Lecture 9

- Reminder: Homework 2 due on Thursday
- Questions?

Outline

- Chapter 4 - Beyond Classical Search
  - Searching with Nondeterministic Actions
  - Searching with Partial Observation
  - Online Search Agents and Unknown Environments

Searching with Partial Information

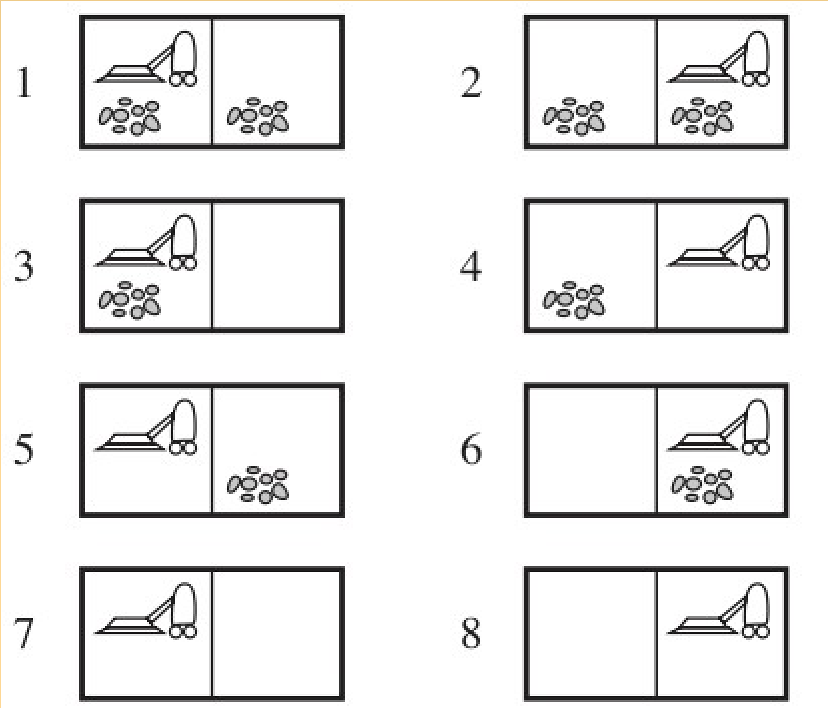
- When the environment is fully observable and deterministic, and the agent knows what the effects of each action are, percepts provide no new information after the agent determines the initial state.
- In partially observable environments, every percept helps narrow down the set of possible states the agent might be in, making it easier for the agent to achieve its goals.

Searching with Partial Information

- In nondeterministic environments, percepts tell the agent which of the possible outcomes has actually occurred.
- In both cases, future percepts cannot be determined in advance and the agent's future actions will depend on those future percepts.
- A solution to this type of problem is a ***contingency plan*** (also know as a ***strategy***) that specifies what to do depending on what percepts are received.

Searching with Nondeterministic Actions

- Example: Erratic Vacuum World, same state space as before. Goal states are 7 and 8.
- *Suck* action is:
  - 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.



Searching with Nondeterministic Actions

- To formulate this problem, generalize notion of transition model from before. Use Results function that returns a ***set*** of possible outcome states.
- E.g., Results (1, *Suck*) = { 5, 7 }
- Also generalize notion of a solution to a contingency plan.
- E.g., From state 1, [*Suck*, if *State* = 5 then [*Right*, *Suck*] else [ ]]

