Problem solving and search

Chapter 3

(Adapted from Stuart Russel, Dan Klein, and others. Thanks!)

Outline

♦ Problem-solving agents

♦ Problem types

♦ Problem formulation

♦ Example problems

♦ Basic search algorithms (the meat, 90%)

Problem-solving agents

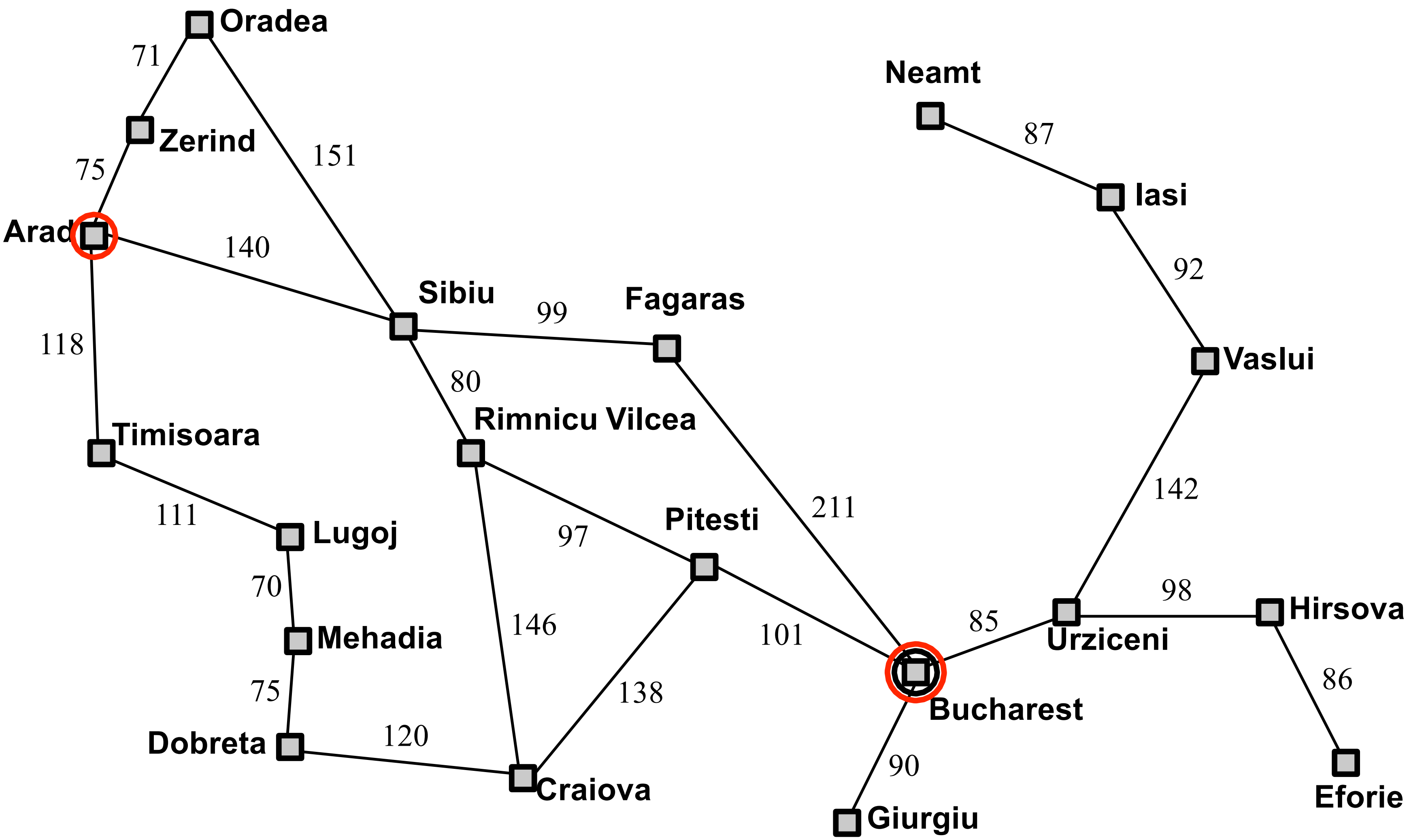
Simplified form of general agent:

```
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
```

Note: this is offline problem solving; solution executed "eyes closed."

