Searching in non-deterministic, partially observable and unknown environments

CE417: Introduction to Artificial Intelligence
Sharif University of Technology
Spring 2016

Soleymani

“Artificial Intelligence: A Modern Approach”, 3rd Edition, Chapter 4

Problem types

- Deterministic and fully observable (single-state problem)
  - Agent knows exactly its state even after a sequence of actions
  - Solution is a sequence

- Non-observable or sensor-less (conformant problem)
  - Agent's percepts provide no information at all
  - Solution is a sequence

- Nondeterministic and/or partially observable (contingency problem)
  - Percepts provide new information about current state
  - Solution can be a contingency plan (tree or strategy) and not a sequence
  - Often interleave search and execution

- Unknown state space (exploration problem)

More complex than single-state problem

- Searching with nondeterministic actions
- Searching with partial observations
- Online search & unknown environment

Non-deterministic or partially observable env.

- Perception become useful
  - Partially observable
    - To narrow down the set of possible states for the agent
  - Non-deterministic
    - To show which outcome of the action has occurred

- Future percepts can not be determined in advance

- Solution is a contingency plan
  - A tree composed of nested if-then-else statements
  - What to do depending on what percepts are received

- Now, we focus on an agent design that finds a guaranteed plan before execution (not online search)

Searching with non-deterministic actions

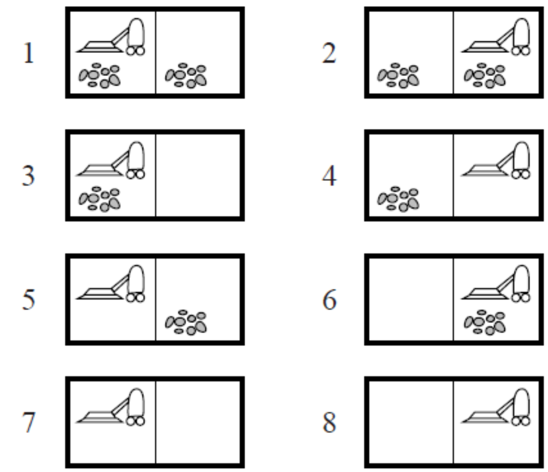
- In non-deterministic environments, the result of an action can vary.
  - Future percepts can specify which outcome has occurred.

- Generalizing the transition function
  - $RESULTS: S \times A \rightarrow 2^S$ instead of $RESULTS: S \times A \rightarrow S$

- Search tree will be an AND-OR tree.
  - Solution will be a sub-tree containing a contingency plan (nested **if-then-else** statements)

Erratic vacuum world

- States
  - {1, 2, …, 8}
- Actions
  - {Left, Right, Suck}
- Goal
  - {7} or {8}

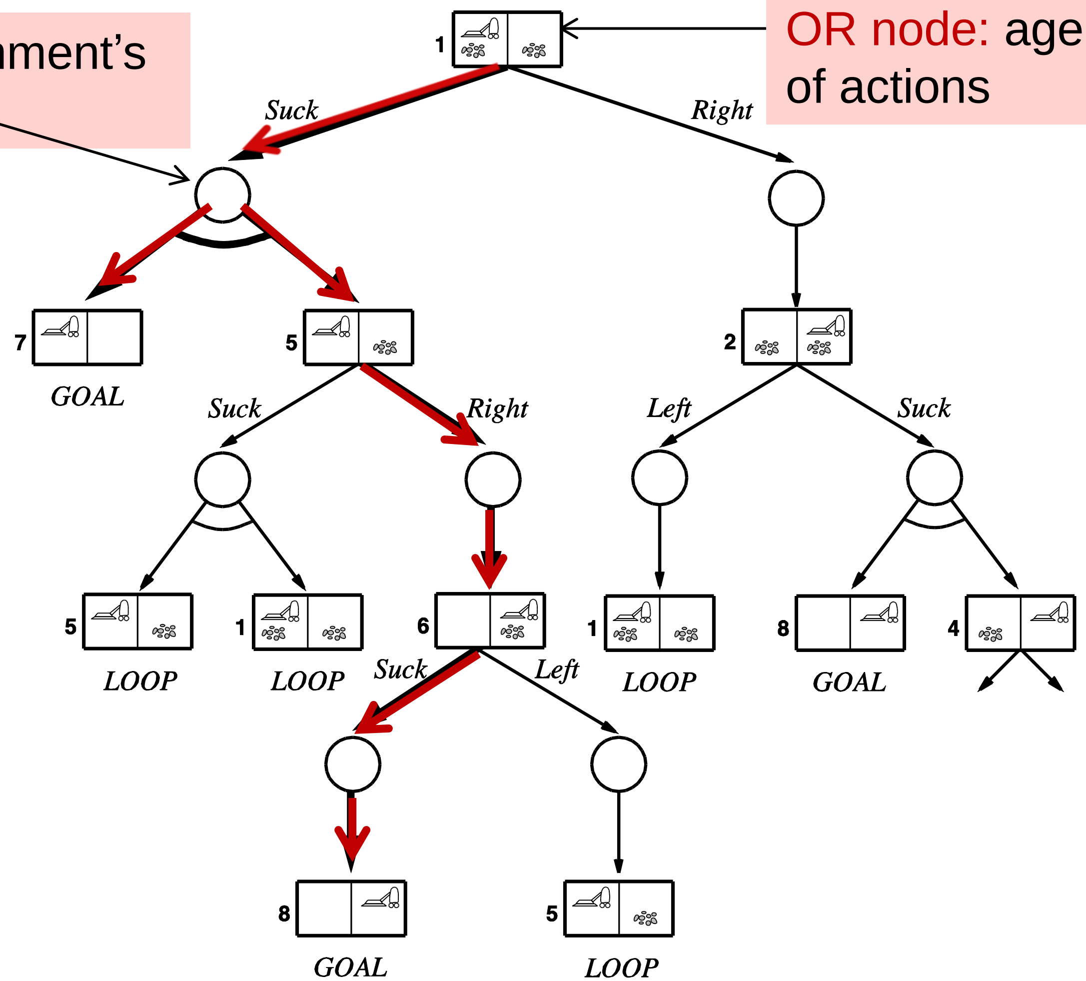


- Non-deterministic:
  - When sucking a dirty square, it cleans it and sometimes cleans up dirt in an adjacent square.
  - When sucking a clean square, it sometimes deposits dirt on the carpet.

