Admin

Tutorials start this week;

See information from ITO;
tutorial groups linked to FAI web page.

Today

- Problem solving agents
- State spaces and search trees
- Components of general state space search algorithm

See Russell and Norvig, Chapter 3.

Problem-solving agent

Think of an agent that:

- receives sensory input
- has goal(s) it wants to achieve
- can perform some actions
- wants to work out what action to perform to achieve its goal.

Actions and perceptions correspond to change of state.

```
function S      -P        -S       -A      ( percept) returns an action
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

   state ← U       -S      (state, percept)
   if seq is empty then
        goal ← F           -G     (state)
        problem ← F            -P          (state, goal)
        seq ← S        ( problem)
   action ← R                  (seq, state)
   seq ← R           (seq, state)
   return action
```

Problem-solving agent

This is a restricted form of general agent.

Note: this is **offline** problem solving; solution executed “eyes closed.”
Online problem solving involves acting without complete knowledge.

Example: Romania

On holiday in Romania; currently in Arad.
Flight leaves tomorrow from Bucharest

Formulate goal: be in Bucharest

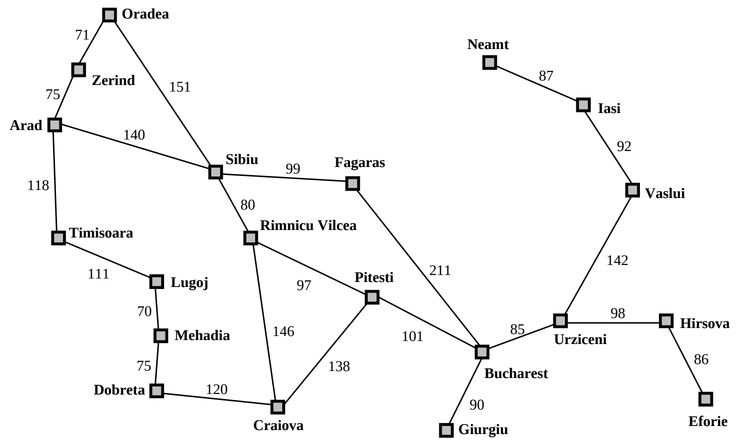
Formulate problem:

states: various cities

actions: drive between cities

Find solution: sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



Problem types

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

Non-observable $\Longrightarrow$ **conformant problem**
Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable $\Longrightarrow$ **contingency problem**
percepts provide **new** information about current state
solution is a **tree** or **policy**
often **interleave** search, execution

Unknown state space $\Longrightarrow$ **exploration problem** (“online”)

Example

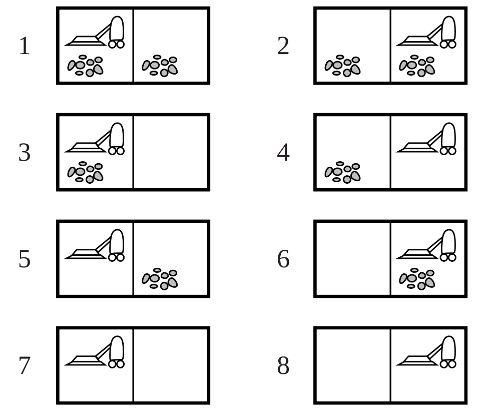
Take for vacuum cleaner:

- Percepts: location and contents, e.g., $[A, Dirty]$
- Actions: *Left*, *Right*, *Suck*, *NoOp*

What is the **right** way to organise the actions dependent on the percept history?

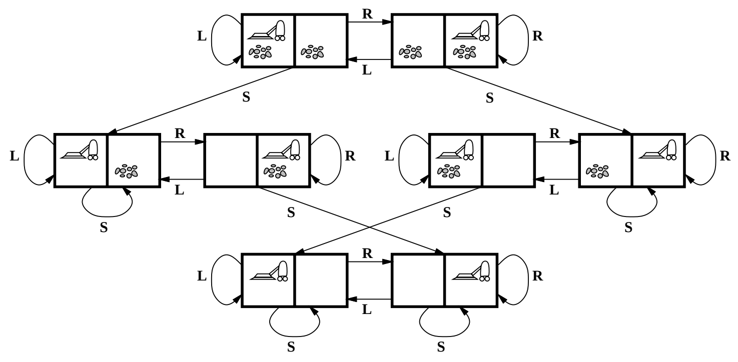
Example: vacuum world

What are the possible **states**, given two rooms?



Example: vacuum world

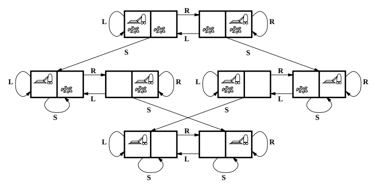
Single-state, start in #5. Solution??



Example: vacuum world

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

Conformant, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., *Right* goes to $\{2, 4, 6, 8\}$. Solution??

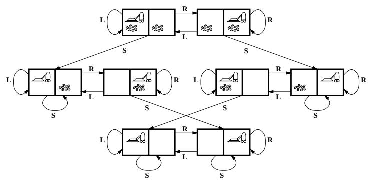


Example: vacuum world

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

Conformant, start in $\{1,2,3,4,5,6,7,8\}$
e.g., *Right* goes to $\{2,4,6,8\}$. Solution??
[*Right,Suck,Left,Suck*]

Contingency, start in #5
Murphy's Law: *Suck* can dirty a clean carpet
Local sensing: dirt, location only.
Solution??

