Artificial Intelligence

Lecture 2

Ph. D. Lect. Horia Popa Andreescu

2017-2018 3rd year, semester 5

ROADMAP

- Uninformed search (Blind search)
  - Are the search strategies in which we have no additional information about states, beyond that provided in the problem definition
  - Breadth-first search
  - Dept-first search
  - Uniform-cost search
  - Depth-limited search: iterative deepening depth-first search
  - Bidirectional search
  - Graph search

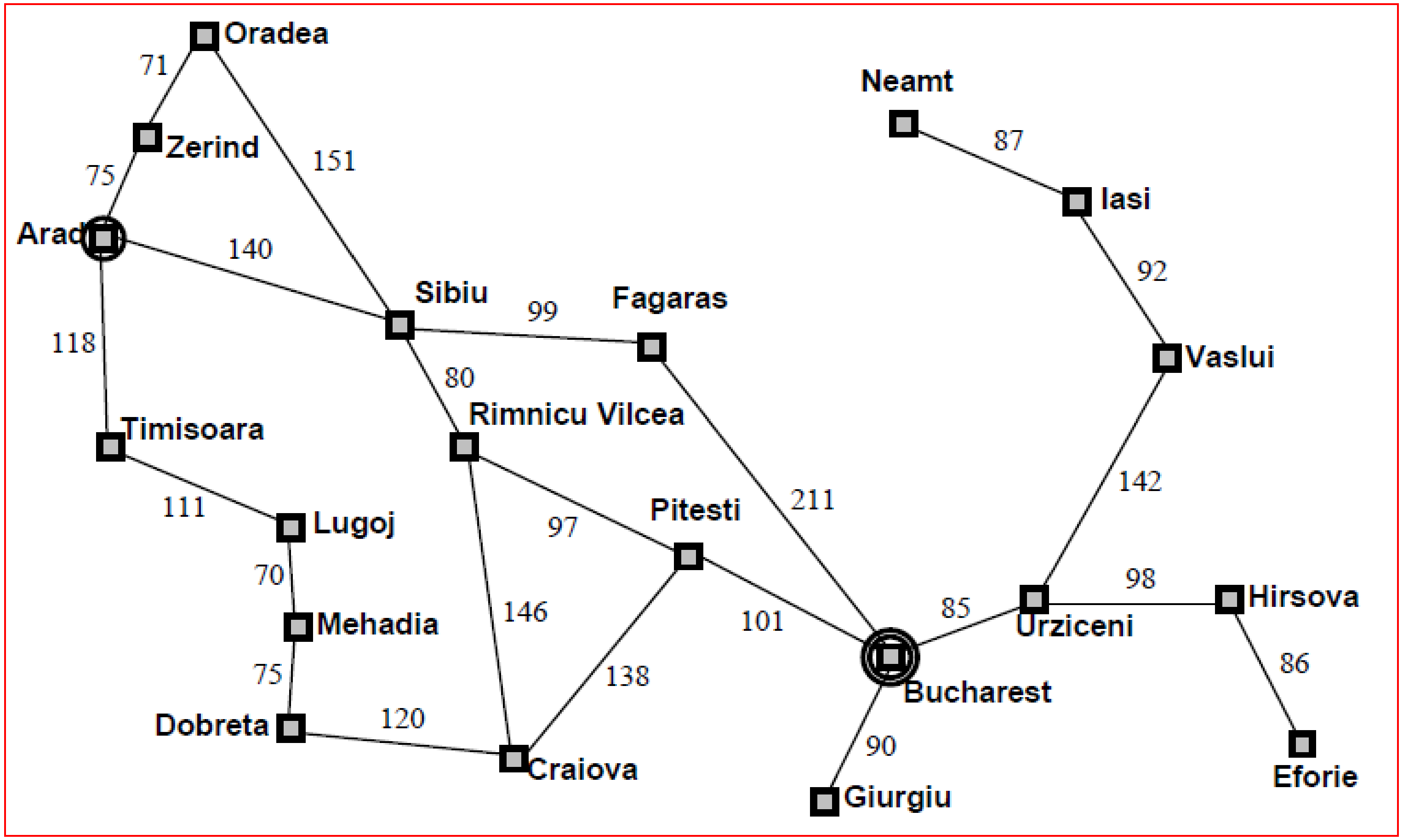
Problem representation

- We can see problem solving as a transition
  - starting from the initial state
  - Use transitions or operations to pass to different states
  - Reach a final state called the goal
- The most natural representation of the problem space is by a tree. A certain state (node) is connected to its parent state and to its children states which are all the states in which we can get by applying the operators on that current state.

