Uninformed Search Strategies

Uninformed search: No clue about how close a state is to the goal.

Outline

I. Breadth-first search

II. Depth-first search

III. Iterative deepening and Bi-directional searches

I. Breadth-First Search

Expand the root first, then all its successors, next their successors, and so on.

- Systematic search.
- Complete even when the state space is infinite.
- Always finds a solution with a minimum number of actions.

BFS Algorithm

- Can call BEST-FIRST-SEARCH by letting the evaluation function $f(n)$ = node depth.

♠ Not efficient: Use a FIFO queue and adopt early goal test (when a node is generated).

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution node or *failure*
  node ← NODE(*problem*.INITIAL)
  if *problem*.IS-GOAL(*node*.STATE) **then return** *node*
  frontier ← a FIFO queue, with *node* as an element
  reached ← {*problem*.INITIAL}
  while not IS-EMPTY(*frontier*) **do**
    node ← POP(*frontier*)
    for each *child* **in** EXPAND(*problem*, *node*) **do**
      s ← *child*.STATE
      if *problem*.IS-GOAL(*s*) **then return** *child*
      if *s* is not in *reached* **then**
        add *s* to *reached*
        add *child* to *frontier*
  return *failure*

Time and Space Complexities

BFS on a uniform tree where every node has b successors.

branching factor

- b nodes at depth 1 generated by the root.
- Each node at depth 1 generates b nodes. $\Rightarrow b^2$ nodes at depth 2.
- And so on.

Solution at depth d $\Longrightarrow$ #nodes $= 1 + b + \cdots + b^d \;= O(b^d)$

$$= \frac{b^{d+1}-1}{b-1} \text{ (assuming } b > 1\text{)}$$

Time & space complexities
(since every node remains in memory)

♣ Only small search problems are solvable due to exponential time complexity.

♣ Memory is a bigger issue than time.

Time and Memory Requirements for BFS

Tree search: $O(b^d)$

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 KB
4	11,110	11 milliseconds	10.6 MB
6	10^6	1.1 seconds	1 GB
8	10^8	2 minutes	103 GB
10	10^{10}	3 hours	10 TB
12	10^{12}	13 days	1 PB
14	10^{14}	3.5 years	99 PB
16	10^{16}	350 years	10 EB

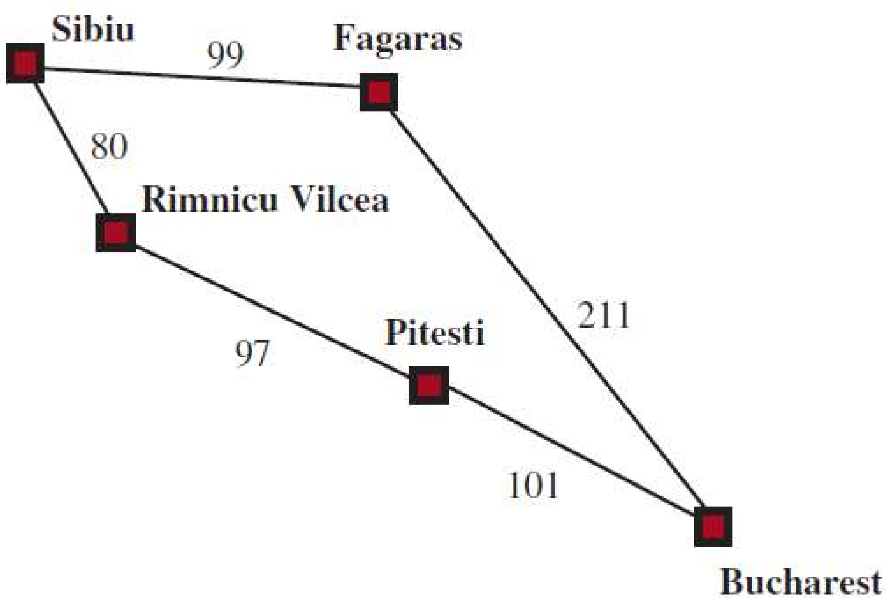
$$b = 10$$

Graph search is preferred since its time and space are proportional to the size of the state space (often less than $O(b^d)$).

Uniform-Cost Search (Dijkstra's Algorithm)

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution node, or *failure*
  return BEST-FIRST-SEARCH(*problem*, PATH-COST)

- Spreads out in waves of uniform path-cost.



Rimnicu Vilcea (80)
Fagaras (99)

$\Downarrow$

Fagaras (99)
Pitesti $(80 + 97 = 177)$

$\Downarrow$

Pitesti (177)
Bucharest $(99 + 211 = 310)$

$\Downarrow$

Bucharest $(177 + 101 = 278 < 310)$

Uninformed Search: Completeness and Complexity

- Completeness: systematic exploration of all paths – no chance of being trapped in one.

- Optimality: following from that of Dijkstra's algorithm.

- Complexity

$$O(b^{1+\lfloor C^*/\epsilon \rfloor})$$

C^*: cost of the optimal solution

ϵ: lower bound on the costs of all actions

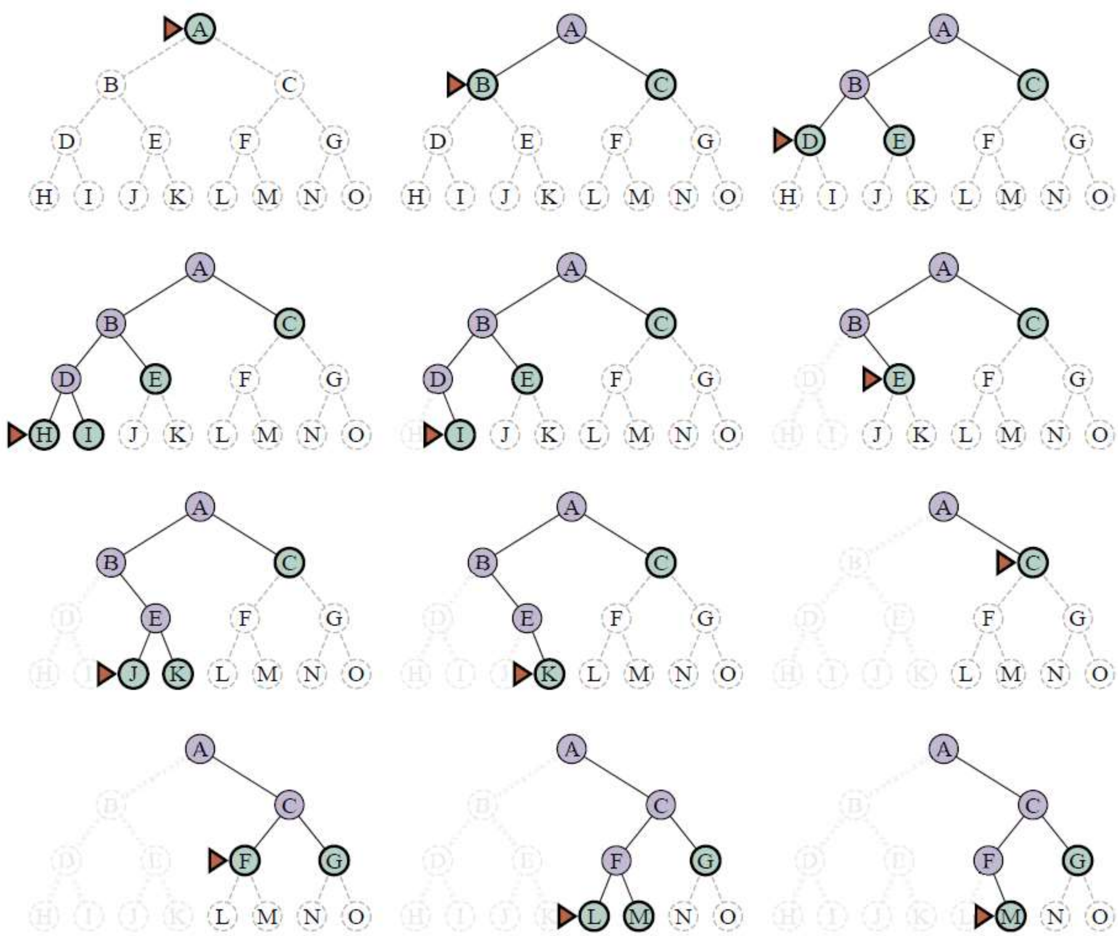
$\gg$ b^d possible.

$=$ b^{d+1} if all actions have the same cost.

II. Depth-First Search

Expand the *deepest* node in the frontier first.

- Implementable as a call to BEST-FIRST-SEARCH by setting $f(n)$ to the negative depth.



Downsides of DFS

- Not optimal

  Returns the first solution it finds, even if it is not the cheapest.

- Inefficient

  May expand the same state many times via different paths, and even systematically the entire space.

  May get stuck in an infinite loop in a cyclic state space.

- Incomplete

  Can go down an infinite path forever.

Why Consider DFS?

- Small memory for problems admitting tree-like search.

  No need for a table of reached nodes.

  Small frontier.

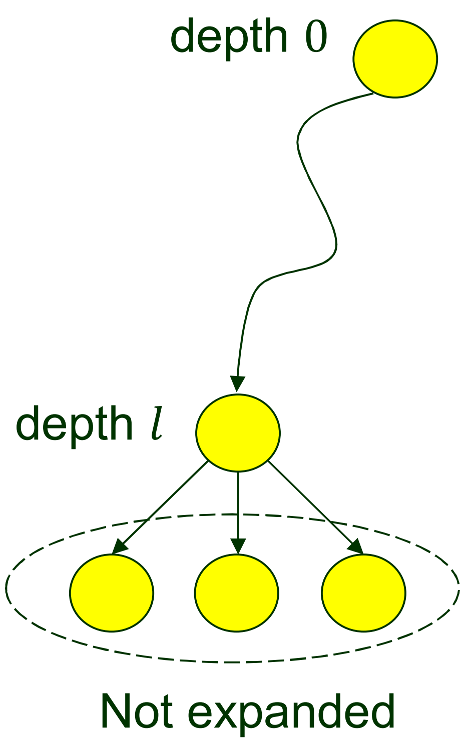
- Search time $O(\#\text{nodes})$.

- Memory consumption $O(bm)$.

  branching factor maximum depth

- Workhorse of constraint satisfaction, logic programming, etc.

Depth-Limited Seach



- To avoid exploring an infinite path, add a *depth limit* l.
- All nodes at depth l are considered dead ends even if they have successors.

Time: $O(b^l)$

Memory: $O(bl)$

♣ Performance sensitive to the choice for l.

What strategy to address this?

III. Iterative Deepening Search

Pick a good value for l by trying all values: 0, 1, 2, and so on.

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution node or *failure*
  for $depth = 0$ **to** ∞ **do**
    $result \leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if $result \neq cutoff$ **then return** *result*

Combines the benefits of DFS and BFS:

- Modest memory requirement like DFS:

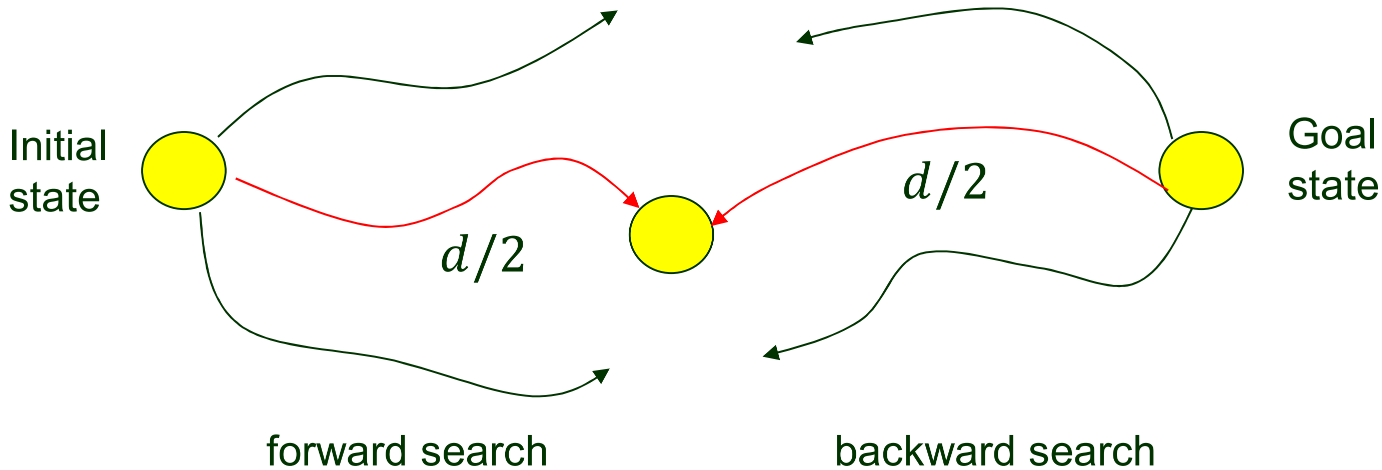
  $O(bd)$ when a solution exists (time $O(b^d)$).

  $O(bm)$ on a finite state space with no solution (time $O(b^m)$).

- Completeness and optimality of BFS:

limit: 0 

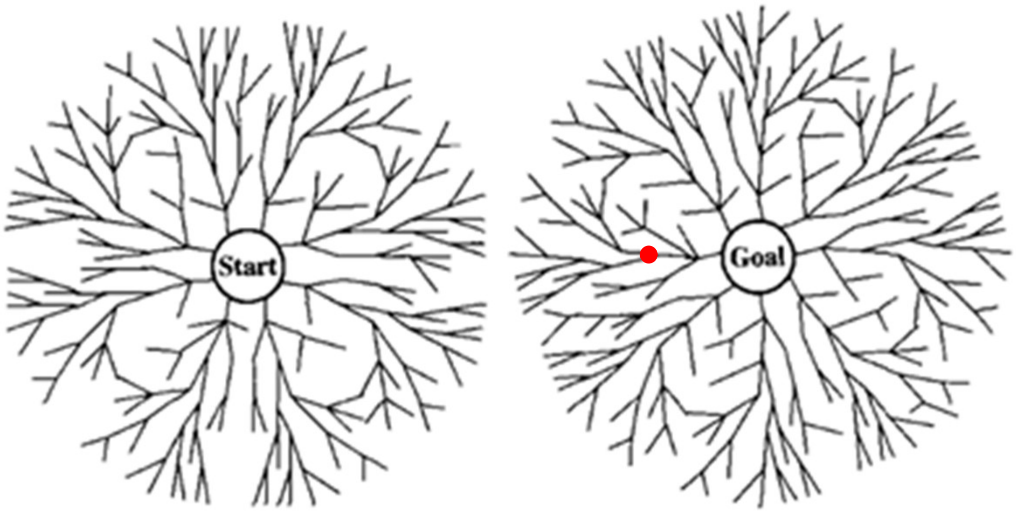
III. Bidirectional Search



Motivation:

$$b^{d/2} + b^{d/2} \ll b^d$$

Backward Reasoning



- Easy if all the actions are reversible.

  8-puzzle

  Finding a route in Romania

- Difficult to conduct if the goal is abstractly specified.

  8-queen

* Figure 3.20 in the 3rd edition

Comparing Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
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	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

b: branching factor

d: minimum depth of a solution

l: depth limit

m: maximum search tree depth

C^*: optimal solution cost

ϵ: minimum action cost