Module3

Artificial Intelligence

Dr.Thyagaraju G S, Professor and HoD, Department of CSE, SDM Institute Of Technology,Ujire-574240.
Source Book : S. Sridhar, M Vijayalakshmi “Machine Learning”. Oxford, 2021

Contents

1. **Informed Search Strategies:**
   - **a. Greedy best-first search,**
   - **b. A*search,**
   - **c. Heuristic functions.**
2. **Logical Agents:**
   - **a) Knowledge–based agents,**
   - **b) The Wumpus world,**
   - **c) Logic,**
   - **d) Propositional logic,**
   - **e) Reasoning patterns in Propositional Logic**

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

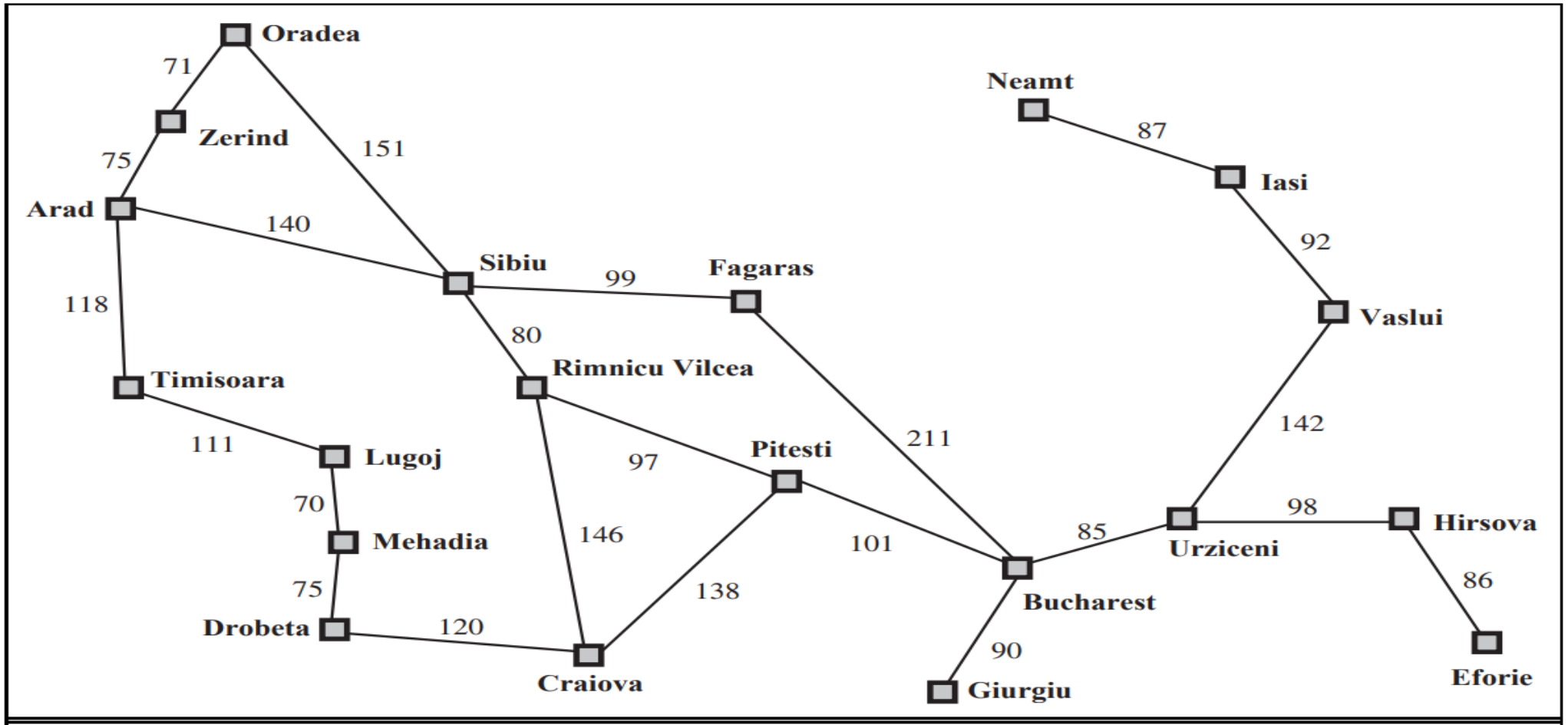


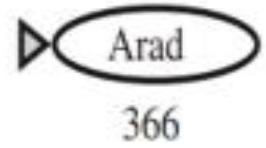
Figure 3.2 A simplified road map of part of Romania.

2.1.a Greedy Best First Search Algorithm

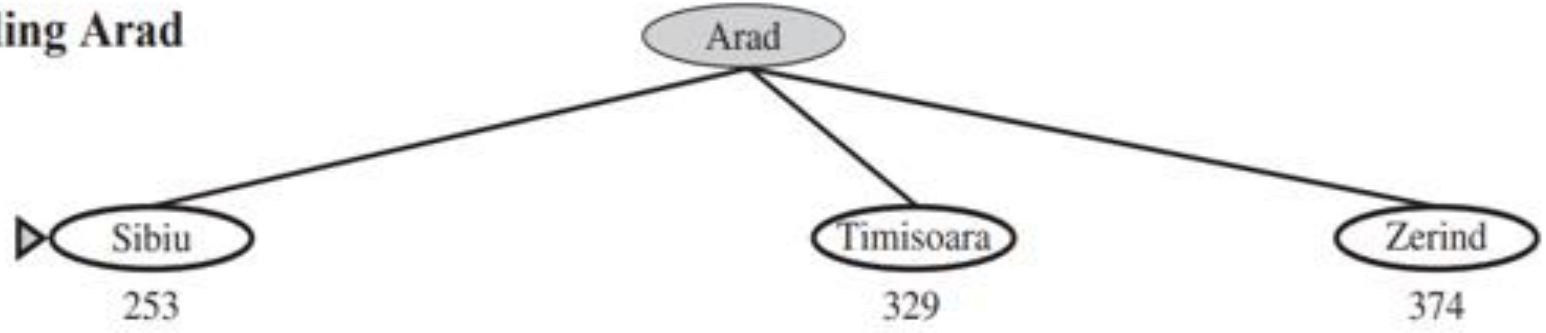
Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

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

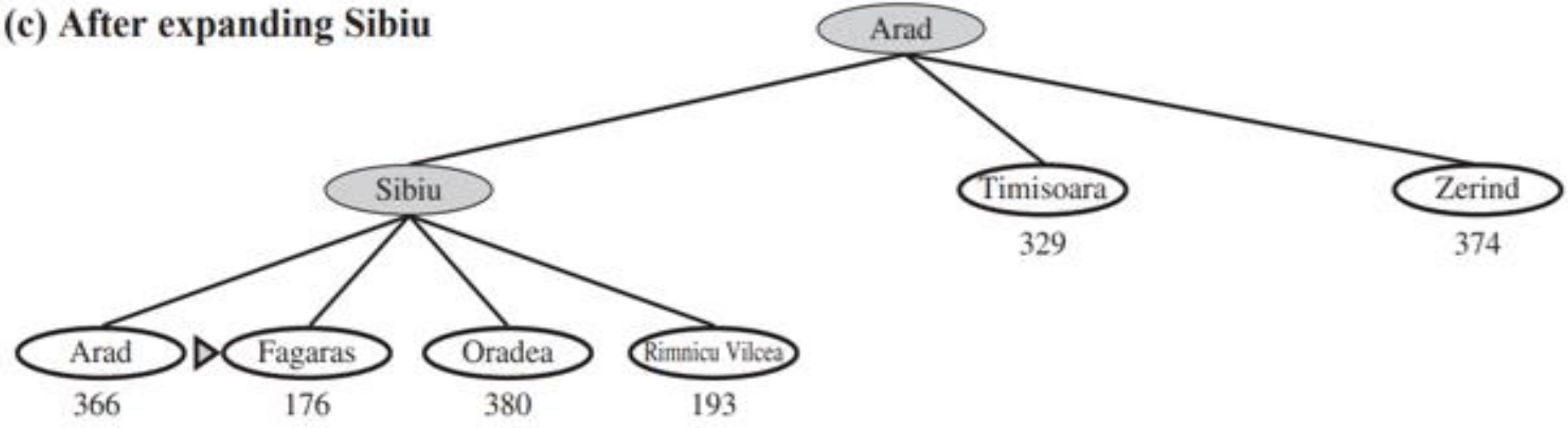
(a) The initial state

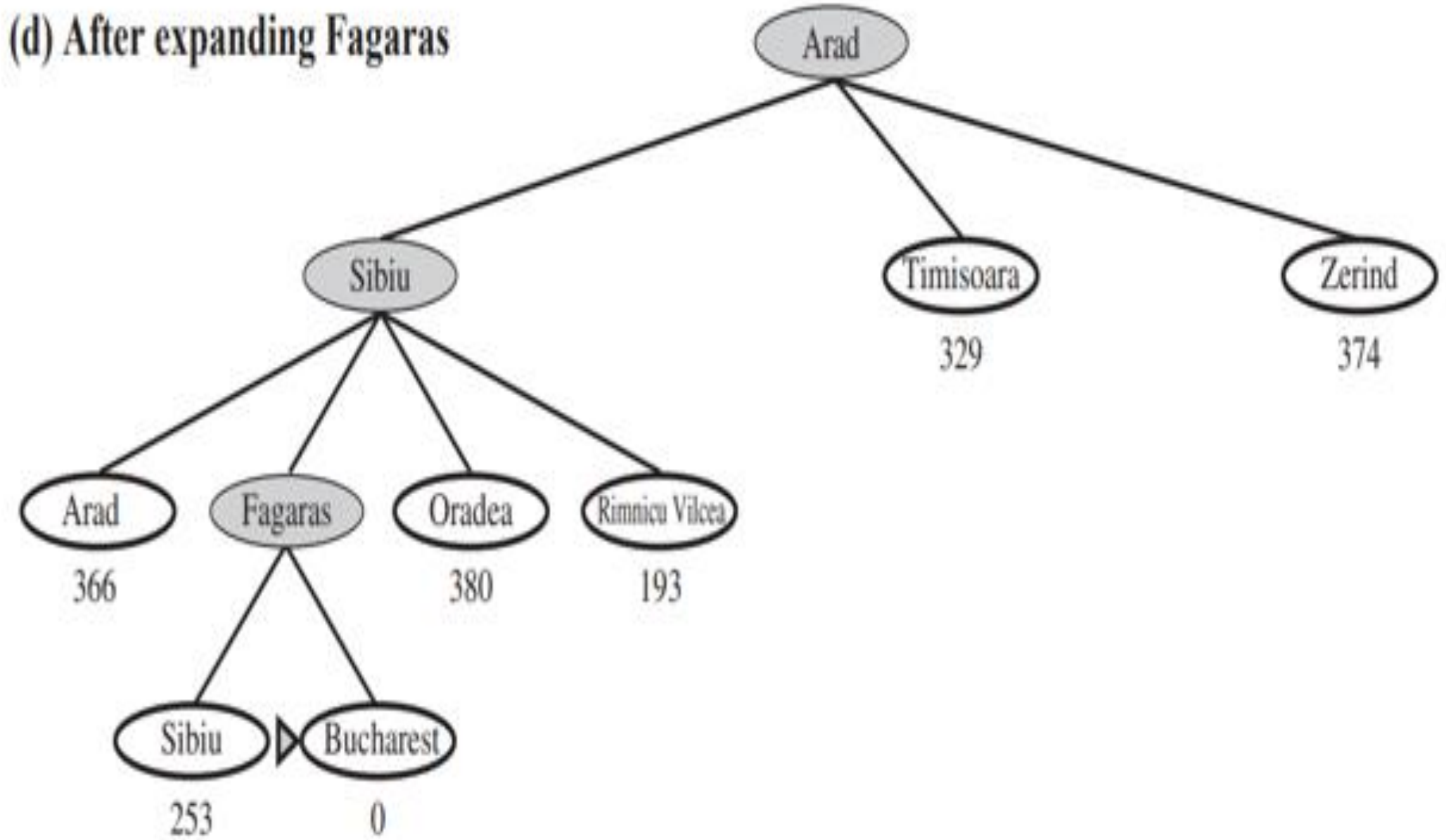


(b) After expanding Arad



(c) After expanding Sibiu




(d) After expanding Fagaras
Arad
Sibiu
Timisoara
Zerind
329
374
Arad
Fagaras
Oradea
Rimnicu Vilcea
366
380
193
Sibiu
Bucharest
253
0

Step	Open List	Closed List	Details
Initialization	[Arad]	[ ]	
Start: Expand Arad	[Sibiu, Timisoara, Zerind]	[Arad]	
Iteration1: Expand Sibiu	[Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind]	[Arad,Sibiu]	h(Sibiu)<h(Timisoara),h(Zerind)
Iteration2: Expand Fagaras	[Bucharest, Oradea, Rimnicu Vilcea, Timisoara, Zerind]	[Arad,Sibiu,Fagarus]	h(Fagaras)<h(Timisoara),h(Zerind),h(Oradea),h(Rimnicu Vilcea)
Iteration3: Visit Goal	[Oradea, Rimnicu Vilcea, Timisoara, Zerind]	[Arad,Sibiu,Fagarus, Bucharest]	h(Bucharest) < h(Timisoara), h(Zerind), h(Oradea), h(Rimnicu Vilcea)

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.

