Artificial Intelligence

3. Search

Shashi Prabh

School of Engineering and Applied Science
Ahmedabad University

Contents

- Goal: use search to solve problems

- Topics
    - Problem-solving agents
    - Searching
        - Uninformed Search
        - Informed Serach

Problem-Solving Agents

Main idea

- When the current action to take in not immediately obvious and when environment is
  - fully observable,
  - static,
  - and discrete,

  then the agents can plan ahead by doing a simple search.
  - Such an agent is called **problem-solving agent**.
    - Uses atomic representation of states
    - Search in deterministic, known and single-agent environments is straightforward.

Problem-Solving Agent

- Finds a sequence of actions that form a path to the goal state(s)
- Steps
  - Goal Formulation: limits the action choices
  - Problem formulation: a description of the states and actions to reach the goal
  - Search: simulates sequences of actions in its model, produces solution
    - In partially observable or nondeterministic environments, the solution would be a branching strategy.
  - Execution
    - In fully observable, deterministic and known environment, the agent can ignore the percepts – open loop system
    - Otherwise, percepts need to be monitored – closed loop system

Navigation example

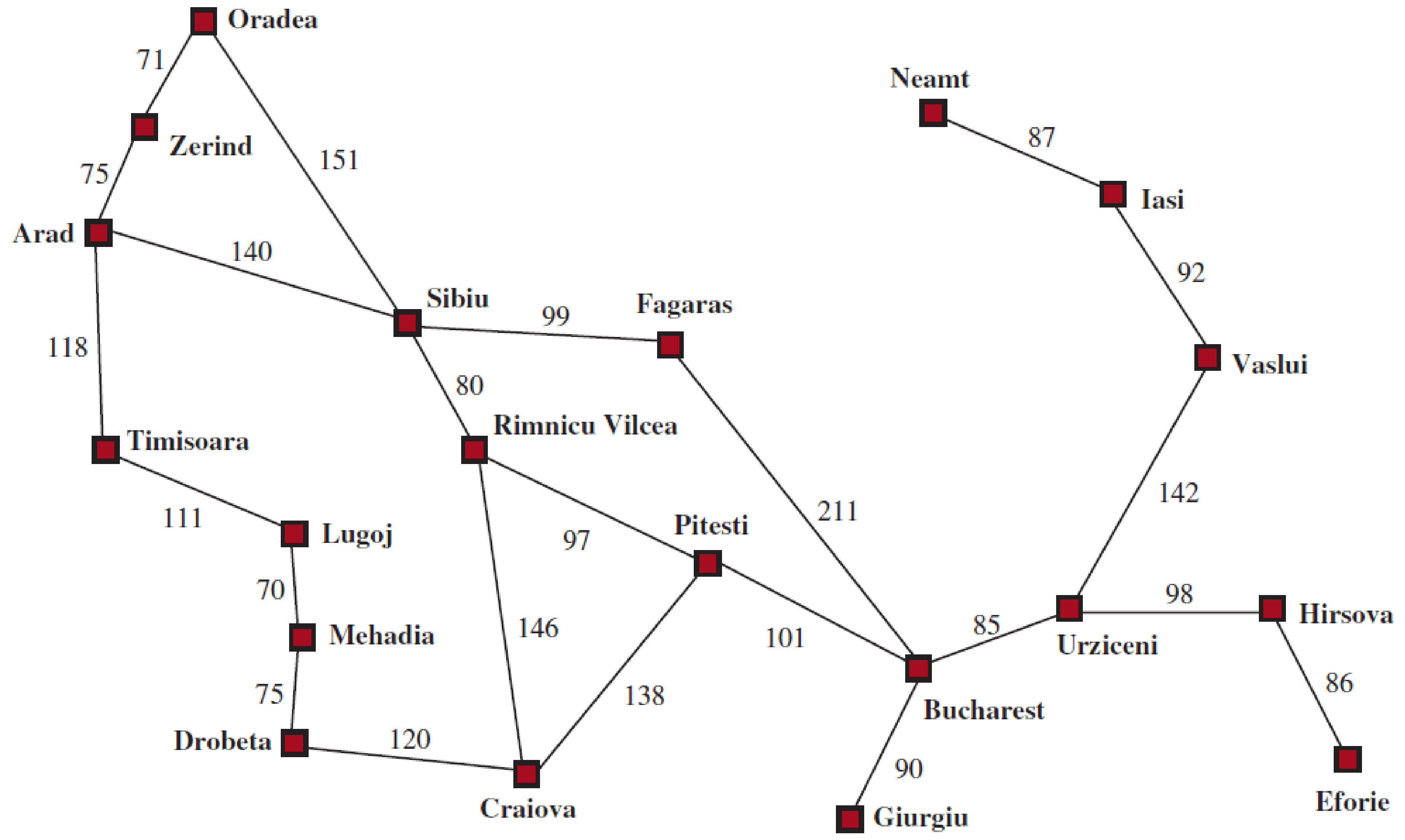


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Defining a search problem - I

1. State Space: set of possible states of the environment
   a. The initial state that the agent starts in. For example: Arad

2. A set of one or more goal states

3. The actions available to the agent.
   - Given a state s, ACTIONS(s) returns a set of actions that can be executed in s
     - We say that each of these actions is applicable in s
   - ACTIONS (Arad) = {ToSibiu, ToTimisoara, ToZerind}

4. Transition model: describes what does each action do
   - RESULT(s, a) = the state that results from doing action a in state s
     - RESULT(Arad, ToZerind) = Zerind .

Defining a search problem - II

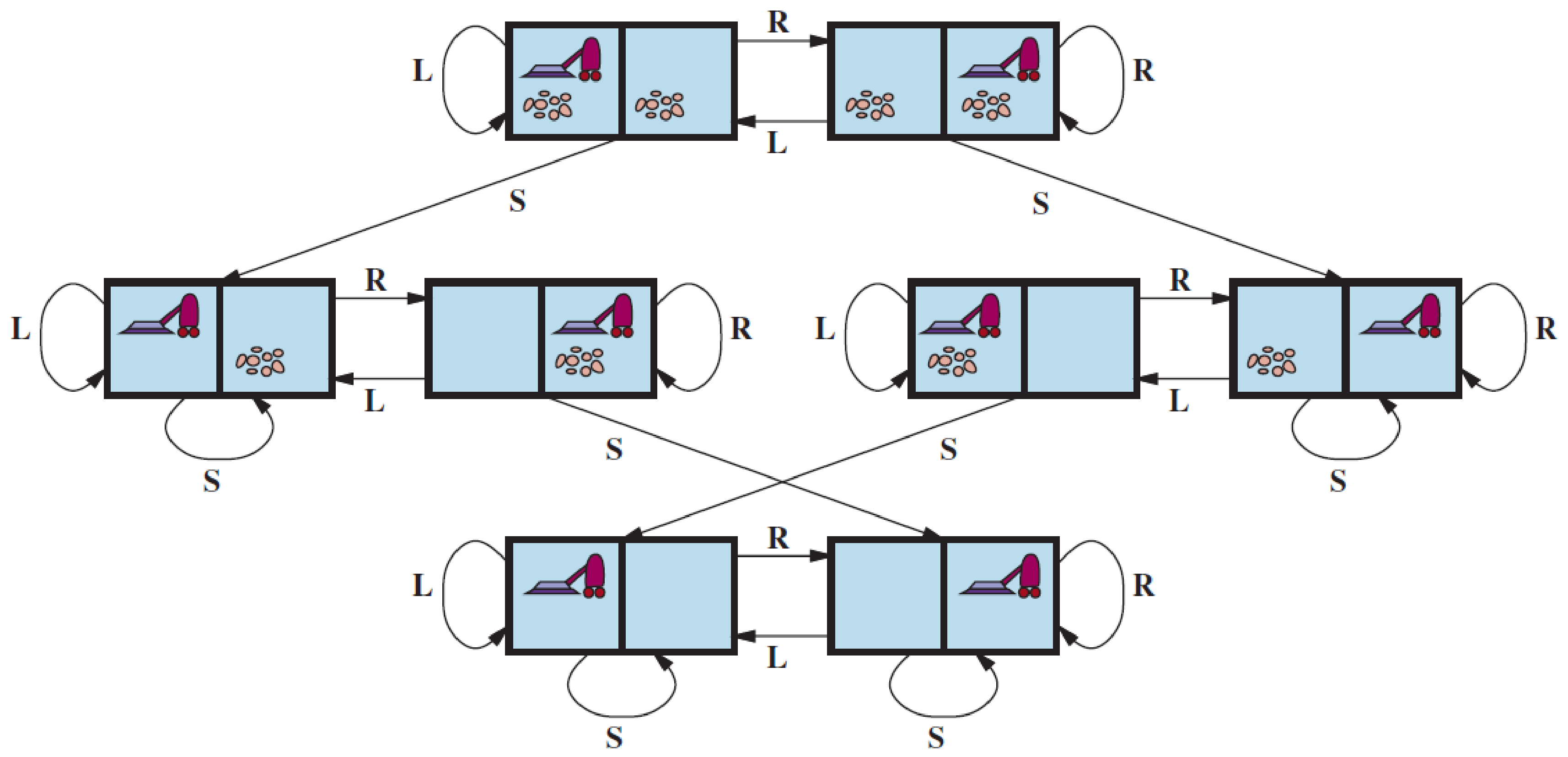
- An action cost function, denoted by ACTION-COST(s, a, s′), gives the numeric cost of applying action a in state s to reach state s′
    - Notation: c(s, a, s′) when we are doing math
    - Should use a cost function that reflects its own performance measure
        - Example: for route-finding agents, the cost of an action might be the length in miles or it might be the time it takes to complete the action.
- A sequence of actions forms a path, and a solution is a path from the initial state to a goal state
- An optimal solution has the lowest path cost among all solutions.

Vacuum-World Example

- State space: 8 atomic states
  - Agent can be in either of the two cells, and each call can have dirt or not
- Initial state: Any one of the 8 states
- Actions: Suck, MoveLeft, and MoveRight
  - In a 2-D multi-cell world Forward, Backward, TurnRight, and TurnLeft.
- Transition model: Suck removes any dirt from a cell, move left/right takes to the other room (unless it hits a wall, in which case the action has no effect)
- Goal states: The states in which every cell is clean
- Action cost: +1 for each action
- What about 3x3 grid world?

