CS444

Nathan Sprague

August 31, 2012

Agents, Environments, Problems

Search Problems

- We'll begin by considering problems that are:
  - Observible
  - Discrete
  - Known
  - Deterministic
- Atomic state representation

Problem Definition

- States
- Initial State
- Actions
- Transition Model:
  - RESULT(s, a)
- Goal Test
- Path Cost Function

- Solution : A sequence of actions leading from initial state to a goal state.
- Optimal Solution : The lowest-cost sequence of actions leading from initial state to a goal state.

Problems as Graphs

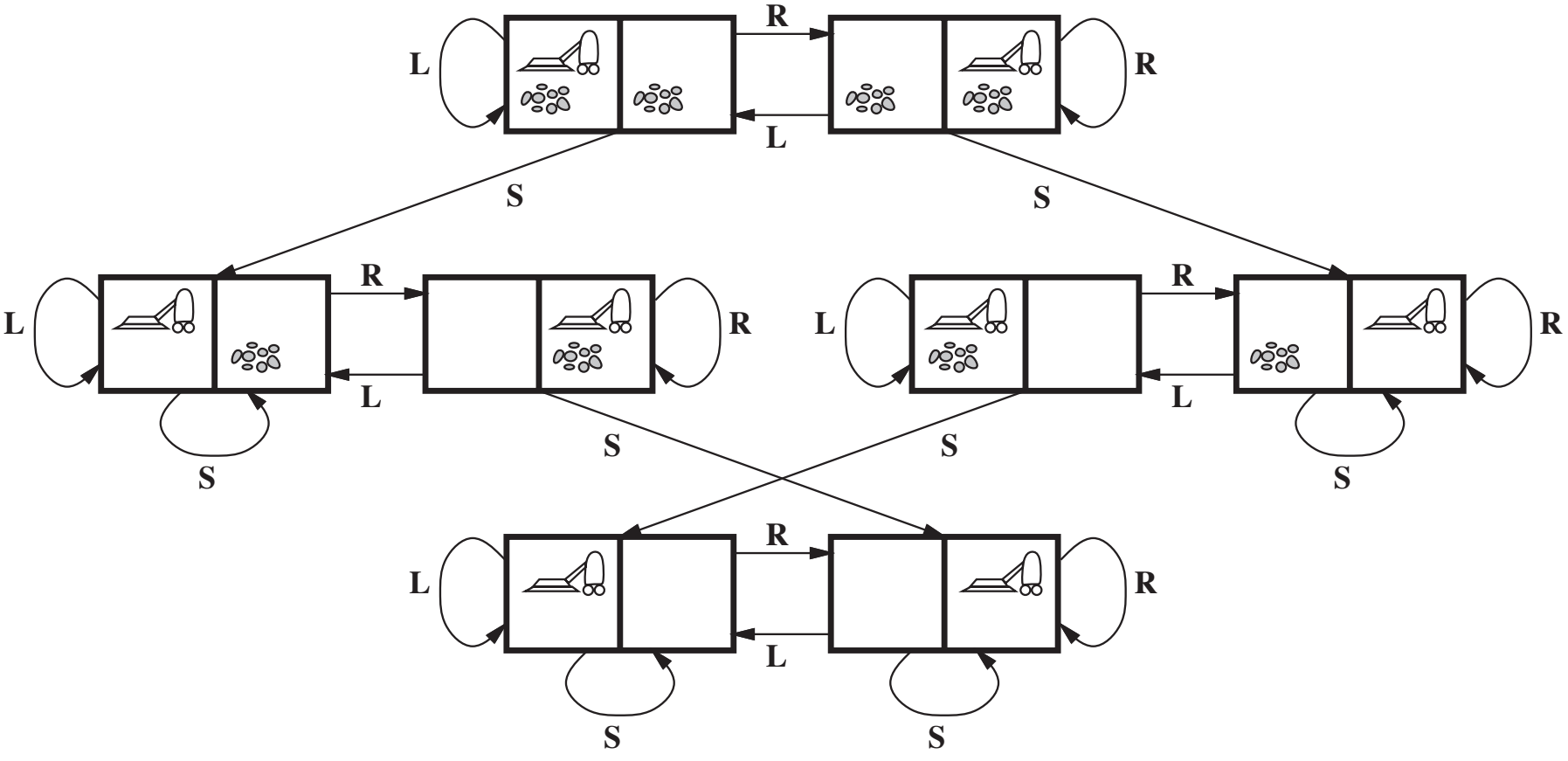
- States are vertices
- Actions are edges
- Costs are weights

Trivial Example

Vacuum World - A vacuuming robot is in one of two possible locations, either or both of which may be dirty.

- States - Location of robot, presence of dirt
  - $2 \times 2^2 = 8$ total states
- Initial State - Any
- Actions
  - LEFT, RIGHT - Move the robot (if possible)
  - SUCK - Remove dirt from current location (if present)
  - All actions are available in all states
- Transition Model:
  - All actions have their expected effect.
- Goal Test - All locations are clean.
- Path Cost - one/step.

Graph Representation



Tree Search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

A Problem

Graph Search

```
function TREE-SEARCH( problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH( problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

Breadth-First Search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Anaysis of Search Algorithms

- Completeness
- Optimality
- Time Complexity
- Space Complexity

Search Algorithms

	BFS	DFS	Iterative Deepening	Uniform Cost Search
TREE-SEARCH				
GRAPH-SEARCH				

(See also Fig. 3.21 in the textbook)