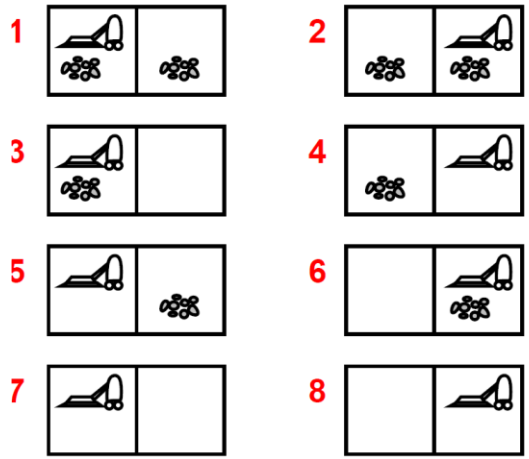
Problem solving Agents

Chapter 3

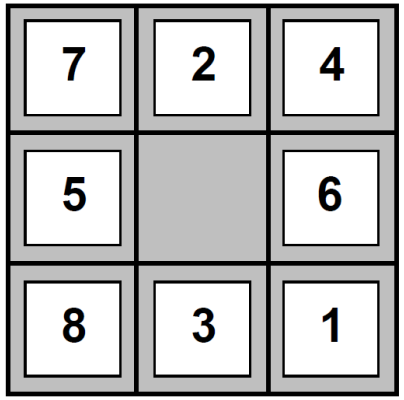
Outline

- Problem-solving agents
- Prob;em types
- Problem formulation
- Examples
- Problem solving as search

