Artificial Intelligence (CS303)

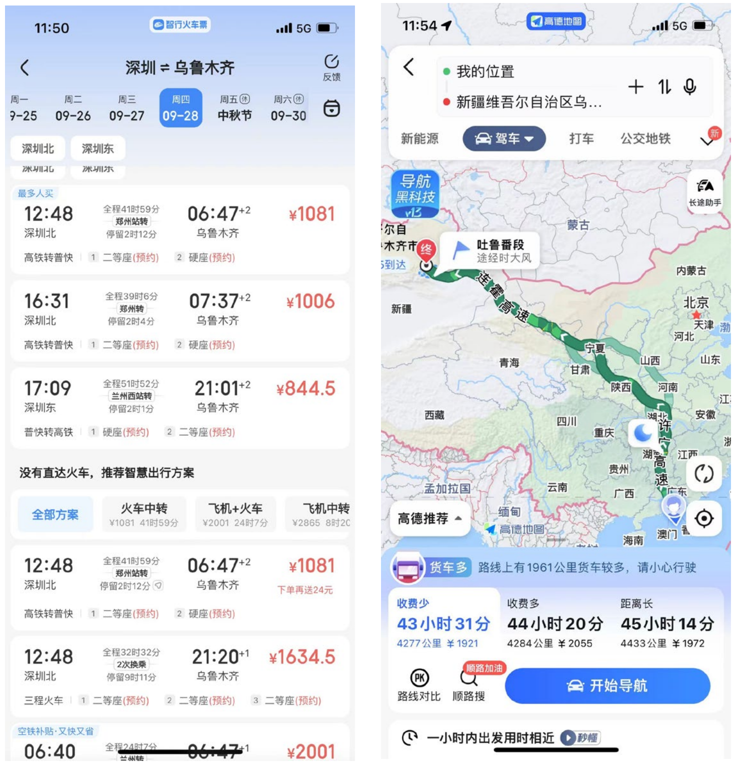
Lecture 1: AI as Search

Outline of This Lecture

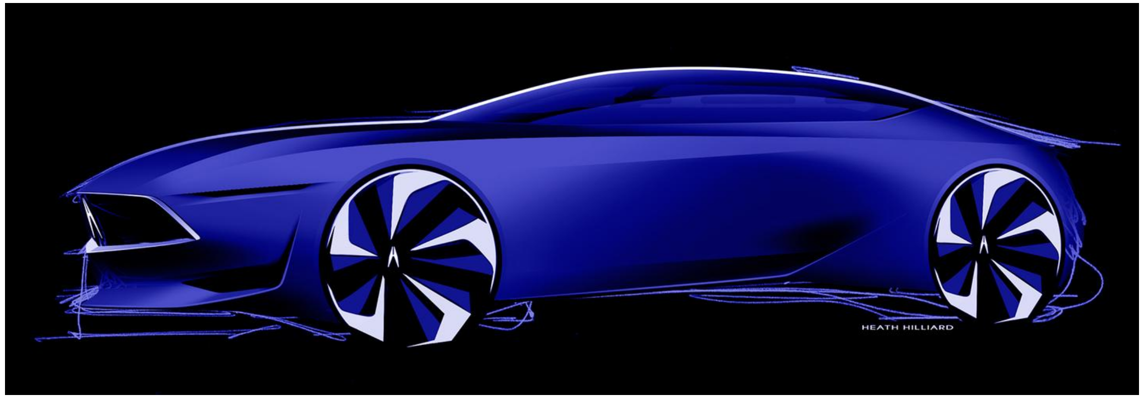
- What is search
- From searching to search tree
- Uninformed Search Methods
- Heuristic (informed) Search
- Further Studies on Heuristics

What is Search?

- Search is ubiquitous (our thought process can usually be viewed as a search process).



Find the most efficient way to Urumqi.



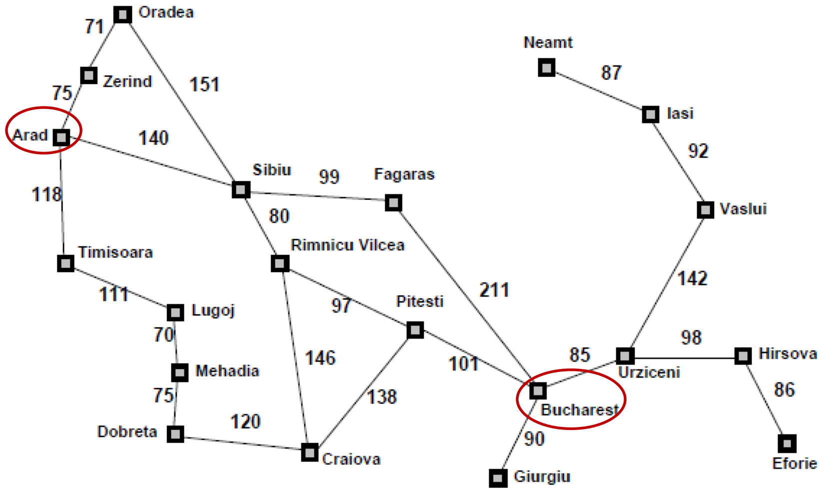
Design a fancy car

What is Search?

- Search is ubiquitous (our thought process can usually be viewed as a search process).

Find the most efficient way to Urumqi.

--> find the shortest path between two points on a graph.