AND-OR search tree

AND node: environment's choice of outcome

OR node: agent's choices of actions



[Suck, **if** State=5 **then** [Right, Suck] **else** []]

Solution to AND-OR search tree

- **Solution** for AND-OR search problem is a **sub-tree** that:
  - specifies **one action** at each **OR** node
  - includes **every outcome** at each **AND** node
  - **has a goal node at every leaf**

- Algorithms for searching AND-OR graphs
  - Depth first
  - BFS, best first, A*, …

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**
        $plan \leftarrow$ AND-SEARCH(RESULTS($state$, $action$), $problem$, [$state$ | $path$])
        if $plan \neq failure$ **then return** [$action$ | $plan$]
    return $failure$

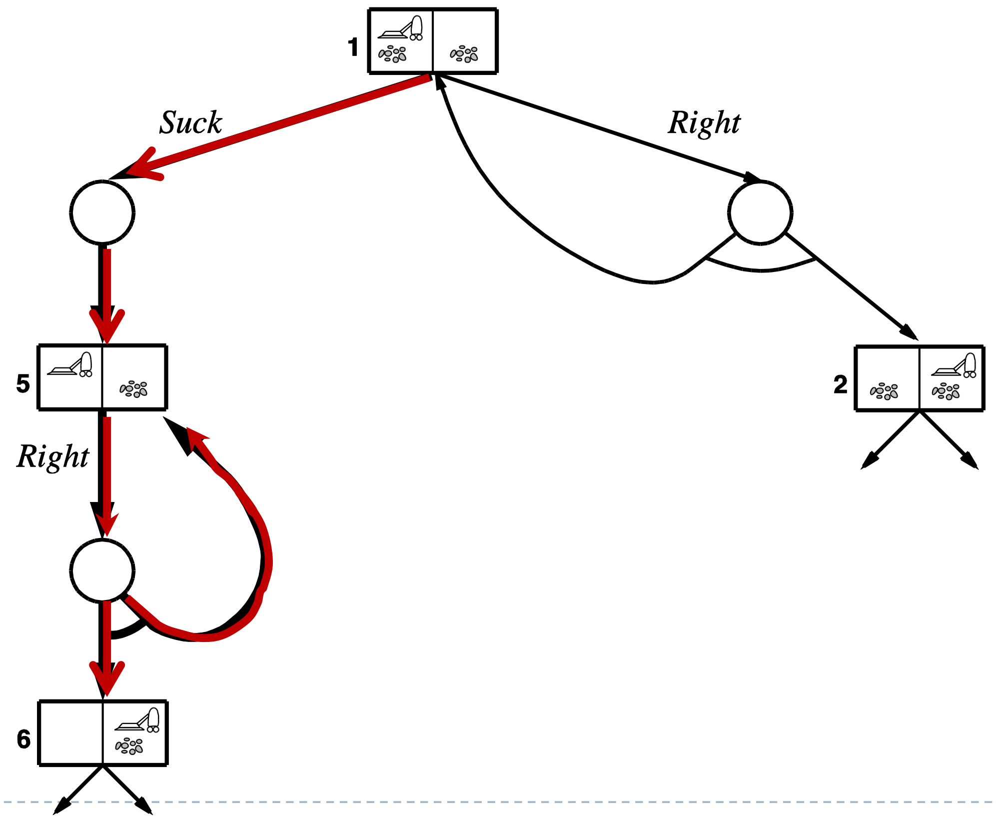
function AND-SEARCH($states$, $problem$, $path$) **returns** a conditional plan, or failure
    for each s_i in $states$ **do**
        $plan_i \leftarrow$ OR-SEARCH(s_i , $problem$, $path$)
        if $plan_i = failure$ **then return** $failure$
    return [**if** s_1 **then** $plan_1$ **else if** s_2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

AND-OR-GRAPH-SEARCH

- Cycles arise often in non-deterministic problems
  - Algorithm returns with failure when the current state is identical to one of ancestors
    - If there is a non-cyclic path, the earlier consideration of the state is sufficient
    - Termination is guaranteed in finite state spaces
      - Every path reaches a goal, a dead-end, or a repeated state

Cycles

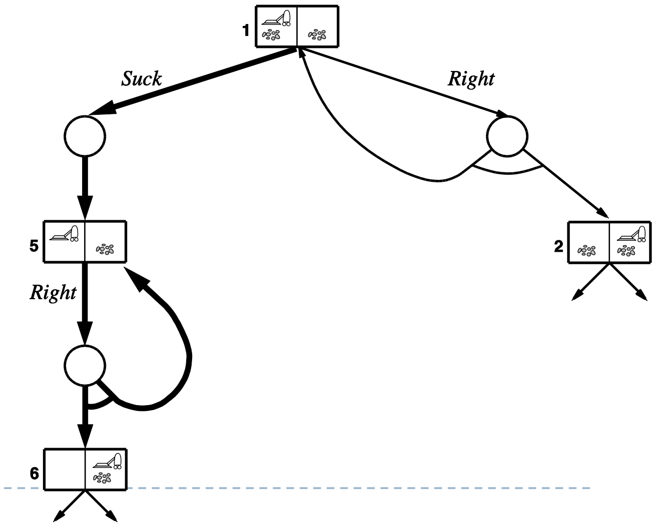
- Slippery vacuum world: Left and Right actions sometimes fail (leaving the agent in the same location)
  - No acyclic solution



Cycles solution

- Solution?
  - Cyclic plan: keep on trying an action until it works.
    - [Suck, L_1: Right, **if** state = 5 **then** L_1 **else** Suck]
      - Or equivalently [Suck, **while** state = 5 **do** Right, Suck]

- What changes are required in the algorithm to find cyclic solutions?