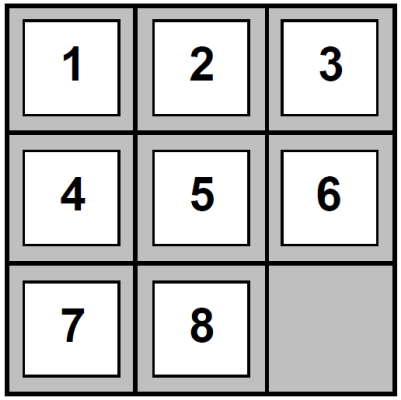
Example: vacuum world



Example: Puzzle-world

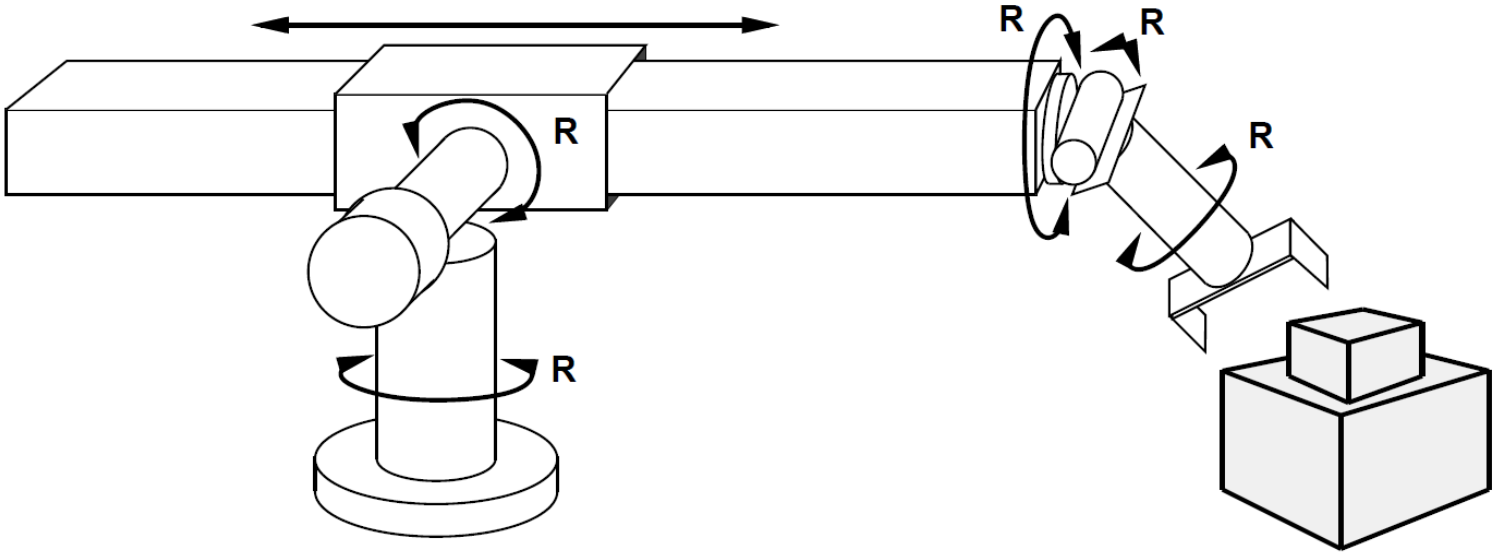


Start State

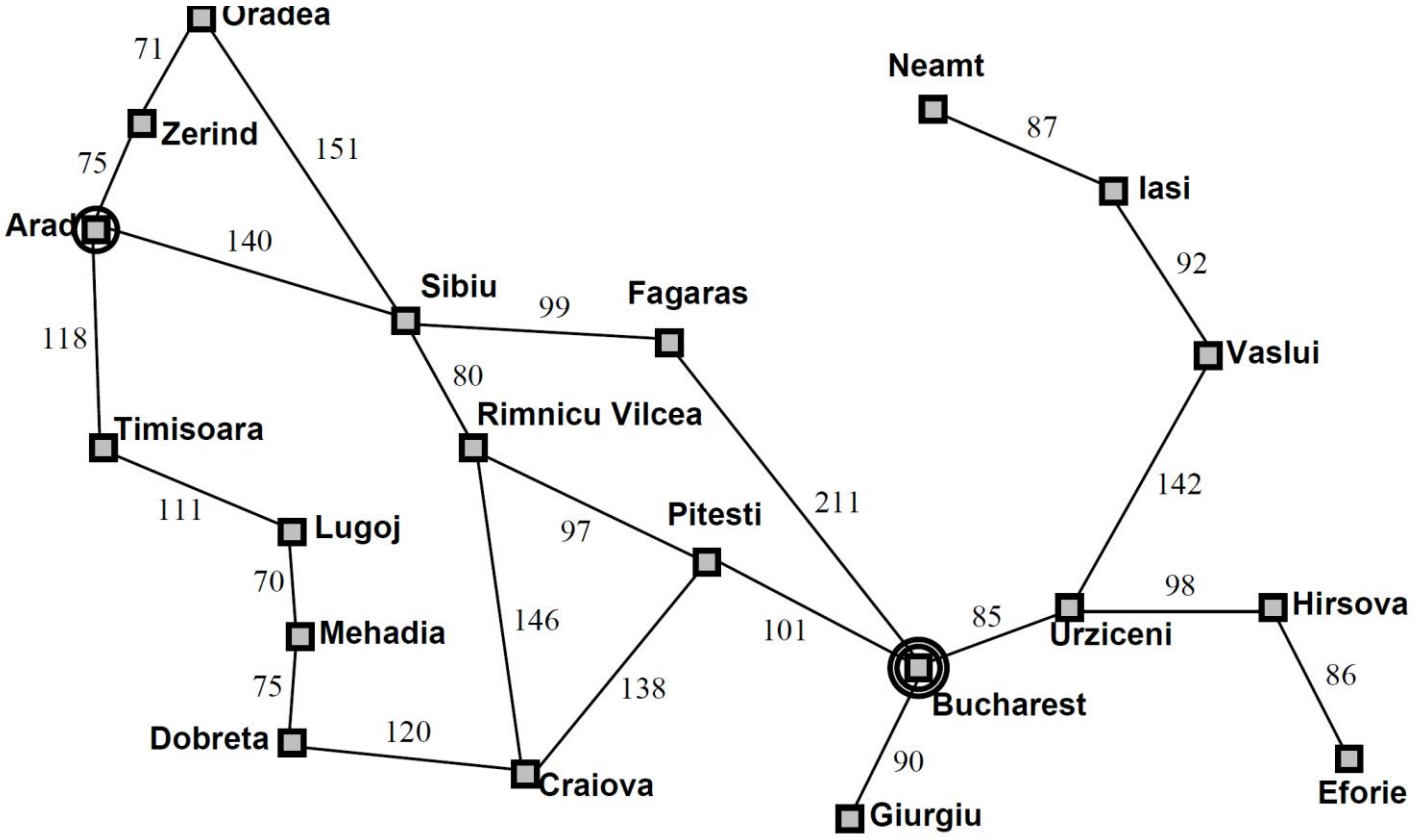


Goal State

Example: Robot-world



Example: Romania



Problem types

Deterministic, fully observable ⟹ single-state problem

Agent knows exactly which state it will be in; solution is a sequence

Non-observable ⟹ conformant problem

Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable ⟹ contingency problem

percepts provide new information about current state

solution is a contingent plan or a policy
often interleave search, execution

Unknown state space ⟹ exploration problem ("online")

Single-state problem formulation

A problem is defined by four items:

initialstate e.g.,"atArad"

successor function S(x) = set of action–state pairs
e.g., S(Arad) = { ⟨Arad → Zerind, Zerind⟩ , . . .}

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.