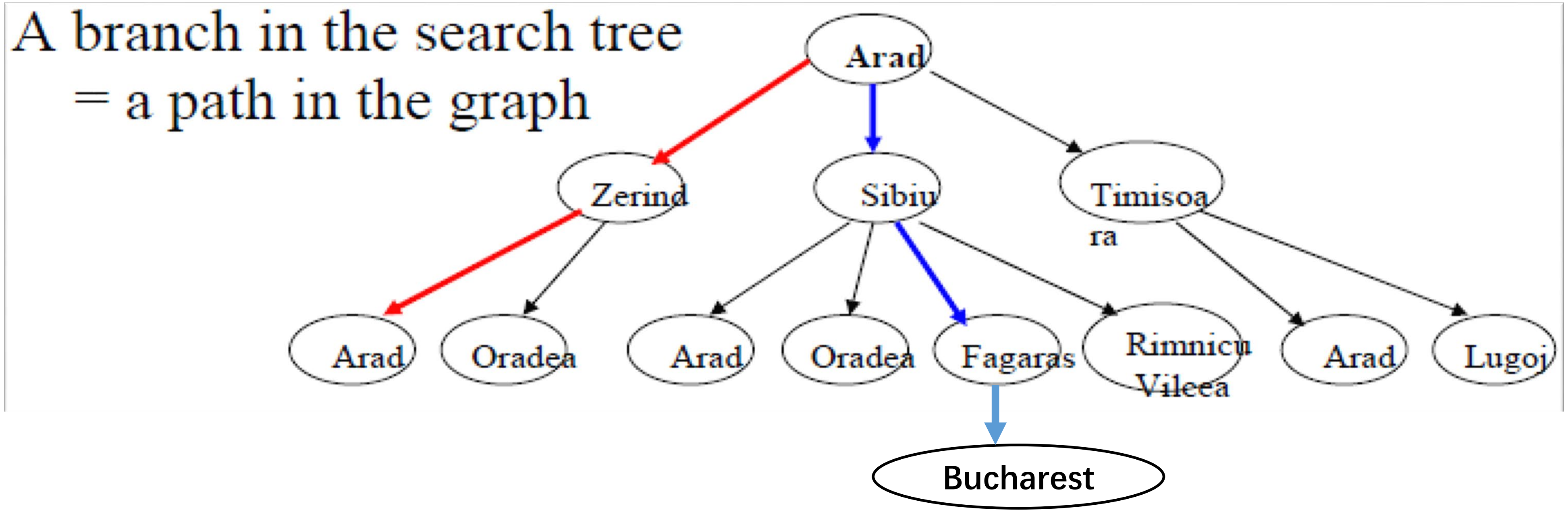


Technical Question

- How could an agent find the shortest path between two points on a graph？
  - Enumerate and compare
    - O(*n*!) options (*n* is the number of nodes), too inefficient
  - Any smarter approach?
    - don‘t enumerate
    - to *test* which options?→need a better organized search process

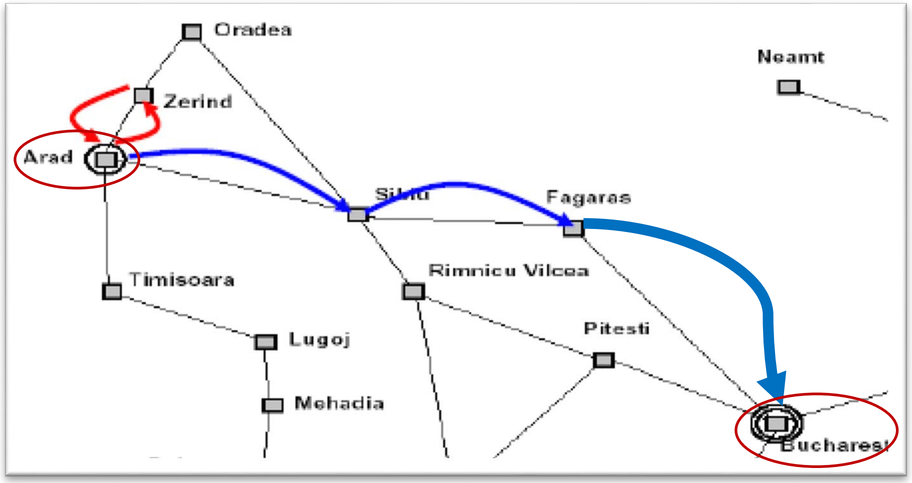
Search in A Tree

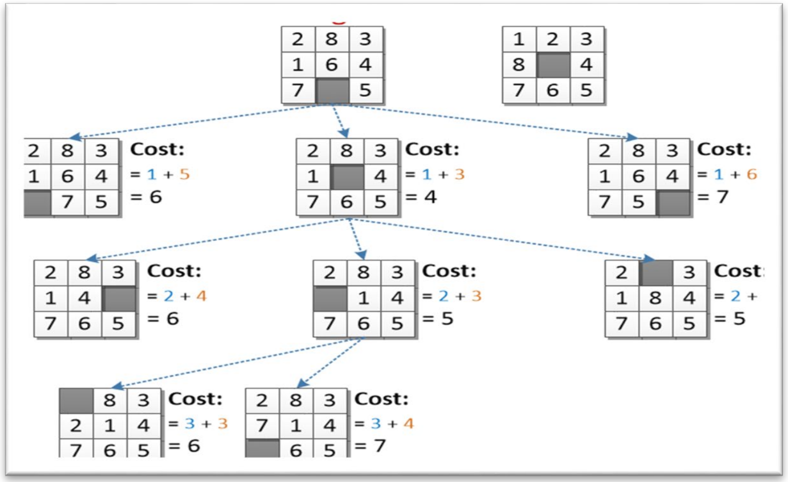
- Search Tree: A universal **representation** to organize the search process



Search in A Tree

- Search methods:
  - defined by picking the order of node expansion.
  - aim to identify the 'best' solution.
  - Search methods differ in how they explore the space, i.e., how they choose the node to expand NEXT.





What is A Good Search Method?

- Completeness: Does it always find a solution if it exists?
- Optimality: Does it always find the least-cost solution?
- Time complexity: # nodes generated/expanded.
- Space complexity: maximum #nodes in memory.

What is A Good Search Method?

In general, time and space complexity depend on:

- b – maximum # successors of any node in search tree.
- d – depth of the least-cost solution.
- m – maximum length of any path in the state space.

Un-informed Search Methods

- Use only the information available in the problem definition.
- Use NO problem-specific knowledge.

- ➢ Breadth-first search (BFS)
- ➢ Uniform-cost search (UCS)
- ➢ Depth-first search (DFS)
- ➢ Depth-limited search (DLS)
- ➢ Iterative deepening search (IDS)
- ➢ Bidirectional search

Breadth-First Search (BFS)

- Expand shallowest unexpected node.
- Implementation: a FIFO queue.