Example (R-3-5): 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 (R-3-6)


Oradea
Neamt
71
Zerind
151
87
Iasi
75
Arad
140
92
Sibiu
99
Fagaras
118
Vaslui
80
Rimnicu Vilcea
Timisoara
142
111
211
Lugoj
97
Pitesti
70
98
Mehadia
146
101
85
Hirsova
Urziceni
86
75
138
Bucharest
Dobreta
120
90
Craiova
Eforie
Giurgiu

Problem types (R-3-7):

- 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")

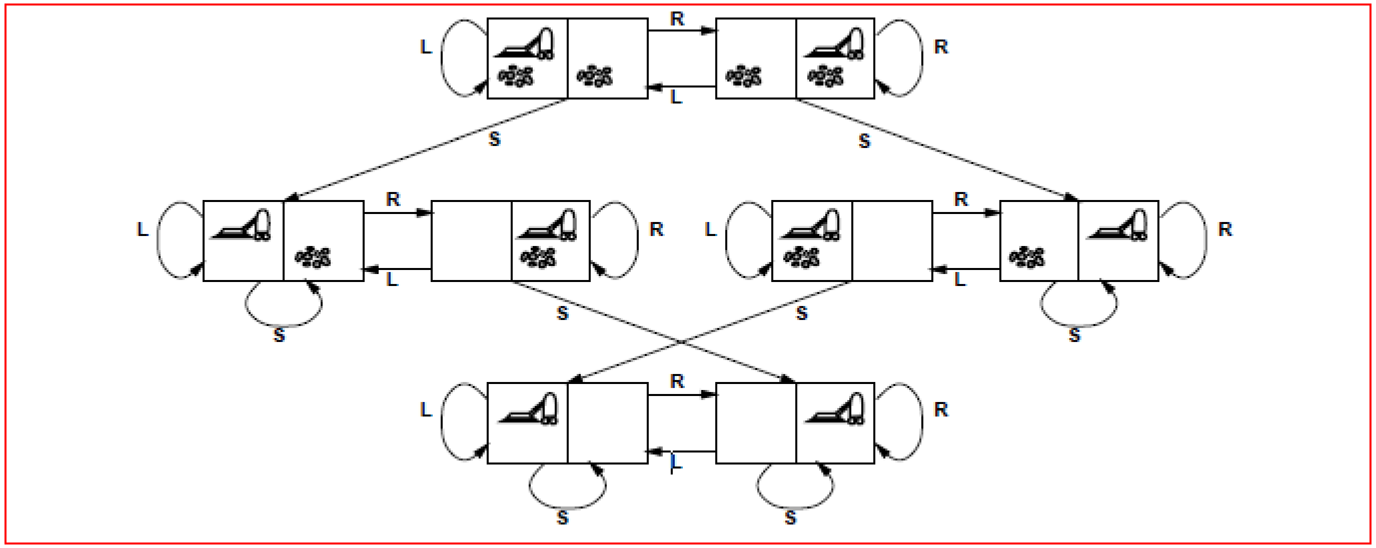
SINGLE-STATE PROBLEM FORMULATION (R-3-12)

- A problem is dened by four items:
  - initial state e.g., \at Arad"
  - 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