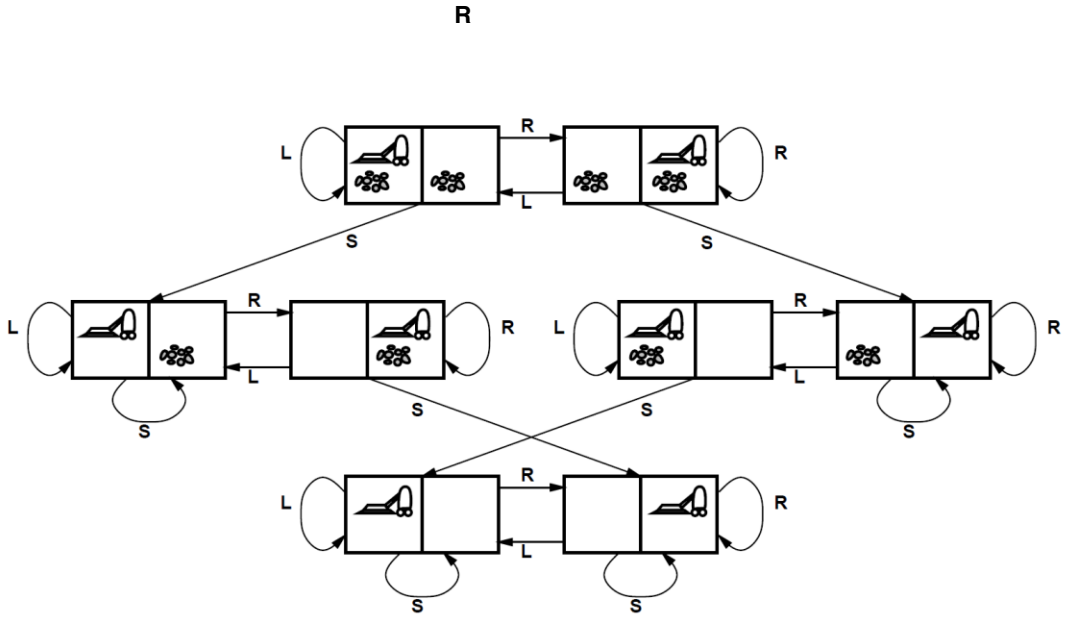
For guaranteed realizability, any real state "in Arad" must get to some real state "in Zerind"

(Abstract) solution =

set of real paths that are solutions in the real world

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

Example: vacuum world state space graph



tes??
ions??

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)

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 ∞)