Searching with Nondeterministic Actions

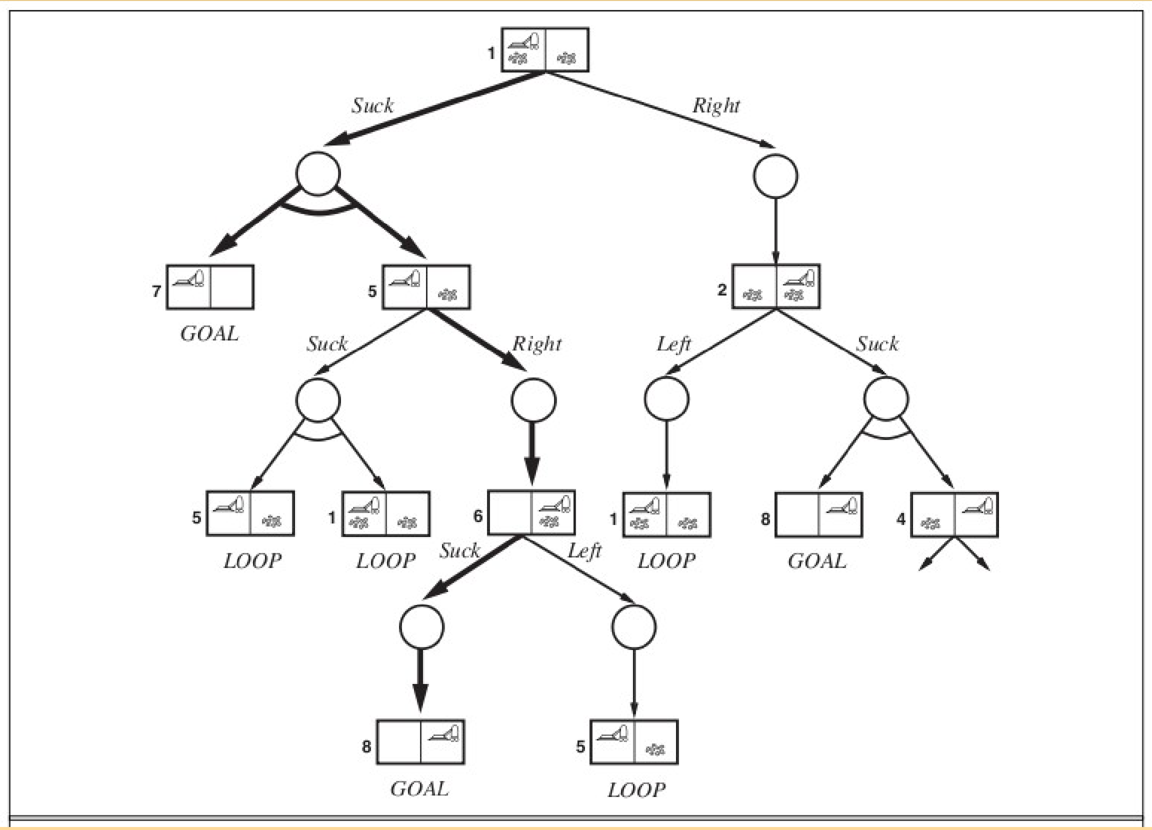
- Solutions for nondeterministic problems can contain nested if-then-else statements that create a ***tree*** rather than a sequence of actions.
- Many problems in real, physical world are contingency problems because exact prediction is impossible.

AND-OR Search Trees

- Augment search trees in the following way
  - Branching caused by agent's choice of action are called ***OR nodes***. E.g., in vacuum world, agent chooses *Left* **or** *Right* **or** *Suck*.
  - Branching caused by environment's choice of action are called ***AND nodes***. E.g., in erratic vacuum world, *Suck* action in state 1 leads to a state in { 5, 7 }, so agent would need to find a plan for state 5 **and** state 7.
- Two kinds of nodes alternate giving ***AND-OR tree***.

AND-OR Search Trees

- Solution is a subtree that
  - has a goal node at every leaf
  - specifies one action at each OR node
  - includes every outcome branch at each AND node.