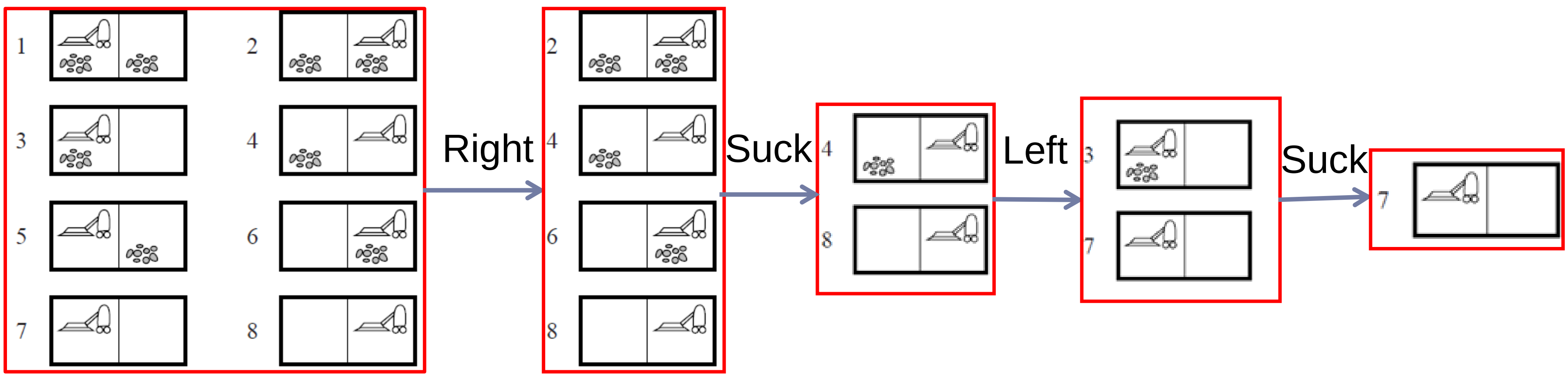
Searching with partial observations

- The agent does not always know its exact state.
  - Agent is in one of several possible states and thus an action may lead to one of several possible outcomes

- **Belief state**: agent's current belief about the possible states, given the sequence of actions and observations up to that point.

Searching with unobservable states (Sensor-less or conformant problem)

- Initial state:
  - belief = {1, 2, 3, 4, 5, 6, 7, 8}
- Action sequence (conformant plan)
  - [Right, Suck, Left, Suck]



Belief State

- Belief state space (instead of physical state space)
  - It is fully observable

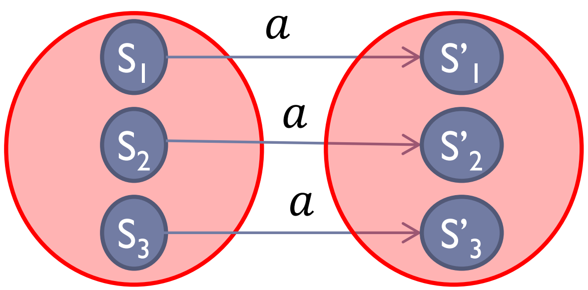
- Physical problem: N states, $ACTIONS_P$, $\mathrm{RESULTS}_P$, $\mathrm{GOAL\_TEST}_P, \mathrm{STEP\_COST}_P$
- Sensor-less problem: Up to 2^N states, $ACTIONS$, $\mathrm{RESULTS}$, $\mathrm{GOAL\_TEST}, \mathrm{STEP\_COST}$

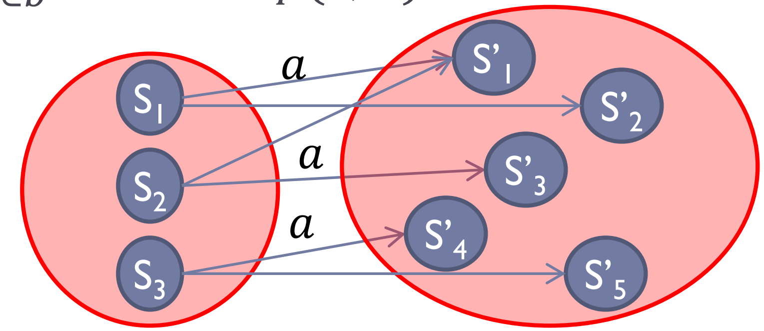
Sensor-less problem formulation (Belief-state space)

- States: every possible set of physical states, 2^N
- Initial State: usually the set of all physical states
- Actions: $ACTIONS(b) = \bigcup_{s \in b} ACTIONS_P(s)$
  - Illegal actions?! i.e., $b = \{s_1, s_2\}$, $ACTIONS_P(s_1) \neq ACTIONS_P(s_2)$
    - Illegal actions have no effect on the env. (union of physical actions)
    - Illegal actions are not legal at all (intersection of physical actions)
- Solution is a sequence of actions (even in non-deterministic environment)

Sensor-less problem formulation (Belief-state space)

