Comparing Uninformed Search Strategies

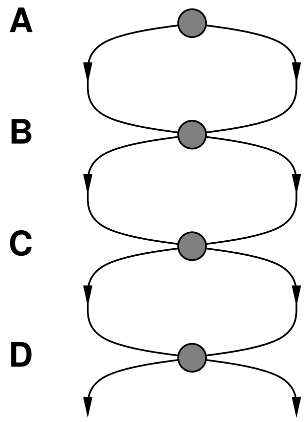
- Breadth-first search
  - Shallowest node in the search tree is expanded first.
  - Complete and optimal for unit-cost operators.
  - Time and space complexity O(b^d).
- Uniform-cost search:
  - Node with least cost so far is expanded.
  - Complete and optimal.
  - Time and space complexity O(b^d).
- Depth-first search
  - Deepest Node in the tree is expanded first.
  - Neither Complete nor optimal.
  - Time complexity O(b^m) and space complexity O(bm).
- Depth-limited search:
  - Depth-first search is performed with limited search depth.
- Iterative deepening search:
  - Repeated Depth-limited search with increasing depth limits.
  - Complete and optimal.
  - Time complexity O(b^d) and space complexity O(bd).
- Bidirectional search:
  - Possible only if all operators are reversible.

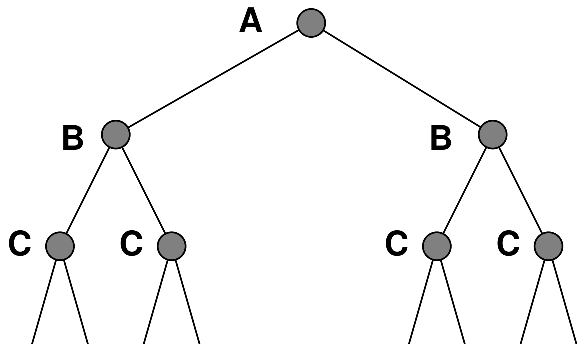
Comparing Uninformed Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Avoiding Repeated States

- A state space that generates an exponentially large search tree.

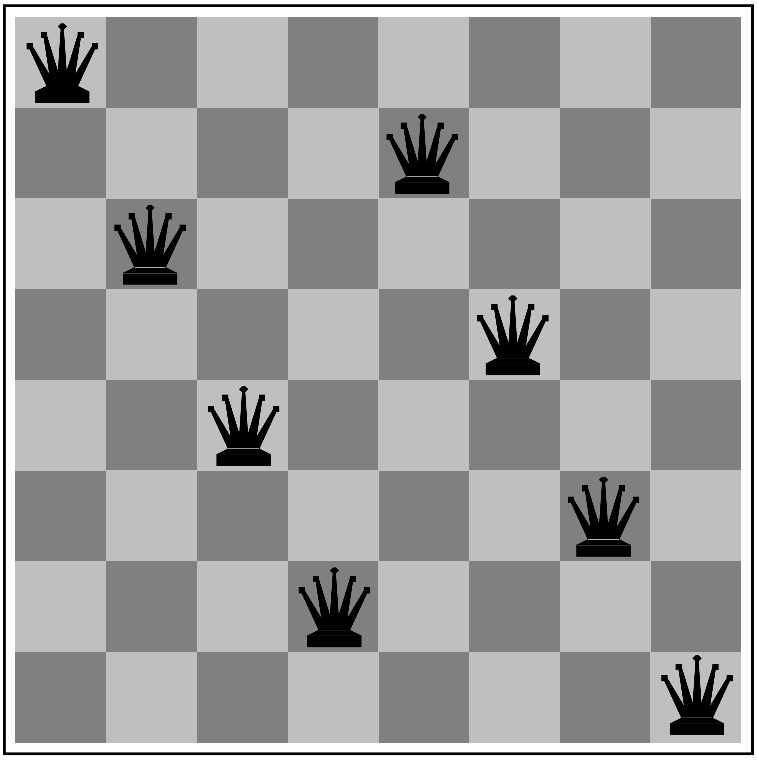




- Ways to deal with repeated states.
  - Do not return to the state you just came from.
  - Do not create paths with cycles in them.
  - Do not generate any state that was ever generated before.

