U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Advanced Artificial Intelligence

Lecture 2b: Solving Search Problems

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence

Problem Solving using Search

Presentation Structure:

- Problem Solving Methods
- Problem Formulation
- State Space
- Blind/Uninformed Search:
  - Breadth First, Depth First, Uniform Cost, Iterative Deepening, Bidirectional Search
- Intelligent/Informed Search:
  - Greedy Search, A* Algorithm
- Practical Application Examples

Solution Search

Methodology to carry out the Solution search:

1. **Start with the initial state**
2. **Execute the goal test**
3. **If the solution was not found, use the operators to expand the current state generating new successor states (expansion)**
4. **Execute the objective test**
5. **If we have not found the solution, choose which state to expand next (search strategy) and carry out this expansion**
6. **Return to 4)**

Solution Search - Search Tree

- **Search tree composed by nodes. Leaf nodes either have no successors or have not yet been expanded!**
- **Important to distinguish between the search tree and the state space!**

- **Tree Node (five components):**
  - **Corresponding state**
  - **Node that gave rise to it (father)**
  - **Operator applied to generate it**
  - **Node depth**
  - **Path cost from the starting node**

- **datatype** NODE

  components: STATE, PARENT-NODE,
  OPERATOR, DEPTH, PATH-COST
- **Border:**
  - **Set of nodes waiting to be expanded**
  - **Represented as a queue**

Search Strategies

- A search strategy is defined by the order of node expansion

- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: How long does it takes (total number of nodes generated)?
  - space complexity: How much memory it needs (maximum number of nodes in memory)?
  - optimality: does it always find the best (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 ∞)

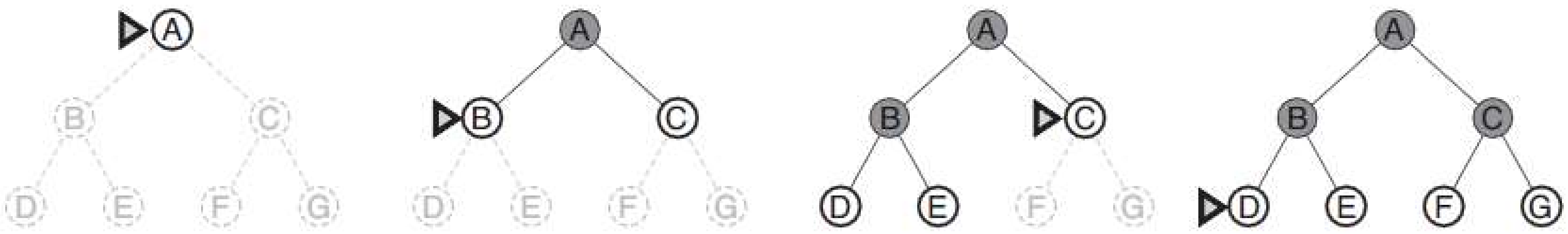
- Types of search Strategies:
  - Uninformed Research (blind)
  - Informed Search (heuristic/intelligent)

Breadth-first Search

Breadth-first search

- **Strategy: Nodes at lowest depth are expanded first**
- **Good: Very systematic search**
- **Bad: Usually it takes a long time and above all it takes up a lot of space**
- **Assuming branching factor b then $n=1+b+b^2+b^3+ \ldots +b^n$**
- **Exponential complexity in space and time: $O(b^d)$**
- **In general, only small problems can be solved like this!function**
  BREADTH-FIRST-SEARCH(problem) returns a solution or failure

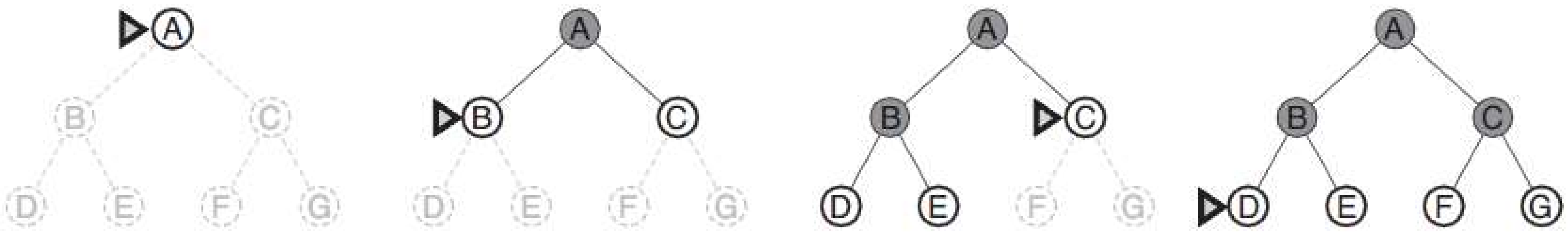
  GENERAL-SEARCH(problem,ENQUEUE-AT-END)



Uniform Cost Search

Uniform Cost Search

- **Strategy: Always expand the border node with the lowest cost (measured by the solution cost function)**
- **Breadth first Search is equal to Uniform Cost Search if g (n) = Depth (n)**