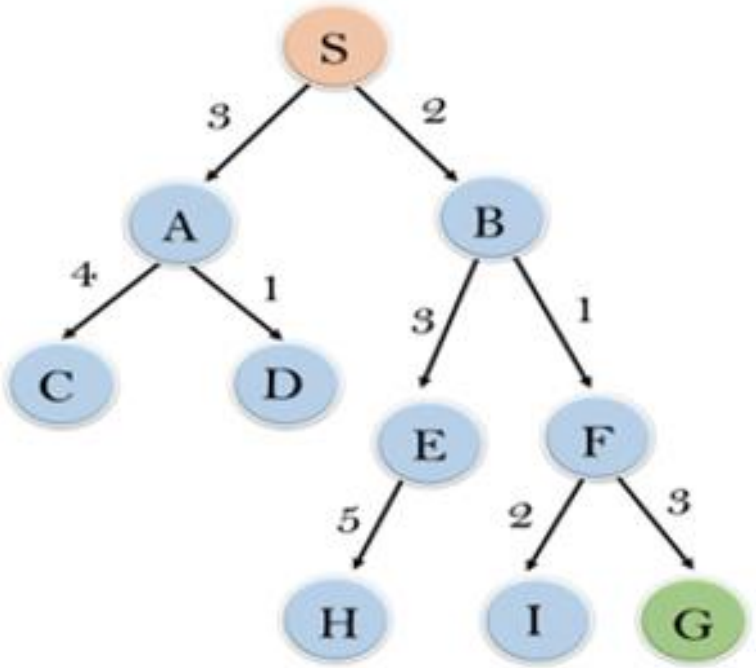
Step 4: Expand the node n, and generate the successors of node n.

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.

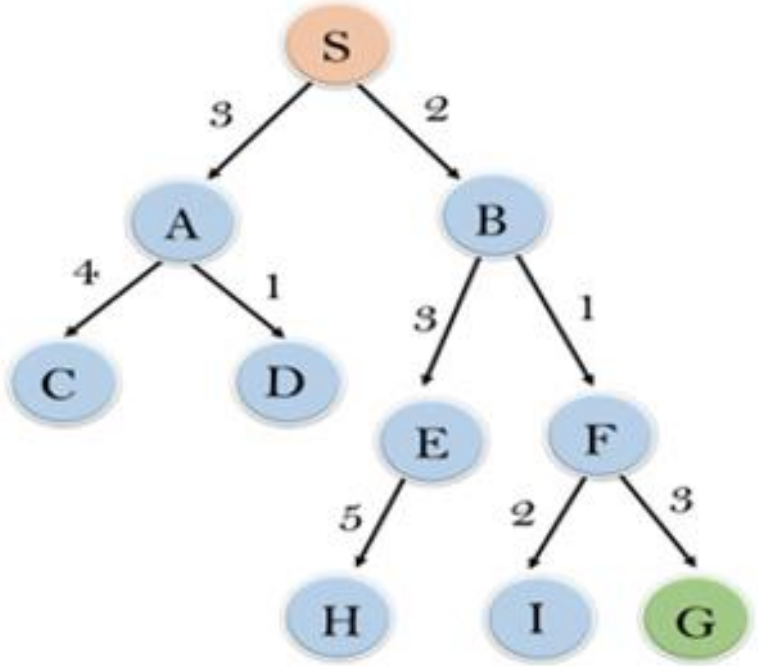
Step 7: Return to Step 2.

Apply BFS



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

Example



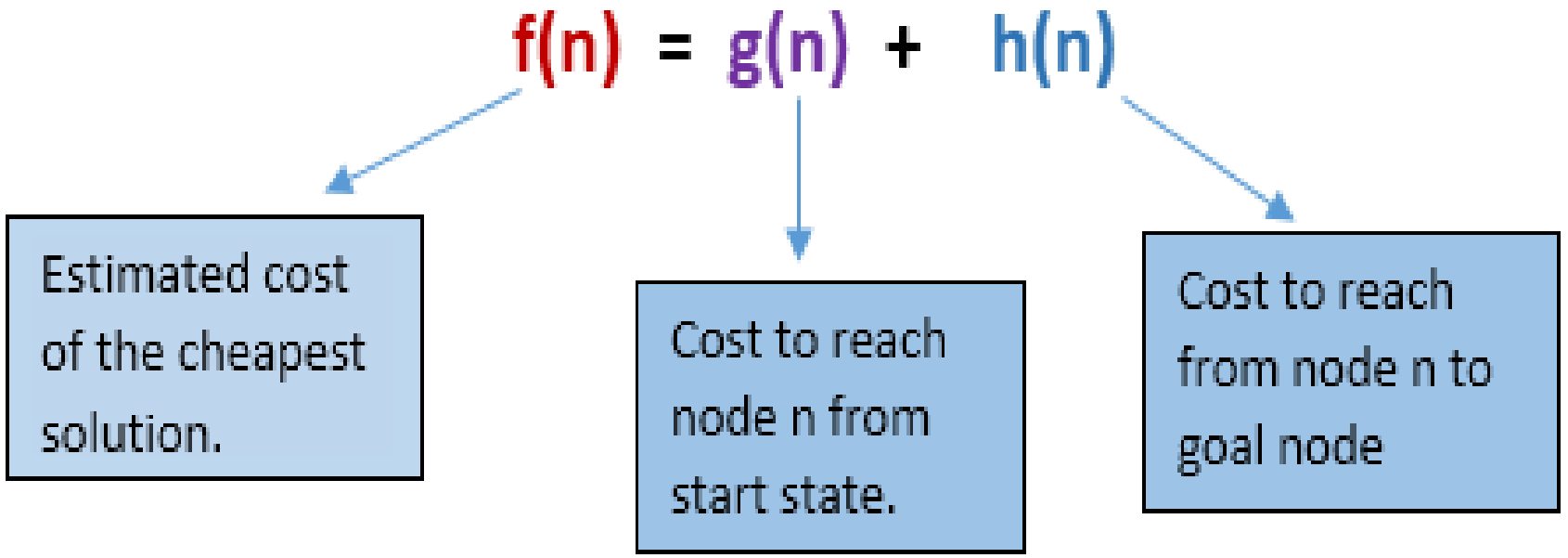
node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Step	Open List	Closed List	Details
Initialization	**[S]**	**[ ]**	
Start: Expand S	**[A,B]**	**[S]**	**h(B)<h(A)**
Iteration1: Expand B	**[E,F,A]**	**[S,B]**	
Iteration2: Expand F	**[I,G,E,A]**	**[S,B,F]**	**h(F)<h(E),h(A)**
Iteration3: Visit Goal	**[I,E,A)**	**[S,B,F,G]**	**h(G)<h(E),h(A),h(I)**

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 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**.

A* Search Algorithm



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.

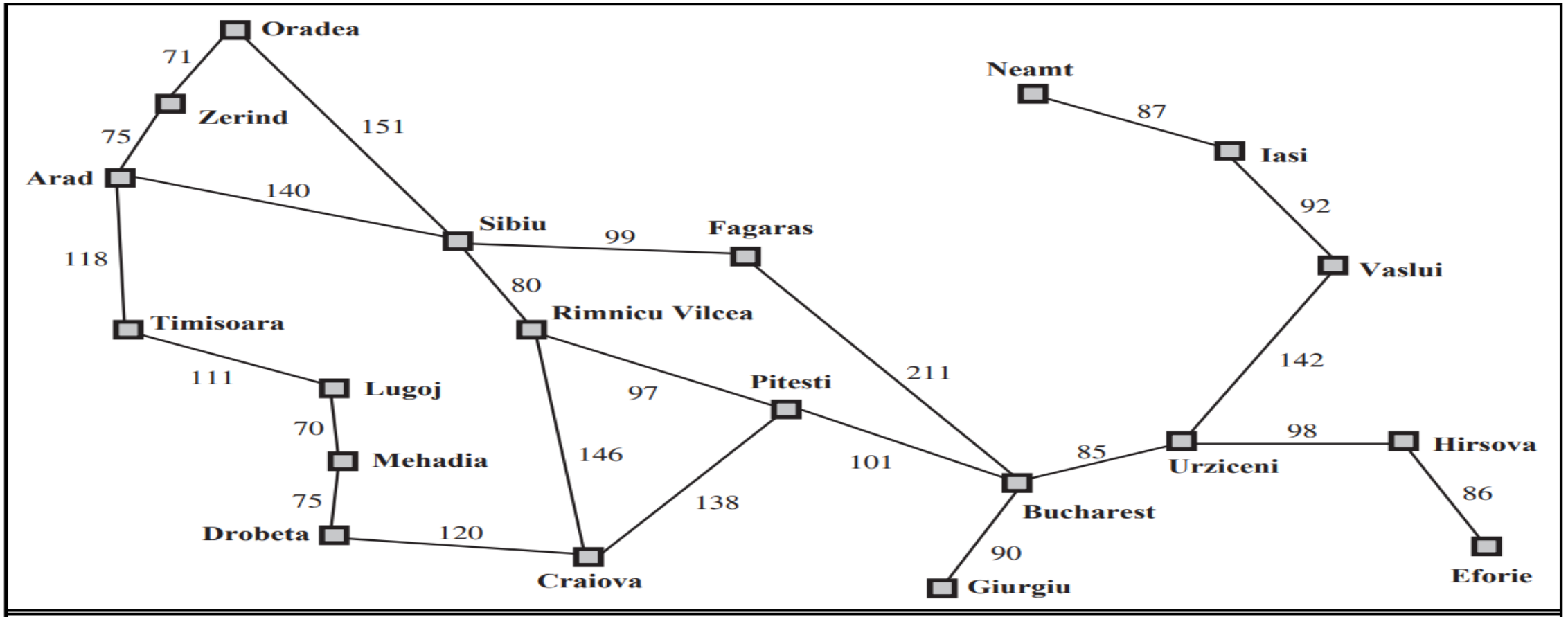


Figure 3.2 A simplified road map of part of Romania.

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.