Online problem solving different: uncertainty, incomplete knowledge, etc

Classic example: route-finding
(in Romania)



Search Gone Wrong?

Problem types

Deterministic, fully observable =⇒ single-state problem

- Agent knows exactly which state it will be in
- Solution is a simple sequence of actions

Non-observable =⇒ conformant problem

- Also known as “sensorless search”
- Agent may have no idea where it really is
- Solution (if any) is a sequence
- Surprisingly useful in many situations (simplifies state space for computing a “likely” solution quickly...which is adjusted during action

Nondeterministic and/or partially observable =⇒ contingency problem

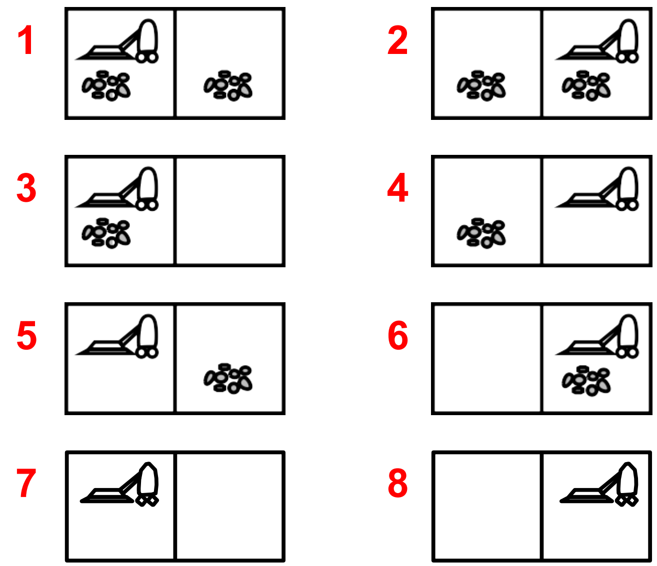
- percepts provide new information about current state
- solution is a contingent plan or a policy
- often interleave search, execution

Unknown state space =⇒ exploration problem

- Online” planning/re-planning

Example: vacuum world

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



Conformant, start in: ?
Solution??

Contingency, start in #5

- Murphy's Law: *Suck* can dirty a clean carpet
- Local sensing: dirt sensed in current location only.

Solution??

Single-state problem formulation

A problem is defined by four items:

1. initial state e.g., "at Arad"

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

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

4. 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 !!
$\Rightarrow$ state space must be abstracted for problem solving