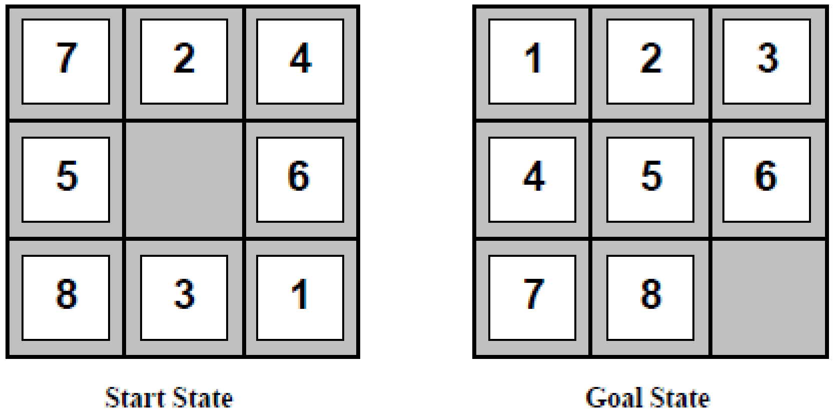
    leading from the initial state to a goal state

EXAMPLE: VACUUM WORLD STATE SPACE GRAPH (R-3-15)



- States: 0 and 1 dirt and robot locations, ignore dirt amounts
- Actions: Left, Right, Suck, NoOp
- Goal test: no dirt
- Path cost: 1 per action (o for NoOp)

Example: The 8-puzzle (R-3-23)



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]

Uninformed search strategies

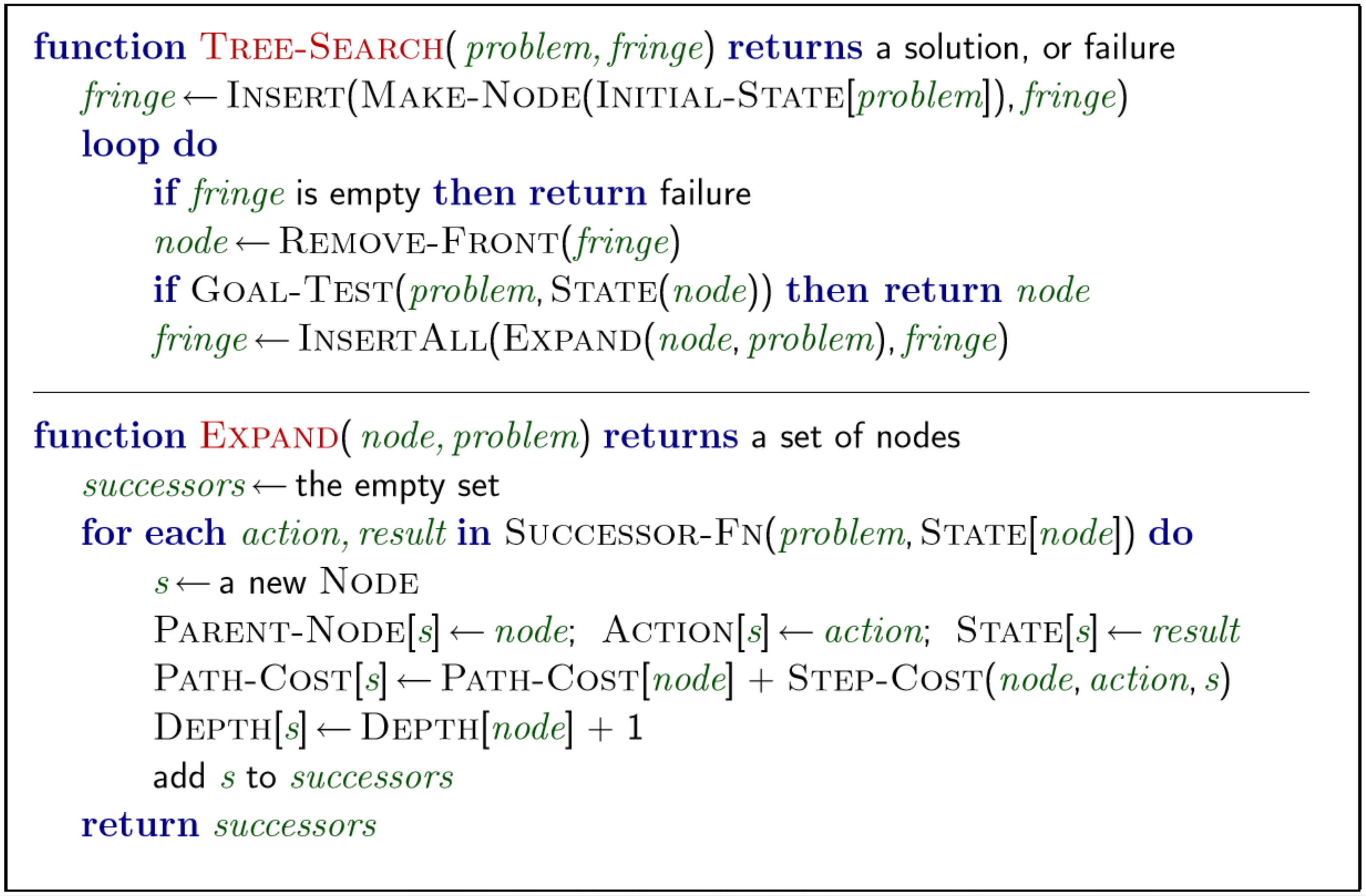
- Uninformed search strategies use only the information available in the problem definition.
- Examples of such algorithms:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

General prototype for the search algorithms R-3-25

- The tree search algorithm can be the pattern from which all the search algorithms can be derived
- The ideea is to explore the state space generating successors of already explored states. If by that we obtain a goal node then we have a solution

```
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
```

General Tree-search algorithm (R-3-30)



```
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, STATE(node)) then return node
        fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

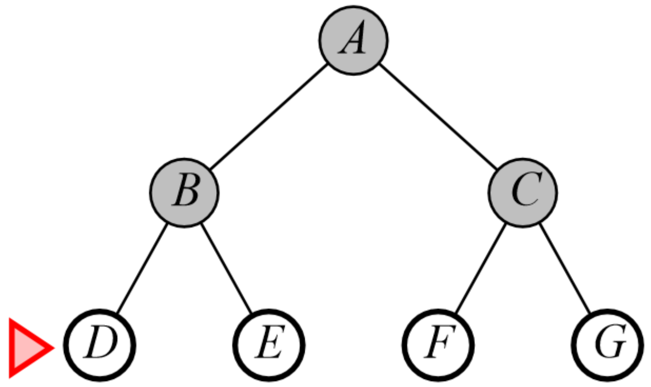
```
function EXPAND(node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node;   ACTION[s] ← action;   STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(node,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

Search strategies (R-3-31)

- 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 ∞)

BREADTH-FIRST SEARCH

- Expand the shallowest unexpanded node
- The fringe is a FIFO queue, the successors go at the end of the queue



Properties of Breadth-First search (R-3-41)

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

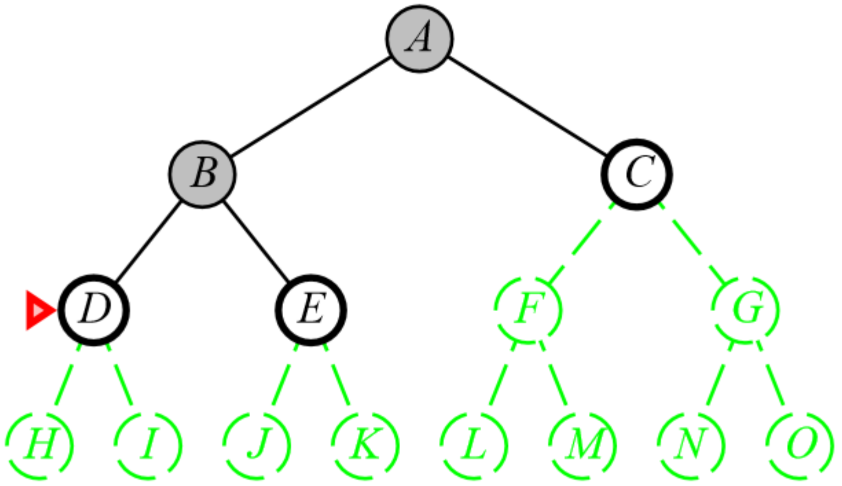
Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? Yes (if cost $=$ 1 per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs $=$ 8640GB.