AND-OR Search Algorithms

Function: AND-OR-Graph-Search
Receives: *problem*; Returns: *conditional plan* or *failure*

1. OR-Search (*problem*.InitialState, *problem*, [ ])

Function: OR-Search
Receives: *state*, *problem*, *path*
Returns: *conditional plan* or *failure*

1. If *problem*.GoalTest(*state*) then return empty plan
2. If *state* is on path then return failure
3. For each action in *problem*.Actions(*state*) do
   3.1 *plan* = AND-Search (Results(*state*, *action*), *problem*, [*state* | *path*])
   3.2 if *plan* ≠ *failure* then return [*action* | *plan*]
4. Return *failure*

AND-OR Search Algorithms

Function: AND-Search
Receives: *states*, *problem*, *path*
Returns: *conditional plan* or *failure*

1. For each s_i in *states* do
   1.1 $plan_i$ = OR-Search(s_i, *problem*, *path*)
   1.2 If $plan_i$ = *failure* then return *failure*
2. Return [if s_1 then $plan_1$ else if s_2 then $plan_2$ else …
   if s_{n-1} then $plan_{n-1}$ else $plan_n$]

- Note loops are handled by looking for a state in the current path and returning failure. This guarantees termination in finite state spaces.

Searching with Nondeterministic Actions

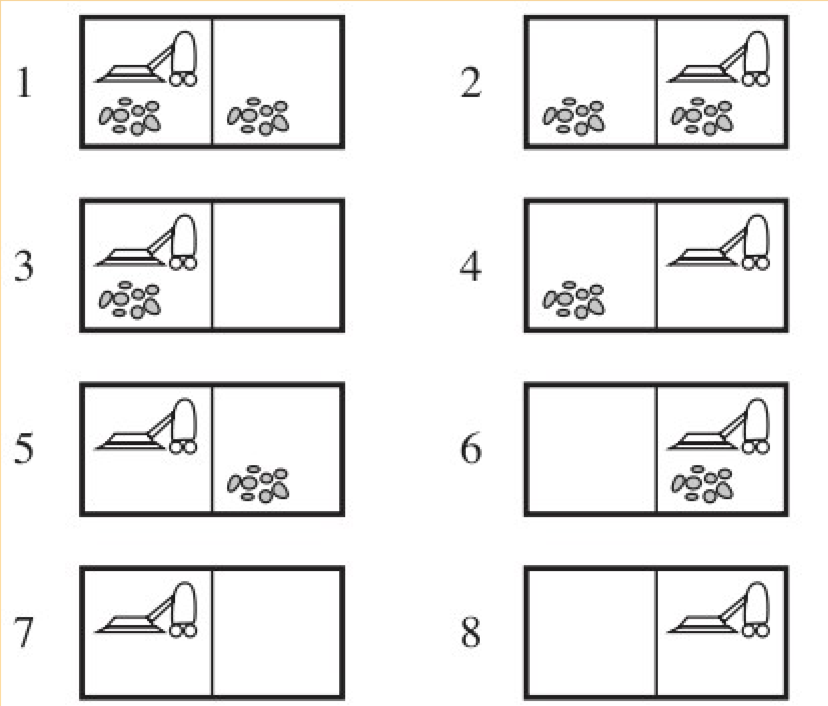
- Given algorithm is DFS. Can also search tree with BFS or best-first, and generalize heuristic function for contingency plans for an analog to A*.
- An alternative agent design is for the agent act ***before*** it has a guaranteed plan and deal contingencies only as they arise in execution.
- This type of ***interleaving*** of search and execution is useful for exploration problems and game playing.

Searching with Partial Observations

- As noted, when an environment is partially observable, an agent can be in one of several possible states. An action leads to one of several possible outcomes.
- To solve these problems, an agent maintains a ***belief state*** that represent the agent's current belief about the possible physical state it might be in, given the sequence of actions and percepts up to that point.

Searching with No Observation

- When an agent's percepts provide no information at all, this is called a ***sensorless*** (or ***conformant***) problem
- Example: deterministic, assume know geography, but not location or dirt.
  - Initially could be in any state { 1, 2, 3, 4, 5, 6, 7, 8 }
  - Do *Right*, must in { 2, 4, 6, 8 }. Do *Suck* results in { 4, 8 }. Doing *Left*, then *Suck* ***coerces*** world into state 7 regardless of initial state



Searching with No Observation

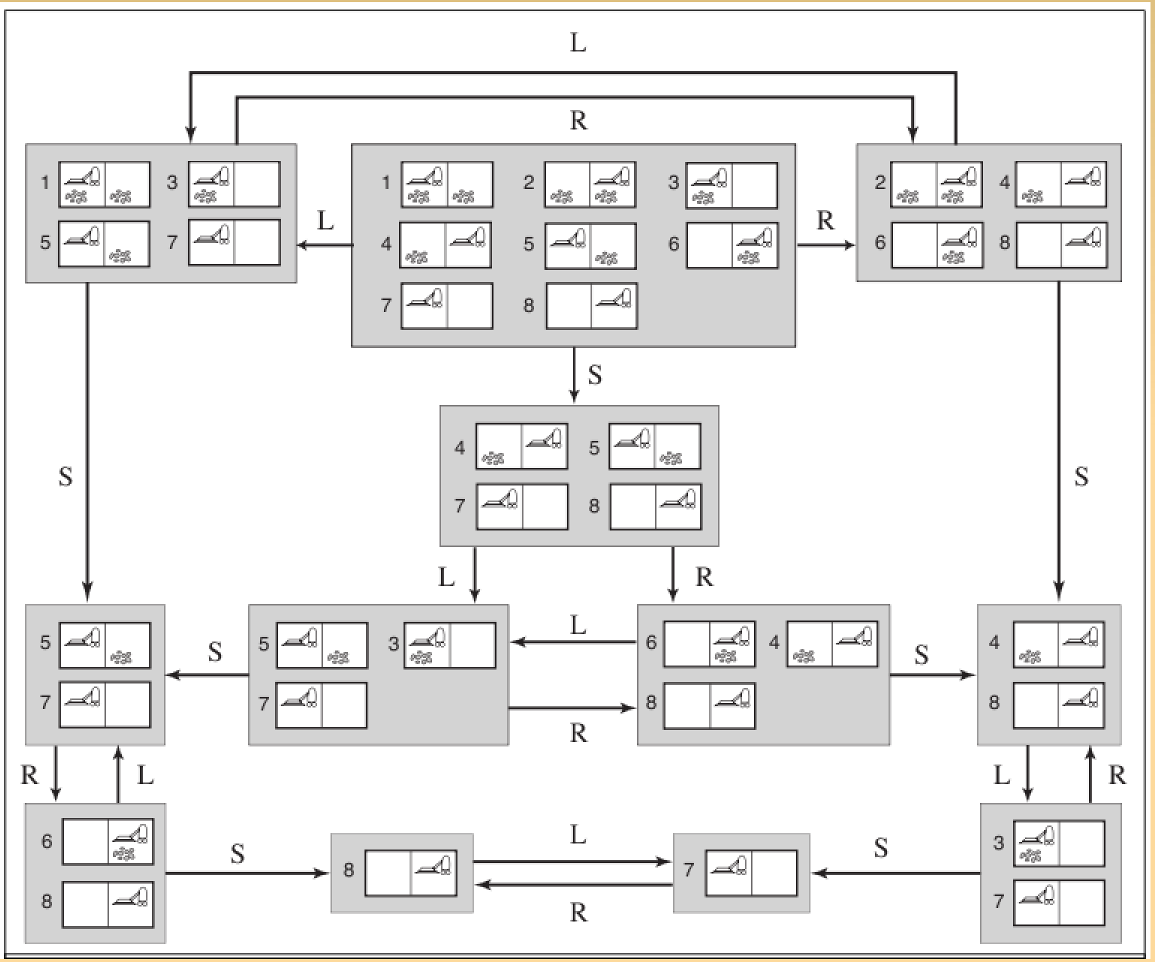
- Solve sensorless problems by searching space of belief states, which is fully observable to agent. Suppose physical problem *P* with Actions_P, Result_P, GoalTest_P and StepCost_P.
- Define corresponding sensorless problem:
  - Belief states: powerset of states of *P*, although many are unreachable from initial state. For *N* states of *P*, there are 2^N possible belief states.
  - Initial state: typically, all the states of *P*

Searching with No Observation

- Actions: if illegal actions are safe, use the union of actions of all states in *b*; otherwise use the intersection. Use to ***predict*** the next belief states.
- Transition model: $b' = \text{Result}(b, a) = \{ s' : s' = \text{Result}_P(s, a) \text{ and } s \in b \}$. Similarly for nondeterministic environments using Results_P.
- Goal test: ***all*** of the physical states in *b* must satisfy GoalTest_P
- Path cost: Assume that action costs same in all states, so transfer from underlying problem.

Searching with No Observation

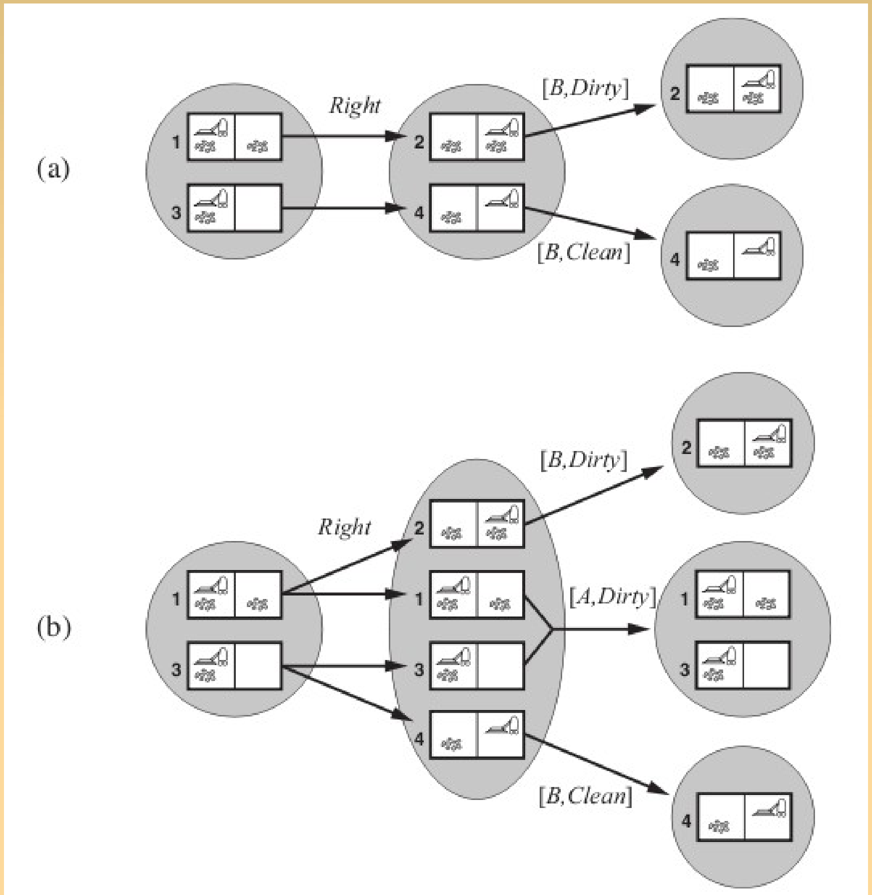
- Complete belief state search space for deterministic environment.
- Note only 12 reachable states out of 2^8 possible belief states.



Searching with Observations

- For a general partially observable problem, need to specify how the environment generates percepts for agent. Encapsulated in function Percept (*state*). Special cases:
  - Percept (*s*) = *s*, for fully observable environments
  - Percept (*s*) = *null*, for sensorless problems
- Example: in vacuum cleaner world
  - Percept (1) = [*A*, *Dirty*]

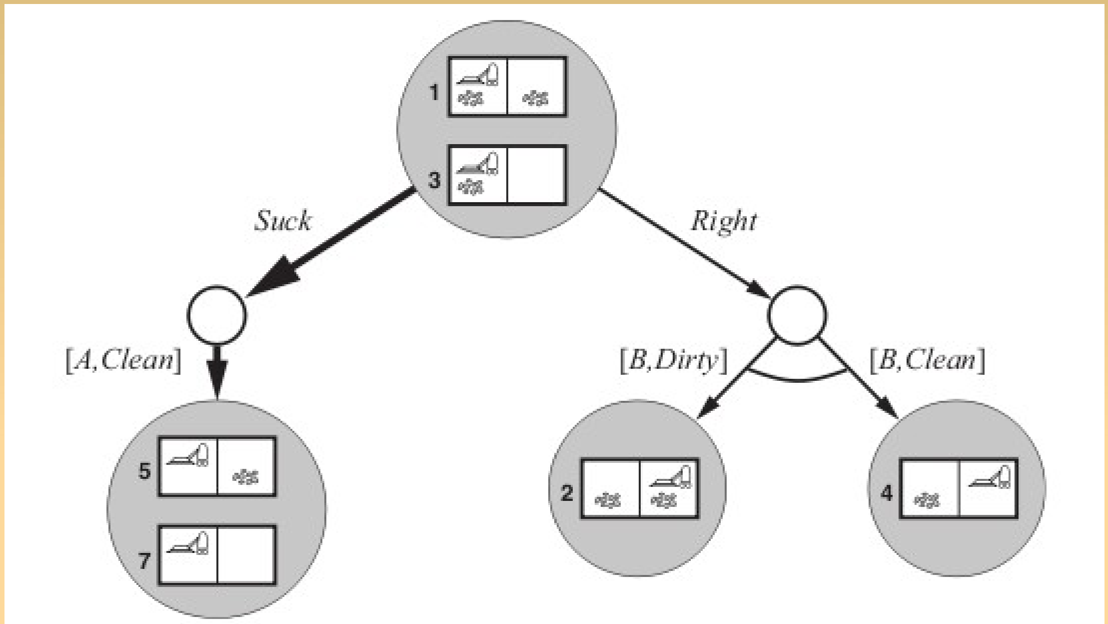
Searching with Observations



- Example: first percept is [*A*, *Dirty*], do *Right*
- In deterministic environments, percepts reduce size of belief state
- In nondeterministic environments, belief state may become larger

Solving Partially Observable Problems

- Use AND-OR search to solve. Top part of tree starting with [A, Dirty].
- Solution is conditional plan:
  - [*Suck*, *Right*, if *Bstate* = { 6 } then *Suck* else [ ]]



Online Search Problems

- Previous problems use ***offline search*** algorithms. Complete solution is computed, then solution is executed.
- In ***online search***, agent interleaves computation with action: it first takes an action, then observes the environment and computes the next action.

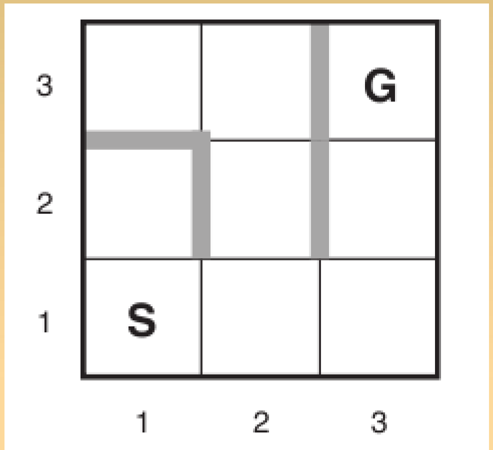
Online Search Problems

- Good idea for dynamic environments where there is a penalty for computing too long.
- Helpful idea for nondeterministic environments. Don't spend time planning for contingencies that are rare.
- ***Necessary*** idea for unknown environments, where agent doesn't know what states exists or the results of its actions. Called an ***exploration problem***. Agent uses actions as experiments to learn about its environment.

Online Search Problems

- Assume deterministic, fully observable environment. Agent only knows:
  - Actions(*s*) – list of actions allowed in state s
  - Step-cost function $c(s, a, s')$ – cannot be used until agent knows that s' is the outcome of doing *a*
  - GoalTest(*s*)
- In particular, it doesn't know Result (s,a) except by actually being in *s* and doing *a*, then observing s'.