(Abstract) state = set of real states

(Abstract) action = complex combination of real actions

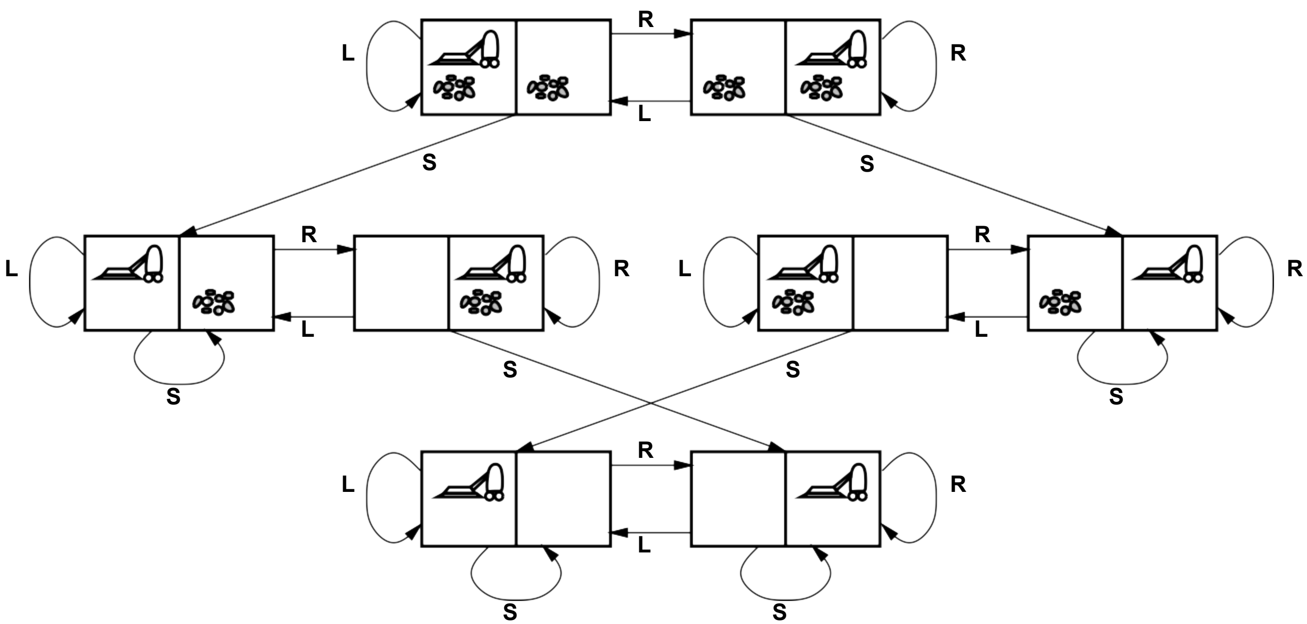
- e.g., “Arad $\rightarrow$ Zerind” represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realizability, any real state “in Arad”must get to some real state “in Zerind”

(Abstract) solution = set of simplified paths that..that can be translated to solutions in the real world

Leads to several definitions for quality of abstractions chosen:

- *Useful* abstraction: Each abstract action should be “easier” than the original problem!
- *Valid* abstraction: any abstract solution can be expanded to solution in real world

Example: vacuum world state space graph



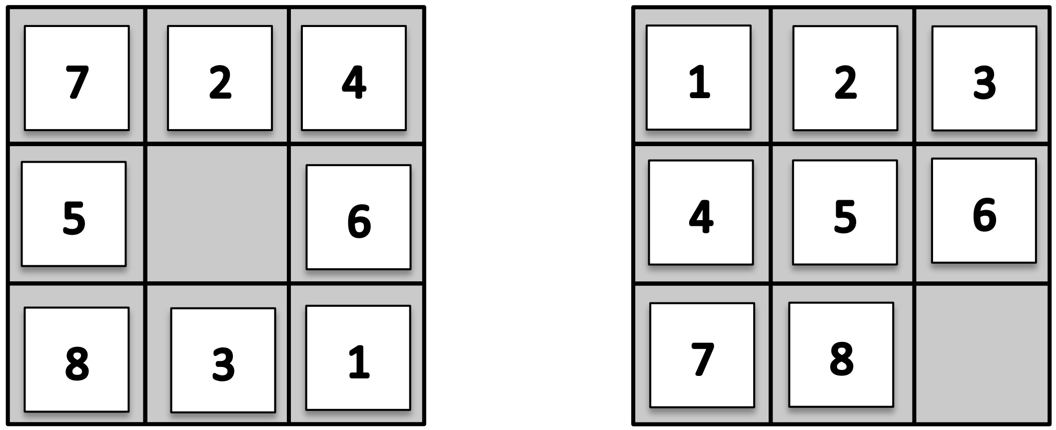
states??

actions??

goal test??

path cost??

Example: The 8-puzzle



Start State   Goal State

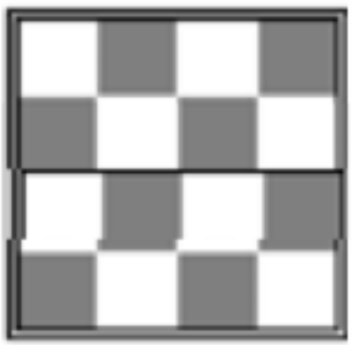
states??

actions??

goal test??