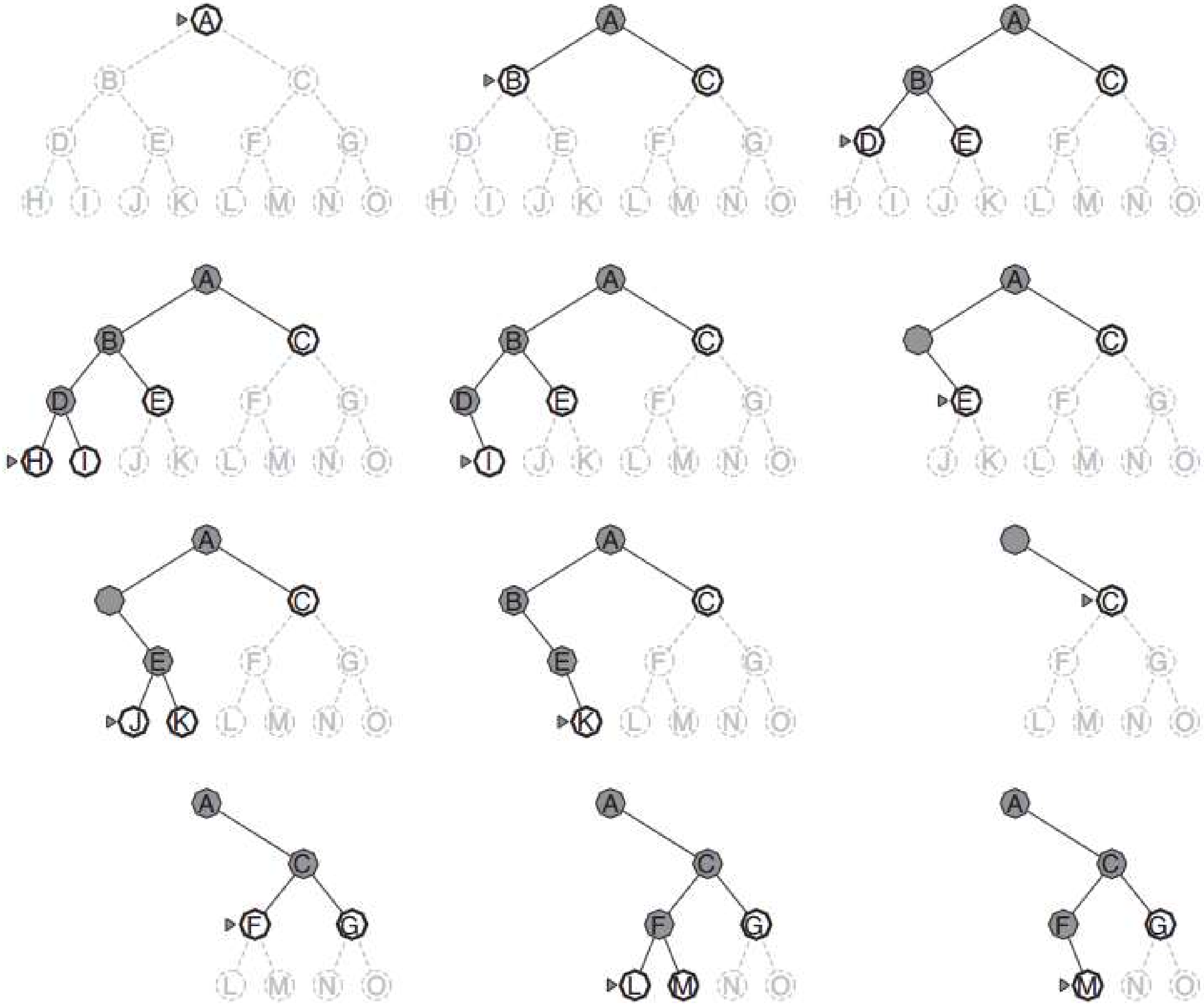
Depth-First Search

Depth-First Search

- **Strategy: Always expand one of the deepest nodes in the tree**
- **Good: Very little memory required, good for problems with lots of solutions**
- **Bad: Cannot be used for trees with infinite depth, can get stuck in wrong branches**
- **Complexity in time O (b^m) and space O (bm).**
- **Sometimes a limit depth is defined and it becomes a Search with Limited Depth**
- **function DEPTH-FIRST-SEARCH(problem) returns a solution or failure**

  GENERAL-SEARCH(problem,ENQUEUE-AT-FRONT)

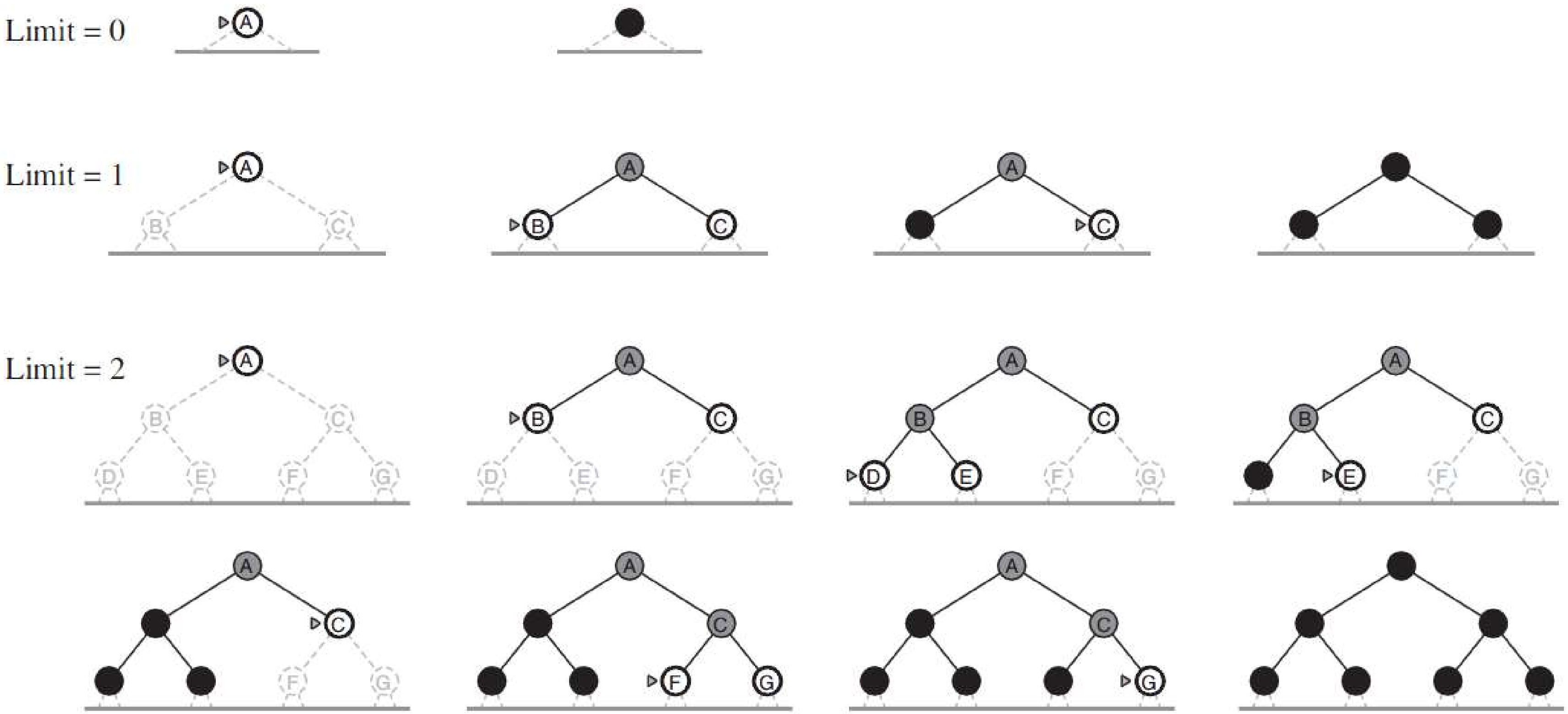
Depth-First Search (2)



