Beyond Classical Search

AIMA V3 Chapter 4, Sections 4.3–4.5

Problem-solving agents

- ♢ Deterministic, fully observable $\Longrightarrow$ single-state problem
  Agent knows exactly which state it will be in; solution is a sequence

- ♢ Nondeterministic and/or inaccessible/partially observable

- ♢ Deterministic, inaccessible/partially observable

- ♢ Unknown state space

Example: vacuum world

◇ Searching with non-deterministic actions

So far, we dealt with problems where the environment was fully observable and deterministic, so the agent knows what the effects of each action are.

Its percepts provide no new information after each action, although of course they tell the agent the initial state.

◇ When the environment is either partially observable or non-deterministic, percepts become useful:

Percepts narrow down the possible states agent may be in or tell the agent which of the possible outcomes of its actions has actually occurred.

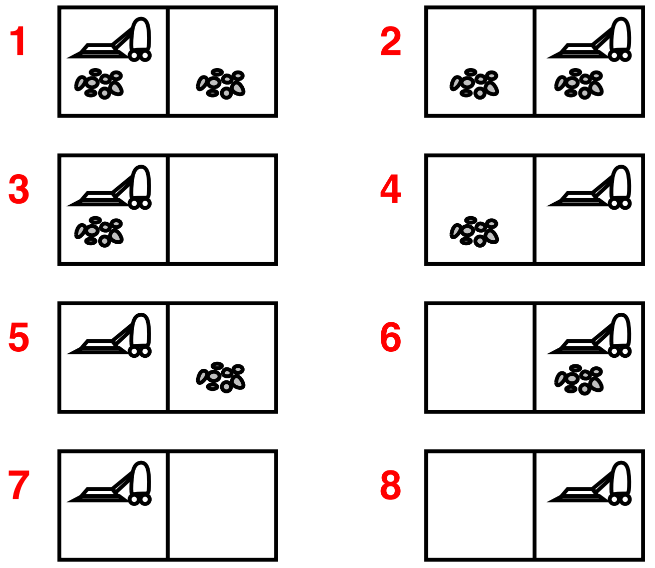
Solution: In either case, the solution is a **contingency plan**.

Example: vacuum world

♢ Single-state, start in #5. Solution??
$[Right, Suck]$

♢ World is stochastic or partially observable
Contingency, start in $\{5\}$
e.g.,. Solution??

♢ Multiple-state (no sensors), start in $\{1,2,3,4,5,6,7,8\}$
e.g., $Right$ goes to $\{2,4,6,8\}$. Solution??



The erratic vacuum world

Suck action works as follows:

- ♢ On a dirty square, it may also clean the adjacent square.
- ♢ On a clean square, it may dump dirt.

If we start for instance in State 1, there is no single sequence of actions that solves the problem.