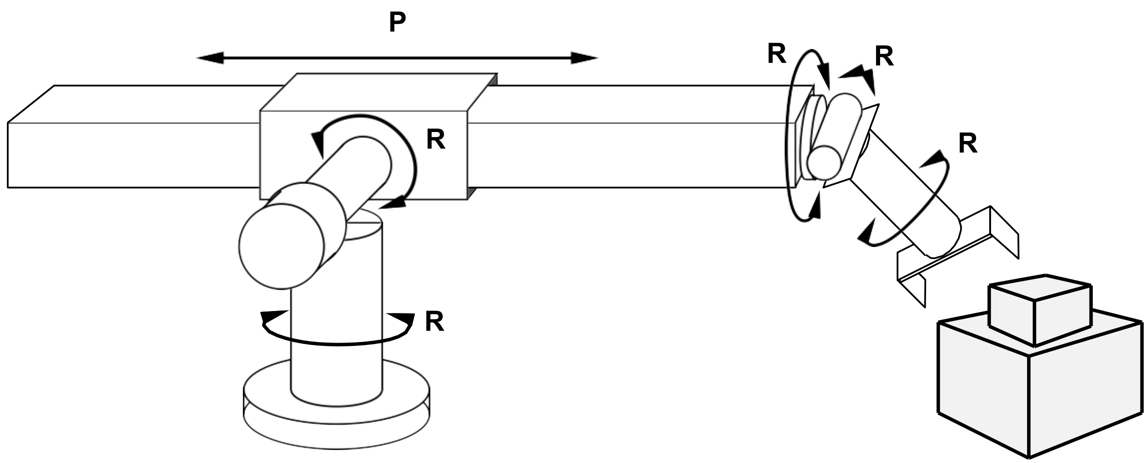
path cost??

Example: N-Queens



- What are the states?
- What is the start?
- What is the goal?
- What are the actions?
- What should the costs be?

Example: robotic assembly



states??:
- real-valued coordinates of robot joint angles
- parts of the object to be assembled (location, orientation)

actions??: continuous motions of robot joints

goal test??: complete assembly with no robot included!

path cost??: time to execute? Number of joints motions (wear and tear)?

