Artificial Intelligence 3e, Lecture 3.1

Presented by Yasin Ceran

September 17, 2024

Learning Objectives

At the end of the class you should be able to:

- define a directed graph
- represent a problem as a state-space graph

Searching

- Often we are not given an algorithm to solve a problem, but only a specification of what is a solution — we have to search for a solution.
- A typical problem is when the agent is in one state, it has a set of deterministic actions it can carry out, and wants to get to a goal state.
- Many AI problems can be abstracted into the problem of finding a path in a directed graph.
- Often there is more than one way to represent a problem as a graph.
- Searching in this context means finding a path to a goal node in a graph.

Example: Google Maps-I

The best route could mean

- the shortest (least distance) route
- the quickest route
- the route that uses the least energy
- the lowest-cost route, where the cost takes into account time, money (e.g., fuel and tolls), and the route's attractiveness.

What is a state?

A state needs to include enough information to

- determine what is the next state
- determine whether the goal is achieved
- determine the cost.

Often there are many options for what to include in the state.
Keep the states as simple as possible but no simpler.

State-space Problem

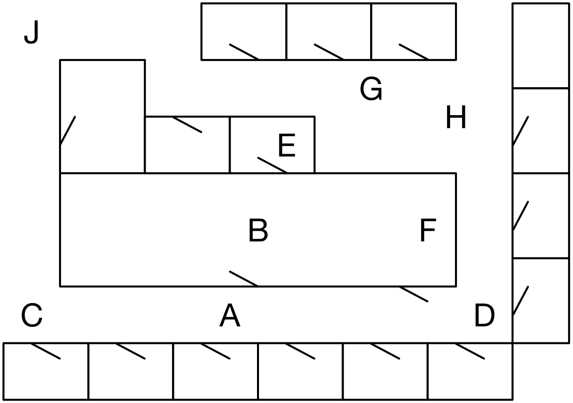
A state-space problem consists of

- a set of states
- a subset of states called the start states
- a set of actions
- an action function: given a state and an action, returns a new state
- a set of goal states, specified as function, $goal(s)$
- a criterion that specifies the quality of an acceptable solution.

A solution to a search problem is a sequence of actions that will get the agent from its current state to a state that satisfies the goal.

Example Problem for Delivery Robot

The robot is at A and the goal is to get to G:



Problem-Solving Agents

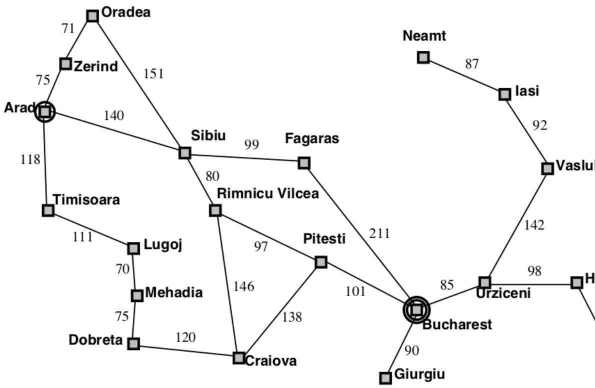
Restricted form of general agent:

```
function Simple-Problem-Solving-Agent( 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 ← Update-State(state, percept)
if seq is empty then
goal ← Formulate-Goal(state)
problem ← Formulate-Problem(state, goal)
seq ← Search( problem)
action ← Recommendation(seq, state)
seq ← Remainder(seq, state)
return action
```

Example: Romania

- current state:
  In Arad
- 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



Single-state problem formulation

A problem is defined by five components:

initial state e.g., "at Arad"

actions possible actions available to the agent
e.g., $\{Go(Sibiru), Go(Timisoara), Go(Zerind)\}$

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

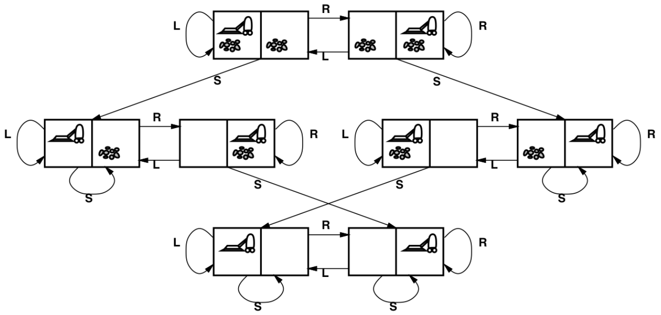
(Abstract) state = set of real states

(Abstract) action = complex combination of real actions
e.g., "Arad → Zerind" represents a complex set
of possible routes, detours, rest stops, etc.

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

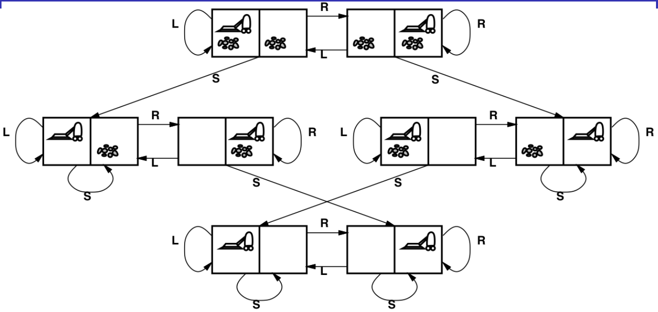
Each abstract action should be "easier" than the original problem!

Example: vacuum world state space graph



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

Example: vacuum world state space graph



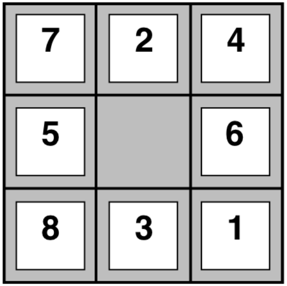
states??: integer dirt and robot locations (ignore dirt amounts etc.)

actions??: *Left*, *Right*, *Suck*, *NoOp*

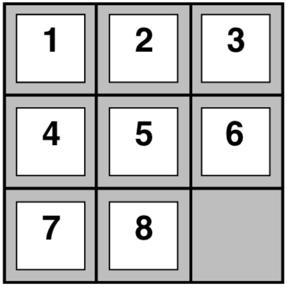
goal test??: no dirt

path cost??: 1 per action (0 for *NoOp*)

Example: The 8-puzzle



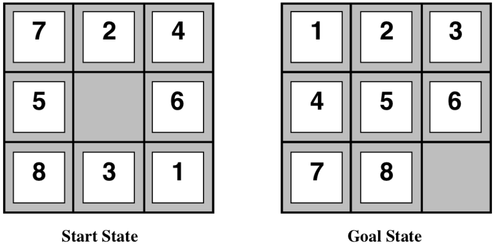
Start State



Goal State

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

Example: The 8-puzzle



states??: integer locations of tiles (ignore intermediate positions)

actions??: move blank left, right, up, down (ignore unjamming etc.)

goal test??: : = goal state (given)

path cost??: 1 per move

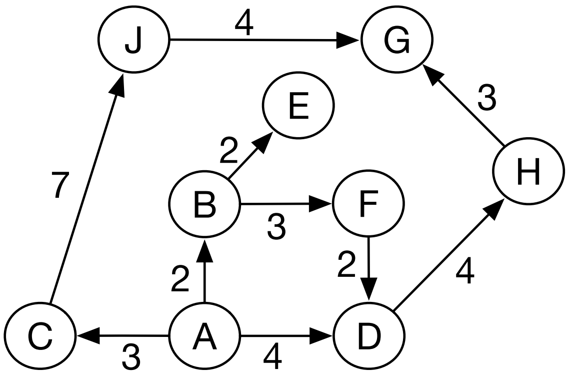
Directed Graphs

- Searching in graphs provides an appropriate abstract model of problem solving independent of a particular domain.
- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.
- A path is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ such that $\langle n_{i-1}, n_i \rangle \in A$.
- Given start nodes and goal nodes, a solution is a path from a start node to a goal node.
- When there is a cost associated with arcs, the cost of a path is the sum of the costs of the arcs in the path:

  $$cost(\langle n_0, n_1, \ldots, n_k \rangle) = \sum_{i=1}^{k} cost(\langle n_{i-1}, n_i \rangle)$$

  An optimal solution is one with minimum cost.

State-Space Graph for the Delivery Robot (Acyclic)

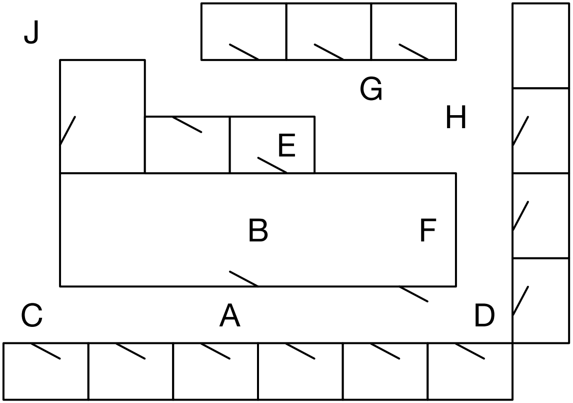


Directed Acyclic Graph (DAG)

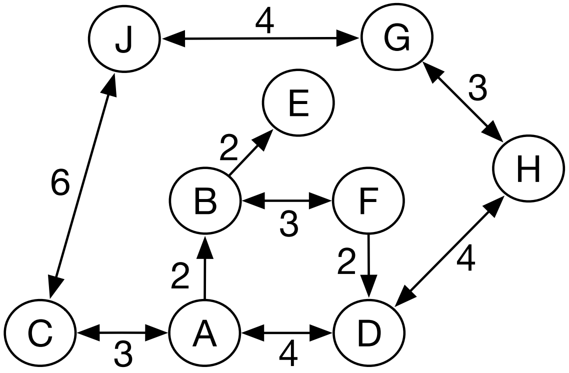
- A cycle is a non-empty path where the end node is the same as the start node – that is, $\langle n_1, n_2, , n_k\rangle$ such that $n_0 = n_k$
- A (directed) graph A directed graph without any cycles is called a directed acyclic graph (DAG).
- A tree is a DAG where there is one node with no incoming arcs and every other node has exactly one incoming arc.
- The node with no incoming arcs is called the root of the tree. A node with no outgoing arcs is called a leaf.
- In a tree, an arc goes from a parent to a child
- The forward branching factor of a node is the number of outgoing arcs from the node.
- The backward branching factor of a node is the number of incoming arcs to the node.

Example Problem for Delivery Robot

The robot is at A and the goal is to get to G:

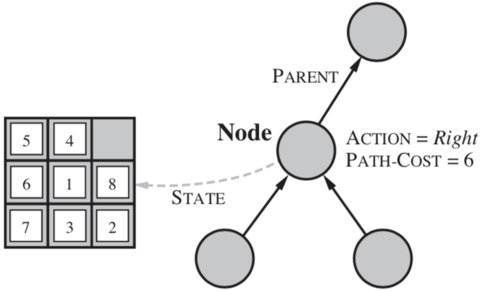


State-Space Graph for the Delivery Robot



Implementation: states vs. nodes

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 $g(x)$
States do not have parents, children, depth, or path cost!



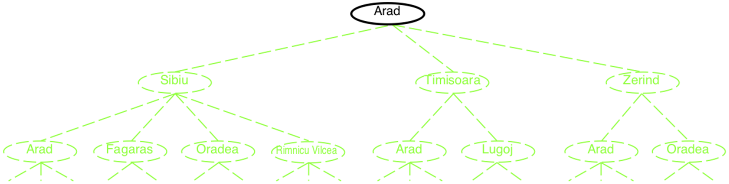
The Expand function creates new nodes, filling in the various fields and using the SuccessorFn of the problem to create the corresponding states.

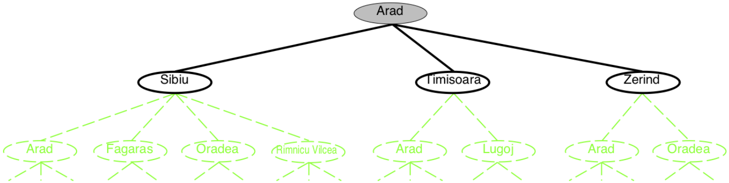
Tree search algorithms

Basic idea:
offline, simulated exploration of state space
by generating successors of already-explored states
(a.k.a. expanding states)

```
function Tree-Search( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

Tree search example



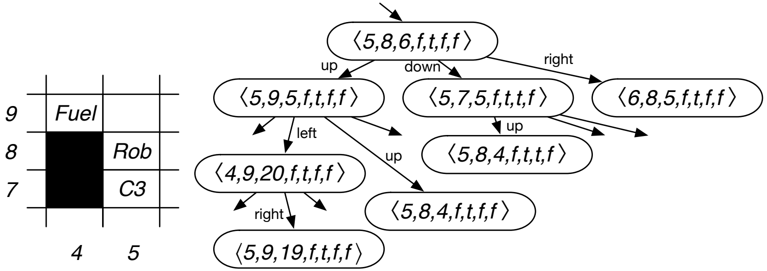


Partial Search Space for a Video Game

Grid game: Rob is on a grid and can move up, down, left or right and needs to collect coins C_1, C_2, C_3, C_4, without running out of fuel, and end up at location $(1, 1)$:

State: $< X\text{-}pos,\ Y\text{-}pos,\ \ Fuel, C_1,\ C_2,\ C_3,\ C_4 >$

Goal: $<$ 1, 1, ?, t, t, t, t$>$



9
8
7
4
5
Fuel
Rob
C3
⟨5,8,6,f,t,f,f⟩
up
down
right
⟨5,9,5,f,t,f,f⟩
⟨5,7,5,f,t,t,f⟩
⟨6,8,5,f,t,f,f⟩
left
up
up
⟨4,9,20,f,t,f,f⟩
⟨5,8,4,f,t,t,f⟩
⟨5,8,4,f,t,f,f⟩
right
⟨5,9,19,f,t,f,f⟩