Online Search Problems

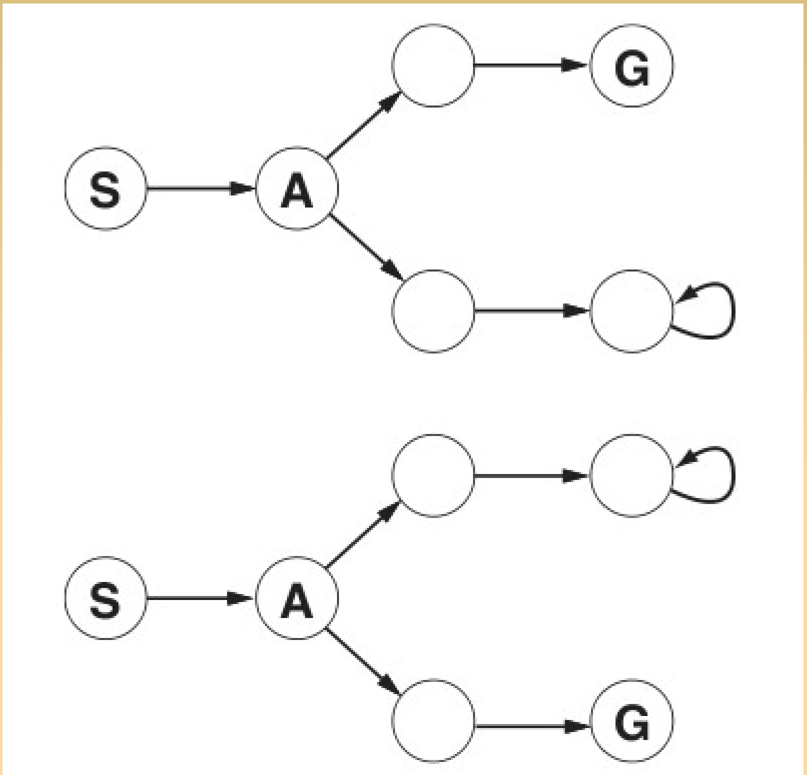


- Example: maze problem. Agent starts in S. Goal is to move to G.
- Agent initially does not know that going *Up* from (1,1) leads to (1,2) nor that going *Down* from there goes back to (1,1).
- Agent may have access to admissible heuristic. E.g., if agent knows where goal is, can use Manhattan distance heuristic.

Online Search Problems

- Typically, objective is to reach goal state while minimizing cost, where cost is total path cost of path an agent actually travels.
- Common to compare this cost with path cost of path agent would follow if it knew the search space in advance, i.e., the actual shortest path. Called the ***competitive ratio***, and would like it to be as small as possible.

Online Search Problems



(a)

- Nice idea, but in some cases competitive ratio is infinite, if online search agent reaches a dead-end state from which no goal state is reachable.
- Claim: no algorithm can avoid dead ends in all state spaces. Two states shown are indistin-guishable, so must result in same action sequence. Will be wrong for one.

Online Search Problems

- To make progress, we assume that state space is ***safely explorable***. I.e., some goal state is reachable from every reachable state. State spaces with reversible actions (mazes, 8-puzzles) are clearly safely explorable.
- Even with this assumption, no bounded competitive ratio can be guaranteed if there are paths of unbounded cost. Common to describe performance in terms of size of entire state space instead of depth of the shallowest node.

Online Search Algorithms

- Online search algorithms are very different from offline search algorithms. Since agent occupies specific physical node, can only expand immediate successors.
- Need to expand nodes in ***local*** order. DFS has this property, but needs reversible actions to support backtracking.

Online Search Algorithms

- Hill-climbing already is a local search algorithm, but can get stuck in local maxima. Add memory to keep track of a "current best estimate", $H(s)$, of the cost to reach goal from each state that has been visited.
- $H(s)$ starts out with $h(s)$, the heuristic estimate and is updated as agent gains experience.