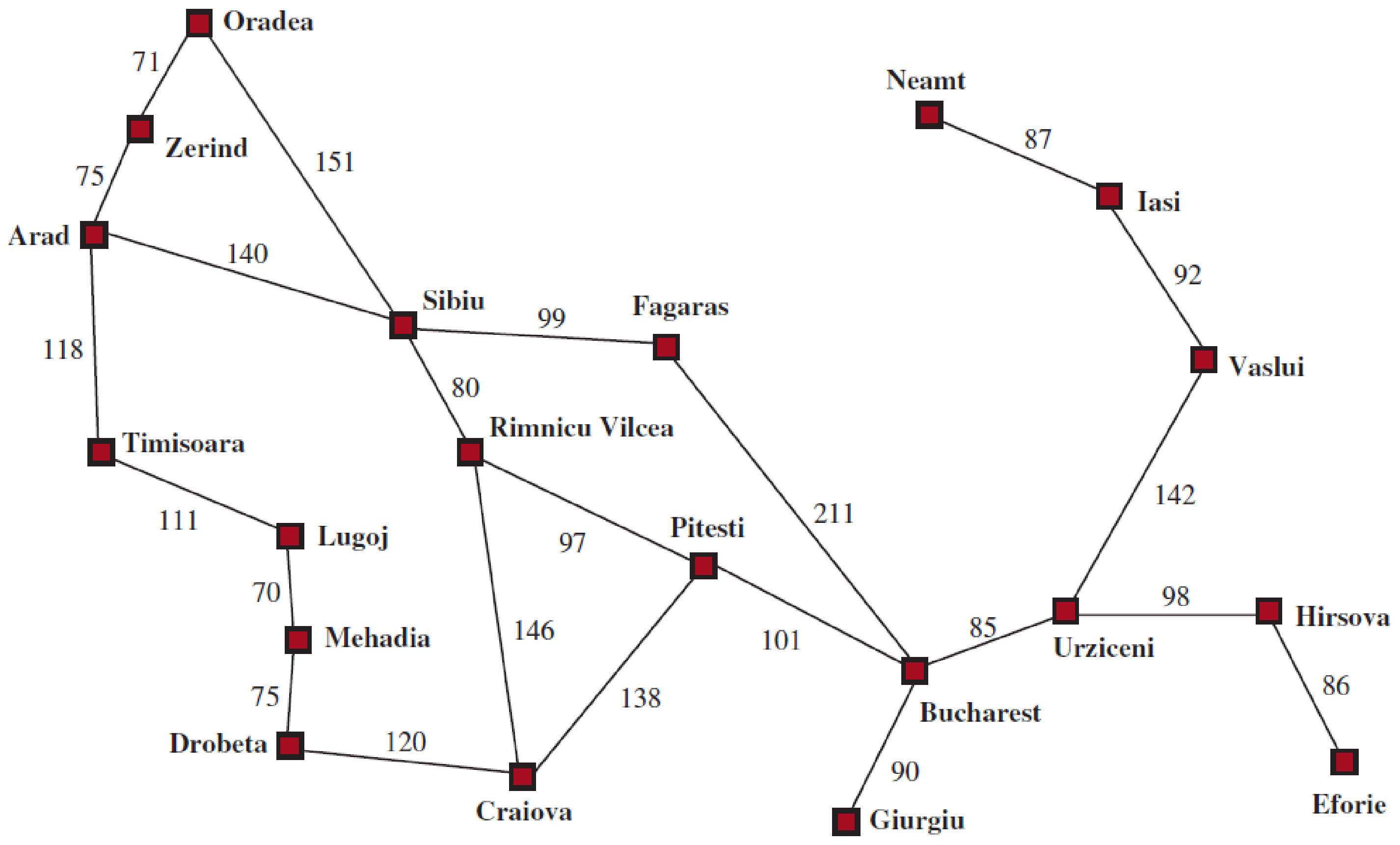
State-Space Graph

- The state space can be represented as a graph in which the vertices are states and the directed edges between them are actions.



Navigation example

- Here the map is a state-space graph.
- Each road indicates two actions, one in each direction.



Quiz

What are the essential components of a search problem?

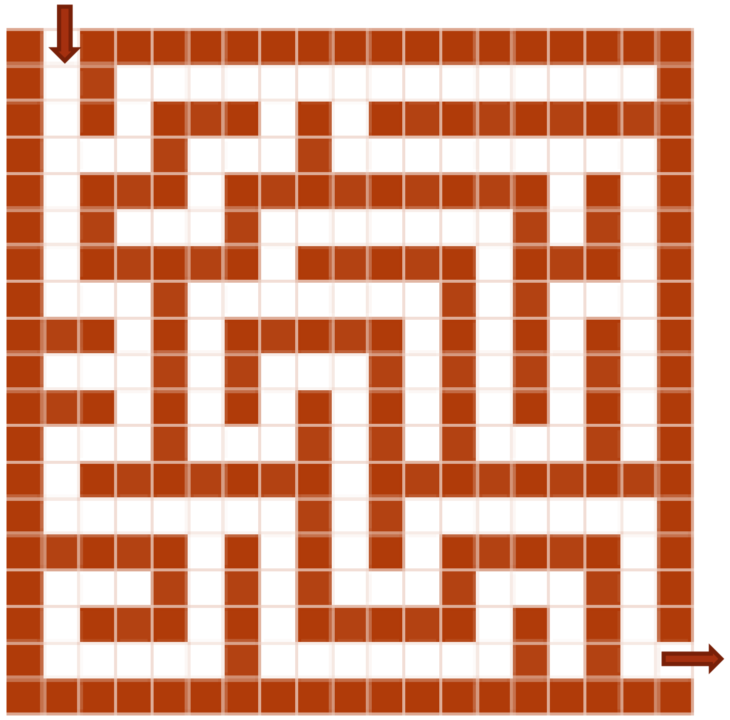
Quiz

What are the essential components of a search problem?

1. Initial state (Need not specify the state space explicitly)
2. Goal state(s) / Goal test
3. Actions
4. Transition model
5. Action cost

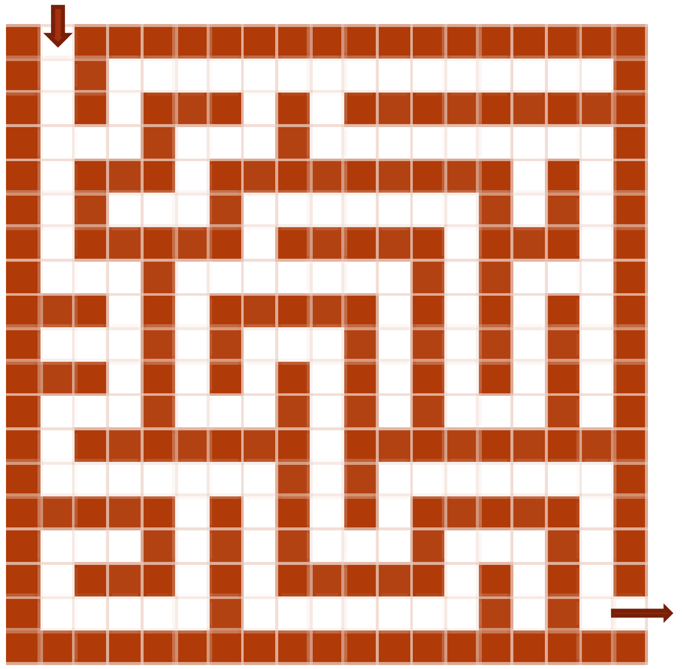
Quiz

Formulate a maze problem on a grid



Quiz

Formulate a maze problem on a grid



1. Initial State: The entrance coordinates (x, y).
2. Goal Test: Is the current state the exit coordinates?
3. Actions: GoNorth, GoSouth, GoEast, GoWest.
4. Transition Model: RESULT((x, y), GoNorth) returns (x, y+1) if there is no wall etc.
5. Path Cost: Each step costs 1.

Search Algorithms

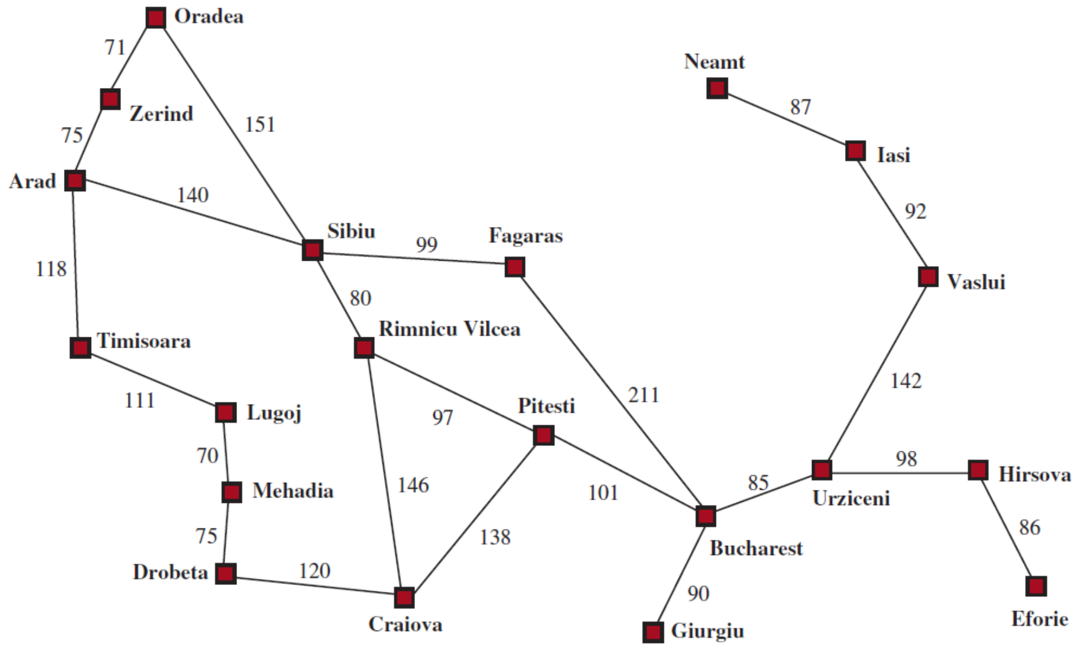
- Once a problem is formulated, the agent uses a search algorithm to find a solution
  - A search algorithm takes a search problem as input and returns a solution or indicates failure

- The search algorithms are evaluated on four criteria:
  1. Completeness: Does it always find a solution when there is one?
     - And correctly reports failure when there is none?
  2. Cost optimality: Does it always find the best or lowest cost solution?
  3. Time complexity: How long does it take to find a solution?
     - Measured in the number of nodes generated (states and actions considered).
  4. Space complexity: How much memory does it need to perform the search?

Search Tree

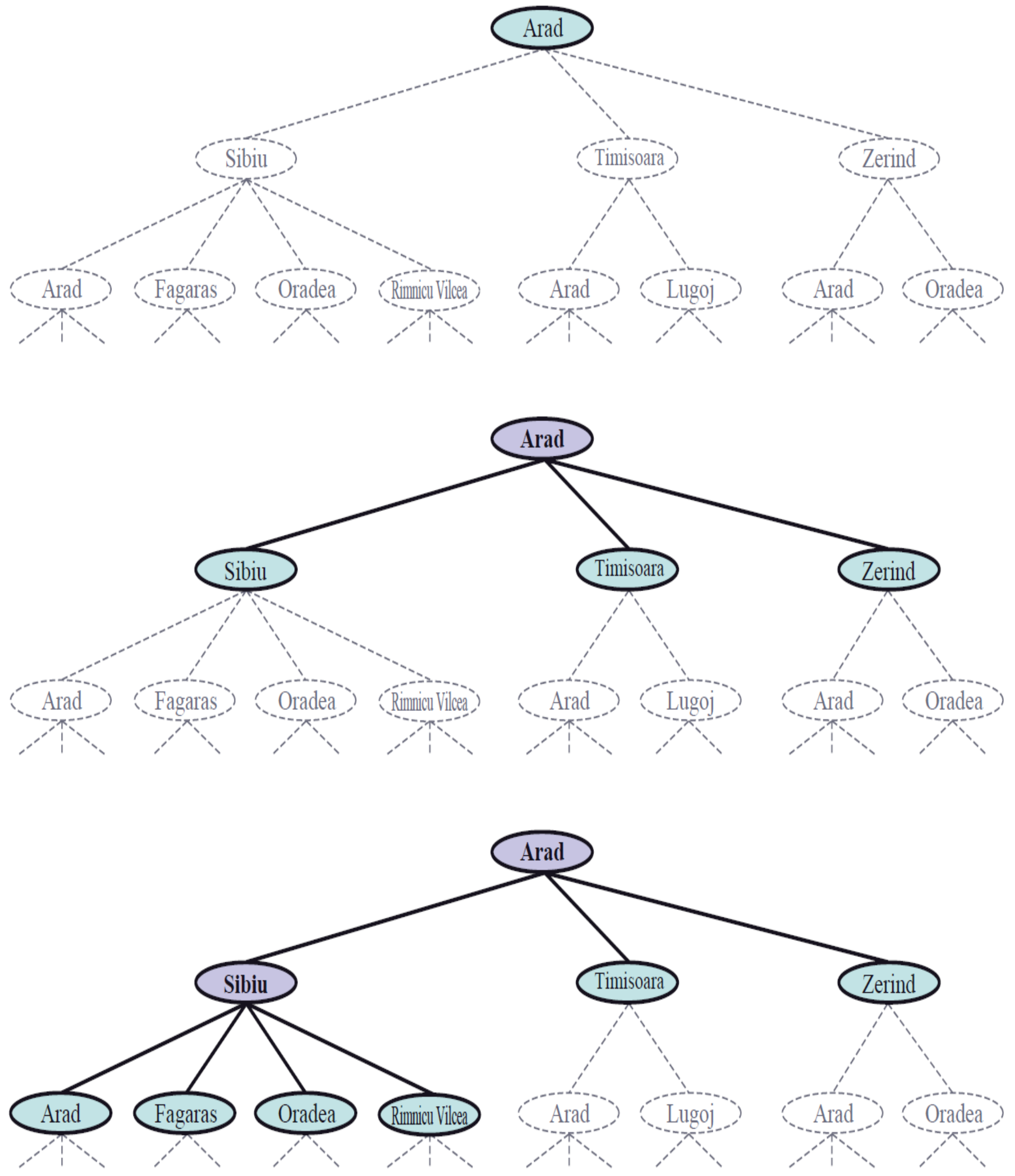
- A **search tree** is a data structure created by a search algorithm to explore a problem's state space.
- It represents the set of paths that the agent has considered, starting from the initial state which forms the root of the tree.
- Node corresponds to a state in the state space.
- Edge corresponds to an action.
- Search tree represents explored paths starting from the initial state (root) leading to the goal.
- Agent builds the search tree as it navigates the state space.
- A state may appear multiple times in the search tree.

