Uninformed search

Uninformed search strategies

Uninformed strategies use only the information available in the problem definition

Breadth-first search

Uniform-cost search

Depth-first search

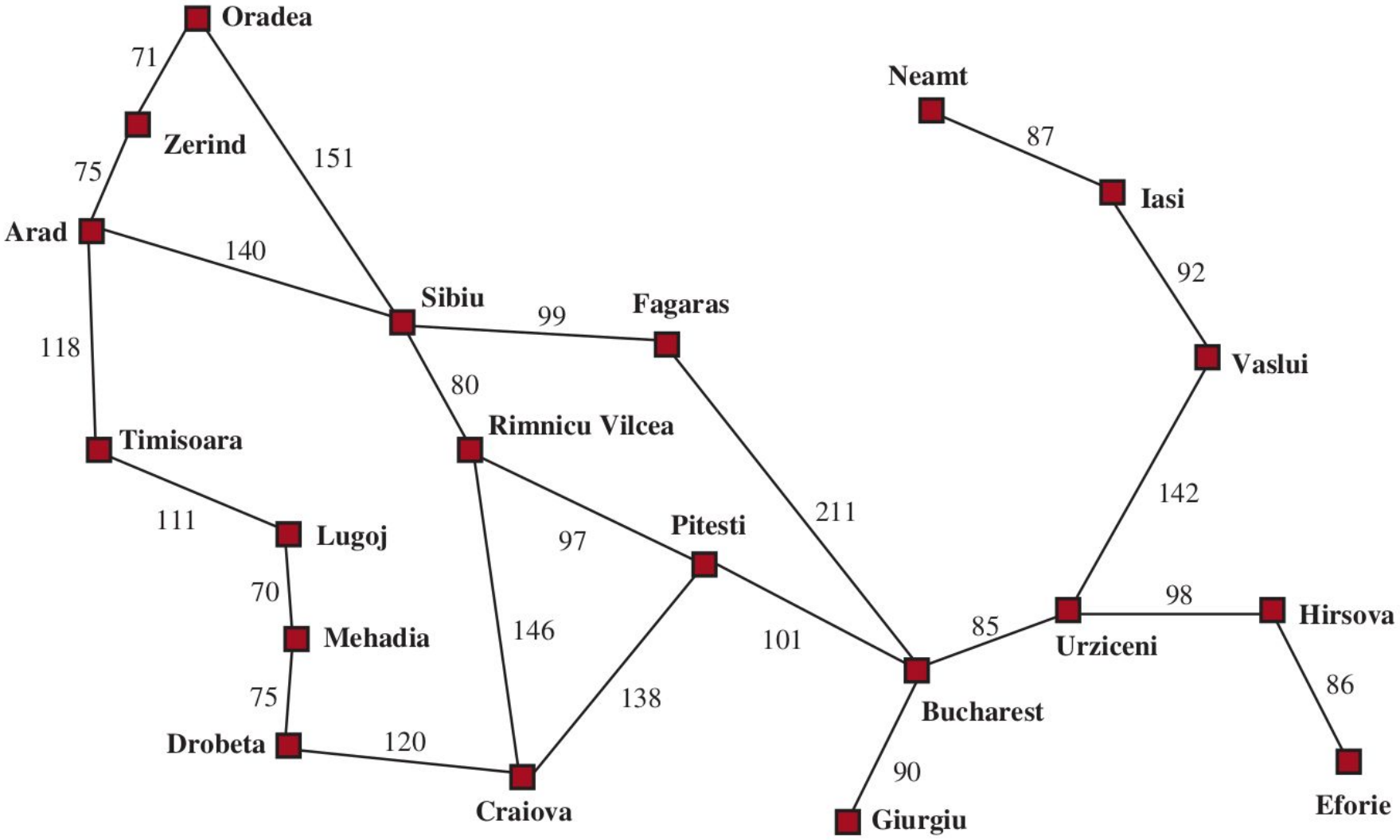
Depth-limited search

Iterative deepening search

Problem Solving

- Goal Formulation
- Problem Formulation
  - Initial State
  - Actions (operators)
  - Transition Model
  - Goal Test
  - Path Cost

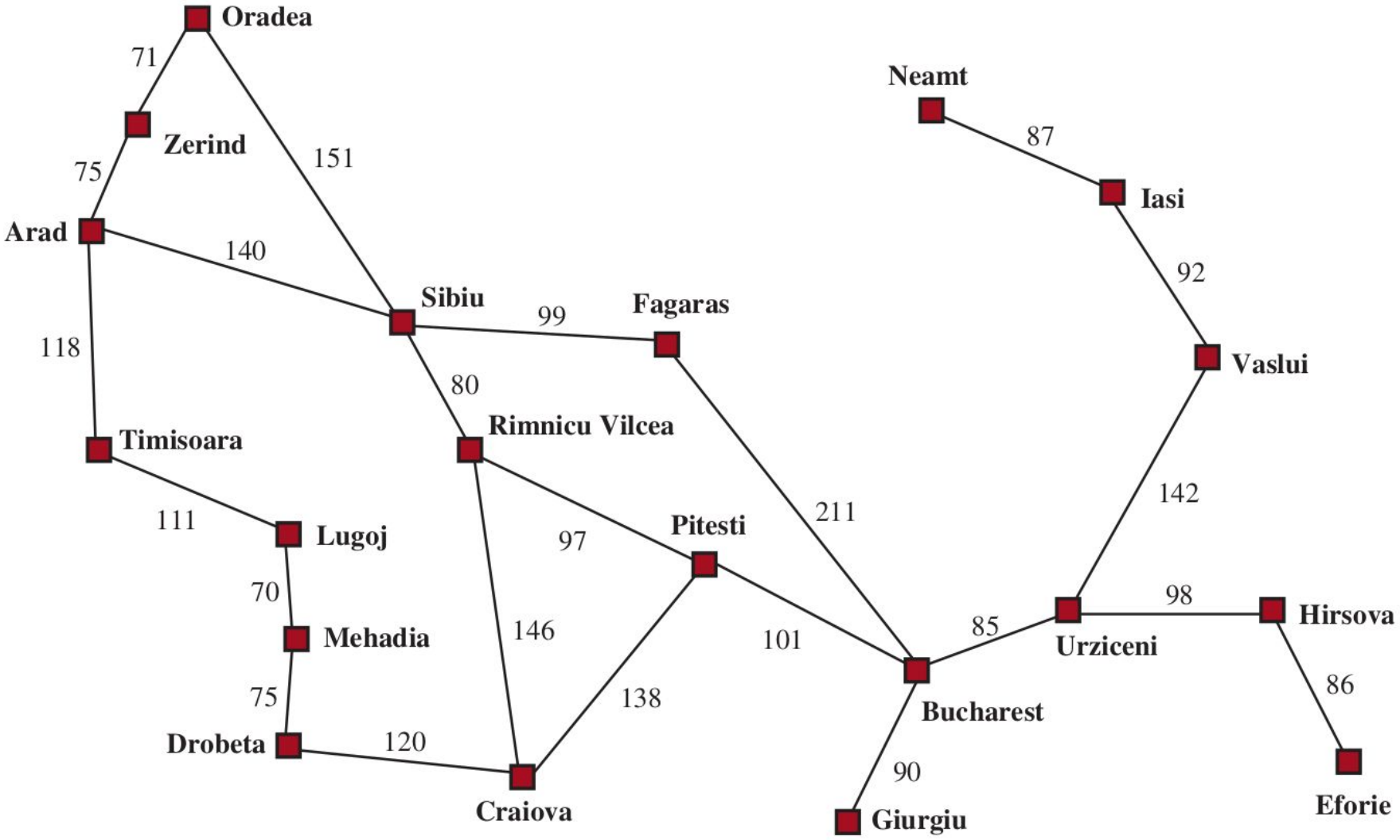
Example: Romania



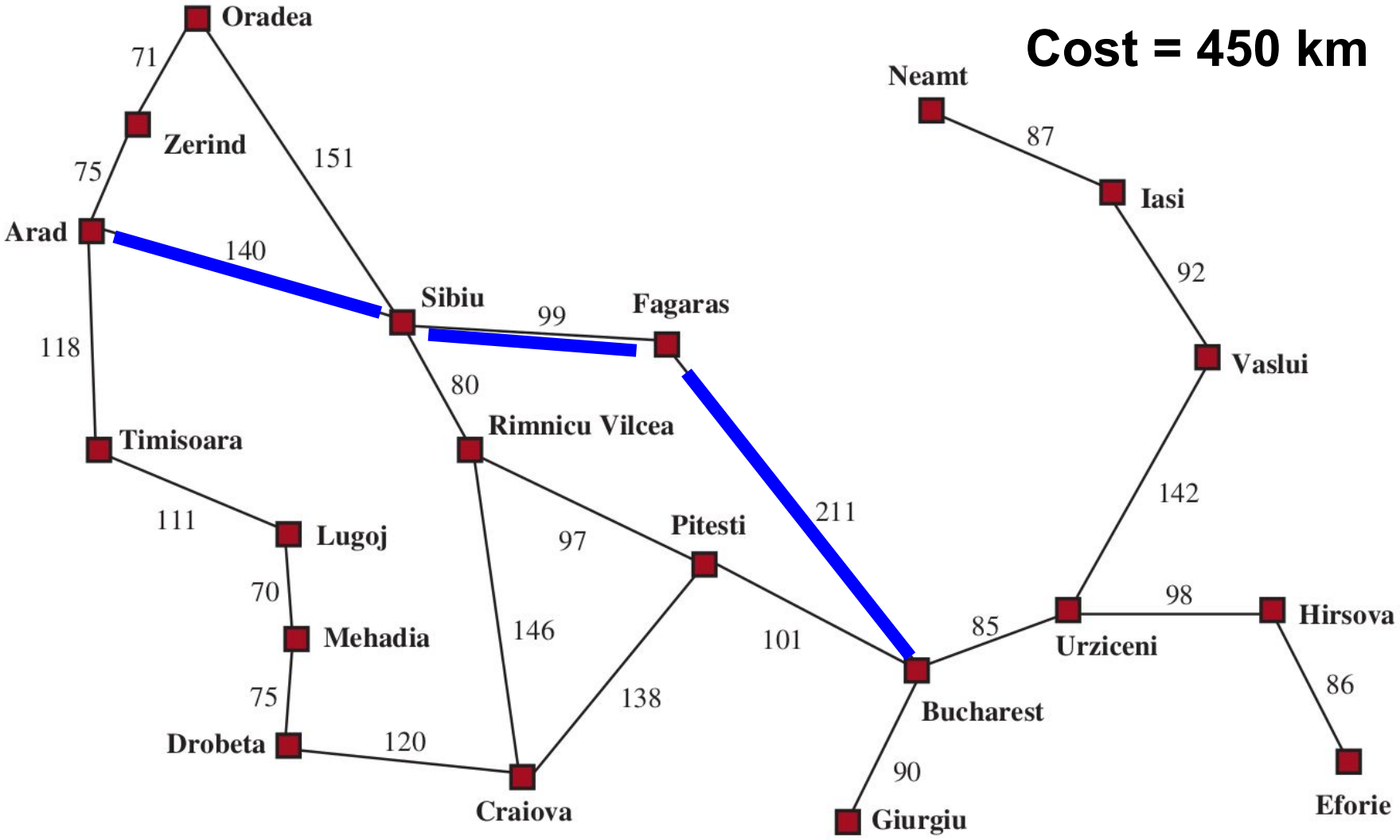
Example: Romania

- **States**: x = any city in Romania
- **Initial state**: e.g., at Arad
- **Actions**: (or successor function S(x)), e.g., driving from Arad to Zerind, Arad -> Zerind
- **Transition model**: returns the city reached by driving
- **Goal state**: x= “at Bucharest”
- **Action cost**: e.g., Sum of the distances, number of operators executed.