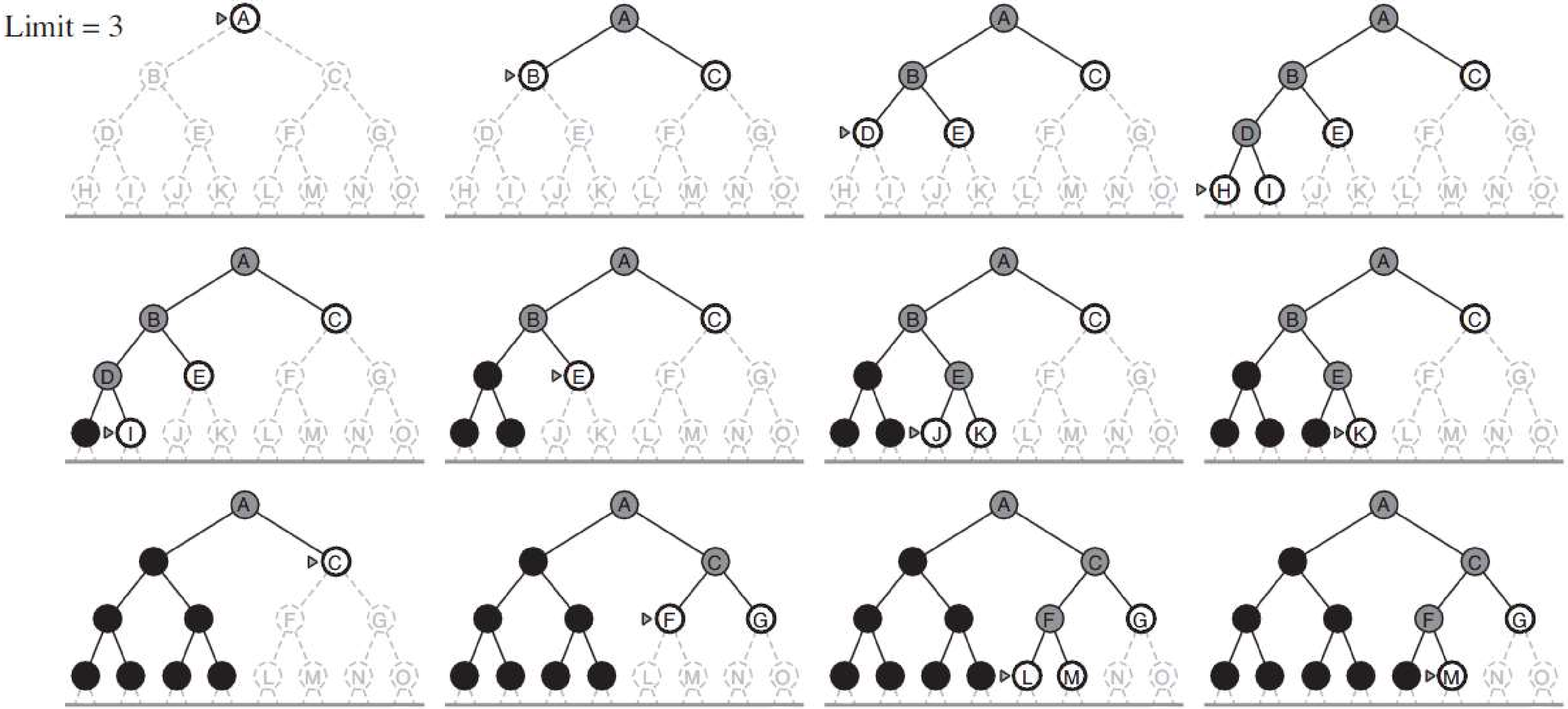
Iterative Deepening Search

- Strategy: Perform limited depth search, iteratively, always increasing the depth limit
- Complexity in time $O(b^d)$ and in space $O(bd)$.
- In general it is the best strategy for problems with a large search space and where the depth of the solution is not known

Iterative Deepening Search (2)

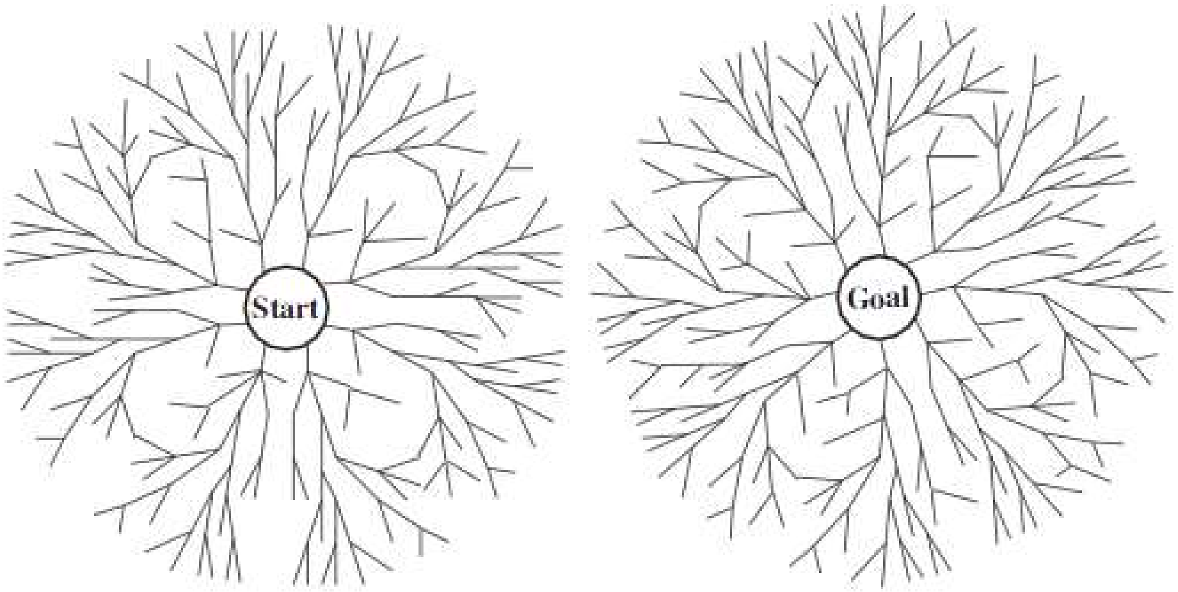


Iterative Deepening Search (3)



Bidirecional Search

- Strategy: Run forward search from the initial state and backward search from the target, simultaneously
- Good: Can greatly reduce complexity over time
- Problems:
  - Is it possible to generate predecessors?
  - What if there are many objective states?
  - How to do the "matching" between the two searches?
  - What kind of research to do in the two halves?



Comparison between Search Strategies

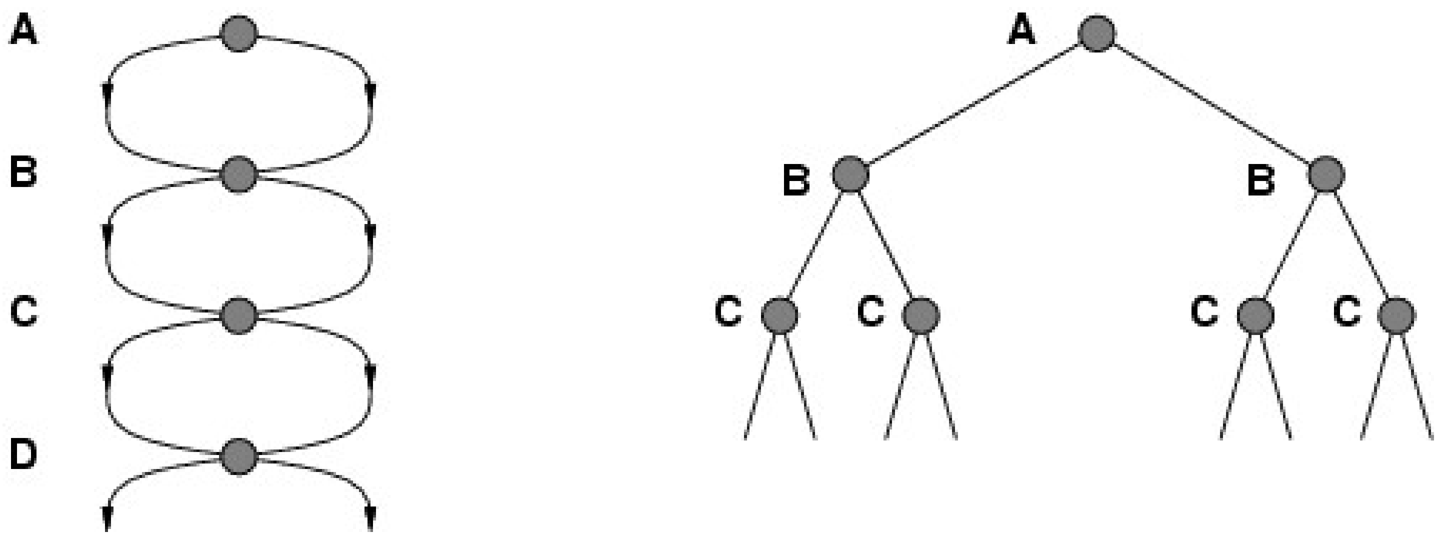
Evaluation of search strategies:

- **b is the branching factor**
- **d is the depth of the solution**
- **m is the maximum depth of the tree**
- **l is the search limit depth**

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

Repeated States

- **Failure to detect repeated states can make a linear problem an exponential problem!**
- **Avoid Repeated States**
  - Do not return to the previous state
  - Do not create cycles (do not return to states where you passed in the sequence)
  - Do not use any repeated states (is it possible? What is the computational cost?)