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

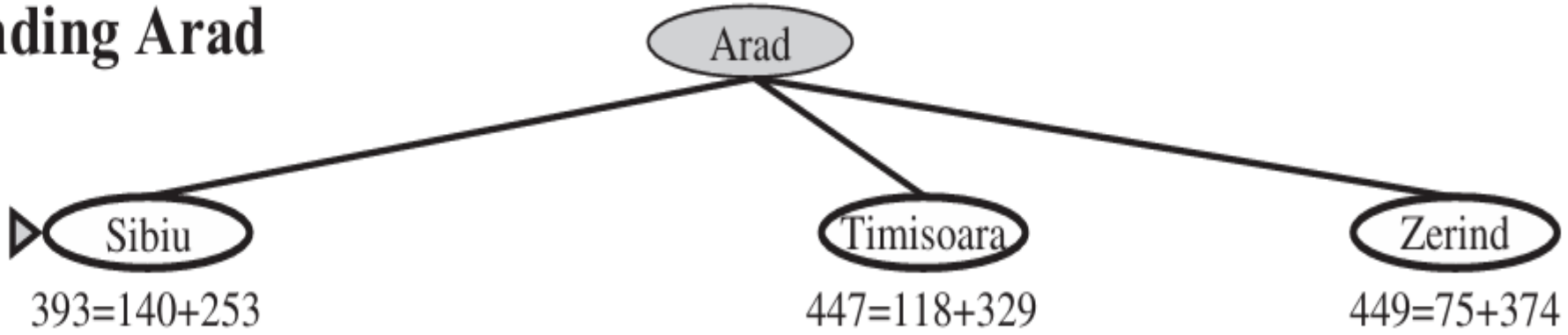
Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

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

(a) The initial state

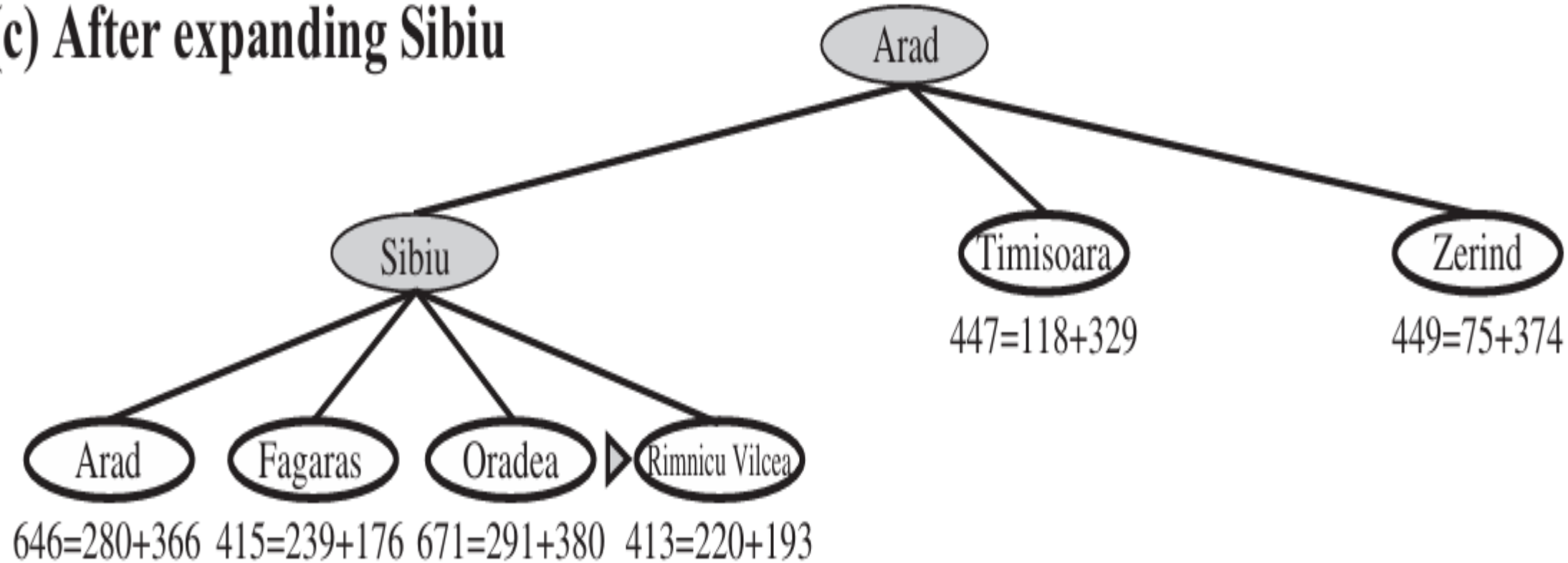
Open List : [Arad] Closed List : [ ]

(b) After expanding Arad



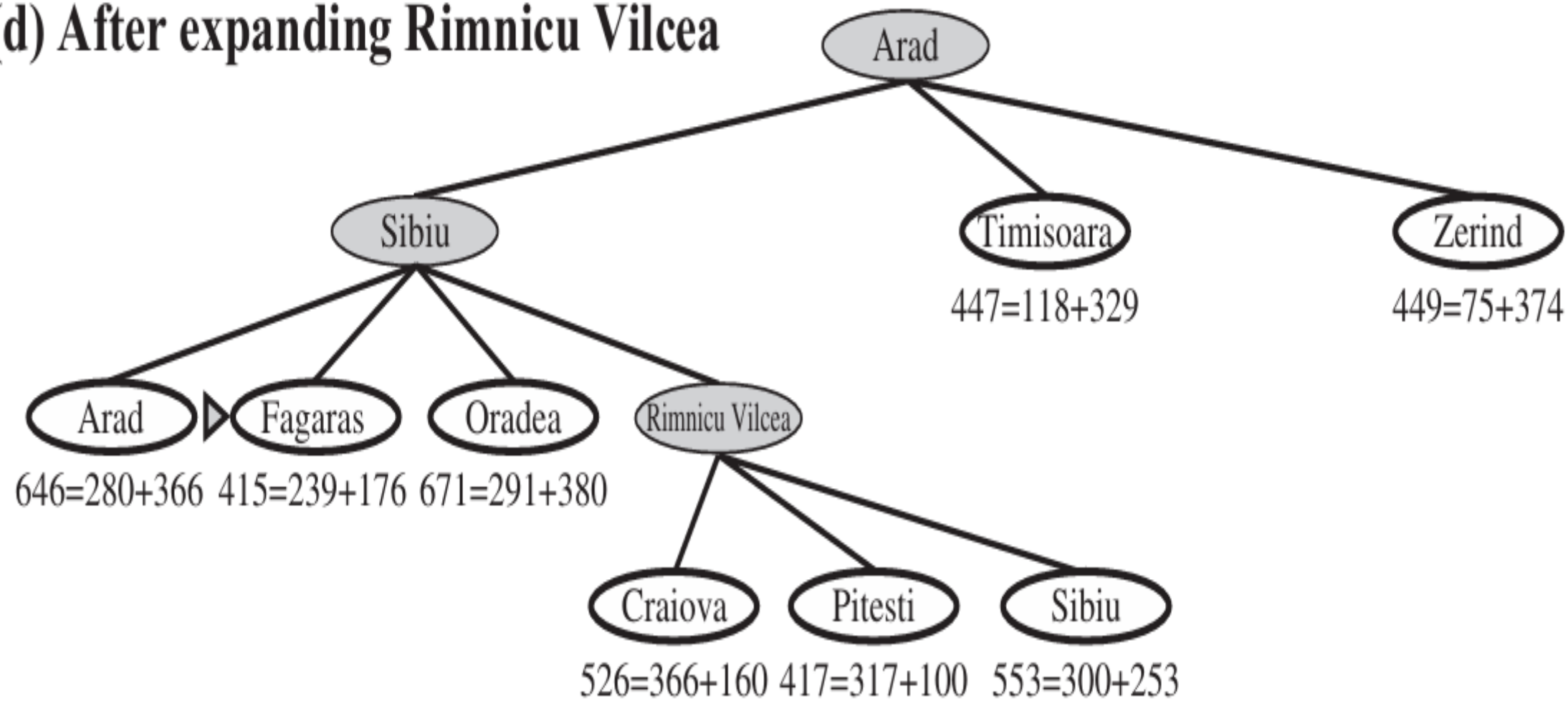
Open List : [Sibiu,Timisoara,Zerind] Closed List : [Arad]

(c) After expanding Sibiu



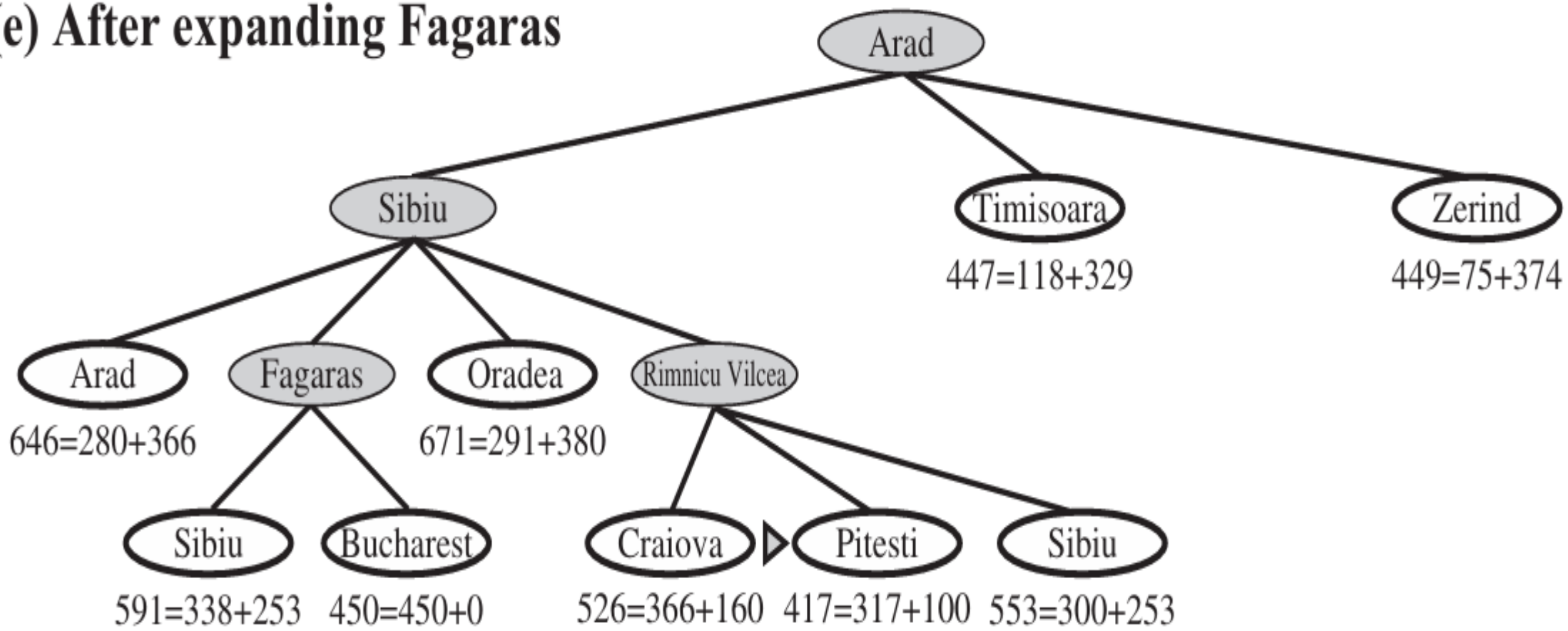
Open List : [Fagaras, Oradea,Rimnicu Vilcea, Timisoara, Zerind] Closed List : [Arad,Sibiu]

(d) After expanding Rimnicu Vilcea



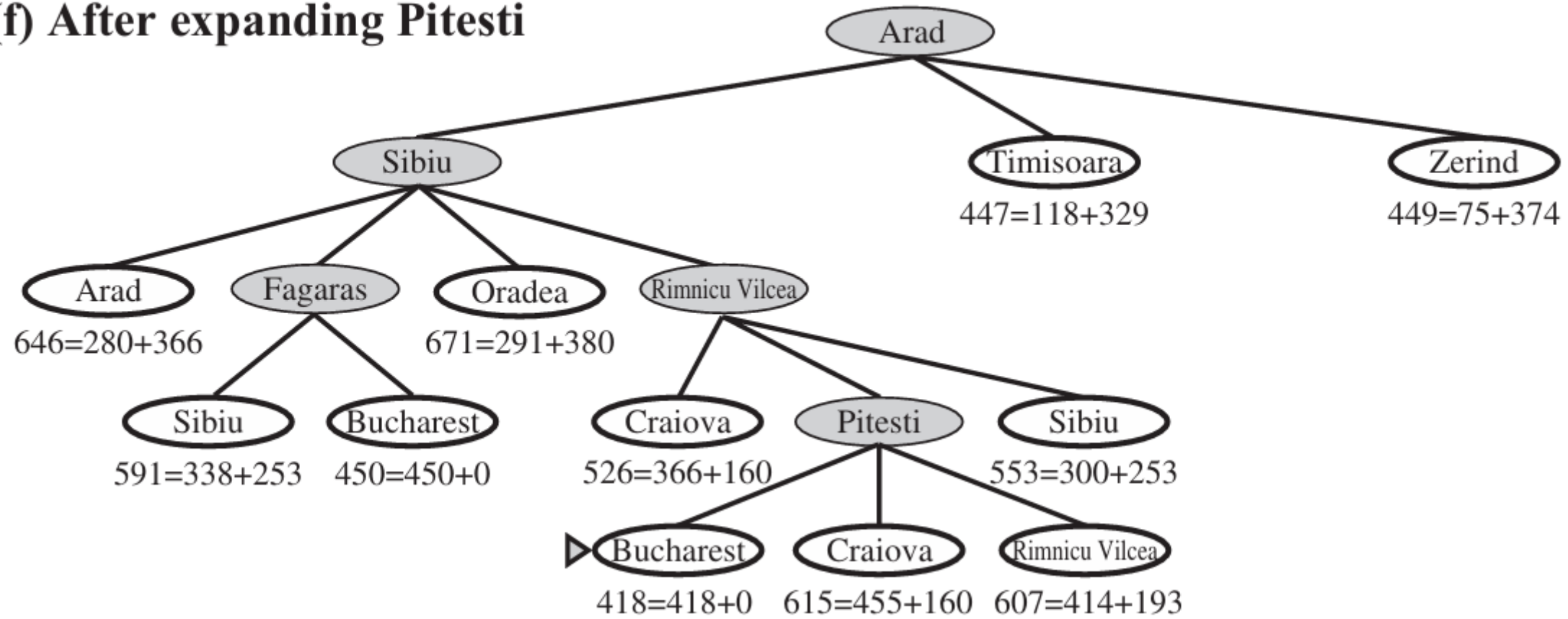
Open List : [Fagaras, Oradea,Craiova,Pitesti,Sibiu, Timisoara, Zerind] Closed List : [Arad,Sibiu,Rimnicu Vilcea]

(e) After expanding Fagaras



Open List : [Oradea,Craiova,Pitesti,Sibiu, Timisoara, Zerind]
Closed List : [Arad,Sibiu,Fagarus,Bucharest] // After unwinding back to Sibiu and expanding Fagaras

(f) After expanding Pitesti



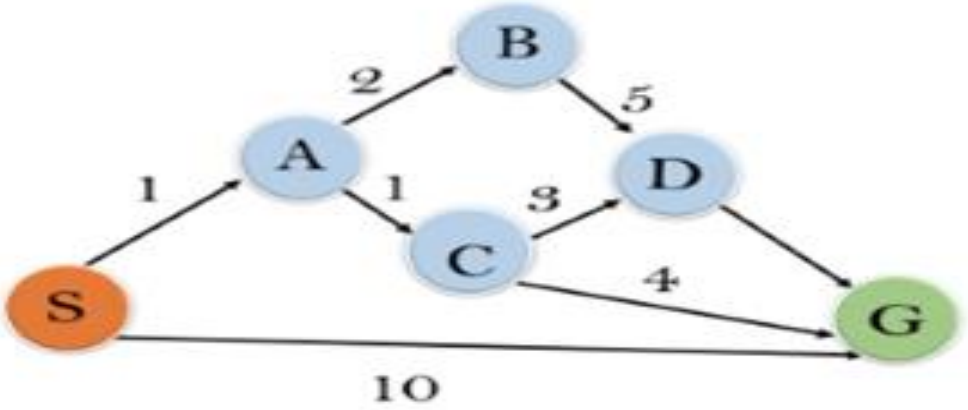
Open List : [Oradea,Craiova,Sibiu, Timisoara, Zerind]

Closed List : [Arad,Sibiu,Rimnicu Vilcea,Pitesti, Bucharest] // After switching back to Rimnicu Vilcea and expanding Pitesti

Step	Open List	Closed List	Details
Initialization	[Arad]	[ ]	
Start: Expand Arad	[Sibiu, Timisoara, Zerind]	[Arad]	
Iteration1: Expand Sibiu	[Fagaras, Oradea, Rimnicu Vilcea, Timisoara, Zerind]	[Arad, Sibiu]	f(Sibiu)<f(Timisoara),f(Zerind)
Iteration2: Expand Rimnicu Vilcea	[Craiova,Pitesti,Sibiu,Fagarus,, Timisoara, Zerind]	[Arad,Sibiu, Rimnicu Vilcea]	f(Rimnicu Vilcea) is Minimum of all
Iteration 3: Return (Back Track) to previous node Sibiu and Expand Fagaraus and Check the total cost to Bucharest [You Will get First Optimal Path]			
Iteration 4: Explore Rimnicu Vilcea branch Expand Pitesti	[Craiova, Bucharest, Sibiu,Fagarus, Timisoara, Zerind]	[Arad,Sibiu, Rimnicu Vilcea, Pitesti,]	f(Pitesti) is Minimum of all
Visit Goal	[Craiova, Bucharest, Sibiu,Fagarus, Timisoara, Zerind]	[Arad,Sibiu, Rimnicu Vilcea, Pitesti,Bucharest]	f(Bucharest) is Minimum and Goal is reached.

Example 2: Apply A* Search for the following

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**.**

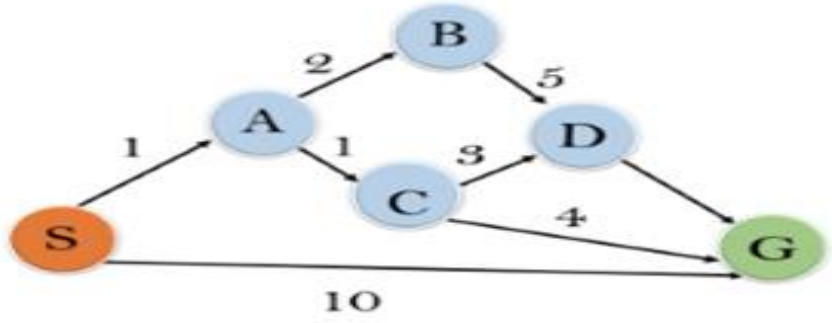


State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

To apply the *A* Search algorithm* to the given graph, we will use the following steps:

Step 1: Define the Components

- **g(n)**: The cost from the start node S to the current node n.
- **h(n)**: The heuristic cost (as provided in the table) from the current node n to the goal G.
- **f(n) = g(n) + h(n)**: The total estimated cost of the path through node n.

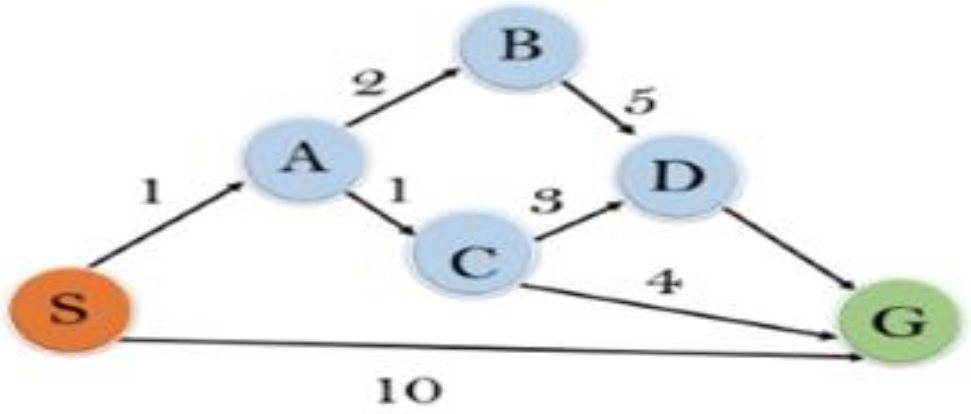


State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Step 2: Initialize Lists

- **Open List**: Contains nodes to be evaluated. Initially, it contains only the start node S.
- **Closed List**: Contains nodes that have been evaluated.

We will begin from S, the starting node.

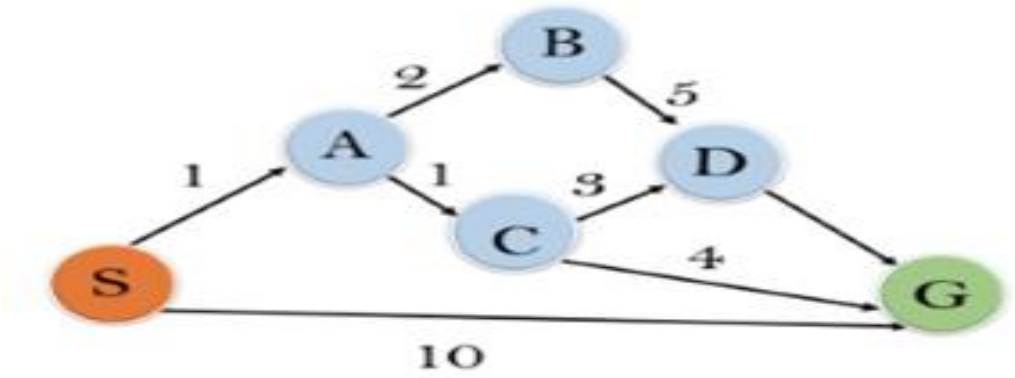


State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Step 3: Stepwise Solution

Initial State:

- **Open List**: [S]
- **Closed List**: [ ]
- **f(S) = g(S) + h(S) = 0 + 5 = 5**



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

Step3 : Iteration1

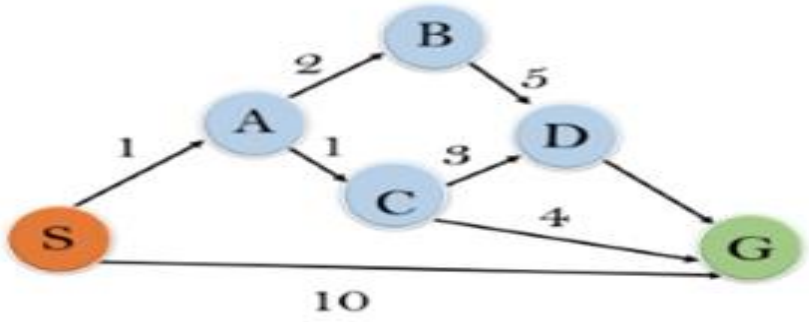
Select the node from the Open List with the lowest f(n) (i.e., S).