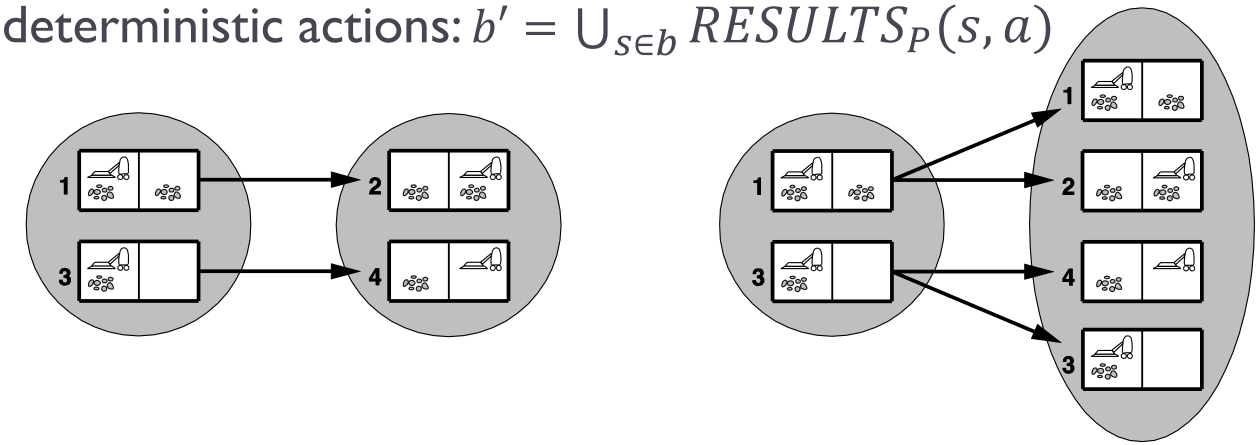
- Transposition model $(b' = PREDICT_P(b, a))$
  - Deterministic actions: $b' = \{s' \colon s' = RESULTS_P(s, a) \text{ } and \text{ } s \in b\}$
  - Nondeterministic actions: $b' = \bigcup_{s \in b} RESULTS_P(s, a)$





Sensor-less problem formulation (Belief-state space)

- Transposition model ($b' = PREDICT_P(b, a)$)
  - Deterministic actions: $b' = \{s' : s' = RESULTS_P(s, a) \text{ and } s \in b\}$
  - Nondeterministic actions: $b' = \bigcup_{s \in b} RESULTS_P(s, a)$



- Goal test: Goal is satisfied when all the physical states in the belief state satisfy $\mathrm{GOAL\_TEST}_P$.
- Step cost: $\mathrm{STEP\_COST}_P$ if the cost of an action is the same in all states

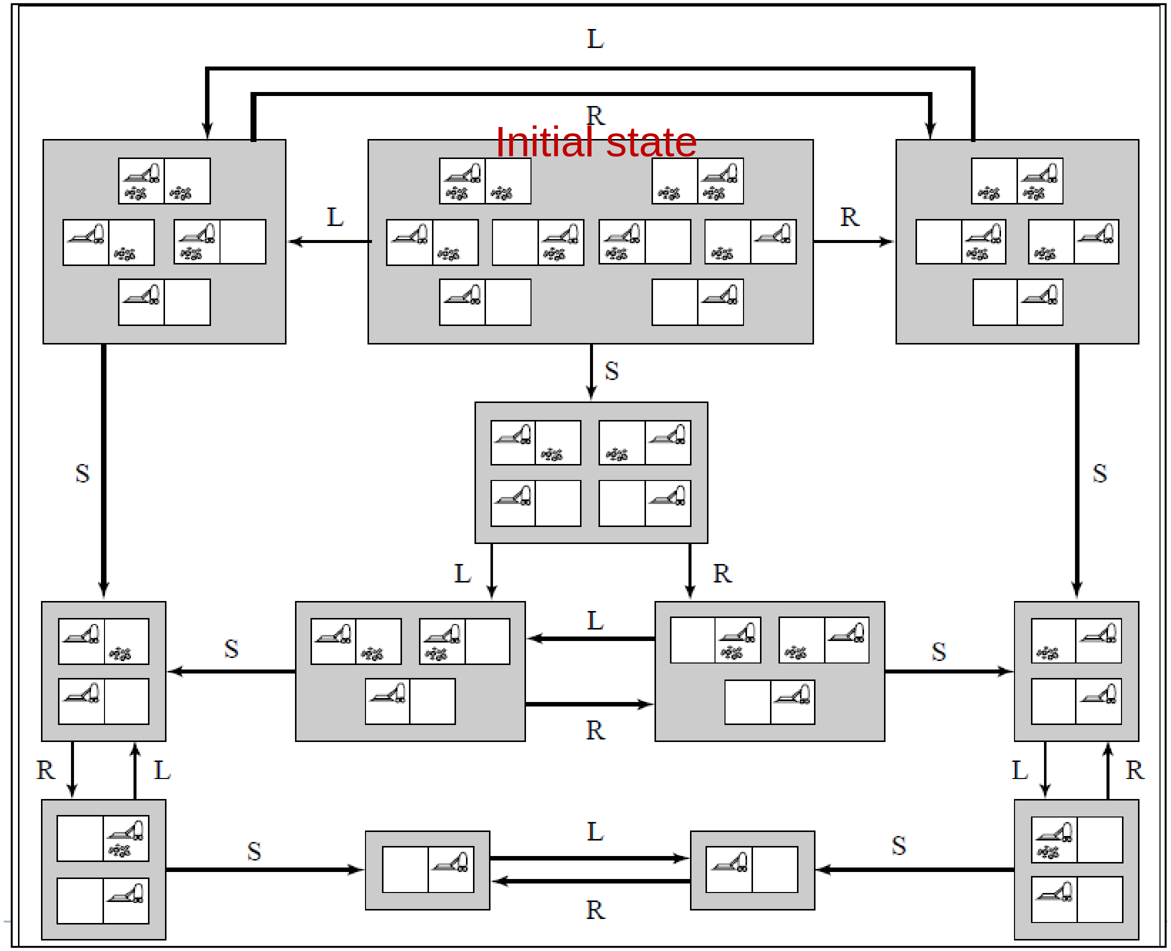
Belief-state space for sensor-less deterministic vacuum world