Dept-first search (R-3-43)

- Expands deepest unexpanded node
- The fringe is a LIFO queue, the successors sre put at the front of the queue
- Backtracking is a varaint of DFS, but not the same!



Properties of the Depth-first search (R-3-59)

Complete?? No: fails in infinite-depth spaces, spaces with loops

Modify to avoid repeated states along path

$\Rightarrow$ complete in finite spaces

Time?? $O(b^m)$: terrible if m is much larger than d

but if solutions are dense, may be much faster than breadth-first

Space?? $O(bm)$, i.e., linear space!

Optimal?? No

Uniform Cost Search (R-3-42)

- Expand least-cost unexpanded node
- Implementation: fringe = queue ordered by path cost, lowest first
- Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

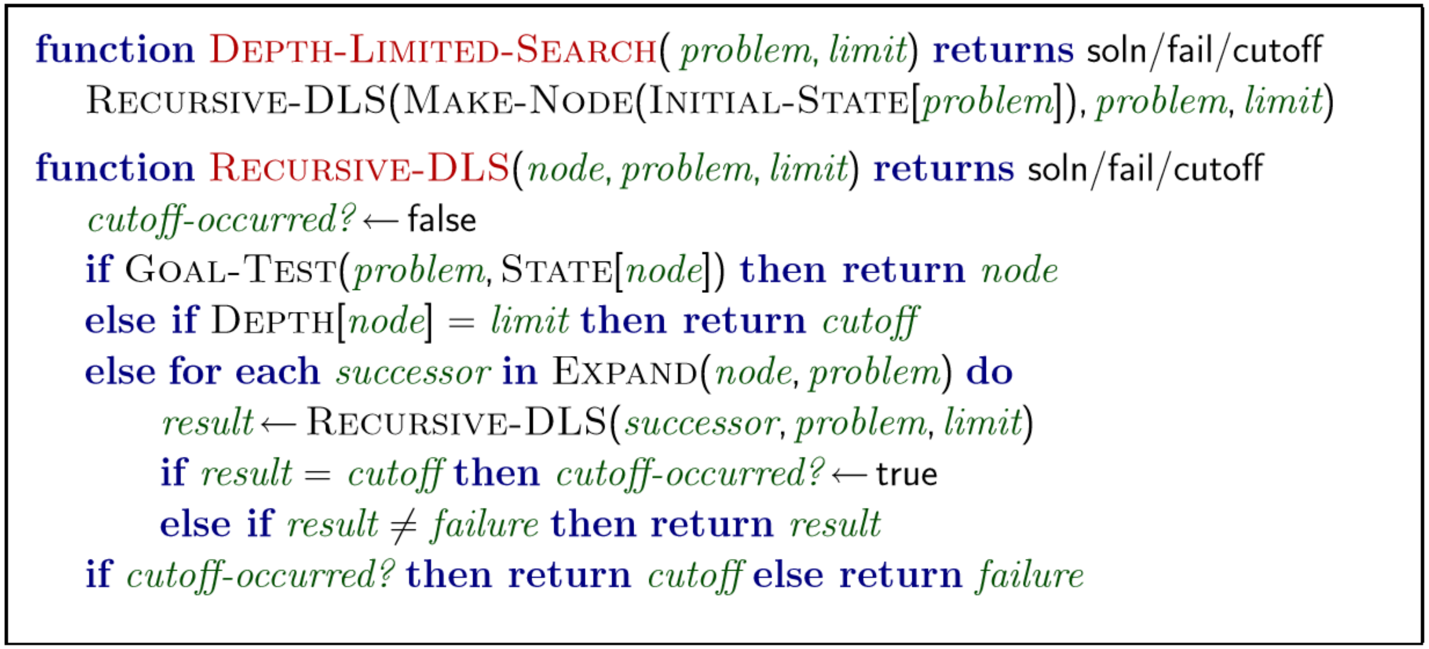
Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

Optimal?? Yes—nodes expanded in increasing order of $g(n)$

Depth-limited search (R-3-60)