Constraint Satisfaction Search

- In a constraint satisfaction problem (CSP),
  - the states are defined by the values of a set of variables
  - the goal test specifies a set of constraints that the values must obey.
- Example: 8-queens problem



Best-First Search

- Evaluation function:
  - A function defined on the nodes quantifying the desirability of expanding each node
- Best-first search always expands the node with the best evaluation, i.e., the node that *appears* to be the best (according to the evaluation function used).

An Implementation of Best-First Search

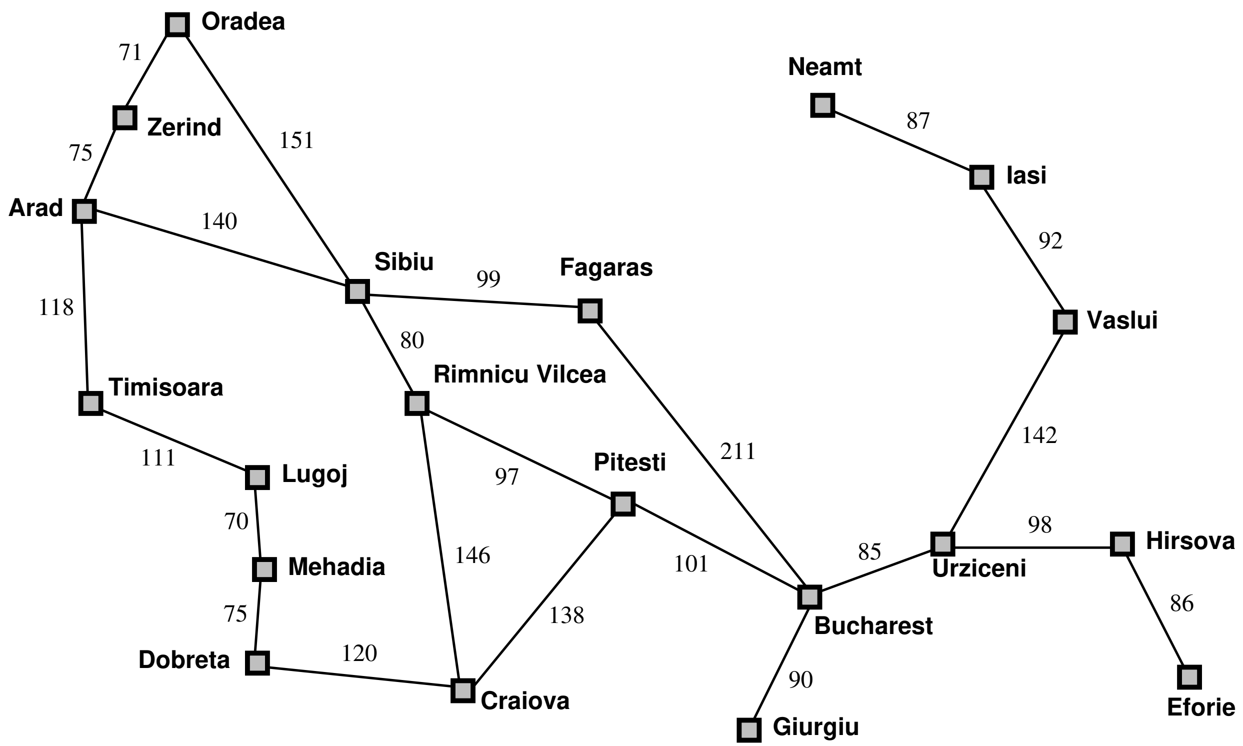
function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn $\leftarrow$ a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

Approach 1: Greedy Search

- Minimize the estimated cost to reach the goal.
- Always expand the node judged to be closest to the goal.
- Evaluation function:
  - Heuristic function, $h(n)$, which calculates the estimated cost of the cheapest path from node n to the goal.
- Neither complete nor optimal.