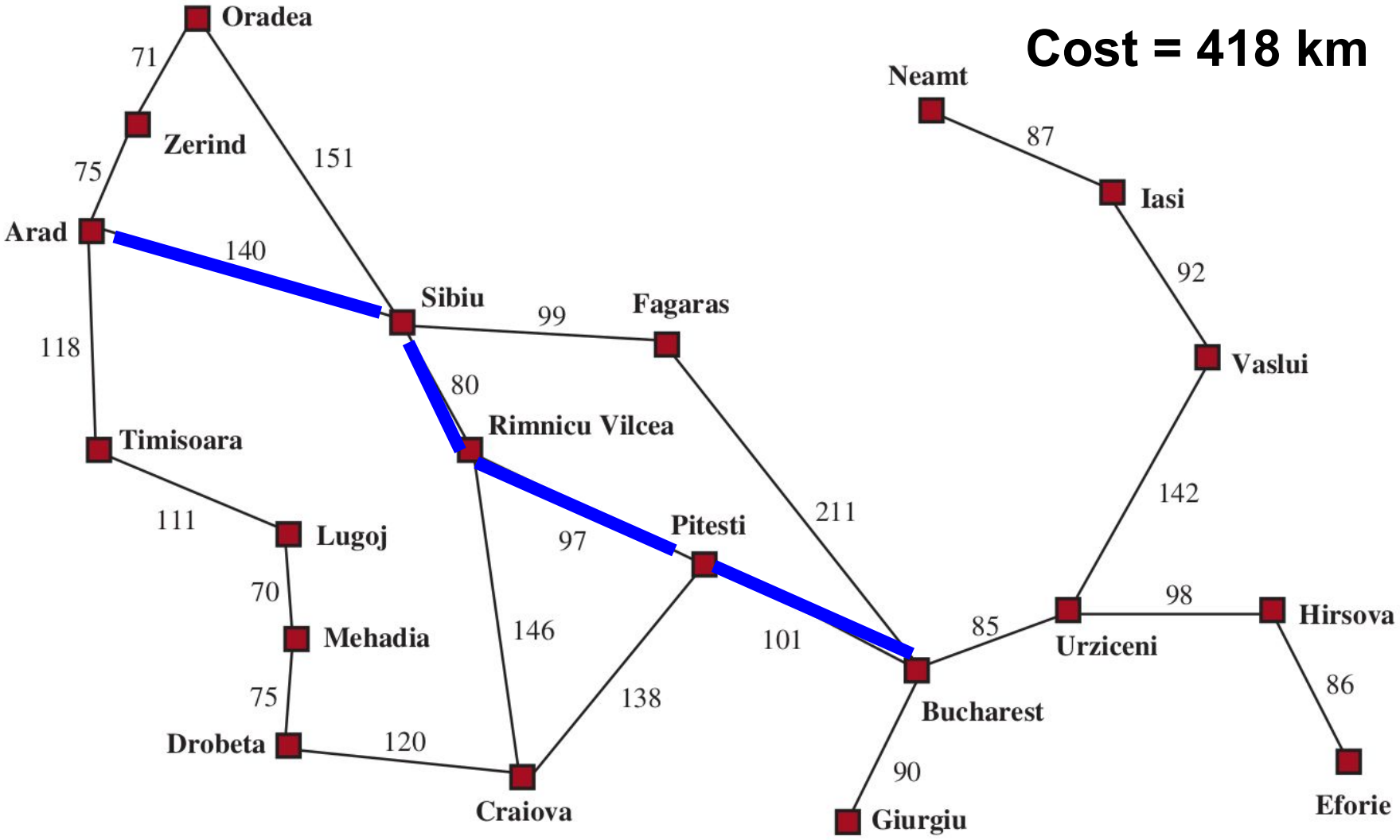
Example: Romania



Example: Romania



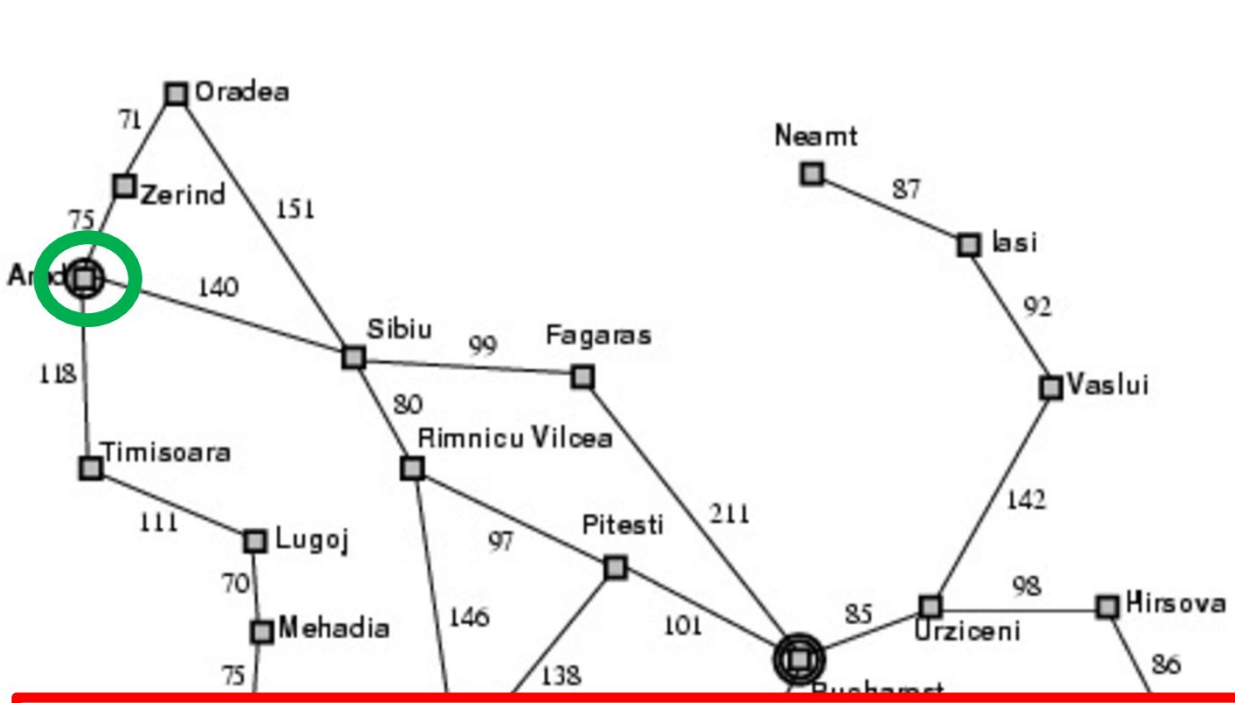
Example: Romania



Uniform Cost Search

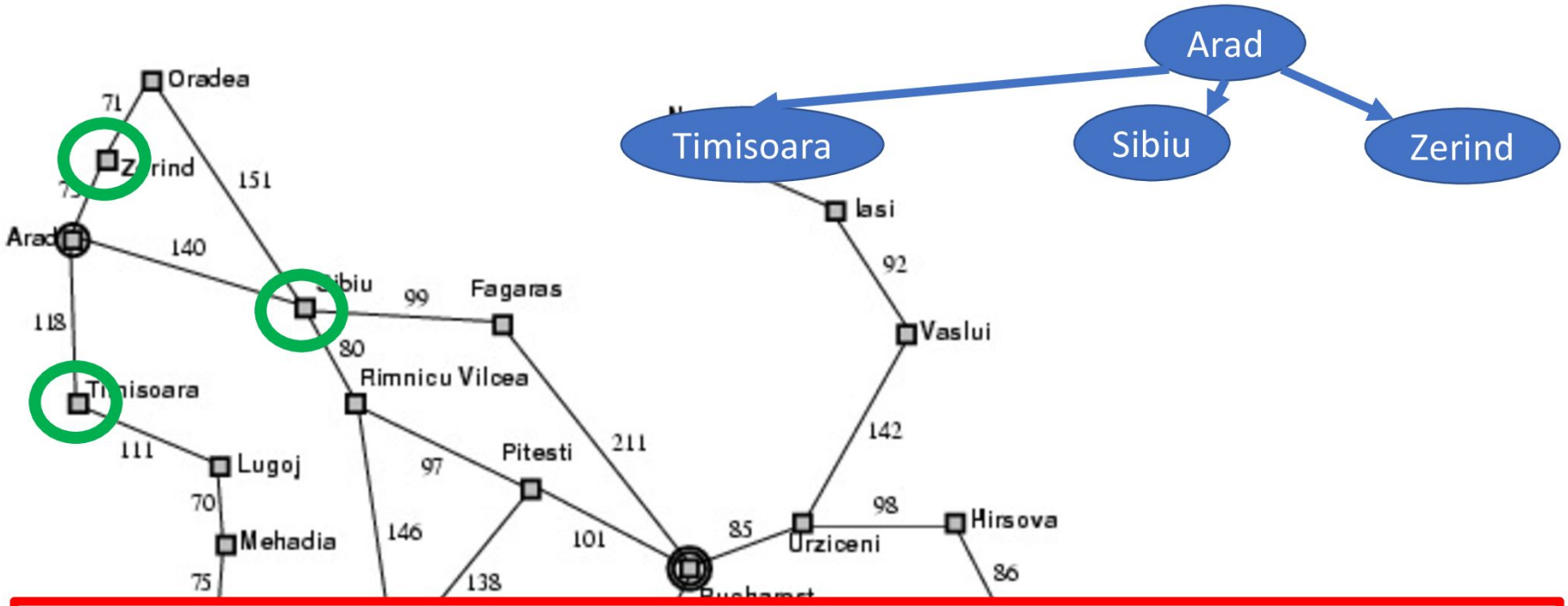
- UCS enumerates states in order of increasing past cost.
- Assumption: All action costs are non-negat $\text{Cost}(s, a) \geq 0$
- Use a Priority Queue to order nodes in Frontier, sorted by path cost
- Let g(n) = cost of path from start node s to current node n
- Sort nodes by increasing value of g

Uniform Cost Search



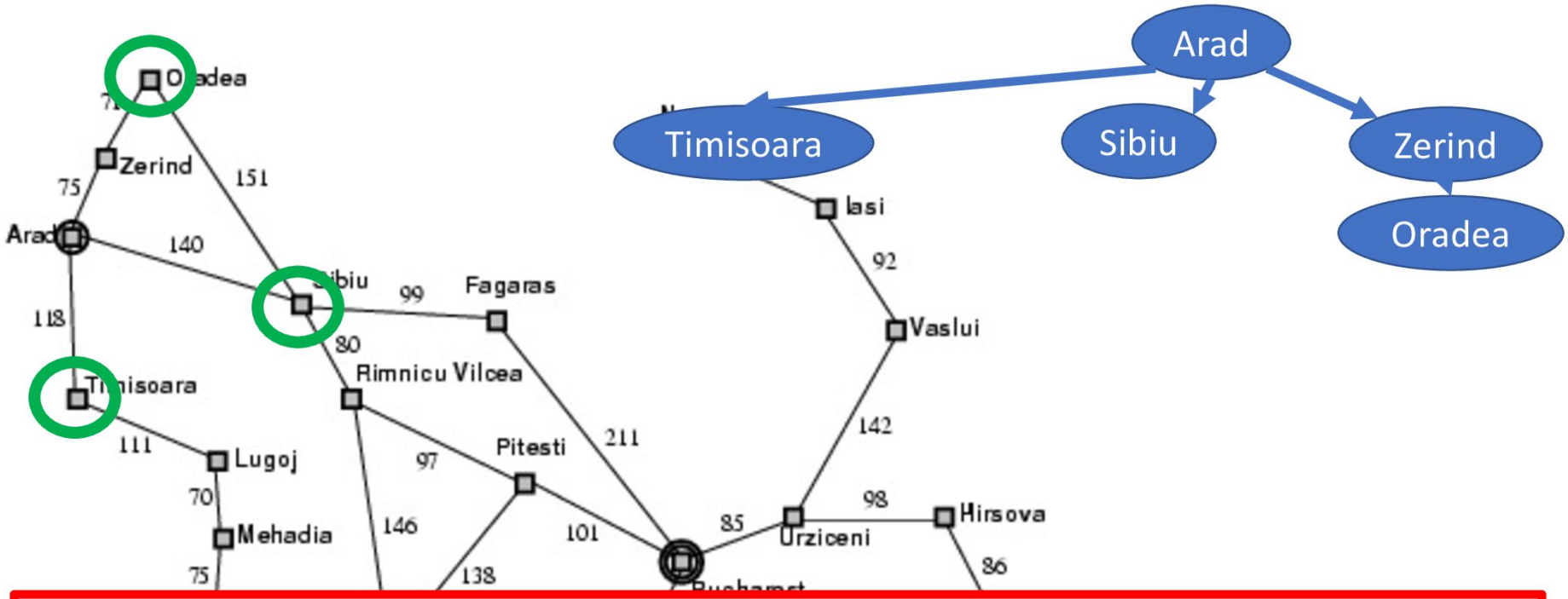
Arad:0

Uniform Cost Search



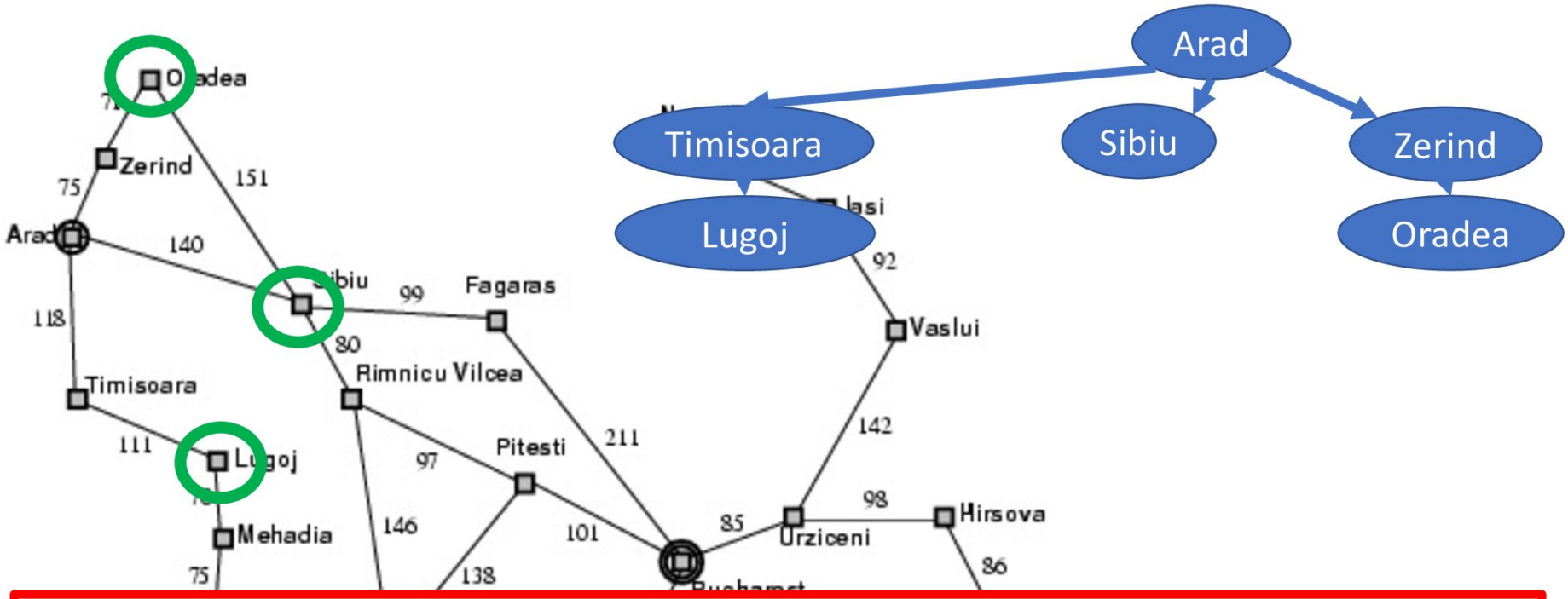
Zerind:75, Timisoara:118, Sibiu:140

Uniform Cost Search



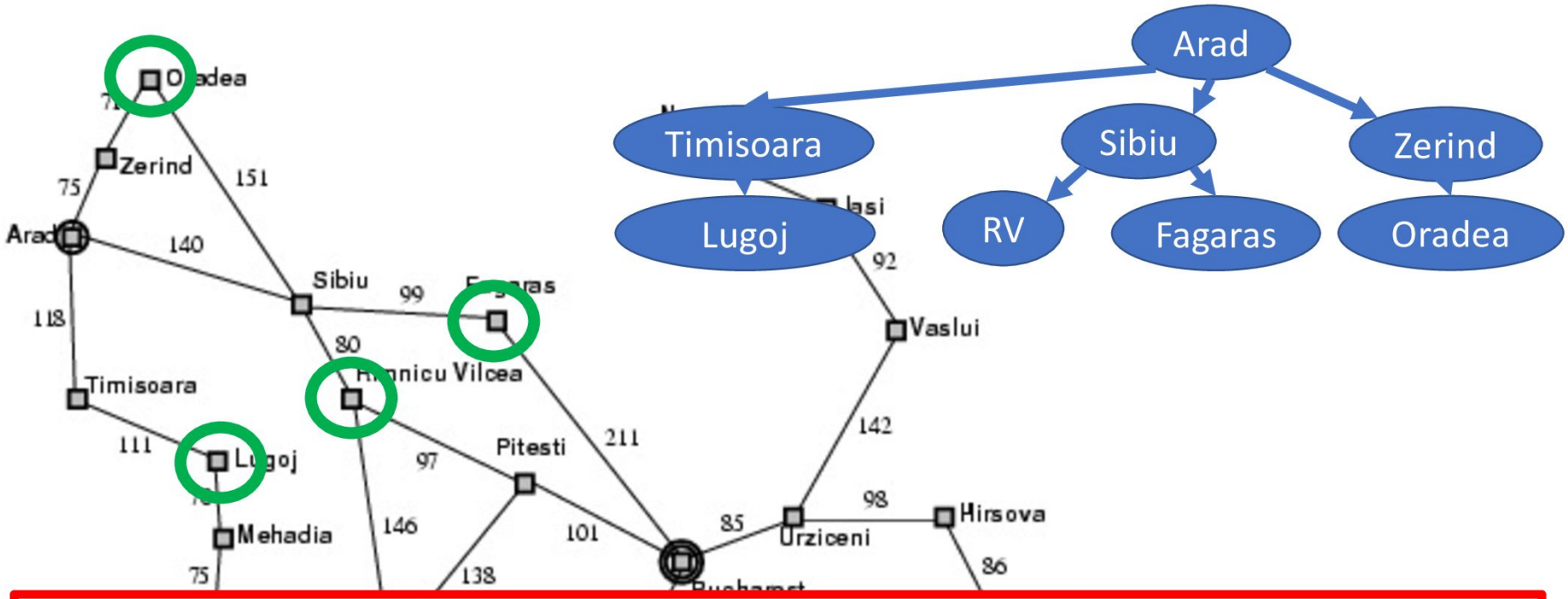
Timisoara:118, Sibiu:140, **Oradea:146**

Uniform Cost Search



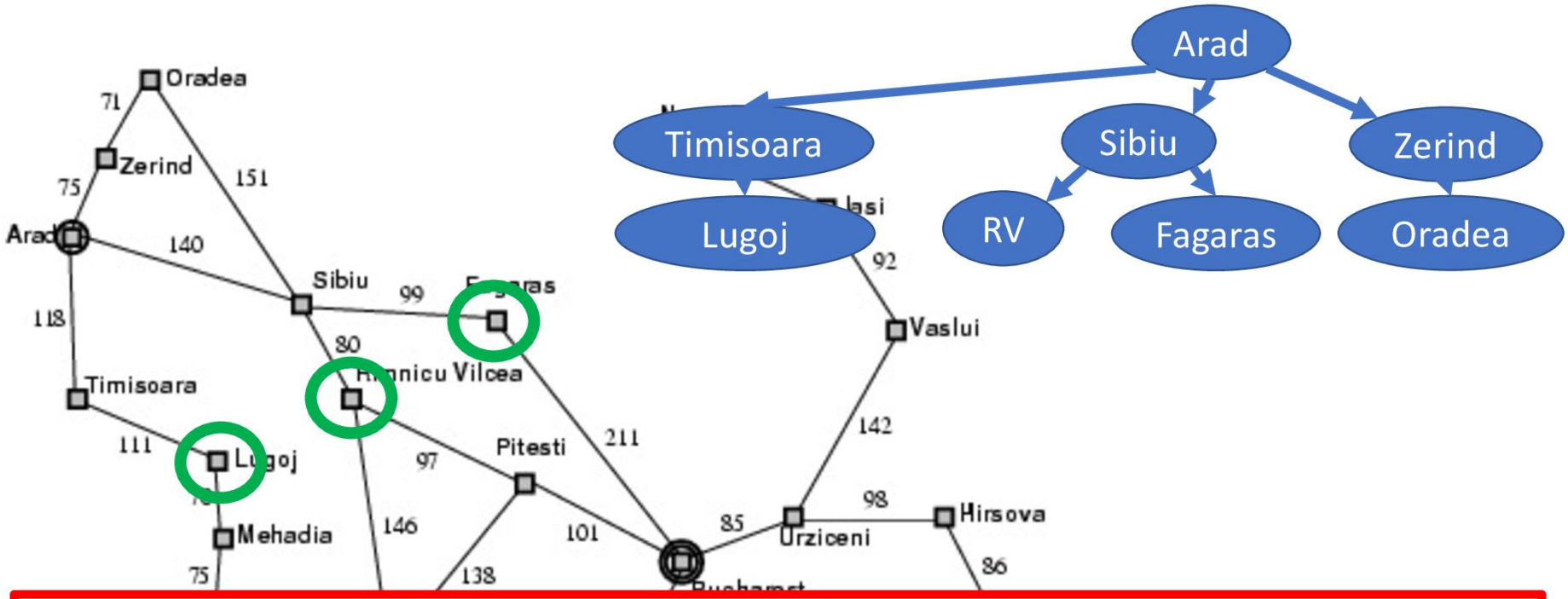
Sibiu:140, Oradea:146, **Lugoj:239**

Uniform Cost Search



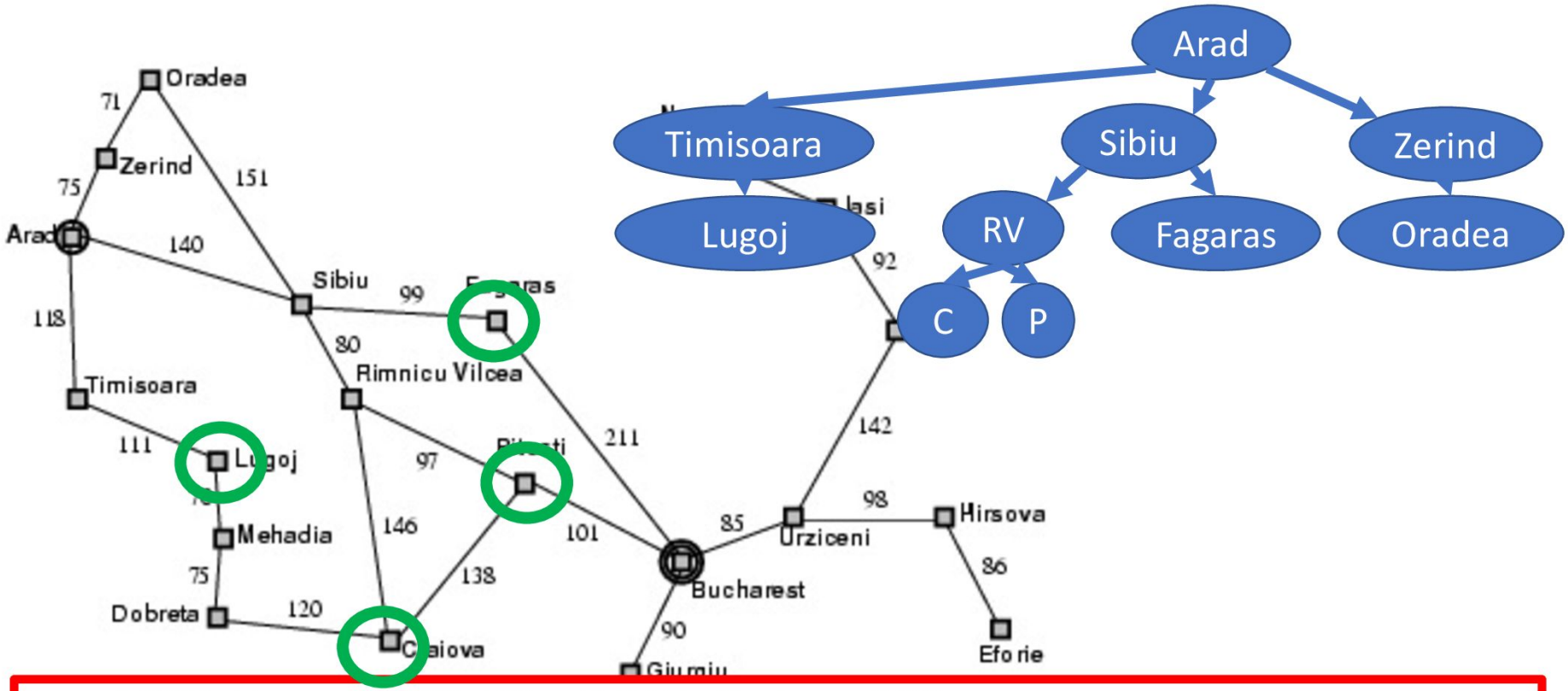
Oradea:146, **Ramnicu Valcea:220**, Lugoj:239, **Fagaras:239**

Uniform Cost Search



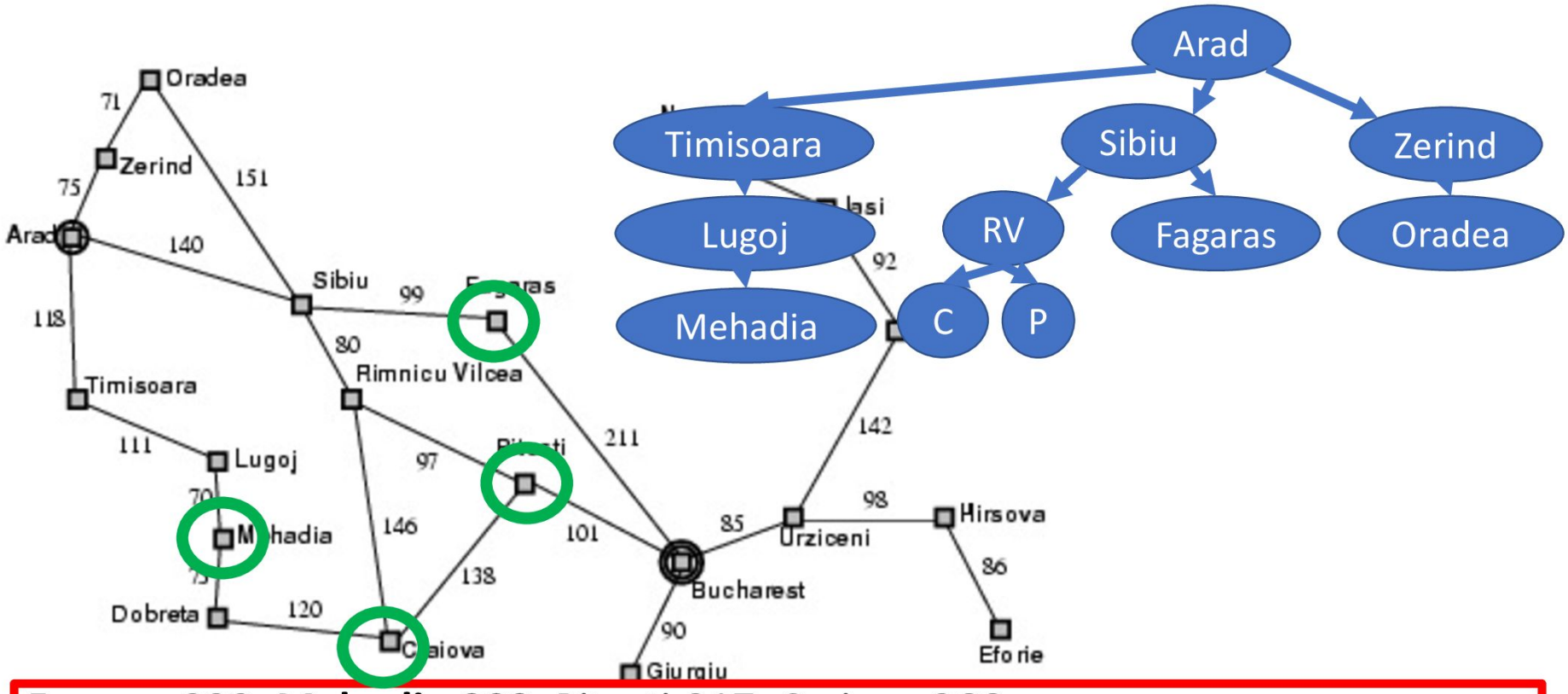
Ramnicu Valcea:220, Lugoj:239, Fagaras:239

Uniform Cost Search



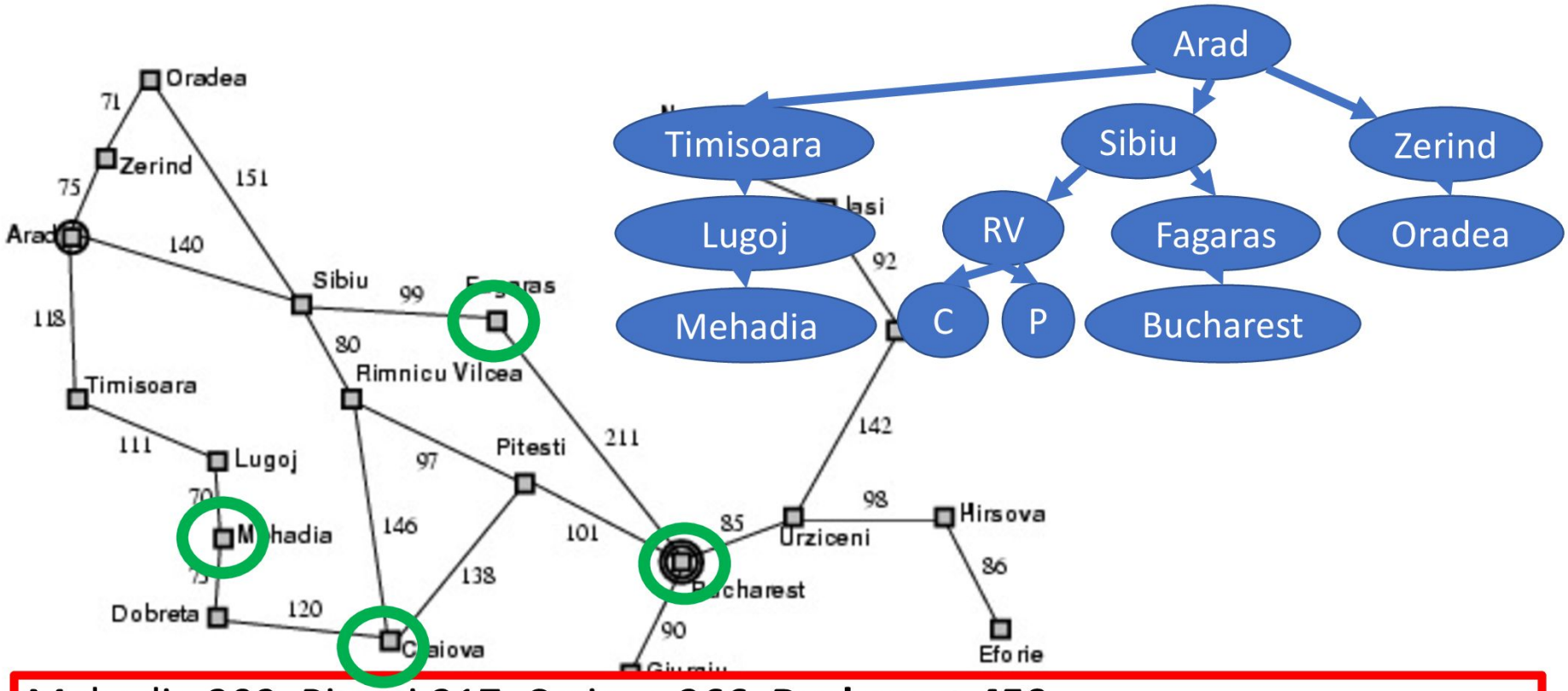
Lugoj:239, Fagaras:239, **Pitesti:317, Craiova:366**

Uniform Cost Search



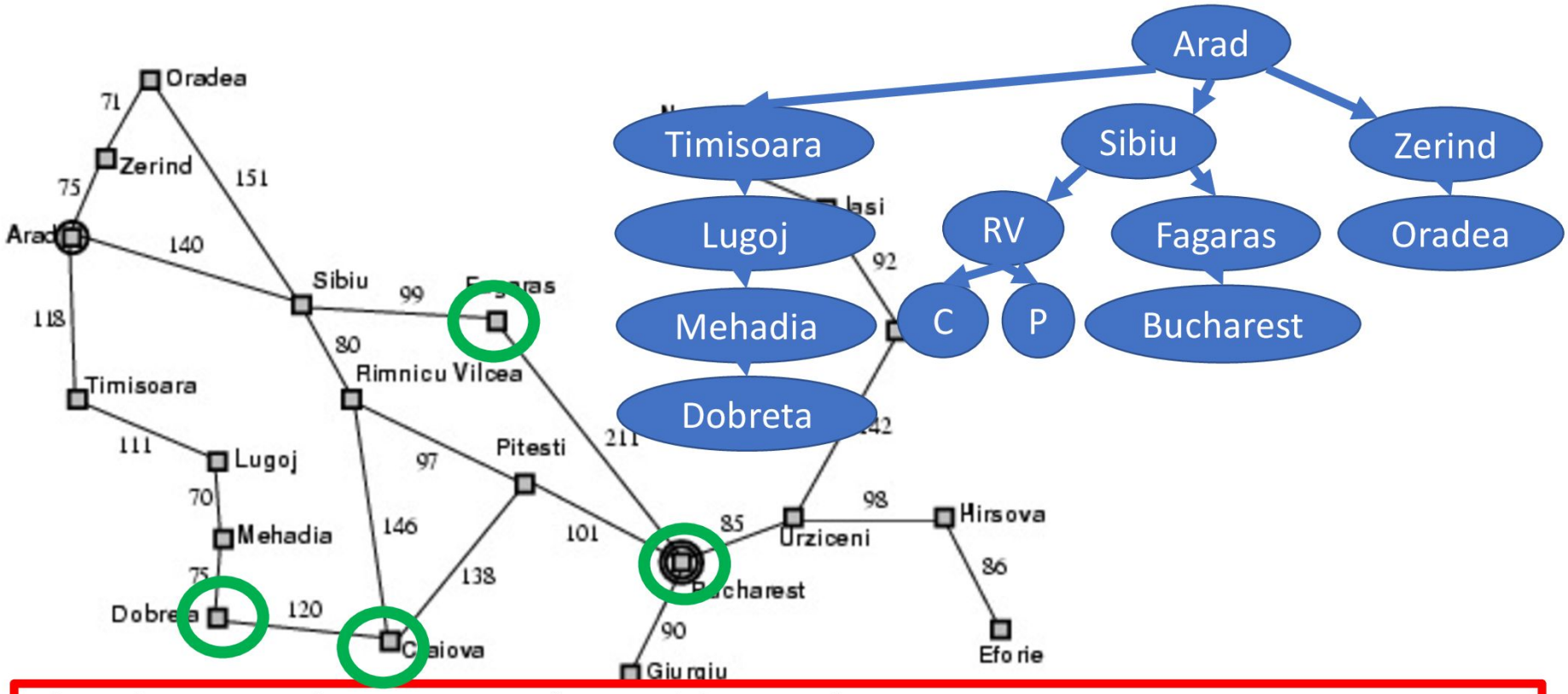
Fagaras:239, **Mehadia:309**, Pitesti:317, Craiova:366

Uniform Cost Search



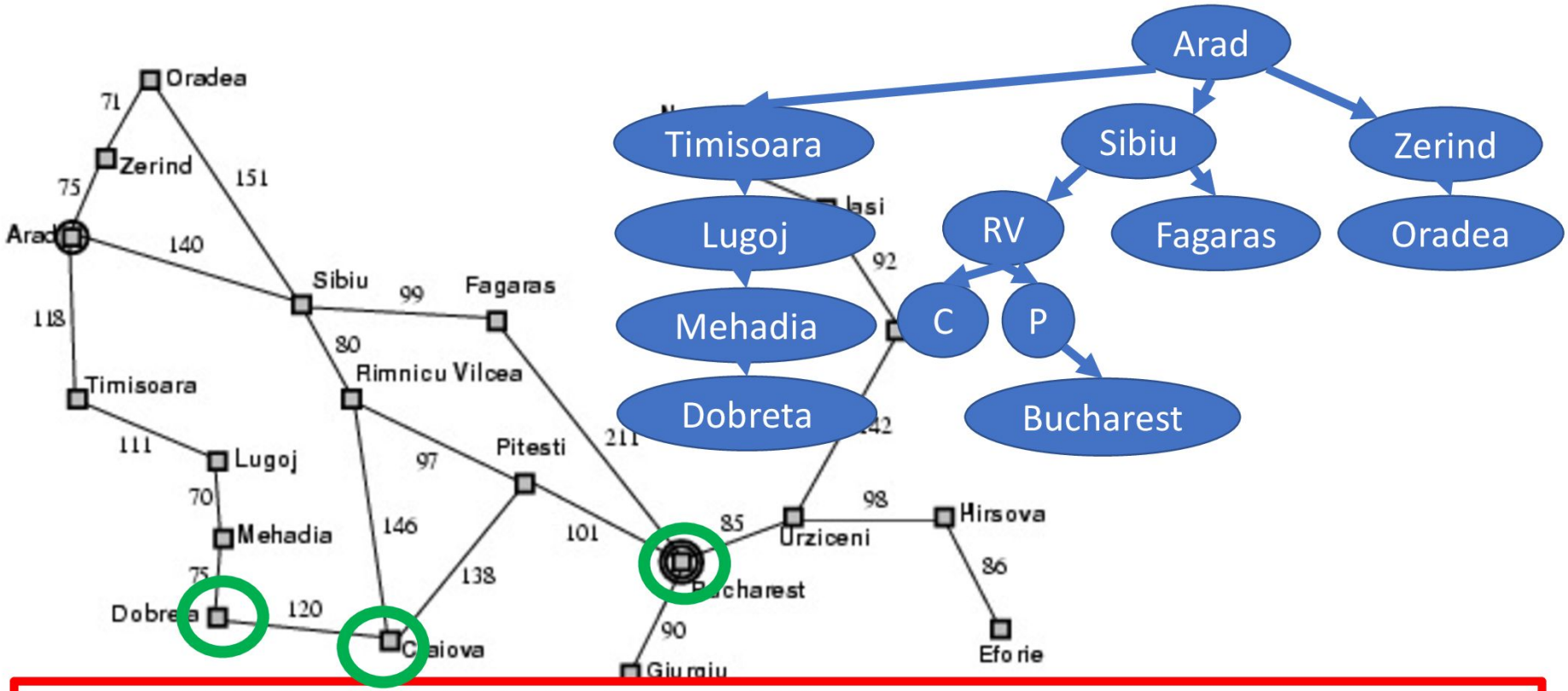
Mehadia:309, Pitesti:317, Craiova:366, **Bucharest:450**

Uniform Cost Search



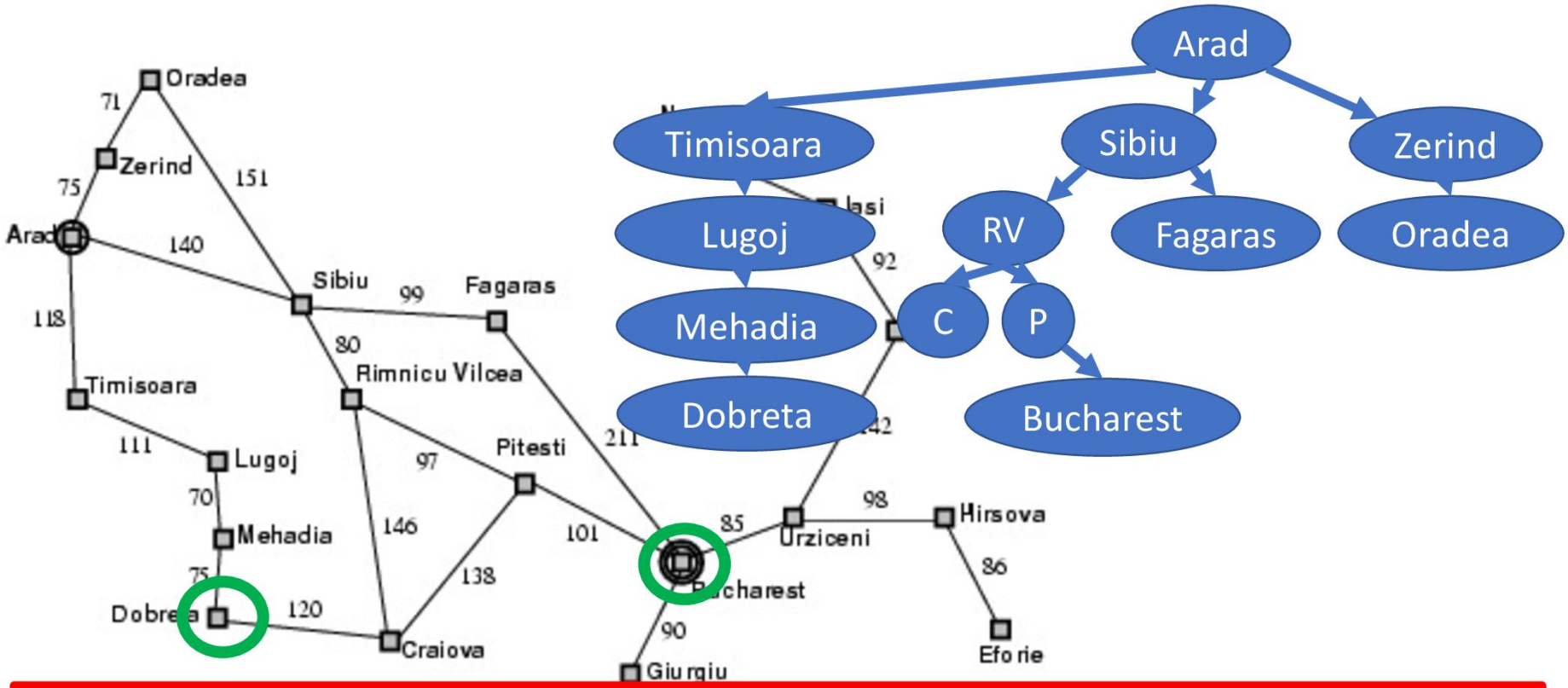
Pitesti:317, Craiova:366, **Dobreta:384**, Bucharest:450

Uniform Cost Search



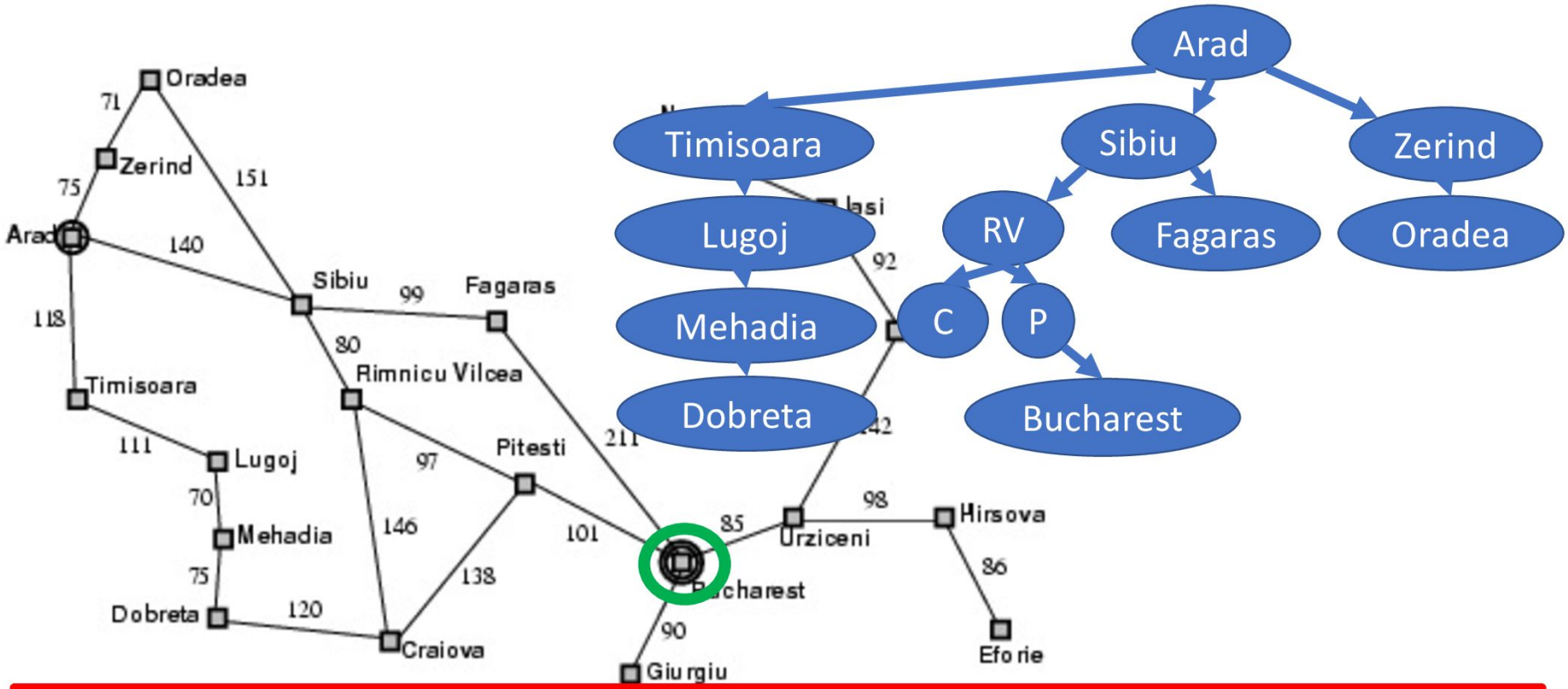
Craiova:366, Dobreta:384, **Bucharest:418**

Uniform Cost Search



Dobreta:384, Bucharest:418

Uniform Cost Search



Bucharest:418

Uniform-Cost Search (UCS)

- UCS is similar to Dijkstra's algorithm. UCS takes as input a search problem, which implicitly defines a large and even infinite graph.

- Complete
- Optimal
  - requires that the goal test is done when a node is ***removed*** from the *Frontier* rather than when node is generated by its parent node

Uniform-Cost Search (UCS)

- Time and space complexity: $O(b^d)$ (i.e., exponential)
  - d is the depth of the solution
  - b is the branching factor at each non-leaf node

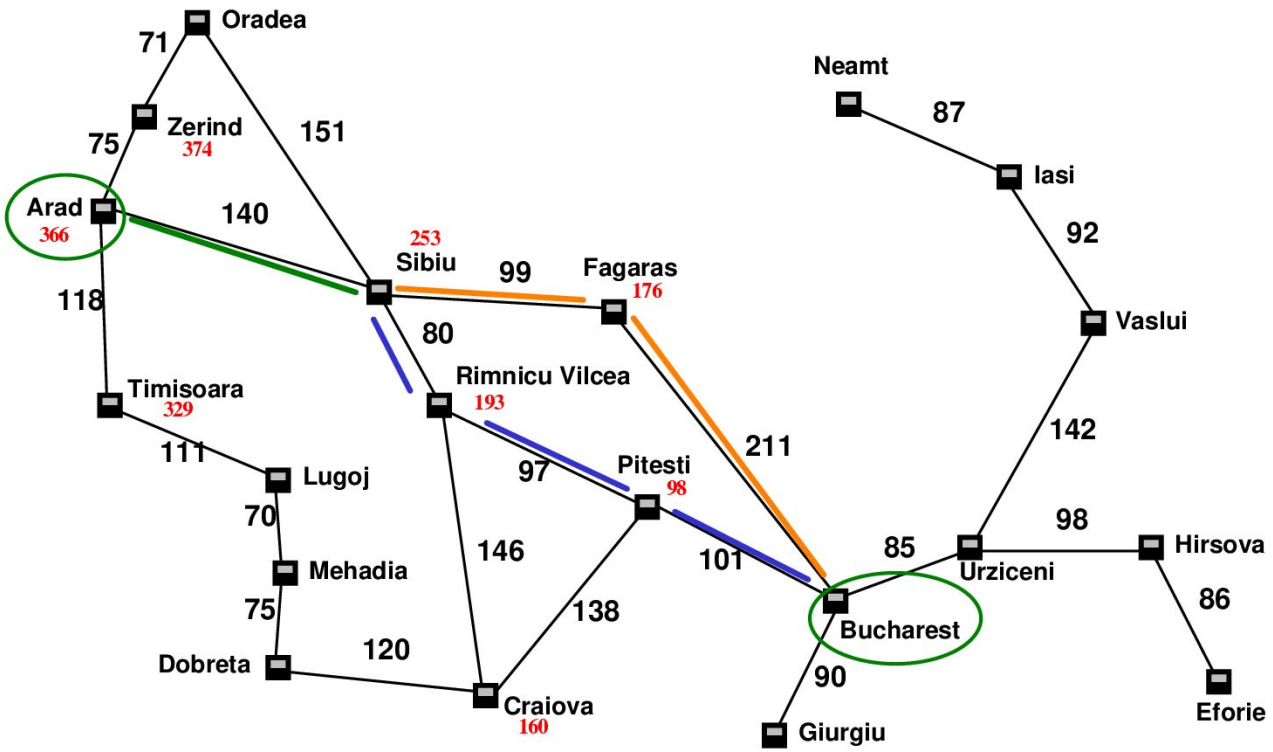
- More precisely, time and space complexity is $O(b^{C^*/\varepsilon})$ where all edge costs are ε, $\varepsilon > 0$, and C^* is the best goal path cost

Informed Search Algorithms

Informed Search Algorithms

- Blind search strategies rely only on exact information (initial state, operators, goal predicate)
- They don’t make use of additional information about the nature of the problem that might make the search more efficient
- If we have a (vague) idea in which direction to look for the solution, why not use this information to improve the search?

Romania with step costs in km



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	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Informed Search Algorithms

- **Heuristics** – rules grounded in experience, which direct the search towards the goal so that it becomes more efficient.
- **Heuristic** function **h: S**$\rightarrow$ **R**$^{+}$ assigns to each state **s** $\in$ **S** an estimate of the distance between that state and the goal state
- The smaller the value h(s), the closer is s to the goal state. If s is the goal state, then h(s) = 0
- Search strategies that use heuristics to narrow down the search are called heuristic (informed, directed) search strategies

Heuristic search

We'll take a look at:

1. Greedy best-first search

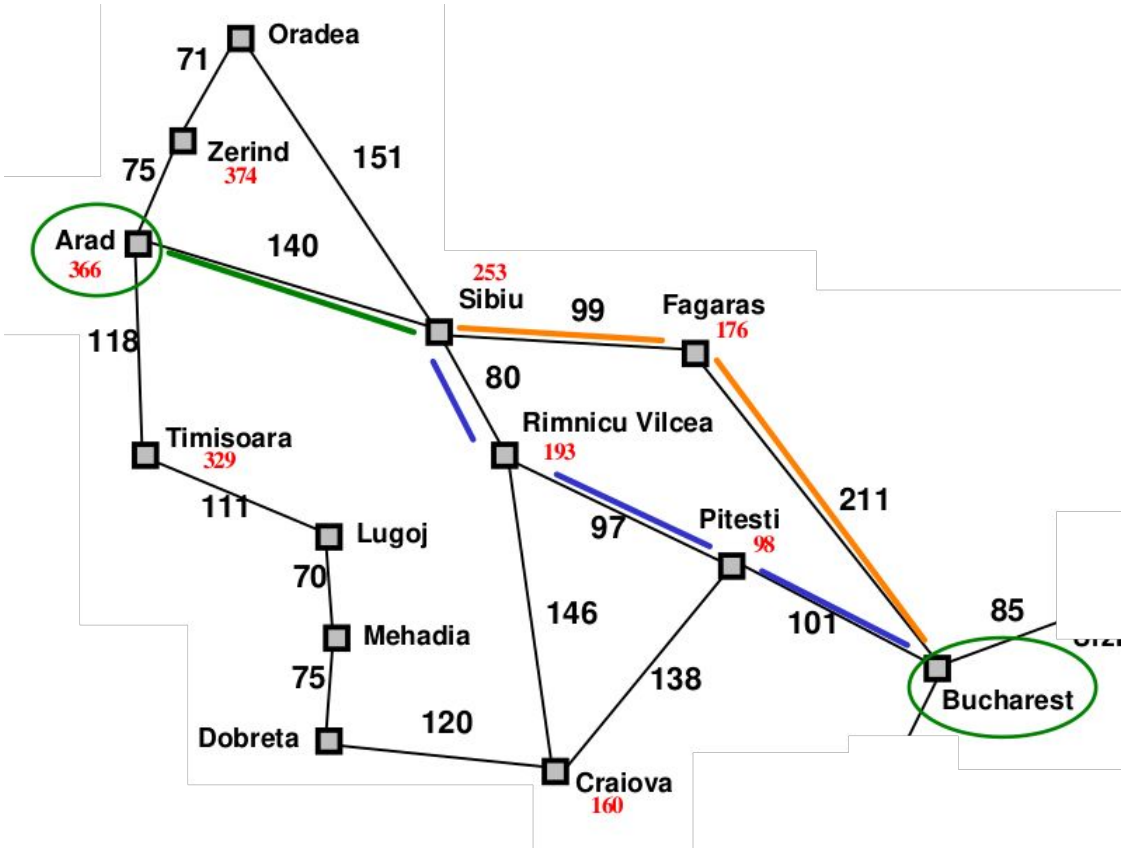
2. A* algorithm

3. Hill-climbing search

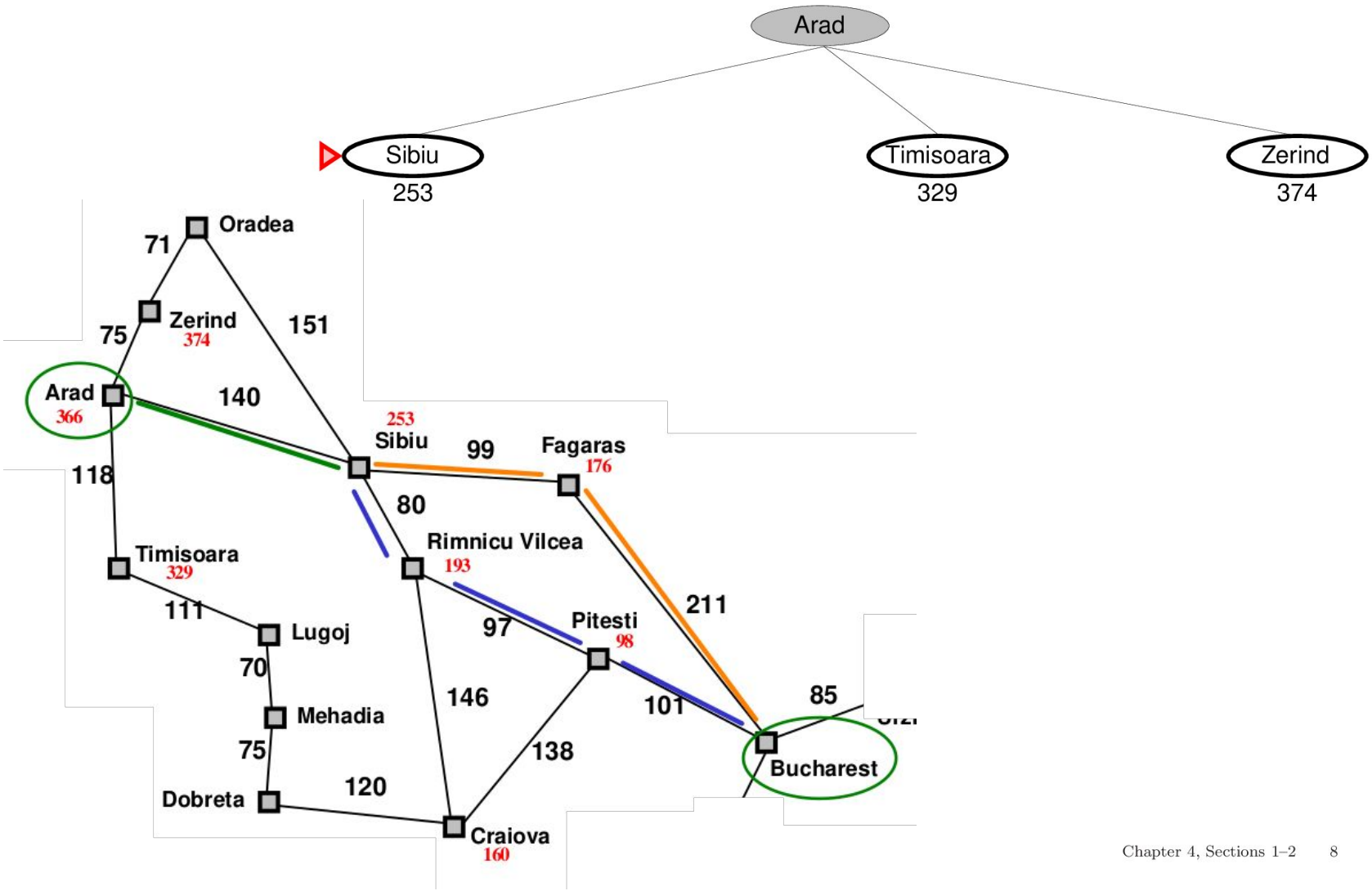
Greedy best-first search

- Always expands the node with the best (lowest) heuristic value (closest to goal), disregarding the total path cost accumulated so far
- The chosen path may not be optimal, but the algorithm doesn’t backtrack to correct this. Hence, a greedy algorithm is not optimal.

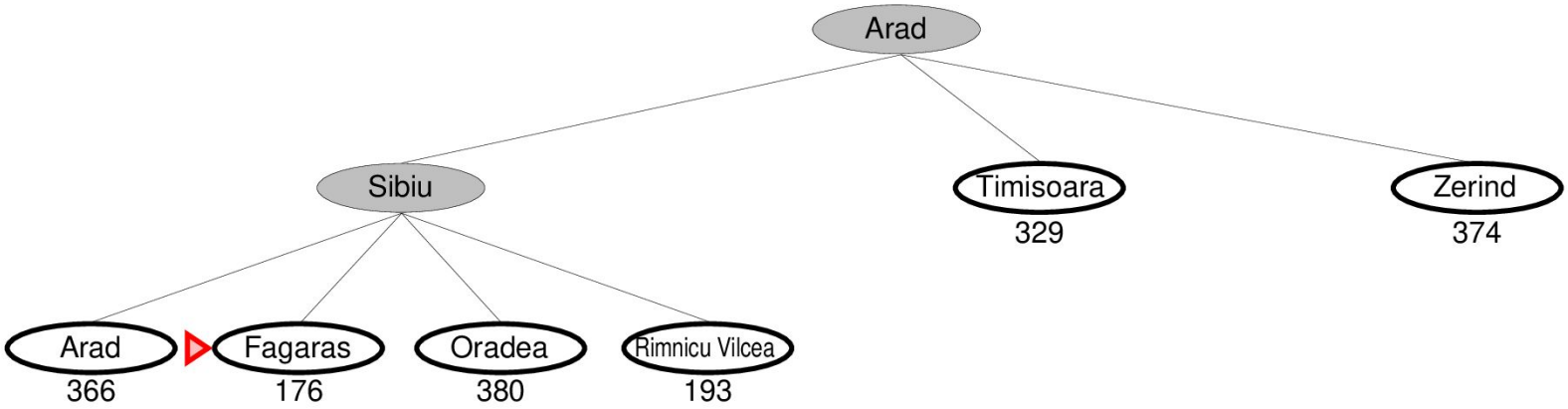
Greedy search example

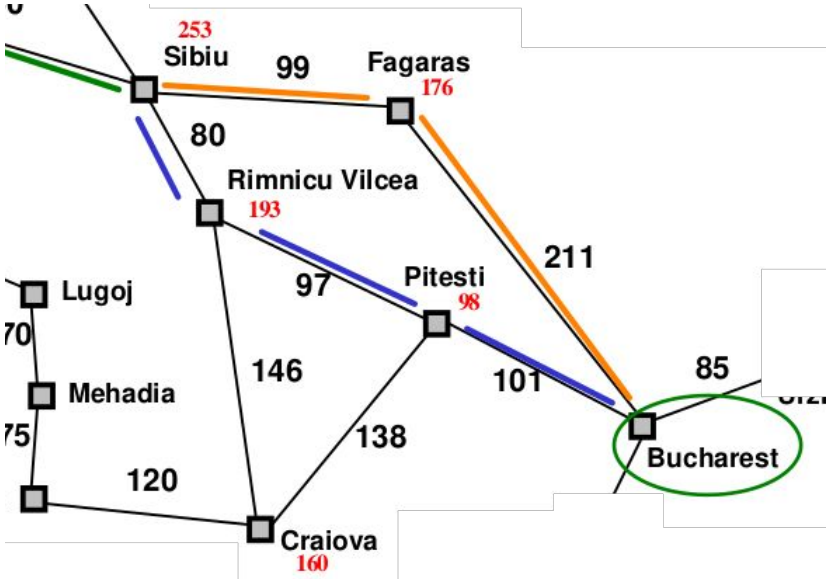


Greedy search example

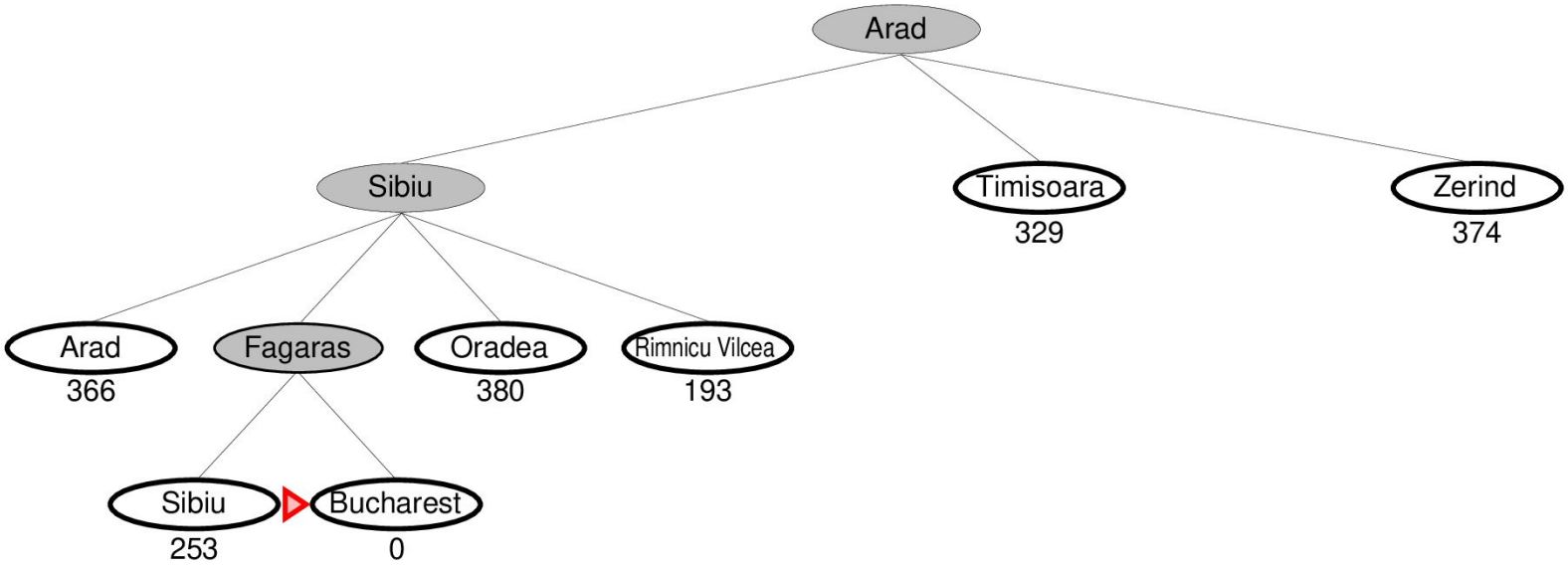


Greedy search example





Greedy search example



Properties of greedy search

Complete??

Time??

Space??

Optimal??

Properties of greedy search

Complete?? No–can get stuck in loops, e.g.,

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$—keeps all nodes in memory

Optimal?? No

A* search

Idea: avoid expanding paths that are already expensive

Evaluation function $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n
$h(n)$ = estimated cost to goal from n
$f(n)$ = estimated total cost of path through n to goal

A* search uses an admissible heuristic
i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the **true** cost from n.
(Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

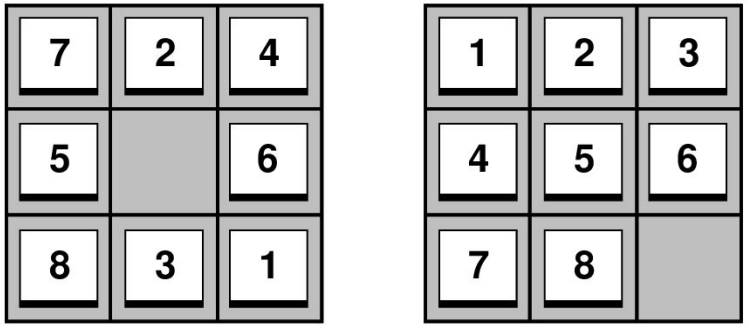
E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance

Theorem: A* search is optimal

Admissible heuristics

E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



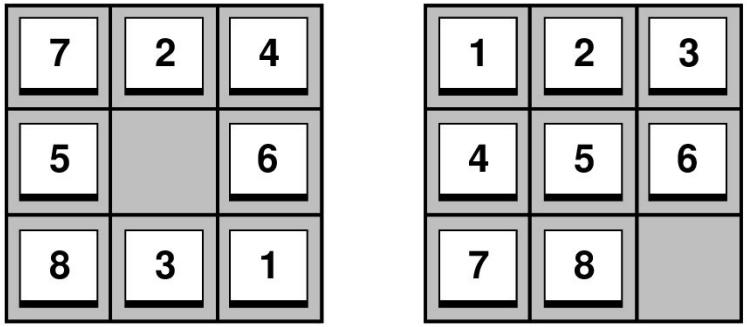
Start State Goal State

$h_1(S) =$??
$h_2(S) =$??

Admissible heuristics

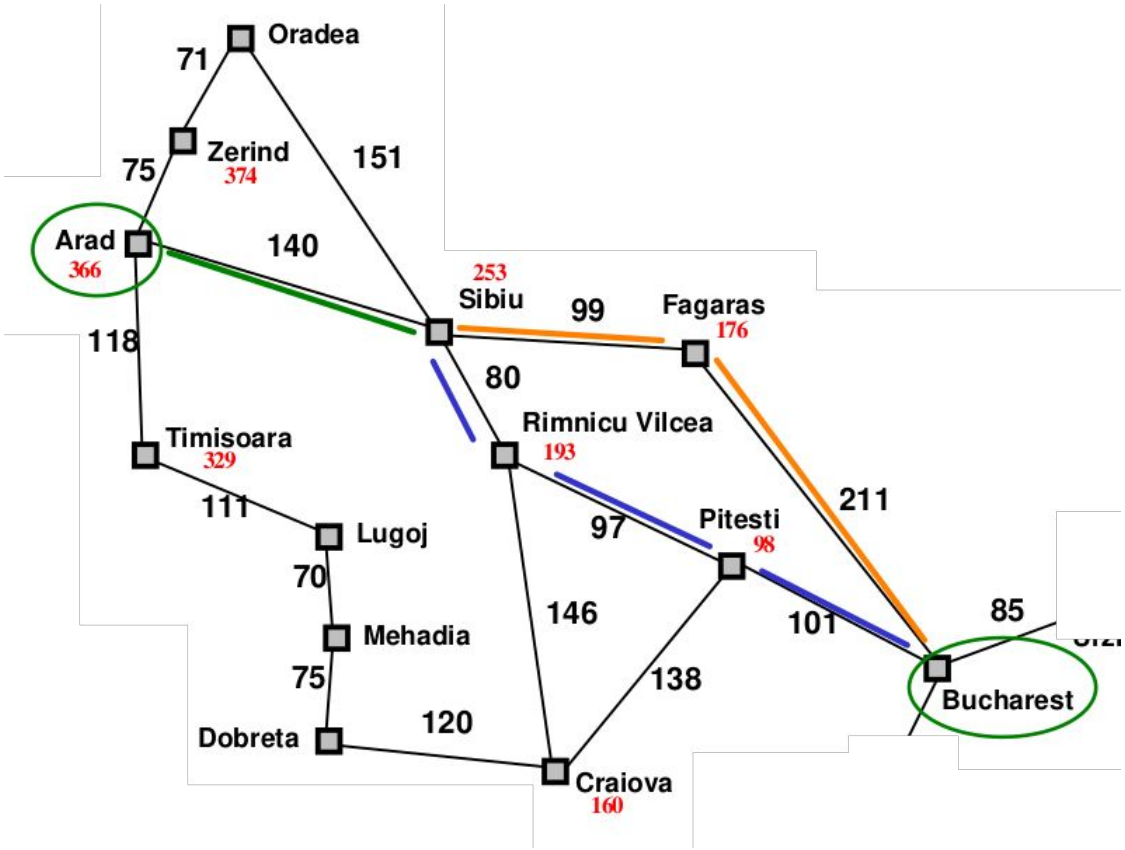
E.g., for the 8-puzzle:

$h_1(n)$ = number of misplaced tiles
$h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)

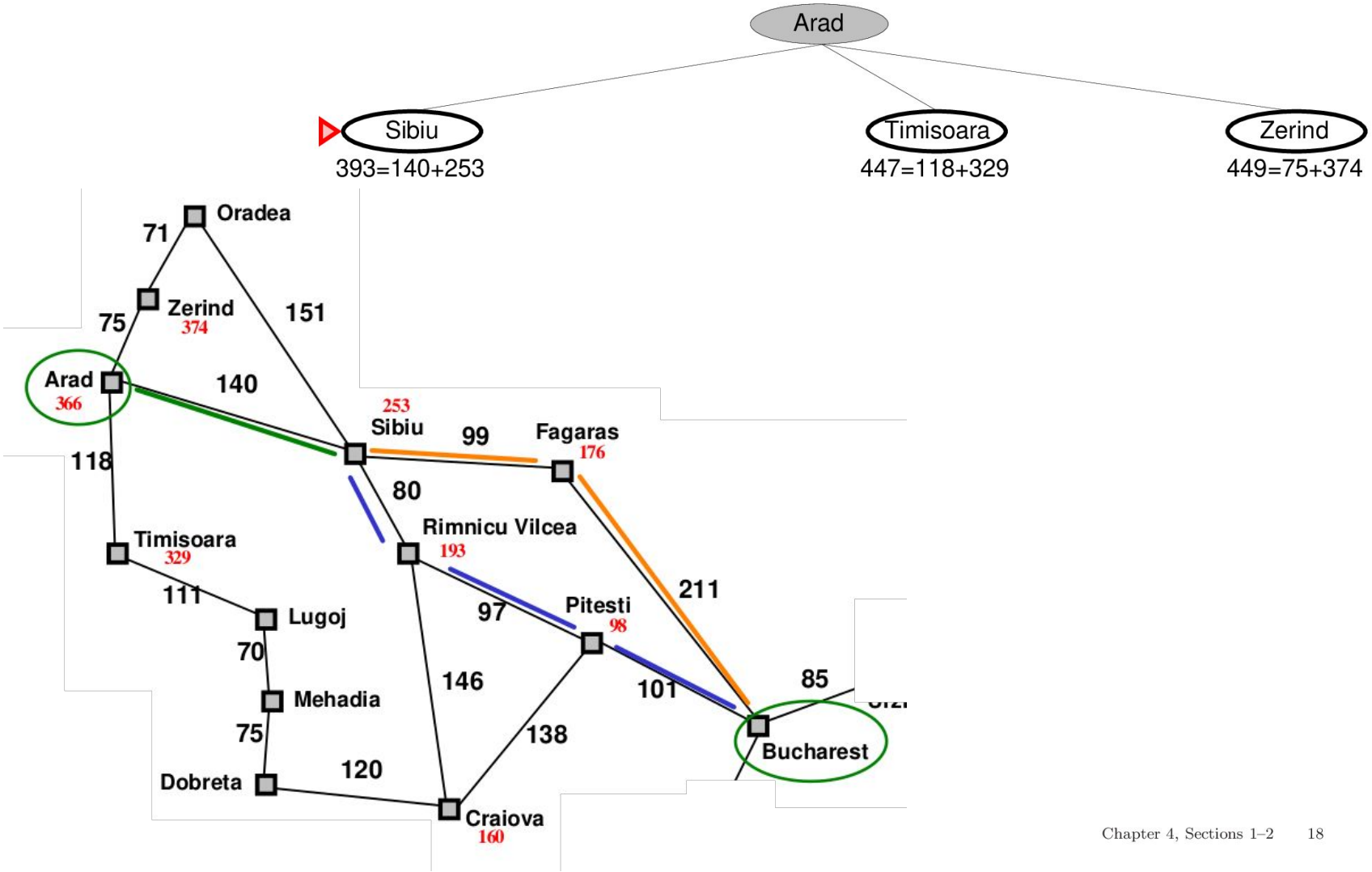


$h_1(S) =$?? 6
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

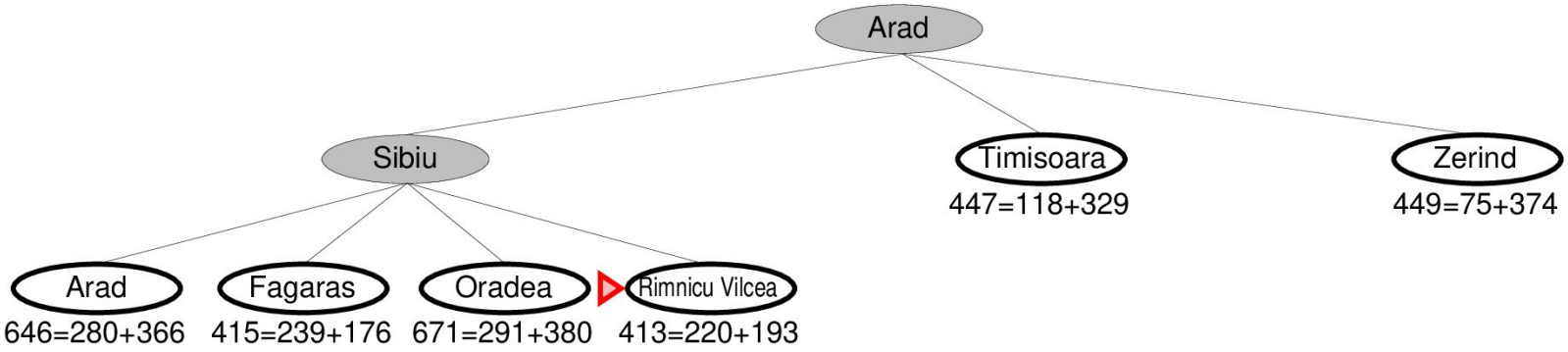
A* search example

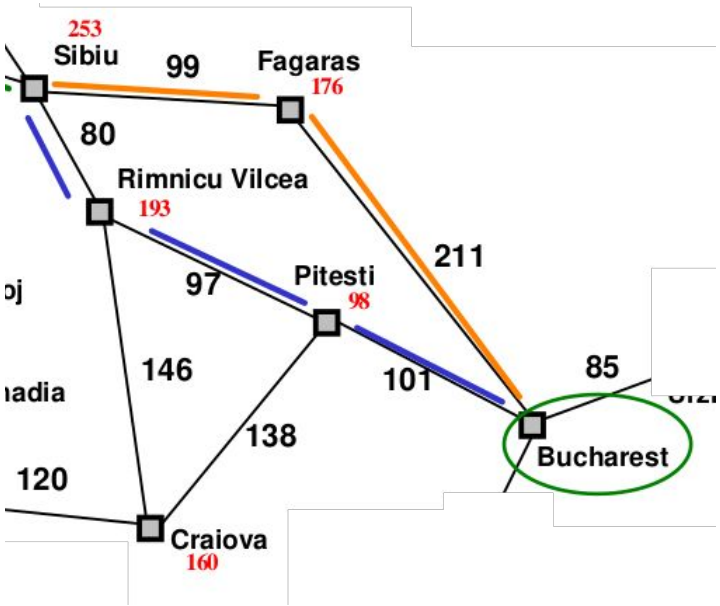


A* search example

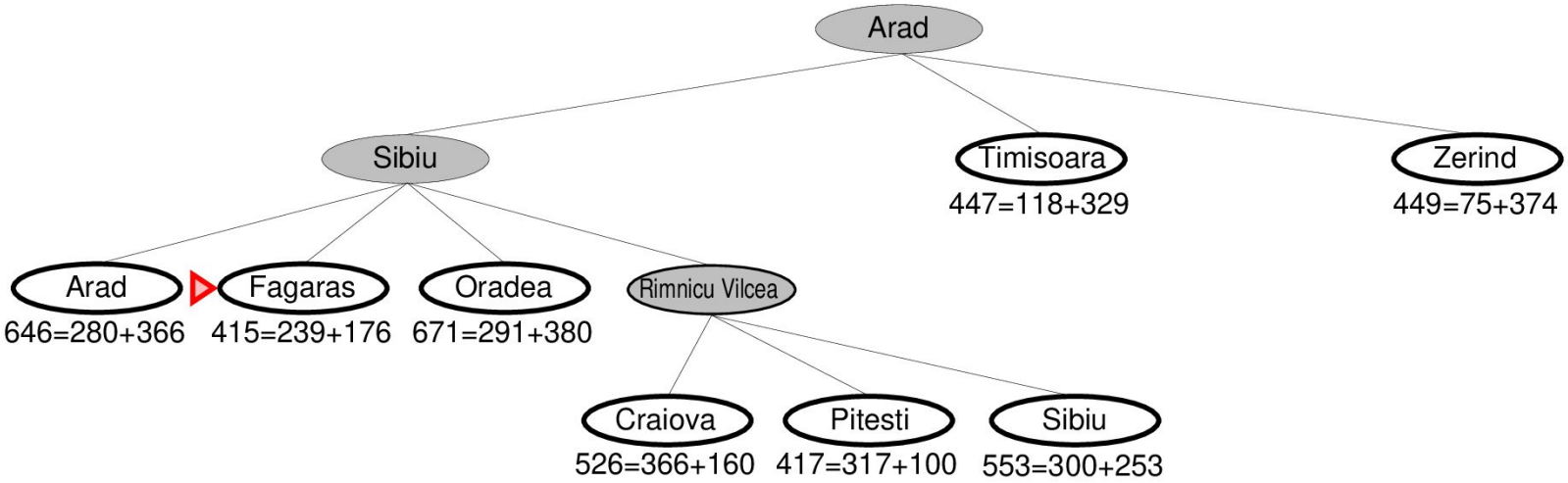


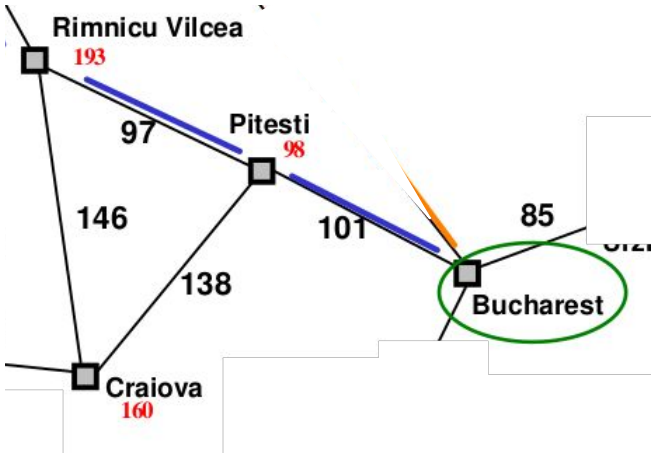
A* search example



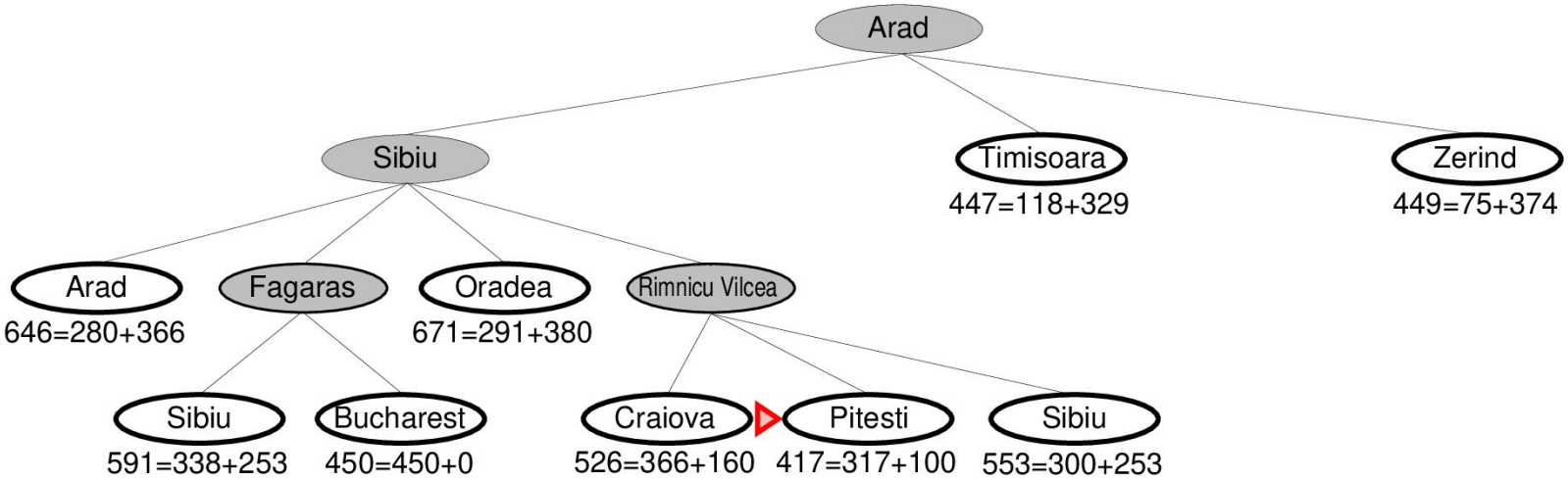


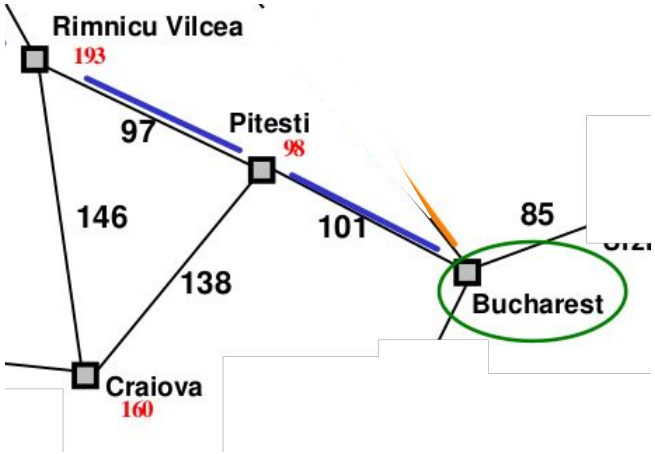
A* search example



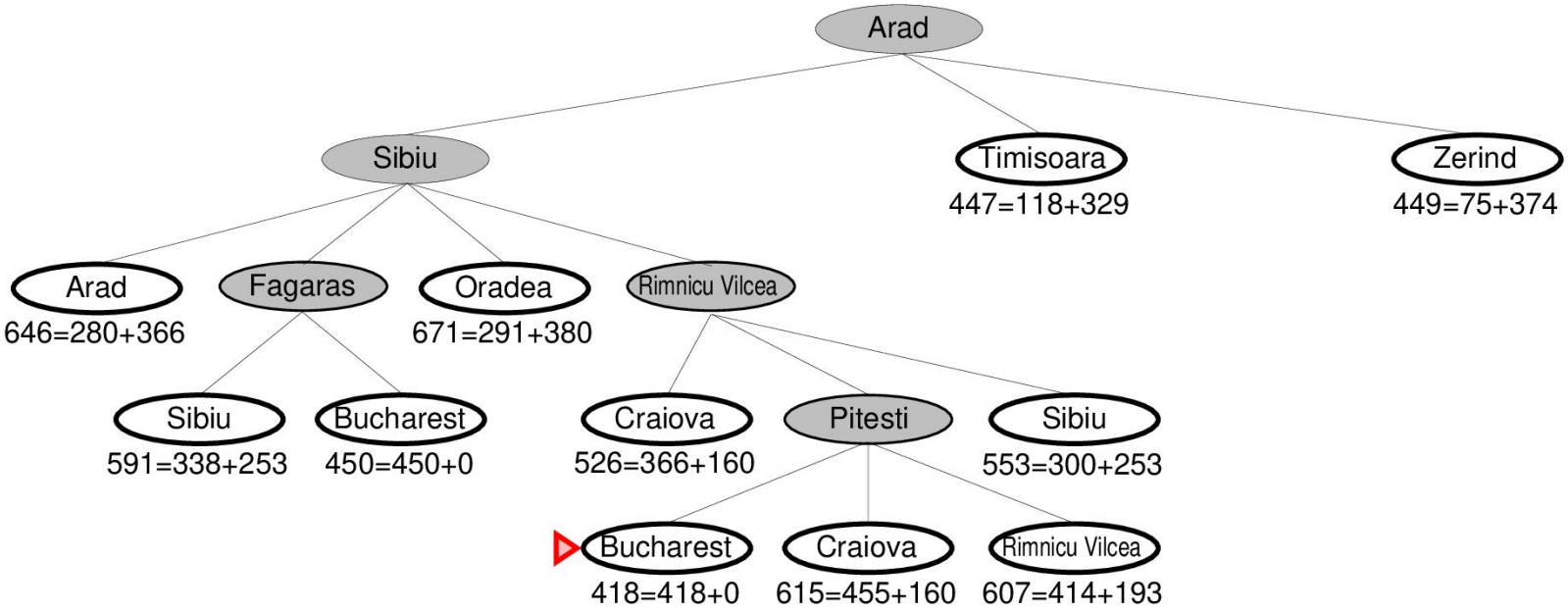


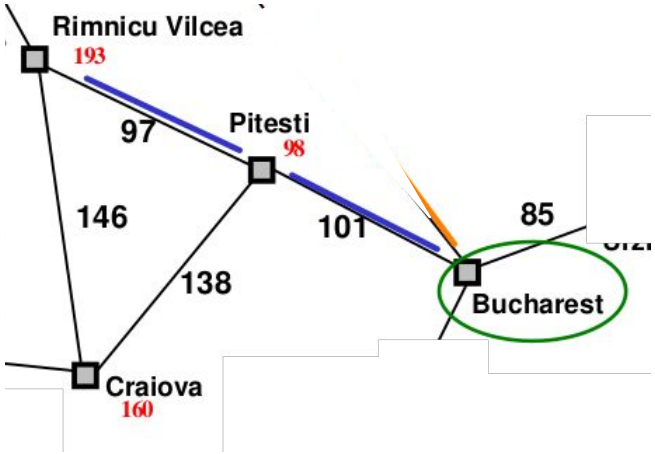
A* search example



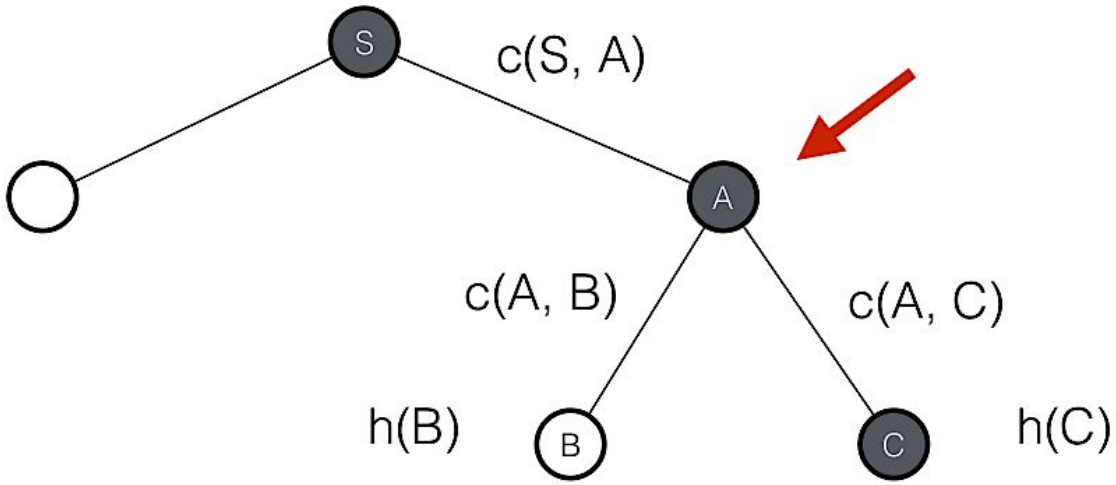


A* search example





Optimality of A* (standard proof)



B: c(S, A) + C(A, B) + h(B)

C: c(S, A) + C(A, C) + h(C)

g(n) + **h(n)**

g(n) is always the exact cost of the **only** path to **n**

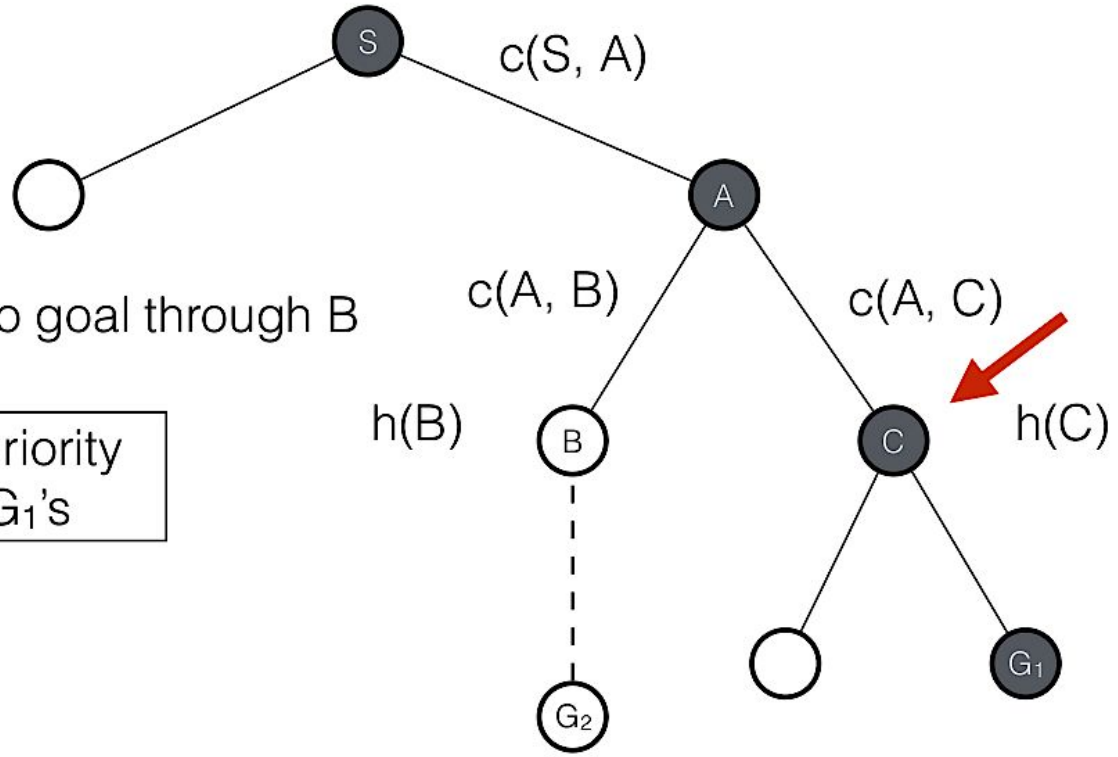
h(n) is an underestimate of cost to goal

Optimality of A* (standard proof)

G_1: true cost to G_1

B: underestimate of true cost to goal through B

if G_2 were cheaper, B's priority would be cheaper than G_1's



Optimal in trees if **admissible** h(n) <= true cost to goal

Proof of lemma: Consistency

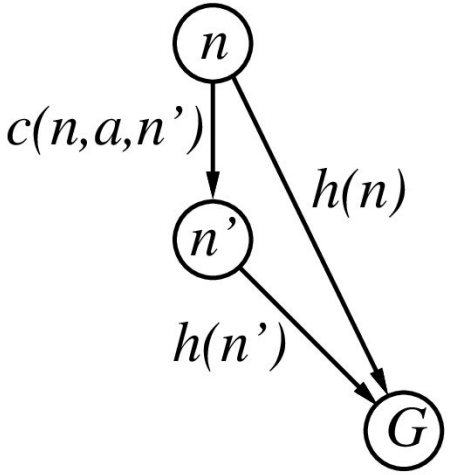
A heuristic is consistent if

$$h(n) \leq c(n, a, n') + h(n')$$

If h is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

I.e., $f(n)$ is nondecreasing along any path.

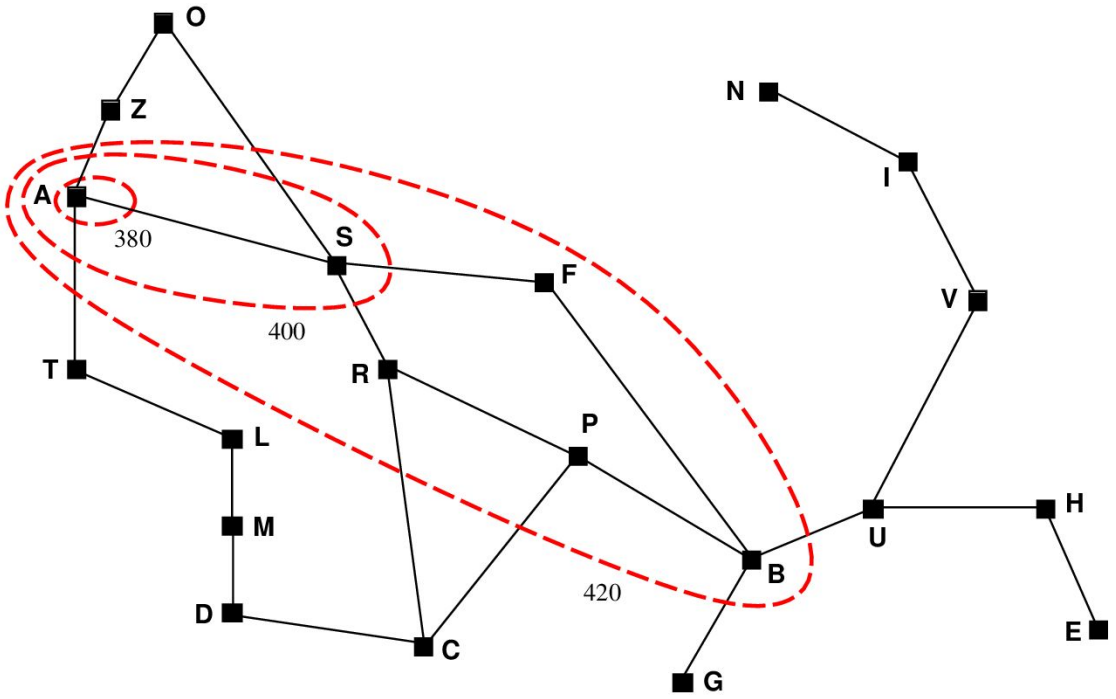


- A* is optimal in graphs if consistent $h(n) <= c(n, a, n') + h(n')$
- Consistent ⇒ admissible (admissible heuristics are also consistent)

Optimality of A* (more useful)

Lemma: A* expands nodes in order of increasing f value*

Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)
Contour i has all nodes with $f = f_i$, where $f_i < f_{i+1}$



Properties of A*

Complete??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

$$O(b^{\epsilon d}) \text{ where } \begin{array}{ll} \epsilon = & |h(n_0) - h^*(n_0)| \\ n_0 = & \text{start state} \\ h^* = & \text{actual cost to goal state} \end{array}$$

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? O(b^m), as in Greedy Best-First — may end up with all nodes in memory

Optimal??

Properties of A*

Complete?? Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time?? Exponential in [relative error in h $\times$ length of soln.]

Space?? O(b^m), as in Greedy Best-First — may end up with all nodes in memory

Optimal?? Yes—cannot expand f_{i+1} until f_i is finished

If h is admissible

Local Search and Optimization Problems

Local Search and Optimization Problems

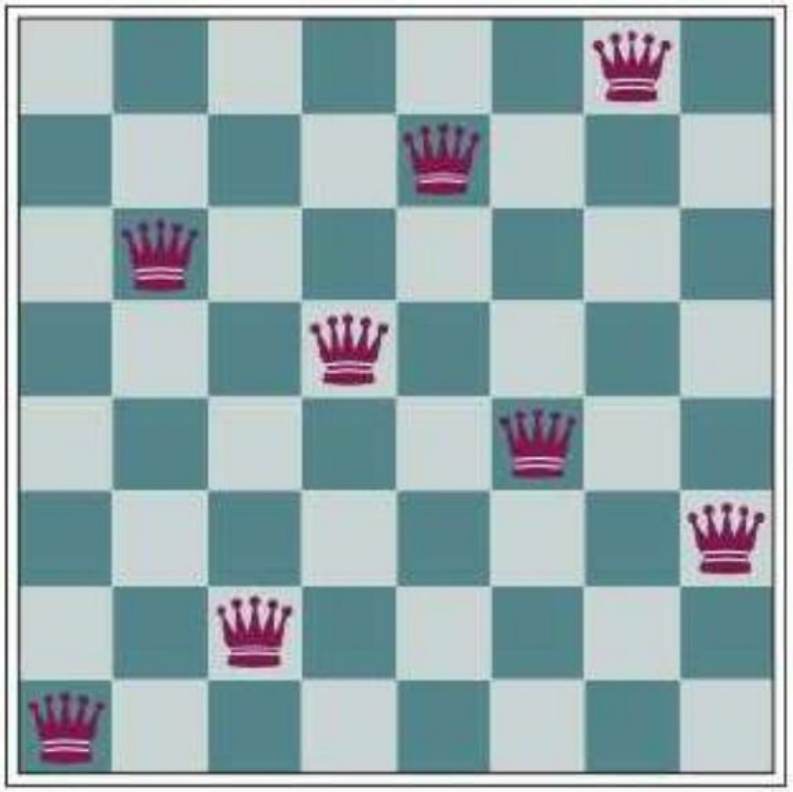
- In the search problems, we discussed how to find paths through the search space, e.g., a path from Arad to Bucharest.
- But sometimes we care only about the final state and not the path. e.g., in the 8-queens problem for valid final configuration of 8 queens.
- **Local Search**: Local search algorithms operate by searching from a start state to neighboring states,
  - without keeping track of the paths
  - nor the set of states that have been reached.

Local Search

- Local Search are not systematic—they might never explore a portion of the search space where a solution actually resides.
- Advantages:
  - (1) they use very little memory.
  - (2) they can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.
- Local search algorithms can also solve optimization problems, in which the aim is to find the best state according to an objective function.

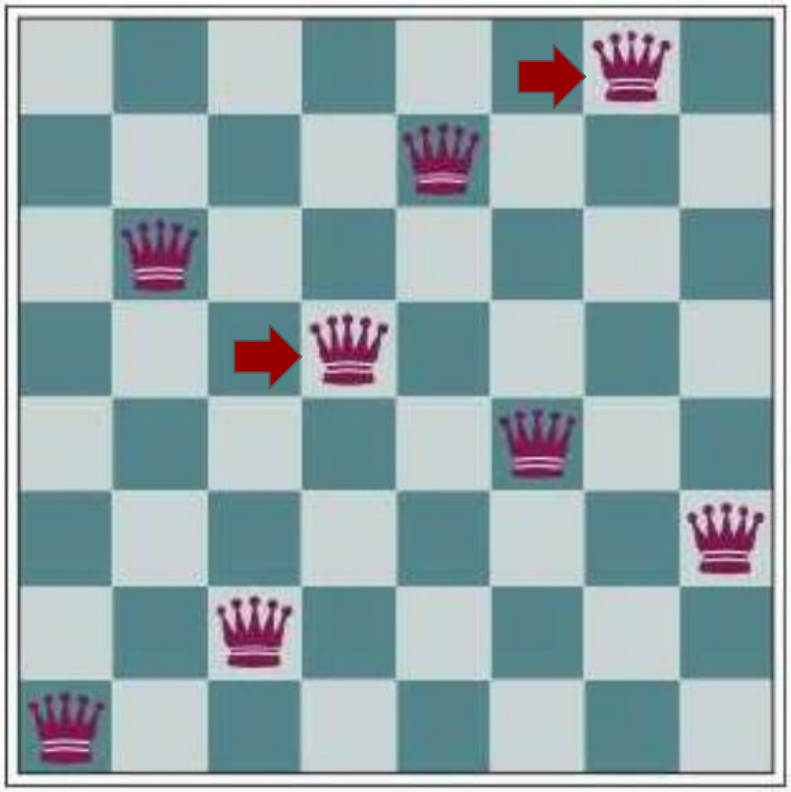
Hill climbing Search (greedy local search)

Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead.



Hill climbing Search (greedy local search)

Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead.



Hill climbing Search (greedy local search)

Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead.

An 8-queens state with heuristic cost estimate h = 17.

The board shows the value of h for each possible successor obtained by moving a queen within its column.

There are 8 moves that are tied for best, with h = 12. The hill-climbing algorithm will pick one of these

Note that a state has 8 × 7 = 56 successors

An 8-queens state with heuristic cost estimate

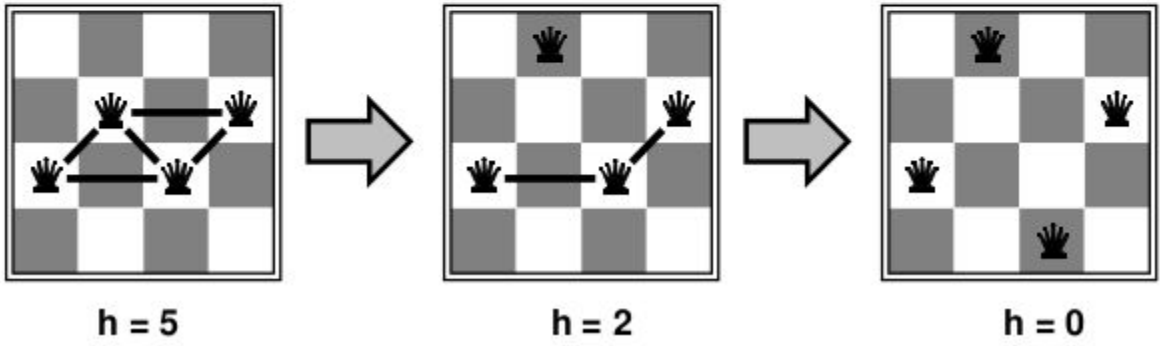
heuristic cost example

States: 4 queens in 4 columns ($4^4 = 256$ states)

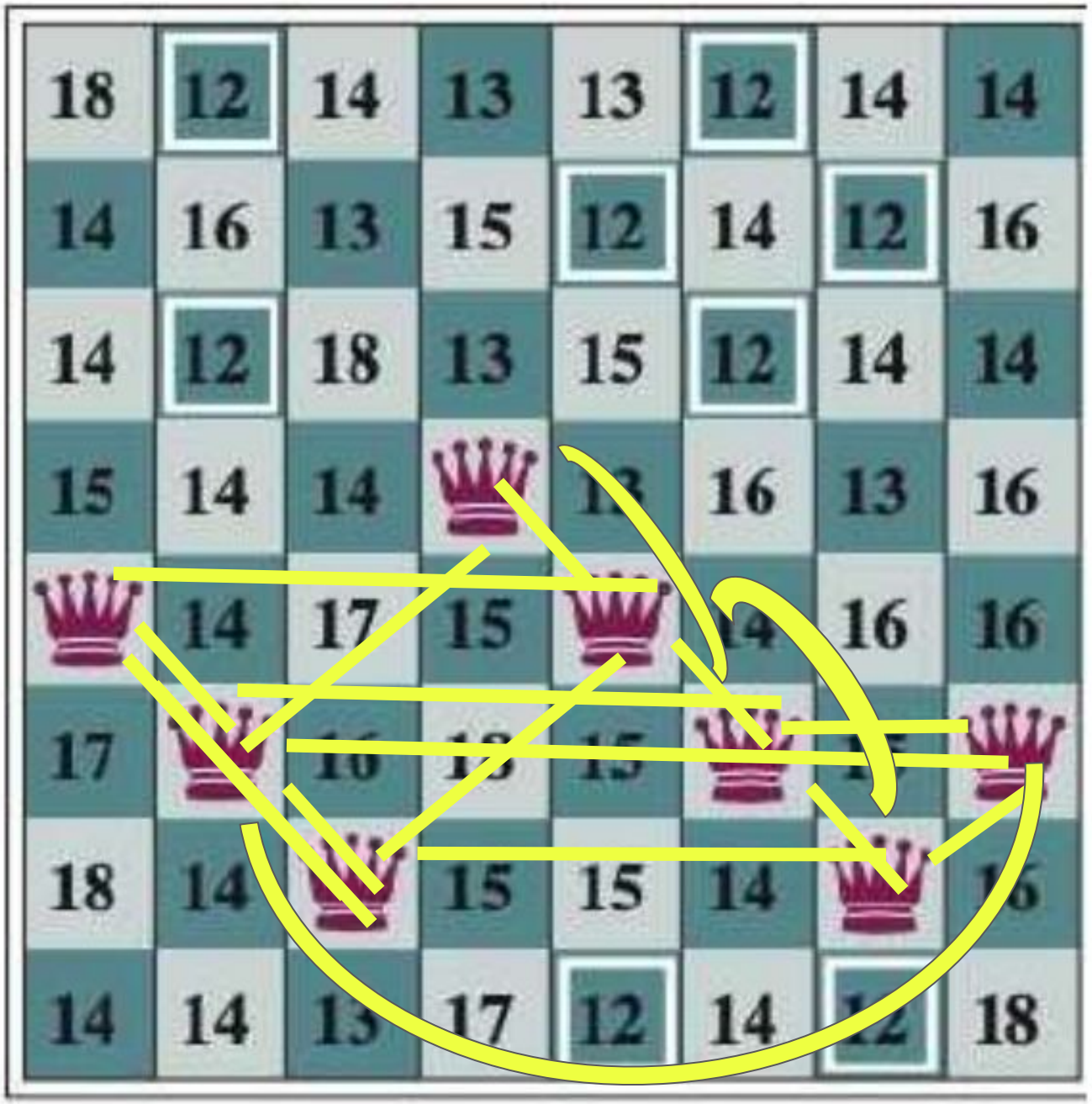
Operators: move queen in column

Goal test: no attacks

Evaluation: $h(n)$ = number of attacks

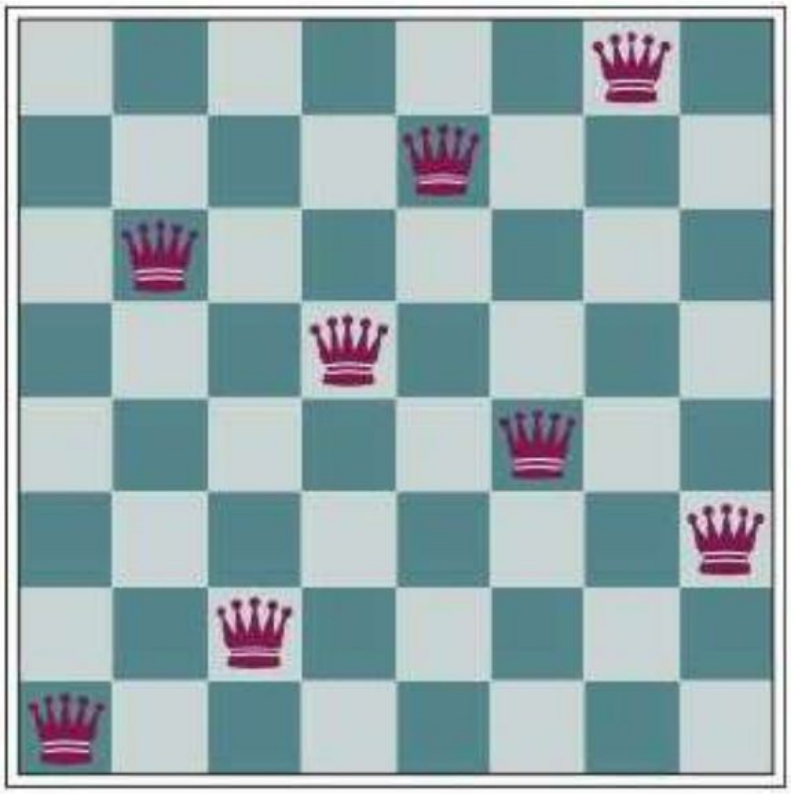


state with heuristic cost estimate h = 17.


18 12 14 13 13 12 14 14
14 16 13 15 12 14 12 16
14 12 18 13 15 12 14 14
15 14 14 16 13 16
14 17 15 16 16
17 15
18 14 15 15 14
14 14 13 17 12 14 18

Hill climbing Search (greedy local search)

Hill climbing is sometimes called greedy local search because it grabs a good neighbor state without thinking ahead.



Just five steps

Hill-Climbing Search

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
```

Local search

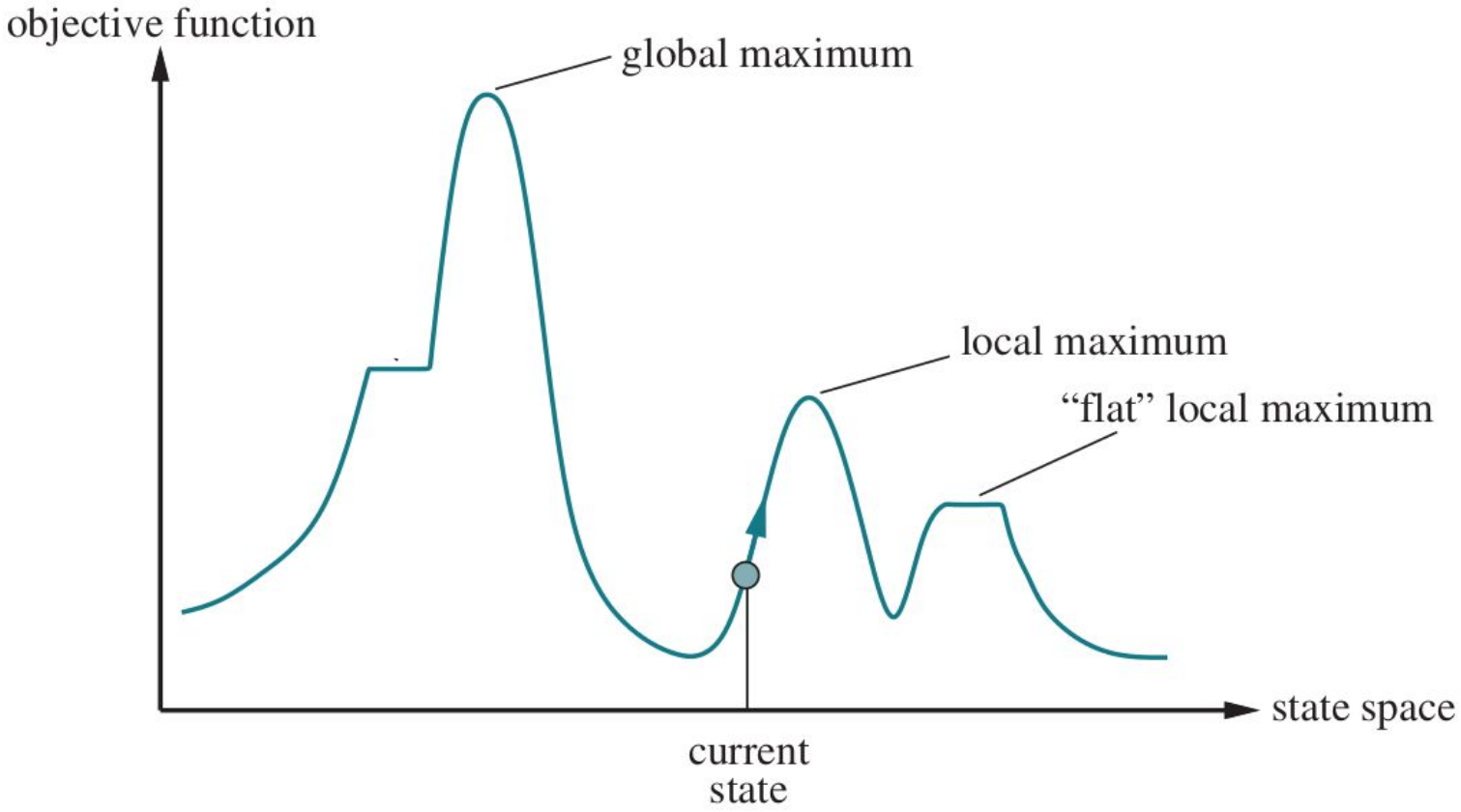


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum.