

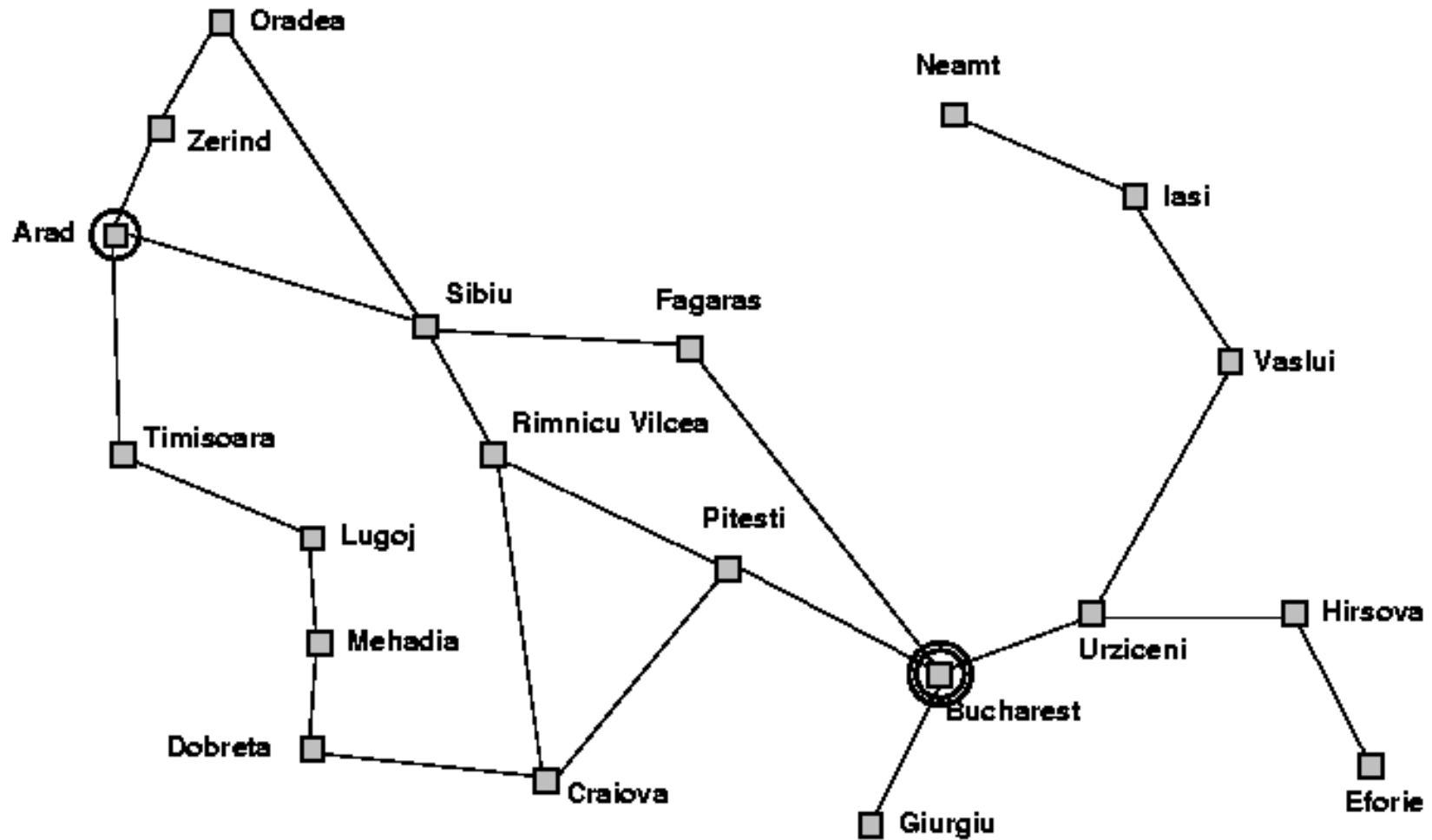
Search – Chapter 3

- ◇ **Problem-solving agents**
- ◇ **Problem types**
- ◇ **Problem formulation**
- ◇ **Example problems**
- ◇ **Basic search algorithms**

Moving beyond reflex agents, first example of how to string together a series of actions

Uninformed search

Map of Area



Goal Oriented Search – Ex:1

On holiday in Romania; currently in Arad.

Flight leaves tomorrow from Bucharest

Formulate goal:

be in Bucharest

Formulate problem:

states: various cities

operators: drive between cities

Find solution:

sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Four Step Process

- Goal formulation, problem formulation, search, execute
- What does this tell you about the environment?
- How are the percepts for $t > 1$ used?