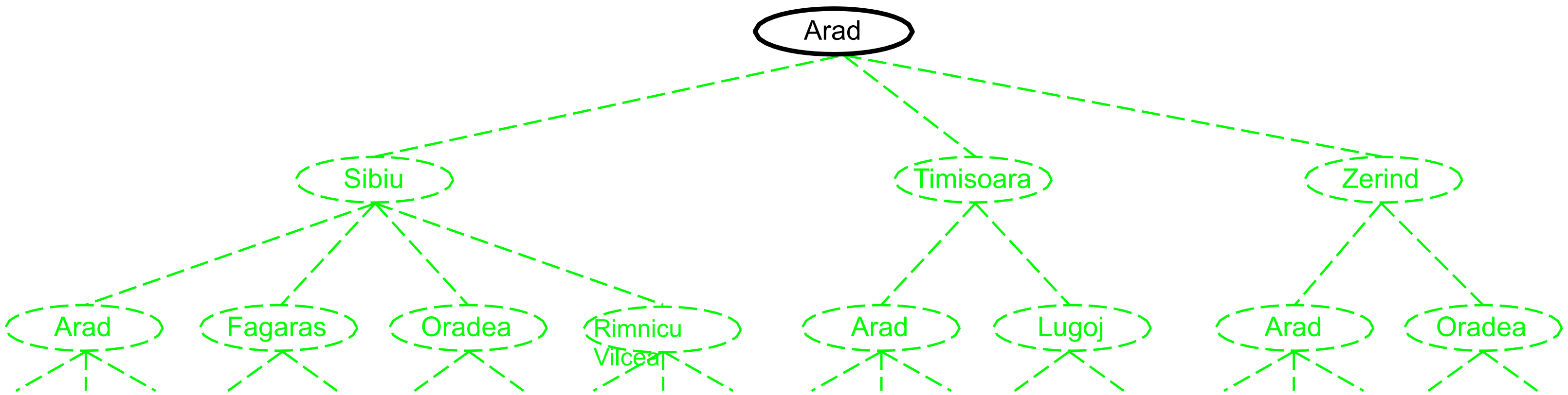
Tree search algorithms

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



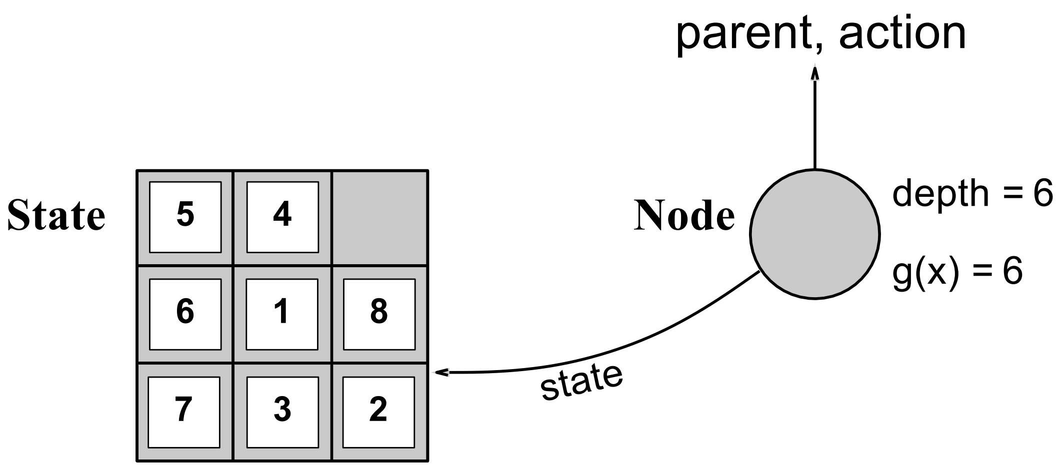
Concepts: states vs. nodes

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 → known as *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.

Implementation: general tree search

```
function Tree-Search( problem, frontier) returns a solution, or failure
    frontier ← Insert(Make-Node(Initial-State[problem]), frontier)  loop do
        if frontier is empty then return failure
        node ← Remove-Front(frontier)
        if Goal-Test(problem, State(node)) then return node  frontier ←
        InsertAll(Expand(node, problem), frontier)


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
```

Graph search

Q: What will happen is the search space is **not** a DAG? (a strict tree)

- Bi-directional arcs? (road can be driven both ways!)
- Cycles in the directional graph

```
function Graph-Search( problem, frontier) returns a solution, or failure
   closed ← an empty set
   frontier ← Insert(Make-Node(Initial-State[problem]), frontier)
   loop do
       if frontier is empty then return failure
       node ← Remove-Front(frontier)
       if Goal-Test(problem, State[node]) then return node
       if State[node] is not in closed then
           add State[node] to closed
           frontier ← InsertAll(Expand(node, problem), frontier)
   end
```

STOP FOR TODAY!

Search strategies

A strategy is defined by picking the order of node expansion

- Specifically: exact action of InsertAll() fn

Strategies are evaluated along the following dimensions:

- Completeness—
- time complexity—
- space complexity—
- Optimality—

Time and space complexity are measured in terms of

- b—
- d—
- m—

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

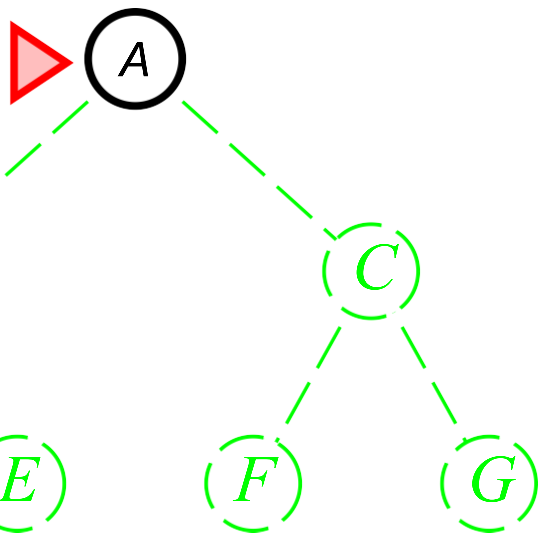
Breadth-first search

Plan: Always expand shallowest unexpanded node

- Shallowest = shortest path from root

Implementation:

frontier is a FIFO queue, i.e., new successors go at end



Properties of breadth-first search

Complete??