- Total number of possible belief states? 2^8
- Number of reachable belief states? 12

It is on A

It is on A & A is clean

It is on B & A is clean



Sensor-less problem: searching

- In general, we can use any standard search algorithm.
- Searching in these spaces is not usually feasible (scalability)
  - Problem1: No. of reachable belief states
    - Pruning (subsets or supersets) can reduce this difficulty.
    - Branching factor and solution depth in the belief-state space and physical state space are not usually such different
  - Problem2 (main difficulty): No. of physical states in each belief state
    - Using a compact state representation (like formal representation)
    - Incremental belief-state search: Search for solutions by considering physical states incrementally (not whole belief space) to quickly detect failure if we reach an unsolvable physical state.

Searching with partial observations

- Similar to sensor-less, after each action the new belief state must be **predicted**

- After each perception the belief state is **updated**
  - E.g., local sensing vacuum world
    - After each perception, the belief state can contain at most two physical states.

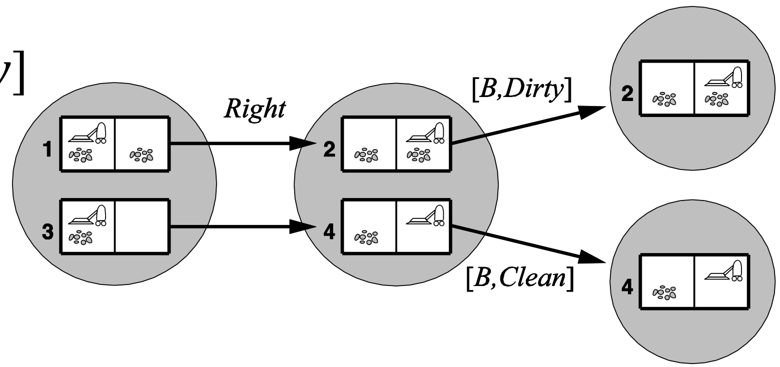
- We must plan for different **possible perceptions**

Searching with partial observations

A position sensor & local dirt sensor

$[A, Dirty]$

Deterministic world



$POSSIBLE\_PERCEPTS(\hat{b})$

Transition model (partially observable env.)

- Prediction stage: How does the belief state change after doing an action?

$$\hat{b} = PREDICT_P(b, a)$$

  - Deterministic actions: $\hat{b} = \{s' \colon s' = RESULTS_P(s, a) \; and \; s \in b\}$
  - Nondeterministic actions: $\hat{b} = \bigcup_{s \in b} RESULTS_P(s, a)$

- Possible Perceptions: What are the possible perceptions in a belief state?

$$POSSIBLE\_PERCEPTS(\hat{b}) = \{o \colon o = PERCEPT(s) \; and \; s \in \hat{b}\}$$

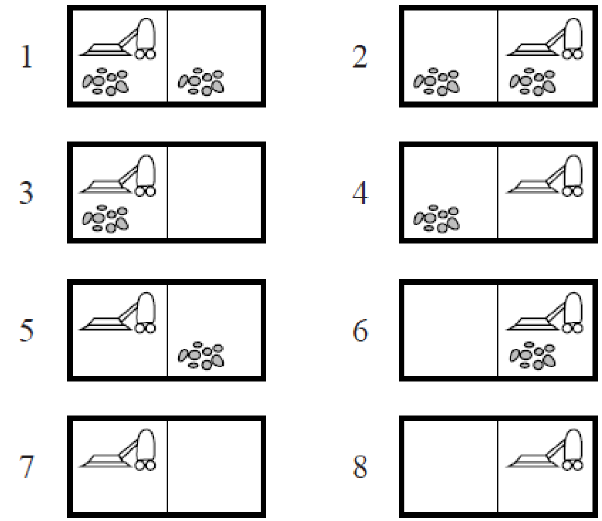
- Update stage: How is the belief state updated after a perception?

$$UPDATE(\hat{b}, o) = \{s \colon o = PERCEPT(s) \; and \; s \in \hat{b}\}$$

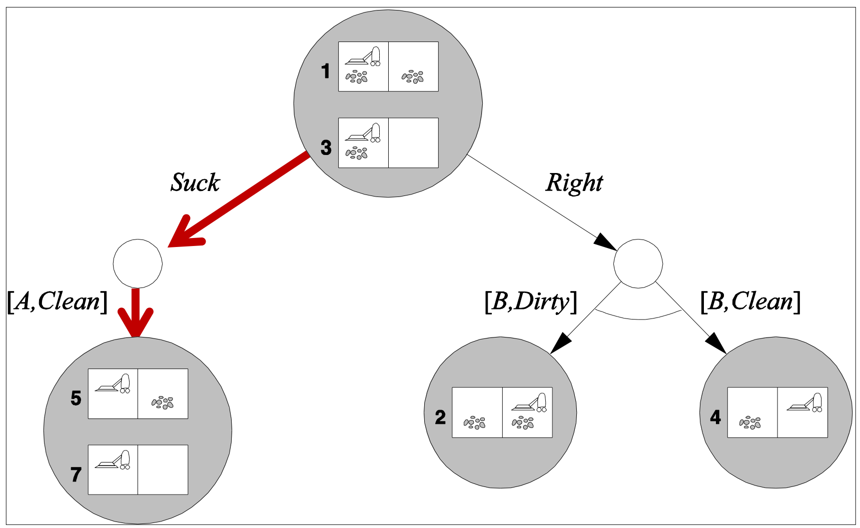
$$\boxed{\begin{array}{c} RESULTS(b, a) = \{b_o \colon b_o = UPDATE(PREDICT(b, a), o) \; and \\ o \in POSSIBLE\_PERCEPTS(PREDICT(b, a))\} \end{array}}$$