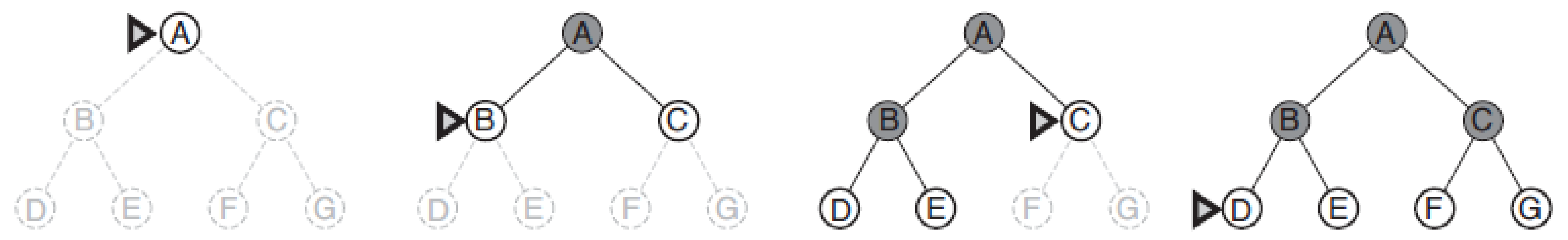


Figure 3.12 Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.

Breadth-First Search (BFS)

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

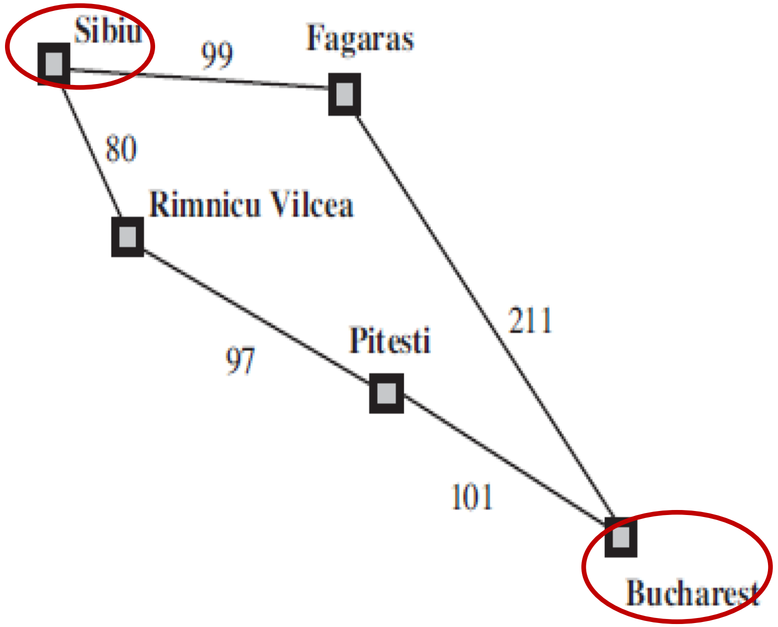
- Complete? Yes, if b is finite.
- Optimal? Yes, if costs on the edge are non-negative.
- Time? $O(b^{d+1})$
  - $1 + b + b^2 + \cdots + b^d + b(b^d - 1) = O(b^{d+1})$
- Space? $O(b^{d+1})$
  - keep every node in memory.

Memory + exponential time complexities are the biggest handicaps of BFS.

Uniform-Cost Search (UCS)

- The costs in the search tree may be different.
- Expand the cheapest unexpanded node.
- Implementation: a queue ordered by path cost, lowest first.

- Task: from Sibiu to Bucharest.
  - [0] {[Sibiu, 0]}
  - [1] {[Sibiu→Rimnicu, 80]; [Sibiu→Fagaras, 99]}
  - [2] {[Sibiu→Rimnicu→Pitesti, 177]; [Sibiu→Fagaras, 99]}
  - [3] {[Sibiu→Rimnicu→Pitesti, 177]; [Sibiu→Fagaras→Bucharest, 310]}
  - [4] {[Sibiu→Rimnicu→Pitesti→Bucharest, 278]; [Sibiu→Fagaras→Bucharest, 310]}



Uniform-Cost Search (UCS)

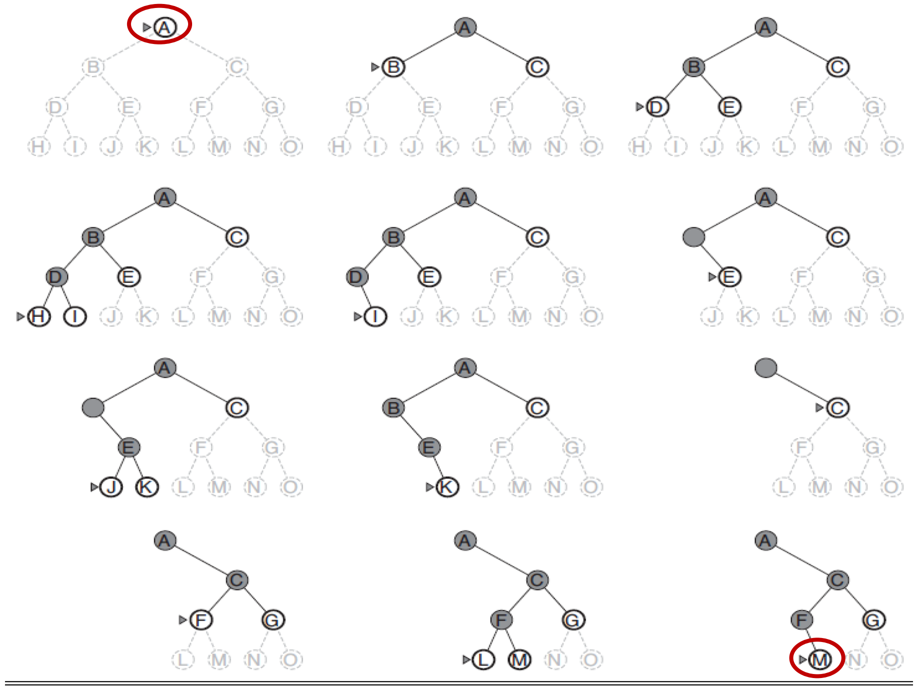
b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

- Complete? Yes, if every step cost $\geq \epsilon$.
- Optimal? Yes, if costs on the edge are non-negative.
- Time? Space? $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
  - C^*: the cost of the optimal solution.
  - every action costs at least ϵ.
  - $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ can be much greater than $O(b^{d+1})$.
  - When all step costs are equal, $O(b^{1+\lfloor C^*/\epsilon \rfloor}) = O(b^{d+1})$.

➢ When all step costs are equal, UCS is similar to BFS.

Depth-first Search (DFS)

- Expand deepest unexpanded node.
- Implementation: LIFO stack.
- Task: Search from A to M.
- Note: Once a node is expanded, it is removed from memory as soon as all its children are explored.



Depth-first Search (DFS)

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

- Complete? No, fail in infinite-depth space and space with loops.
- Optimal? No.
- Time? $O(b^m)$
  - Terrible if m is much larger than d.
  - If solutions are dense, may be much faster than BFS.
- Space? $O(bm)$ – linear!

➢ Space complexity is much lower than BFS.

Depth-Limited Search (DLS)

- DFS with depth limit l: nodes at depth l have no successors.
  - Limit l is defined based on domain knowledge.
    - e.g. a traveller problem with 20 cities → $l < 20$.
  - DLS is the variant of DFS.

- DLS overcomes the failure of DFS in infinite-depth space.

Depth-Limited Search (DLS)

- Complete? No, (Eps. $l < d$).
- Optimal? No.
- Time? $O(b^l)$
- Space? $O(bl)$

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

Iterative Deepening Search (IDS)

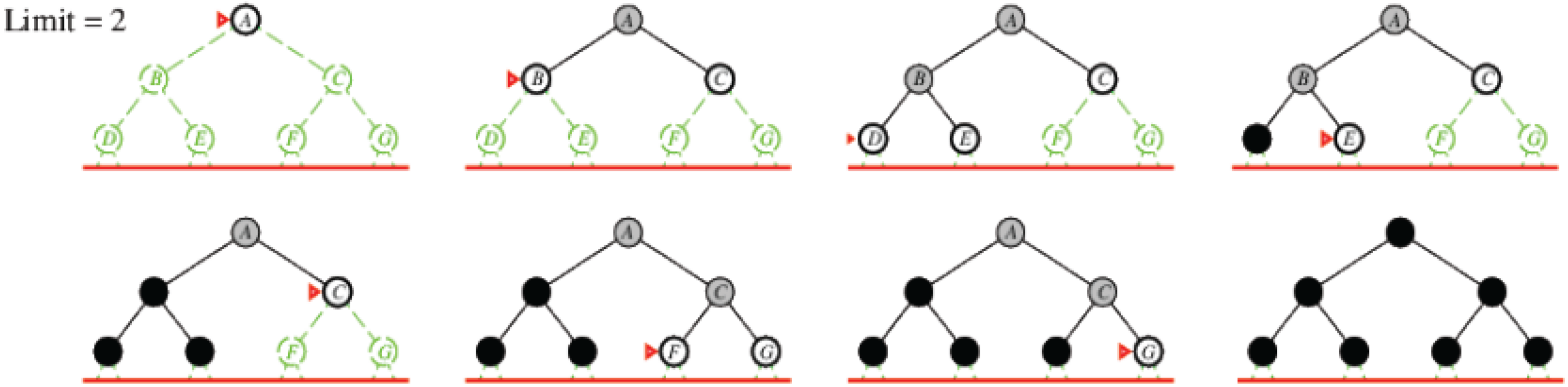
- Apply DLS with increasing limits.
- Combine the benefits of BFS and DFS.
  - Like BFS, complete when b is finite & optimal when the path cost is non-decreasing regarding depth of the nodes.
  - Like DFS, space complexity is $O(bd)$.

Iterative Deepening Search (IDS)

- Apply DLS with increasing limits.
- Combine the benefits of BFS and DFS.
  - Like BFS, complete when b is finite & optimal when the path cost is non-decreasing regarding depth of the nodes.
  - Like DFS, space complexity is $O(bd)$.

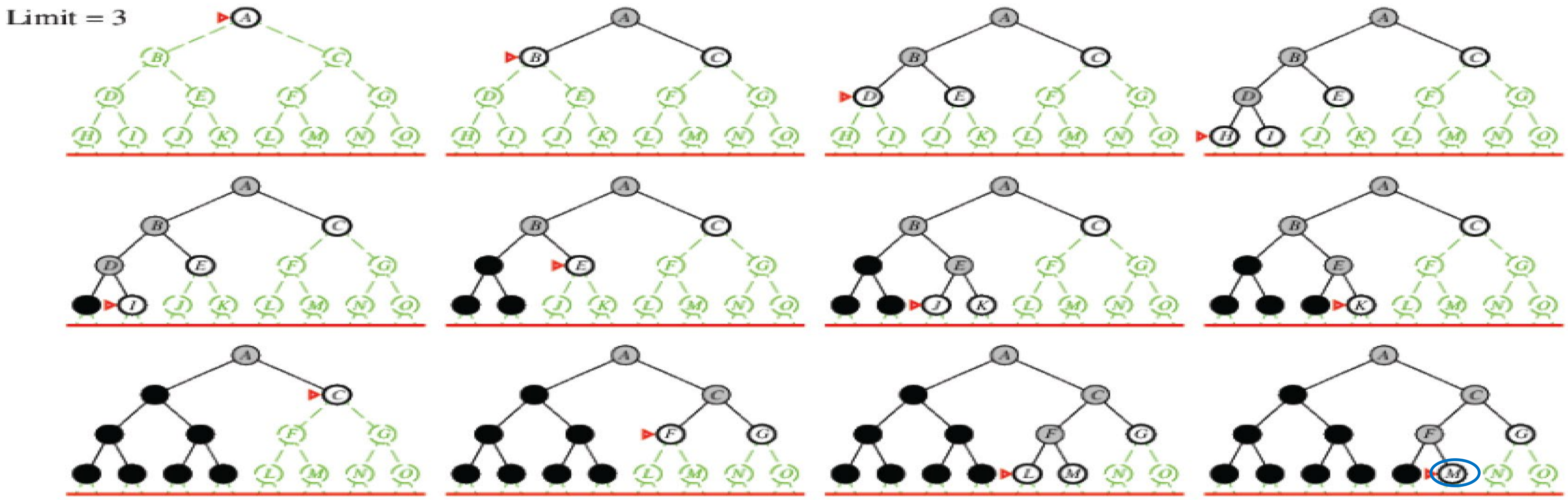
Iterative Deepening Search (IDS)

- Apply DLS with increasing limits.
- Combine the benefits of BFS and DFS.



Iterative Deepening Search (IDS)

- Apply DLS with increasing limits.
- Combine the benefits of BFS and DFS.



Iterative Deepening Search (IDS)

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

- Complete? Yes.
- Optimal? Yes, if costs on the edge are non-negative.
- Time? $O(b^d)$
  - $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d = O(b^d)$.
- Space? $O(bd)$

IDS is the preferred uninformed search method when the search space is large and the depth of the solution is unknown.

Bidirectional Search

- Search from both directions simultaneously.
  - Replace single search tree with two smaller sub trees.
  - Forward tree: forward search from source to goal.
  - Backward tree: backward search from goal to source.
- Goal test: two sub-graphs intersect.
- Complete? Yes, if BFS is used in both search.
- Optimal? Yes, if BFS is used & paths have uniform cost.
- Time? Space? $O(b^{d/2})$
- Disadvantage: Not always applicable
  - Reversible actions?
  - Explicitly stated goal state?

Basic Search Methods

b – maximum # successors of any node in search tree.
d – depth of the least-cost solution.
m – maximum length of any path in the state space.

PF Metric	Breadth-first Search	Uniform-cost Search	Depth-first Search	Depth-limited Search	Iterative Deepening	Di-directional Search
Complete?	Yes*, if b is finite.	Yes*, if step costs≥ ϵ.	No, infinite loops can occur.	No. (Eps. $l < d$)	Yes	Yes*, if BFS used for both search.
Optimal?	Yes*, if costs on the edge are non-negative.	Yes	No,	No	Yes*, if costs on the edge are non-negative.	Yes*, if BFS is used & paths have uniform cost.
Time?	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space?	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$

From Basic Search to Heuristic Search

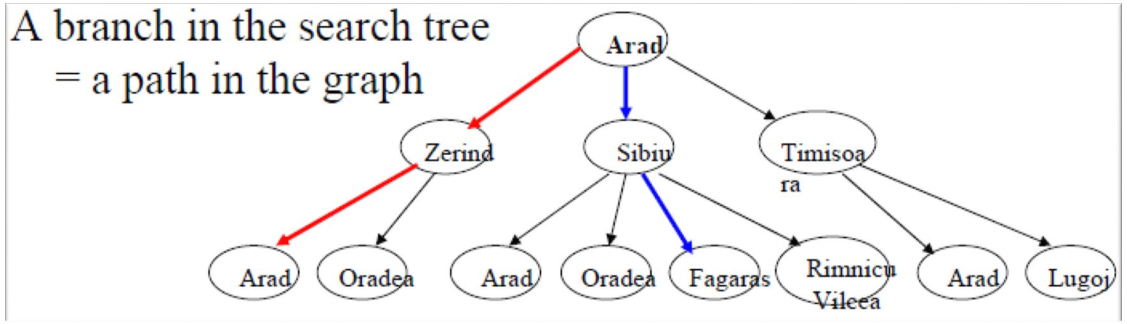
- Uninformed search uses NO domain (problem-specific) knowledge.

- Which node to expand is decided arbitrarily (sometimes randomly).

- Heuristic search uses domain knowledge.
  - Heuristics: which part of the search space to explore? ⇒ help direct the search.

Heuristic Search

- How to make use of domain knowledge?
  - Define an evaluation function $f(x)$ at node x.
  - node x with the **lowest** $f(x)$ is expanded first
- Define a good f endows with some intelligence ➔ A little AI now.



How to Design the Evaluation Function?

- Depends on your own...
- Basically, introduce a *heuristic function* $h(x)$ **estimates** the cheapest cost from x to the goal state.

  (1) $h(x) = 0$ if x is the goal node.
  (2) nonnegative.
  (3) problem-specific.

- Example: $h_{SLD}(x) =$ straight-line distance from node x to goal.

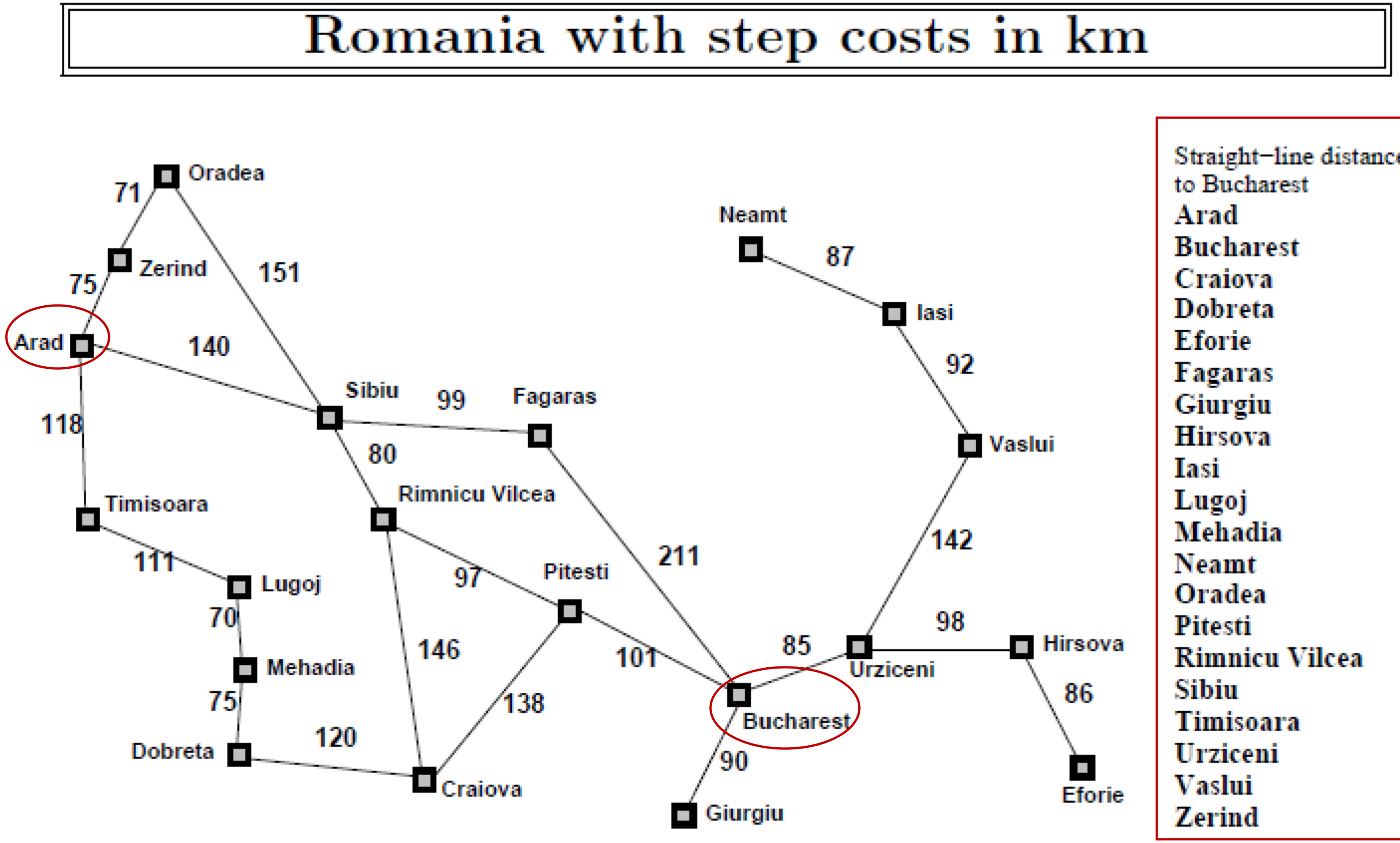
Greedy Best-first Search

- Expand node n that has the minimal $f(x) = h(x)$.

- Interpretation: Expand the node that seems closest to the goal.

- A generalization of UCS, i.e., use $f(x)$ instead of the path cost from the start to node n.

Greedy Search: Illustration



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

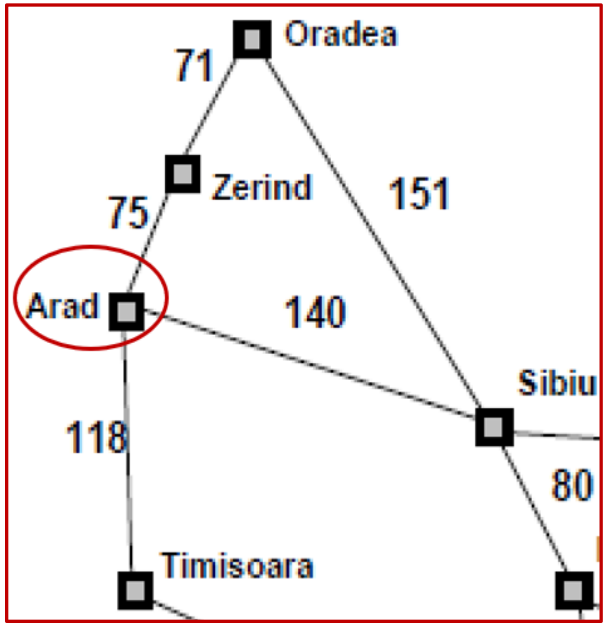
$$f(x) = h_{SLD}(x)$$

domain-knowledge

Greedy Search: Illustration

Straight−line distance to Bucharest

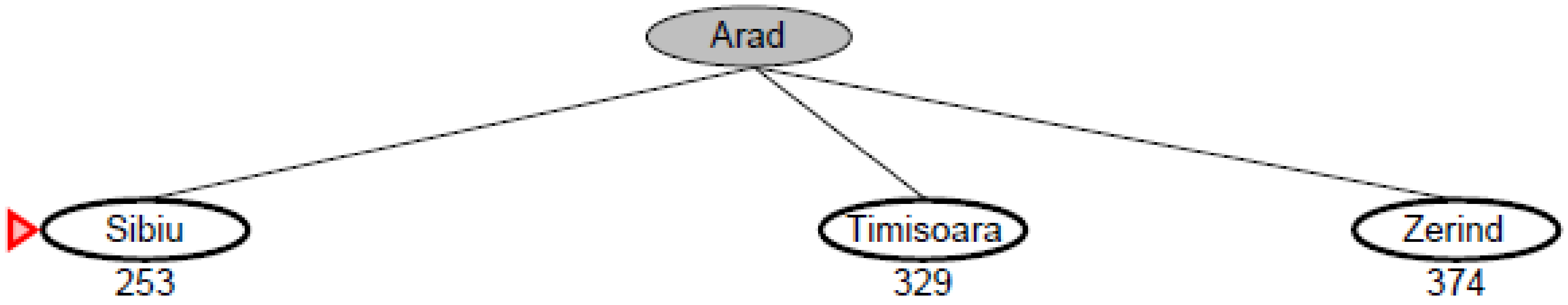
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

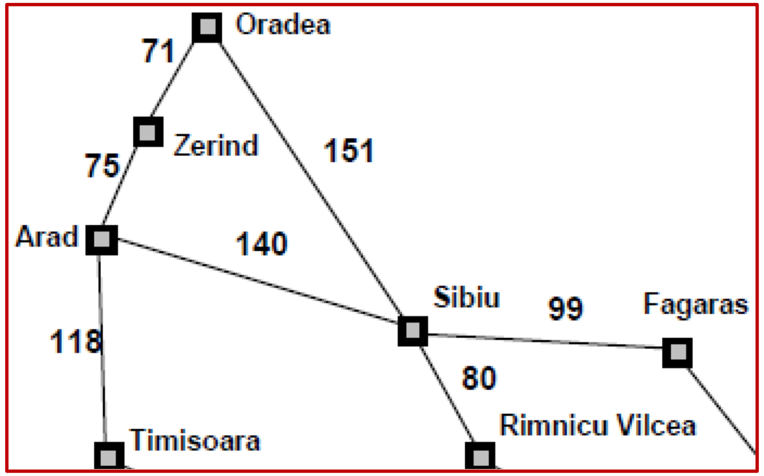


Greedy Search: Illustration

Straight−line distance to Bucharest

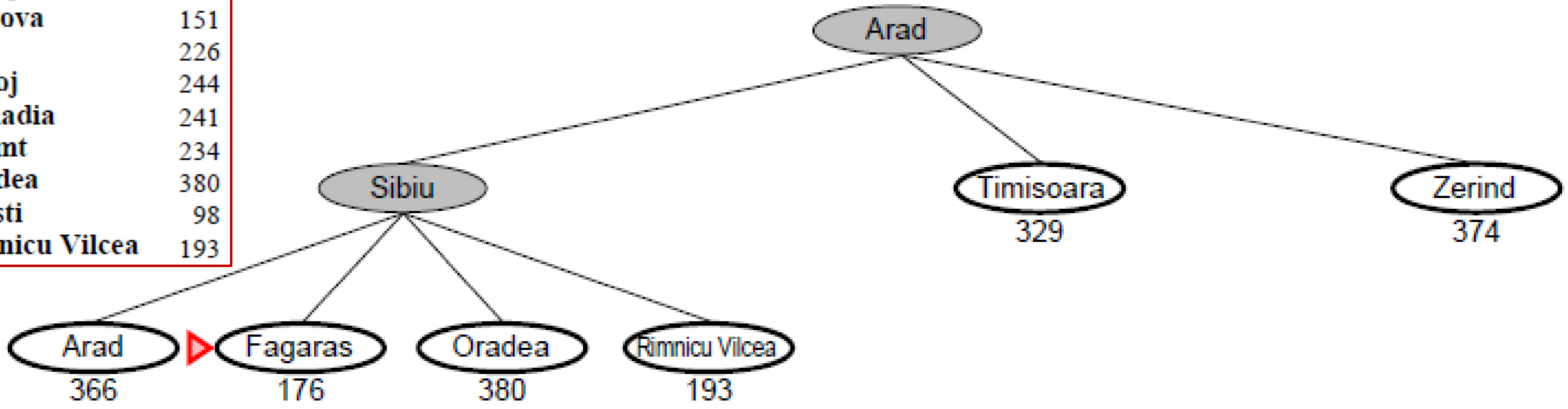
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



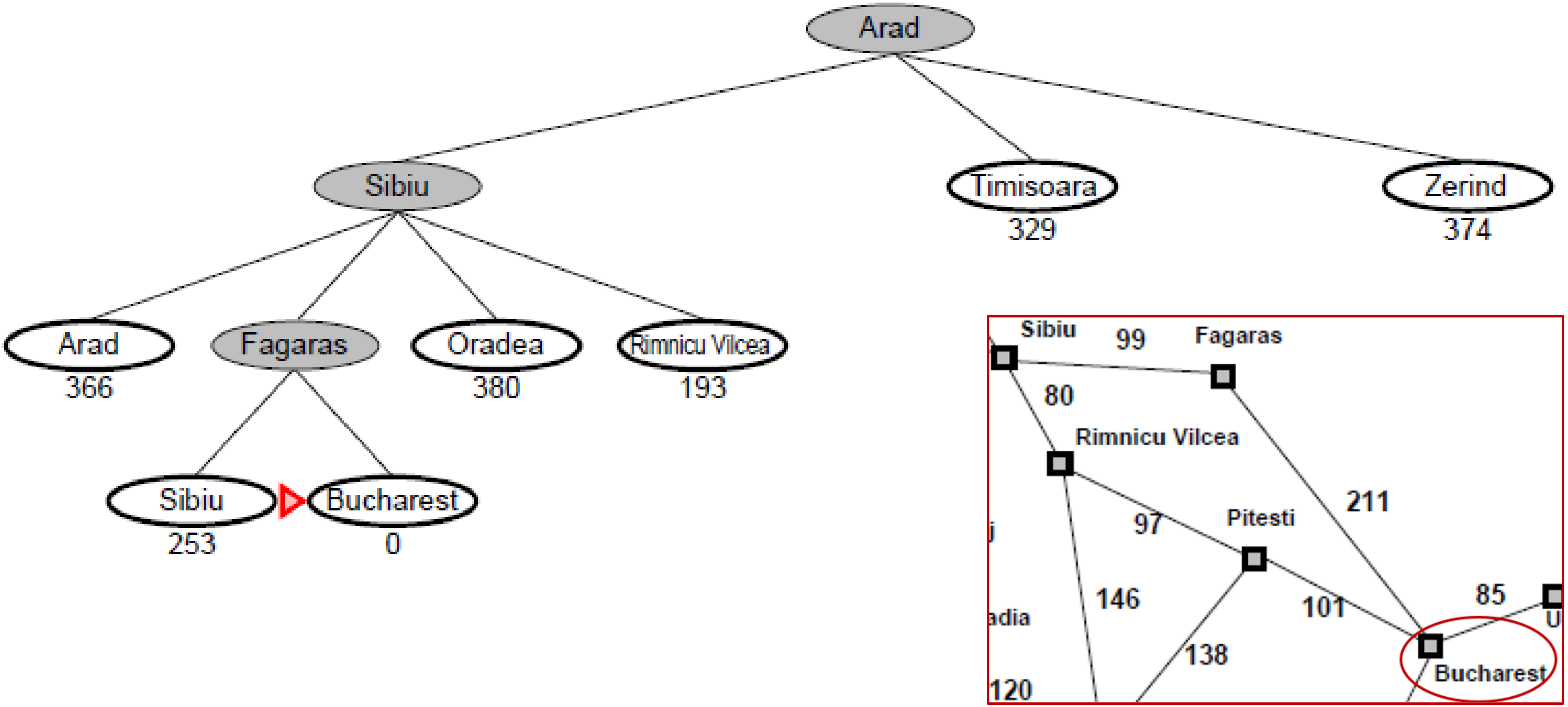


Greedy Search: Illustration

Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193



Greedy Search: Illustration



PF Metrics

- b – maximum branching factor of the search tree.
- d – depth of the least-cost solution.
- m – maximum depth of the state space.

- Complete? Yes in finite space with repeated-state checking.
- Optimal? No.
- Time? $O(b^m)$, but a good heuristic can give drastic improvement.
- Space? $O(b^m)$, keep all nodes in memory.

A* Search

- Idea: avoid expanding paths that are already expensive.
- Expand the node n that has the minimal $f(x) = h(x) + g(x)$
  - $g(x)$: cost so far to reach x.
  - $h(x)$: estimated cost to goal from x.
  - ➢ $f(n)$: estimated total cost of path through n to goal.

➢ A generalization of UCS as well.

A*: PF Metrics

- b – maximum branching factor of the search tree.
- d – depth of the least-cost solution.
- m – maximum depth of the state space.

- Complete? Yes.
- Optimal? Yes*, if h is admissible.
- Time? $O(b^d)$.
- Space? $O(b^d)$, keep every node in memory.

Admissible Heuristic

- Heuristic function h is admissible if:

$$\forall x \rightarrow h(x) \leq h^*(x), \qquad h^*(x)\text{: the true cost from } x \text{ to goal.}$$

- A∗ with h is optimal if h is admissible.

Notation: S – start, G – goal, n – a node on optimal path, n' – a non-optimal goal, c^* – cost of the optimal path.

[Proof] A∗ is always finds the optimal path ⇔ A∗ always pick n over n' ⇔ $f(n) < f(n')$.

Known: h is admissible ⇒ $h(n) < c^*(n, G)$. (∵ G is a goal state)

① $f(n') = g(n') + h(n') = g(n') + 0 > c^*$. (∵ n' is the goal node & c^* the smallest cost)

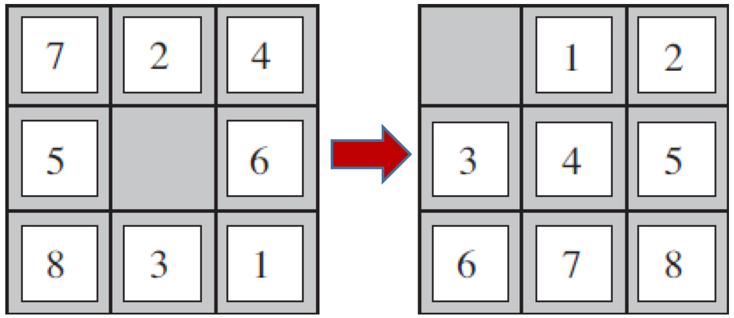
② $f(n) = g(n) + h(n) < g(n) + c^*(n, G) = c^*(S, n) + c^*(n, G) = c^*$.

Conclusion: combine ① &② ⇒ $f(n) < f(n')$ □

Admissible Heuristic: Examples

- Route Planning: $h_{SLD}(n)$: Admissible, since no solution path will ever shorter than straight-line connection.

- Eight-puzzle: $h_{mis}(n) = $ #misplaced titles $\in [0{,}8]$: Admissible.



Search Efficiency of Admissible Heuristic

- In general, an A* with an admissible heuristic
  - expands all nodes with $f(x) < c^*$
  - output the optimal solution.
- Different heuristic functions could lead to **significantly different** efficiency.

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Search Efficiency of Admissible Heuristic

- For admissible h_1 and h_2, if $h_2(x) \geq h_1(x), \forall n$, then h_2 dominates h_1 and is more efficient for search.

- Why? See two extreme examples

(1) The trivial $h_0(x) = 0$: No help for searching ⇒ Not efficient.

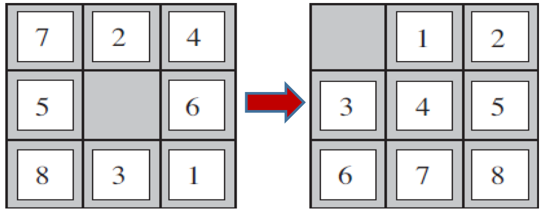
(2) The perfect $h^*(x)$ = the true cost from x to the goal: lead directly to the best path

How to Design Admissible Heuristics

Consider relaxed problems

For eight puzzle:

- Rule: A tile can only move to the adjacent empty square.
- Relaxed rules:
  - R1: A tile can move anywhere $\Rightarrow h_{mis}(n) = \#$(misplaced titles).
  - R2: A tile can move one step in any direction regardless of an occupied neighbour $\Rightarrow h_{1stp}(n) = \#$(1-step move) to reach goal.
  - ➢ h_{mis} and h_{1stp} are admissible.
- Note: optimal solutions with R1, R2 are easier to find.

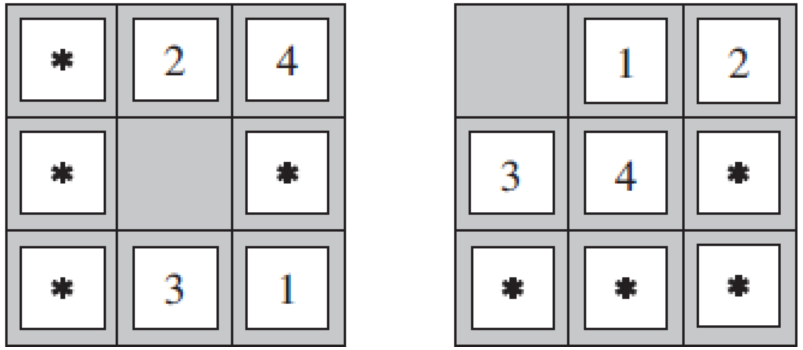


How to Design Admissible Heuristics

Consider sub-problems

For eight puzzle:

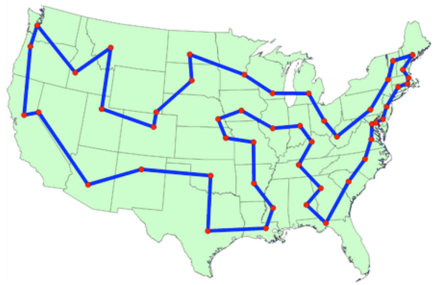
- Subproblem: See Fig.1.
    - Task: get tiles 1, 2, 3 and 4 into their correct positions.
    - Relaxation: move them disregard the other tiles.
- Theory: cost*(subproblem)<cost*(8-puzzle).



A sub-problem of 8-puzzle.

Summary of Heuristic Search

- Leverage on problem-specific knowledge to introduce search biases into a tree search process, to more efficiently obtain a desired solution (than un-informed search).
- Search bias: among many options, which to test in the next step?
- Completeness and Optimality is less emphasized in practice than in our lectures, for:
  - Very difficult (if not impossible) to design a good admissible heuristic.
  - For many problems, **optimality could NOT be ensured** with affordable time complexity.



The Christofides' algorithm has an approximation ratio of 3/2 and time complexity of $O(n^2 \log(n))$ for the metric Traveling Salesman Problem.

To be continued