Problem Types

Deterministic, fully observable \implies *single-state problem*

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

Non-observable \implies *conformant problem*

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

Nondeterministic and/or partially observable \implies *contingency problem*

percepts provide *new* information about current state

solution is a *tree* or *policy*

often *interleave* search, execution

Unknown state space \implies *exploration problem* ("online")

Single State Problem Definition

A *problem* is defined by four items:

initial state e.g., "at Arad"

operators (or *successor function* $S(x)$)

e.g., Arad \rightarrow Zerind Arad \rightarrow Sibiu etc.

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 operators executed, etc.

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

State Space Definition

Real world is absurdly complex

⇒ state space must be *abstracted* for problem solving

(Abstract) state = set of real states

(Abstract) operator = 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!

Eight State Puzzle

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

states??

operators??

goal test??

path cost??

Eight State Puzzle

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

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

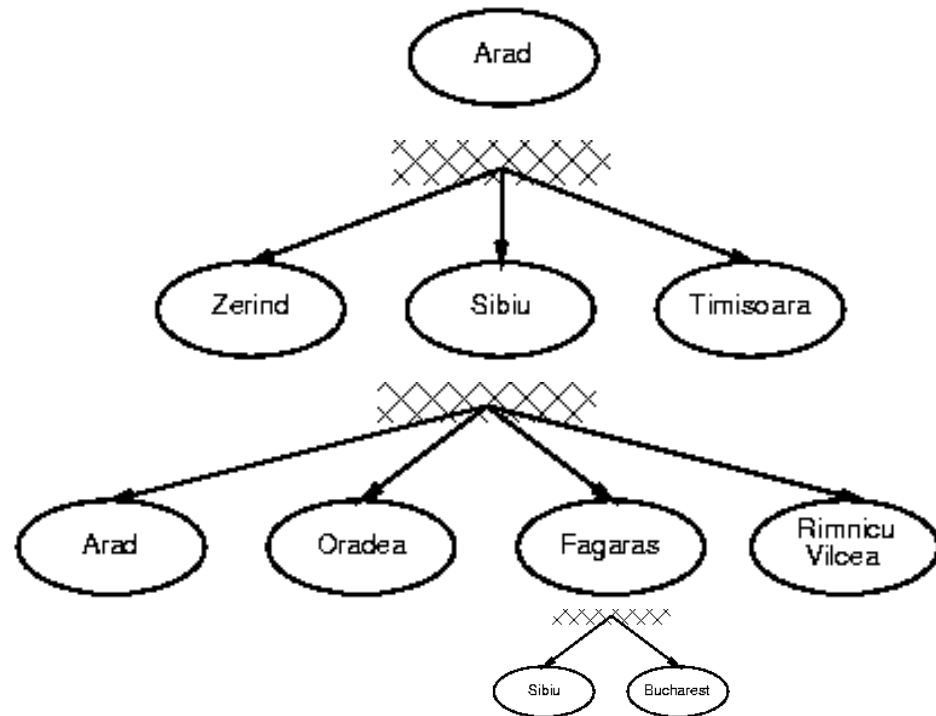
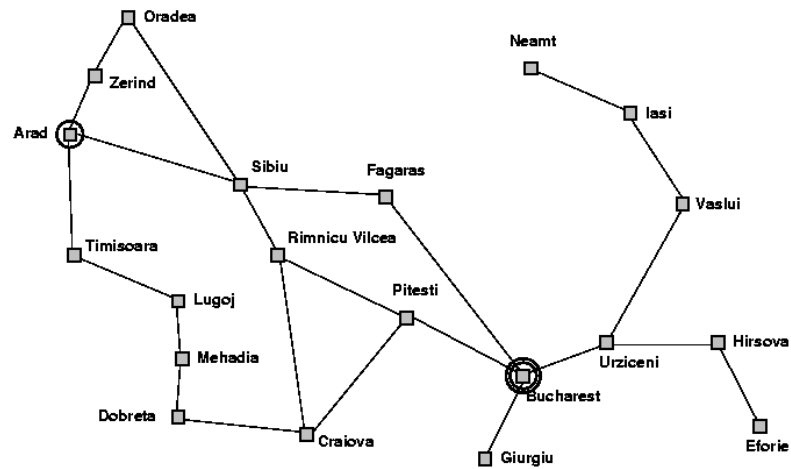
actions??: move blank left, right, up, down (ignore unjamming etc.)

