Heuristic Searches
Fundamentals of AI (AE1FAI)

Slides created by Dr Rong Qu & Prof. Ho Sooi Hock
Instructor: Dr Qian Zhang

Session 4 (2023)

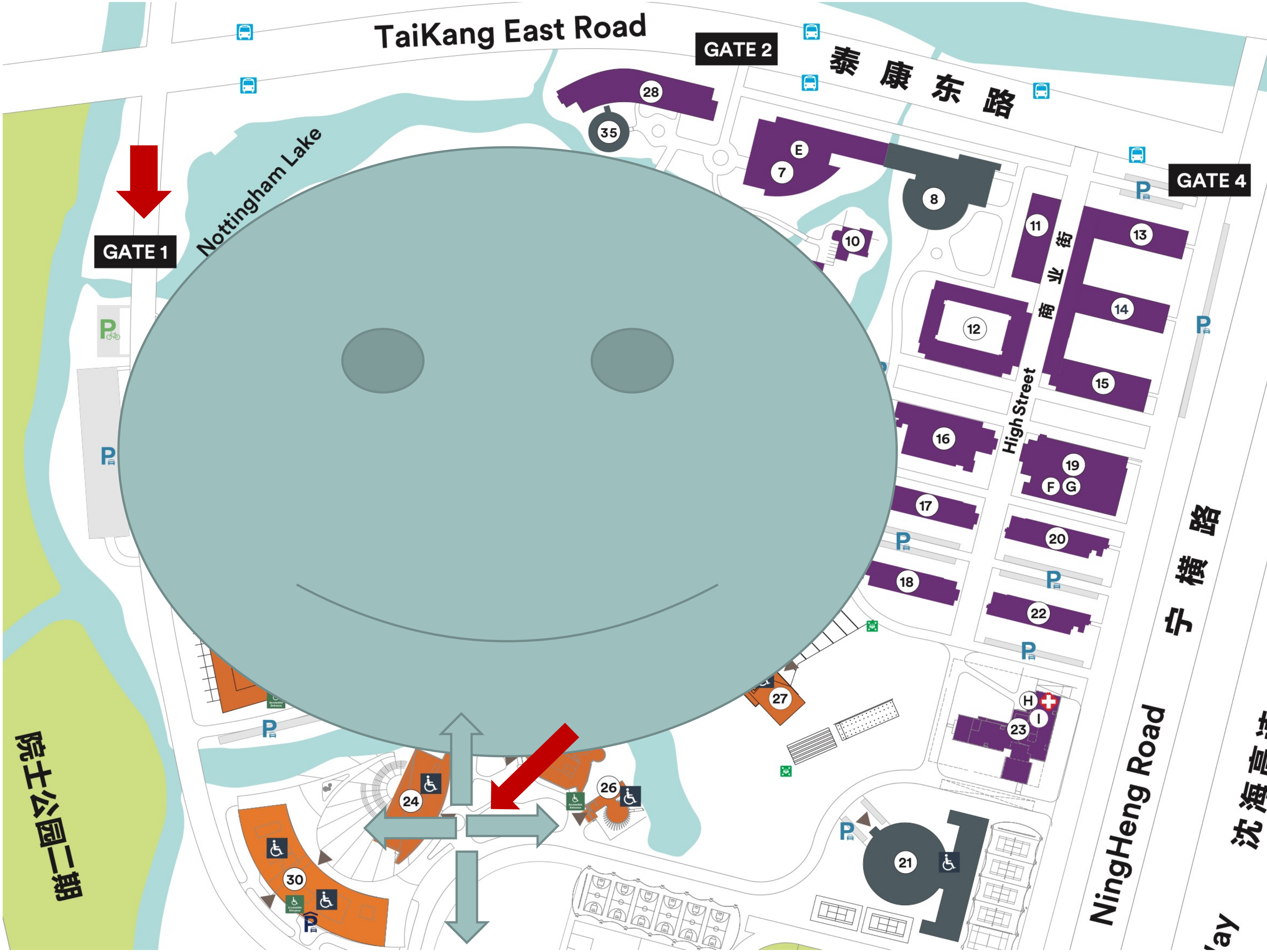
Blind Search vs. Heuristic Searches

Blind search

- **Blindly** choose where to search in the search tree
- When problems get large, not practical any more

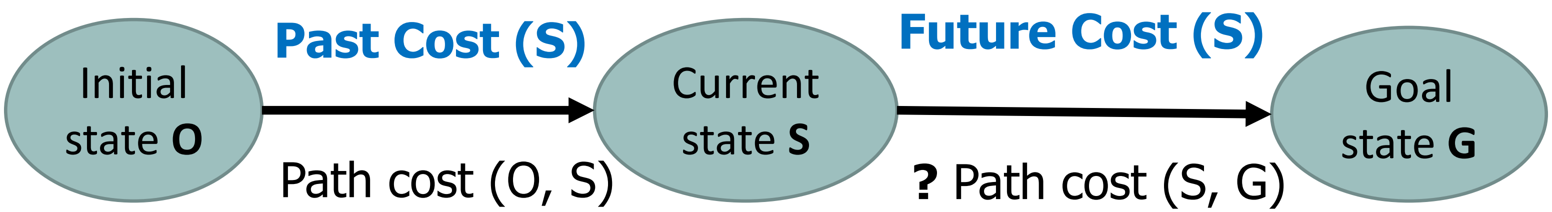
Heuristic search

- Explore the node: more **likely** to lead to the goal state
- Using knowledge, so called informed search
- Heuristics: educated guesses, intuitive judgments or simply common sense.
- Greedy search, A* search



TaiKang East Road
泰康东路
GATE 1
GATE 2
GATE 4
Nottingham Lake
商业街
High Street
NingHeng Road
宁横路
沈海高速
院士公园二期
28
35
E
7
8
10
11
12
13
14
15
16
17
18
19
F
G
20
22
H
I
23
21
24
26
27
30

Heuristic Search



Path cost = Past Cost (S) + Future Cost(S)

Path cost = Past Cost (S) + h (S) Heuristic Function

Estimation of future cost

HEURISTICS

- Heuristic function $h(n)$ estimates the "**goodness**" of a node n
  - $h(n)$ = estimation of minimal cost path (or distance) from n to a goal state

- All domain knowledge used in the search is encoded in $h(n)$, which is computable from the current state

- In general, $h(n) \geq 0$ for all nodes n *and* $h(n) = 0$ implies that n is a goal node

Heuristic Search

- Add domain-specific information to select the best path along which to continue searching
- Sometimes known as *informed search,* it is *u*sually more efficient than blind searches
- A heuristic method is particularly used to rapidly come to a solution that is hoped to be close to the best possible answer, or 'optimal solution' (wikipedia)

GREEDY SEARCH

Work by deciding which is the **next best node** to expand to reach the goal using **domain knowledge**