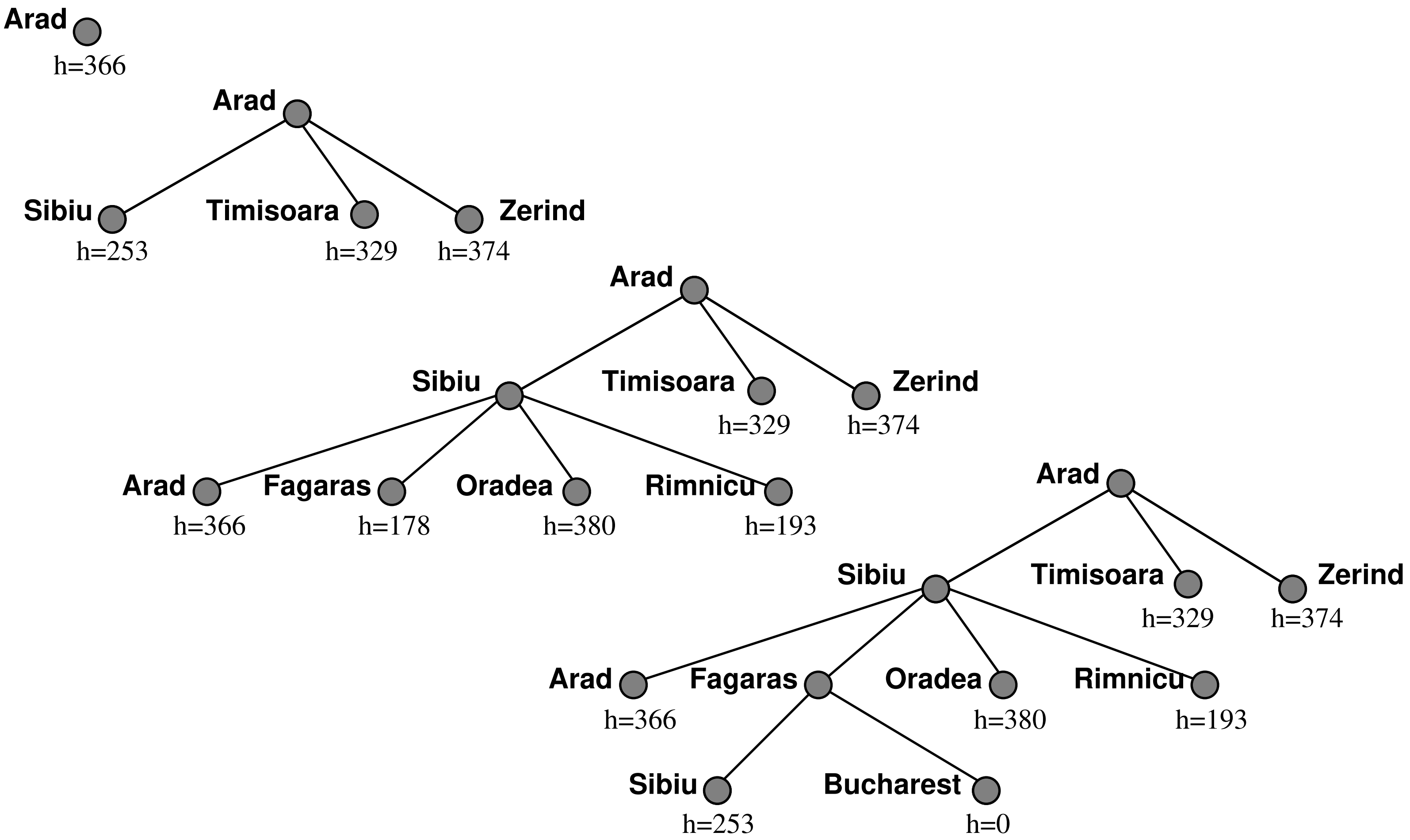
Greedy Search for Route Finding



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search for Route Finding (cont'd)



Approach 2: A* Search

- Combine greedy search and uniform-cost search.
- Evaluation function:
  - Sum of:
    - actual path cost $g(n)$ from the start node to node n
    - estimated cost $h(n)$
  - Estimated cost of the cheapest path through node n
- Idea: Try to find the cheapest solution.
  (Greedy Search: Try to find a solution as soon as possible.)
- Implementation:

  function A*-SEARCH(*problem*) **returns** a solution or failure

  return BEST-FIRST-SEARCH(*problem*, *g*+*h*)

- Degenerate cases:
  - Greedy search ($g = 0$)
  - Uniform-cost search ($h = 0$)

A* Search for Route Finding