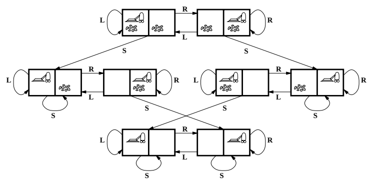


Example: vacuum world

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

Conformant, start in $\{1,2,3,4,5,6,7,8\}$
e.g., *Right* goes to $\{2,4,6,8\}$. Solution??
[*Right,Suck,Left,Suck*]

Contingency, start in #5
Murphy's Law: *Suck* can dirty a clean carpet
Local sensing: dirt, location only.
Solution??
[*Right*,**if** *dirt* **then** *Suck*]



Single-state problem formulation

A **problem** is defined by four items:

initial state e.g., "at Arad"

successor function $S(x)$ = set of action–state pairs
e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$

goal test, can be
explicit, e.g., x = "at Bucharest"
implicit, e.g., $NoDirt(x)$

path cost (additive)
e.g., sum of distances, number of actions executed, etc.
$c(x,a,y)$ is the step cost, assumed to be ≥ 0

A **solution** is a sequence of actions leading from the initial state to a goal state

Selecting a state space

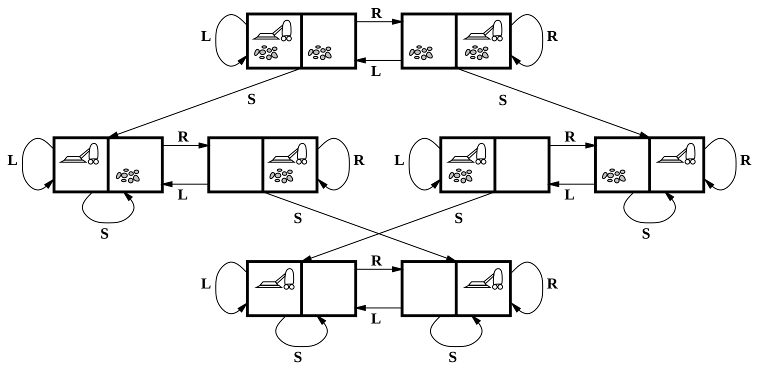
Real world is absurdly complex
⇒ state space must be **abstracted** for problem solving

(Abstract) state = set of real states

(Abstract) action = complex combination of real actions
e.g., "Arad → Zerind" represents a complex set
of possible routes, detours, rest stops, etc.
For guaranteed realizability, any real state "in Arad"
must get to **some** real state "in Zerind"

(Abstract) solution =
set of real paths that are solutions in the real world

Each abstract action should be "easier" than the original problem!

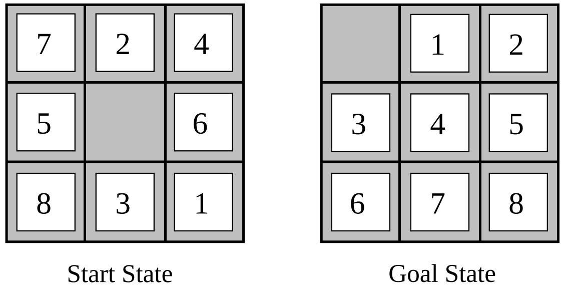


states??
actions??
goal test??
path cost??

Example: vacuum world state space graph

states??: integer dirt and robot locations (ignore dirt **amounts**)
actions??: *Left*, *Right*, *Suck*, *NoOp*
goal test??: no dirt
path cost??: 1 per action (0 for *NoOp*)

Example: The 8-puzzle

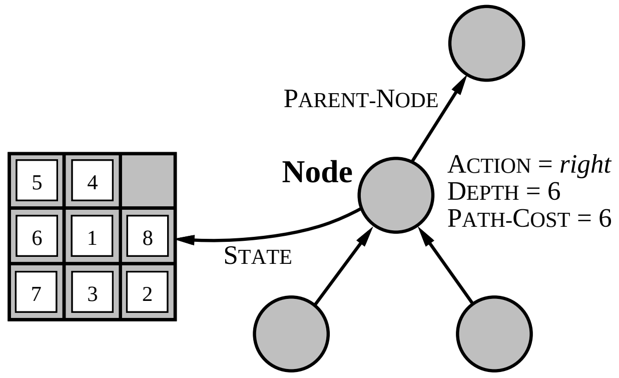


states??
actions??
goal test??
path cost??

Trees and states

A **state** is a (representation of) a physical configuration
A **node** is a data structure constituting part of a search tree
includes **parent**, **children**, **depth**, **path cost**
States do not have parents, children, depth, or path cost!

Example



The E   function creates new nodes, filling in the various fields and using the S   F   of the problem to create the corresponding states.

```
function T    -S       ( problem, fringe) returns a solution, or failure
  fringe ← I       (M    -N    (I       -S      [problem]), fringe)
  loop do
      if fringe is empty then return failure
      node ← R        -F      (fringe)
      if G    -T    [problem] applied to S     (node) succeeds return node
      fringe ← I        A  (E        (node, problem), fringe)
```

```
function E       ( node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in S            -F  [problem](S     [node]) do
      s ← a new N
      P       -N     [s] ← node;  A       [s] ← action;  S     [s] ← result
      P   -C     [s] ← P    -C     [node] + S    -C     (node, action, s)
      D       [s] ← D       [node] + 1
      add s to successors
  return successors
```

Search strategies

A strategy is defined by picking the **order of node expansion**

Strategies are evaluated along the following dimensions:
- completeness—does it always find a solution if one exists?
- time complexity—number of nodes generated/expanded
- space complexity—maximum number of nodes in memory
- optimality—does it always find a least-cost solution?

Time and space complexity are measured in terms of
- b—maximum branching factor of the search tree
- d—depth of the least-cost solution
- m—maximum depth of the state space (may be infinite)

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

Depth-limited search

Iterative deepening search

We'll look at these in the next lecture.

Summary

- Problem solving agents
- State spaces and search trees
- Components of general state space search algorithm