Expand S. Its neighbors are A and G.

- g(A)=1, h(A)=3 → f(A)=1+3=4
- g(G)=10,h(G)=0 → f(G)=10+0=10

Add A and G to the Open List and move S to the Closed List.

- **Open List**: [A(4),G(10)]
- **Closed List**: [S]



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

Step3 : Iteration2

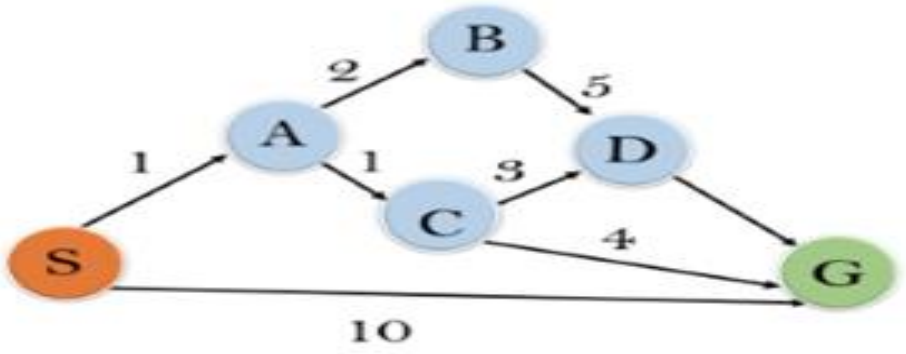
Select A (smallest f(n)).

Expand A. Its neighbors are B and C.

- g(B)=1+2=3, h(B)=4 → f(B)=3+4=7
- g(C)=1+1=2, h(C)=2 → f(C)=2+2=4

Add B and C to the Open List, and move A to the Closed List.

- **Open List**: [C(4), B(7)]
- **Closed List**: [S, A]



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

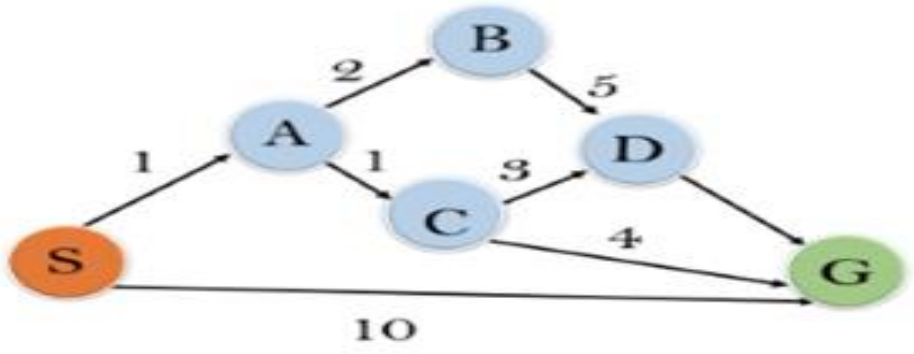
Step3 : Iteration3

Select node with the lowest f(n): C(smallest f(n)=4).
Expand C. Its neighbours are D and G.

g(D)=5, h(D)=6, so f(D)=g(D)+h(D)=5+6=11.

g(G)=6, h(G)=0, so f(G)=g(G)+h(G)=6+0=6.

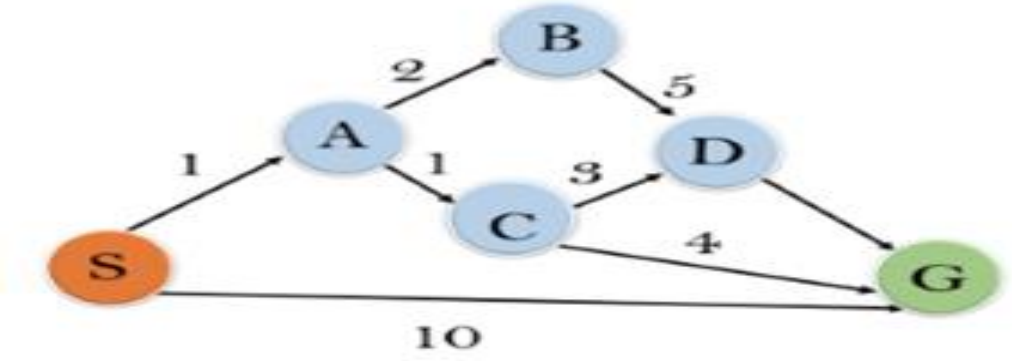
Add G and D to the Open List. Move C to the Closed List.
Open List: [G(6),B(7),D(11)]
Closed List: [S,A,C]



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

Step3 : Iteration4

Select node with the lowest f(n): G (**smallest f(n)=6**).
G is the goal node, so the algorithm terminates here.

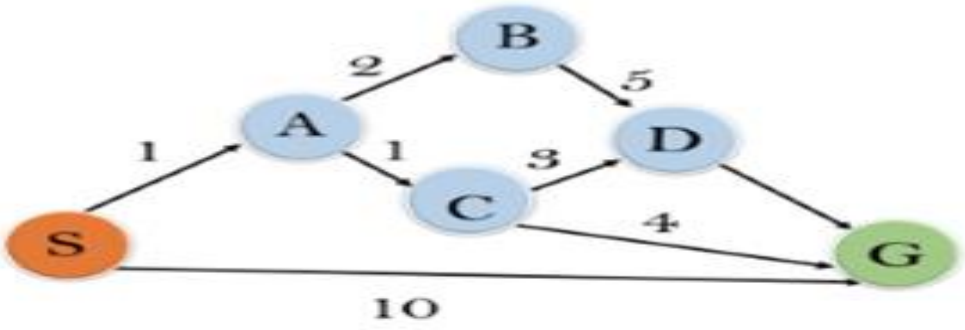


State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Final Solution

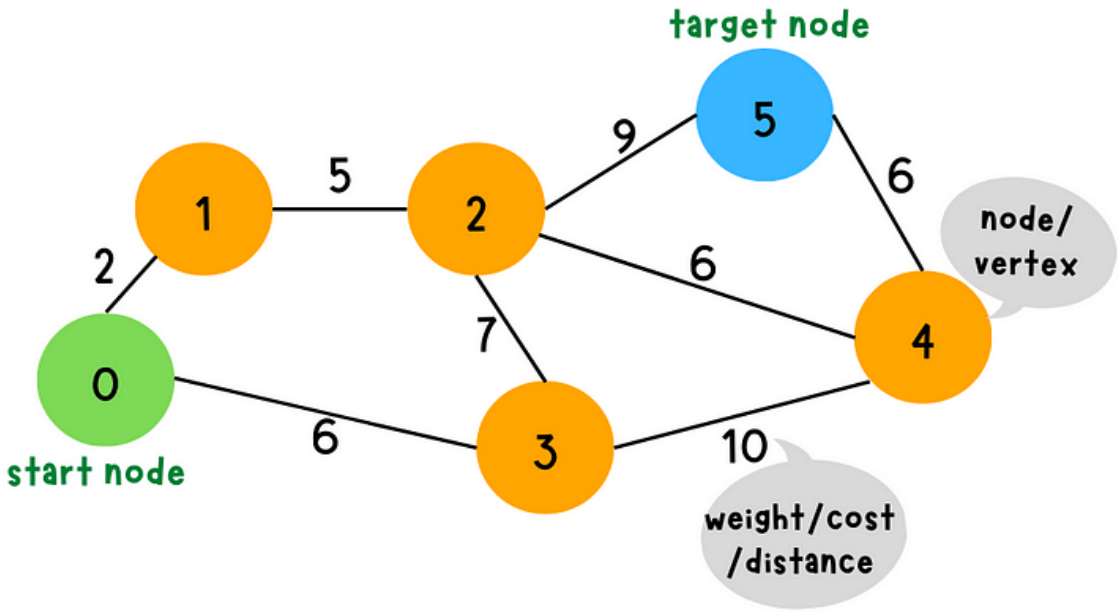
The path found is S→A→C→G.

Total Cost: The total cost is the path S→A→C→G which is 1+1+4=6.



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

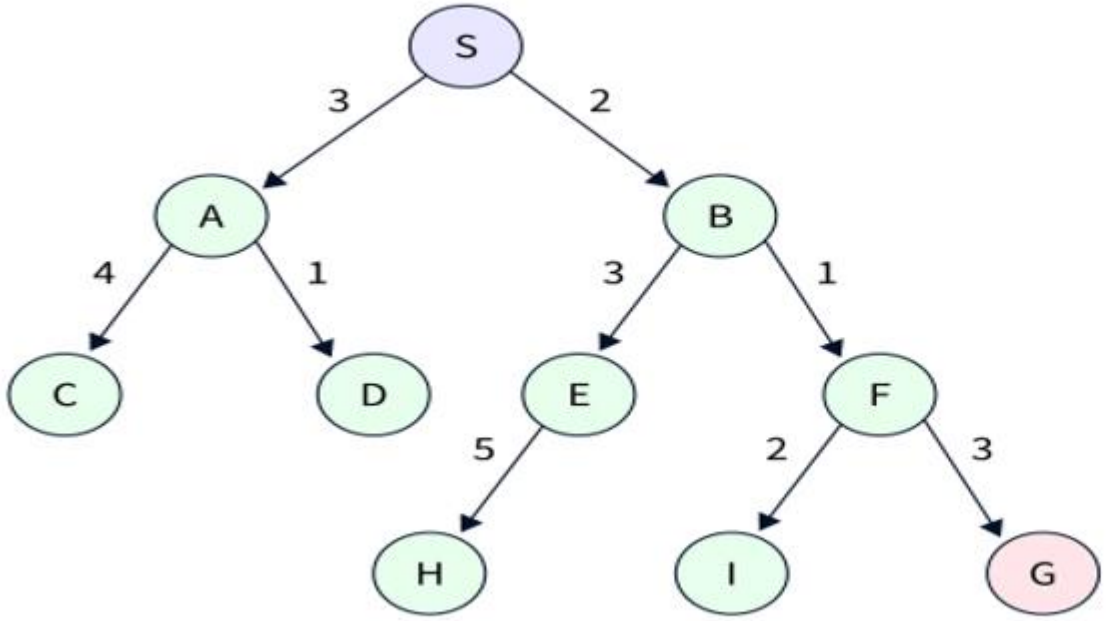
Solve the following using A* Algorithm



h(n)

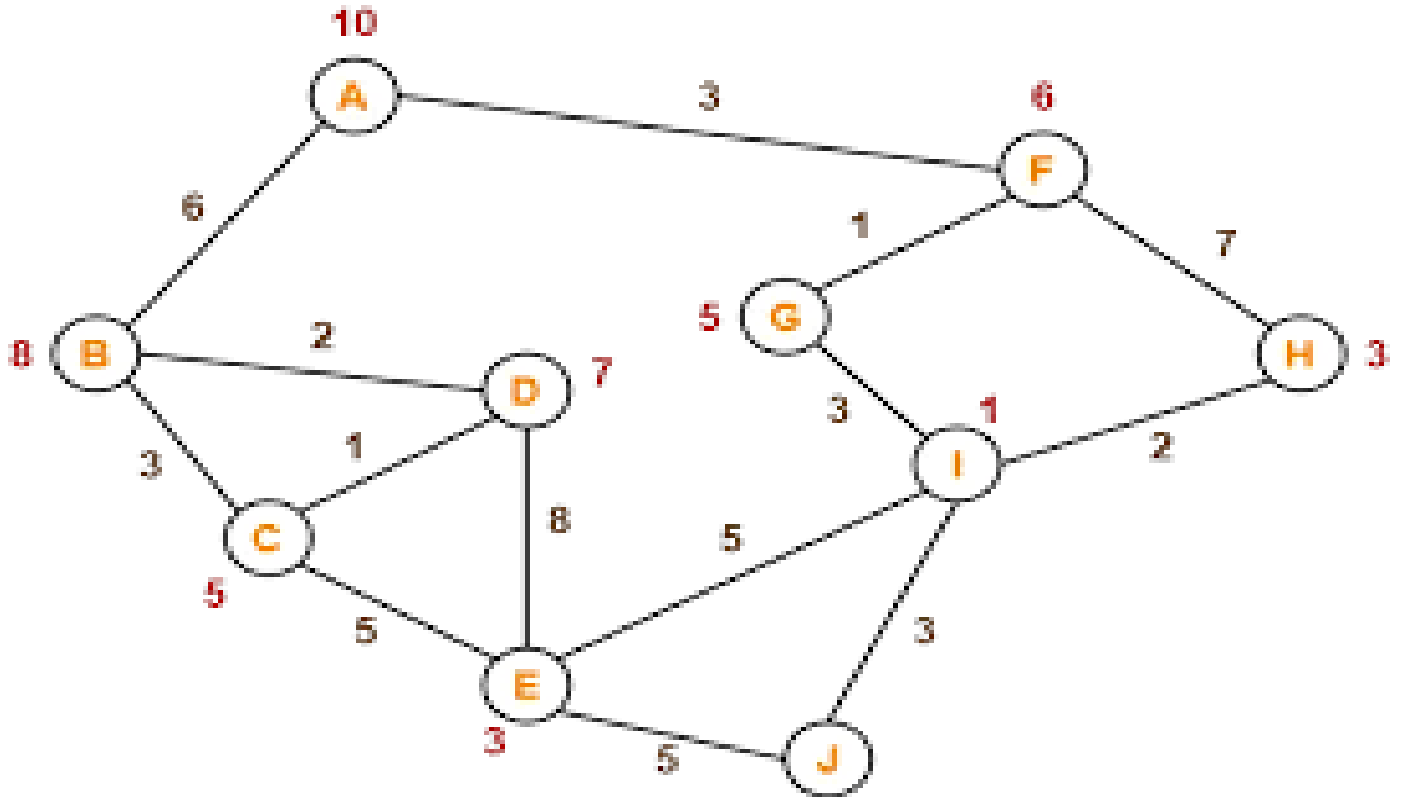
node	Heuristic distance to target node
0	20
1	16
2	6
3	10
4	4
5	0

Solve the following using A* Algorithm



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

Solve the following using A* Algorithm



State Space of the 8 Puzzle Problem

8-puzzle: 181,440 states
15-puzzle: 1.3 trilion
24-puzzle: 10^25

Search space exponential

Use Heuristics as people do

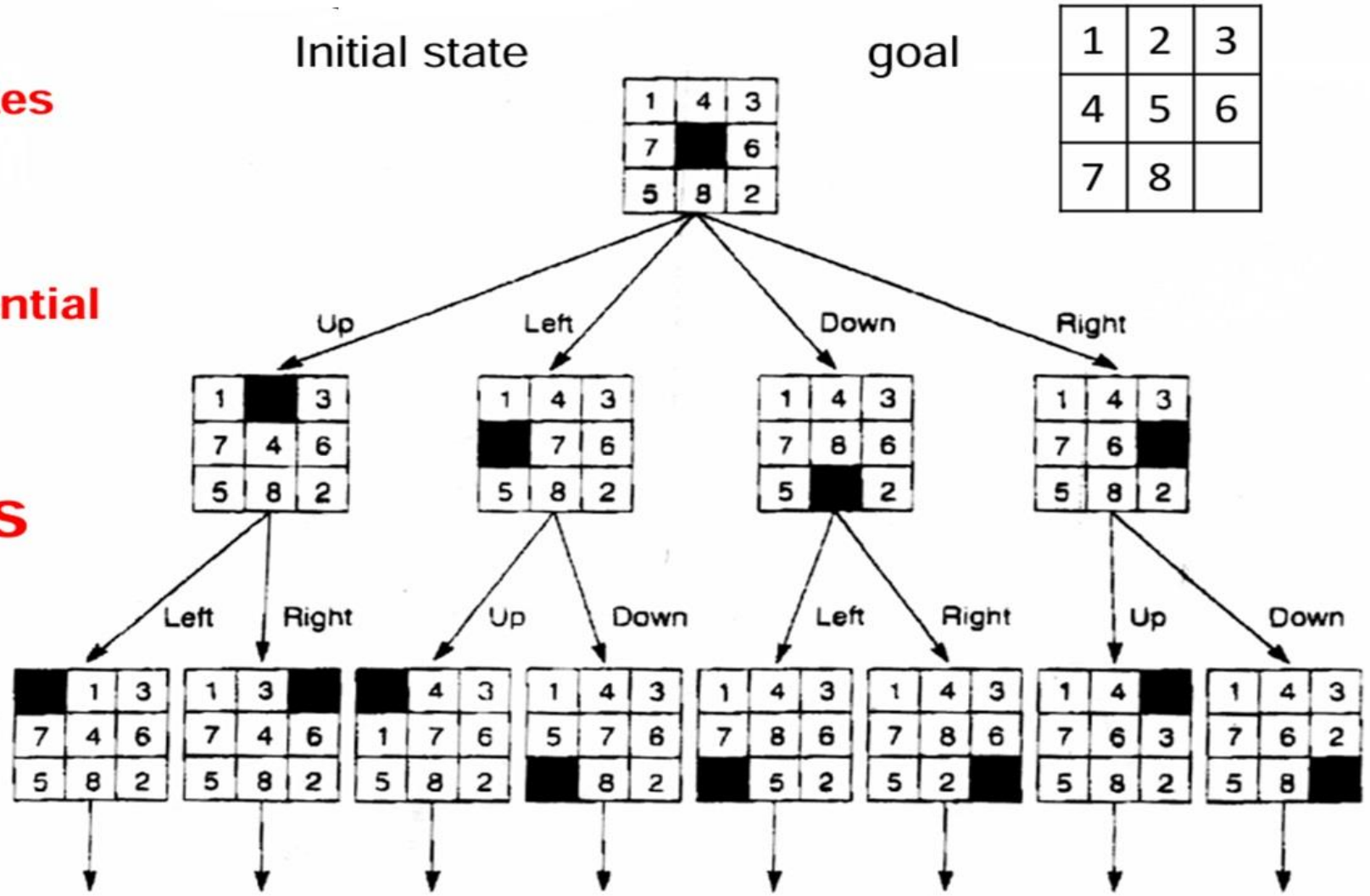


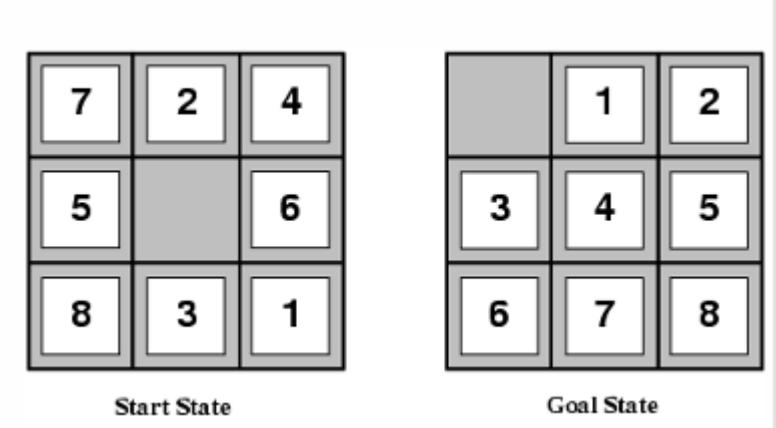
Figure 3.6 State space of the 8-puzzle generated by "move blank" operations.

What are Heuristics and Heuristic Functions

- A quick way to estimate how close we are to the goal. How close is a state to the goal..
- **Heuristic Functions h(n) guide search algorithms by estimating the cost or distance to a goal state from the current state(n).**
- **8-puzzle**
  - h1(n): number of misplaced tiles
  - h2(n): Manhattan distance
- **Path Finding on a Map**:
  - Euclidean Distance

What are Heuristics and Heuristic Functions

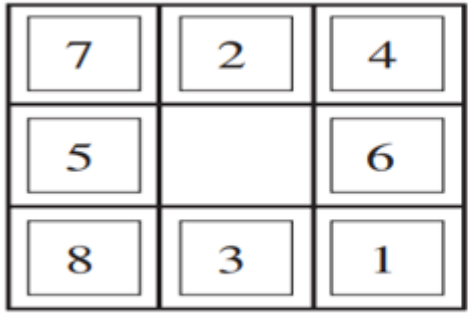
- A quick way to estimate how close we are to the goal. How close is a state to the goal..
- **Heuristic Functions h(n) guide search algorithms by estimating the cost or distance to a goal state from the current state(n).**
- **8-puzzle**
  - h1(n): number of misplaced tiles
  - h2(n): Manhattan distance, **the sum of the distances of the tiles from their goal positions**
- **Path Finding on a Map**:
  - Euclidean Distance



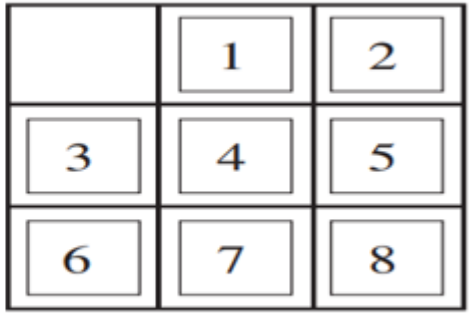
Dr.Thy
[Sou
Intelligence, 3rd Edition, Pearson,2015]

Heuristics Functions and 8 Puzzle games

- **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

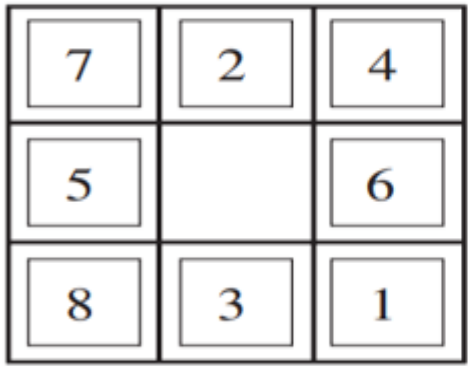
- **Figure illustrates the A typical instance of the 8-puzzle. The solution is 26 steps long.**
- **The average solution cost for a randomly generated 8-puzzle instance is about 22 steps.**
- **The branching factor is about 3. (When the empty tile is in the middle, four moves are possible; when it is 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 and 8 puzzle game

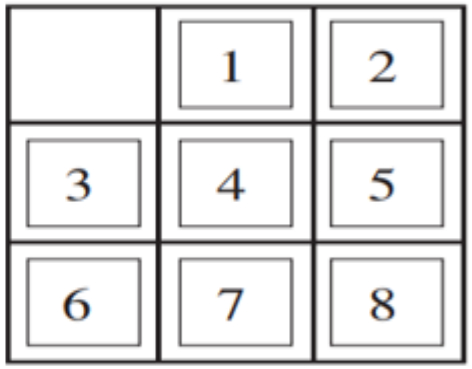
The two commonly used candidates for 8 puzzles are as follows:

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

h2 =(Manhattan distance) **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. 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

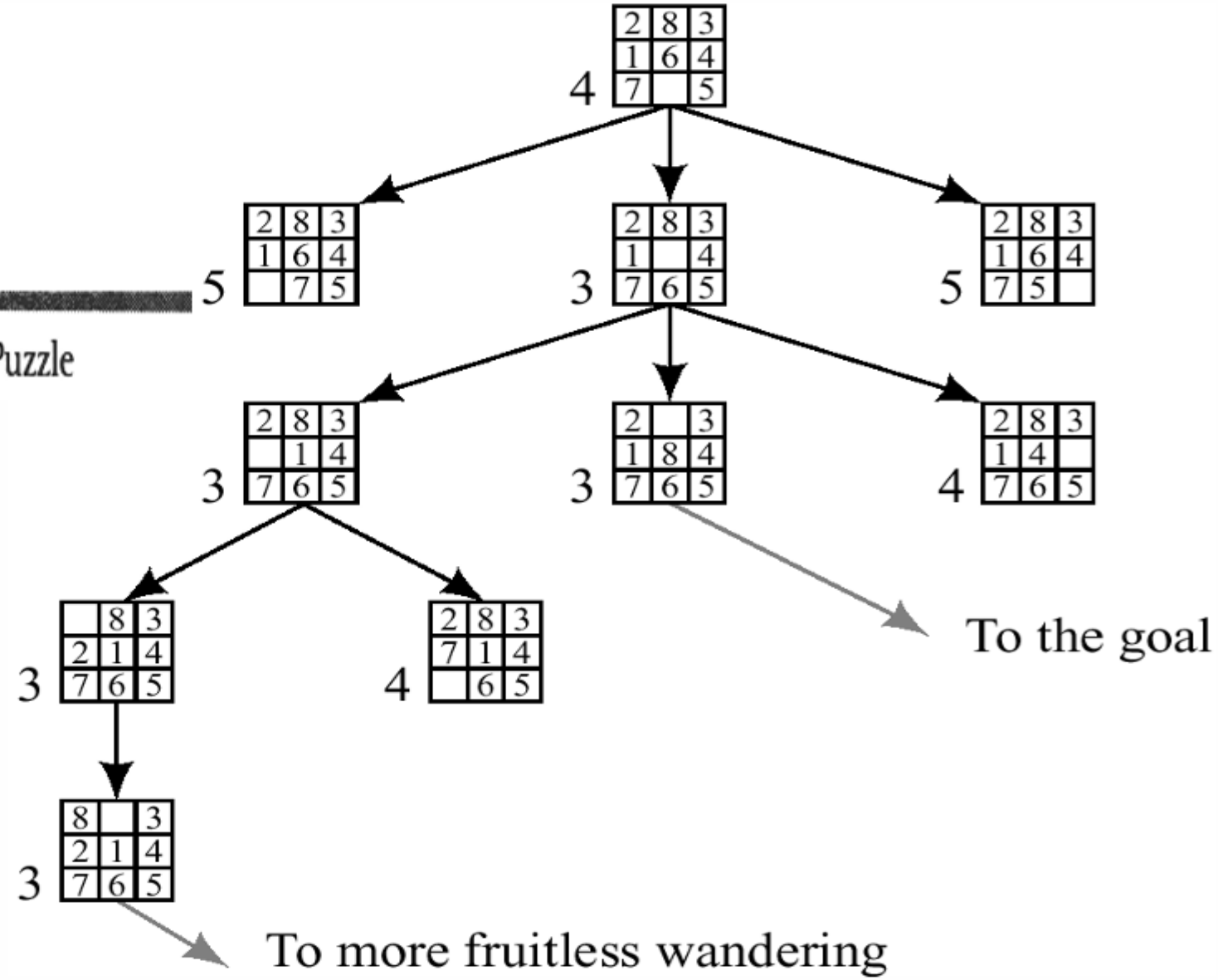


Goal State

Best-First (Greedy) Search:
f(n) = number of misplaced tiles

Figure 8.1

Start and Goal Configurations for the Eight-Puzzle



A study on Heuristic Functions

1. **The effect of heuristic accuracy on performance**
2. **Generating admissible heuristics from relaxed problems**
3. **Generating admissible heuristics from sub problems: 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.

$$h(n) = \max\{h1(n), \ldots, hm(n)\}$$

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.

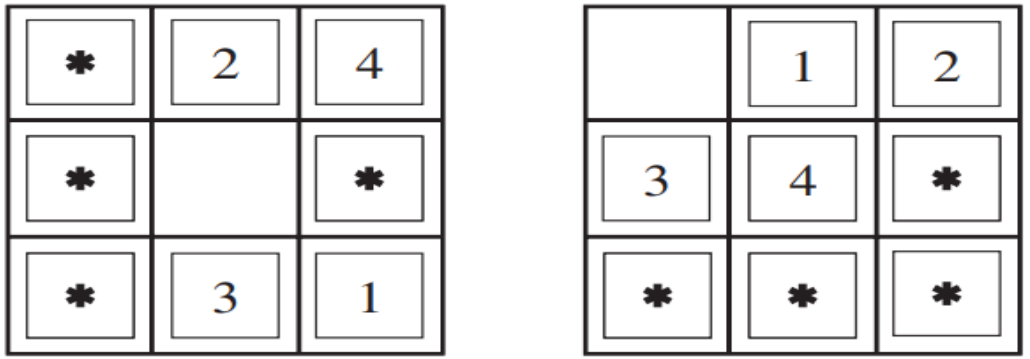


Fig: A subproblem of the 8-puzzle instance. The task is to get tiles 1, 2, 3, and 4 into their correct positions, without worrying about what happens to the other tiles.

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:

$$h(n) = c_1x_1(n) + c_2x_2(n).$$

Module3.2

Dr. Thyagaraju G S

Contents

1. **Informed Search Strategies:**
   - **a. Greedy best-first search,**
   - **b. A*search,**
   - **c. Heuristic functions.**
2. **Logical Agents:**
   - **a) Knowledge–based agents,**
   - **b) The Wumpus world,**
   - **c) Logic,**
   - **d) Propositional logic,**
   - **e) Reasoning patterns in Propositional Logic**

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.

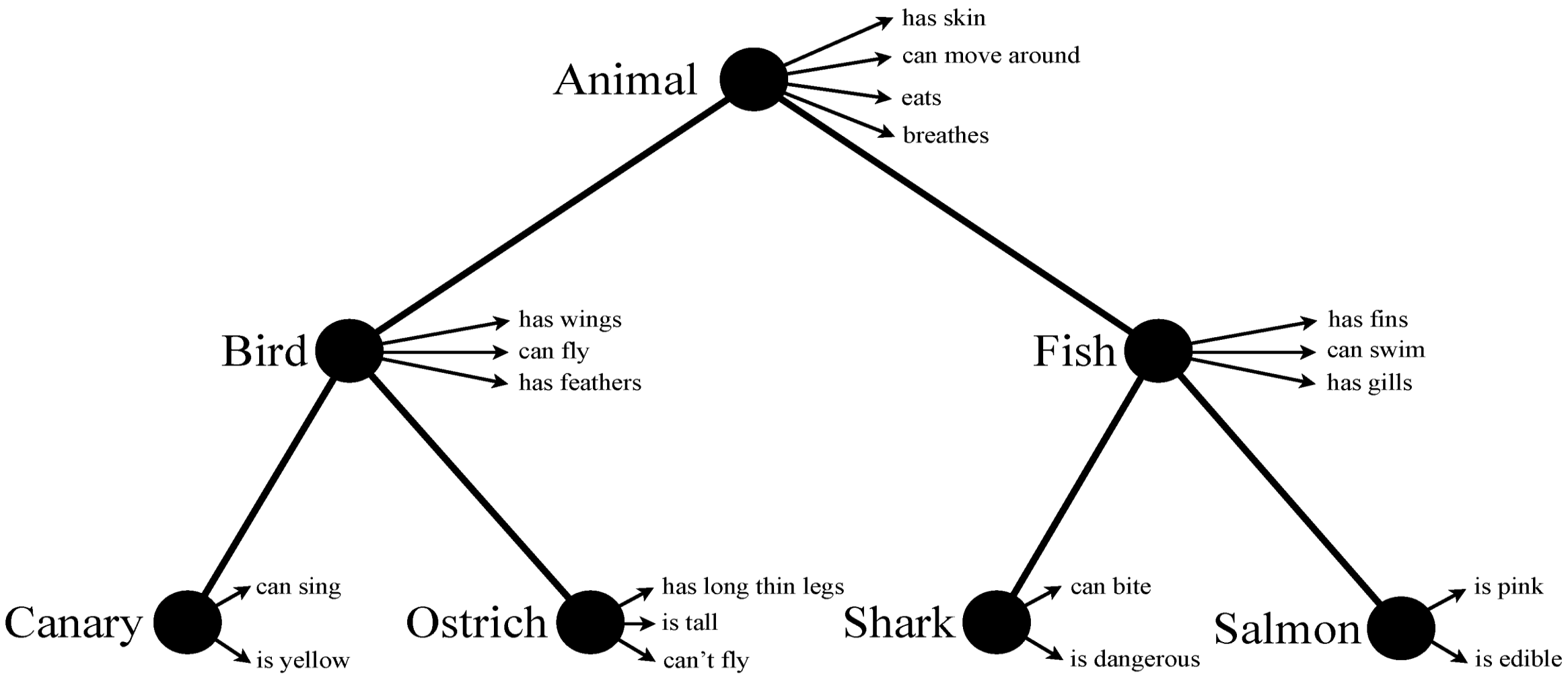
What is Knowledge Representation?

- Knowledge representation in **artificial intelligence (AI)** refers to the process of creating a **structured and formalized representation of information** in a way that can be used by a computer system to **reason, make decisions, or perform tasks.**
- The goal is to model knowledge in a manner that facilitates effective **problem-solving, learning, and communication within an AI system.**

Example : Semantic Networks

- **Semantic networks** are a graphical representation of knowledge that **uses nodes to represent concepts and arcs (edges)** to represent relationships between these concepts.
- **Each node** in the network represents an **entity or concept**, and the **arcs depict the relationships between** them.
- This form of knowledge representation is often used to model **hierarchies, associations, and dependencies**.

Knowledge Representation using Semantic Networks



Example: Propositional Logic

- In propositional logic, knowledge is represented using propositions, which are statements that can be either **true or false**. Logical operators such as **AND, OR, and NOT** are used to combine propositions.
- Consider the following knowledge about a weather prediction system:
- **P:** It is raining.
- **Q:** The sky is cloudy.
- **R:** The weather forecast predicts rain.

Now, we can represent some logical relationships:

- If the sky is cloudy **(Q)**, and the weather forecast predicts rain **(R)**, then we can infer that it might be raining **(P)**. This relationship can be represented as: $(Q \wedge R) \rightarrow P$.

- If it is not raining (NOT P), then the weather forecast predicting rain **(R)** must be false. This relationship can be represented as: $\neg P \rightarrow \neg R$.

What is Logical Reasoning?

- Logical reasoning is a cognitive process of **making inferences or drawing conclusions** based on **logical principles, rules, and relationships.**
- It involves analyzing information and using **valid deductive or inductive arguments** to reach a sound or reasonable conclusion.
- Logical reasoning is an essential aspect **of problem-solving, decision-making, and critical thinking**.

Example: Syllogism

- A syllogism is a form of deductive reasoning where a conclusion is drawn from **two given or assumed propositions** (premises).

Example: Syllogism

- **Premise 1**: All humans are mortal.
- **Premise 2:** Socrates is a human.
- **Conclusion**: **Therefore, Socrates is mortal.**

Knowledge Based Agents

- A knowledge-based agent is a type of intelligent agent that makes decisions and takes actions based on **knowledge** it possesses.
- A knowledge-based agent is characterized by its ability to represent and manipulate knowledge in a structured way, allowing it to reason, make decisions, and take actions based on the information stored in its knowledge base.