Exercises – Search

Formulate the following problems (analysed in the previous class) as search problems and solve them using the various search strategies studied:

- Missionaries and Cannibals
- Bucket Filling problem
- Towers of Hanoi
- Cryptograms
- 8-Queens and N-Queens
- 8-Puzzle and N-Puzzle

Note: For all problems try to make the solution as generic as possible in order to allow solving versions with different data

Informed/Intelligent/Heuristic Search

Informed/Intelligent/Heuristic Search:

- **Use problem information to prevent the search algorithm from being "lost wandering in the dark"!**

Search Strategy:

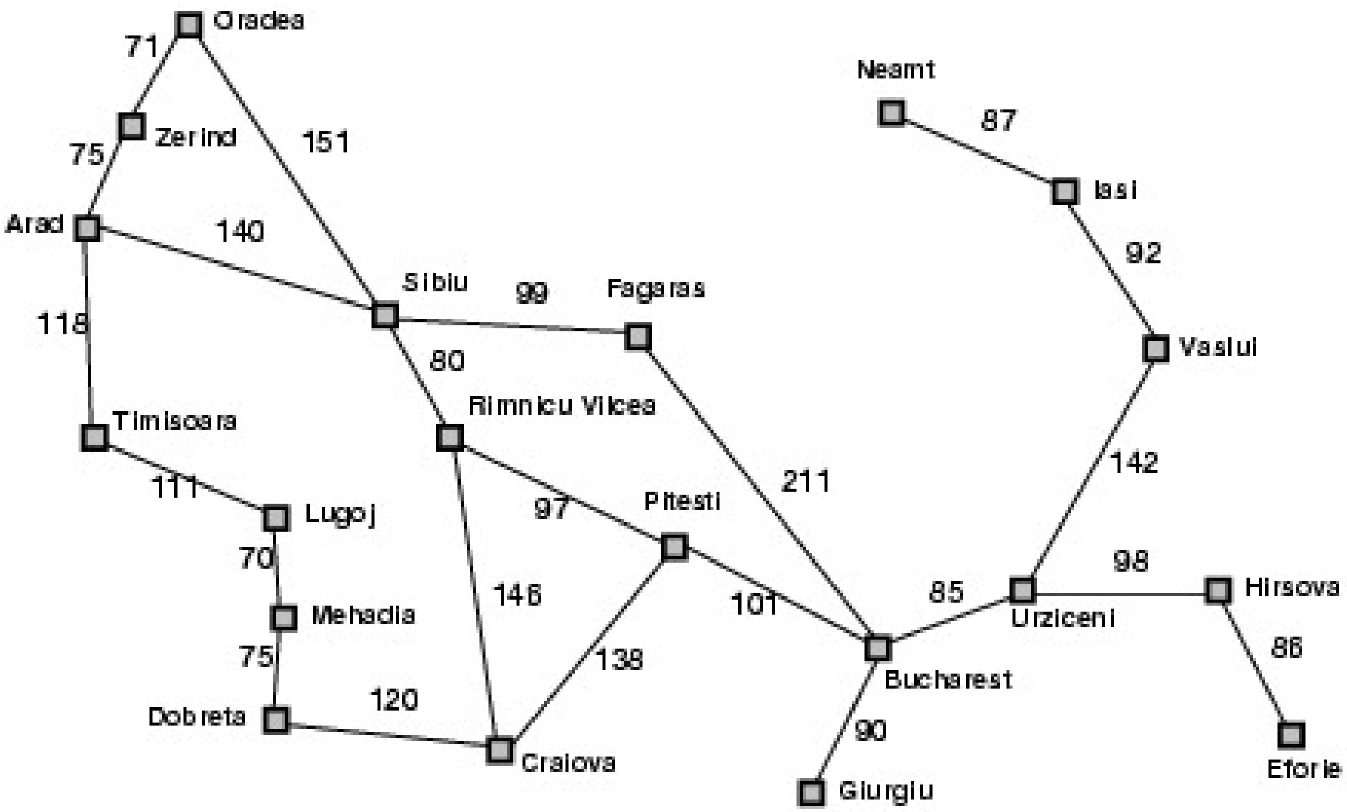
- **Defined by choosing the order of expansion of the nodes!**

Best-First Search

- **Uses an evaluation function that returns a number indicating the interest to expand a node**
- **Greedy-Search - $f(n) = h(n)$ - estimates distance to solution**
- **A* Algorithm - $f(n) = g(n) + h(n)$ - estimates cost of the best solution that passes through n**

Informed Search Example

- **Initial State: Arad; Objective: Bucharest; h(n)= straight line distance**



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy-Search

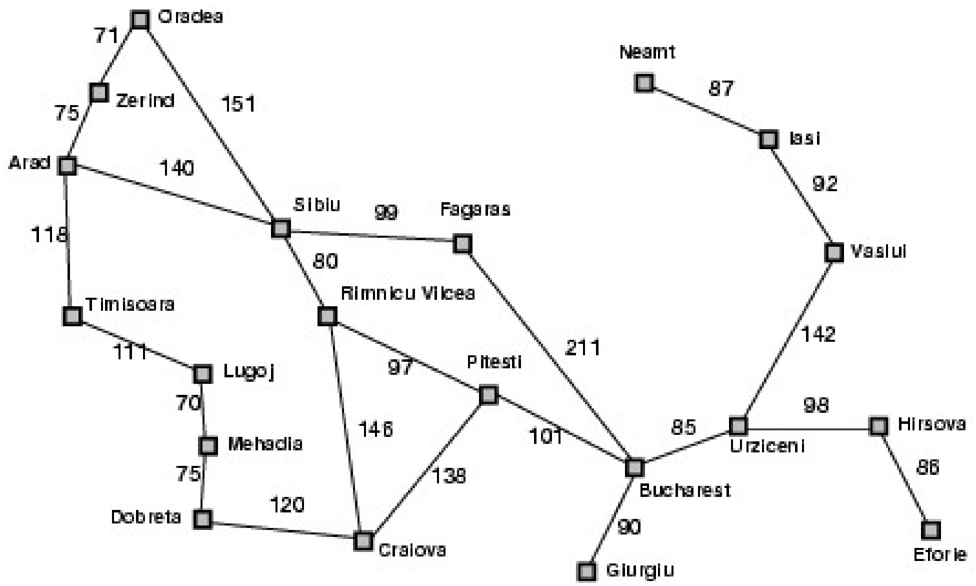
Strategy:

- **Expand the node that appears to be closest to the solution**
- **h (n) = estimated cost of the shortest path from state n to the objective (heuristic function)**
- **function GREEDY-SEARCH(problem) returns a solution or failure**

  return BEST-FIRST-SEARCH(problem,h)
- **Example:**
  - $h_{SLD}(n)$ = straight line distance between n and the objective

Greedy-Search

- **Initial State: Arad; Objective: Bucharest; h(n)= straight line distance**

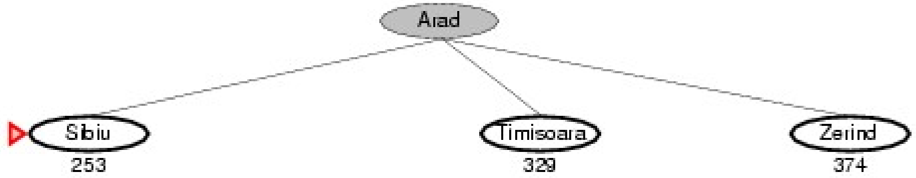


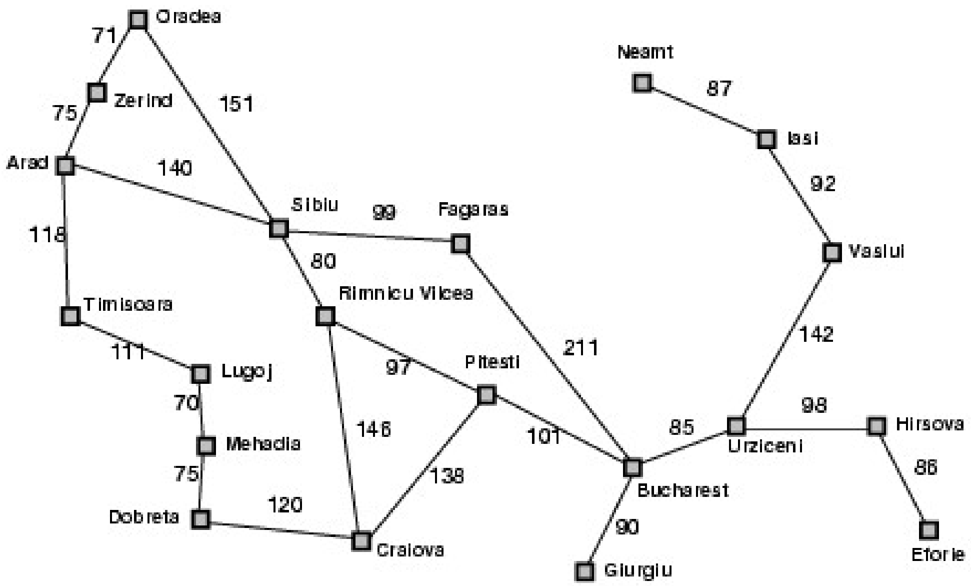
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy-Search