Search Tree



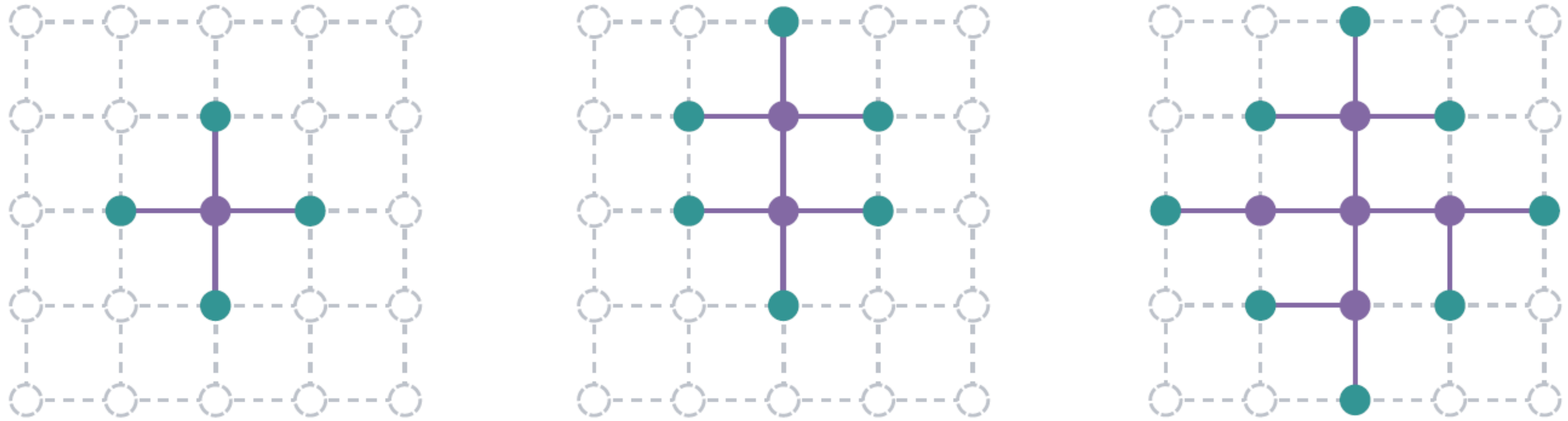
- The search tree is infinite
- State space size is only 20

Partial Search Trees



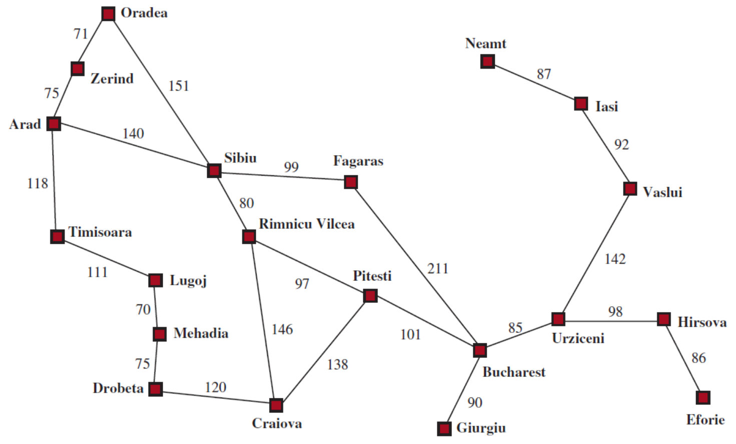
Search Tree

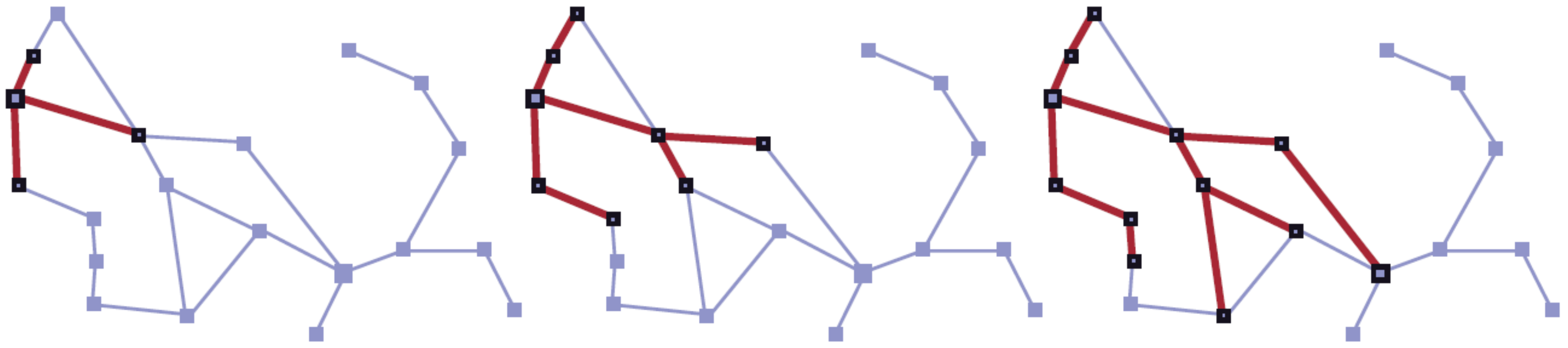
- Frontier (green) separates interior (purple) from exterior (grey)
    - A frontier node is expanded till goal is reached
- Search algorithm: which frontier node to expand next?

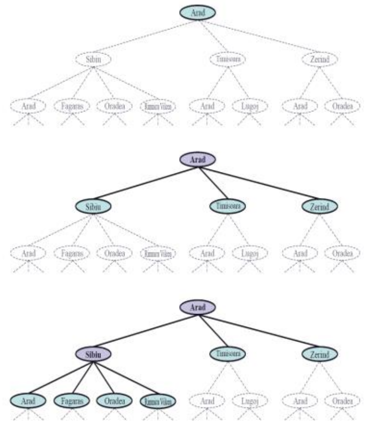


Search Tree

We will superimpose a search tree over the state-space graph, forming various paths from the initial state, trying to find a path to a goal state



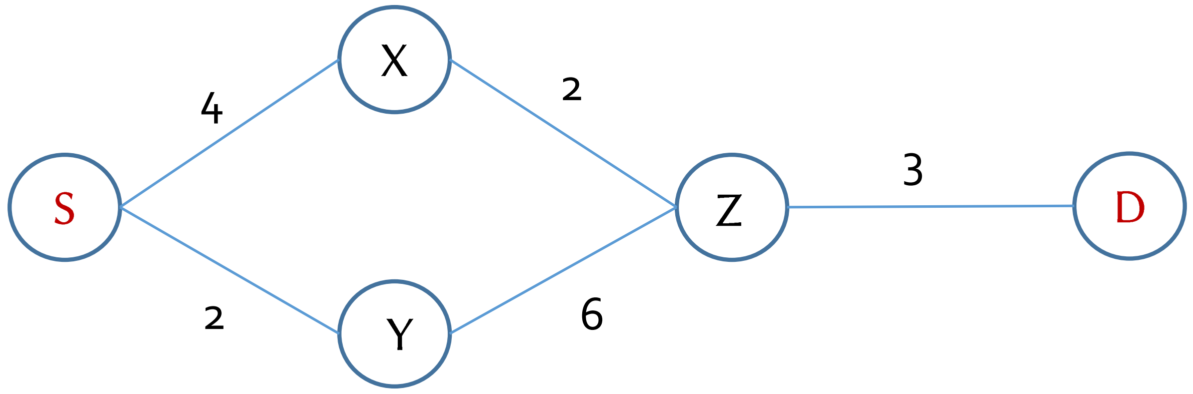




Partial Search Trees

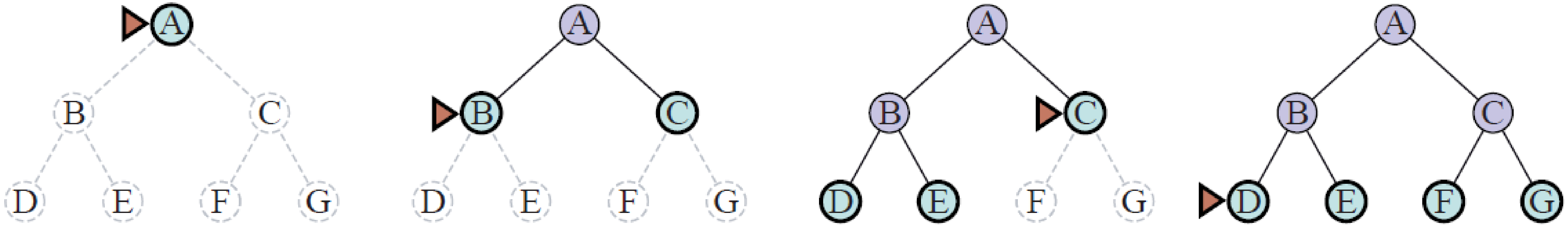
Best-First Search

- Evaluation function for each node f(n)
  - Different f(n) result in different search algorithms…
  - f(n) can change with time
- Out of all nodes in the frontier, select the node with the smallest f(n)
  - A node may be added multiple times to the frontier if it is reached by lower cost path



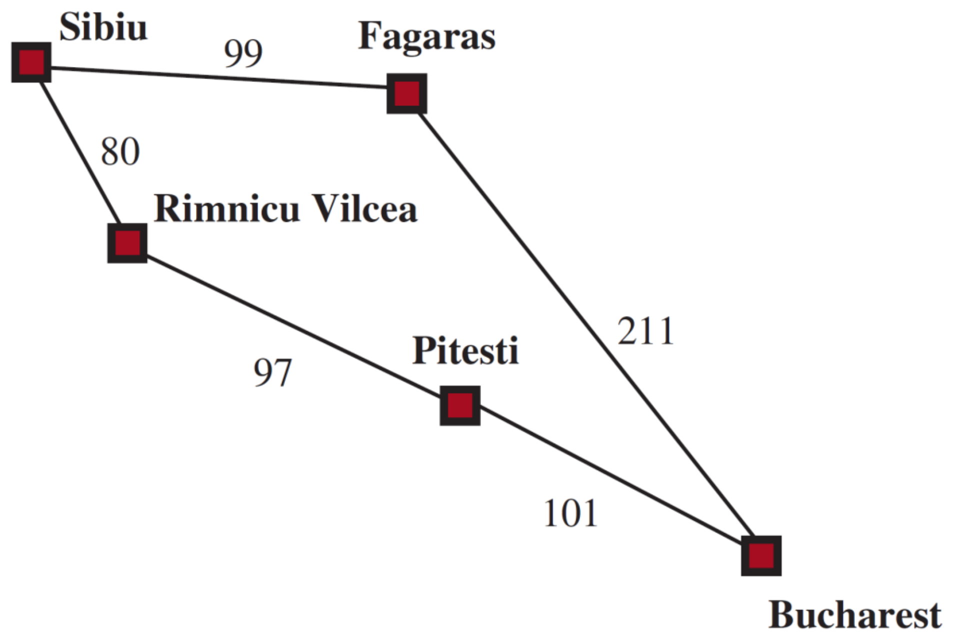
Breadth-First Search

- BFS: f(n) = depth of node n
- BFS is complete, but may not be optimal if cost varies
- Time and space complexity are $O(b^d)$, b is the branching factor and d depth.
- Not good…
  - At 1 KB per node, the memory needed to search till depth 10 and branching factor 10 is 10 TB



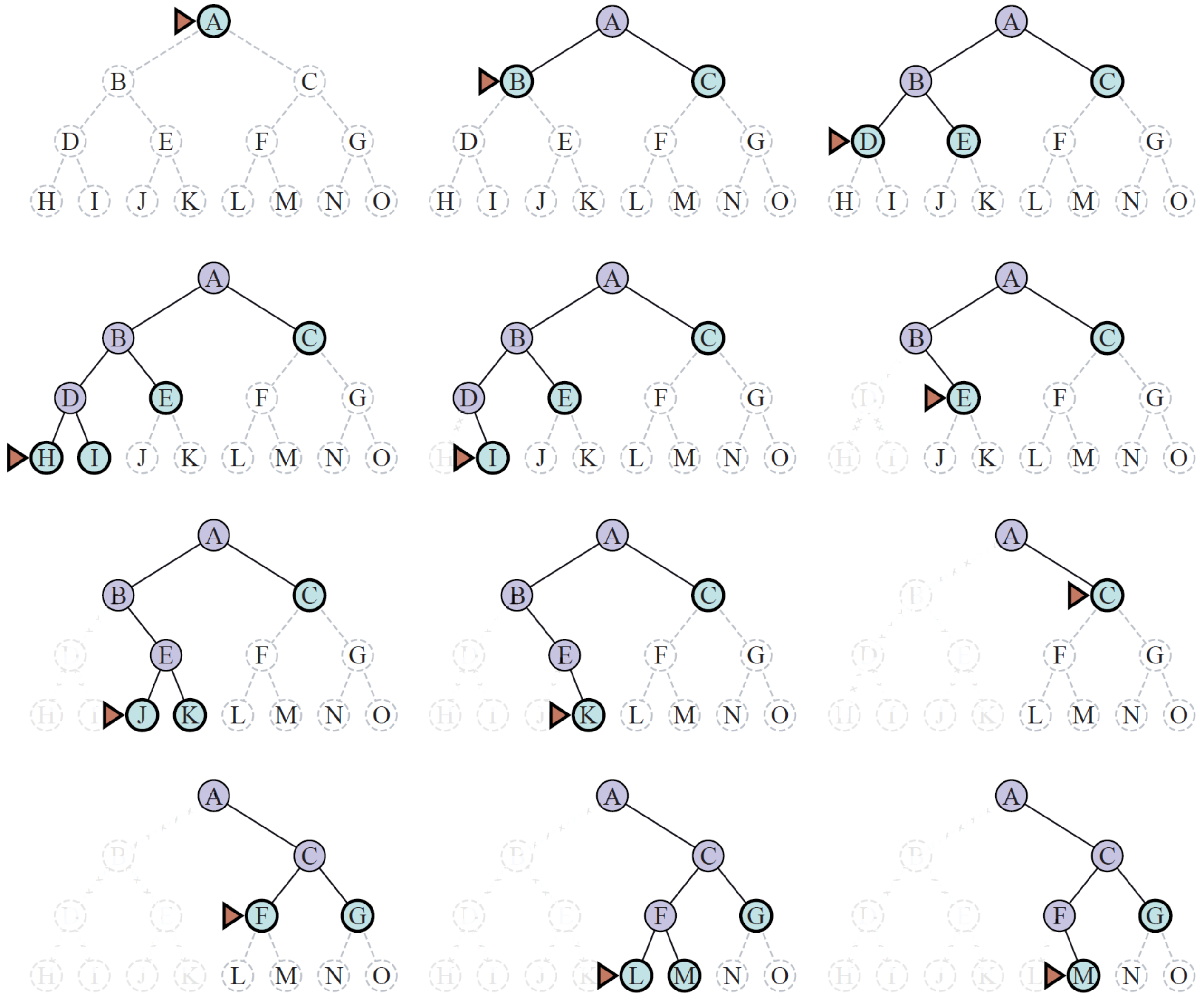
Dijkstra’s Algorithm

- Uniform-cost search
  - Expand the node with the least cost first
- Complete and optimal
- Time and space complexity: $O(b^{1 + \lfloor \frac{C^*}{\varepsilon} \rfloor})$
  - Can be worse than BFS



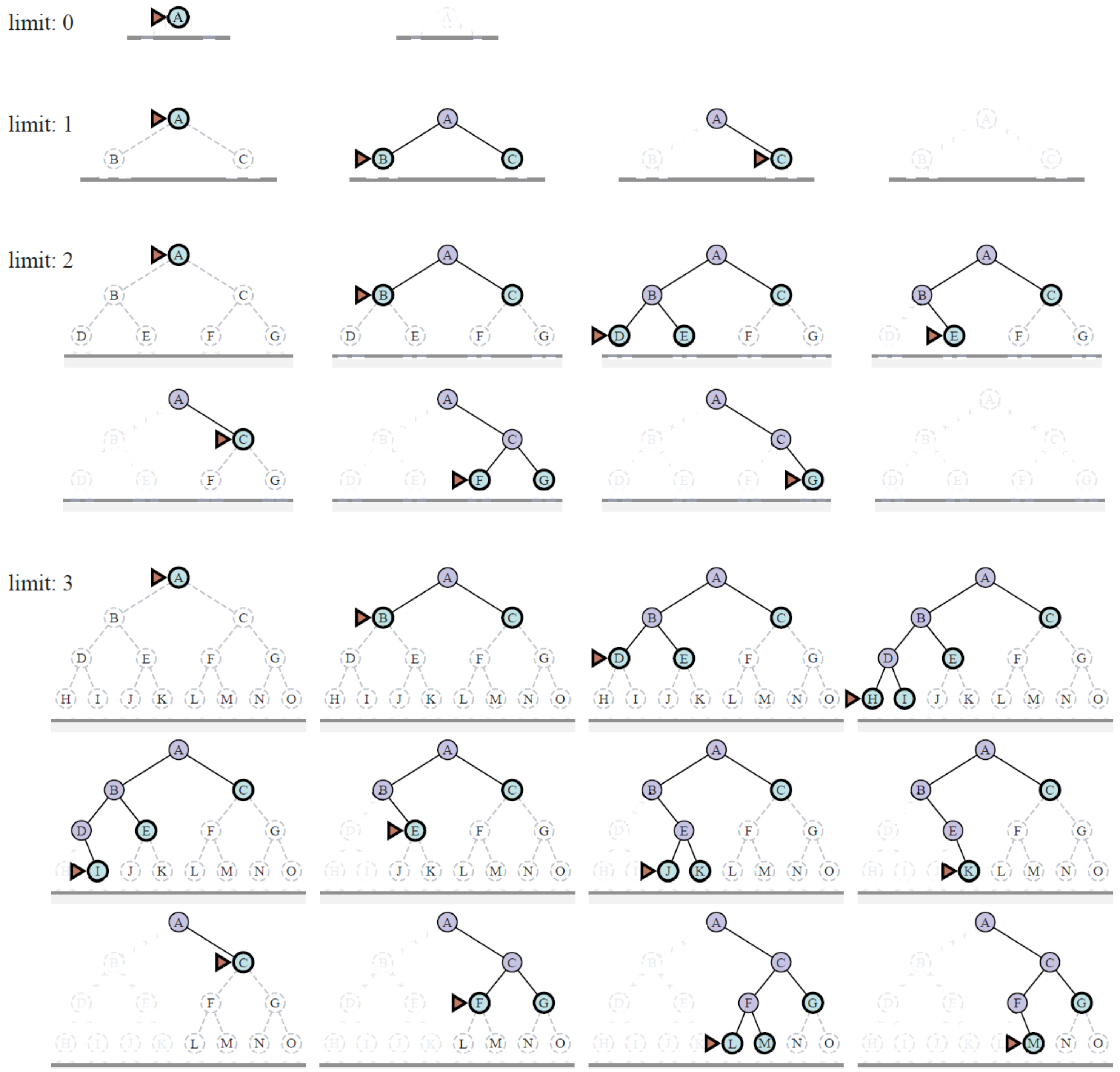
Depth-First Search

- $f(n) = -$ (depth of node n)
- Pre-order DFS
- Complete if state space is
  - Tree or DAG
  - Else incomplete
- Not optimal
- Smaller memory requirement
  - O(bm), m is the max depth
- Time complexity is $O(b^m)$



Improvements

- Depth-limited search
  - Set the maximum depth limit and do DFS
    - E.g., set depth = 19 for the Romania map navigation problem
  - Neither complete nor optimal
- Iterative deepening search
  - Set the depth limit as 0, 1, 2, 3, … and do depth-limited search
    - Most nodes are at the bottom level
  - Combines BFS and DFS
  - Memory requirements of DFS O(bd), is complete, but not optimal in general
    - optimal if action costs are all the same


limit: 0
limit: 1
limit: 2
limit: 3

Bidirectional Search

- Simultaneous search from the initial and goal states
  - Why?
    - $b^{d/2} + b^{d/2}$ vs b^d
- Can use BFS or some other search algorithm
- Keep track of two sets of frontiers and two sets of reached states
  - Opposite parent-child relationships
  - Solution when the two frontiers meet
  - If BFS: $O(b^{d/2})$ time and space complexity

Quiz

1. Which search algorithm is guaranteed to find the shallowest solution first, if one exists?
   A) Depth-First Search
   B) Uniform Cost Search
   C) Breadth-First Search
   D) Iterative Deepening

2. What is the primary drawback of Breadth-First Search?
   A) It cannot guarantee finding a goal
   B) It uses a lot of memory
   C) It always gets stuck in loops
   D) It doesn't work on graphs

3. Why does Uniform Cost Search produces optimal solution?
   A) It always visits the most promising-looking node first
   B) It avoids cycles by using a visited list
   C) It expands nodes in order of lowest path cost
   D) It only works when all edge costs are the same

Quiz

1. Which search algorithm is guaranteed to find the shallowest solution first, if one exists?
   A) Depth-First Search
   B) Uniform Cost Search
   C) Breadth-First Search
   D) Iterative Deepening
2. What is the primary drawback of Breadth-First Search?
   A) It cannot guarantee finding a goal
   B) It uses a lot of memory
   C) It always gets stuck in loops
   D) It doesn't work on graphs
3. Why does Uniform Cost Search produces optimal solution?
   A) It always visits the most promising-looking node first
   B) It avoids cycles by using a visited list
   C) It expands nodes in order of lowest path cost
   D) It only works when all edge costs are the same

Informed Search

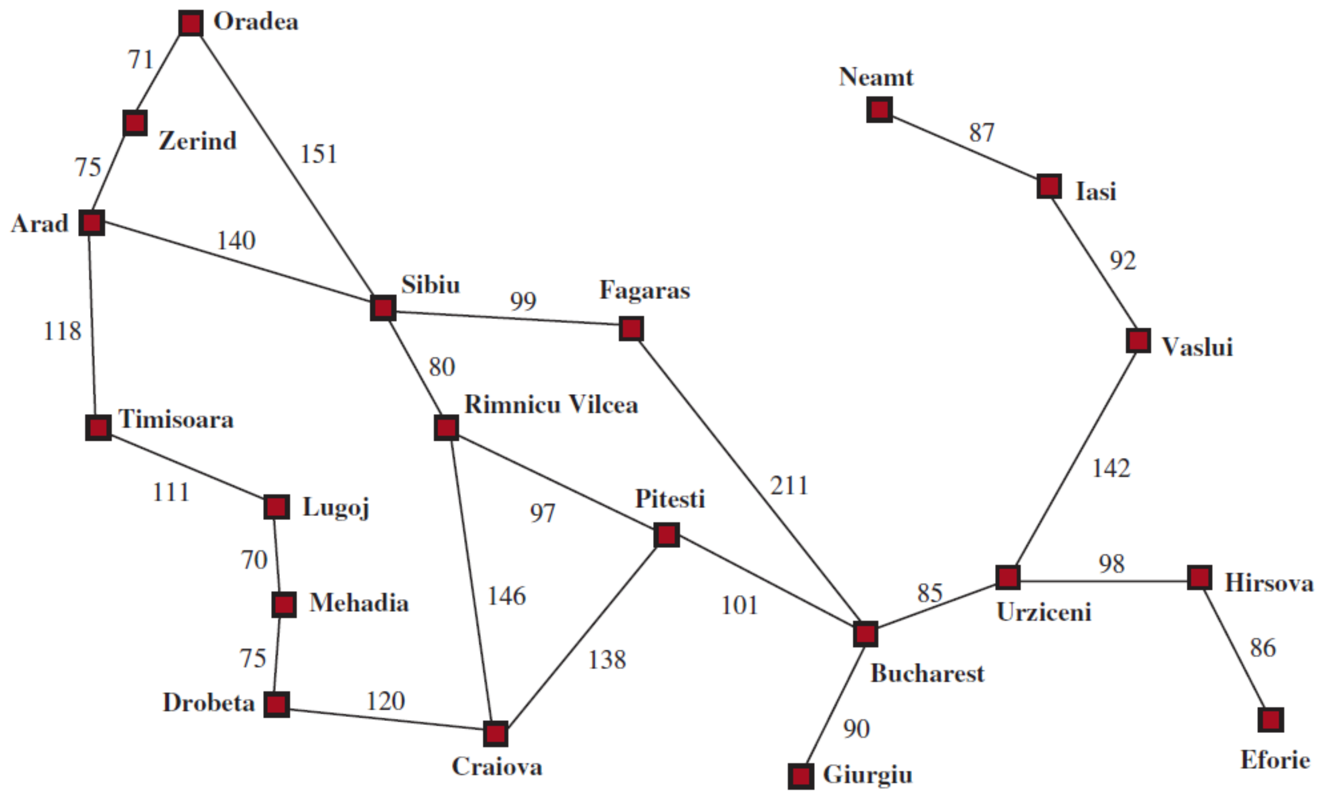
Informed Search

- Search process uses domain specific hints about goals
- Hints are given by heuristic function h(n) where
  - h(n) = estimated cost of the cheapest path from n to goal
- Study of informed search = study of heuristic functions

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

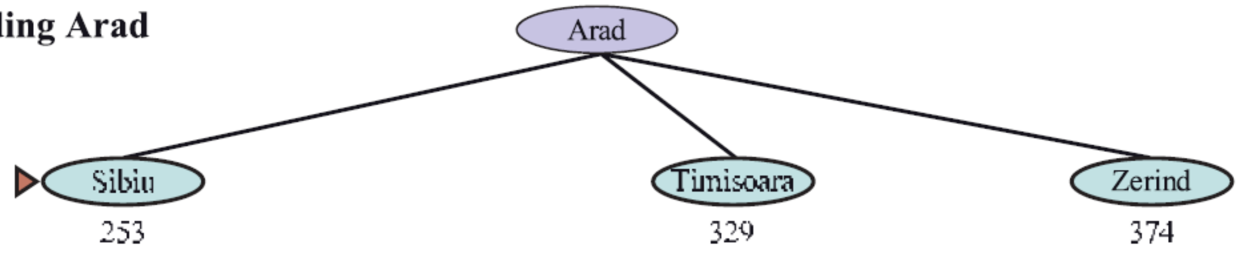
Greedy Best First Search

- Expand the node with the smallest h(n) first
- O(|V|) time and space complexity

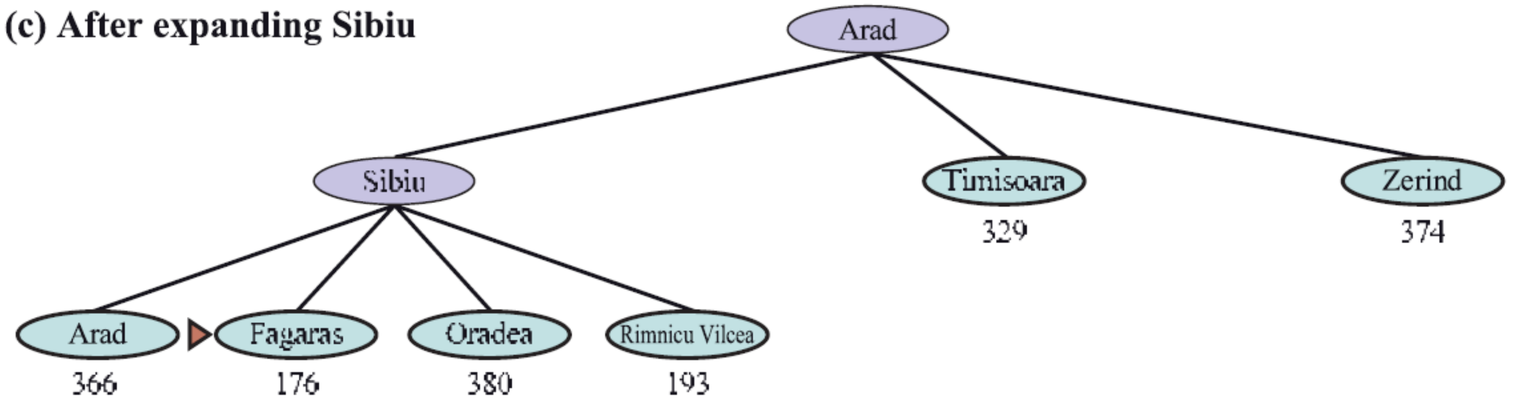


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

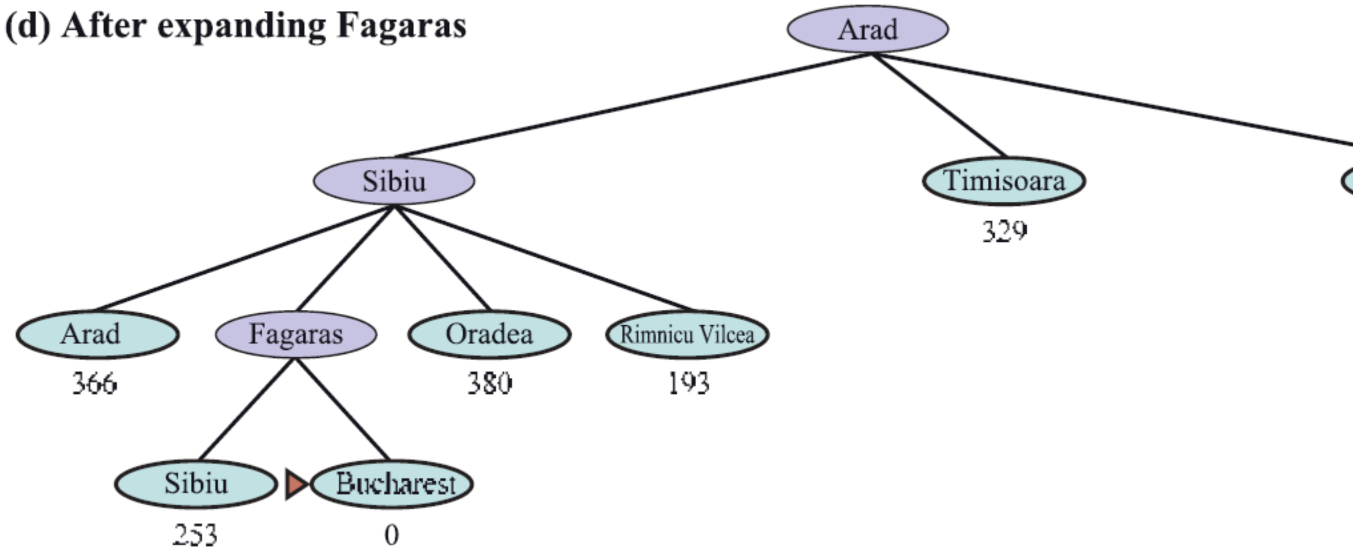
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



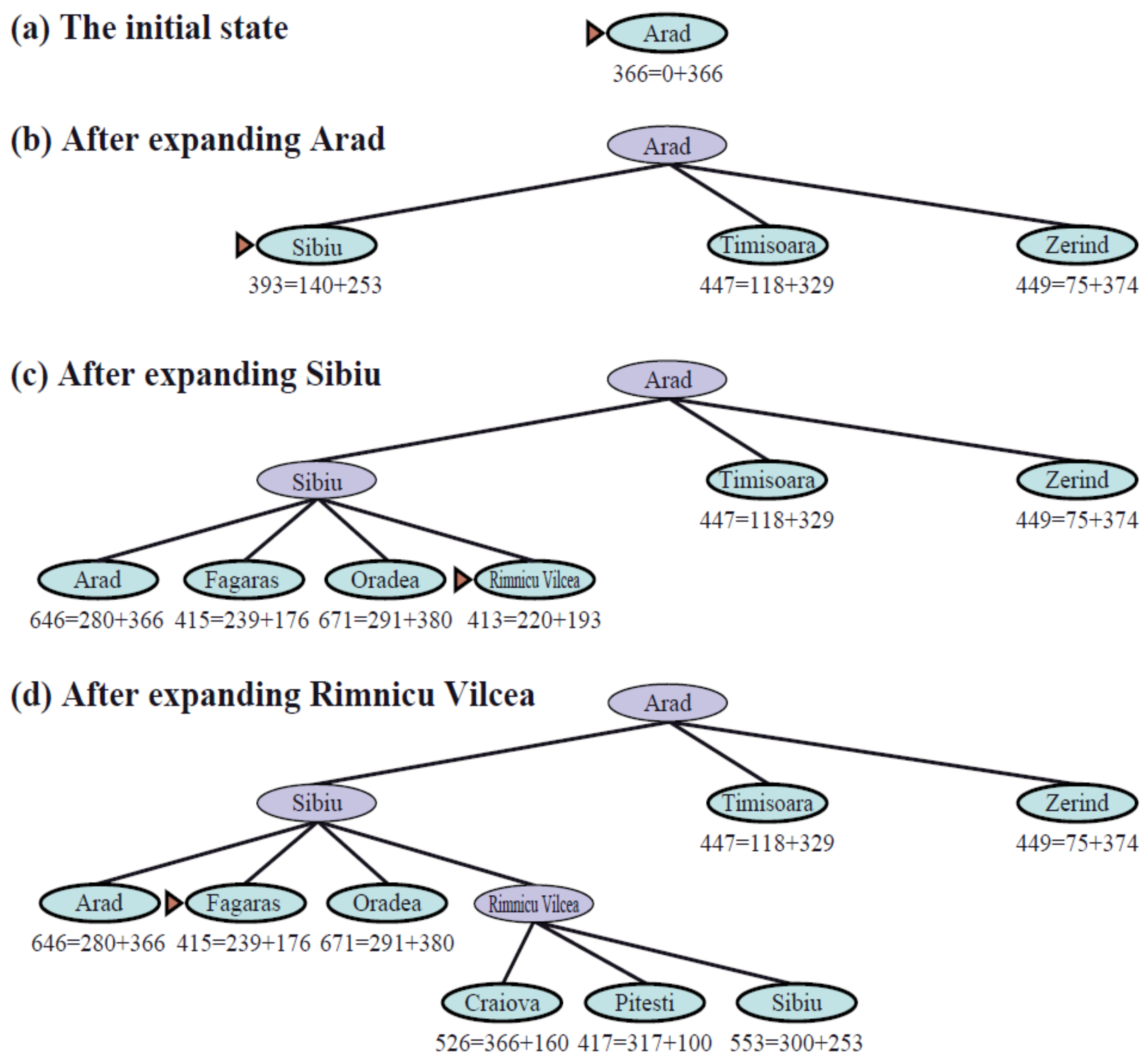
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

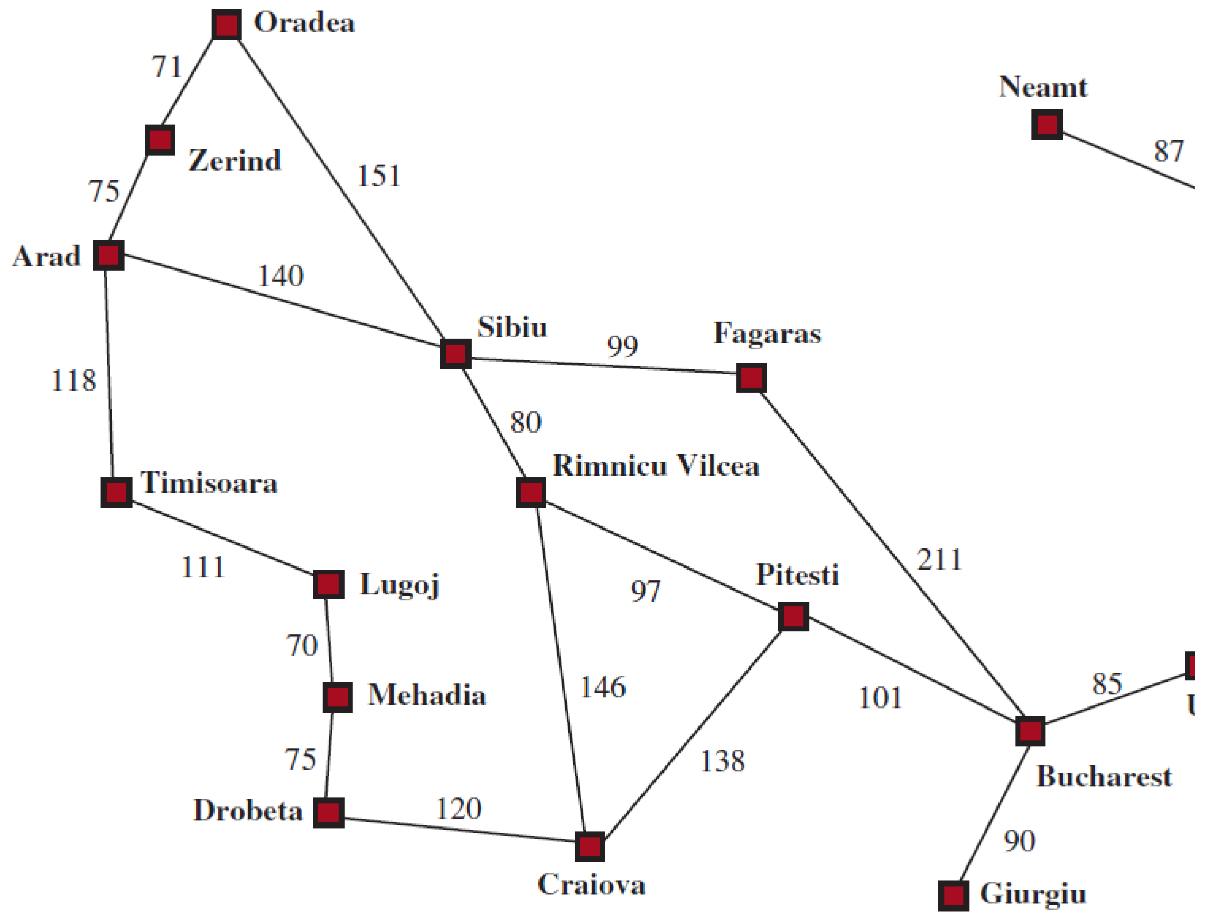
A* Search

- Numerous applications
  - h(n) = estimated cost of the cheapest path from n to goal
  - g(n) = actual cost from start state to n
  - A* uses f(n) = g(n) + h(n) as the estimated cost from start to goal via n

https://www.redblobgames.com/pathfinding/a-star/introduction.html

A* Search





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

A* Search

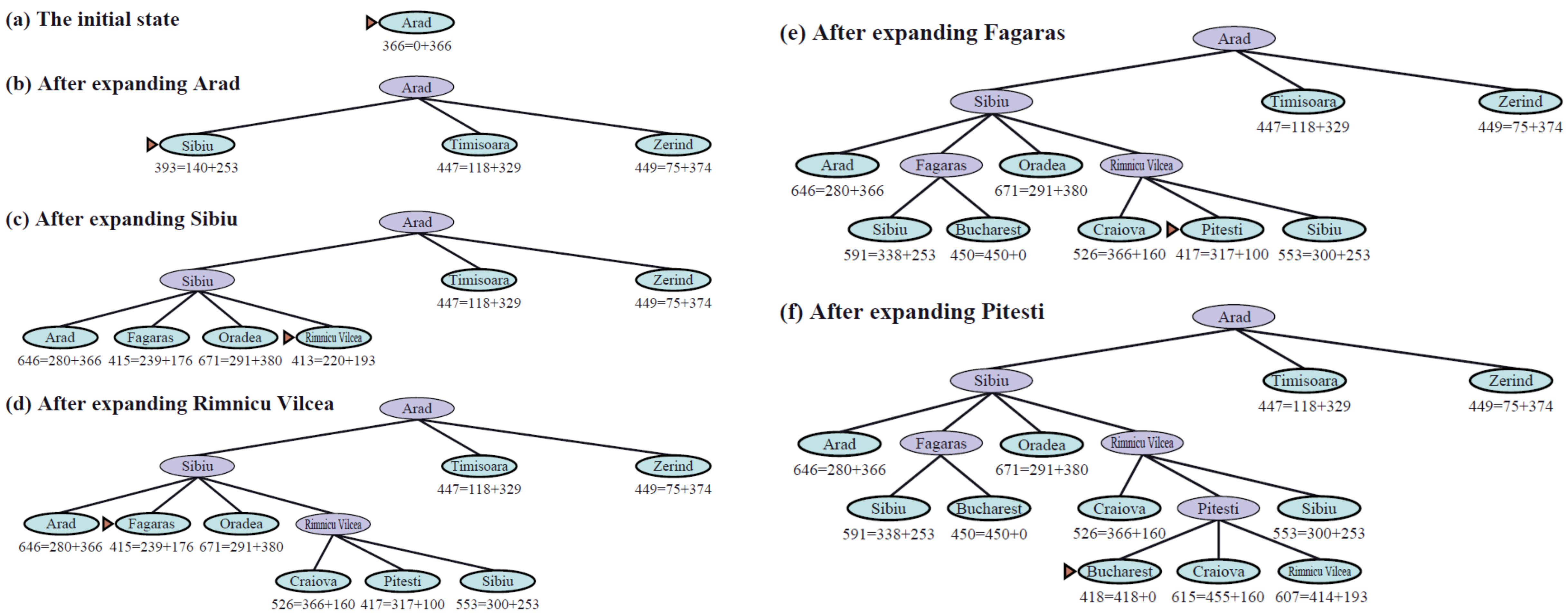
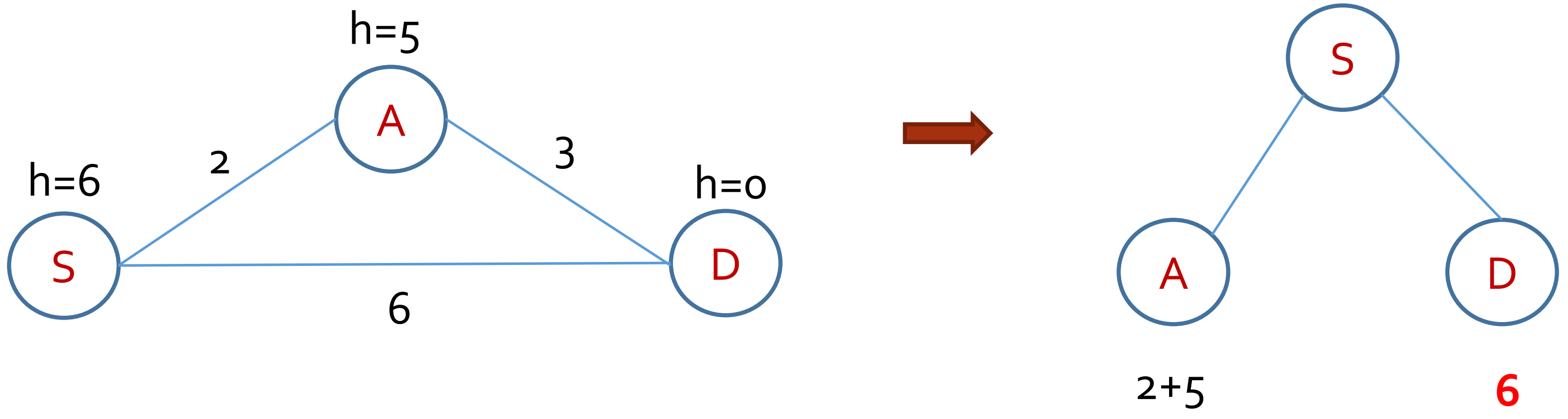


Figure 3.18 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.16.

A* Search

- Heuristic estimates must be optimistic or realistic
    - Estimates ≤ Actual costs
- A* heuristic is called admissible if it never overestimates the cost to a goal
    - 0 ≤ h(n) ≤ h(n), where h(n) is the actual cost
- A counterexample



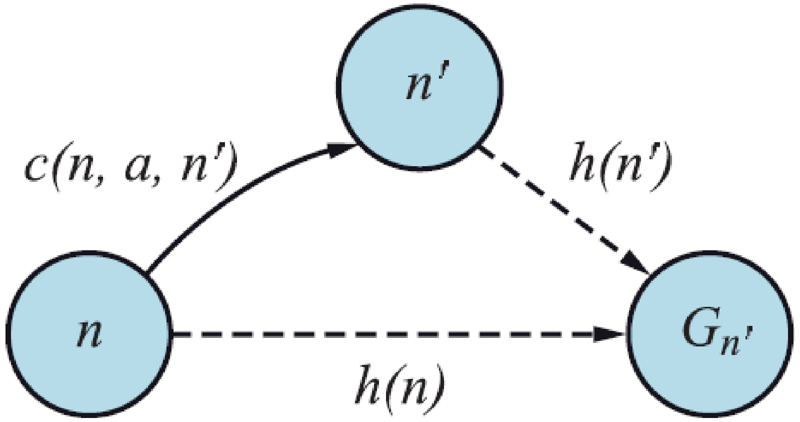
A* Search Properties

- Complete
- Optimal if heuristic is admissible
- Proof by contradiction
  - The cost of A* solution C > C where C is the optimal cost.
  - Let n be a node which is on the path to optimal solution but not in A solution. Therefore, f(n) ≥ C > C which can't be true.

$$
\begin{aligned}
f(n) &> C^* \quad \text{(otherwise } n \text{ would have been expanded)} \\
f(n) &= g(n) + h(n) \quad \text{(by definition)} \\
f(n) &= g^*(n) + h(n) \quad \text{(because } n \text{ is on an optimal path)} \\
f(n) &\leq g^*(n) + h^*(n) \quad \text{(because of admissibility, } h(n) \leq h^*(n)\text{)} \\
f(n) &\leq C^* \quad \text{(by definition, } C^* = g^*(n) + h^*(n)\text{)}
\end{aligned}
$$

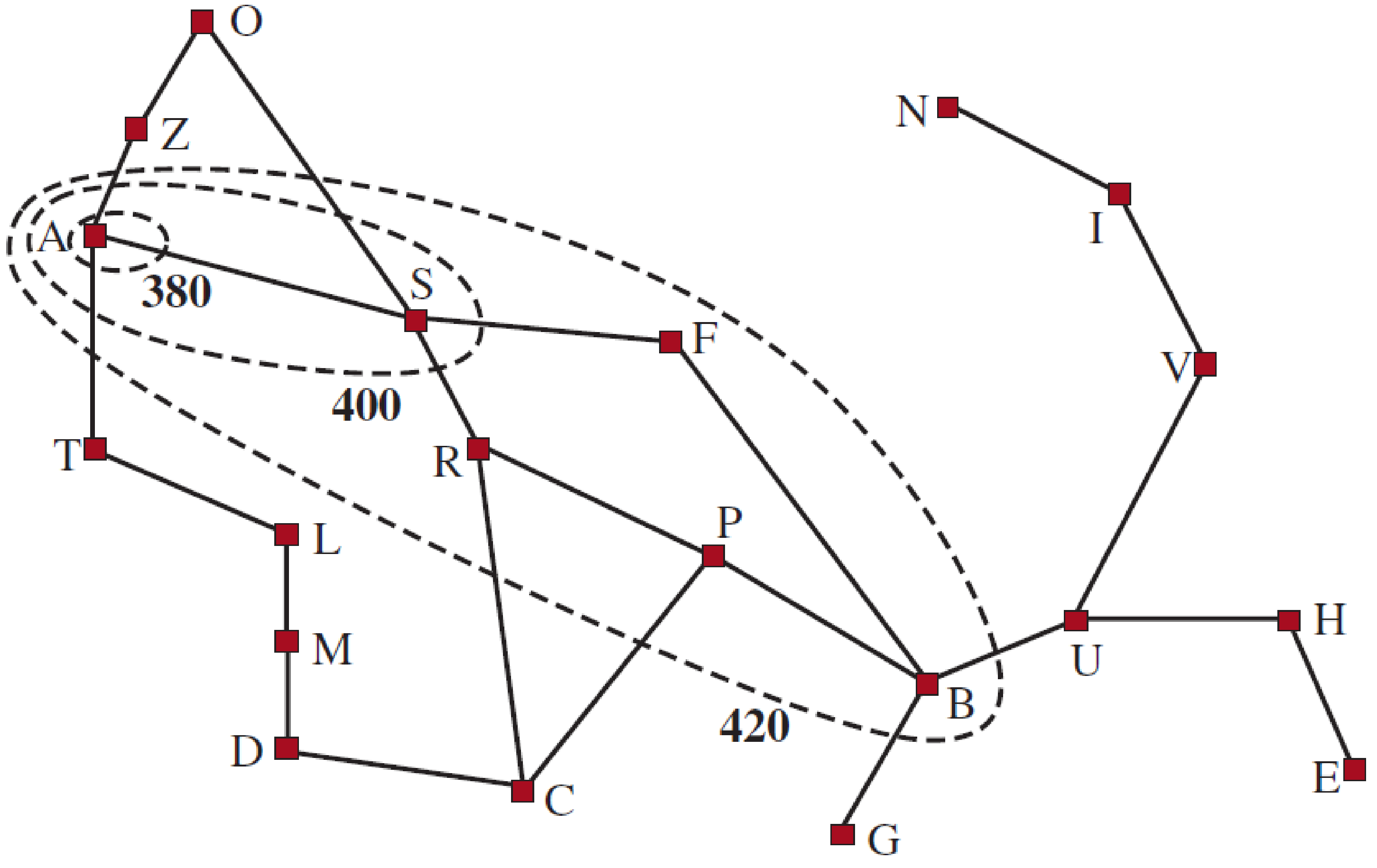
Consistent heuristic

- A heuristic is consistent if it obeys triangle inequality
    - h(n) ≤ c(n, a, n’) + h(n’)
    - Going via n’ should not reduce the cost
- Every consistent heuristic is admissible but not vice-versa
    - Stronger condition than consistency
- With a consistent heuristic, the first time we reach a state, it will be on an optimal path
    - If C is the optimal cost, A won’t expand any node with f(n) > C



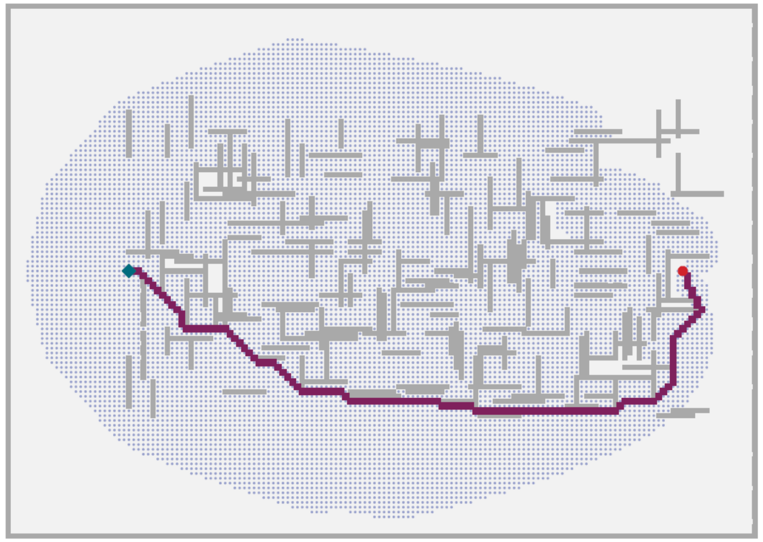
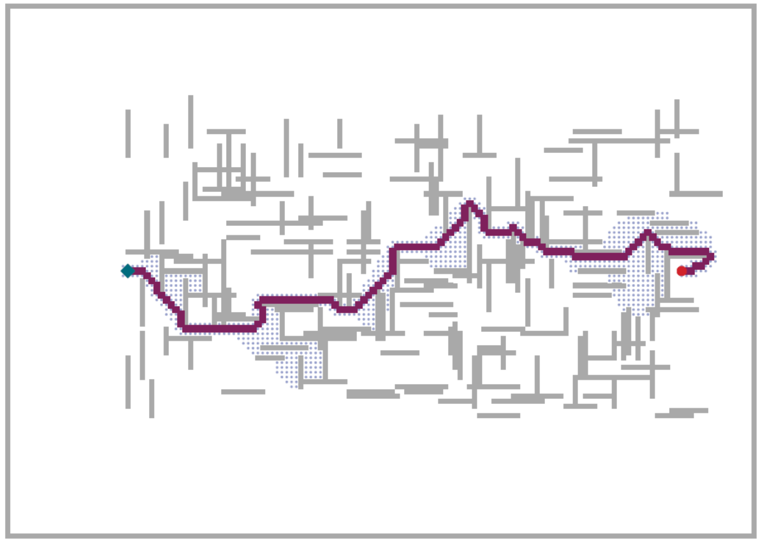
A Search Contours

- A expands lowest f-cost node at the frontier
    - Contours have bias towards the goal



Weighted A* - Satisficing Search

- A* expands too may nodes
- Satisficing: accept suboptimal but "good enough" solutions
- Detour index: multiplier to straight line distance to account for road curvatures
- Weighted A* search: $f(n) = g(n) + W \times h(n), W > 1$
  - "Somewhat greedy best-first search"

Weighted A* Search

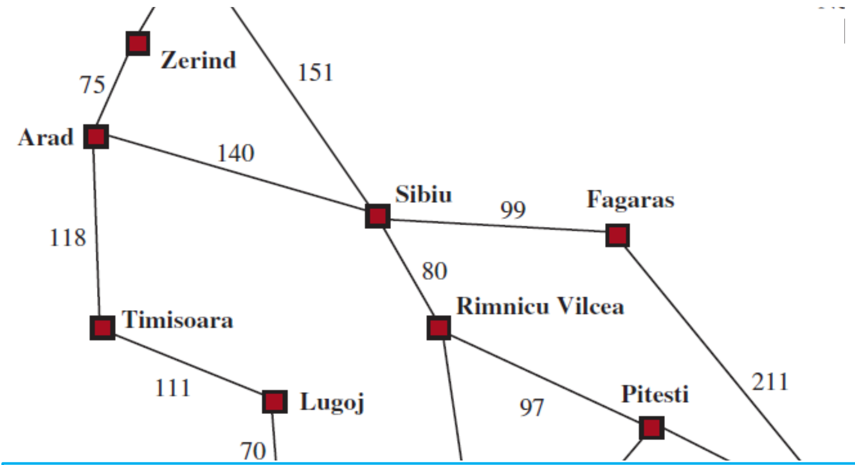
$$\begin{array}{rcl}
\text{A}^* \text{ search:} & g(n) + h(n) & (W = 1) \\
\text{Uniform-cost search:} & g(n) & (W = 0) \\
\text{Greedy best-first search:} & h(n) & (W = \infty) \\
\text{Weighted A}^* \text{ search:} & g(n) + W \times h(n) & (1 < W < \infty)
\end{array}$$

Improvements to A* Search

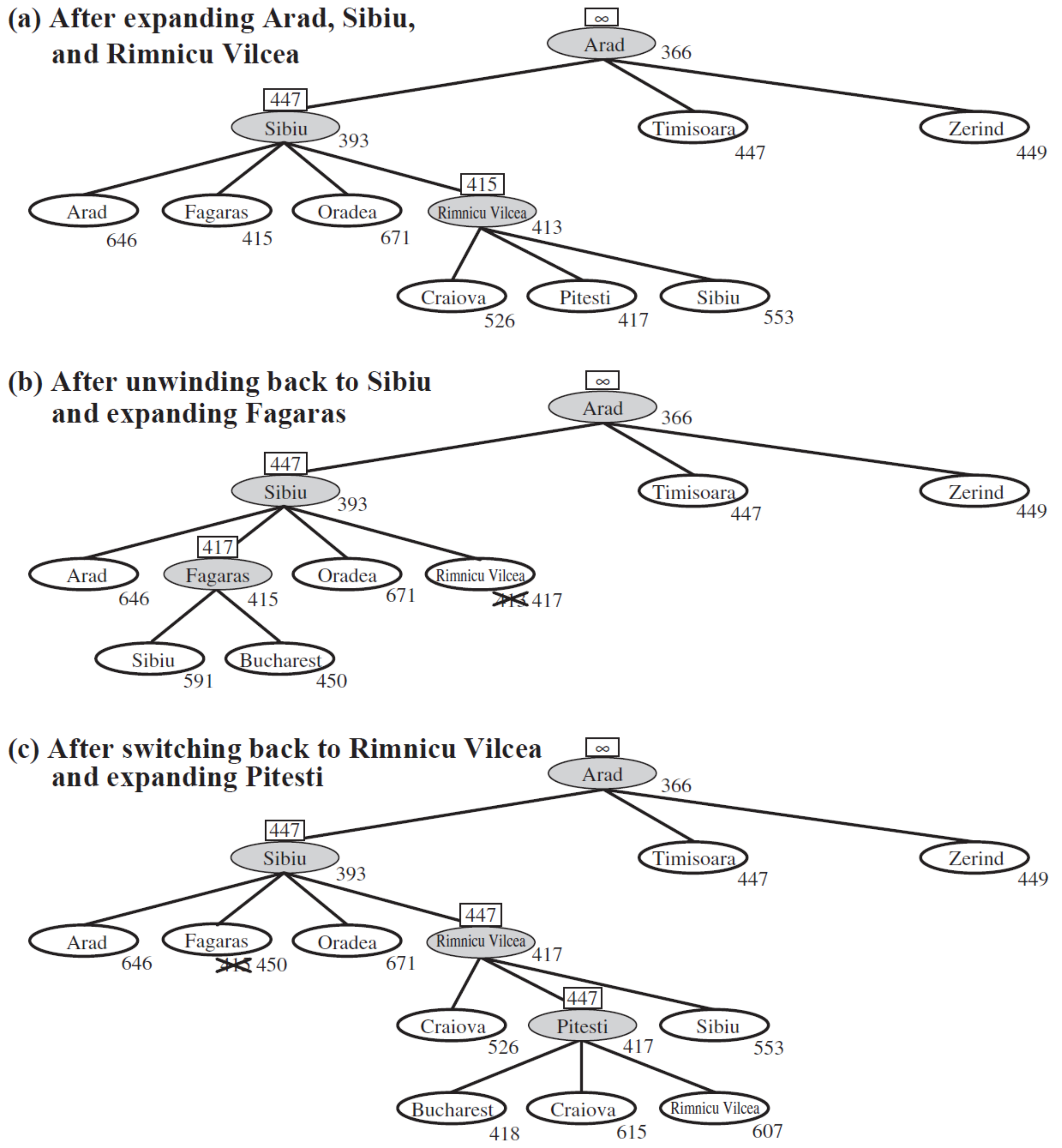
- A* is memory hungry
- Iterative deepening A* search (IDA*)
  - Cutoff is f-cost (g+h) instead of depth
  - Increase the cutoff by the smallest f-cost of the node beyond the search contour
    - Number of iterations is bounded by C if f-cost is an integer
- Recursive best-first search (RBFS)
  - f-limit keeps track of the f-value of the best alternative path from any ancestor of the current node
    - If the recursion exceeds this limit, the search unwinds

RBFS

- Frequent switches
  - Increases near the goal
- O(d) space complexity

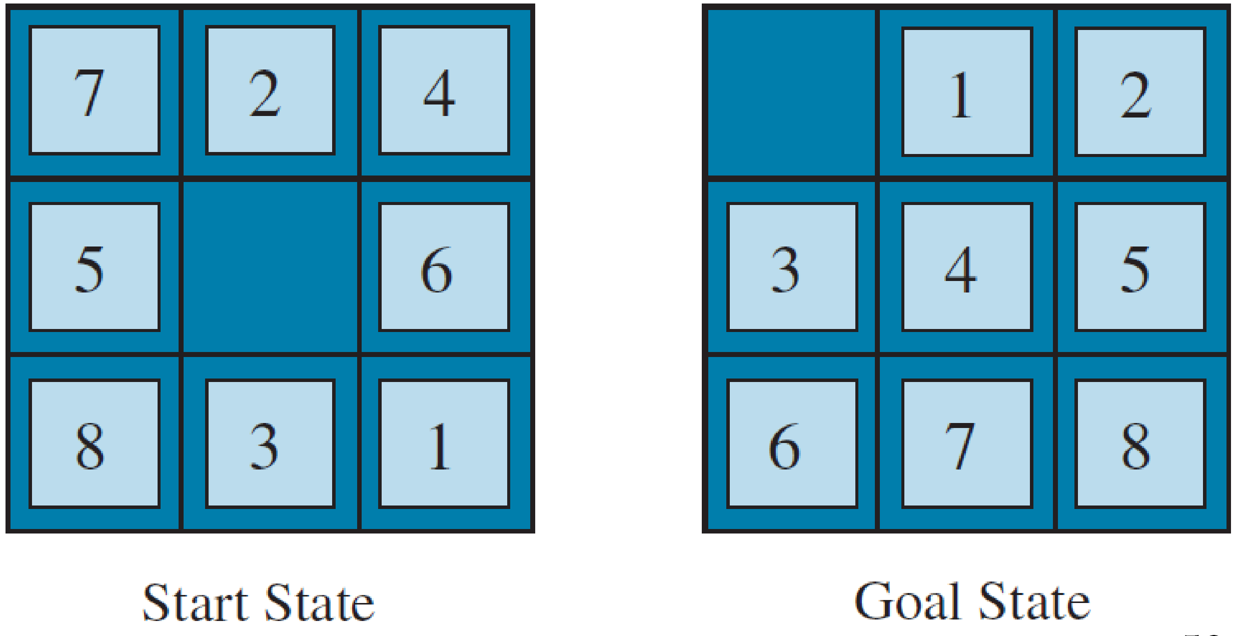


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Creating admissible heuristic

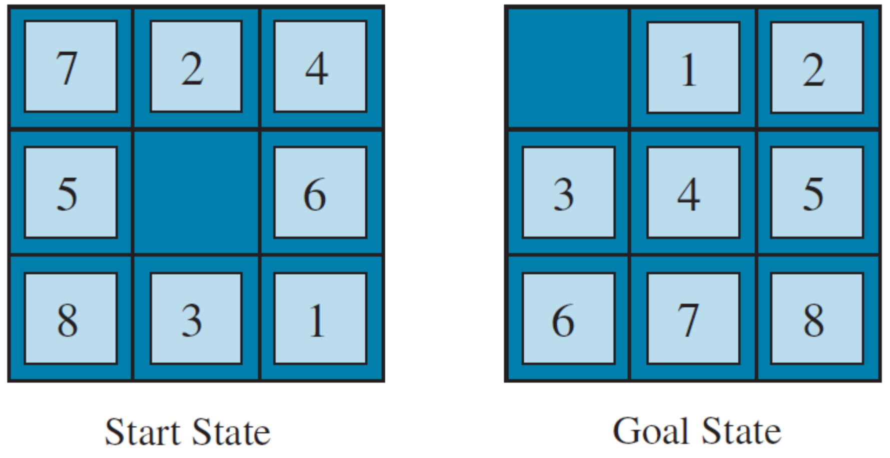
- Much of the hard work
- Solve a relaxed version of the problem, use pattern databases, use precomputed landmark solutions, learn (what to look for)
- Example: 8-Puzzle
  - 9!/2 = 181,400 reachable states
  - Good heuristics?



Creating good heuristic

- Heuristic choices for 8-Puzzle
  - H_1: No. of misplaced tiles
  - H_2: Sum of Manhattan distances to the correct position
  - Here H_2 dominates H_1, i.e., $H_2 \geq H_1$
    - A with H_1 will expand all the nodes that A with H_2 does and possibly some more

- The effect of using a heuristic in A* search is a reduced effective depth of the search compared to that of the uniform search (Korf & Reid, 1998)
  - $O(b^{d-k})$ vs $O(b^d)$



$H_1 = 8$, $H_2 = 18$

Dominating heuristic is more efficient

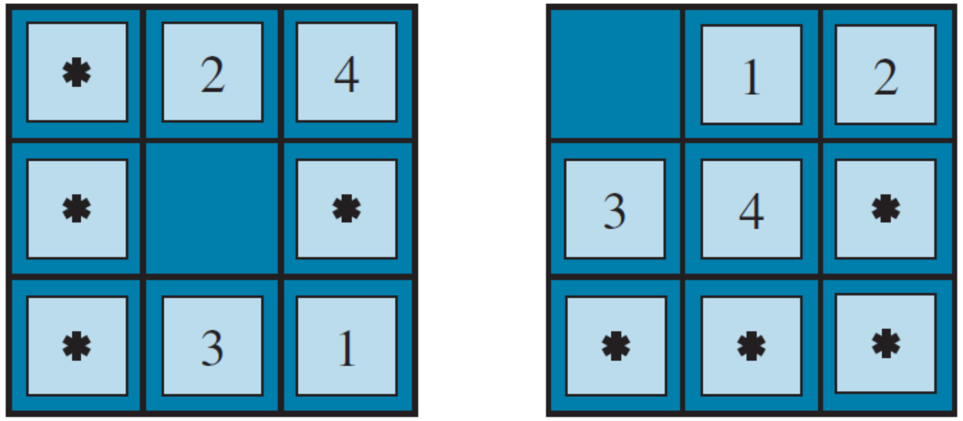
	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	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

Generate heuristic from relaxed problems

- The state-space graph of the relaxed problem is a supergraph of the original problem state-space graph
  - Relaxation results in extra edges added to the graph
- The cost of an admissible solution to a relaxed problem becomes less.
  - Hence, the solution of relaxed problem is an admissible heuristic to the original problem
- Heuristic cost needs to be generated fast
- Generating heuristic costs can be automated
  - Absolver (Prieditis, 1993) generated heuristic was better than known ones for 8-Puzzle and could generate for Rubik's cube
- Can combine admissible heuristics: $h(n) = \max (h_1(n), \ldots, h_k(n))$

Generate heuristic from subproblems

- Cost of the optimal solution of a subproblem is a lower bound on the cost of the complete problem
- Store the exact solution cost of every subproblem in a pattern database
    - Example: pattern for 1-2-3-4
    - Can combine the heuristic cost for multiple patterns (take max)
    - More accurate than Manhattan distance
    - Large speedups in practice



Start State Goal State

Quiz

1. What evaluation function does A* use to choose which node to expand next?
   A. $f(n) = g(n)$
   B. $f(n) = h(n)$
   C. $f(n) = g(n) + h(n)$
   D. $f(n) = \max(g(n), h(n))$
2. What property must a heuristic satisfy to guarantee A* finds an optimal path?
   A. Heuristic must be admissible
   B. Heuristic must be consistent
   C. Heuristic must be admissible and consistent
   D. Heuristic must be consistent but not admissible
3. What is an admissible heuristic?
   A. One that always overestimates the cost to the goal
   B. One that never overestimates the cost to the goal
   C. One that equals the exact cost of the goal
   D. One that always satisfies triangle inequality

Quiz

1. What evaluation function does A* use to choose which node to expand next?
   A. $f(n) = g(n)$
   B. $f(n) = h(n)$
   C. $f(n) = g(n) + h(n)$
   D. $f(n) = \max(g(n), h(n))$
2. What property must a heuristic satisfy to guarantee A* finds an optimal path?
   A. Heuristic must be admissible
   B. Heuristic must be consistent
   C. Heuristic must be admissible and consistent
   D. Heuristic must be consistent but not admissible
3. What is an admissible heuristic?
   A. One that always overestimates the cost to the goal
   B. One that never overestimates the cost to the goal
   C. One that equals the exact cost of the goal
   D. One that always satisfies triangle inequality

Quiz

4. Which statement about IDA* is correct?
   A. IDA* uses breadth-first search
   B. IDA* uses simple depth cut-offs only
   C. IDA* uses an $f(n) = g(n) + h(n)$ threshold in iterative deepening
   D. IDA* stores all visited nodes in memory

5. Why is IDA* more memory-efficient than A*?
   A. It only stores the heuristic values
   B. It uses breadth-first search
   C. It only stores nodes along the current path
   D. It compresses the search tree

Quiz

4. Which statement about IDA* is correct?
   A. IDA* uses breadth-first search
   B. IDA* uses simple depth cut-offs only
   C. IDA* uses an f(n) = g(n) + h(n) threshold in iterative deepening
   D. IDA* stores all visited nodes in memory

5. Why is IDA* more memory-efficient than A*?
   A. It only stores the heuristic values
   B. It uses breadth-first search
   C. It only stores nodes along the current path
   D. It compresses the search tree

Summary

- Problem-solving agents determine sequences of actions in a search problem defined on atomic states leading to goal states
- A search problem has five components: initial state, goal test, action, transition model, and path cost function
- Search methods are evaluated based on completeness, time complexity, space complexity, and optimality
- Uninformed search methods
    - Breadth-First Search (BFS), Uniform-Cost Search (UCS), Depth-First Search (DFS), Depth-Limited Search, Iterative Deepening Search (IDS), and Bidirectional Search

Summary

- Informed search methods use heuristic function to estimate cost to the goal
- Greedy Best-First Search uses f(n) = h(n)
  - Its search is biased towards the goal but lacks optimality
- **A^* Search** uses actual past cost g(n) with estimated cost h(n)
  - Uses **f(n) = g(n) + h(n)** to decide which node to expand next
  - With an admissible or consistent heuristic, A^* is both complete and optimal
  - The efficiency of A^* is optimal — no other algorithm is guaranteed to expand fewer nodes
- Weighted A^* search
  - $f(n) = g(n) + W \times h(n)$, $W > 1$ (W is a measure of suboptimality)

- Reading: Chapter 3
- Assignments: PS 2, problem solving agent programming exercise

- Next: CSP, Chapter 6