goal test??: = goal state (given)

path cost??: 1 per move

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

Converting the Problem to a Graph



Basic Uninformed Search

Basic idea:

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

```
function GENERAL-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
```

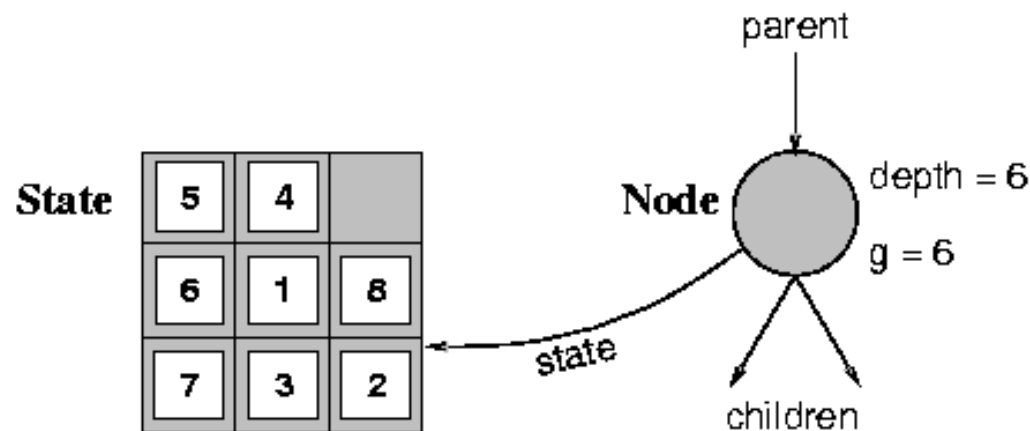
What is a State and a Node?

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! **Why?**



The **EXPAND** function creates new nodes, filling in the various fields and using the **OPERATORS** (or **SUCCESSORFN**) of the problem to create the corresponding states.

Search Strategies and Effectiveness

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 ∞)