AND-OR search tree
local sensing vacuum world

- <u>AND-OR search tree on belief states</u>
- First level



$$PERCEPT = [A, Dirty]$$



- Complete plan

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

Solving partially observable problems

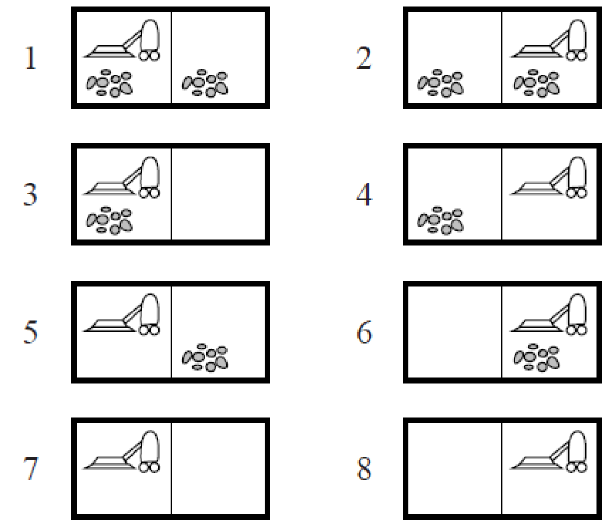
- AND-OR graph search

- Execute the obtained contingency plan
  - Based on the achieved perception either then-part or else-part of a condition is run
  - Agent's belief state is updated when performing actions and receiving percepts
    - Maintaining the belief state is a core function of any intelligent system

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

Kindergarten vacuum world example
Belief state maintenance

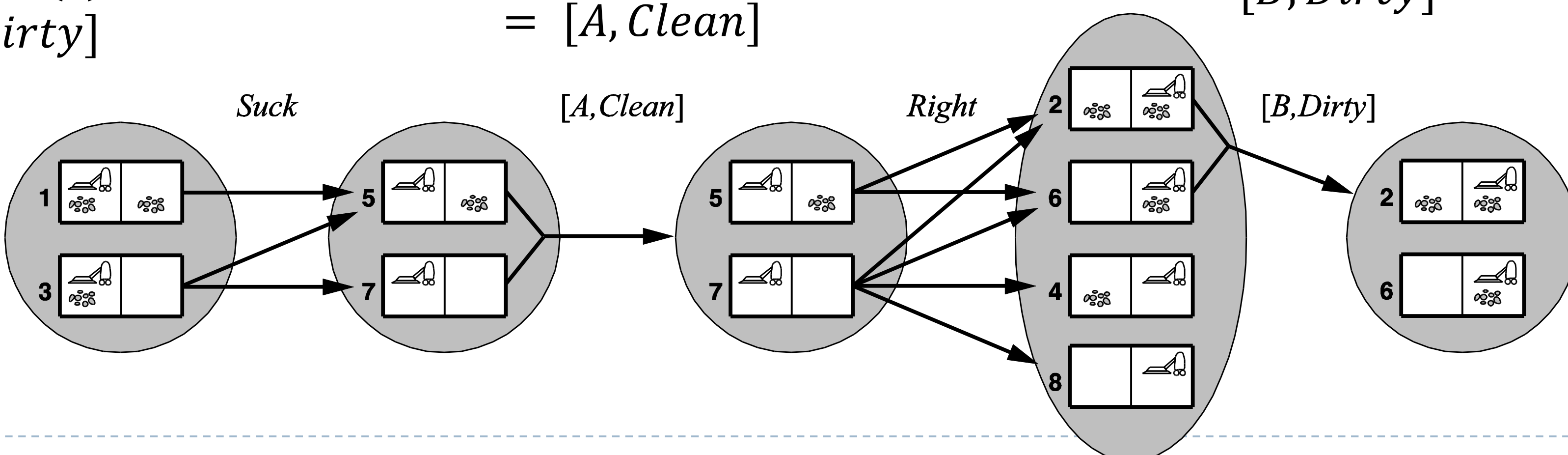
- Local sensing
- Any square may be dirty at any time (unless the agent is now cleaning it)