- Is a dept-first search with a depth limit = l
  - i.e., nodes at depth l have no successors

Recursive implementation:

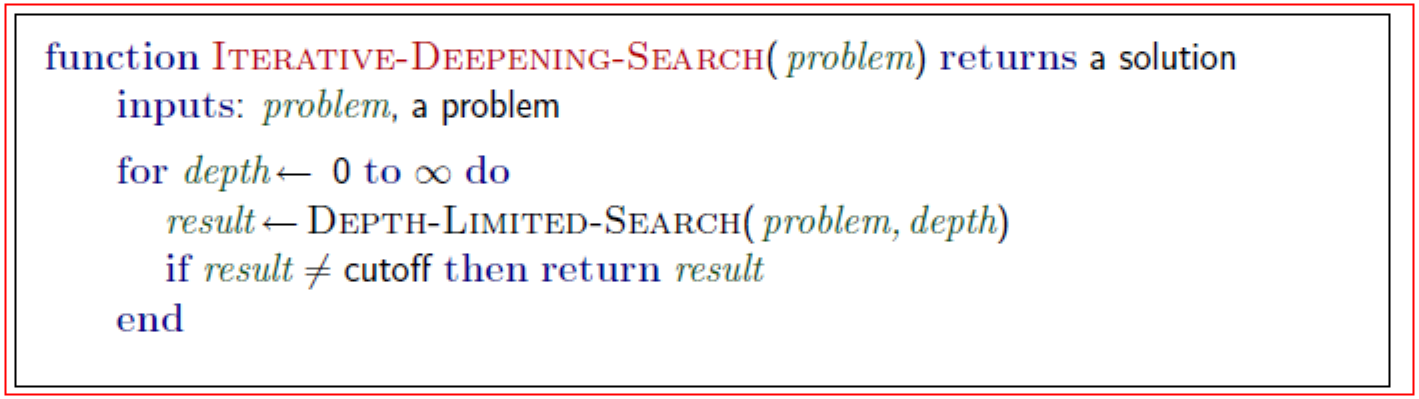


```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

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 DEPTH-FIRST SEARCH (R-3-70)

- It is a depth-limited search that progressively increases the depth if a solution is not found



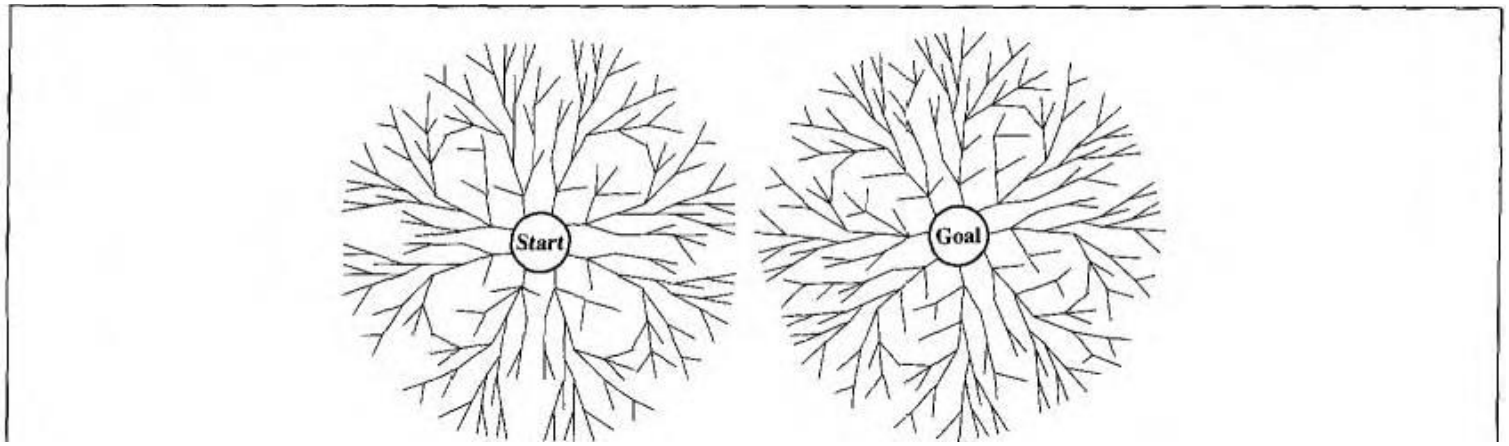
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to $\infty$ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result $\neq$ cutoff then return result
    end
```