Nondeterministic: AND-OR search tree

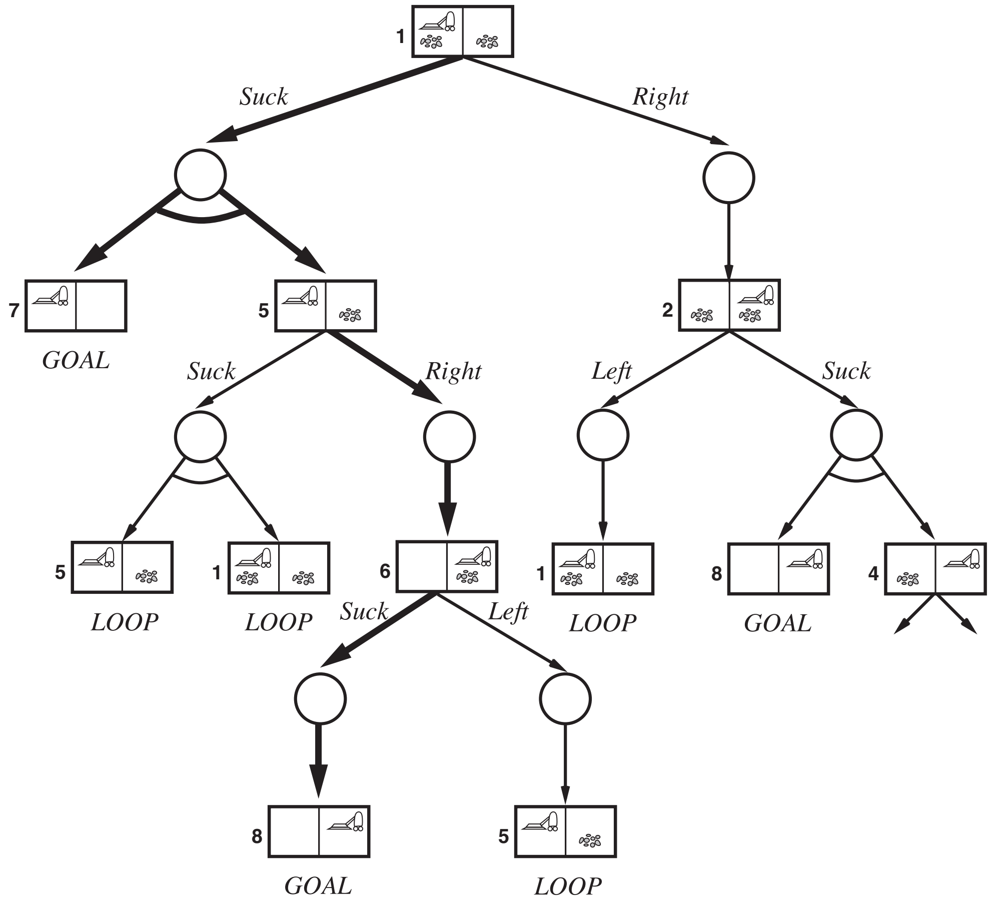
How can we find the contingency solution to nondeterministic problems?

AND-OR Search Trees:
♢ At an OR node, the agent chooses among its available actions i.e., (Left, Right, Suck) ♢ On an AND node, the environments choice of outcome are given.

A solution for an AND-OR search problem is a subtree that:
1) has a goal node at every leaf
2) specifies an action at each of its OR nodes
3) includes every outcome branch at each of its AND nodes

Solution may be found using a modified DFS or BFS or Best First Search.

AND-OR search tree: erratic vacuum world



The solution found is shown in bold lines.

AND-OR search tree: erratic vacuum world

```
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 $plan_n$]
```

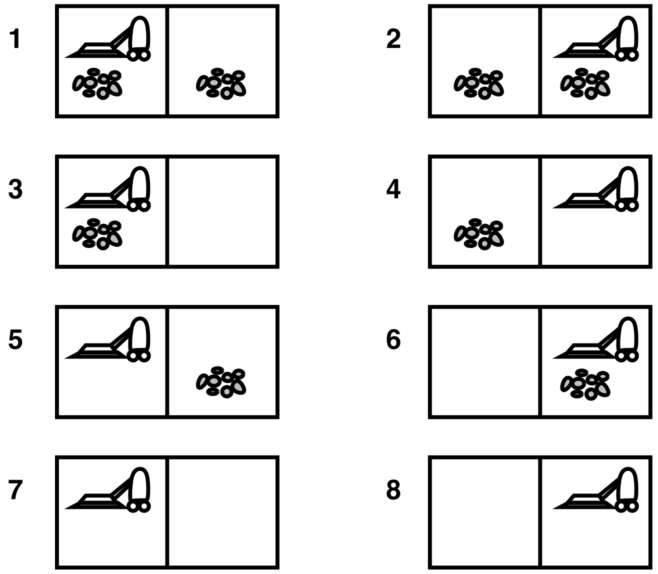
Figure 4.11 An algorithm for searching AND–OR graphs generated by nondeterministic environments. It returns a conditional plan that reaches a goal state in all circumstances. (The notation $[x \mid l]$ refers to the list formed by adding object x to the front of list l.)

Notice that we don't allow loops.

Slippery Vacuum World

Consider a slippery vacuum world which is identical to the original (non-erratic) vacuum world except that movement actions may sometimes fail, leaving the agent in its spot.

E.g. Moving Right in State 1 leads to 1,2



Slippery Vacuum World

♢ There is no longer any acyclic solution from state 1.

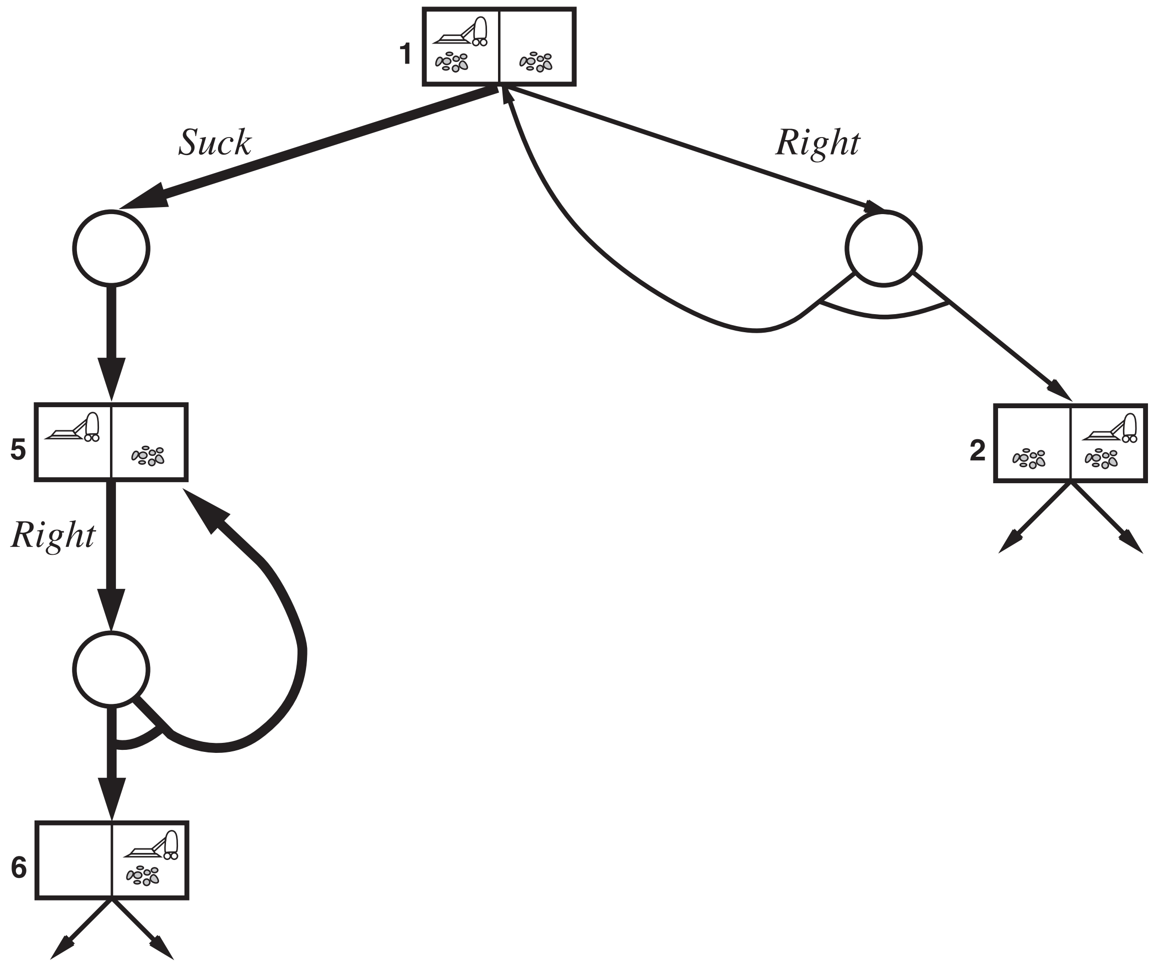
♢ There is however a **cyclic** solution which is to keep trying Right until it works.

[Suck, While State $=$ 5 do Right]

An agent executing such a solution will eventually reach the goal state provided that each outcome of a non-deterministic action eventually occurs.

rolling a dice vs
having the wrong key card.

Slippery Vacuum World - Solution



Searching with Partial Observations

Agent's percepts do not suffice to pin down the exact state.

◊ Belief states: Represent the agent's current belief about the possible physical states it might be in.

◊ No observation (sensorless problem):
no sensor cost
good engineering design eg: orienting parts correctly from an unknown initial position

◊ Partial observation:
E.g. only a local dirt sensor which does not know about the dirt in the other room
Several states could have produced a percept such as [A, Dirty] may come from S1 or S3.

Searching with no observations

When no percept is given, the problem is a sensorless problem (or conformant problem

Set of belief sates (coerced world) is search rather than physical states.

NOTE: the problem is *fully observable* in the belief-state space because the agent always knows its own belief state. Solution also becomes a sequence of actions.

Belief-state search:

$\diamondsuit$ Belief states: contatins every possible set of physical state. 2^N number of states where N is the number of physical state. $\diamondsuit$ Initial state: typically the set of all states

$\diamondsuit$ Actions: If we assume illigal actions have no effect on the enviornment $\rightarrow$ take the $union$ of all actions possible in the belief state. Otherwise, take the $intersection$ of actions.

$\diamondsuit$ Transition model (prediction step): Deterministic actions $\rightarrow$ subset of current belief reachable by the action

Non-deterministic actions $\rightarrow$ union of states reachable by the action for each physical state in the current belief state.

$\diamondsuit$ Goal Test: all physical states in the belief sates satisfies the goal.

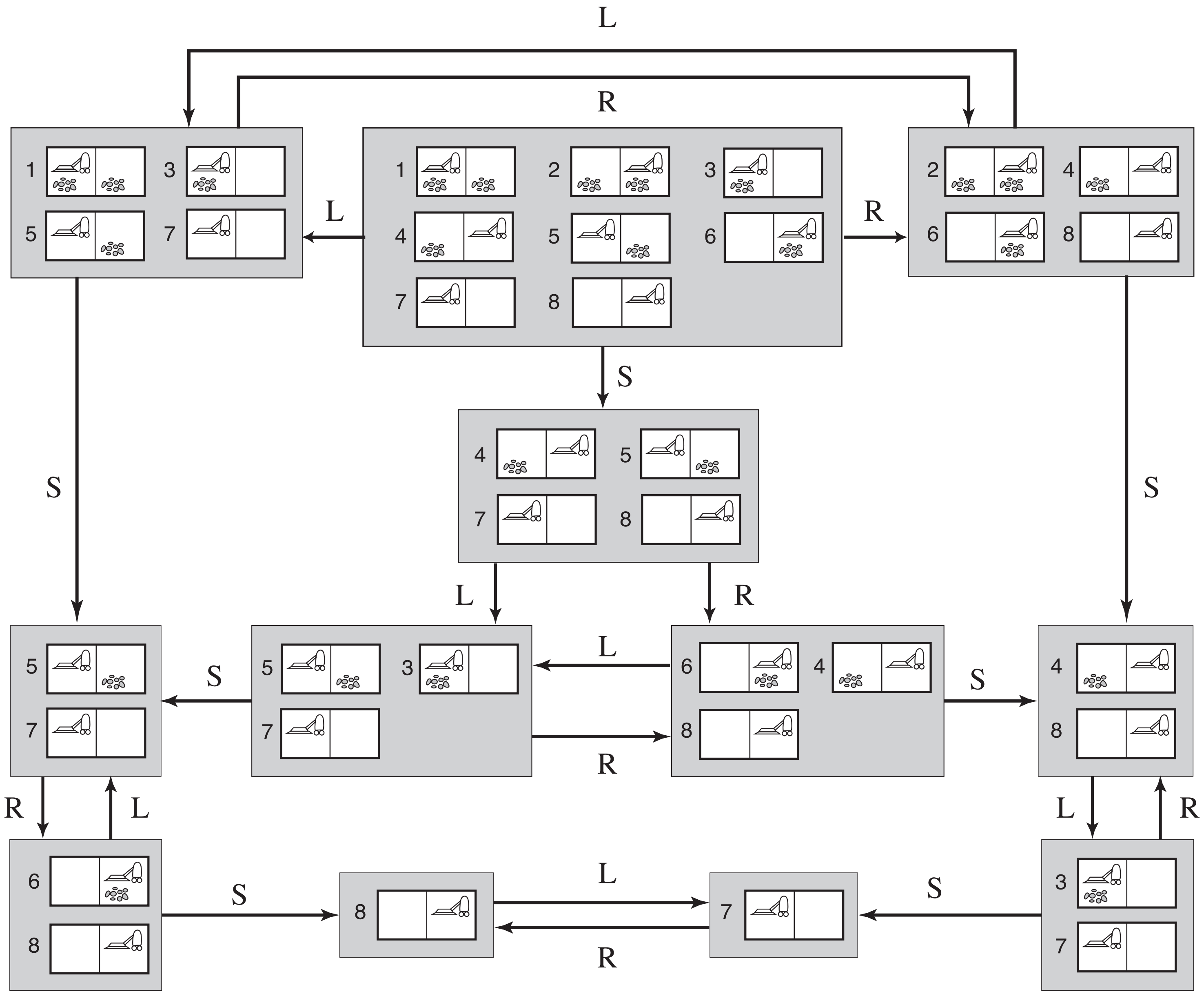
$\diamondsuit$ Path cost: tricky if we have different step costs for each actions.