$PERCEPT(s) = [A, Dirty]$

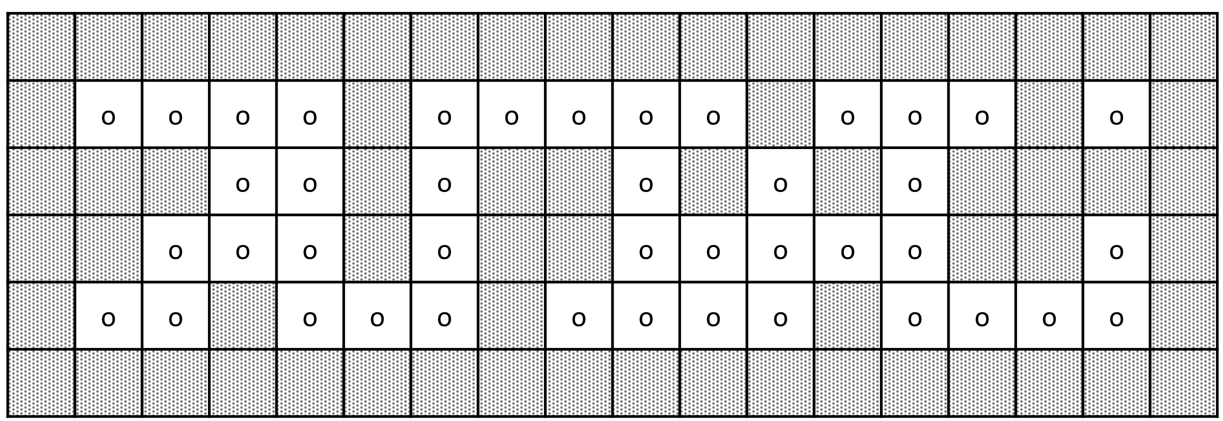
$PERCEPT(s) = [A, Clean]$

$PERCEPT(s) = [B, Dirty]$



Robot localization example

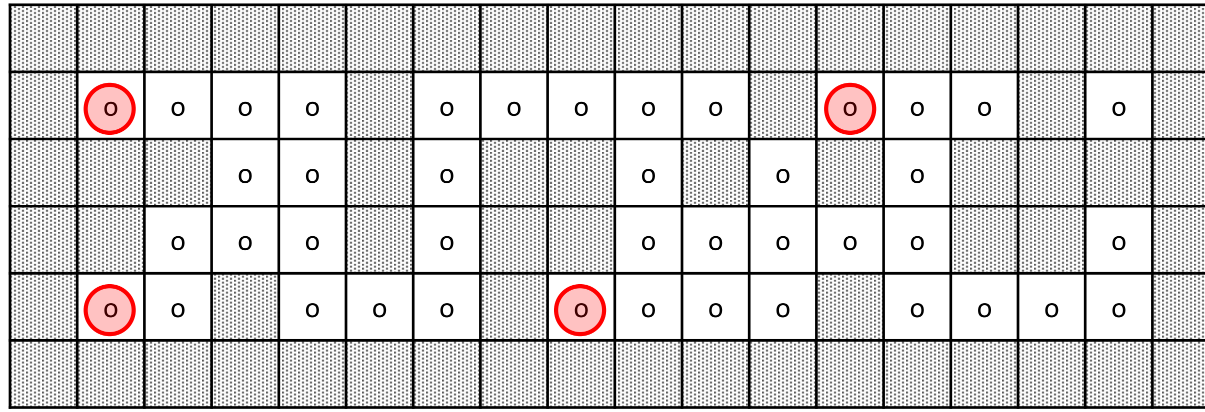
- **Determining current location** given a map of the world and a sequence of percepts and actions
- **Perception**: one sonar sensor in each direction (telling obstacle existence)
  - E.g., percepts=NW means there are obstacles to the north and west
- **Broken navigational system**
  - Move action randomly chooses among {Right, Left, Up, Down}



Robot localization example (Cont.)

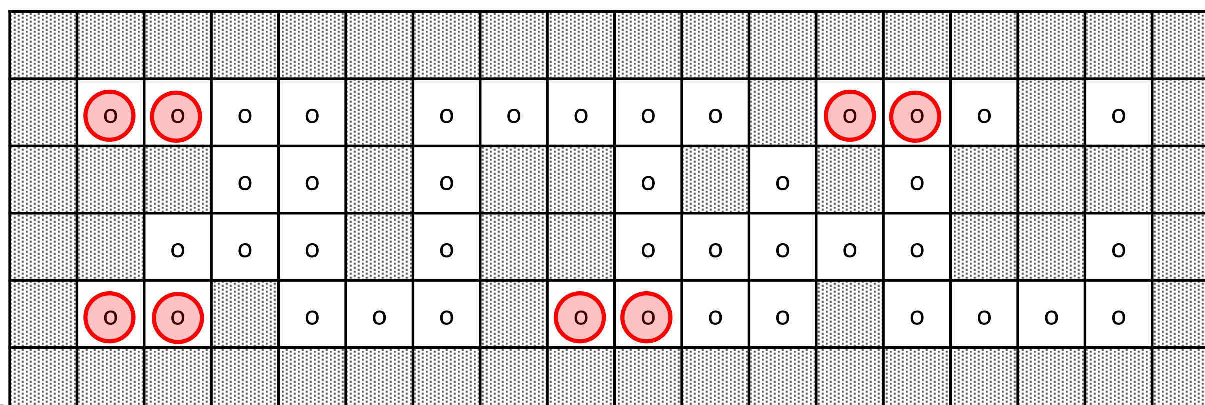
- b^0: o squares
- Percept: NSW
- $b^1 = UPDATE(b^o, NSW)$

  (red circles)



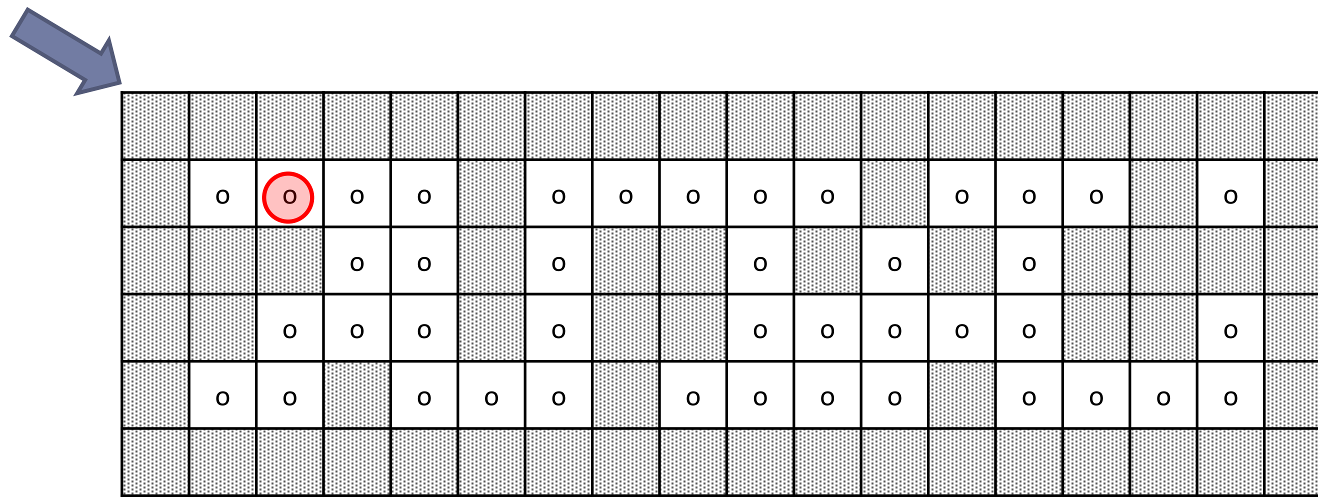
- Execute action $a = Move$
- $b_a^1 = PREDICT(b^1, a)$