Basic Search Algorithm

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure  
fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)  
loop do  
  if fringe is empty then return failure  
  node ← REMOVE-FRONT(fringe)  
  if GOAL-TEST[problem] applied to STATE(node) succeeds return node  
  fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND(node, problem) returns a set of nodes  
successors ← the empty set  
for each action, result in SUCCESSOR-FN[problem](STATE[node]) do  
  s ← a new NODE  
  PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result  
  PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)  
  DEPTH[s] ← DEPTH[node] + 1  
  add s to successors  
return successors
```

Uninformed Search

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

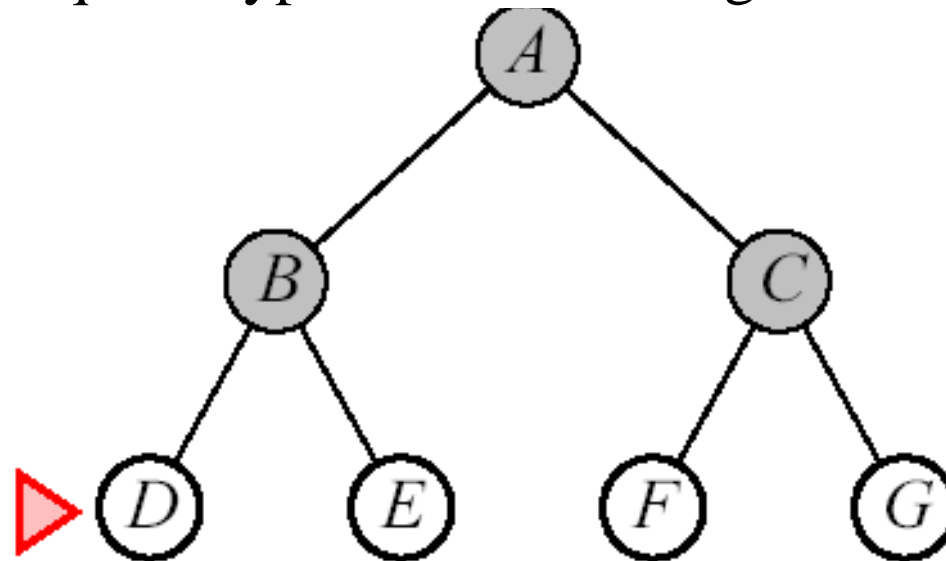
Breadth First Search

Expand shallowest unexpanded node

Implementation:

What queue type should the fringe be?

Complete?
Time?
Space?
Optimal?



Uniform Search

Expand least-cost unexpanded node

Implementation:

Fringe queue order?

Equivalent to breadth-first if step costs all equal

Complete?

Time?

Space?

Optimal?

Depth First and Depth Limited

Expand deepest unexpanded node

Implementation:

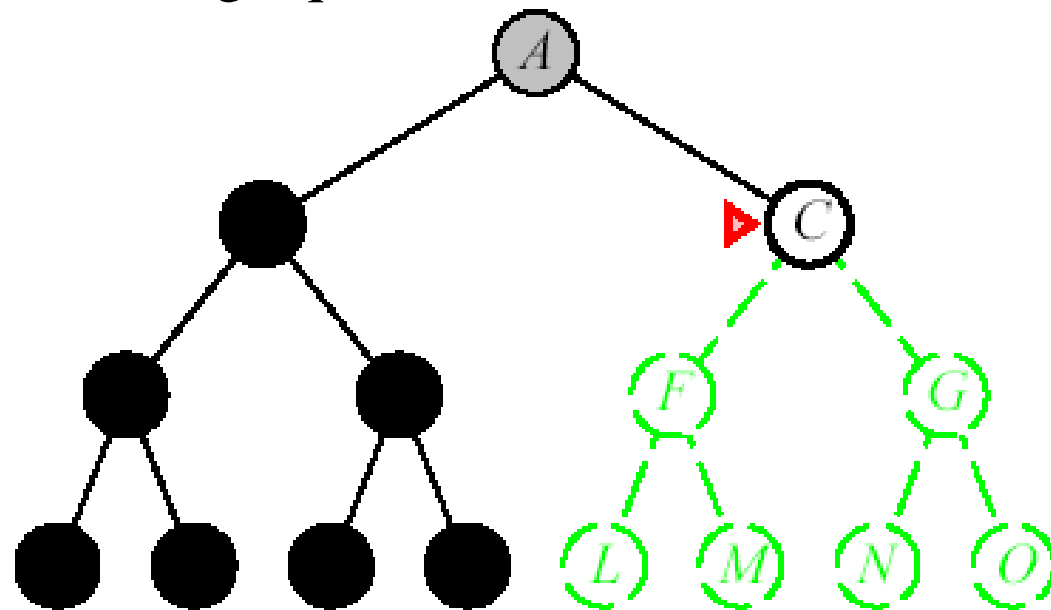
Complete?

Time?

Space?

Optimal?

Fringe queue order?

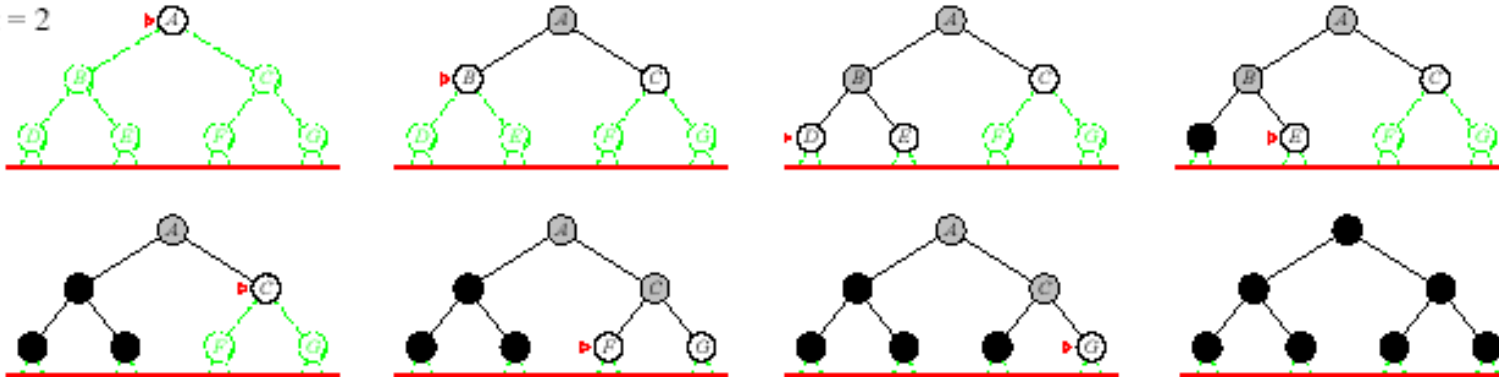


Iterative Deepening and Bi-directional Search

Limit = 1



Limit = 2



Complete?

Time?

Space?

Optimal?