Module3

Dr. Thyagaraju G S

Contents

2. 1 Informed Search Strategies

- **Informed Search:** Informed search is a search strategy that utilizes problem-specific knowledge, to find solutions more efficiently. Informed search methods make use of heuristics and evaluation functions to guide the search towards more promising paths.

- **Heuristic Function(h(n))**: It is a heuristic function that provides an estimate of the cost from the current node to the goal node. This heuristic is admissible if it never overestimates the true cost to reach the goal. In other words, **h(n)** is always less than or equal to the actual cost.

- **Actual cost function(g(n)): It is** Cost of the path from the **start node to node n**.It represents the actual cost incurred to reach the current node from the initial node. For the initial node (start node), **g(n)** is usually 0.

- **Evaluation Function (f(n))**: The evaluation function, denoted as **f(n),** is the total estimated cost of the cheapest path from the start node to the goal node that passes through node n. It is the sum of g(n) and h(n): **f(n) = g(n) + h(n).**

2.1.a Greedy Best First Search Algorithm

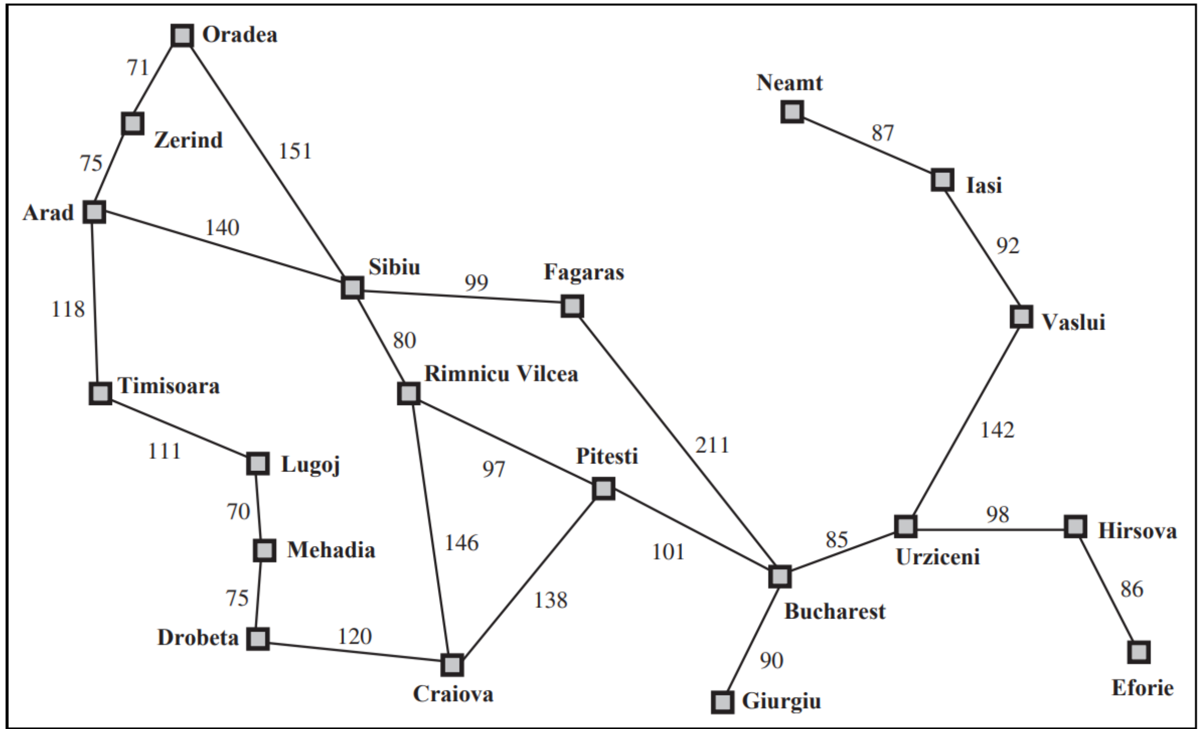


Figure 3.2 A simplified road map of part of Romania.

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

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

Figure: Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic hSLD . Nodes are labeled with their h-values.

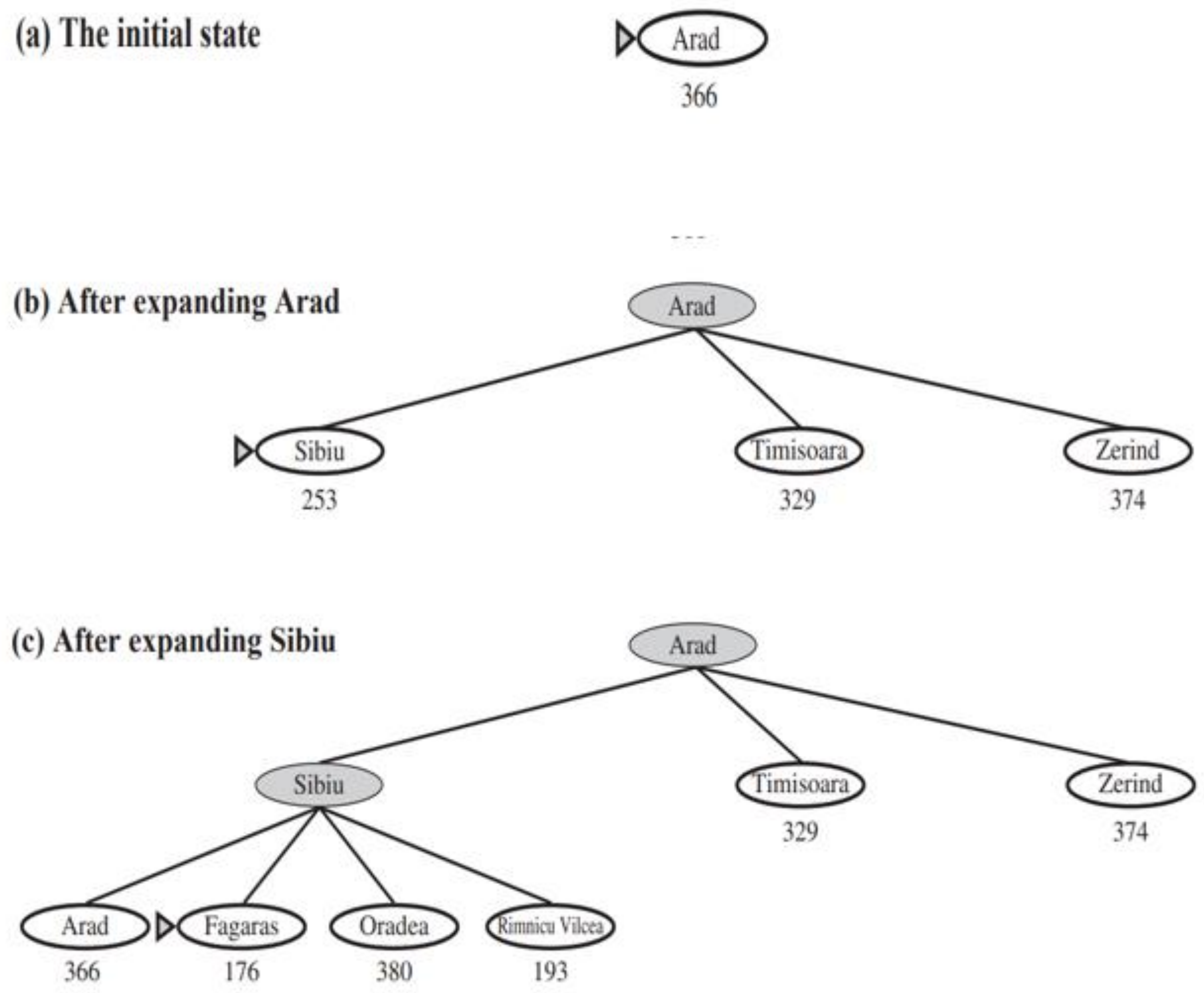
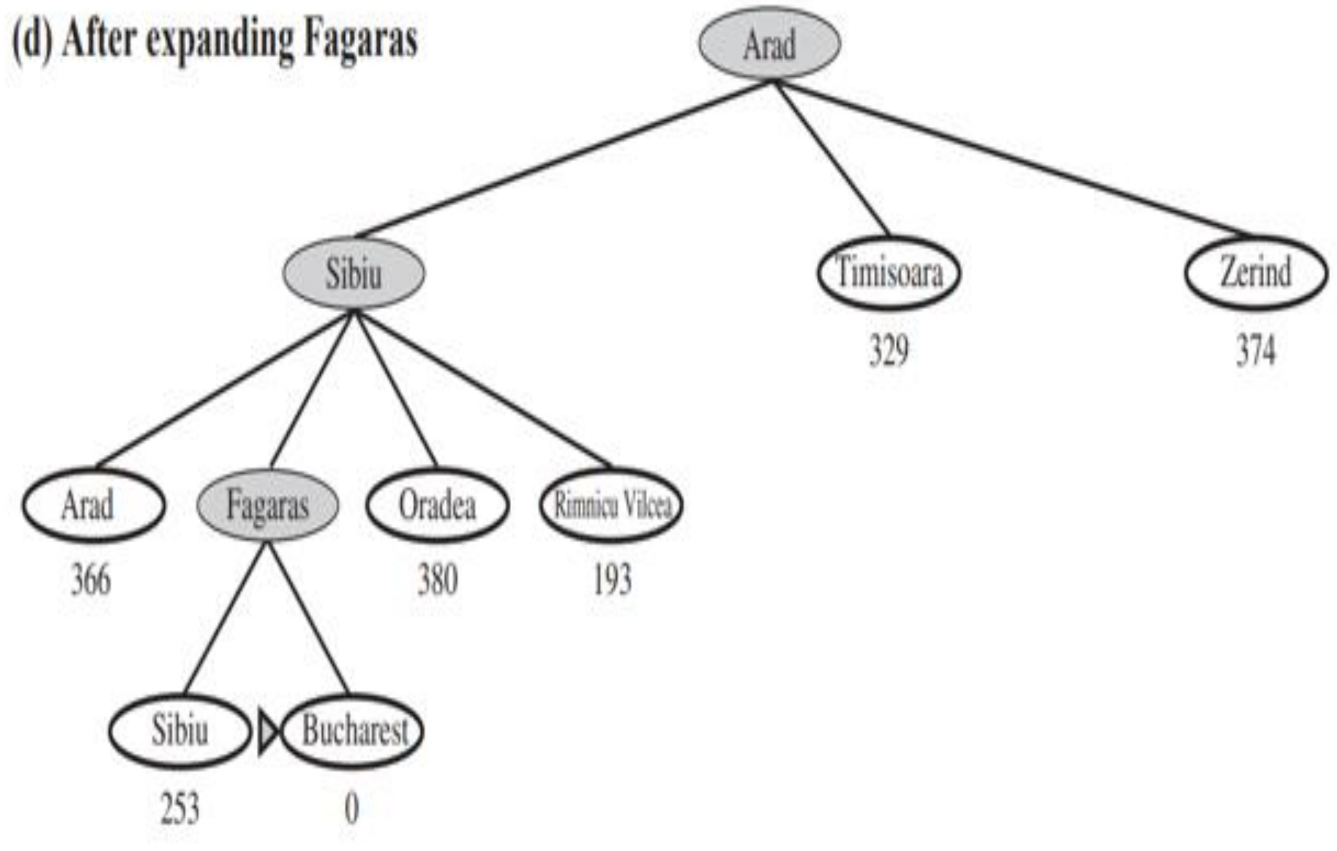


Figure: Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic hSLD . Nodes are labeled with their h-values.



Best first search algorithm

Step 1: Place the starting node into the OPEN list.

Step 2: If the OPEN list is empty, Stop and return failure.

Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.

Step 4: Expand the node n, and generate the successors of node n.

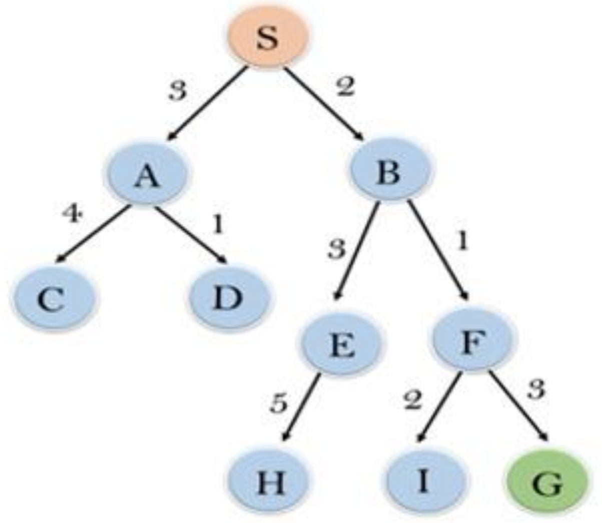
Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.

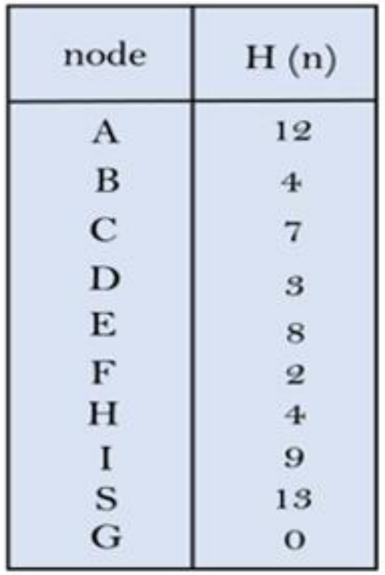
Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.

Step 7: Return to Step 2.

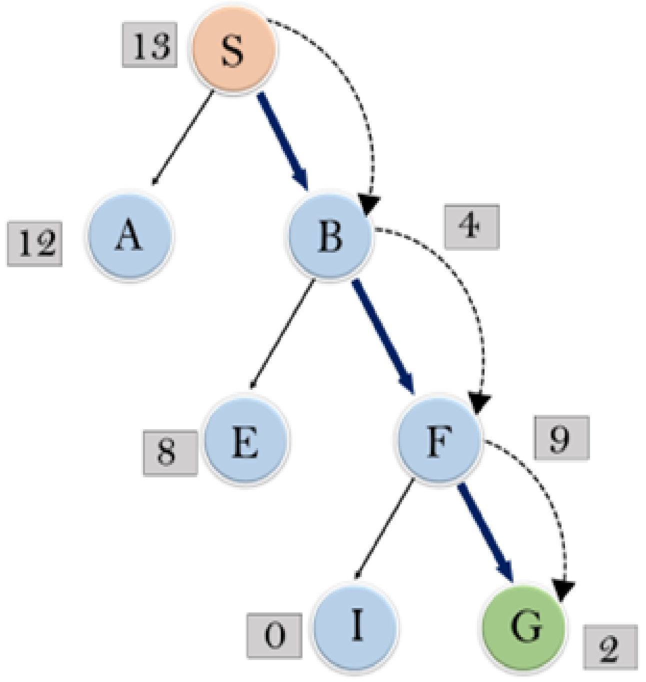
Example

Step	OPEN List	CLOSED List	Details
Initialization	[A, B]	[S]	
Iteration1: Expand B	[A]	[S,B]	h(B)<h(A)
Iteration2: Expand F	[E,F,A] [E,A]	[S,B] [S,B,F]	H(F)<H(E),H(A)
Iteration3: Visit Goal G	[I,G,E,A] [I,E,A]	[S,B,F] [S,B,F,G]	H(G)< H(E),H(A),H(I)





node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

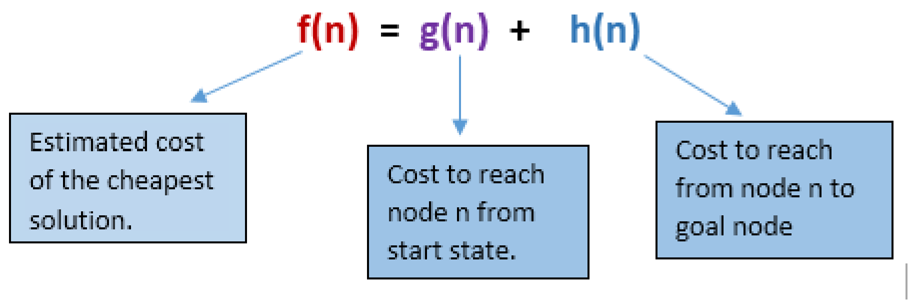


Hence the final solution path will be: **S----> B----->F----> G**

A* Search Algorithm

- A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

- In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.


f(n) = g(n) + h(n)
Estimated cost of the cheapest solution.
Cost to reach node n from start state.
Cost to reach from node n to goal node

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to Step 2.

Example1

Let's explore the application of this method to route-finding challenges in Romania for the map given in figure 3.2 .

We will employ the straight-line distance heuristic, denoted as hSLD. Specifically, for our destination in Bucharest, we require knowledge of the straight-line distances to Bucharest, as illustrated in Figure 3.22.

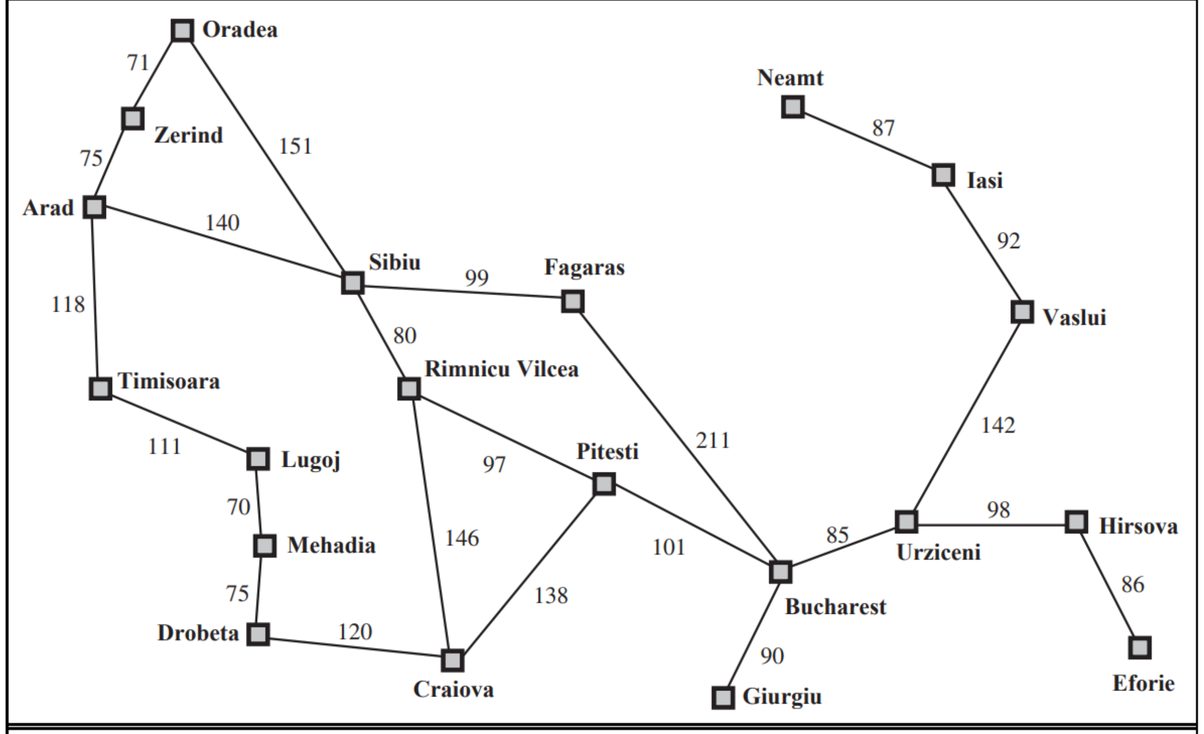
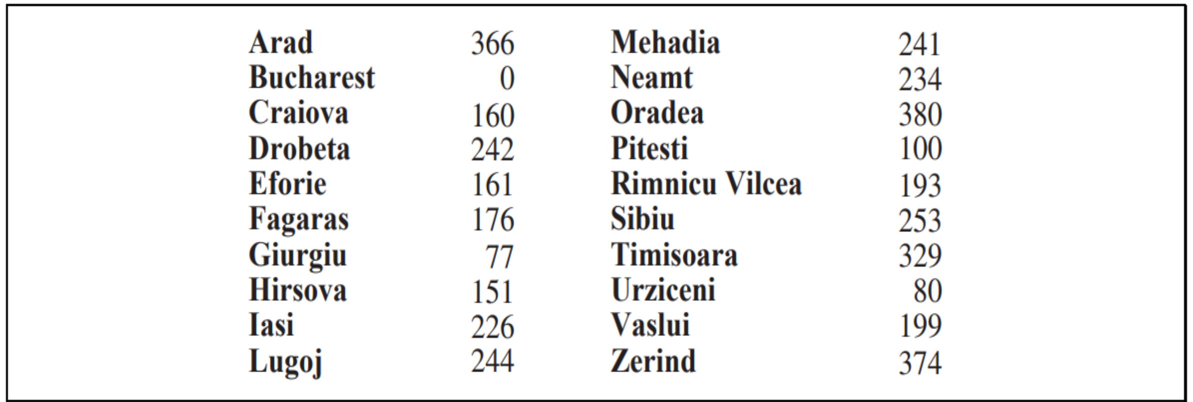


Figure 3.2 A simplified road map of part of Romania.



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

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

Figure below illustrates the Stages in **an A∗ search for Bucharest**. Nodes are labeled with f = g + h. The h values are the straight-line distances to Bucharest

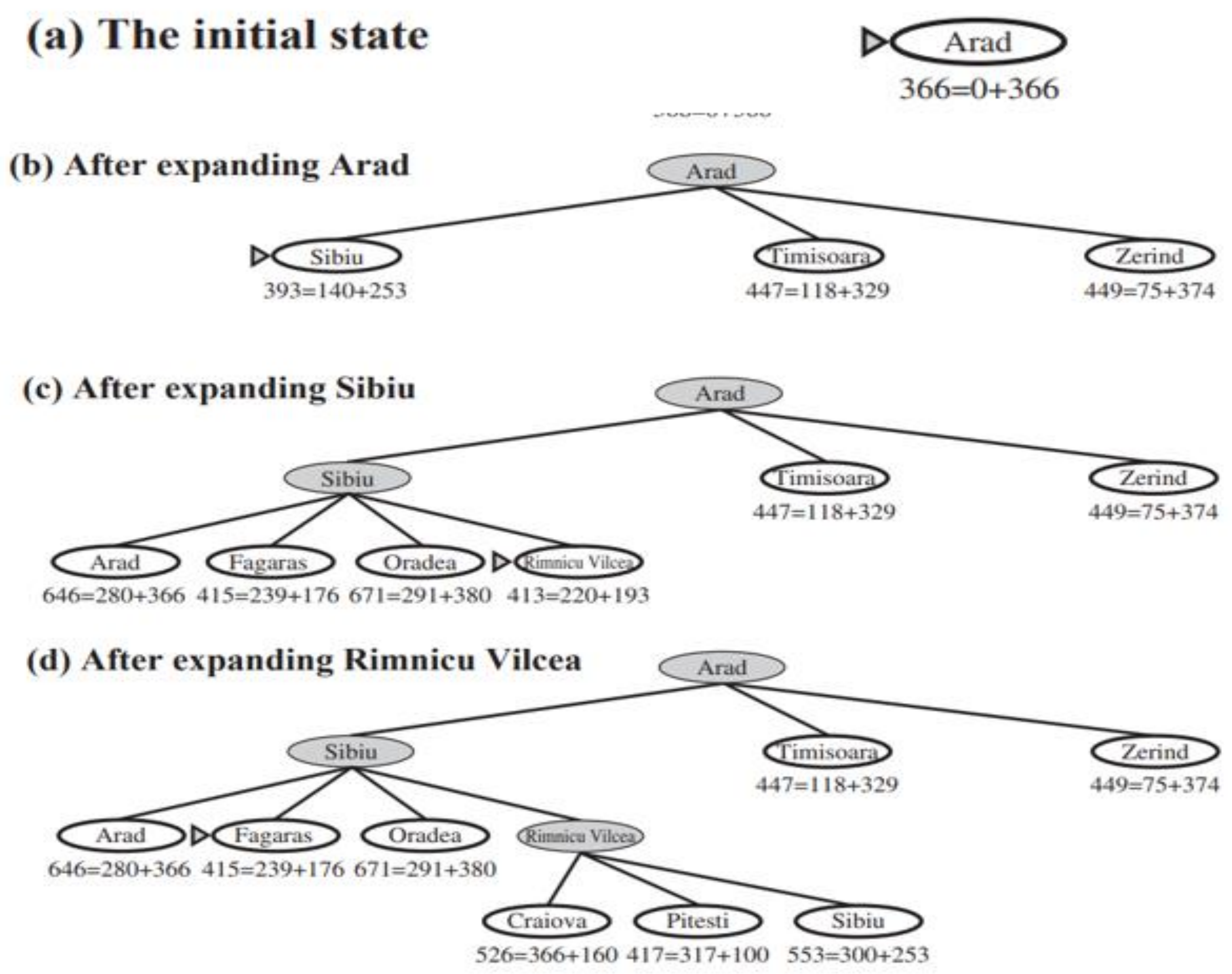
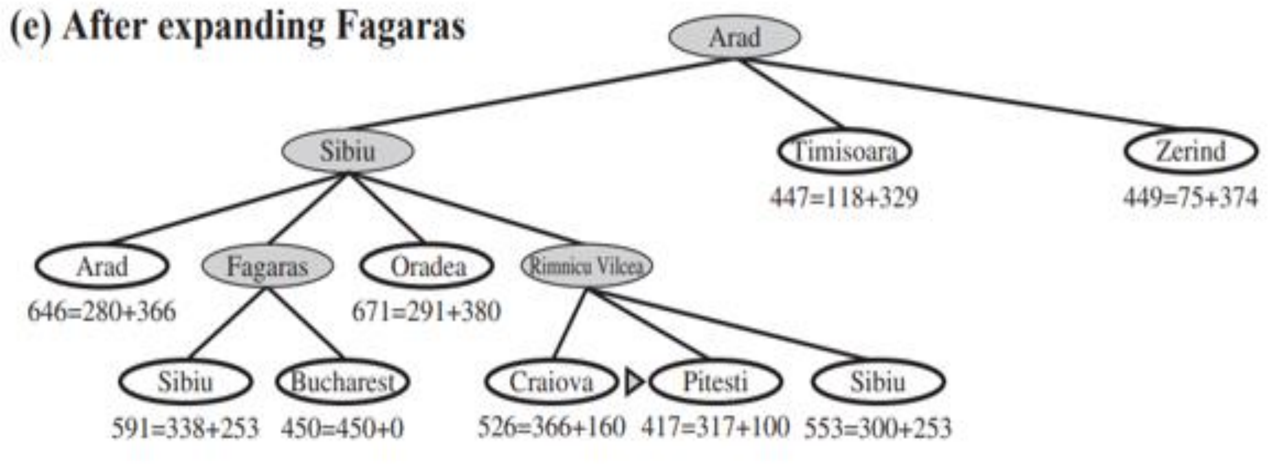
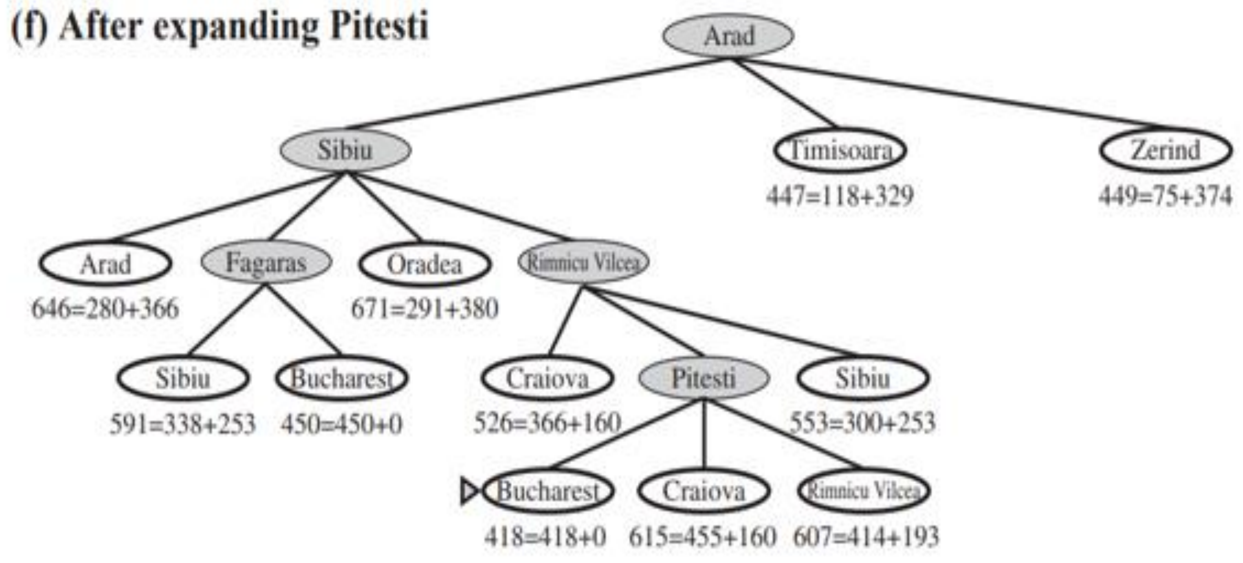


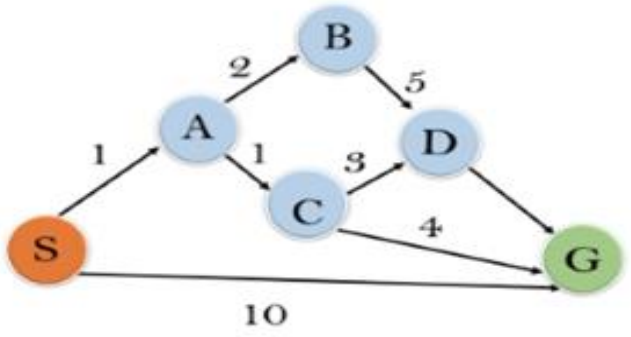
Figure below illustrates the Stages in **an A∗ search for Bucharest**. Nodes are labeled with f = g + h. The h values are the straight-line distances to Bucharest





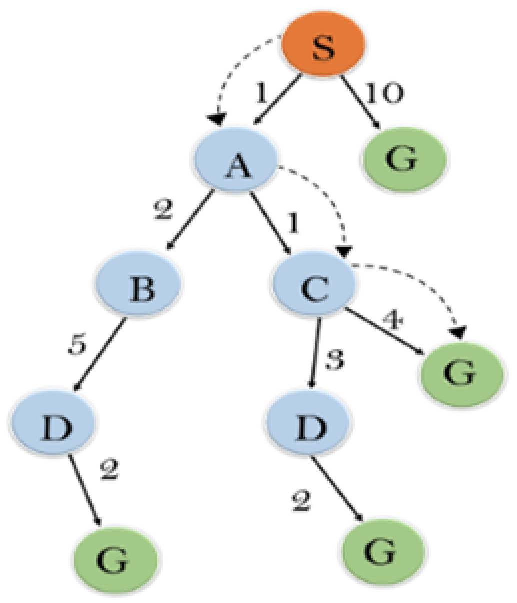
Example 2

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state. Here we will use OPEN and CLOSED list**.**



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Example 2 : Solution



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

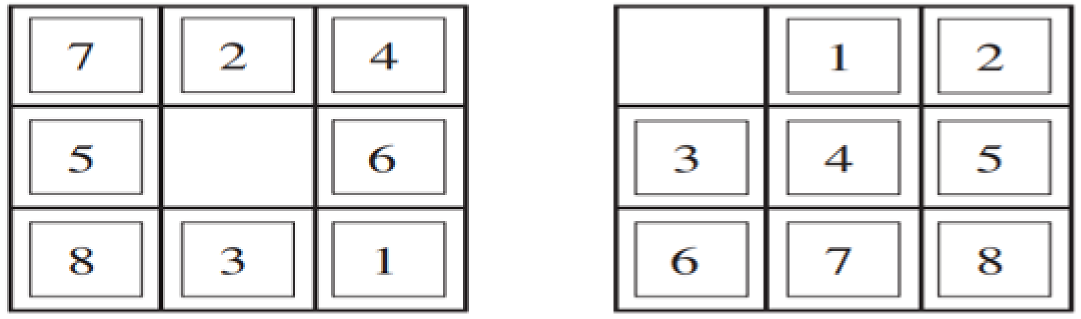
Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

2.1.c Heuristics Functions

- **Heuristic Functions h(n) guide search algorithms by estimating the cost or distance to a goal state from the current state(n).**
- **Consider the 8-puzzle game. The object of the 8 puzzle is to slide the title horizontally or vertically into the empty space until the configuration matches the goal configuration.**



Start State Goal State

- **Figure illustrates the A typical instance of the 8-puzzle. The solution is 26 steps long.**
- **The average solution cost for a randomly generated 8-puzzle instance is about 22 steps. The branching factor is about 3. (When the empty tile is in the middle, four moves are possible; when it is in a corner, two; and when it is along an edge, three.) This means that an exhaustive tree search to depth 22 would look at about $3^{22} \approx 3.1 \times 10^{10}$ states.**

2.1.c Heuristics Functions

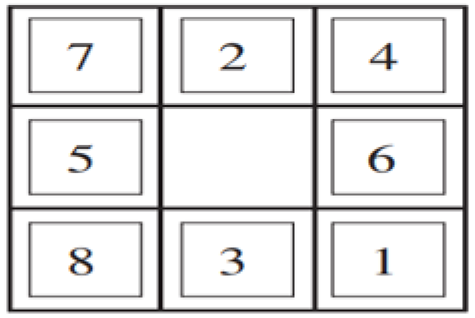
The two commonly used candidates for 8 puzzles are as follows:

h1 = the number of misplaced tiles. For Figure, all of the eight tiles are out of position, so the start state would have **h1 = 8.**

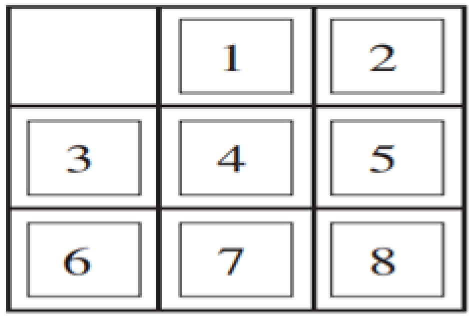
h1 is an admissible heuristic because it is clear that any tile that is out of place must be moved at least once

h2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances. This is sometimes called the city block distance or Manhattan distance. **h2** is also admissible because all any move can do is move one tile one step closer to the goal.

Tiles 1 to 8 in the start state give a Manhattan distance of **h2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18 .**



Start State



Goal State

A study on Heuristic Functions

- **The effect of heuristic accuracy on performance**
- **Generating admissible heuristics from relaxed problems**
- **Generating admissible heuristics from subproblems: Pattern databases**

A Study on heuristic functions

1. The effect of heuristic accuracy on performance:

- Experimentally it is determined that h_2 is better than h_1.
- That is for any node n, **$h_2(n) \geq h_1(n)$.** This implies that **h_2 dominate h_1**.
- Domination translates directly into efficiency.
- A^* using h_2 will never expand more nodes than A^* using h_1.

A Study on heuristic functions

2. Generating admissible heuristics from relaxed problems:

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The state-space graph of the relaxed problem is a super graph of the original state space because the removal of restrictions creates added edges in the graph.

A Study on heuristic functions

For example, if the 8-puzzle actions are described as

- **A tile can move from square A to square B if**
- **A is horizontally or vertically adjacent to B and B is blank,**

we can generate three relaxed problems by removing one or both of the conditions:

a) **A tile can move from square A to square B if A is adjacent to B.**
b) **A tile can move from square A to square B if B is blank.**
c) **A tile can move from square A to square B.**

h(n) = max{h1(n), . . . , hm(n)}

A Study on heuristic functions

3. Generating admissible heuristics from subproblems: Pattern databases

Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem. For example, **Figure below** shows a subproblem of the 8-puzzle instance.

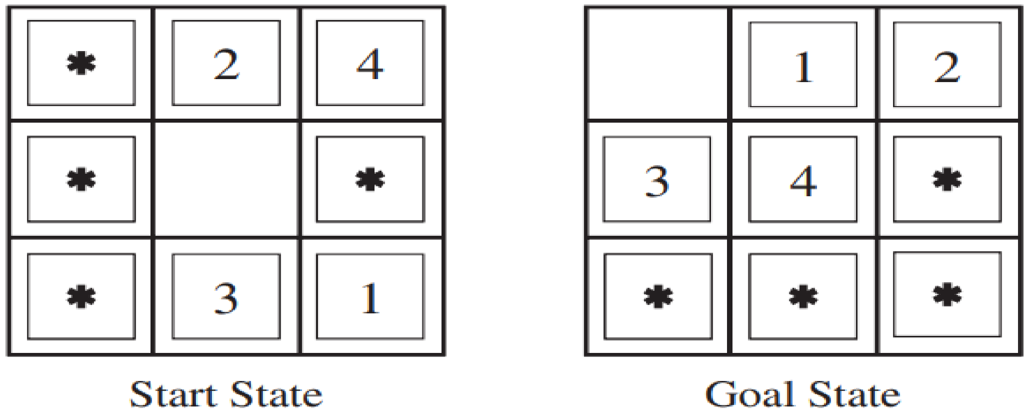


Fig: A subproblem of the 8-puzzle instance. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

A Study on heuristic functions

4. Learning heuristics from experience

- A heuristic function, denoted as **h(n)**, aims to approximate the solution cost starting from the state represented by node **n**. One of the strategies is to learning from practical experiences. In this context, "experience" refers to solving numerous instances of problems like 8-puzzles.
- For instance, a feature like "**number of misplaced tiles**" (**x_1(n)**) can be useful in predicting the distance of a state from the goal in an 8-puzzle. By gathering statistics from randomly generated 8-puzzle configurations and their actual solution costs, one can use these features to predict **h(n)**.
- Multiple features, such as **x_2(n)** representing the **"number of pairs of adjacent tiles that are not adjacent in the goal state**," can be combined using a linear combination approach:

$$h(n) = c_1 x_1(n) + c_2 x_2(n).$$

2. Logic Agents

1. **Knowledge–based agents,**
2. **The Wumpus world,**
3. **Logic,**
4. **Propositional logic,**
5. **Reasoning patterns in Propositional Logic**

Logical Agents

- Stuart Russell and Peter Norvig, in their influential textbook "Artificial Intelligence: A Modern Approach," describe ***logical agents as those that operate based on knowledge representation and logical inference***.
- According to their framework, an agent perceives its environment **through sensors, maintains an internal state (knowledge base), and acts upon the environment through effectors**.
- Logical agents specifically use **logical reasoning** to make decisions.

Examples of Logical Agents

1. Expert Systems
2. Theorem Provers
3. Logic based Planning Agents
4. Semantic Web Agents

Expert Systems

- ***Description:*** Expert systems are designed to emulate the decision-making abilities of human experts in specific domains. They use a knowledge base of facts and rules, often represented using propositional or first-order logic, to make decisions and provide advice.
- ***Example:*** MYCIN, an expert system developed for medical diagnosis, is an example where logical rules are used to identify potential bacterial infections and recommend antibiotic treatments.
-

Theorem Provers:

- ***Description:*** Theorem proving systems use logical reasoning to prove mathematical theorems. They manipulate symbolic expressions and apply logical rules to demonstrate the validity of a given theorem.
- ***Example**:* **Prover9** is an automated theorem prover that uses first-order logic to prove mathematical statements.

Logic-Based Planning Agents

- ***Description****:* Planning agents use logic to generate plans or sequences of actions to achieve specific goals. They represent the initial state, goal state, and possible actions in a logical framework to determine the optimal plan.
- ***Example****:* The STRIPS (Stanford Research Institute Problem Solver) planning system uses logical representations of states, actions, and goals to plan sequences of actions for an agent in an environment.

Semantic Web Agents:

- *Description:* Agents on the Semantic Web use logic-based languages like RDF (Resource Description Framework) and OWL (Web Ontology Language) to represent and reason about knowledge on the web.
- *Example:* A semantic web agent might use logical reasoning to infer relationships between entities based on RDF triples, enhancing the understanding of information on the web.