- Complete?? Yes
- Time?? (d + 1)b0 + db1 + (d □ 1)b2 + : : : + bd = O(bd)
- Space?? O(bd)
- Optimal?? Yes, if step cost = 1
  Can be modified to explore uniform-cost tree
- Numerical comparison for b = 10 and d = 5, solution at far right leaf:
  N(IDS) = 50 + 400 + 3; 000 + 20; 000 + 100; 000 = 123; 450
  N(BFS) = 10 + 100 + 1; 000 + 10; 000 + 100; 000 + 999; 990 = 1; 111; 100
- IDS does better because other nodes at depth d are not expanded
- BFS can be modified to apply goal test when a node is generated

Bidirectional Search (RN-80)

- The idea is to run 2 searches simultaneously:
  - One forward from the original state
  - One backward from the goal
- We stop when the 2 searches meet

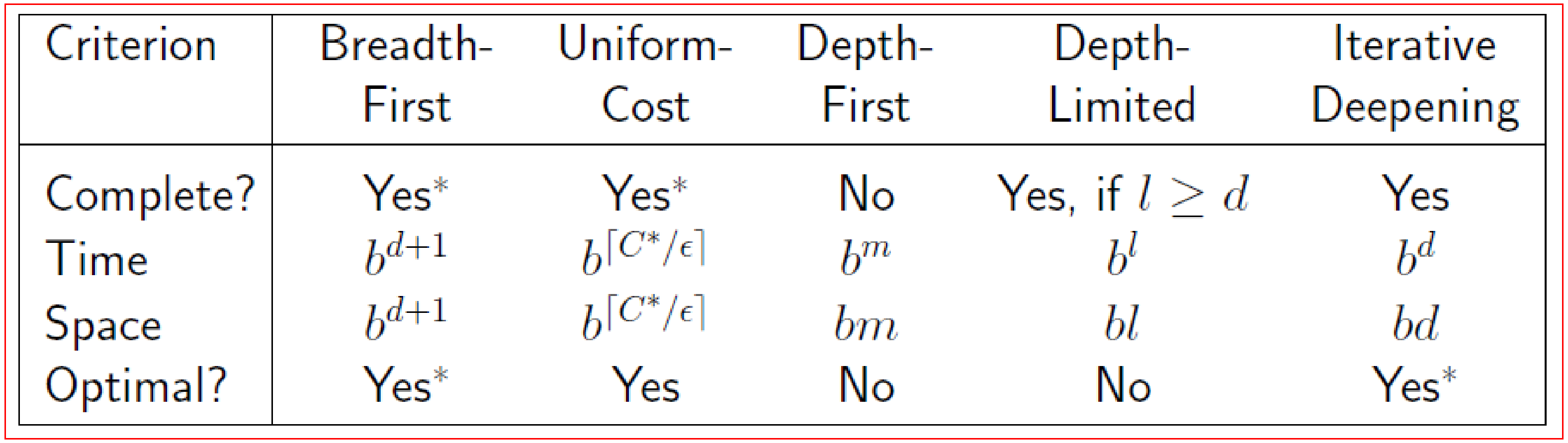


- The motivation is that $b^{d/2} + b^{d/2}$ is much less than b^d
- Bidirectional search is implemented by having one or both of the searches check each
- Bidirectional search is implemented by having one or both of the searches check each node before it is expanded to see if it is in the fringe of the other search tree;

Graph Search

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    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, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

Summary of algorithms (R-3-71)



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Bibliography

- [RN] Russel S., Norvig P. – Artificial Intelligence – A Modern Approach, 2nd ed. Prentice Hall, 2003 (1112 pages)
- [R] Stuart Russel – Course slides (visited oct. 2012 at http://aima.cs.berkeley.edu/instructors.html#homework)