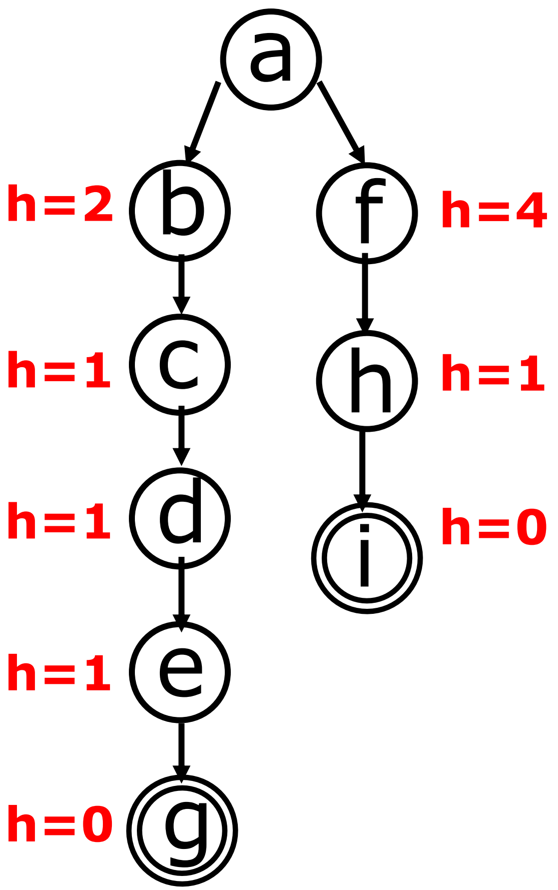
- **Function f(n)**: how close the node is to the **goal state**

There is no guarantee that the node explored is the best

	Order of nodes to explore	Cost(n)
UCS	Nodes of the lowest path cost (**past cost**)	**f(n)=g(n)**: actual path cost thus far
Greedy search	Nodes which are the closest to the goal (**future cost**)	**f(n)=h(n)**: estimate cost to the goal (using heuristic)

Heuristic Searches – greedy search

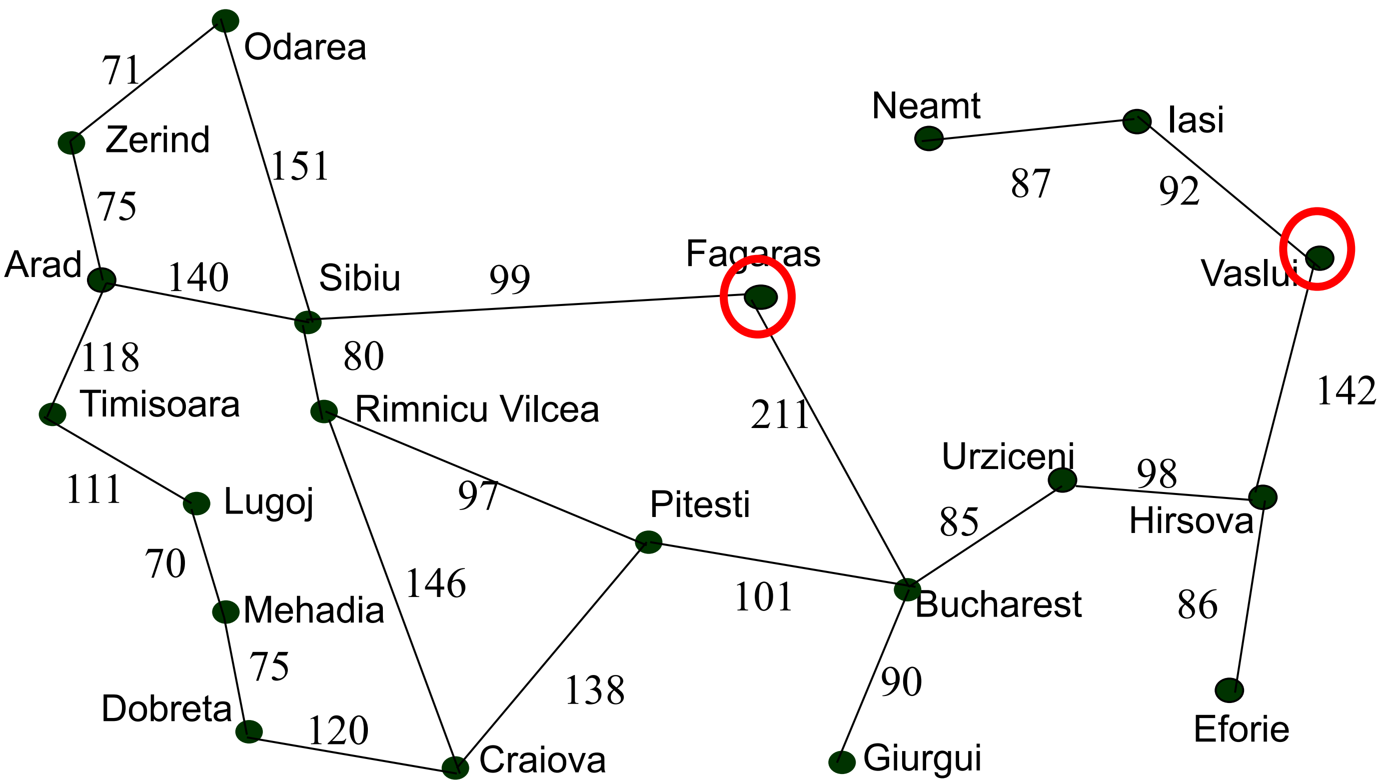
- Use as an evaluation function $f(n) = h(n)$, sorting nodes by increasing values of f
- Selects node to expand believed to be *closest* (hence “greedy”) to a goal node (i.e., select node with smallest f value)
- Not complete
- Not optimal, as in the example.
  - assuming all arc costs are 1, then greedy search will find goal g, which has a solution cost of 5
  - however, the optimal solution is the path to goal I with cost 3



Heuristic Searches – greedy search

Go to the city which is the nearest to the goal city

It is possible to get stuck in an infinite loop: not complete

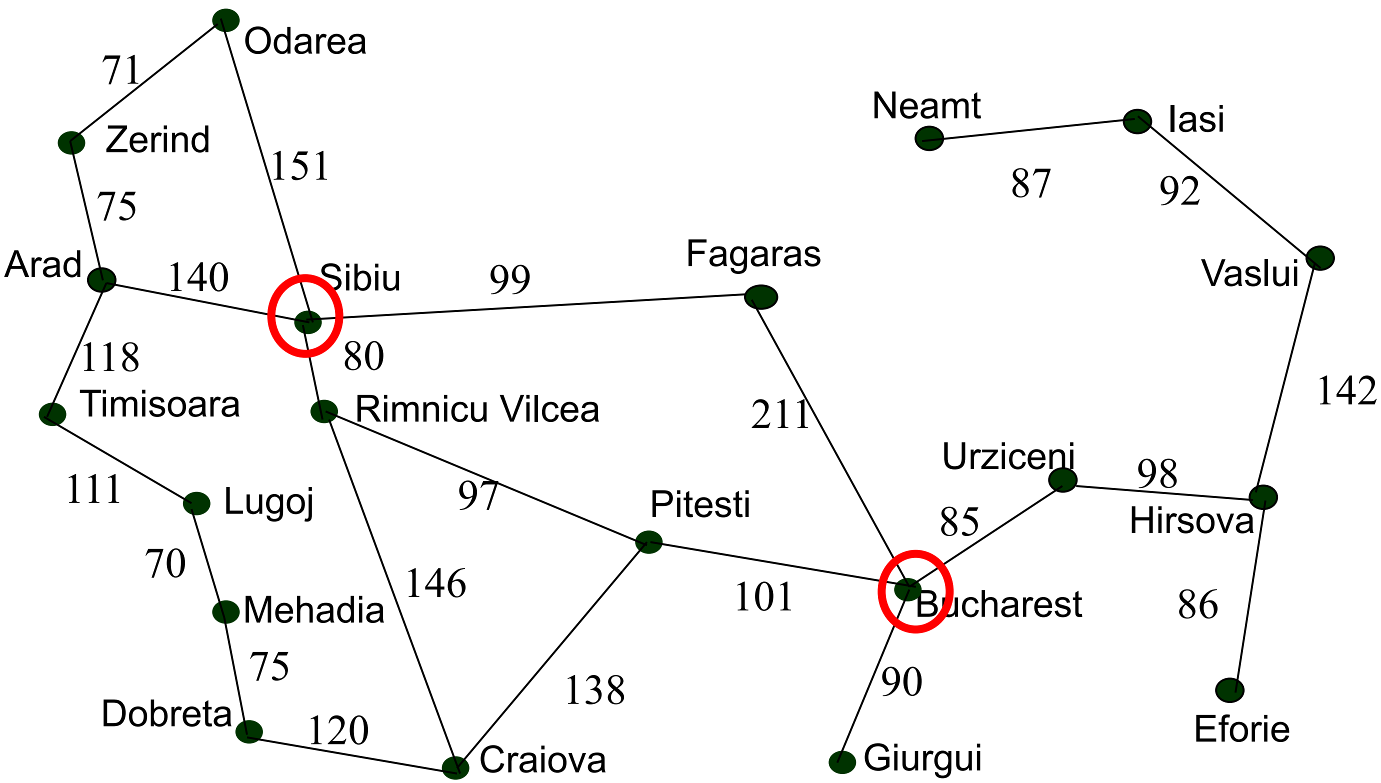


$h_{sld}(n)$ = straight line distance between n and the goal

Heuristic Searches – greedy search

Go to the city which is the nearest to the goal city

Is it optimal, using h to order nodes to be explored?

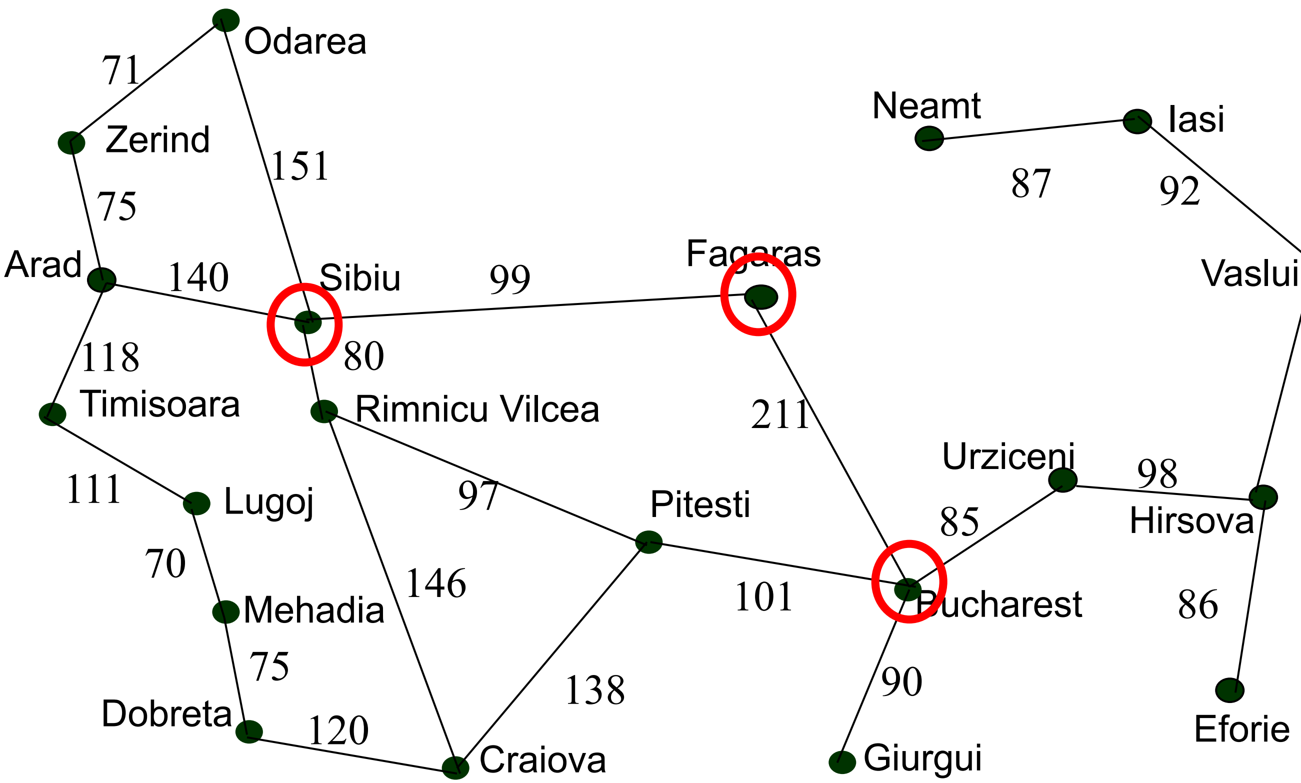


$h_{sld}(n)$ = straight line distance between n and the goal

GREEDY SEARCH

Function GREEDY-SEARCH(problem) returns a solution of failure

Return GENERAL-SEARCH(problem, h)



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

Performed well, but not optimal

Greedy Search

- It is possible to get stuck in an infinite loop (consider being in Iasi and trying to get to Fagaras) unless mechanism for avoiding repeated states is in place
- It is not optimal
- It is not complete
- Time and space complexity is $O(b^d)$;
  - where d is the depth of the search tree

UNIFORM COST SEARCH

- In the previous lecture, we saw BFS and DFS operating on trees where all branches have an equal cost

- Uniform Cost Search (UCS) is a special case of BFS which can be applied to trees where the costs of each branch vary

- UCS works by expanding the **lowest cost node** on the frontier

UCS

Nodes Expanded

1.Sibiu 2.Rimnicu 3.Faragas 4.Arad 5.Pitesti 6.Zerind 7.Craiova 8.Timisoara

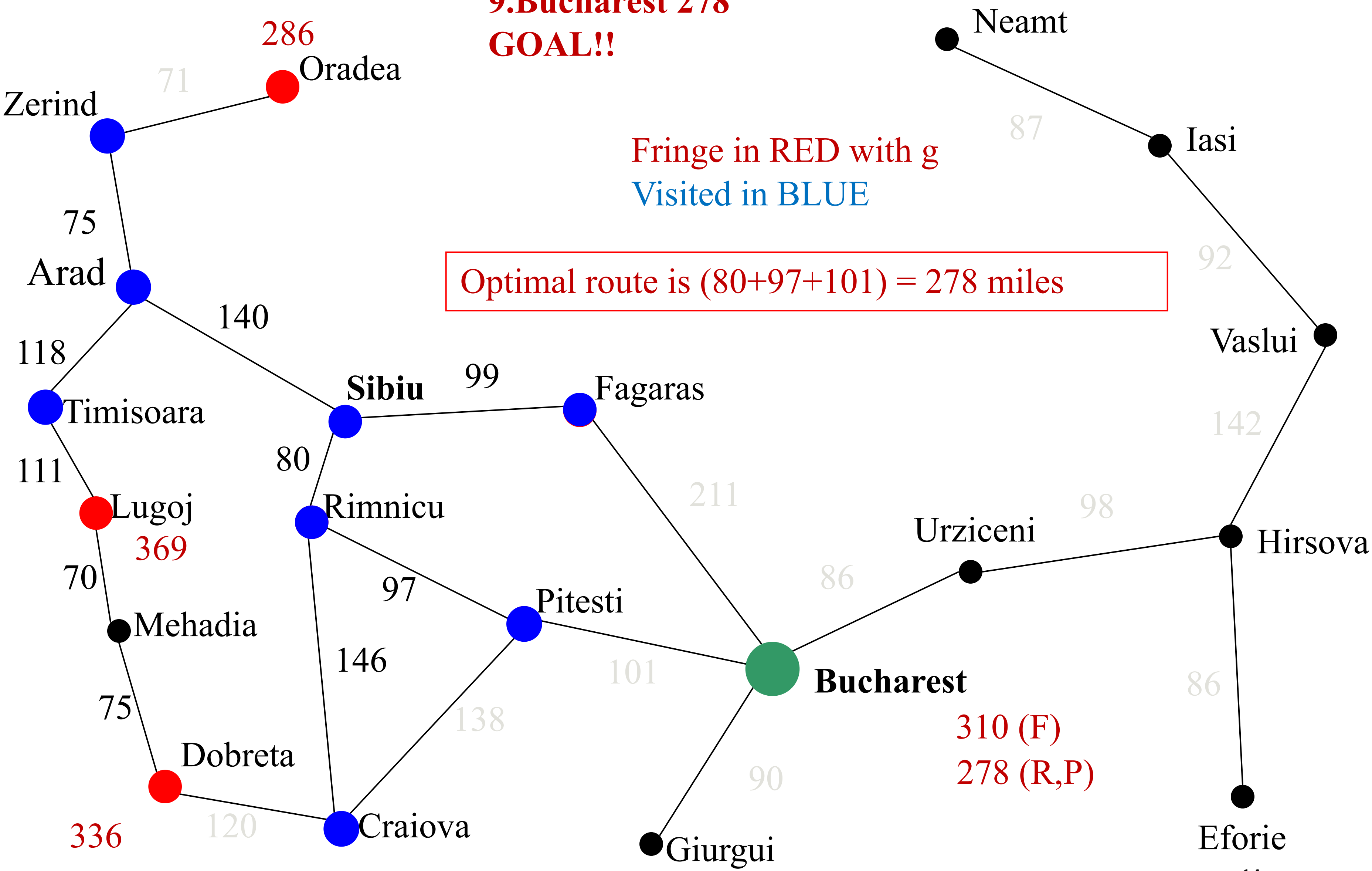
9.Bucharest 278

GOAL!!

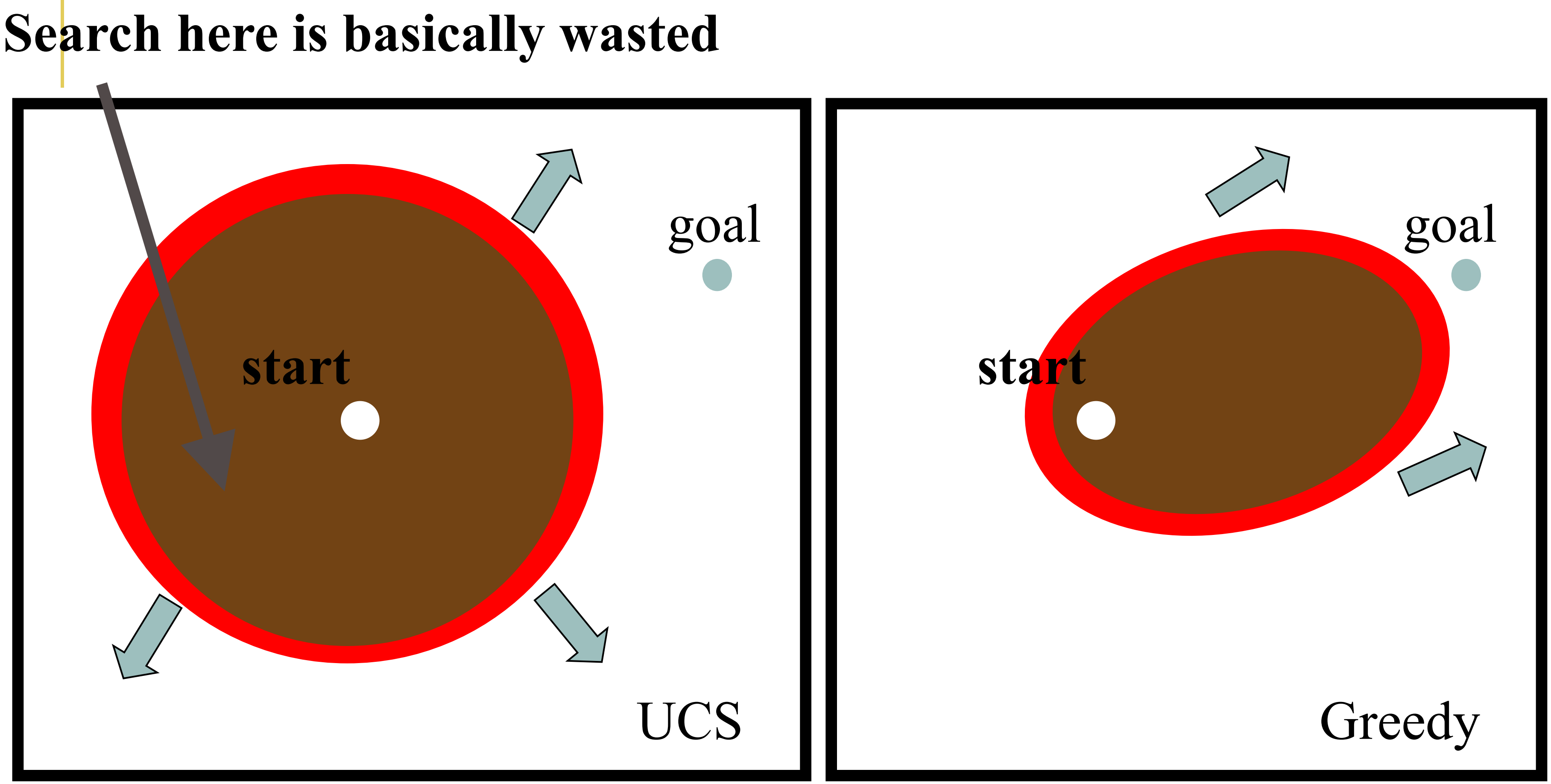
Fringe in RED with g

Visited in BLUE

Optimal route is (80+97+101) = 278 miles



Greedy Search vs. UCS



Bias “too much”: could miss shorter paths

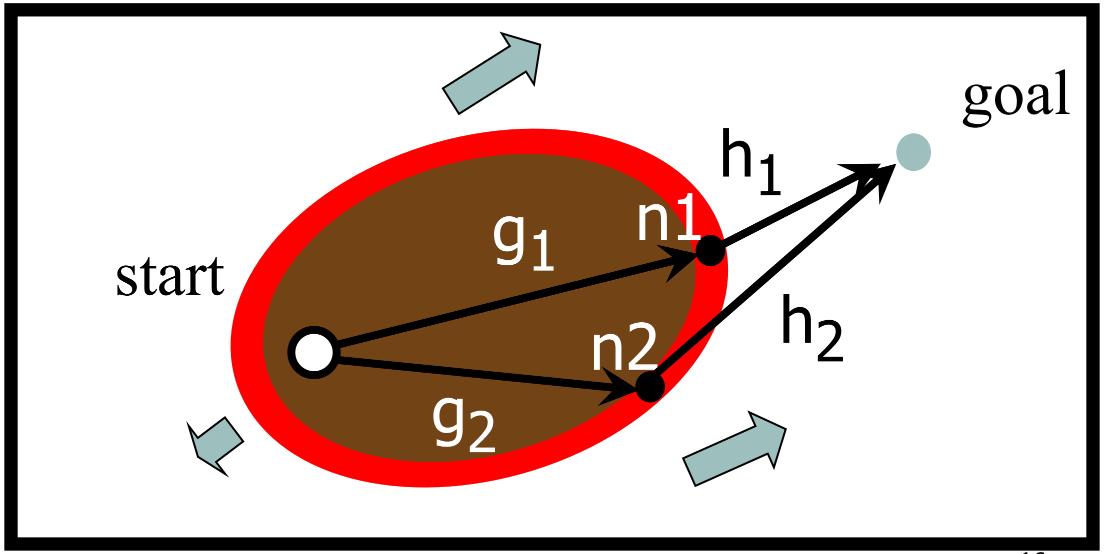
Want to achieve this but stay: complete and optimal

Heuristic Searches - A* Algorithm

Combines the cost so far and the estimated cost to the goal

f(n) = g(n) + h(n)

- **g** - cost from the **initial state to the current state**
- **h** - the cost from the **current state to a goal state**
- **f** - an evaluation of the state: estimated cost of the cheapest solution **through** *n*



Heuristic Searches - A* Algorithm

- A*: search algorithm to find the shortest path to a goal state using a heuristic

$$f = g + h$$

  - **h=0**
    - A* becomes UCS
      - complete & optimal*, but search undirected
  - **g=0**
      - A* becomes Greedy, lose optimality

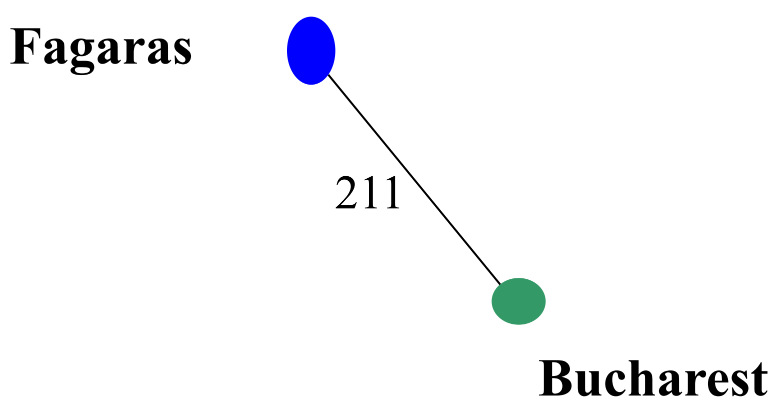
      *when cost along path never decrease

Return GENERAL-SEARCH(problem, **g** + **h**)

Heuristic Searches - A* Algorithm

How to estimate **h**

- Optimal and complete: if the heuristic is **admissible**.
- **Admissible**: the heuristic must **never over estimate** the cost to reach the goal
  - h(n): a valid **lower bound** on cost to the goal



Town	SLD
...	...
Fagaras	178
...	...

STRAIGHT LINE DISTANCES TO BUCHAREST

Straight line distances: an admissible heuristic
Will never overestimate the cost to the goal (no shorter distance between two cities than the straight line distance)

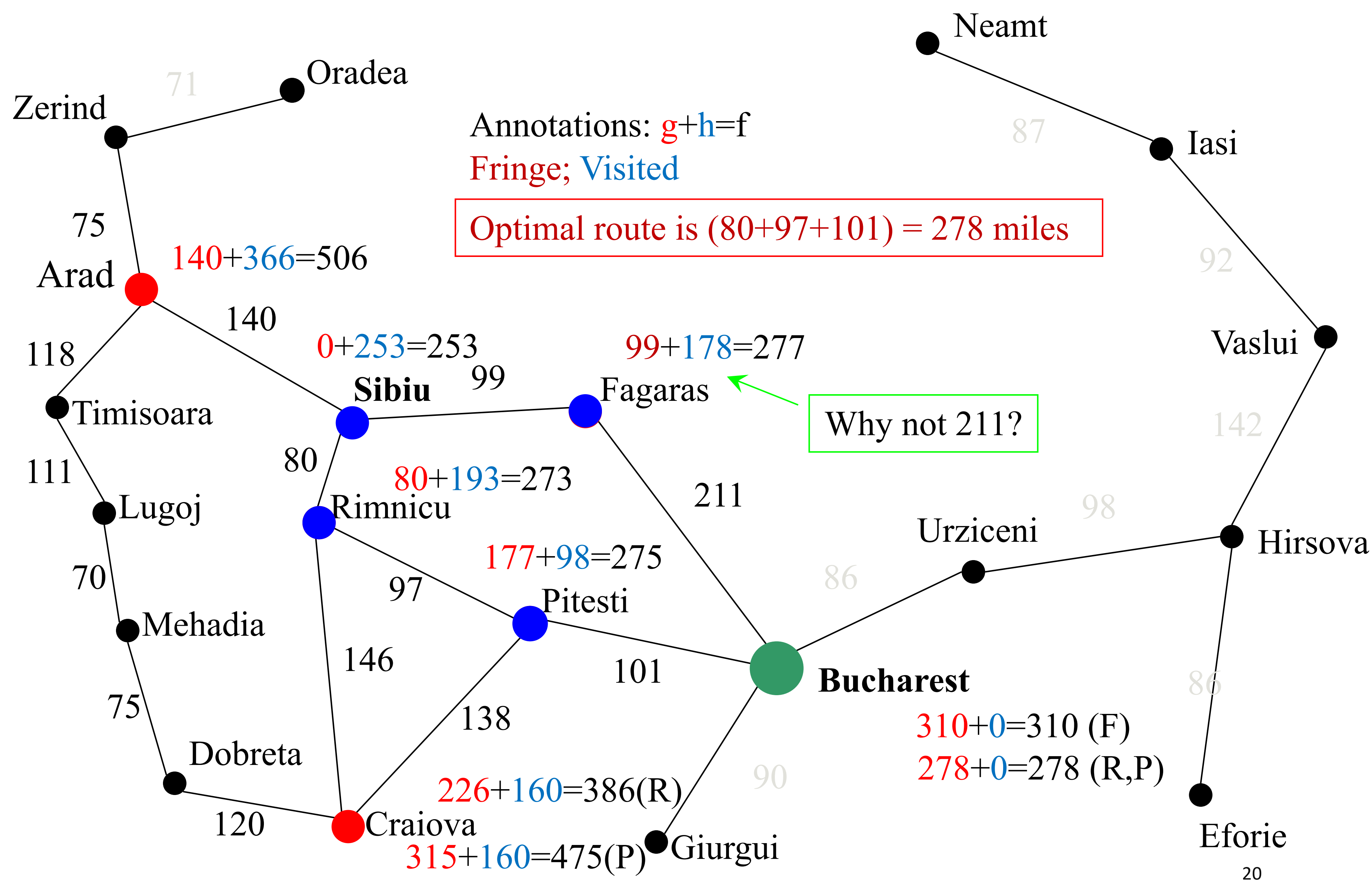
Town	SLD
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

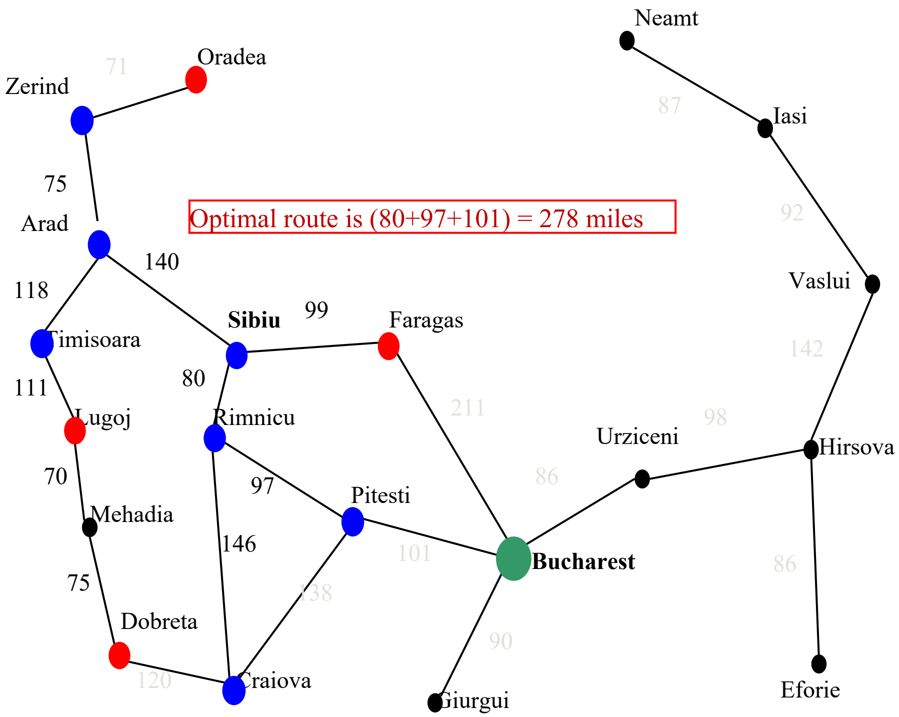
Town	SLD
Mehadai	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

ANIMATION OF A*.

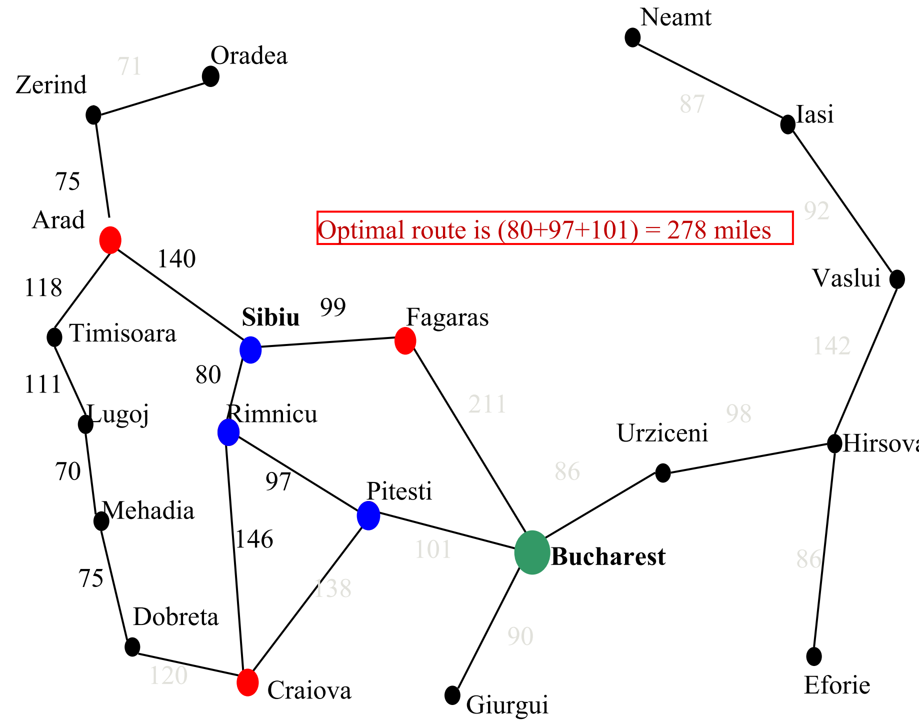
Nodes Expanded

1.Sibiu 2.Rimnicu 3.Pitesti 4.Fagaras **5.Bucharest 278 GOAL!!**





UCS



A*

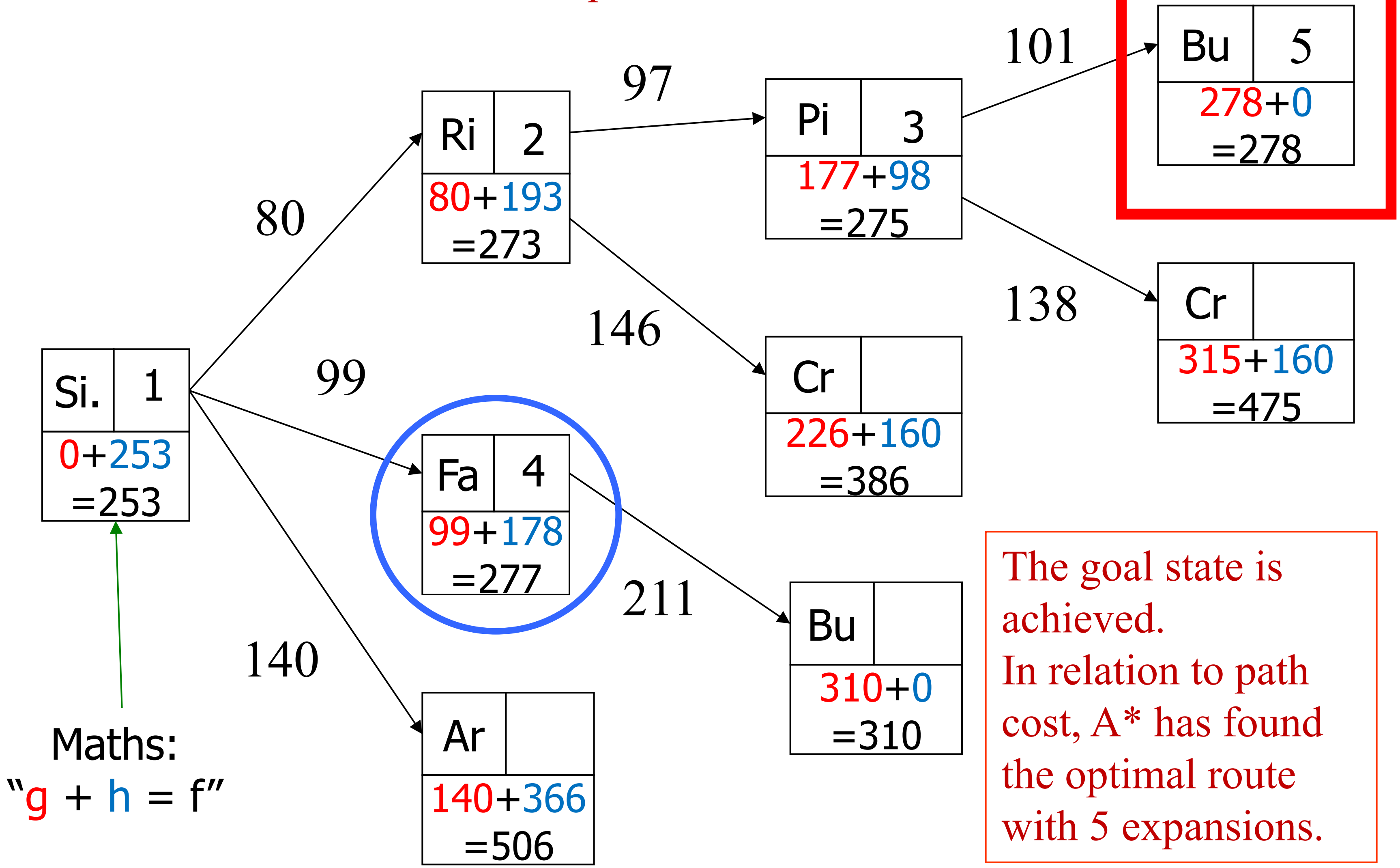
Nodes expanded:

1.Sibiu; 2.Rimnicu; 3.Faragas; 4.Arad; 5.Pitesti; 6.Zerind; 7.Craiova; 8.Timisoara; **9.Bucharest 278**

Nodes Expanded:

1.Sibiu; 2.Rimnicu; 3.Pitesti; 4.Fagaras; **5.Bucharest 278**

In terms of a search tree we could represent this as follows ....



The goal state is achieved.
In relation to path cost, A* has found the optimal route with 5 expansions.

Press space to begin the search

A* SEARCH TREE

A* - example

Typical solution: about twenty steps

Branching factor 3: ~ 3^{20} states

Number of states: 9! /2 states

Aim: develop an admissible heuristic in A* that does not over estimate, to find the optimal solution

Initial State

1	3	4
8	6	2
7		5

Goal State

1	2	3
8		4
7	6	5

A* - possible heuristics

h_1 = no. of tiles in the wrong position

h_2 = sum of the distances of the tiles to their goal positions using the Manhattan Distance

Admissible heuristics: never over estimate

Both are admissible: which one is better?

$h_1 = 4$

$h_2 = 5$

1	3	4
8	6	2
7		5

1	2	3
8		4
7	6	5

Heuristics in A* Algorithm

h_2 = the sum of the distances of the tiles from their goal positions using the Manhattan Distance (=5)

1	**3**	**4**
8	**6**	**2**
7		**5**

5

1	2	3
8		4
7	6	5

What's wrong with this search? is it A*?

Heuristics in A* Algorithm

h_1 = the number of tiles that are in the wrong position (=4)

1	3	4
8	6	2
7		5

4

1	2	3
8		4
7	6	5

What's wrong with this search? is it A*?

Fig 3.17 Breadth-first search of the 8-puzzle, showing order in which states were removed from open

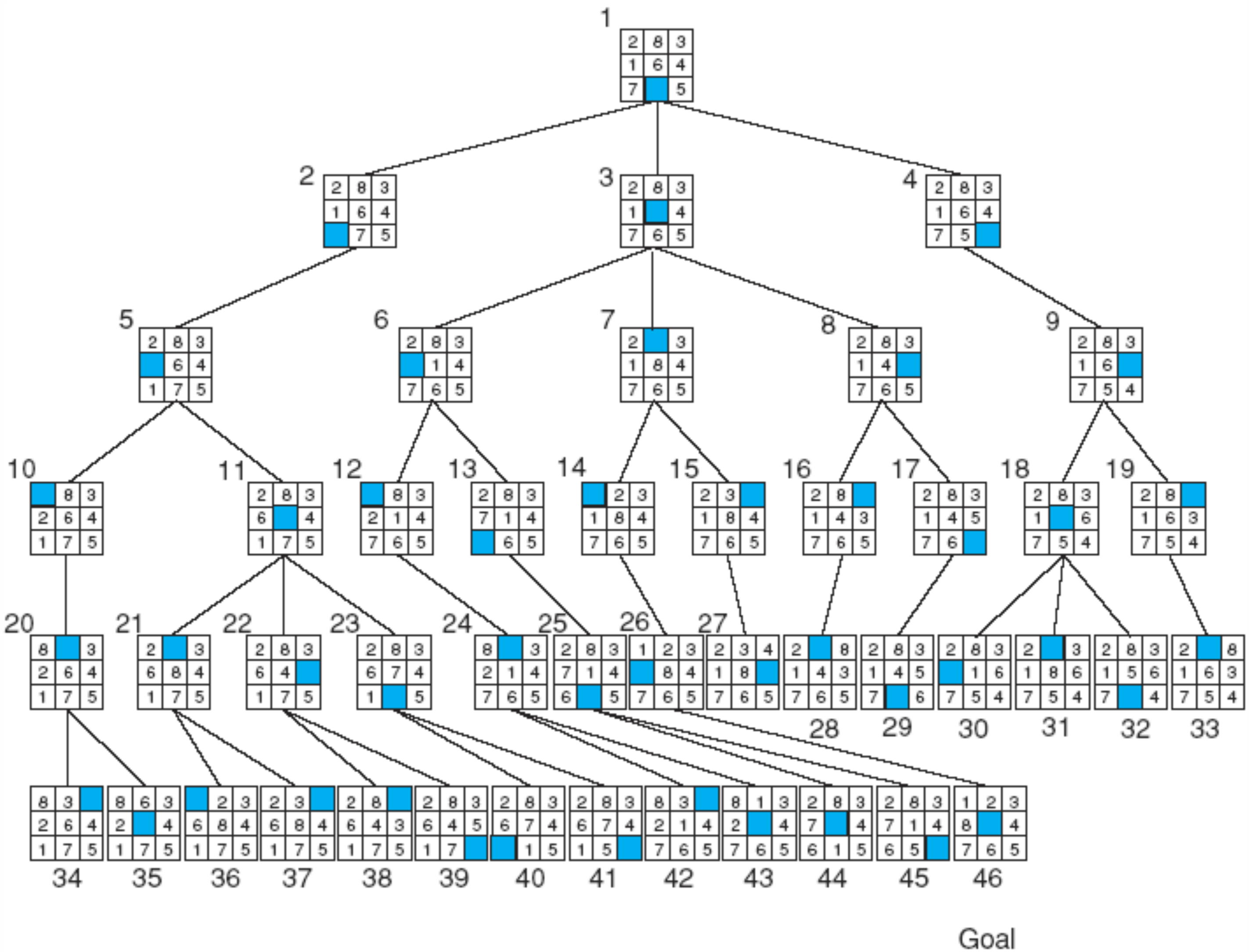
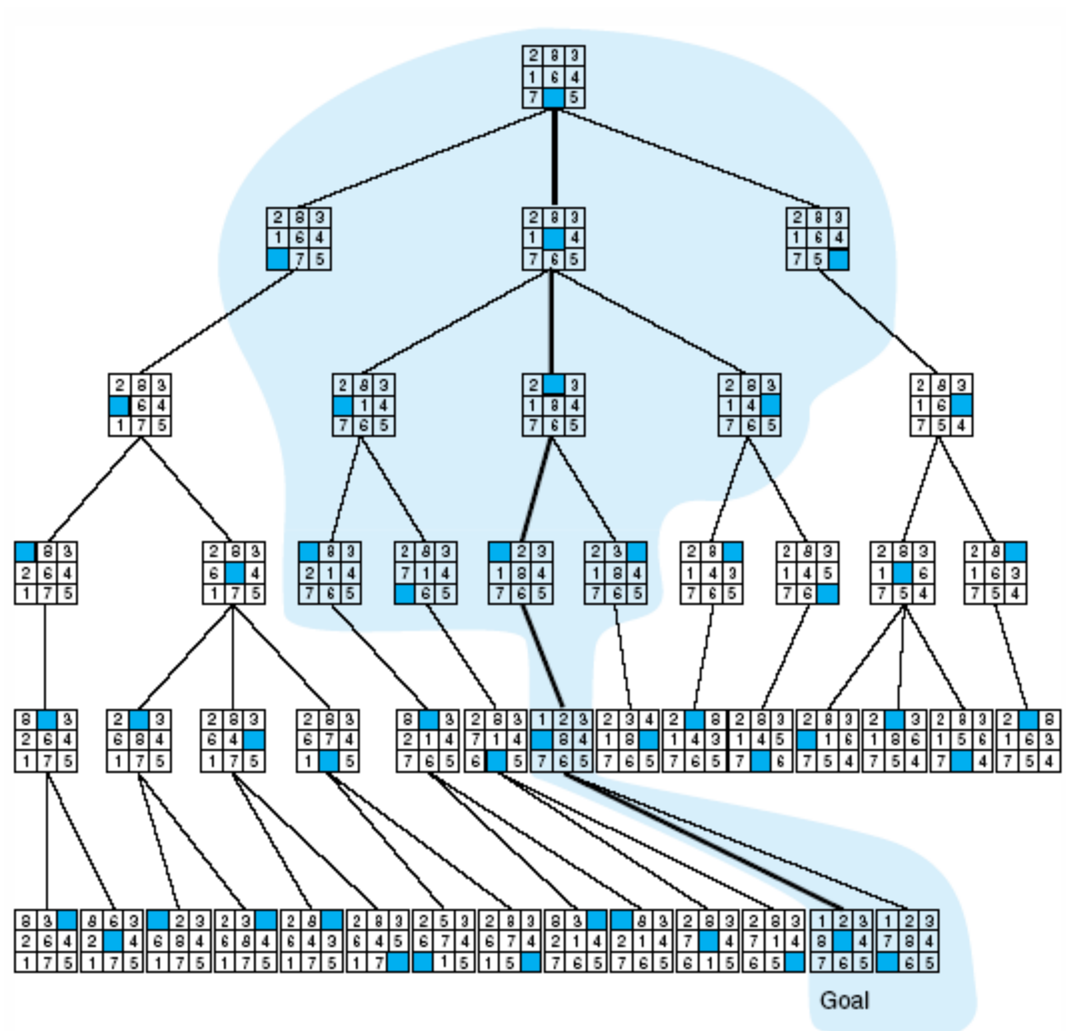


Fig 4.18 Comparison of state space searched using heuristic search with space searched by breadth-first search. The proportion of the graph searched heuristically is shaded. The optimal search selection is in bold. Heuristic used is f(n) = g(n) + h(n) where h(n) is tiles out of place



INFORMEDNESS

- For two A^* heuristics h_1 and h_2, if $h_1(n) <= h_2(n)$, for all states n in the search space, we say h_2 dominates h_1 or heuristic h_2 is more informed than h_1.
- Domination translate to efficiency: A^* using h_2 will never expand more nodes than A^* using h_1.
- Hence it is always better to use a heuristic function with higher values, provided it does not over-estimate and that the computation time for the heuristic is not too large

EFFECTIVE BRANCHING FACTOR

	Search Cost		EBF	
Depth	$A^*(h_1)$	$A^*(h_2)$	$A^*(h_1)$	$A^*(h_2)$
2	6	6	1.79	1.79
4	13	12	1.48	1.45
6	20	18	1.34	1.30
8	39	25	1.33	1.24
10	93	39	1.38	1.22
12	227	73	1.42	1.24
14	539	113	1.44	1.23

- Test from 100 runs with varying solution depths using h1 and h2
- h_2 looks better: fewer nodes expanded. But why?

- Effective branching factor: average no. of branches expanded
- Quality of a heuristic: **average effective branching factor**
- A good heuristic
  - The **closer the estimate** of the heuristic, the better
  - Lower average **effective branching factor**
  - **Admissible**

FURTHER READING

Heuristic searches (Chapter 3.5-3.6 AIMA)

- Greedy search
- A* Search
- Heuristic functions

The start node and the goal node for the state space in Figure 1 are S and G respectively.

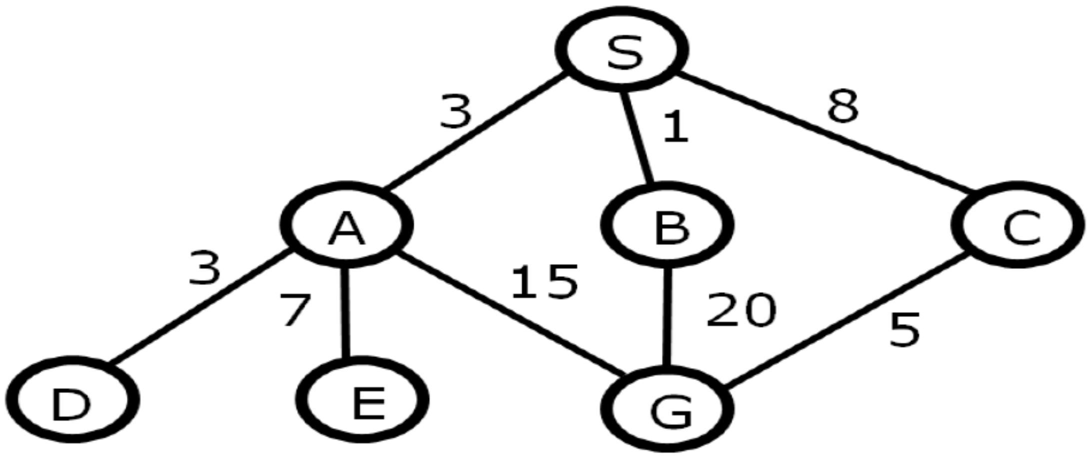


Figure 1.

For each of the search strategies below, work out the *solution path* and the *number of nodes expanded*. Show at each step what *nodes* are in the *queue*. Assume that processed nodes will be ignored.

(i) *depth-first search*;

(ii) *uniform cost search*.

EXAMPLE

Work out the solution path using:

- greedy search
- A* search