- **Initial State: Arad; Objective: Bucharest; h(n)= straight line distance**



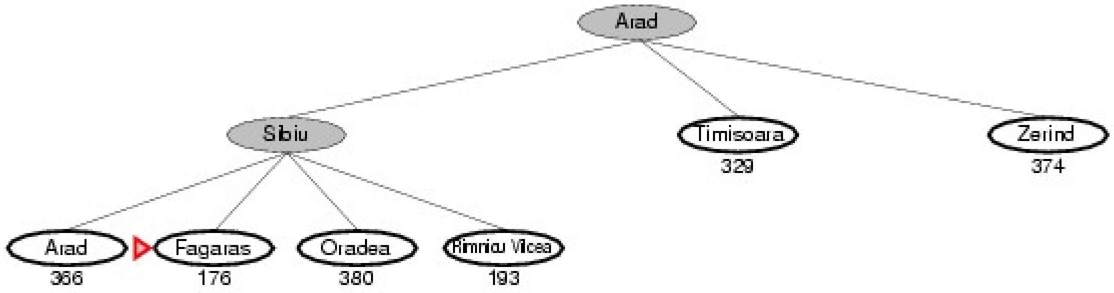


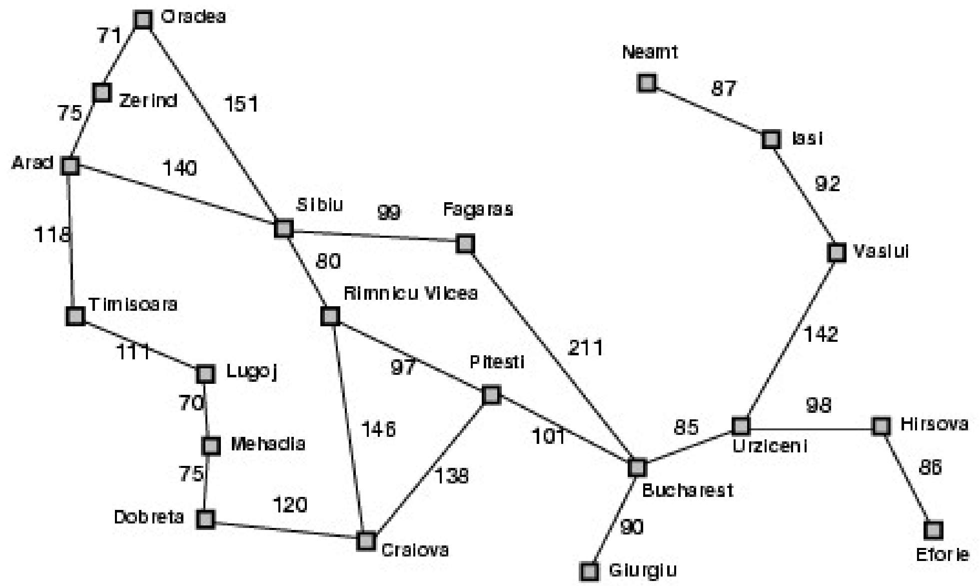
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy-Search

- **Initial State: Arad; Objective: Bucharest; h(n)= straight line distance**



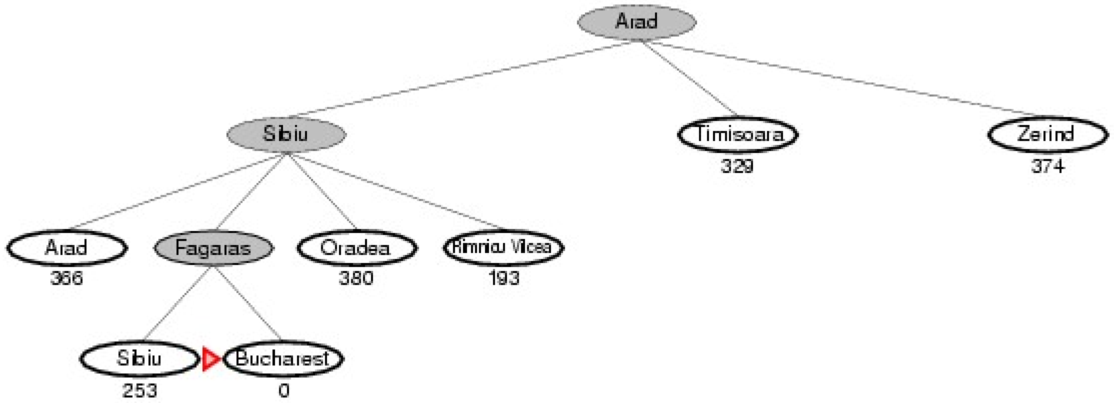


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy-Search

- **Estado Inicial: Arad; Objetivo: Bucharest; h(n)= distância em linha reta**



Greedy-Search

Greedy Search Properties:

- **Complete? No! Cycles! (ex:, Iasi → Neamt → Iasi → Neamt → Iasi …)**
- **Susceptible to false starts**
- **Time complexity? O(b^m)**
  - but with a good heuristic function it can decrease considerably
- **Complexity in space? O(b^m)**
  - Keeps all nodes in memory
- **Optimal? No! You don't always find the optimal solution!**
- **It is necessary to detect repeated states!**

A* Algorithm

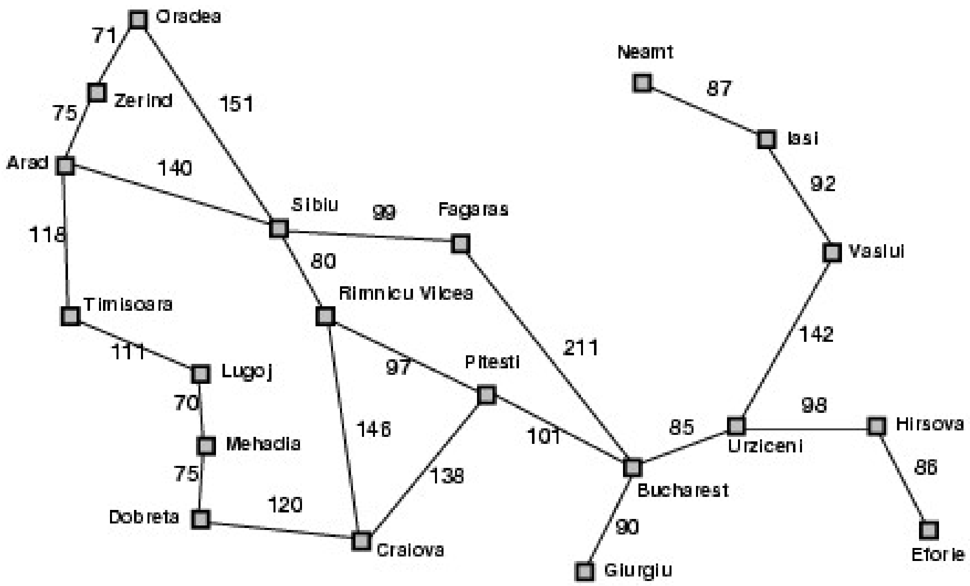
A* Algorithm Strategy:

- **The A * algorithm combines the greedy with the uniform search minimizing the sum of the path already carried out with the minimum expected until the solution.**
- **It Uses the function: f (n) = g (n) + h (n)**
  - g (n) = total cost, so far, to reach state n
  - h (n) = estimated cost to reach the objective (you cannot overestimate the cost to reach the solution! You have to be optimistic!)
- **f (n) = estimated cost of the cheapest solution that passes through node n**
- **function A*-SEARCH(problem) returns a solution or failure**

  return BEST-FIRST-SEARCH(problem,g+h)
- **Algorithm A * is optimal and complete!**
- **Exponential time complexity (but depends on the quality of the heuristic)**
- **Complexity in space: Keeps all nodes in memory!**

A* Algorithm

- **Initial State: Arad; Objective: Bucharest; f(n) = g(n) + h(n);**
- **g(n) = cost from start to node n: h(n)= straight line distance to objective**

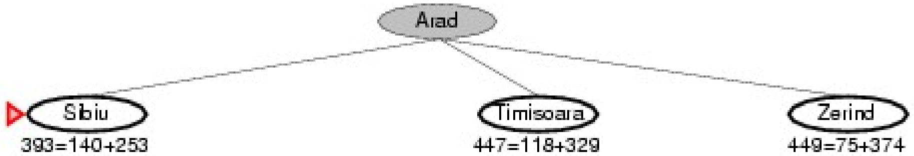


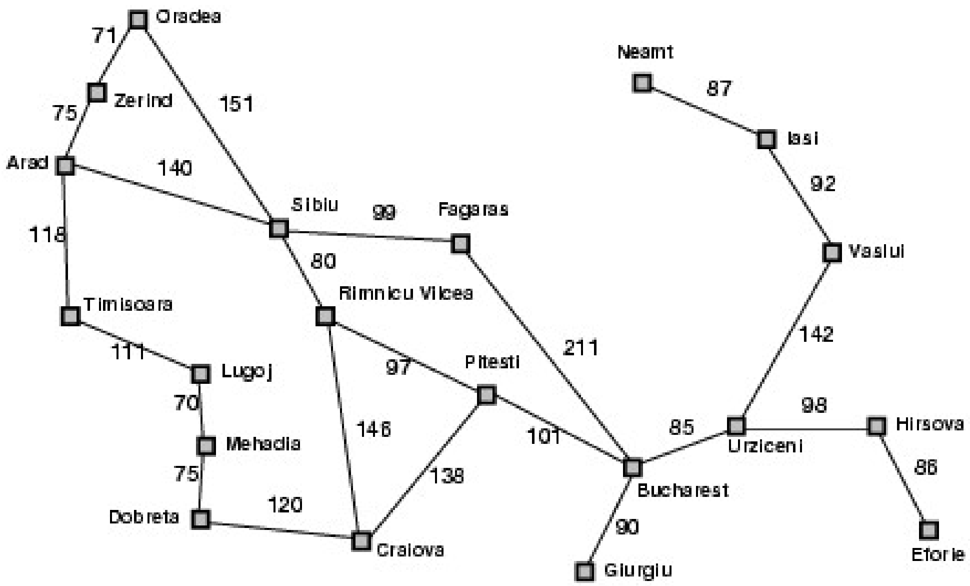
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Algorithm

- **Initial State: Arad; Objective: Bucharest; f(n) = g(n) + h(n);**
- **g(n) = cost from start to node n: h(n)= straight line distance to objective**



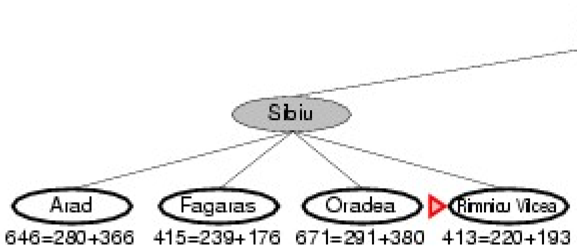


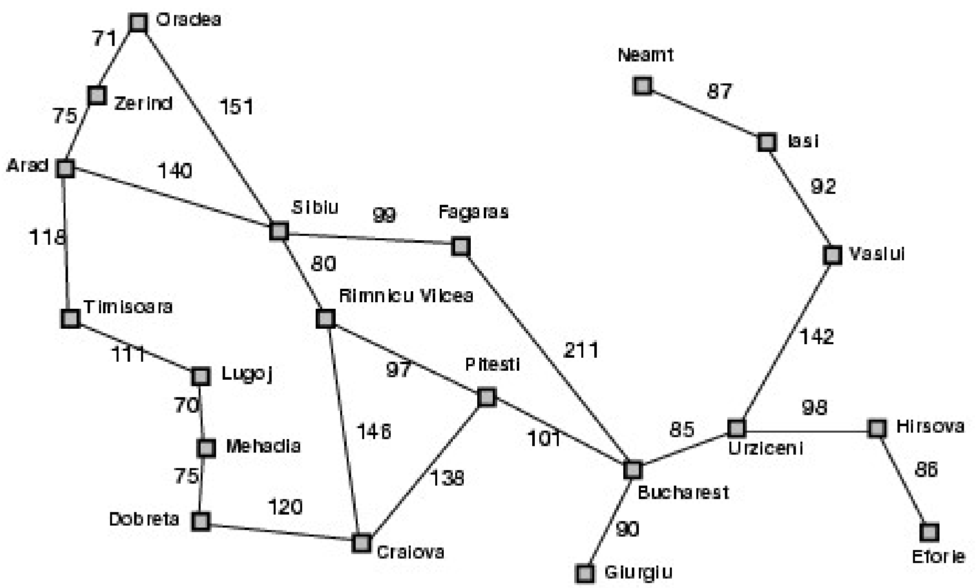
Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Algorithm

- **Initial State: Arad; Objective: Bucharest; f(n) = g(n) + h(n);**
- **g(n) = cost from start to node n: h(n)= straight line distance to objective**

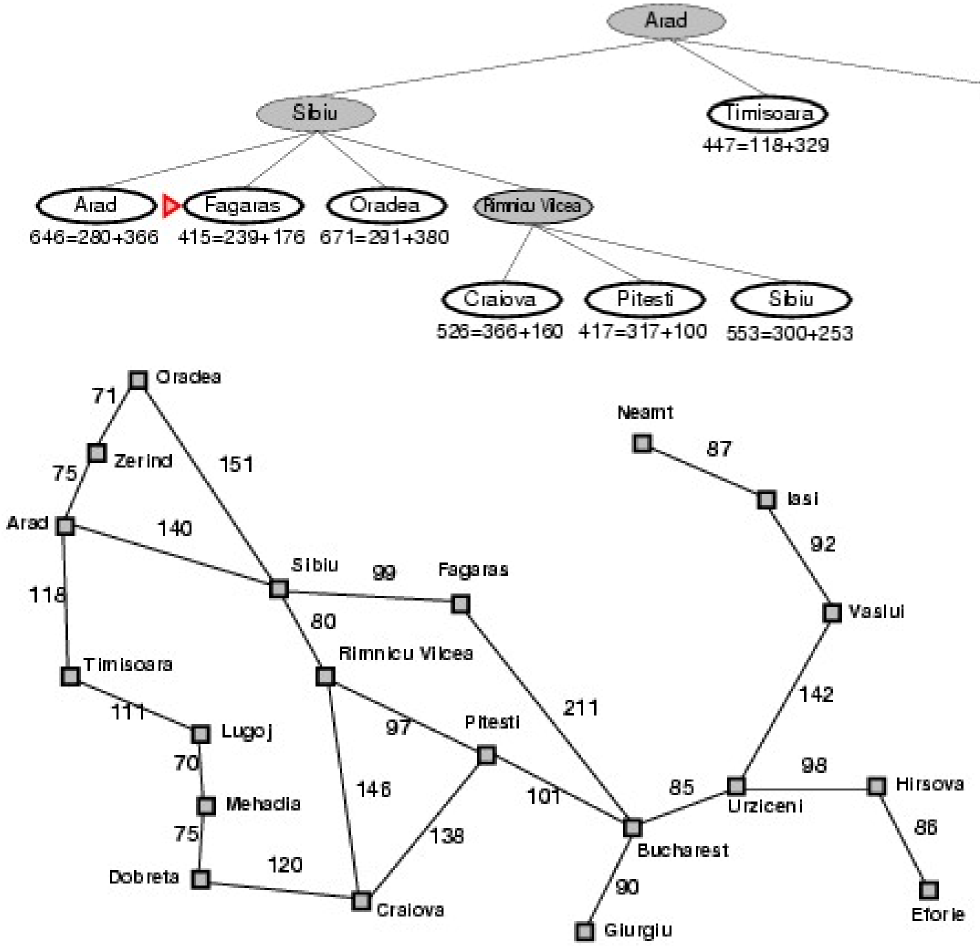




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Algorithm

- **Initial State: Arad; Objective: Bucharest; f(n) = g(n) + h(n);**
- **g(n) = cost from start to node n: h(n)= straight line distance to objective**

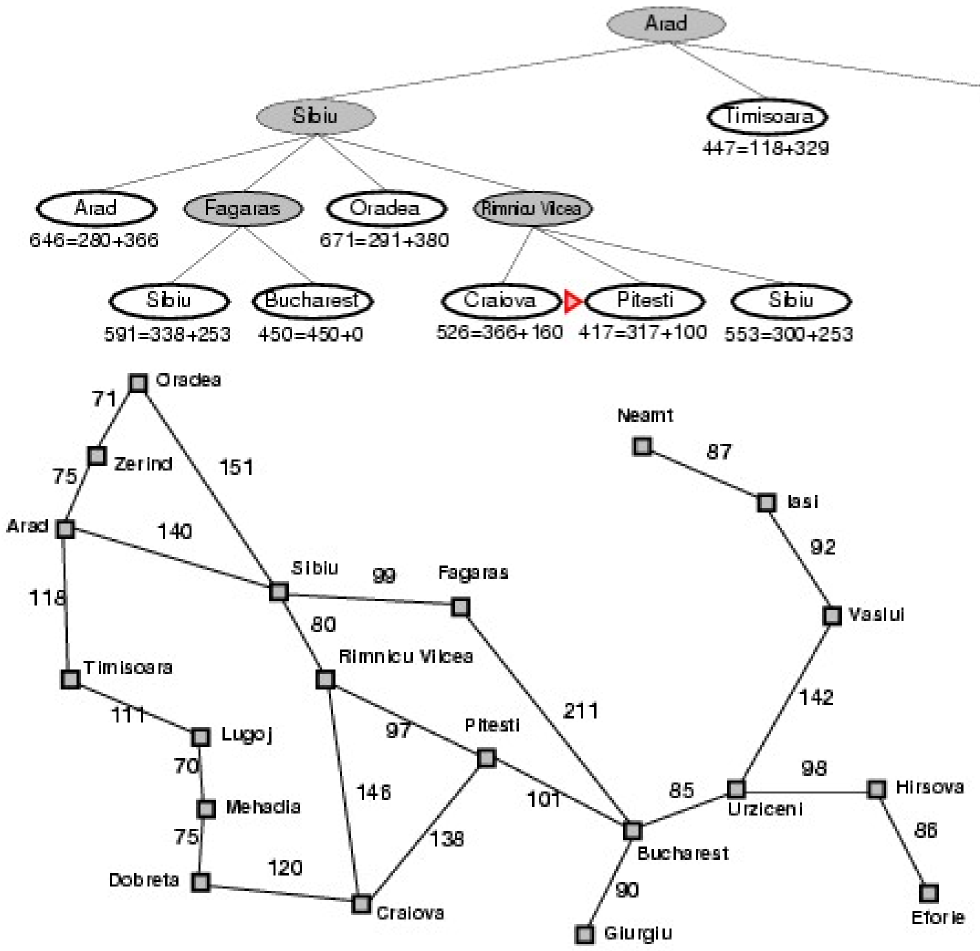


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Algorithm

- **Initial State: Arad; Objective: Bucharest; f(n) = g(n) + h(n);**
- **g(n) = cost from start to node n: h(n)= straight line distance to objective**