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

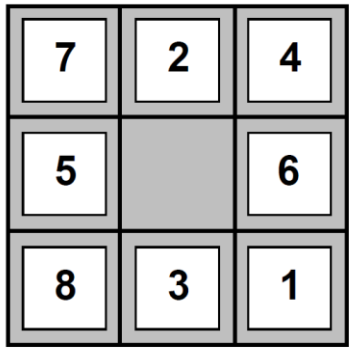
Uniform-cost search

Depth-first search

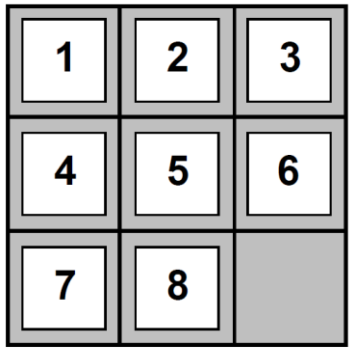
Depth-limited search

Iterative deepening search

Example: Puzzle-world



Start State



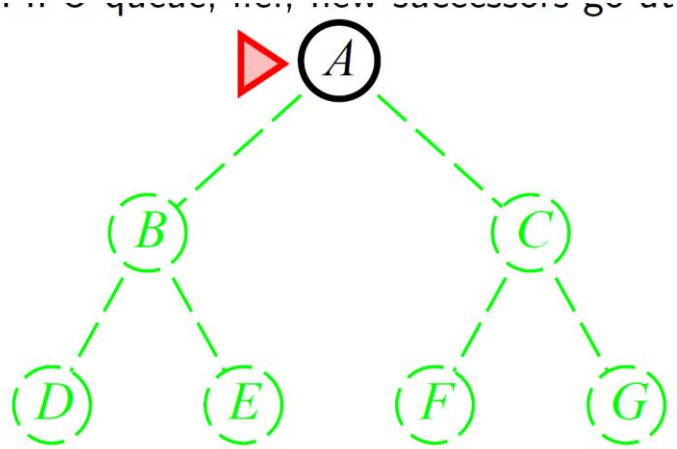
Goal State

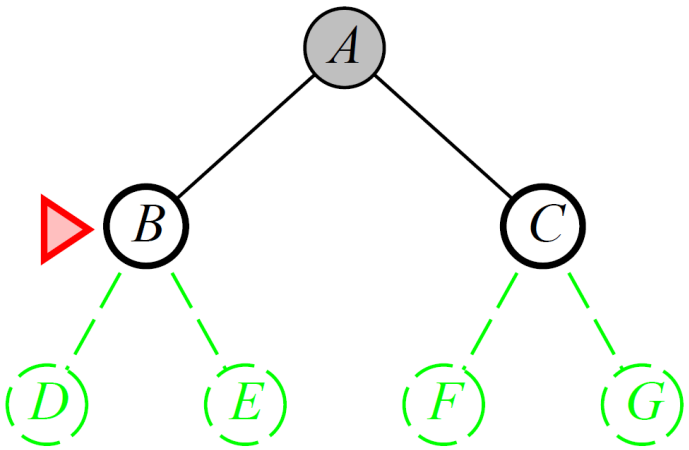
states??: integer locations of tiles (ignore intermediate positions)
actions??: move blank left, right, up, down (ignore unjamming etc.) goal test??: = goal state (given)
path cost??: 1 per move

[Note: optimal solution of n-Puzzle family is NP-hard]

Example: Breadth-first search

Expand shallowest unexpanded





Properties of breadth-first search

Complete?? Yes (if b is finite)

Time?? Exp. in depth d

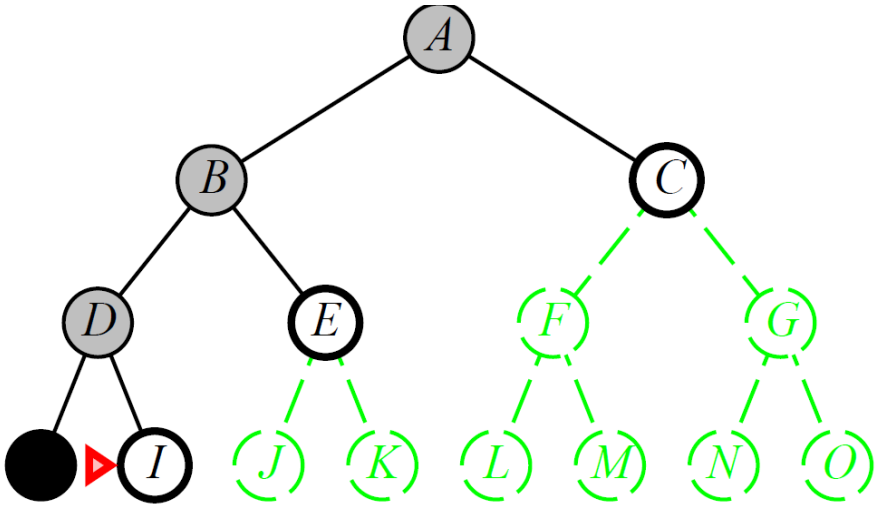
Space?? Keeps all "frontier" nodes in memory

Optimal?? Yes (if cost = 1 per step)

Space is the problem: can easily generate nodes at 100MB/sec

Depth-first search

Expand deepest unexpanded node



begin active:= {0}; (**comment**: ``0'' is a conventional starting point) C:= inf; currentbest:= anything; **while** active is not empty **do begin** remove first node k from active; (**comment**:

A General-Purpose Graph Search

```
begin
    active:= {0}; (comment: ``0'' is a conventional starting point)
    C:= inf;
    currentbest:= anything;
    while active is not empty do begin
        remove first node k from active; (comment: k is a branching no
        generate the children i=1,...,Nk of node k, and
            compute corresponding costs Ci and
                lower bounds on costs Li;
        for i = 1,...,Nk do
            if Li >= C then prune child i
            else begin
                if child i is a complete solution and Ci < C then begi
                        C:= Ci, currentbest:= child i;
                        prune nodes in active with lower bounds > than
                end
                insert child i to active (comment: active is sorted)
            end
    end
end
```

Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored

Variety of uninformed search strategies

Iterative deepening search uses only linear space
and not much more time than other uninformed algorithms

A problem-solving agent can use a general-purpose graph-search procedure

To Do

Think about this

What happens if the "Active" list is implanted as a FIFO queue?

What happens if the "Active" list is implemented as a LIFO queue?

What happens if the "Active" list is implemented as a priortised queue ordered by g(n), where g(n) is the cost of node n?

Use an LLM to obtain code for uninformed and informed search

Use the code to solve problems in the puzzle-world and in Romania