Time??

Space??

Optimal??

Uniform-cost search

Plan: Expand least-cost unexpanded node

- "least cost" = Having the lowest path cost
- Equivalent to breadth-first if step costs all equal

Implementation:

frontier = queue ordered by path cost, lowest first

Complete??

Time??

Space??

Optimal??

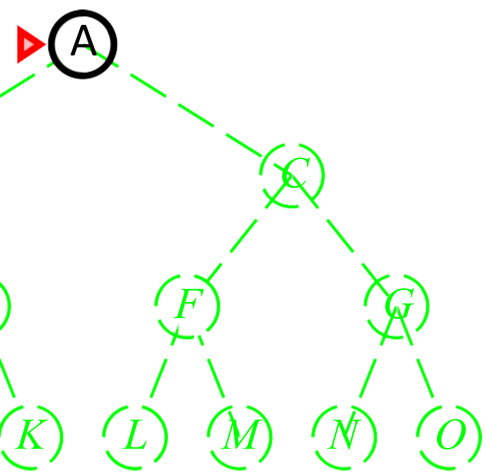
Depth-first search

Plan: Expand deepest unexpanded node

- Deepest= longest path from root

Implementation:

frontier = LIFO queue, i.e., put successors at front



Properties of depth-first search

Complete??

Time??

Space??

Optimal??

Depth-limited search

Plan: depth-first search with depth limit l,

- i.e., nodes at depth l have no successors

Recursive implementation:

```
function Depth-Limited-Search( problem, limit) returns soln/fail/cutoff
    Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit)

function Recursive-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if Goal-Test(problem, State[node]) then return node
    else if Depth[node] = limit then return cutoff
        else for each successor in Expand(node, problem) do  result ←
        Recursive-DLS(successor, problem, limit)  if result = cutoff then
        cutoff-occurred? ← true
        else if result /=  failure then return result
    if cutoff-occurred? then return cutoff else return  failure
```

Iterative deepening search

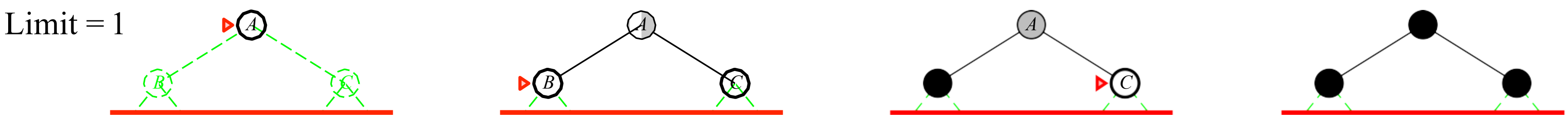
```
function Iterative-Deepening-Search( problem) returns asolution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← Depth-Limited-Search( problem, depth)
        if result ≠ cutoff then return result
    end
```

Iterative deepening search $l = 0$

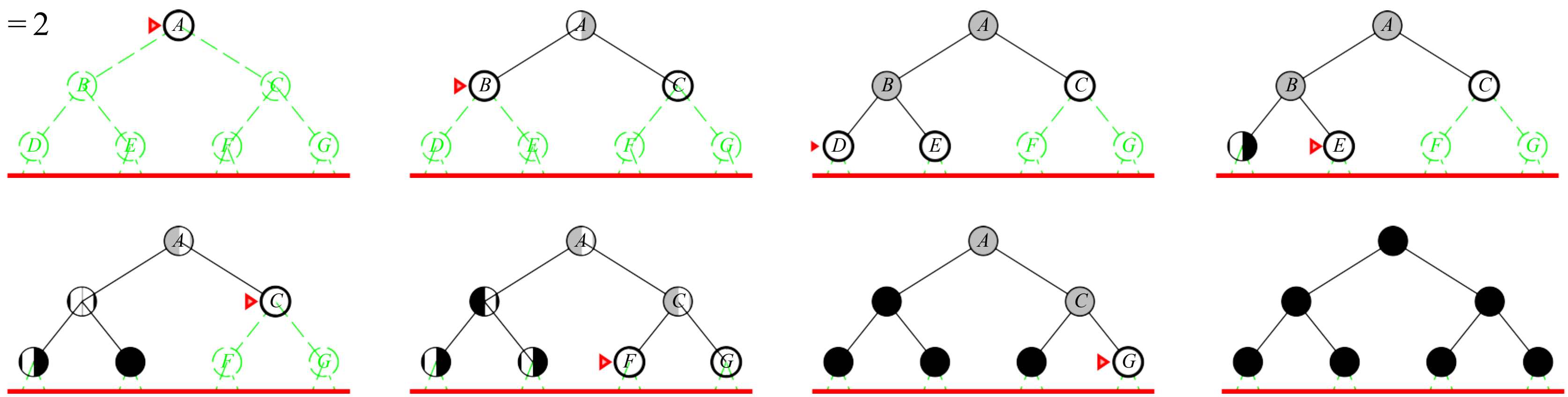
Limit = 0

Iterative deepening search $l = 1$



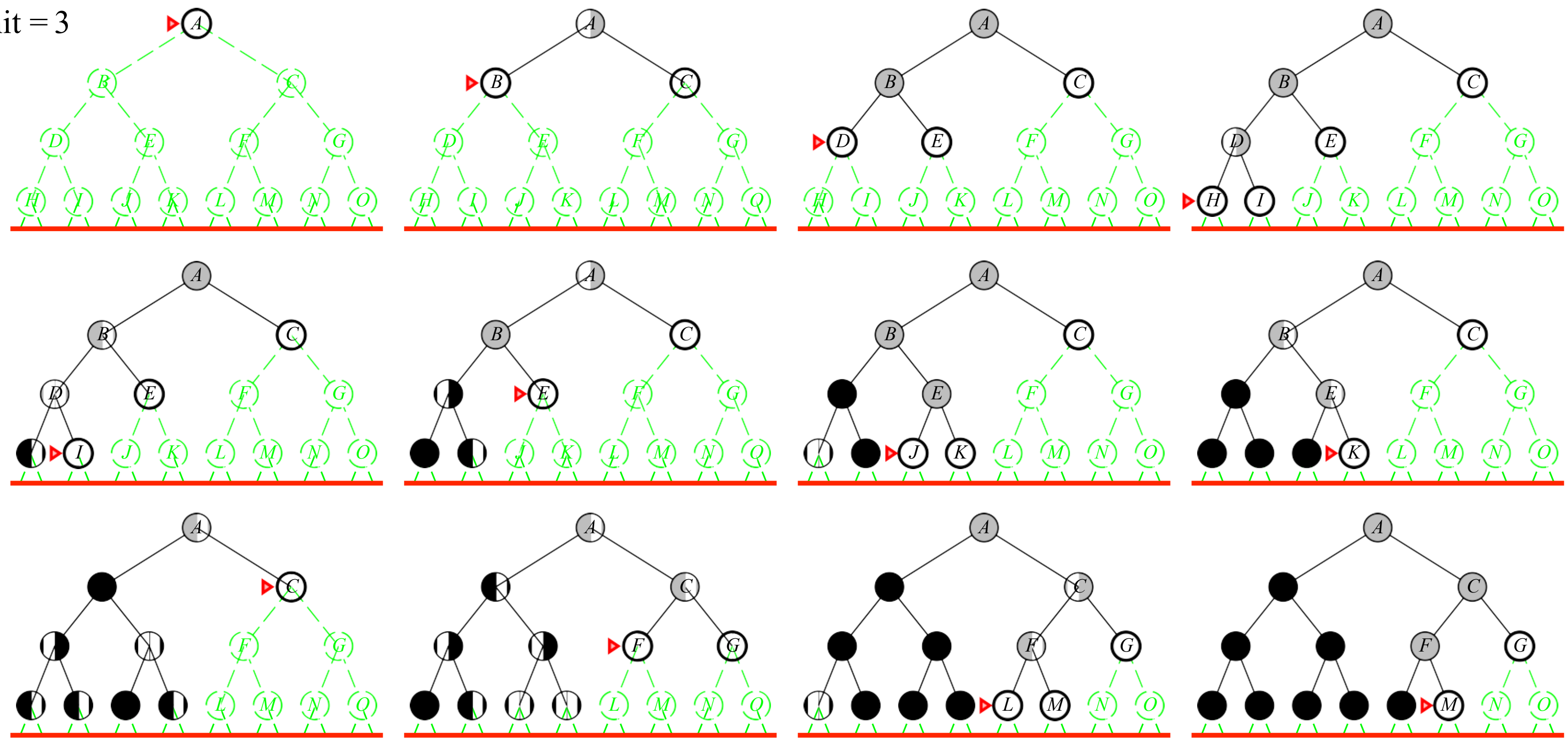
Iterative deepening search *l* = 2

Limit = 2



Iterative deepening search $l = 3$

Limit = 3



Properties of iterative deepening search

Complete??

Time??

Space??

Optimal??

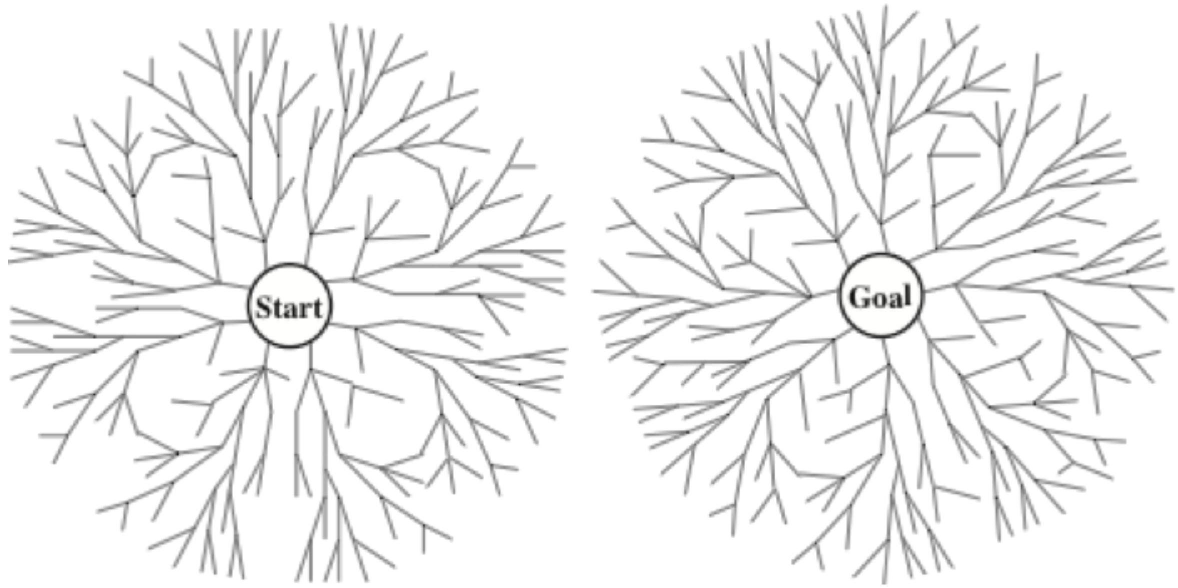
Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}450$$
$$N(\text{BFS}) = 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 + 999{,}990 = 1{,}111{,}100$$

Bi-Directional Search

Plan: Standard BFS…but search from both start and goal state

- Goal test: success when they meet (intersect of frontiers)



Advantages:

Concerns:

Summary of uninformed algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Legend:

- b = branching factor
- d= depth of shallowest solution
- m = maximum depth of tree
- l = depth limit

Superscripts:

a = complete if b is finite
b = complete if step costs > 0
c = optimal if step costs all identical
d = if both directions use breadth-first