Vacuum world with No Observation

♢ No observation (sensorless problem):

no sensor cost

good engineering design eg: orienting parts correctly from an unknown initial position

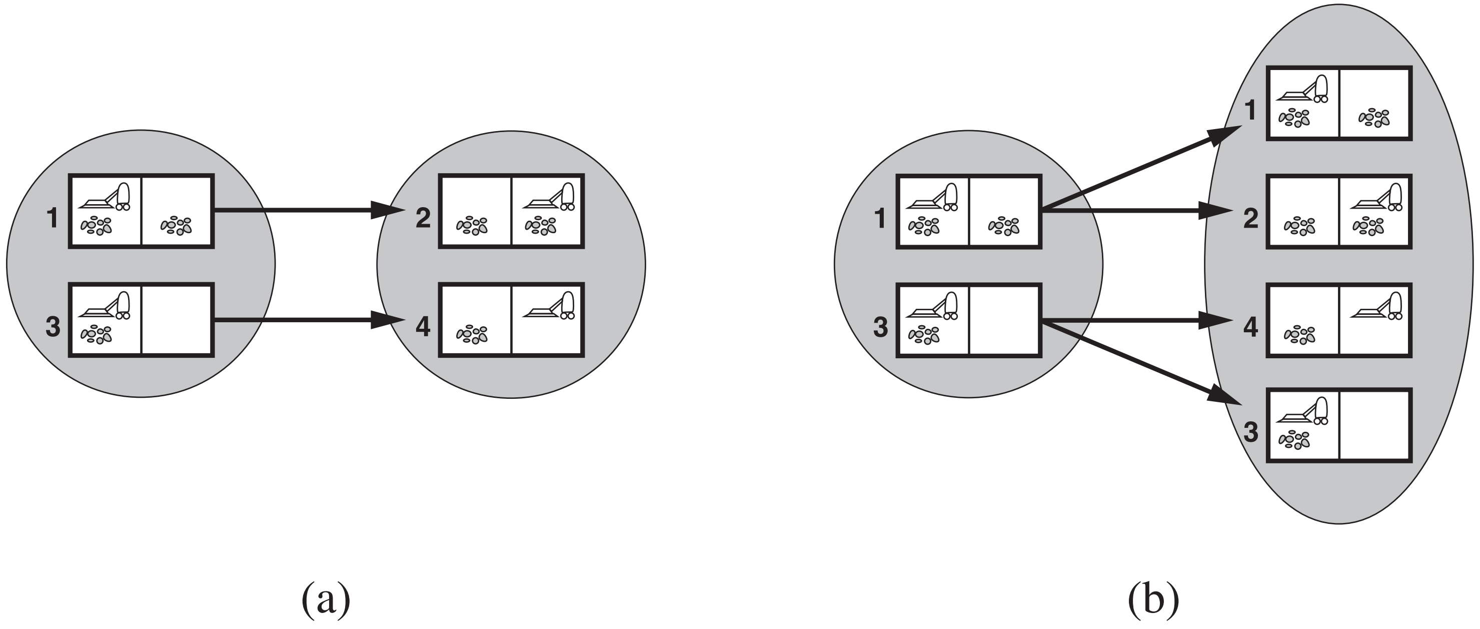

L
R
1
3
5
7
L
1
2
3
4
5
6
7
8
R
2
4
6
8
S
S
S
4
5
7
8
L
R
5
7
S
5
3
7
L
R
6
4
8
S
4
8
R
L
L
R
6
8
S
8
L
R
7
S
3
7

Vacuum world with Partial Observation

♢ Partial observation:

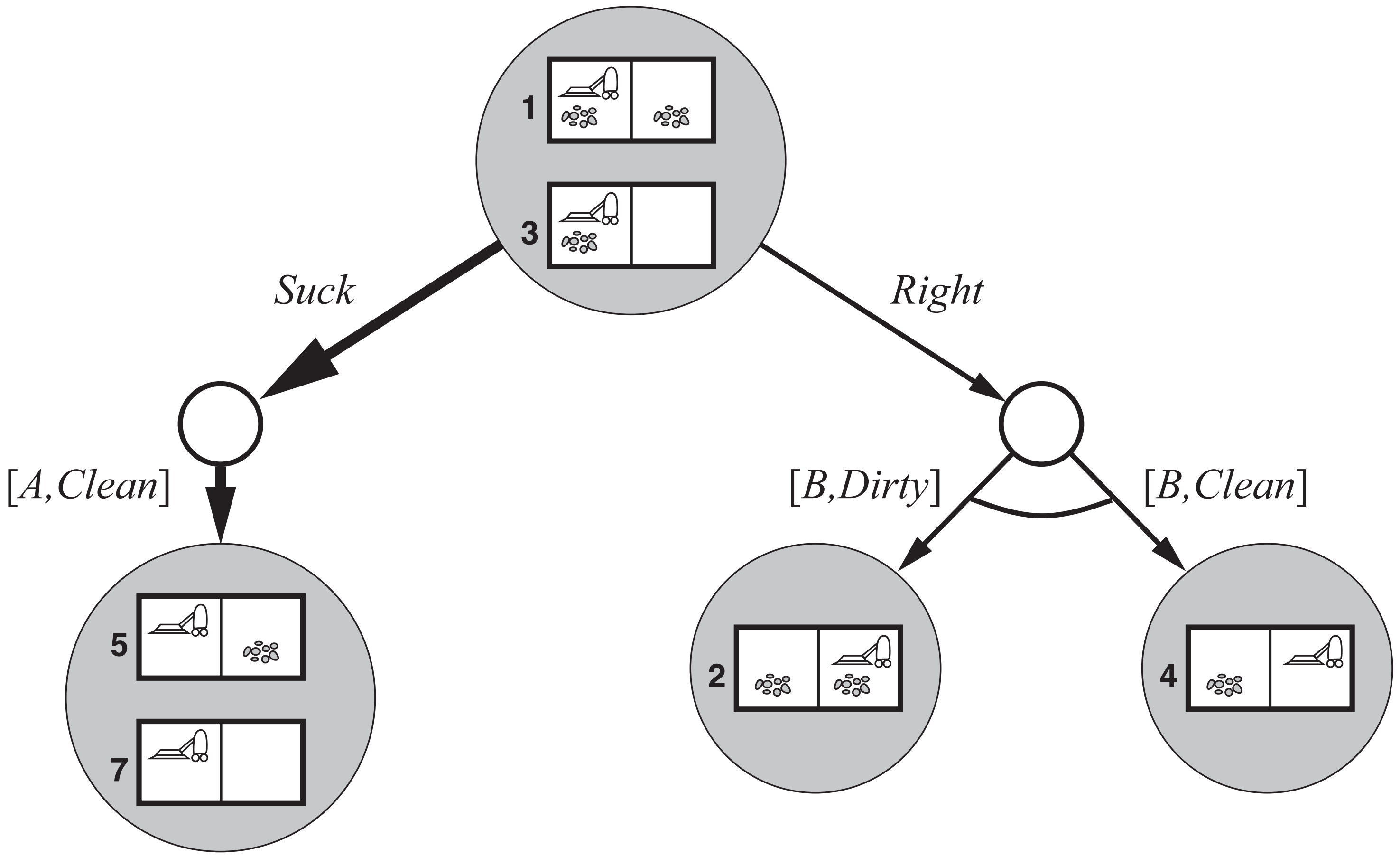
E.g. only a local dirt sensor which does not know about the dirt in the other room

Several states could have produced a percept such as [A, Dirty] may come from S1 or S3.



a) Deterministic world b) Slippery world

Vacuum world with Partial Observation



Chp 4, Section 4.5: Online Search Algorithms

♢ Offline search: complete solution before setting foot in the real world and then executes the solution.

♢ Online search: interleaves computation and execution (e.g. factory robot, rescue robot,...) - exploration

Chp 4: Online Search

♢ Good in dynamic environments

♢ Good in nn-deterministic environments: reduce computation of all contingencies

♢ Necessary for exploration problems, where the environment (states) and possibly the actions are not known to the agent

Examples: robot exploring Mars, baby exploring the world...

Chp 4: Online Search

We assume a deterministic and fully observable environment, where the agent knows only the following:

♢ ACTIONS(s) - list of actions available in state s

♢ Step cost function c(s,a,s') - this cannot be used until the agent knows that s' is the outcome

♢ Goal-Test(s)

After each action, the agent receives a percept telling what state it has reached; from this information, it can augment its map of the environment.

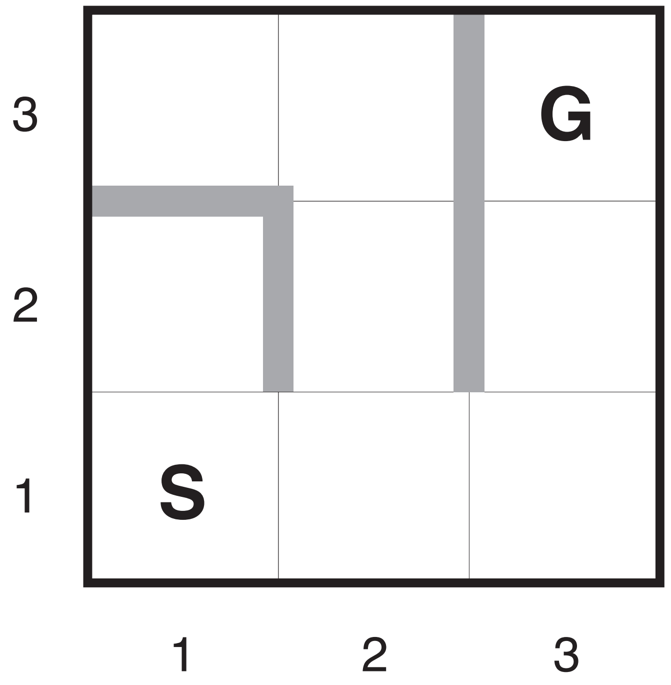
Agent cannot determine RESULT(s,a) except by actually being in s and doing a.

Chp 4: Online Search

Online agents

- ♢ can typically recognize a previously visited state
- ♢ minimizes the cost of reaching the goal (e.g. number of steps)
- ♢ may have a heuristic stating how close to the goal

Chp 4: Online Search



Chp 4: Online Search

♢ *Competitive ratio*: ratio of the cost of online solution to that of offline one for the same problem.

We would like to minimize the competitive ratio, but in cases involving infinite path costs or irreversible states, the competitive ratio will be infinite.

♢ *Safely explorable* spaces

Since no agent can avoid dead-ends in all state-spaces, let's assume that we have safely explorable spaces.

Still no guarantee for a bound for the competitive ratio (adversary argument)

Online Search Algorithms - DFS

♢ A* is offline: expands a node in one part of the space, then a node in another part of the space...

♢ The online search algorithm can expand only those states that it physically occupies..
Search algorithms that expand in local order are more suitable:

Depth-first search: how to do backtracking (keep track of predecessors)? (see algorithm in 4.21)

Online Search Algorithms - Hill Climbing

Already an online search! However, basic version is not very useful due to local maxima.

How to do random restarts?

Online Search Algorithms

♢ Random walk: select at random one of the available actions (possibly preferring actions not tried before)

If the space is finite, a random walk will eventually find a goal or complete its exploration.

Online Search Algorithms- LRTA*

♢ LRTA*: Learning Real Time A*

Augmenting hill-climbing with memory (rather than randomness): store the *current best estimate* $H(s)$ of reaching the goal from each state that has been visited.

♢ $H(s) = h(s)$ initially, but is updated as the agent gains experience.

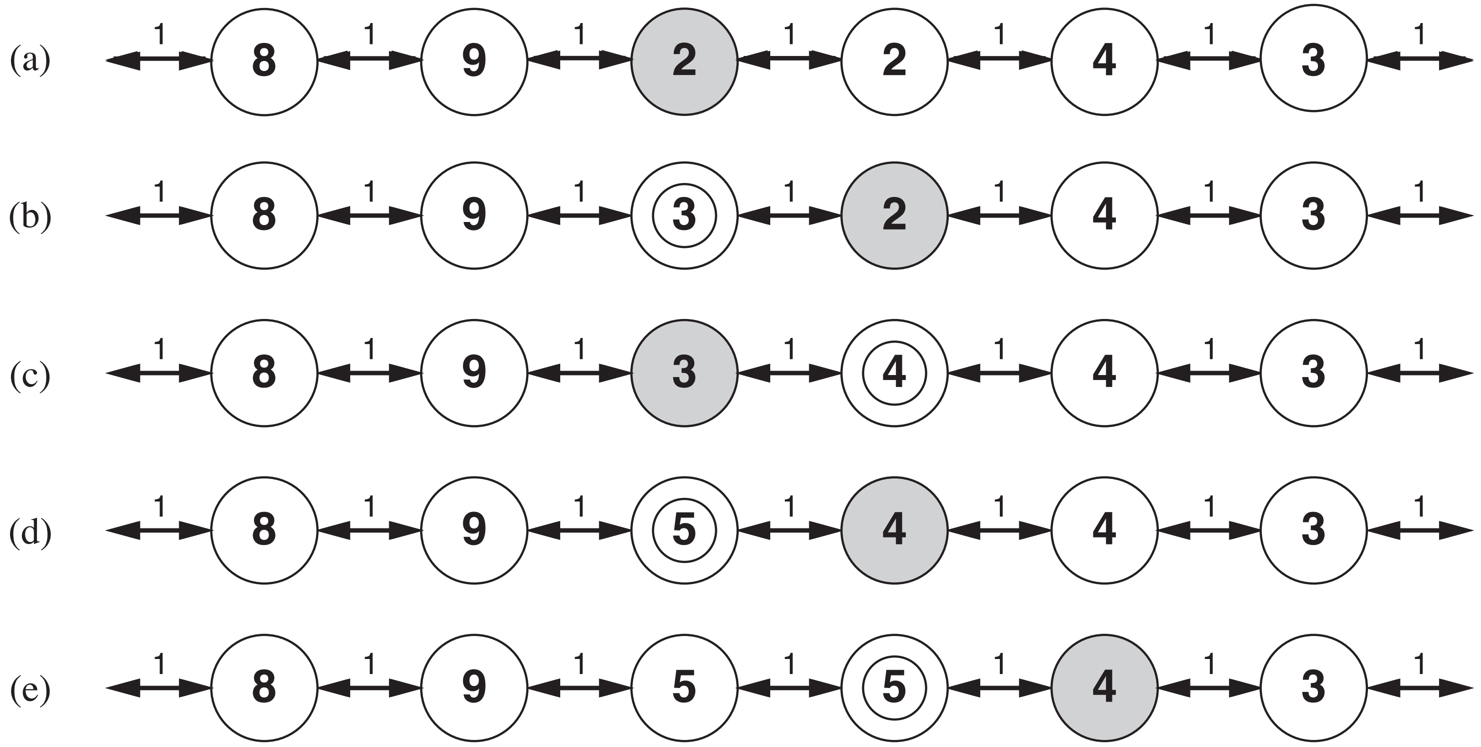
♢ Agent updates the H() estimate for the state it has just left.

$H(s') = h(s')$ when a state is first encountered

$H(s) = c(s, s') + H(s')$

♢ Assumes that h(s') is reliable, improvement comes from the actual cost being used).

Online Search Algorithms - LRTA*



Notice that the updated H values are more correct (9-2-2-4 was not meaningful in a 1D environment)

Online Search Algorithms - LRTA*

♢ An LRTA* gent is guaranteed to find a goal in any finite, safely explorable environment
$O(n^2)$ steps in the worst-case in an environment with n states.

♢ Unlike $A*$, it is not complete in infinite spaces