  (red circles)



Robot localization example (Cont.)

- Percept: NS
- $b^2 = UPDATE(b_a^1, NS)$



$$UPDATE(PREDICT(UPDATE(b^0, NSW), Move), NS)$$

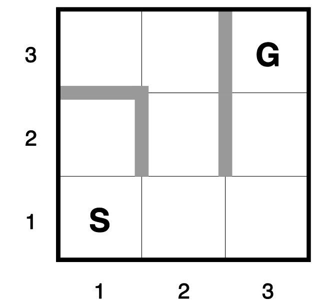
Online search

- Off-line Search: solution is found before the agent starts acting in the real world
- On-line search: interleaves search and acting
  - Necessary in unknown environments
  - Useful in dynamic and semi-dynamic environments
  - Saves computational resource in non-deterministic domains (focusing only on the contingencies arising during execution)
    - Tradeoff between finding a guaranteed plan (to not get stuck in an undesirable state during execution) and required time for complete planning ahead
- Examples
  - Robot in a new environment must explore to produce a map
  - New born baby
  - Autonomous vehicles

Online search problems

- Agent must perform an action to determine its outcome
  - $RESULTS(s, a)$ is found by actually being in s and doing a
  - By filling $RESULTS$ map table, the map of the environment is found.

- Different levels of ignorance
  - E.g., an explorer robot may not know "laws of physics" about its actions
    - [Up, Down] action sequence gets back it to the current location



- May access to a heuristic function

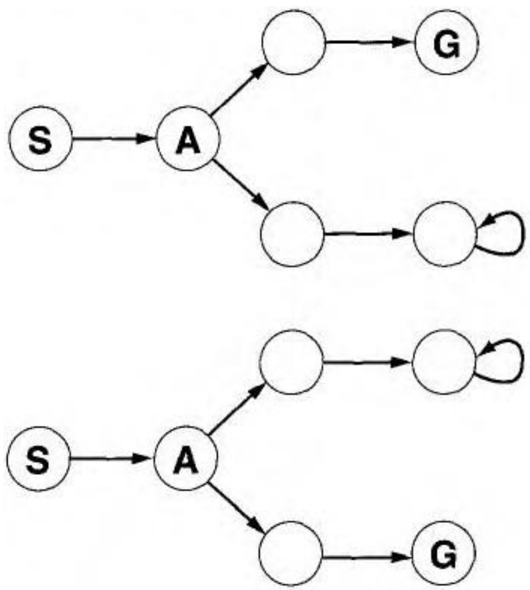
- We assume deterministic & fully observable environment here
  - Also, we assume the agent knows $ACTIONS(s)$, $c(s, a, s')$ that can be used after knowing s' as the outcome, $GOAL\_TEST(s)$

Competitive ratio

- Online path cost: total cost of the path that the agent actually travels
- Best cost: cost of the shortest path "if it knew the search space in advance"
- Competitive ratio = Online cost / Best cost
  - Smaller values are more desirable
- Competitive ratio may be infinite
  - Dead-end state: no goal state is reachable from it
    - irreversible actions can lead to a dead-end state

Dead-end

- No algorithm can avoid dead-ends in all state spaces



- Simplifying assumption: Safely explorable state space
  - A goal state is achievable from every reachable state

Online search vs. offline search

- Offline search: node expansion is a simulated process rather than exerting a real action
  - Can expand a node somewhere in the state space and immediately expand a node elsewhere

- Online search: can discover successors only for the physical current node
  - Expand nodes in a local order
  - Interleaving search & execution

Online search agents

- Online DFS
  - Physical backtrack (works only for reversible actions)
    - Goes back to the state from which the agent most recently entered the current state
    - Works only for state spaces with reversible actions

- Online local search: hill-climbing
  - Random walk instead of random restart
    - Randomly selecting one of available actions (preference to untried actions)
  - Adding Memory (Learning Real Time A*): more effective
    - To remember and update the costs of all visited nodes.

function ONLINE-DFS(s') **returns** an action

inputs: s', a percept that identifies the current state

persistent: $result$, a table indexed by state and action, initially empty

$untried$, a table that lists for each state the actions not yet tried

$unbacktracked$, a table that lists for each state the untried backtracks

s, a, the previous state and action, initially null

if GOAL-TEST(s') **then return** stop

if s' is a new state (not in $tried$) **then return** $untried[s'] \leftarrow \mathrm{ACTIONS}(s')$

if s is not null **then**

$result[s, a] \leftarrow s'$

add s to the front of $unbacktracked[s']$

if $untried[s']$ is empty **then**

if $unbacktracked[s']$ is empty **then return** $stop$

else $a \leftarrow$ an action b such that $result[s', b] = \mathrm{POP}(unbacktracked[s'])$

else $a \leftarrow \mathrm{POP}(untried[s'])$

$s' \leftarrow s$

return a