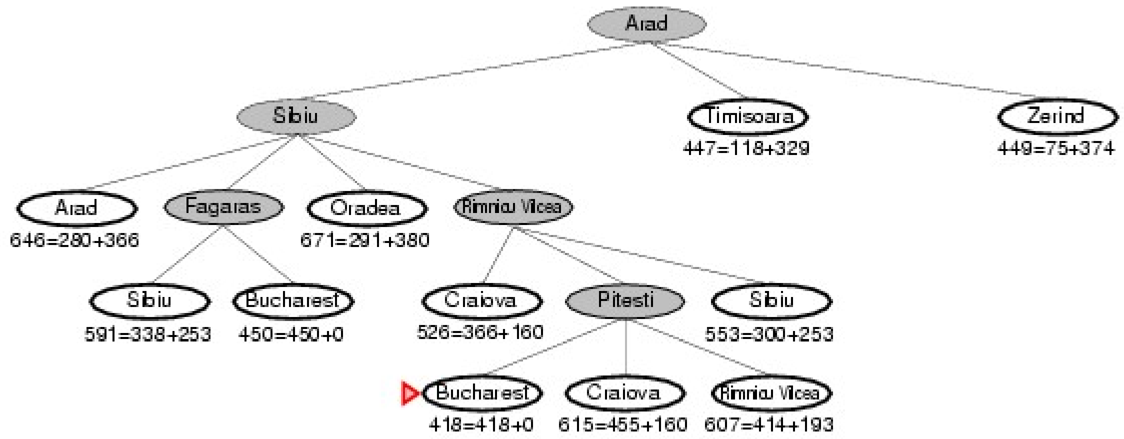


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

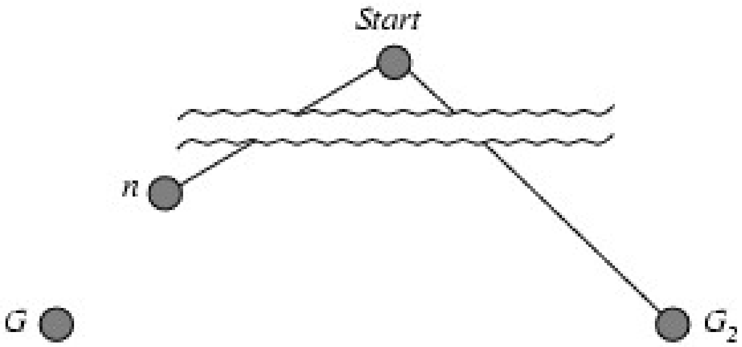
A* Algorithm

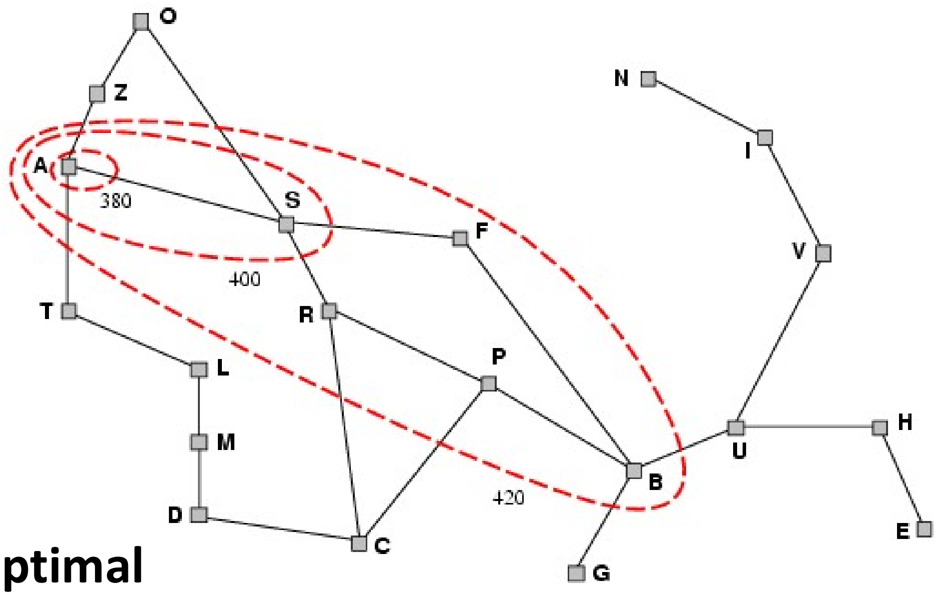
- **Initial State: Arad; Objective: Bucharest; f(n) = g(n) + h(n);**
- **g(n) = cost from start to node n: h(n)= straight line distance to objective**



A* Algorithm Optimality

- **Assuming that a sub-optimal G2 objective has been generated and is on the list. n being an unexpanded node that leads to the optimal goal G1**

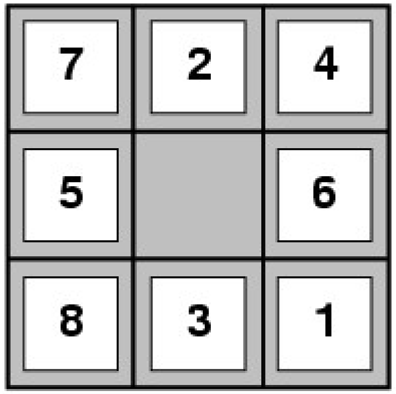




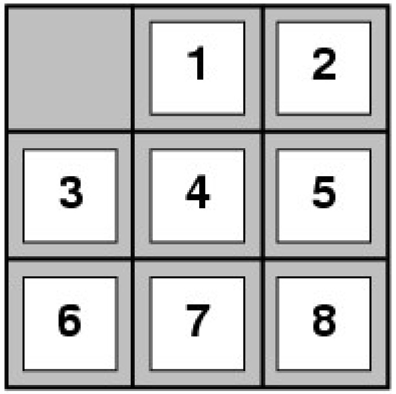
- **f(G2) = g(G2) bacause h(G2)=0**
- **f(G2) > g(G1) because G2 is sub-optimal**
- **f(G2) >= f(n) because h is an admissible heuristic**
- **Thus, the A* Algorithmo never chooses G2 for expansion!**

Heuristic Functions - 8 Puzzle

- Typical 20-step solution with average branching factor: 3
- Number of states: 3^{20} = 3.5 * 109
- Nº States (without repeated states) = 9! = 362880
- Heuristics:
  - h1 = Number of pieces outside the correct placement
  - h2 = Sum of pieces distances to their correct positions



Start State



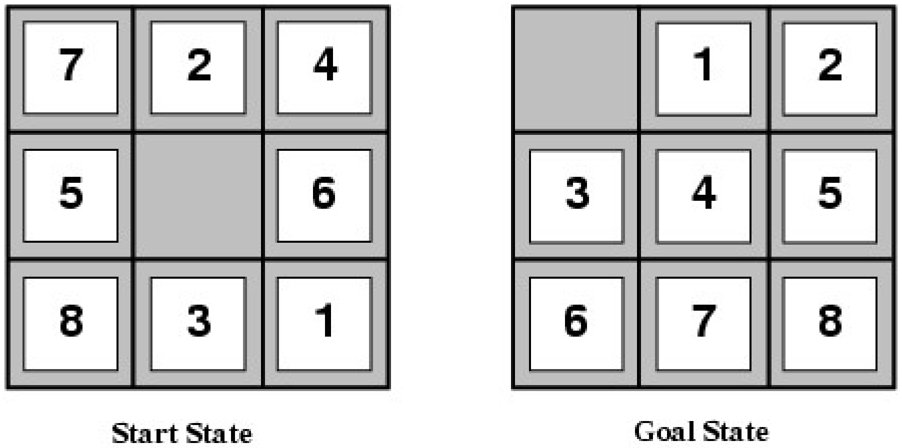
Goal State

Heuristic Functions - 8 Puzzle (2)

Problem Relaxation as a way to invent heuristics:

Piece can move from A to B if A is adjacent to B and B is empty

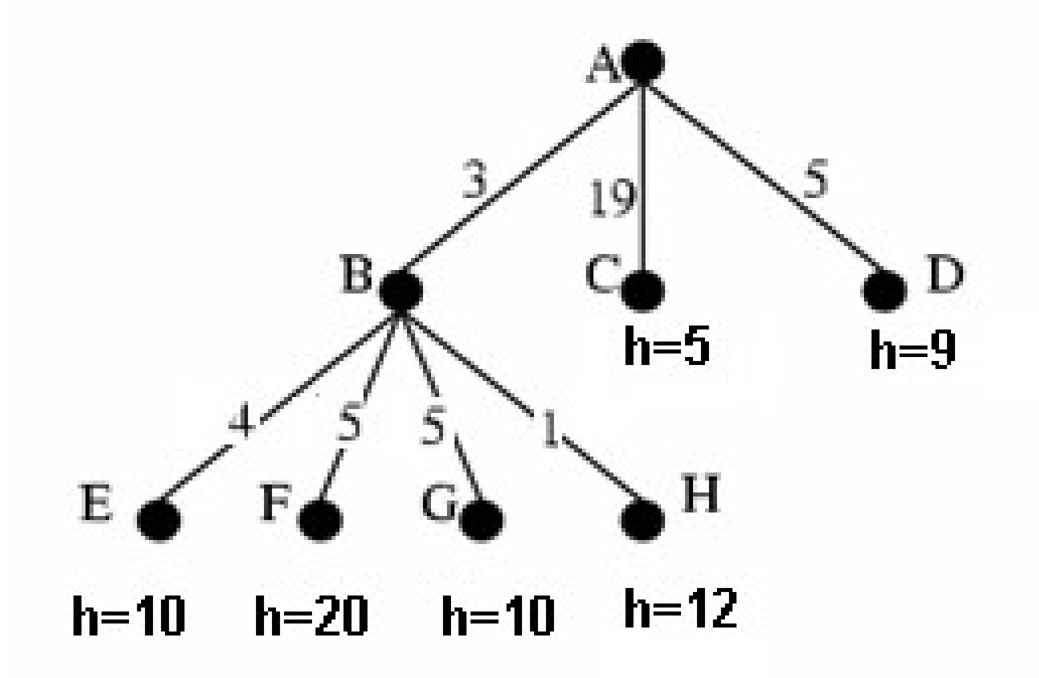
- **a) Piece can move from A to B if A is adjacent to B**
- **b) Piece can be moved from A to B if B is empty**
- **c) Piece can be moved from A to B**



Exercise

Assuming the following search tree in which each arc shows the cost of the corresponding operator, indicate justifying, which node is expanded next using each of the following methods:

a) Breadth-First Search; b) Depth-first Search; c) Uniform Cost search; d) Greedy Search; e) A* Algorithm



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Advanced Artificial Intelligence

Lecture 2b: Solving Search Problems

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence