

# CS 480 – INTRODUCTION TO ARTIFICIAL INTELLIGENCE

TOPIC: SEARCH  
CHAPTER: 3



**Mustafa Bilgic**



<http://www.cs.iit.edu/~mbilgic>



<https://twitter.com/bilgicm>

# OUTLINE

- Problem solving agent
- The search process
- Tree search
- Graph search
- Complexity analysis

# MOTIVATION

- We have discussed several types of agents
  - Reflex agents, goal-based agents, utility-based agents, etc.
- Reflex agents act purely based on current percept, which is limited
- We now discuss, in detail, the goal-based agents, where a sequence of actions are necessary to reach a desired world state
- Actions have associated costs with them and we'd like to minimize the total cost to reach the goal state

# REPRESENTATION

- The world is represented as an atomic state
- An action takes us from one world state to another
- The environment is
  - Observable
  - Discrete
  - Deterministic
  - Known

# AN EXAMPLE

- We are now in Chicago
- We'd like to drive to Pittsburgh
- We represent the world as “In X” where X is a member of a predefined set of major cities in US
- An action takes us from being in one major city (a world state) to being in a neighboring major city (another world state)
- The cost of the action is the distance traveled
- We would like to travel from Chicago to Pittsburgh while minimizing the distance traveled

# THE SEARCH PROCESS

- We will start at an initial state
- At each state, there is a set of actions we can perform
  - The action that can be performed can be different depending on the current world state
- We have a test that can specify whether a given state is a goal state
- Each action has a cost
- We want to find the optimal solution, i.e., the minimum-cost sequence to reach the goal state

# MORE FORMALLY

- The **initial state**
  - $In(Chicago)$
- A set of **actions**
  - $Go(Milwaukee), Go(Madison), Go(Bloomington)$ , etc.
- A **transition model**:  $Result(s, a): S \rightarrow S$ 
  - $Result(In(Chicago), Go(Madison)) = In(Madison)$
- The **goal test**
  - $\{In(Pittsburgh)\}$
- A **path cost**: Sum of step costs  $c(s, a, s')$ 
  - $c(In(Chicago), Go(Madison), In(Madison)) = 150$

# ASSUMPTIONS AND SIMPLIFICATIONS

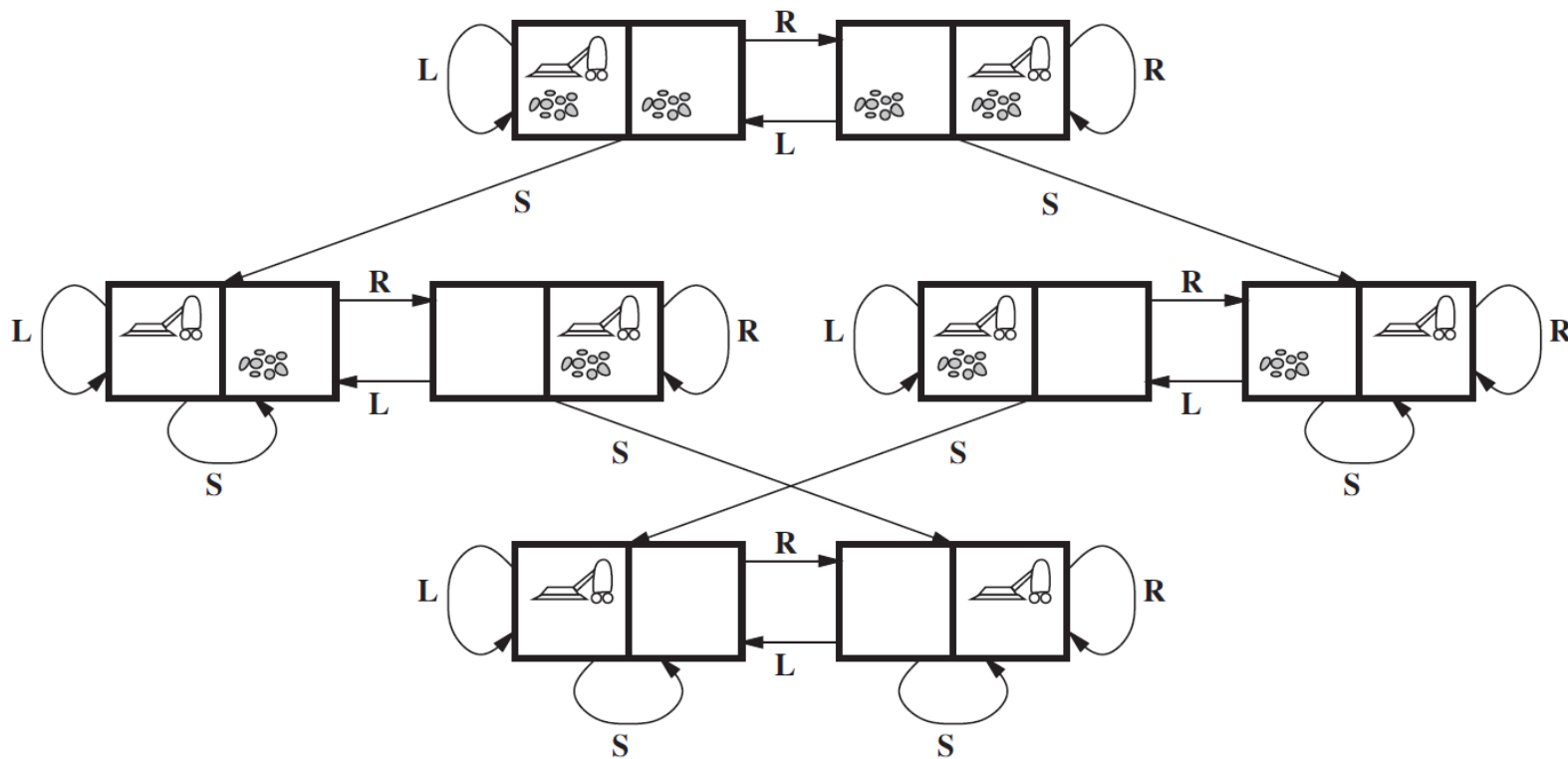
- For the Chicago to Pittsburgh problem, we made several assumptions and simplifications
- What are some of these?



# THE VACUUM WORLD

- **States:** The cross-product of the agent's location and dirt locations
  - *How many possible states?*
- **Initial state:** Any state
- **Actions:** Left, Right, and Suck
- **Transition model:** Expected effects. Except Left on the left square, Right on the right square, and Suck on a clean square has no effect
- **Goal test:** Test whether all squares are clean
- **Path cost:** Each step costs 1; the path cost is then the total number of steps

# THE VACUUM STATE SPACE



# THE 8-PUZZLE

7	2	4
5		6
8	3	1

Start State

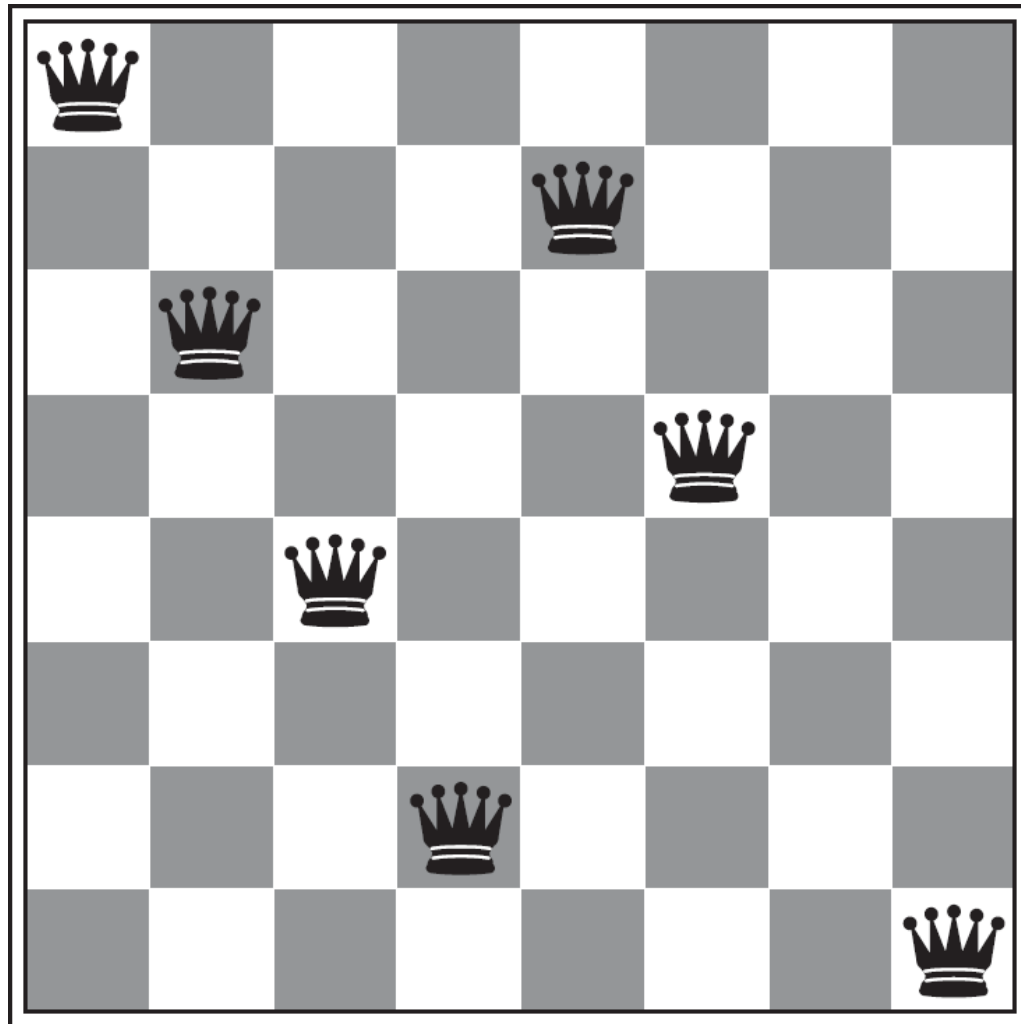
	1	2
3	4	5
6	7	8

Goal State

# THE 8-PUZZLE

- **States:** All possible locations of the eight tiles and the blank tile
  - *How many?*
- **Initial state:** Any state
- **Actions:** Left, Right, Up, Down, except on the edges
- **Transition model:** The state with the blank the numbered tile switched
- **Goal test:** Blank on the top left, others are ordered in increasing order
- **Path cost:** Each step costs 1; the path cost is then the total number of steps

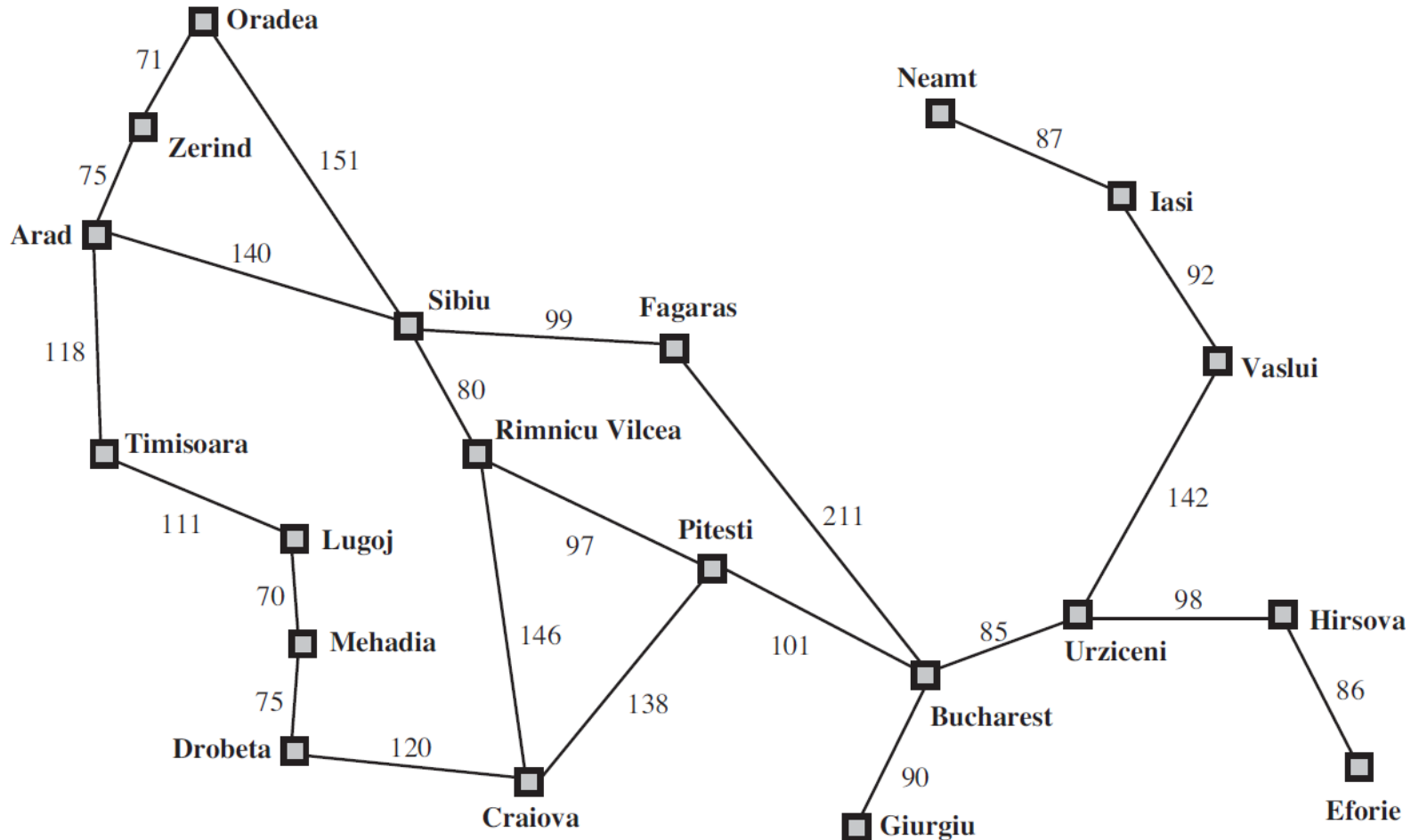
# THE 8-QUEENS PROBLEM



# THE 8-QUEENS PROBLEM

- **States:** Any arrangement of 0 to 8 queens on the board
  - *How many?*
- **Initial state:** No queens on the board
- **Actions:** Add a queen to an empty square
- **Transition model:** The board with the added queen
- **Goal test:** 8 queens on the board, none attacked
- **Path cost:** Each step costs 1; the path cost is then the total number of steps

# ROMANIA



# ROMANIA

- **States:** Any city
- **Initial state:** Arad
- **Actions:** Go to a neighboring city
- **Transition model:** The city that was just traveled
- **Goal test:** Bucharest
- **Path cost:** Step costs are distances between the neighboring cities; path cost is the total distance traveled



Can you think of any real-world problems?

# LET'S SOLVE TOY VERSIONS

- The 4-Queens problem
  - Initial state: Empty board
  - Action: Place a queen on the left most empty column
- The 3-Puzzle problem
  - Initial state: First row: 2 – Blank, Second row: 3 – 1
  - Action: Move the blank tile

# SEARCH

- The key algorithm of the goal-based agent is *search*
  1. Start with the initial state
  2. Expand that state through possible actions
  3. Pick an unexplored state
  4. If goal, return solution; otherwise, go to step 2

# TREE SEARCH

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

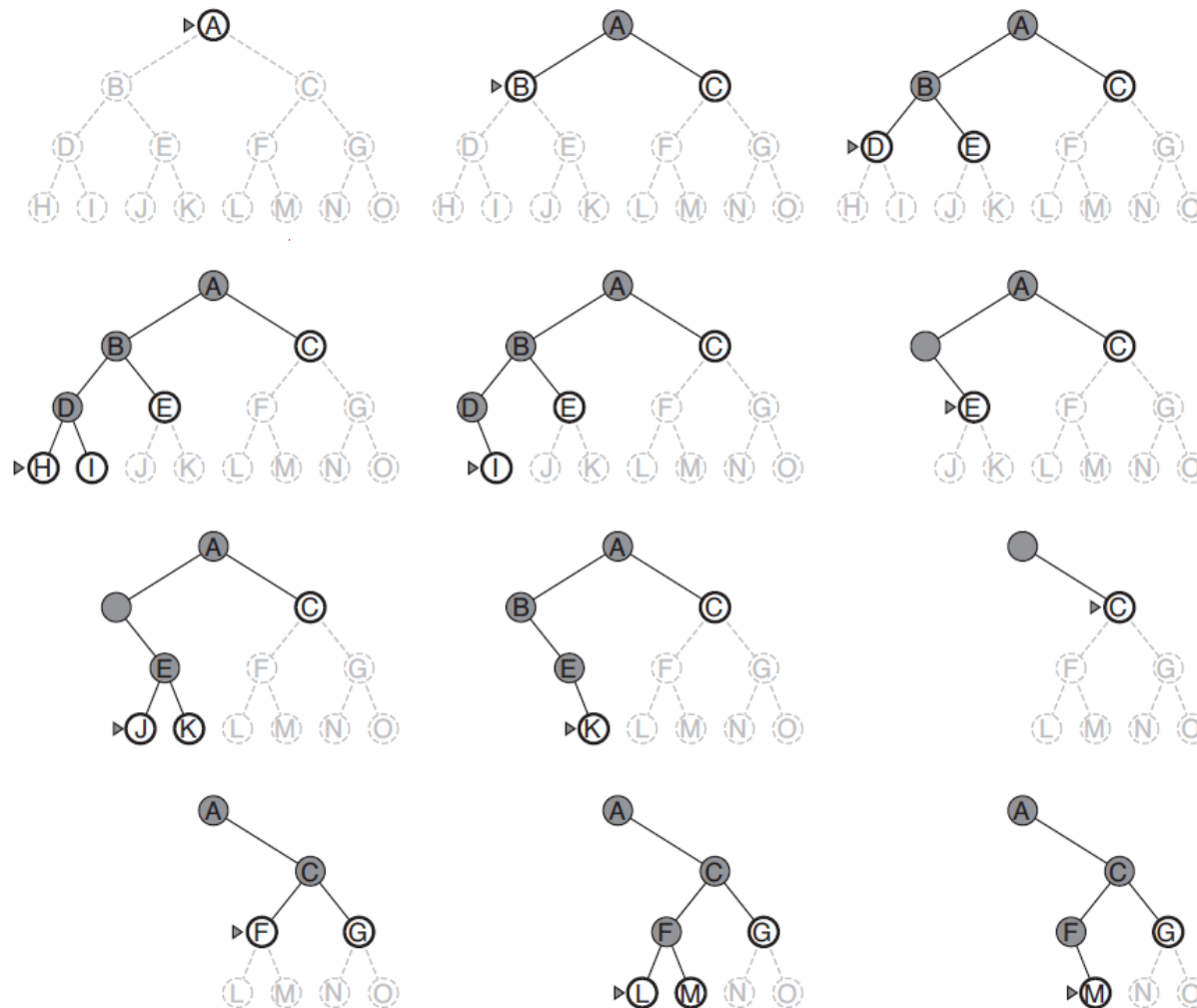
**The key question is which leaf node to pick for expansion.**

## TWO FAMOUS AND IMPORTANT SEARCH STRATEGIES

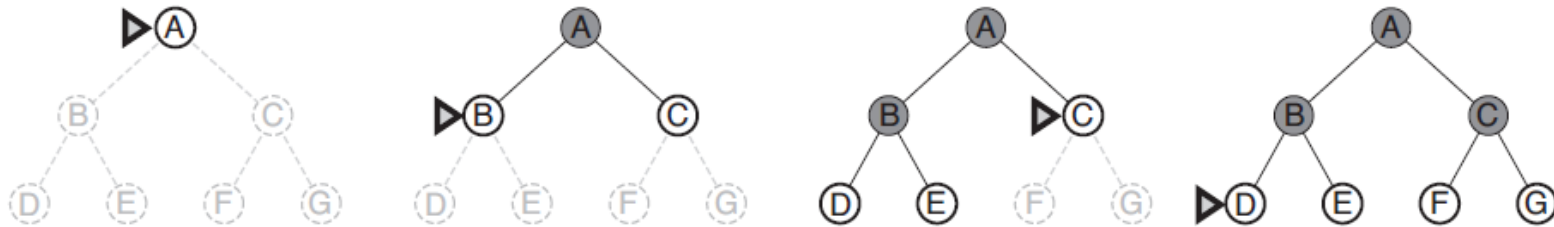
- Depth-first search
  - i.e., go as deep as possible
  - **frontier** = LIFO queue (stack)
- Breadth-first search
  - i.e., go broad; expand all the nodes at level  $i$  before expanding the nodes at level  $i+1$
  - **frontier** = FIFO queue

# DEPTH-FIRST SEARCH

*Right most*  
Is this LIFO or recursive?  
*Left most*



# BREADTH-FIRST SEARCH



# LET'S SEE THEM IN ACTION

- 4-Queens
- 3-Puzzle
- Travel
  - $A \rightarrow B, C$
  - $B \rightarrow D, E$
  - $C \rightarrow F, G$
  - $D \rightarrow H, I$
  - $E \rightarrow J, K$
  - $F \rightarrow L, M$
  - $G \rightarrow N, O$
  - H, I, J, K, L, M, N, O are terminal states
  - Start: A, Goal: O



# COMPLEXITY

- We're concerned with
  - Time complexity
  - Space complexity
  - Completeness (if there is a solution, can it find it?)
  - Optimality (is the found solution the optimal one?)
- Three relevant quantities
  - The number of nodes expanded
  - The maximum size of the **frontier**
  - The cost of the solution

# ANALYSIS – TREE SEARCH

Criterion	Depth-first	Breadth-first
Complete?		
Time		
Space		
Optimal?		

# MEMORY IS A BIG PROBLEM

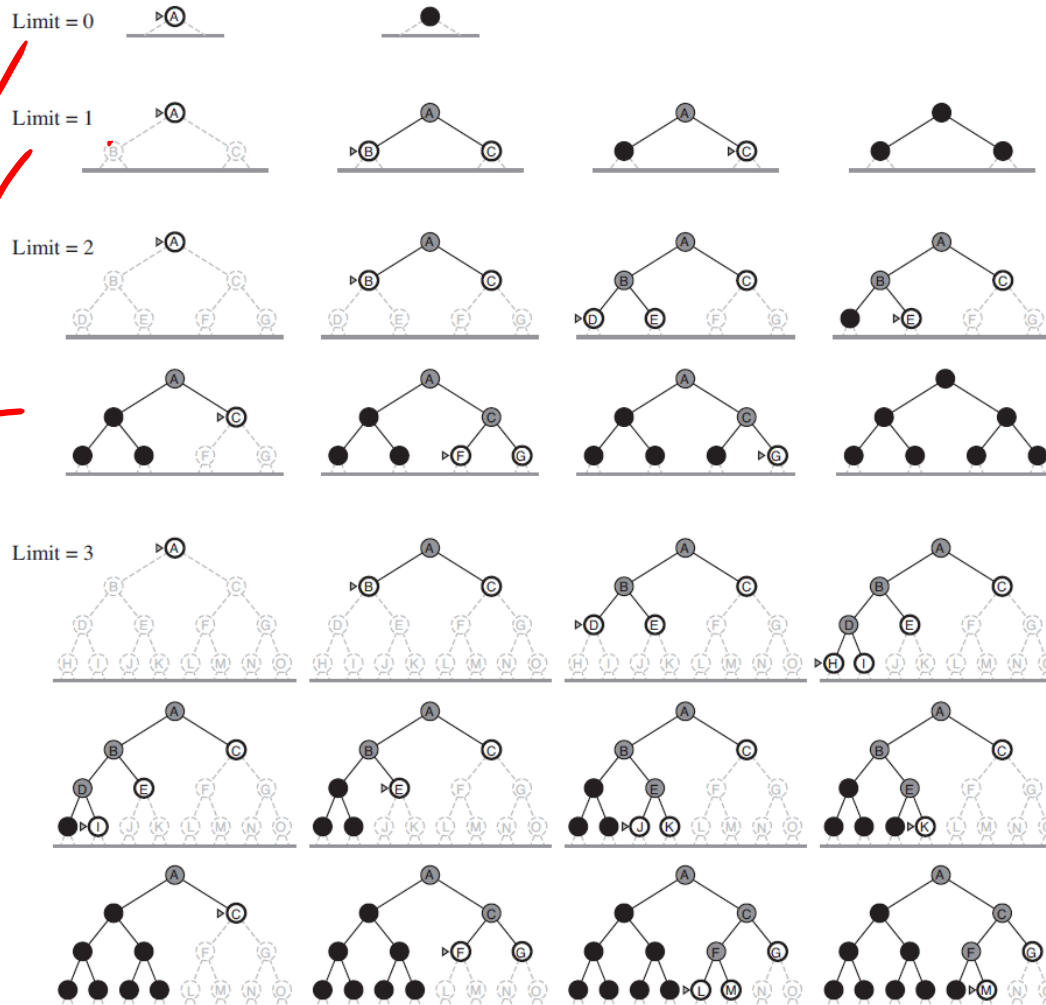
- Breadth-first search has an exponential memory requirement
- Depth-first search requires polynomial size memory, but it is not complete and it is not optimal
- What can we do?
  - Depth-limited search
    - Impose a limit on the depth
    - Is it complete? Optimal?
  - Iterative-deepening search
    - Start with a depth limit of 0, and run depth-limited search; if no goal found, increase the depth limit and repeat the process
    - Is it complete? Optimal?

This is the textbook version; we  
do the DFS part differently.

What are the time and space  
complexities for IDS?

# ITERATIVE-DEEPENING SEARCH

IDS  
complete ✓  
optimal ✓



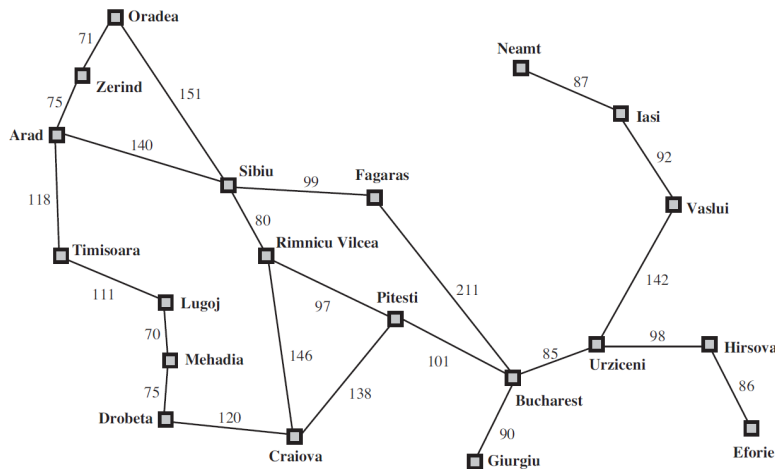
# WHAT IF THE STEP COSTS ARE NOT IDENTICAL?

- BFS and Iterative Deepening are not guaranteed to be optimal anymore
- Define  $f(n) = h(n) + g(n)$ 
  - $h(n)$  : Cost estimate from  $n$  to the goal
  - $g(n)$  : Cost from the initial state to  $n$
- 1. Uniform-cost search
  - Expand using  $g(n)$
- 2. Greedy best-first search
  - Expand using  $h(n)$
- 3. A\* search
  - Expand using  $f(n)$

# THE RUNNING EXAMPLE

Edge costs

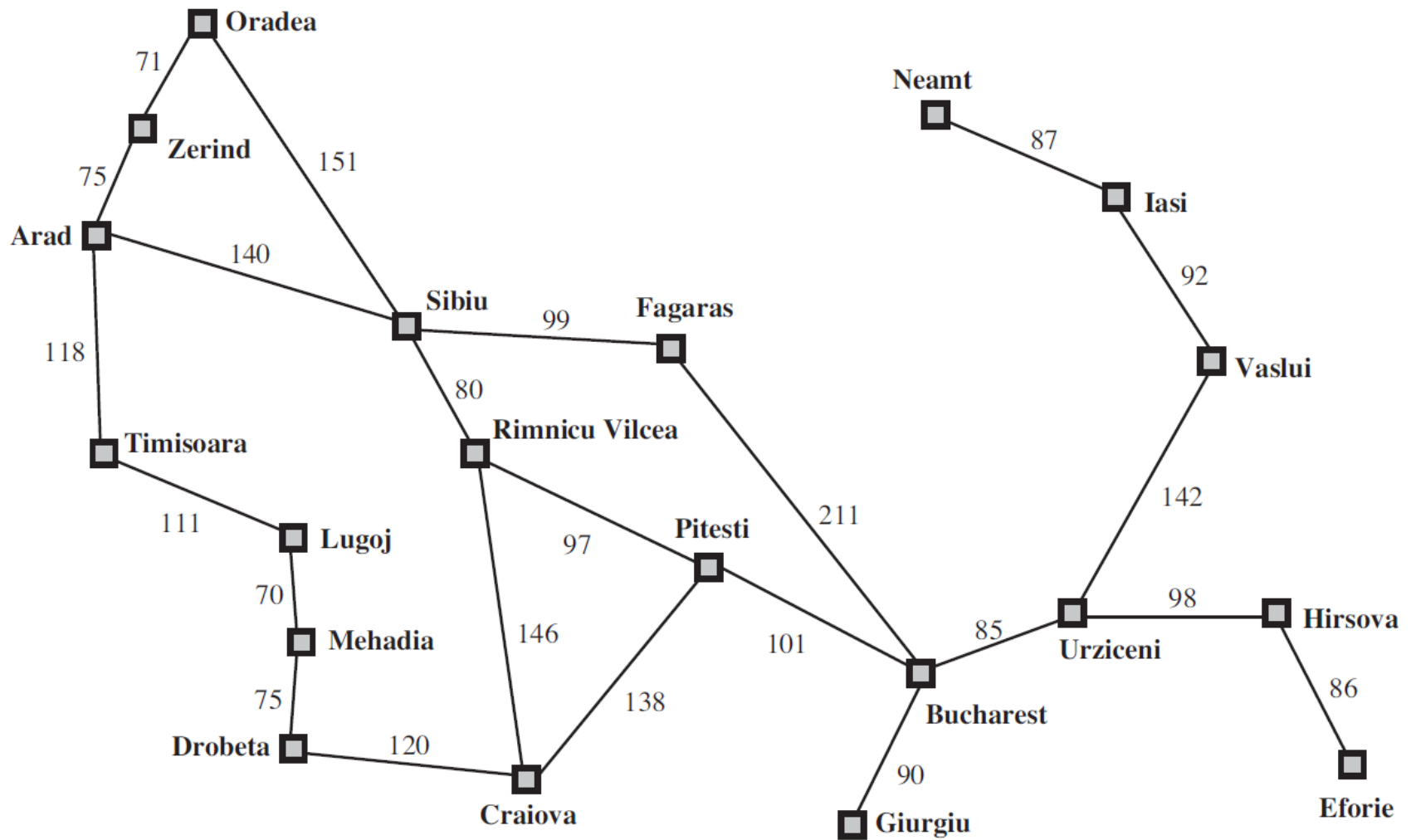
$h(n)$



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

# UNIFORM-COST SEARCH

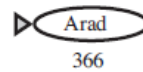
- Arad to Bucharest



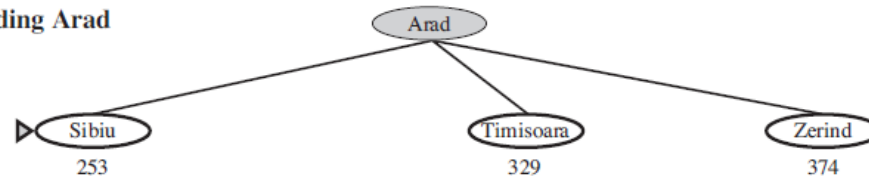


# GREEDY BEST-FIRST SEARCH

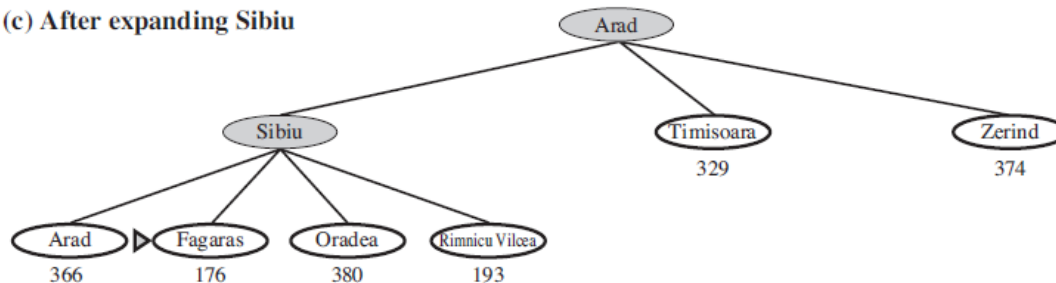
(a) The initial state



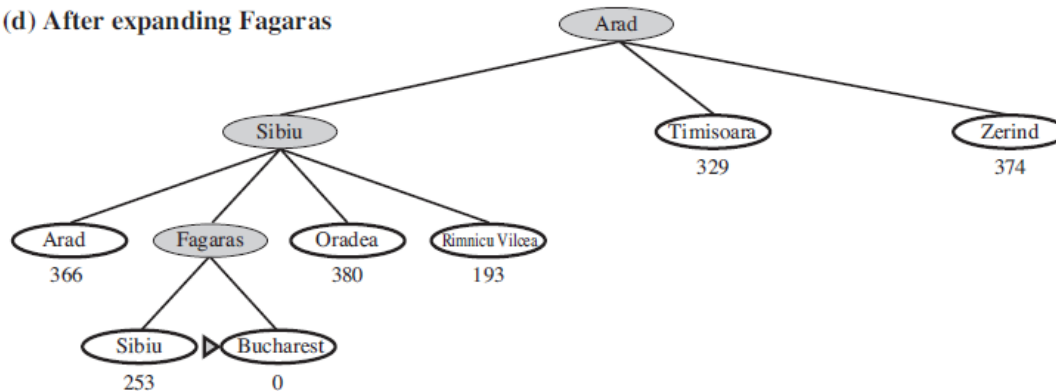
(b) After expanding Arad

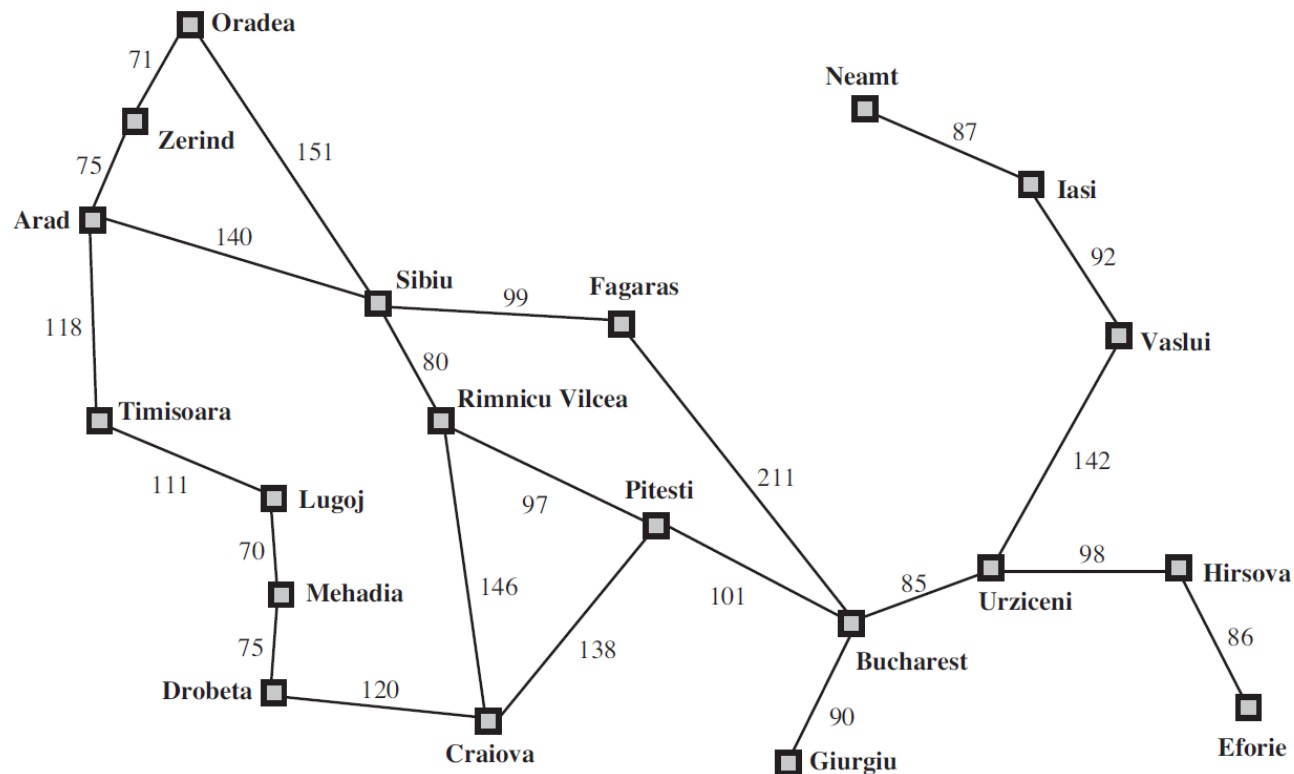


(c) After expanding Sibiu



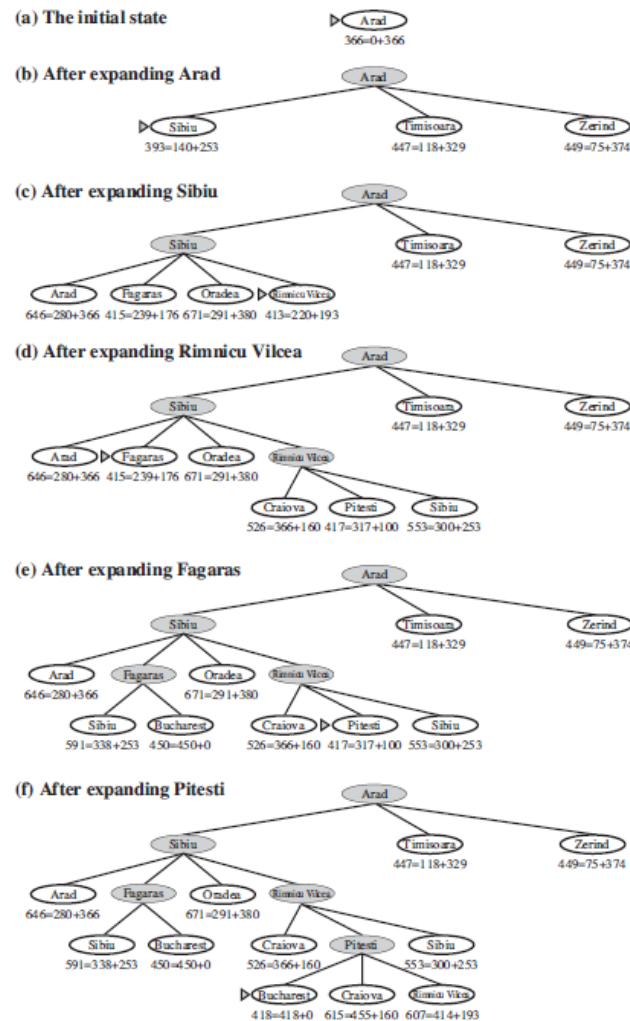
(d) After expanding Fagaras





<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

# A\* SEARCH



# REPEATED STATES

- Repeated states can be a major problem
- Repeated states are especially common for problems where actions are reversible
  - But, they also occur for problems where there are multiple distinct paths between two states
- One way to avoid this problem is to remember the set of explored and generated states

# GRAPH SEARCH

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

    initialize the frontier using the initial state of *problem*

*initialize the explored set to be empty*

**loop do**

**if** the frontier is empty **then return** failure

        choose a leaf node and remove it from the frontier

**if** the node contains a goal state **then return** the corresponding solution

*add the node to the explored set*

        expand the chosen node

        for child in children of the chosen node:

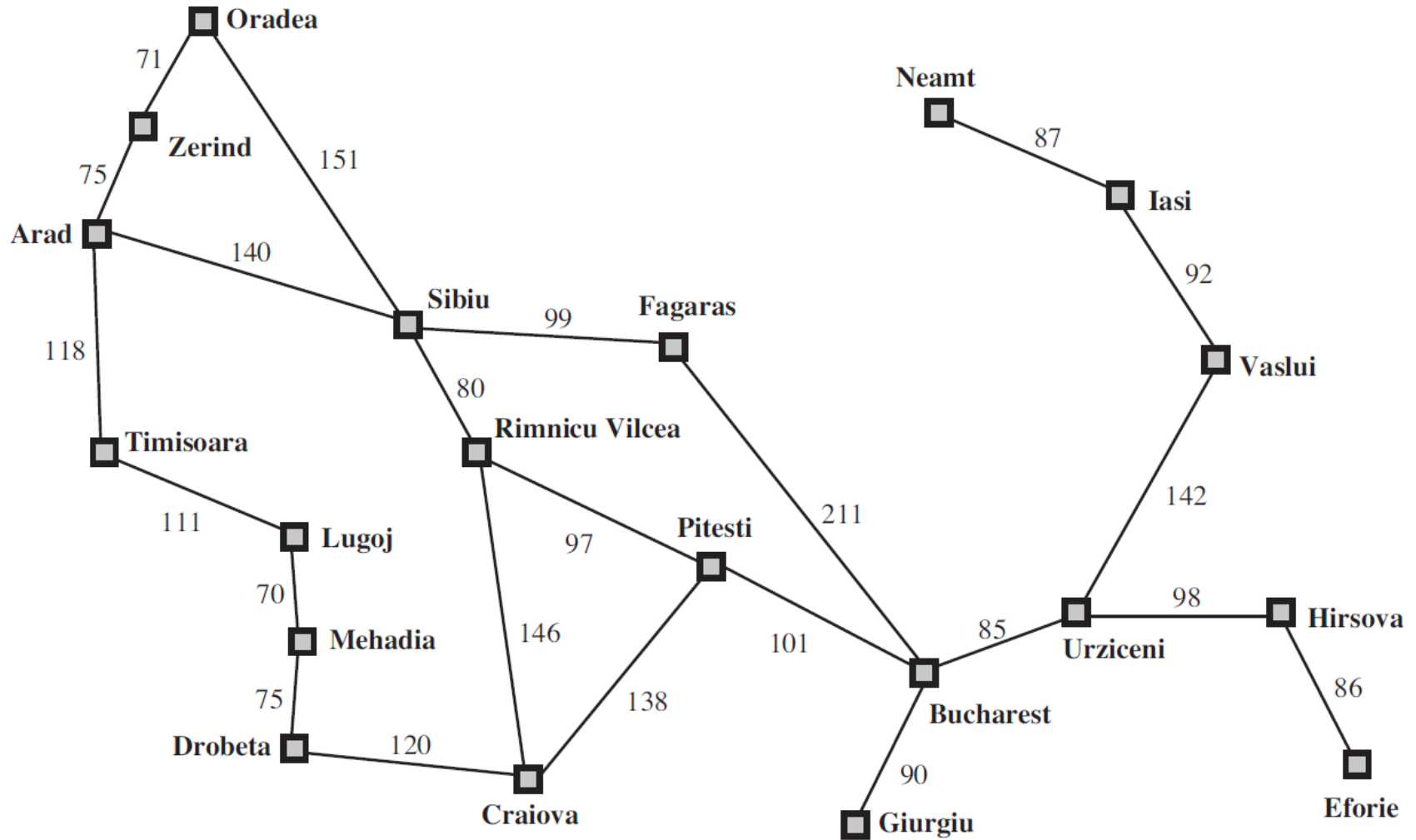
**if** child is not in explored:

                add child to frontier only if

                    child is not in frontier or

                    child is better than the one in frontier

# GRAPH SEARCH EXAMPLES



# QUESTIONS TO THINK ABOUT

How does graph search affect

- Completeness,
- Time complexity,
- Space complexity, and
- Optimality

of various search algorithms?



# A\* OPTIMAL?

- Tree version is optimal if
  - $h(n)$  is admissible
- Graph version is optimal if
  - $h(n)$  is consistent

# ADMISSIBLE? CONSISTENT?

## ○ **Admissible** if

- $h(n)$  never overestimates the optimal cost
- That is  $h(n)$  is always optimistic
- E.g., straight line distance between two cities

## ○ **Consistent** if

- If  $h(n) \leq c(n, n') + h(n')$
- $n'$  is the successor of  $n$
- Triangle inequality

- If a heuristic is consistent, it is also admissible  
(the other way around is not guaranteed)



# A\* OPTIMALITY – GRAPH SEARCH – PROOF

1. If  $h(n)$  is consistent, then  $f(n)$  along any path is non-decreasing
2. When  $n$  is expanded, the optimal path to it has been found

# A\* VS UNIFORM-COST SEARCH

- Compare and contrast A\* and UCS
  - Which one is better and why?
- Given the same heuristic function, can you design a complete and optimal algorithm that expands fewer nodes than A\*?

# 8-PUZZLE HEURISTICS

- $h_1$  = The number of misplaced tiles
- $h_2$  = Manhattan distance

$h_2 > h_1$

Let's see an example

5	6	1
2	4	8
3	7	

# EFFECTIVE BRANCHING FACTOR

- If the total number of nodes generated is  $N$  and the solution depth is  $d$ , then
  - $b^*$  is the branching factor that a uniform tree of depth  $d$  would need to have in order to contain  $N+1$  nodes
- $N+1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$
- If  $A^*$  finds a solution at depth 4 using 40 nodes, what is  $b^*$ ?
  - $\approx 2.182$
- A good heuristic function achieves  $b^* \approx 1$

# BFS vs A\*

$$1 + b^* + b^{*2} - - - (b^*)^{28}$$

$d$	Search Cost (nodes generated)			Effective Branching Factor		
	BFS	A*( $h_1$ )	A*( $h_2$ )	BFS	A*( $h_1$ )	A*( $h_2$ )
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

**Figure 3.26** Comparison of the search costs and effective branching factors for 8-puzzle problems using breadth-first search, A\* with  $h_1$  (misplaced tiles), and A\* with  $h_2$  (Manhattan distance). Data are averaged over 100 puzzles for each solution length  $d$  from 6 to 28.

# HEURISTIC FUNCTIONS – $h(n) = 0$

- Let  $h(n) = 0$  for  $\forall n$ 
  - Is it admissible? *Yes*
  - Is it consistent? *Yes*
  - Is it any <sup>useful?</sup> good? *No*

$h(n) = 0 \forall n$   
 $A^*$   $\equiv$  ?  $\overline{BFS}$   $\approx$  ?  $BFS$   
 $C = k$



# HEURISTIC FUNCTIONS – DOMINATES?

- Can we say an admissible heuristic function  $h_i$  is always better than another admissible heuristic function  $h_j$ ?

- $h_i$  **dominates**  $h_j$  iff

- $h_i(n) \geq h_j(n) \forall n$

Manhattan  
Distance  
8 Puzzle

$\geq$   
Misplaced  
Tiles

$$h(n) \quad \forall n$$

## HEURISTIC FUNCTIONS – UCS AS HEURISTIC?

- Run uniform-cost search, find the solution, return the cost of the solution as the heuristic value. Can you design an admissible heuristic that dominates this heuristic?

$$\exists n \quad h_i(n) > h_{ucs}(n) \quad \times$$

$$\begin{aligned} & \text{UCS} \quad \forall n \rightarrow G \\ & S \rightarrow G \end{aligned}$$

## HEURISTIC FUNCTIONS – MULTIPLE HEURISTICS

- Let  $h(n) = \max(h_1(n), h_2(n), \dots, h_m(n))$ , where each  $h_i(n)$  is admissible
  - Is  $h(n)$  admissible? *Yes*
  - Can we say  $h(n)$  dominates every  $h_i(n)$ ? *Yes*
  - What's the catch? *Time*

# SUMMARY

- Depth-first search
- Breadth-first search
- Depth-limited depth-first search
- Iterative deepening
- Uniform cost search:  $g(n)$
- Greedy best-first search:  $h(n)$
- A\* search:  $g(n) + h(n)$

Which one is better under what conditions?

Complete  
Optimal  
Time  
Space

# OTHERS

## ○ Bidirectional search

- Expands two frontiers: one around the start state and one around the goal state

## ○ Beam search

- Puts a limit on the size of the frontier

## ○ Modified versions of A\*

- Bidirectional A\*
- Iterative deepening A\*
- Weighted A\*:  $f = w \cdot h + g$

- Which algorithms do we get if  $w=0$ ,  $w=1$ ,  $w=\text{inf}$ ?
- Is the algorithm still optimal if  $w > 1$ ?

UCS A\*      GBFS  
 $w \rightarrow \infty$   
No

# HISTORICAL NOTES FROM THE BOOK

- BFS – 1959
- Dijkstra's algorithm – 1959
  - Modified to be UCS – 1971
- Iterative deepening used for timed chess – 1977
- Heuristic search – 1965
- A\* – 1968
  - Variations are still under research
    - Bidirectional A\* applied to maps and landmarks in 2006
      - 24-million point on the graph
      - An optimal path between any two points searches only 0.1% of the graph
- Search is still an active area of research
  - <https://aaai.org/Conferences/AAAI/aaai.php>

# NEXT

- Skip Chapter 4
- Chapter 5: Adversarial search and games