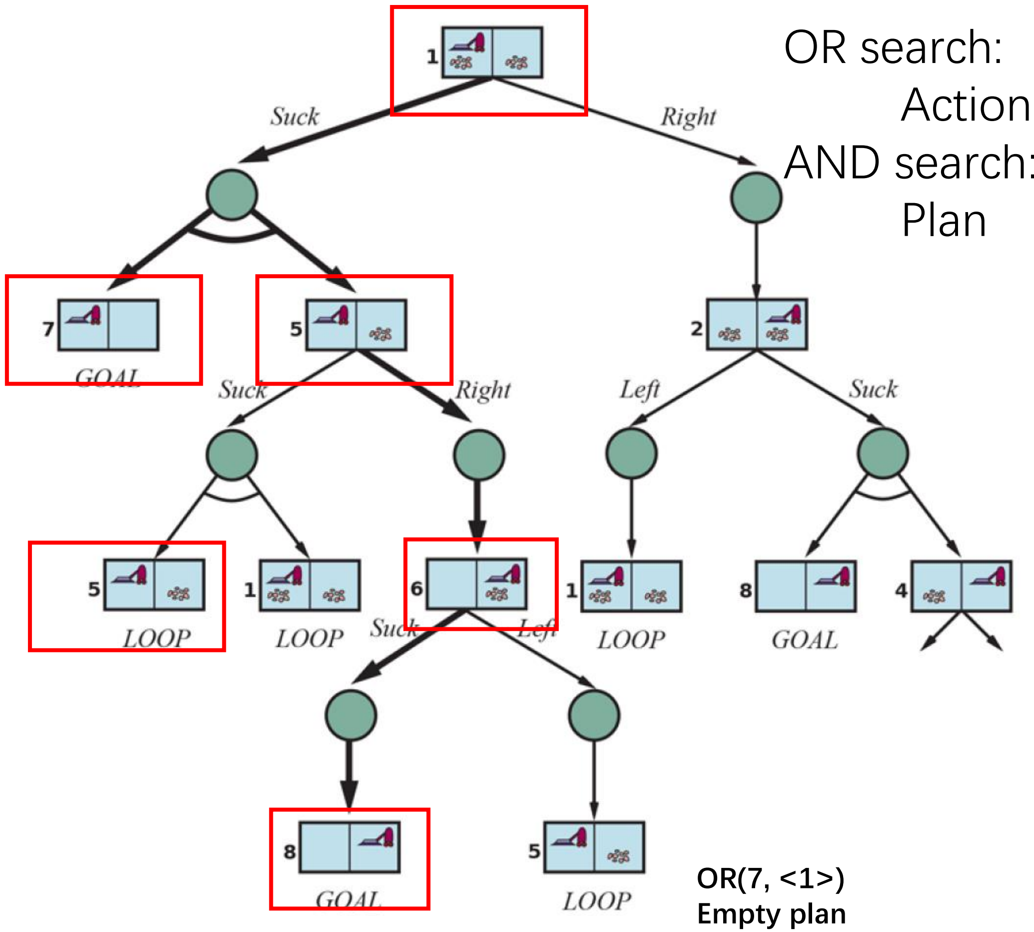
AND-OR graph search

- Depth-first
- Deal with cycles: if the current state is identical to a state on the path from the root, it returns a failure.
- OR search
  - Action + plan
- AND search
  - Plan

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

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

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(problem, s_i, 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]
```



OR search:

Action + plan

AND search:

Plan

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

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

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
   for each s_i in states do
      plan_i ← OR-SEARCH(problem, s_i, 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]
```

OR-S(1, NULL)
Suck ···
Right ···

Suck

AND-S({7, 5}, <1>)
If 7 ···
Else if 5 ···

OR(7, <1>)
Empty plan

OR(5, <1>)
Suck
Right

Suck

AND-S({5, 1}, <5, 1>)
If 5 ···
Else if 1 ···

OR(5, <5, 1>)
Failure

Right

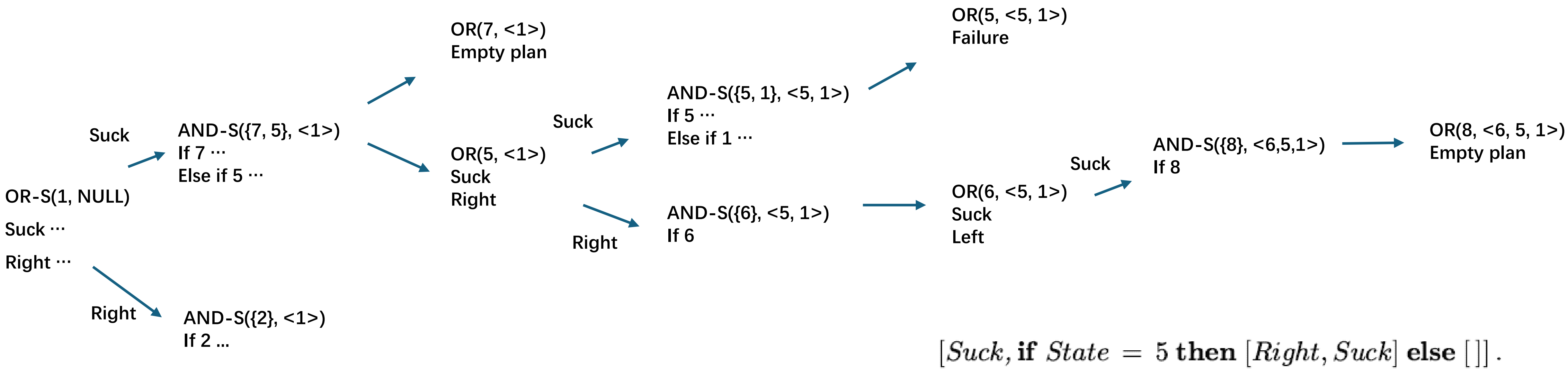
AND-S({6}, <5, 1>)
If 6

OR(6, <5, 1>)
Suck
Left

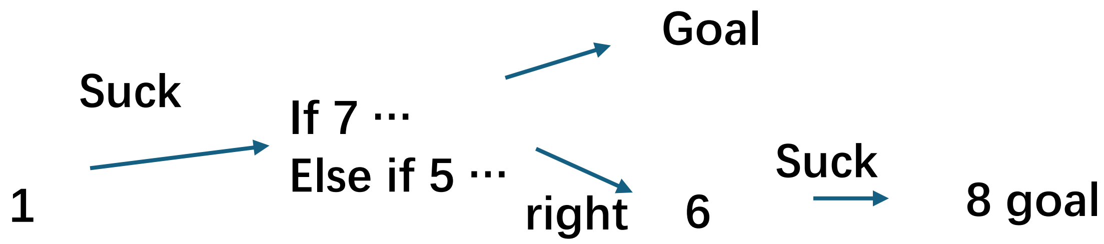
Suck

AND-S({8}, <6,5,1>)
If 8

OR(8, <6, 5, 1>)
Empty plan



$$[Suck, \textbf{if } State = 5 \textbf{ then } [Right, Suck] \textbf{ else } [\,]] .$$



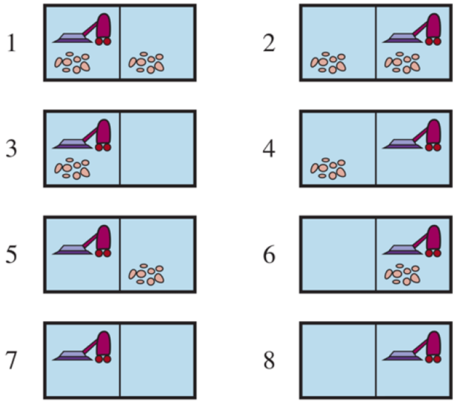
- A solution for an AND-OR search problem is a subtree of the complete search tree that
  - Has a goal node at every leaf
  - Specifies one action at each of its OR nodes
  - Includes every outcome branch at each of its AND nodes

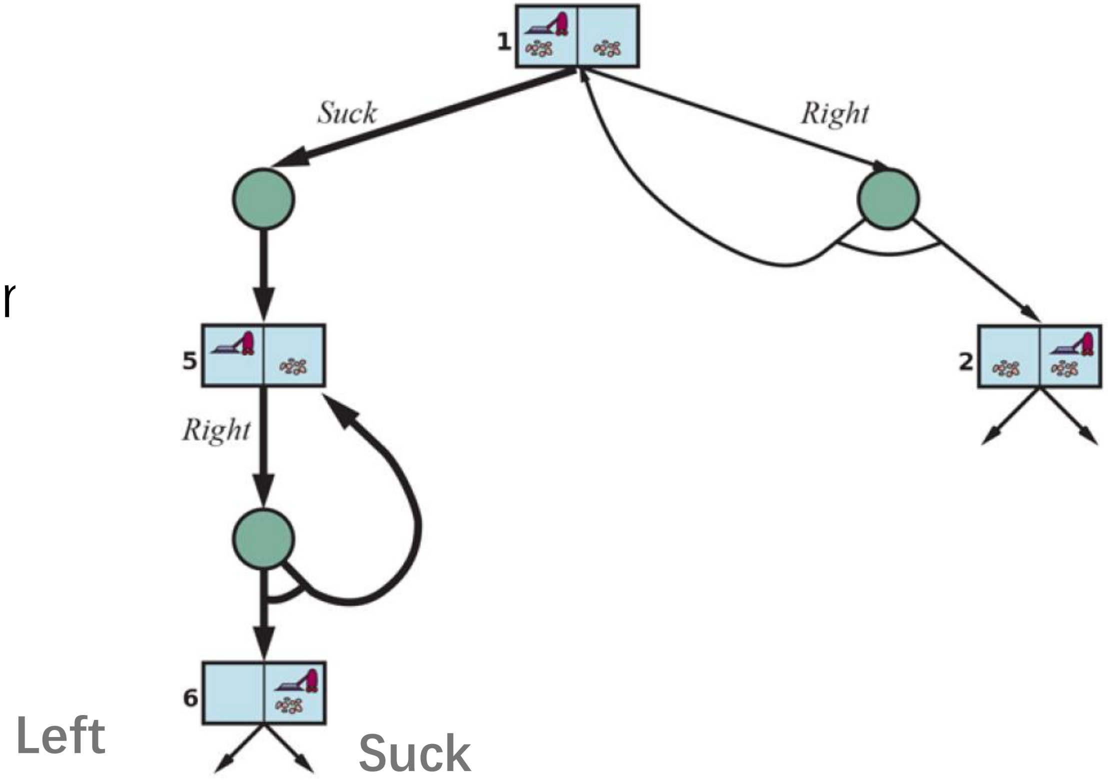
Try, try again

- Slippery vacuum world
  - Identical to the ordinary vacuum world, except
  - Movement actions sometime fail, leaving the agent in the same locatior
- There are no longer any acyclic solutions from state 1

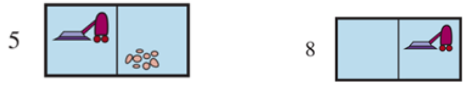
$$[\mathit{Suck}, \textbf{while } \mathit{State} = 5 \textbf{ do } \mathit{Right}, \mathit{Suck}]$$

Results(1, Right)={1, 2}





Part of the search graph for a slippery vacuum world, where we have shown (some) cycles explicitly. All solutions for this problem are cyclic plans because there is no way to move reliably.



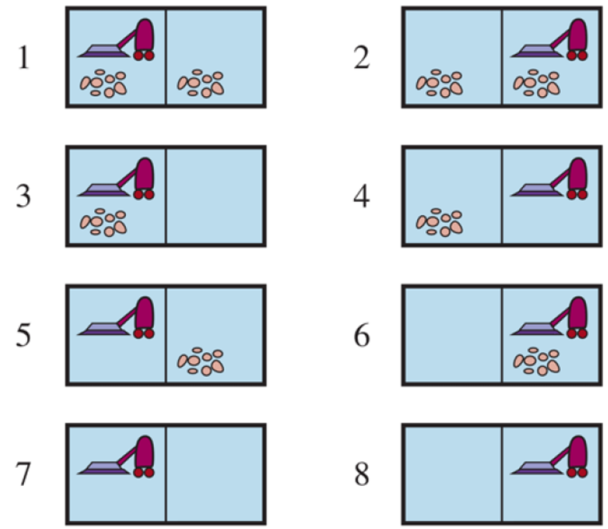
Search in Partially Observable Environments

- The agent's percepts are not enough to pin down the exact state.
  - No observation at all, sensorless problems
  - Partially observable environments

- Search with no observation
  - doctors often prescribe a broad-spectrum antibiotic rather than using the conditional plan of doing a blood test, then waiting for the results to come back, and then prescribing a more specific antibiotic. The sensorless plan saves time and money, and avoids the risk of the infection worsening before the test results are available.

Sensorless of (deterministic) vacuum world

- Assume that the agent knows the geography of its world, but not its ow location or the distribution of dirt.
- Initial belief state {1, 2, 3, 4, 5, 6, 7, 8}
- Right: {2, 4, 6, 8}
    - the agent has gained information without perceiving anything
- Suck: {4, 8}
- Left: {3, 7}
- Suck: {7}



The eight possible states of the vacuum world; states 7 and 8 are goal states.

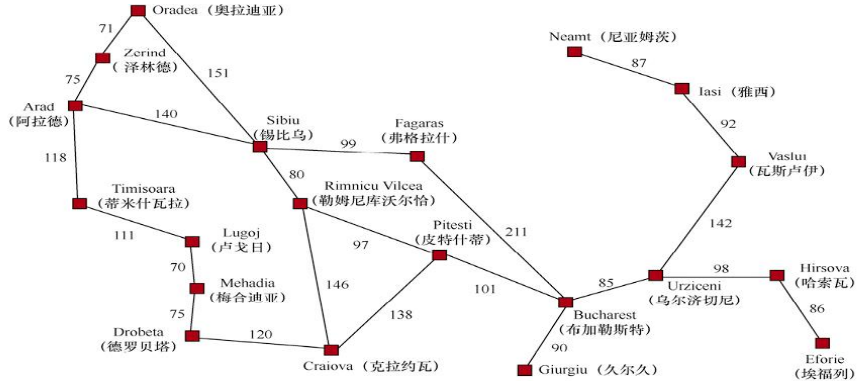
- [Right, suck, left, suck] is guaranteed to reach the goal state 7, no matter what the start state.
- the agent can coerce the world into state 7.

Search for sensorless problems

- The solution to a sensorless problem is a sequence of actions, not a conditional plan (because there is no perceiving).

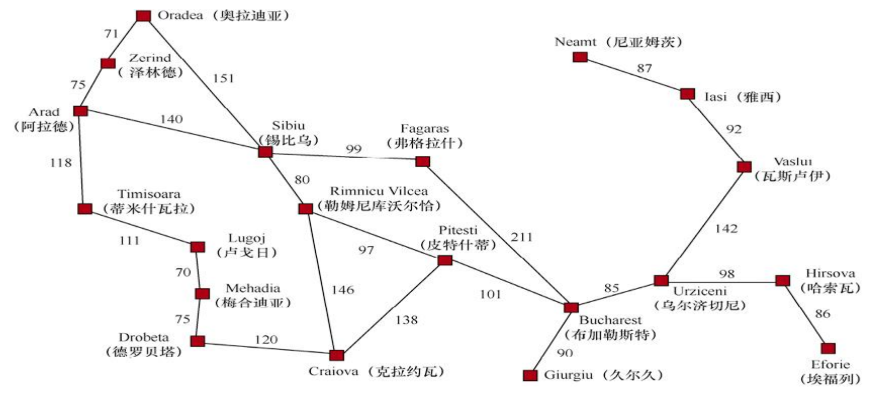
- We search in the space of belief states rather than physical states. In belief-state space, the problem is fully observable because the agent always knows its own belief state.

Search problems from lec 2



- Original problem P
- State space
  - A set of possible states that the environment can be in, say N states.
- Initial state
  - Where the agent starts.
  - E.g., Arab
- Goal states
  - One goal vs. many goals
  - E.g., Bucharest vs. no dirt in any location (vacuum-cleaner)
  - Is-Goal method

Search problems from lec 2



- Actions
  - The actions available to the agent at state s.
  - Actions(s) returns a finite set of actions that can be executed in state s.
  - Action(Arab) = {ToSibiu, ToTimisoara, ToZerind}
- Transition model
  - Result(s, a): returns the state from doing action a in state s.
  - Result(Arab, ToZerind) = Zerind
- Action cost function
  - Action-Cost(s, a, s’): numeric cost of applying action a in state s to reach a new state s’
  - Action-Cost(Arab, ToZerind, Zerind)=75

Belief-state problem

- **STATES:** The belief-state space contains every possible subset of the physical states. If P has N states, then the belief-state problem has 2^N belief states, although many of those may be unreachable from the initial state.
- **INITIAL STATE:** Typically the belief state consisting of all states in P, although in some cases the agent will have more knowledge than this.
- **ACTIONS:** This is slightly tricky. Suppose the agent is in belief state $b = \{s_1, s_2\}$, but $\text{ACTIONS}_P(s_1) \neq \text{ACTIONS}_P(s_2)$; then the agent is unsure of which actions are legal. If we assume that illegal actions have no effect on the environment, then it is safe to take the *union* of all the actions in any of the physical states in the current belief state b:

$$\text{ACTIONS}(b) = \bigcup_{s \in b} \text{ACTIONS}_P(s) .$$

On the other hand, if an illegal action might lead to catastrophe, it is safer to allow only the *intersection*, that is, the set of actions legal in *all* the states. For the vacuum world, every state has the same legal actions, so both methods give the same result.

- **TRANSITION MODEL:** For deterministic actions, the new belief state has one result state for each of the current possible states (although some result states may be the same):

$$b' = \textsc{Result}(b, a) = \{s' : s' = \textsc{Result}_P(s,a) \text{ and } s \in b\}\,.$$

(4.4)

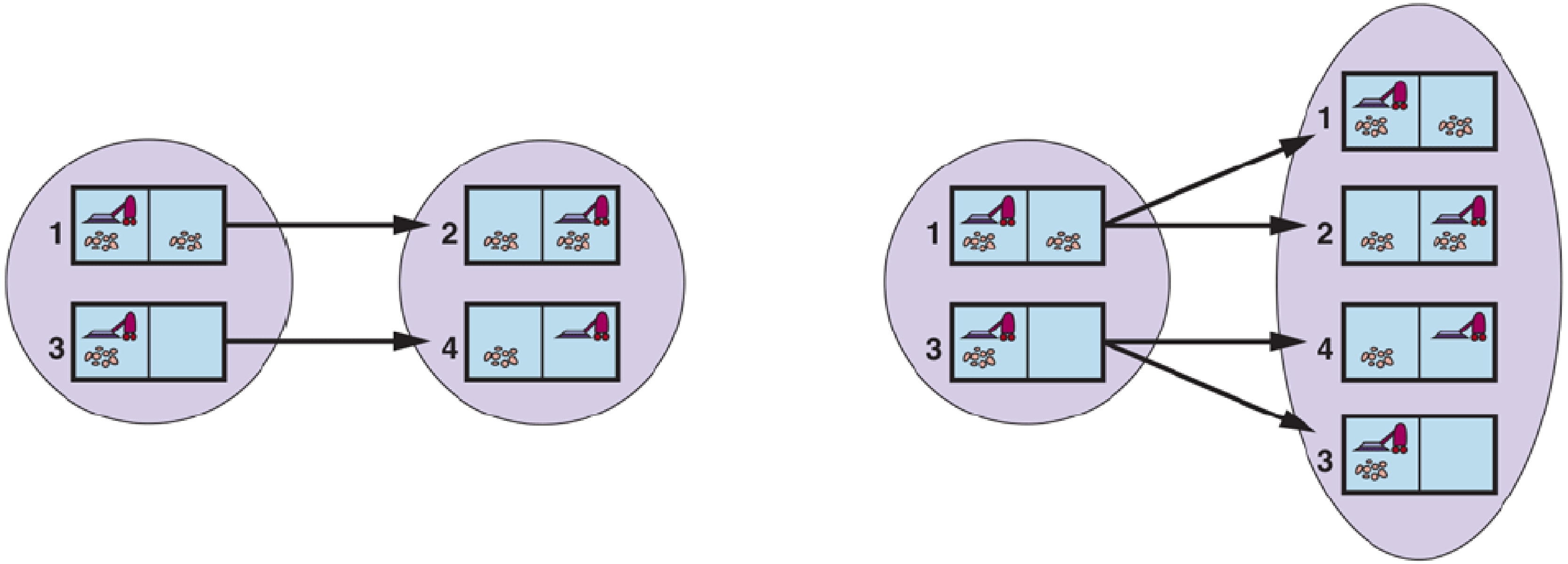
With nondeterminism, th

the action to any of the st

$b' = \textsc{Resu}$

The size of b' will be the s

larger than b with nondet



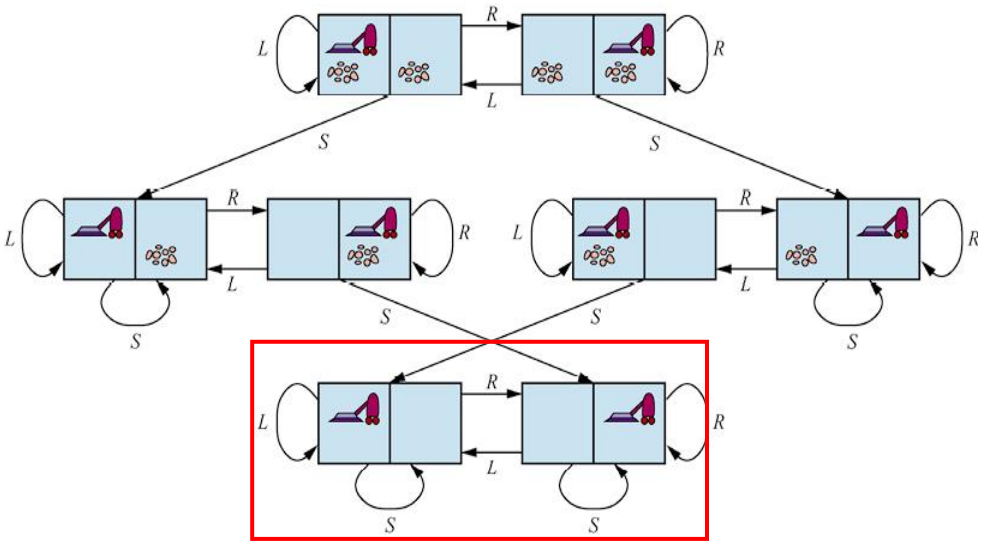
(a)

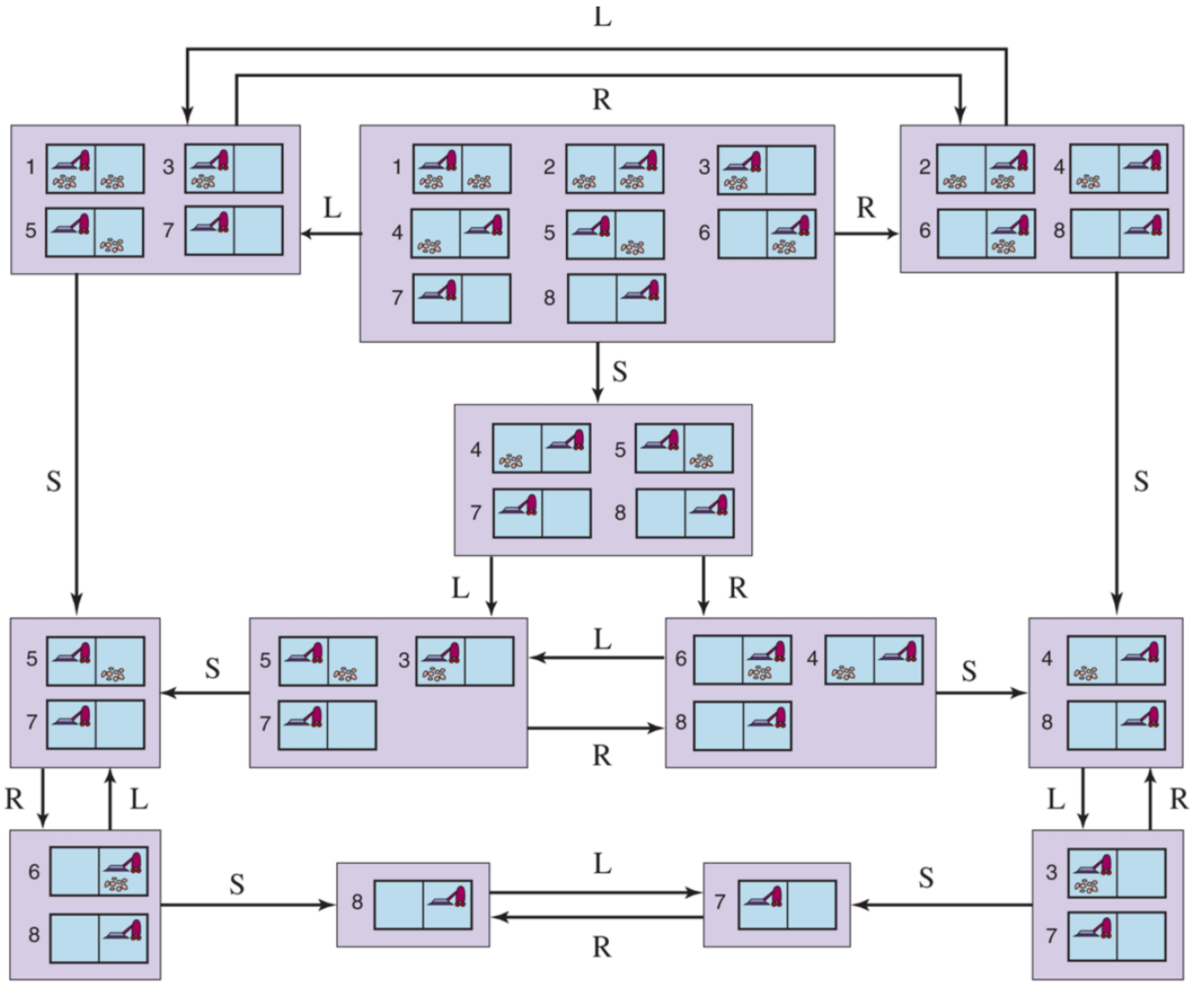
(b)

(a) Predicting the next belief state for the sensorless vacuum world with the deterministic action, *Right*. (b) Prediction for the same belief state and action in the slippery version of the sensorless vacuum world.

Reachable belief-state space for the deterministic, sensorless vacuum world

- Only 12 out of 2^8 possible belief states
- Solve it using ordinary search algorithms



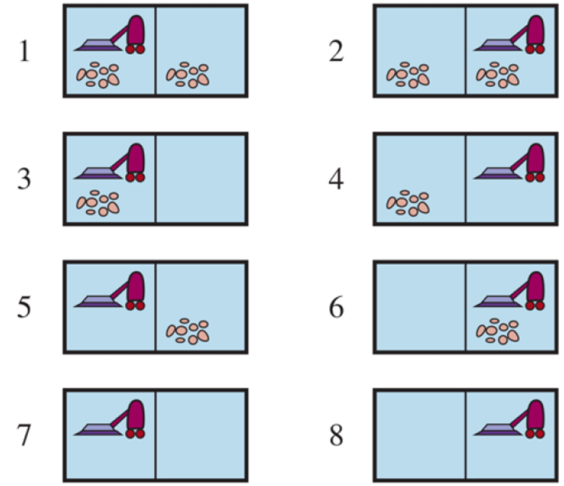


The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each rectangular box corresponds to a single belief state. At any given point, the agent has a belief state but

Outline

- Local search in continuous spaces
- Search with nondeterministic actions
- Search in partially observable environments
    - Sensorless problems
    - Partially observable environments
- Online search agents and unknown environments

Search in partially observable environments

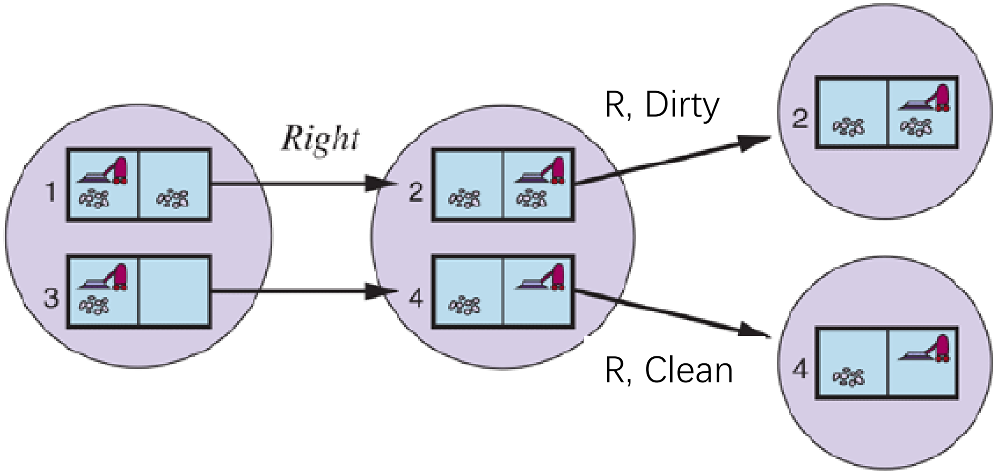


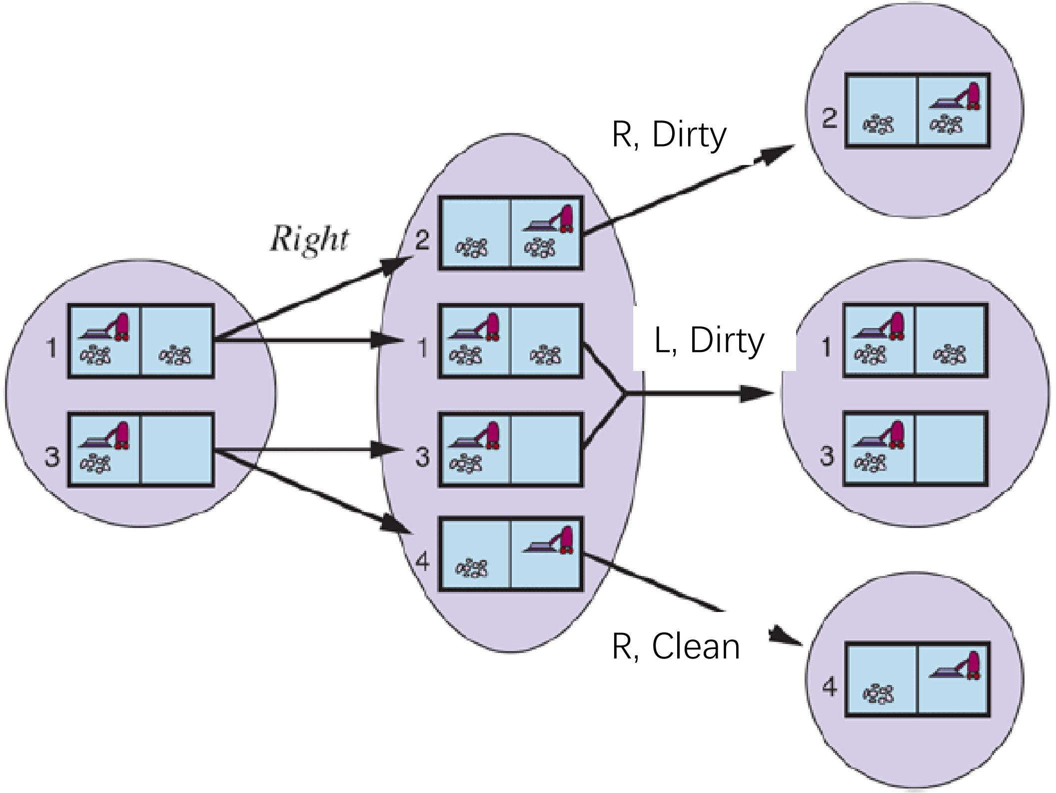
The eight possible states of the vacuum world; states 7 and 8 are goal states.

- Percept(s)
  - Returns the percept received by the agent in state s.
  - Fully observable environments: Percept(s)=s
  - Sensorless environments: Percept(s)=null
- Local-sensing vacuum world
  - A position sensor that yields L in the left square and R in the right square
  - A dirt sensor yields Dirty/Clean
- When percept is [L, Dirty], the belief state is {1, 3}

Partially-observable
Deterministic vs. nondeterministic (slippery)

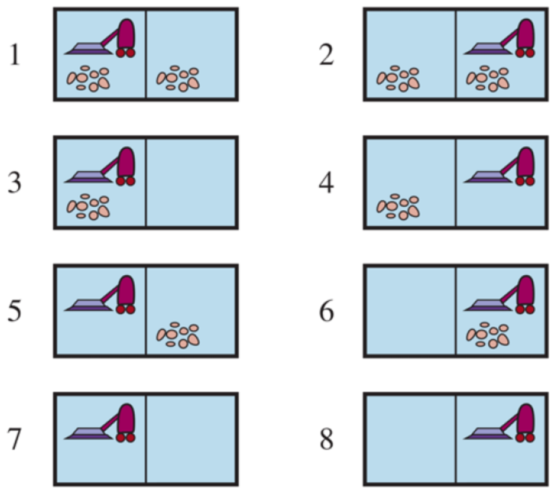
- [L, Dirty], Right





Solving partially observable problems

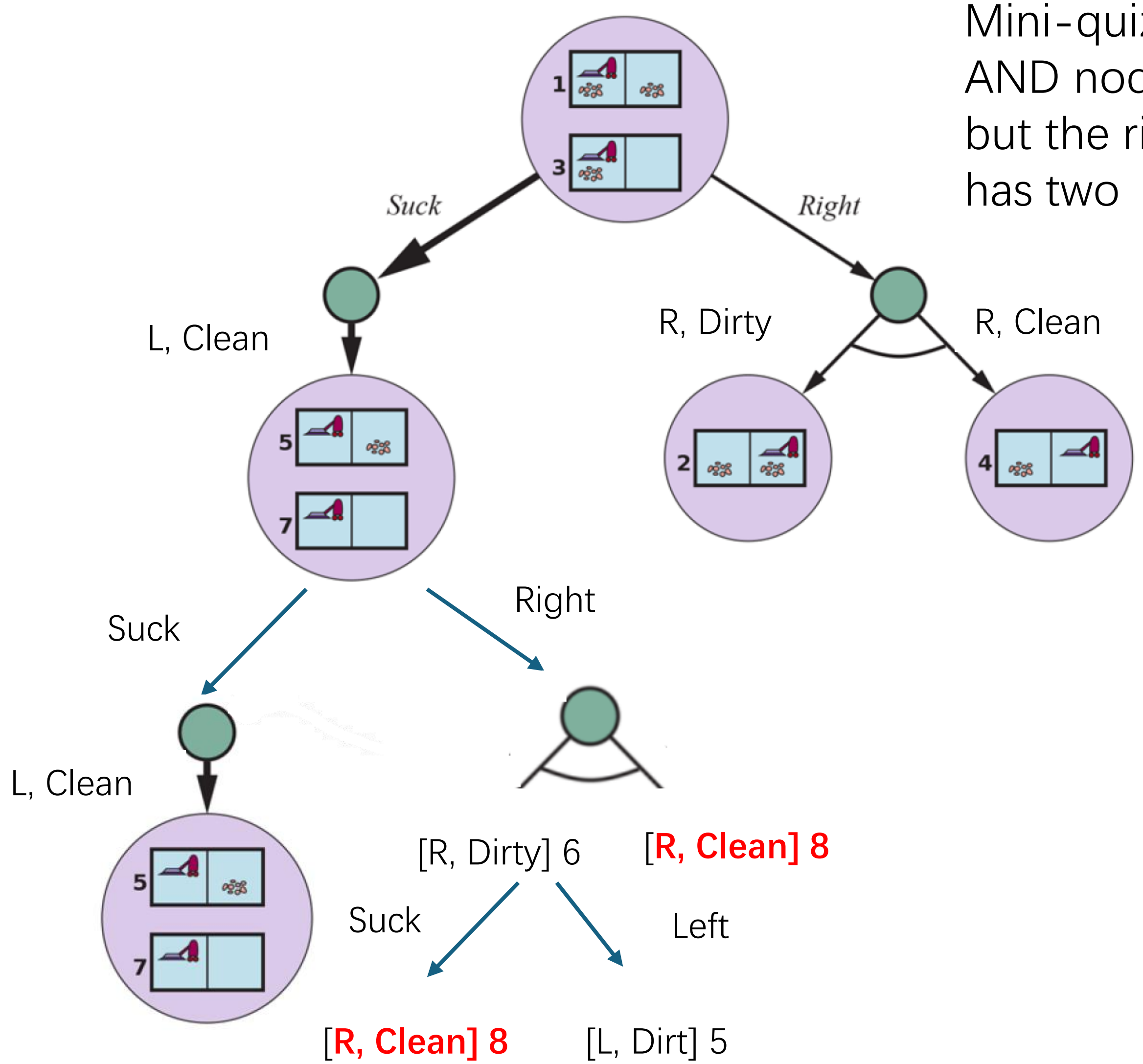
- Use AND-OR search



The eight possible states of the vacuum world; states 7 and 8 are goal states.

$[Suck, Right, \textbf{if } Bstate = \{6\} \textbf{ then } Suck \textbf{ else } [\,]]$.

Mini-quizz: why left AND node has one, but the right And node has two



Online search agents and unknown environments

- So far, offline search
  - Computer a complete solution before taking the first action
  - Either a sequence of actions or a condition plan
- Online search
  - Interleaves computation and action
  - Take an action, then observe the environment and computes the next action
  - Good for dynamic or semi-dynamic environments, where there is a penalty for sitting around and computing too long.
- Examples of online search
  - Mapping problems
  - New born babies

Online search problems

- Deterministic, fully observable environment
  - ACTIONS(s), the legal actions in state s;
  - $c(s,a,s')$, the cost of applying action a in state s to arrive at state s'. Note that this cannot be used until the agent knows that s' is the outcome.
  - IS-GOAL(s), the goal test.

- The agent cannot determine Result(s, a) except by being state s and doing action a
- Competitive ratio
  - The agent's objective is to reach a goal state while minimizing cost. The cost is the total path cost that the agent incurs as it travels.
  - The path cost the agent would incur if it knew the search space in advance—that is, the optimal path in the known environment.
  - The ratio between the two costs

Online Depth First Search

- This agent stores its map in a table, result[s, a], that records the state resulting from executing action a in state s.

function ONLINE-DFS-AGENT(*problem*, s') **returns** an action
  persistent: s, a, the previous state and action, initially null
    result, a table mapping (s, a) to s', initially empty
    untried, a table mapping s to a list of untried actions
    unbacktracked, a table mapping s to a list of states never backtracked to

  if *problem*.IS-GOAL(s') **then return** *stop*
  if s' is a new state (not in *untried*) **then** *untried*[s'] ← *problem*.ACTIONS(s')
  if s is not null **then**
    result[s, a] ← s'
    add s to the front of *unbacktracked*[s']
  if *untried*[s'] is empty **then**
    if *unbacktracked*[s'] is empty **then return** *stop*
    else a ← an action b such that *result*[s', b] = POP(*unbacktracked*[s'])
  else a ← POP(*untried*[s'])
  $s \leftarrow s'$
  return a

An online search agent that uses depth-first exploration. The agent can safely explore only in state spaces in which every action can be "undone" by some other action.

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x}),$$

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x}),$$

Lecture 4 ILOs

- Search in complex environments
  - Local search in continuous spaces
    - Steepest ascent hill climbing, Newton method
  - Search with nondeterministic actions
    - AND-OR search, conditional plan
  - Search in partially observable environments
    - Sensorless problems
      - Belief-state space, ordinary search, sequences of actions in belief-state
    - Partially observable environments
      - AND-OR search, conditional plan
  - Online search agents and unknown environments