Key Components of Knowledge Base

1. **Knowledge Base (KB)**
2. **Knowledge Representation Language**
3. **Knowledge Representation Language**
4. **Axioms**
5. **TELL Operation**
6. **ASK Operation**
7. **Inference**
8. **Background Knowledge**
9. **Agent Program**

1. Knowledge Base

- The central component of a knowledge-based agent is its knowledge base. The knowledge base is a collection of **sentences** expressed in a knowledge representation language.
- **Each sentence represents an assertion about the world**. The knowledge base is where the agent stores information that it uses to make decisions and take actions.
- Sometimes we dignify a sentence with the name **axiom**, when the sentence is taken as given without being derived from other sentences.

2. Knowledge Representation Language:

- The sentences in the knowledge base are expressed in a language called a **knowledge representation language**. This language allows the agent to formally represent information about the world in a way that the agent can understand and manipulate.

3. Axioms

- Some sentences in the knowledge base may be dignified with the name "axiom," especially when they are taken as given without being derived from other sentences. Axioms are fundamental statements that serve as foundational knowledge for the agent.

4. TELL Operation:

- There is a mechanism for adding new sentences to the knowledge base. This operation is referred to as TELL. It allows the agent to incorporate new information into its knowledge base.

5. ASK Operation

- The agent needs a way to query the knowledge base to retrieve information. The standard operation for querying is referred to as ASK. It allows the agent to ask questions about what is known.

6. Inference

- Both TELL and ASK operations may involve inference, which is the process of deriving new sentences from existing ones. Inference must adhere to the requirement that answers derived from the knowledge base follow logically from the information previously TELLed to the knowledge base.

7. Background Knowledge

- The knowledge base may initially contain some background knowledge. This knowledge provides a foundational understanding of the environment in which the agent operates.

8.Agent Program

- The knowledge-based agent program outlines the overall structure of the agent. It takes a **percept (input)** and returns an **action as output**. The agent program incorporates the knowledge base and other components to facilitate decision-making and action-taking.

Agent Program : A generic knowledge-based agent. Given a **percept**, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Agents **knowledge and goals is more important**

At the **knowledge level**, we only need to specify the

- **agent's knowledge** and
- **goals to determine its behavior**.

Agents knowledge and goals is more important

- For instance, consider an **automated taxi** with the goal of transporting a passenger from **San Francisco to Marin County**. If the taxi knows that the **Golden Gate Bridge is the sole link** between these locations, we can expect it to cross the bridge, understanding that it aligns with its goal.

Agents **knowledge and goals is more important**

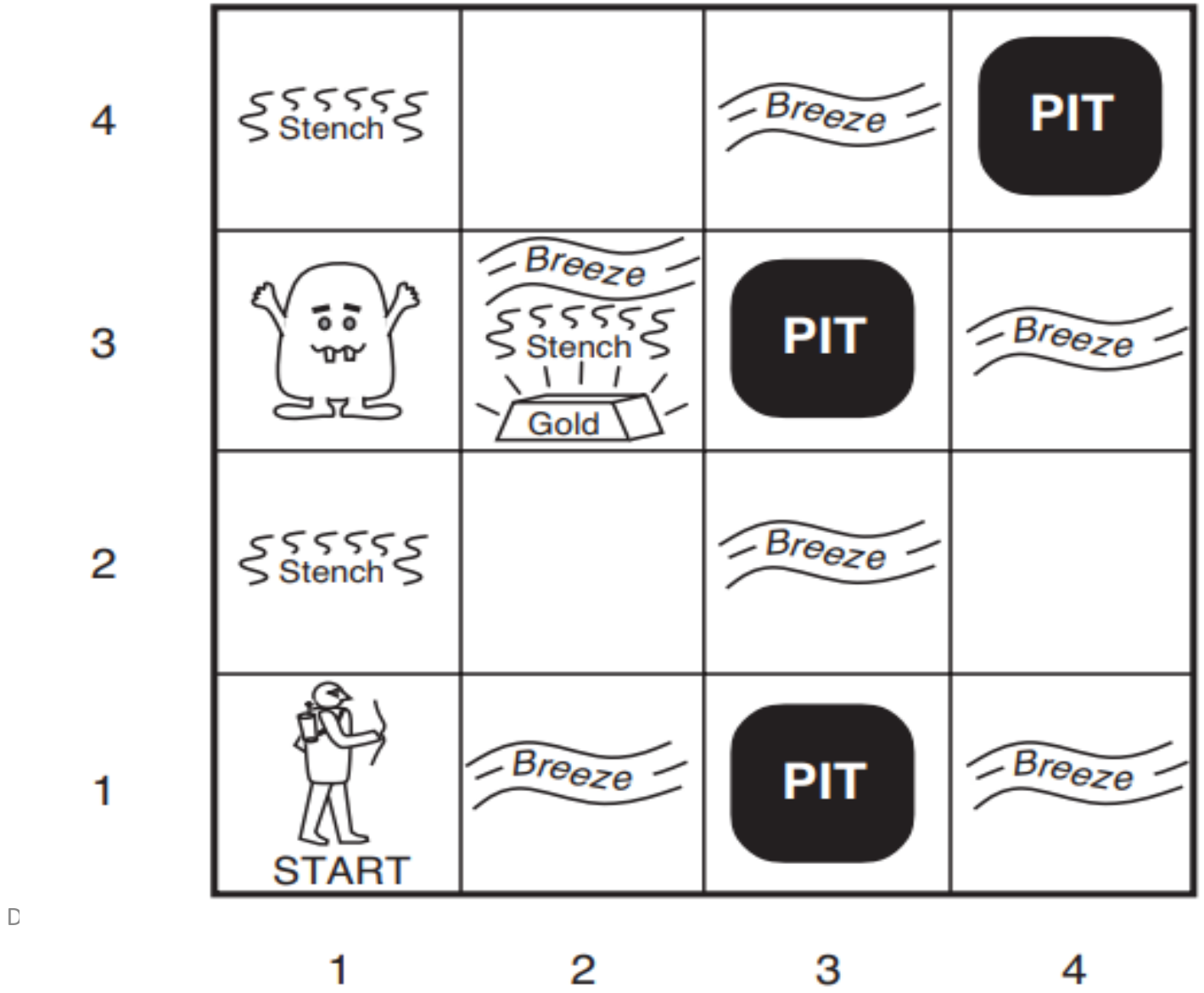
- Importantly, this analysis remains independent of the taxi's implementation details.
- Whether its geographical knowledge is represented through ***linked lists, pixel maps, or if it reasons using symbolic strings stored in registers or through neural network signal propagation***, the behavior is determined solely by its **knowledge and goals.**

2. The wumpus world

- The **wumpus** world is a cave consisting of rooms connected by passageways.
- Lurking somewhere in the cave is the terrible wumpus, a beast that eats anyone who enters its room.
- The wumpus can be shot by an agent, but the agent has only one arrow.
- Some rooms contain bottomless pits that will trap anyone who wanders into these rooms (except for the wumpus, which is too big to fall in).
- The only mitigating feature of this bleak environment is the possibility of finding a heap of gold.

A sample wumpus world

A typical wumpus world. The agent is in the bottom left corner, facing right.



PEAS description

- **Performance measure**: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.
- **Environment**: A 4 × 4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.

- **Actuators:** The agent can **move Forward, TurnLeft by 90◦ , or TurnRight by 90◦** . The agent dies a miserable death if it enters a square containing a **pit or a live** wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and **bumps** into a wall, then the agent does not move. The action **Grab** can be used to pick up the gold if it is in the same square as the agent. The action **Shoot** can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the **wumpus or hits a wall**. The agent has only one arrow, so only the first Shoot action has any effect. Finally, the action **Climb** can be used to climb out of the cave, but only from square [1,1]

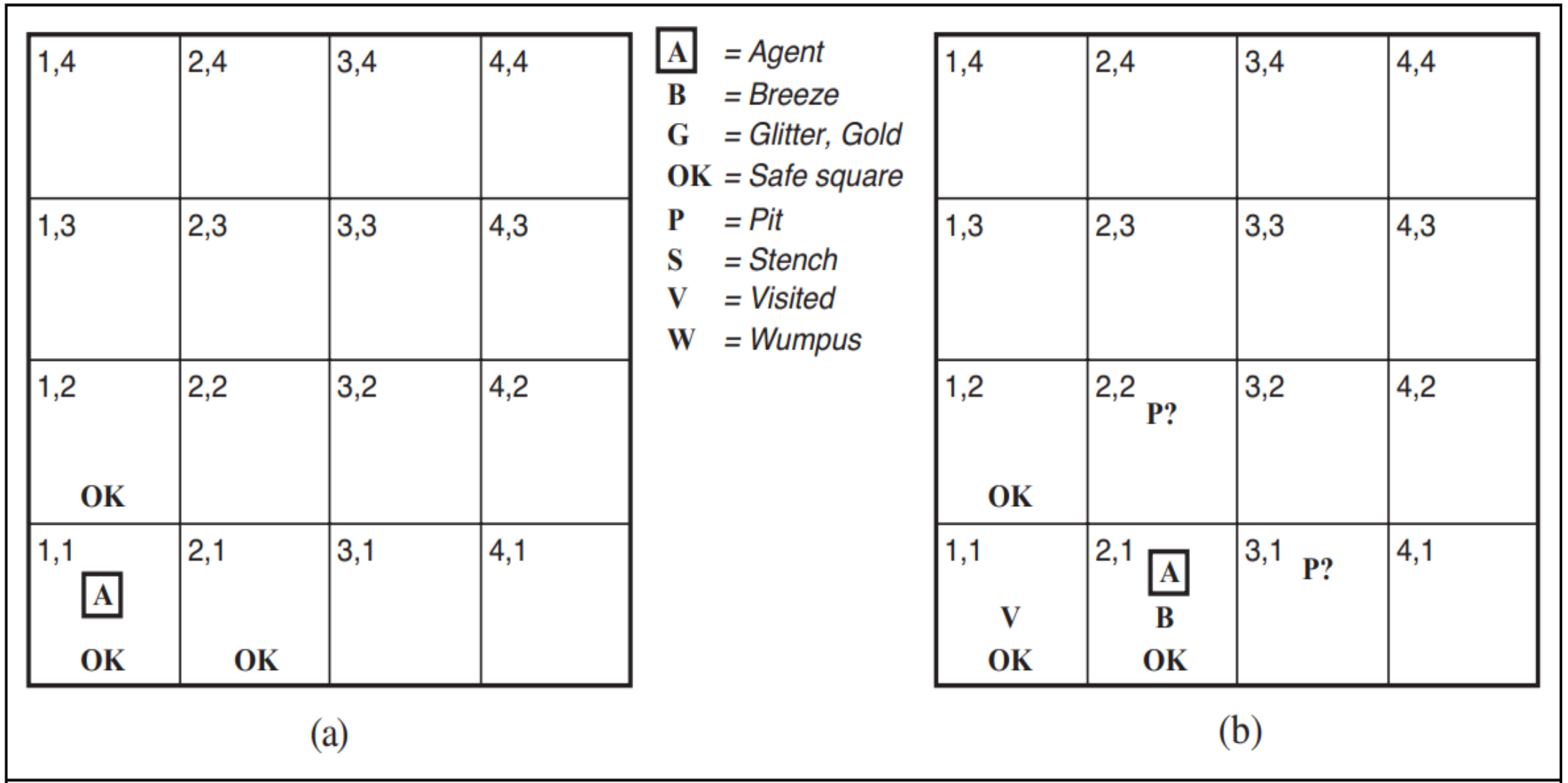
Sensors: The agent has five sensors, each of which gives a single bit of information:

1. In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a Stench.
2. In the squares directly adjacent to a pit, the agent will perceive a Breeze.
3. In the square where the gold is, the agent will perceive a Glitter.
4. When an agent walks into a wall, it will perceive a Bump.
5. When the wumpus is killed, it emits a woeful Scream that can be perceived anywhere in the cave.

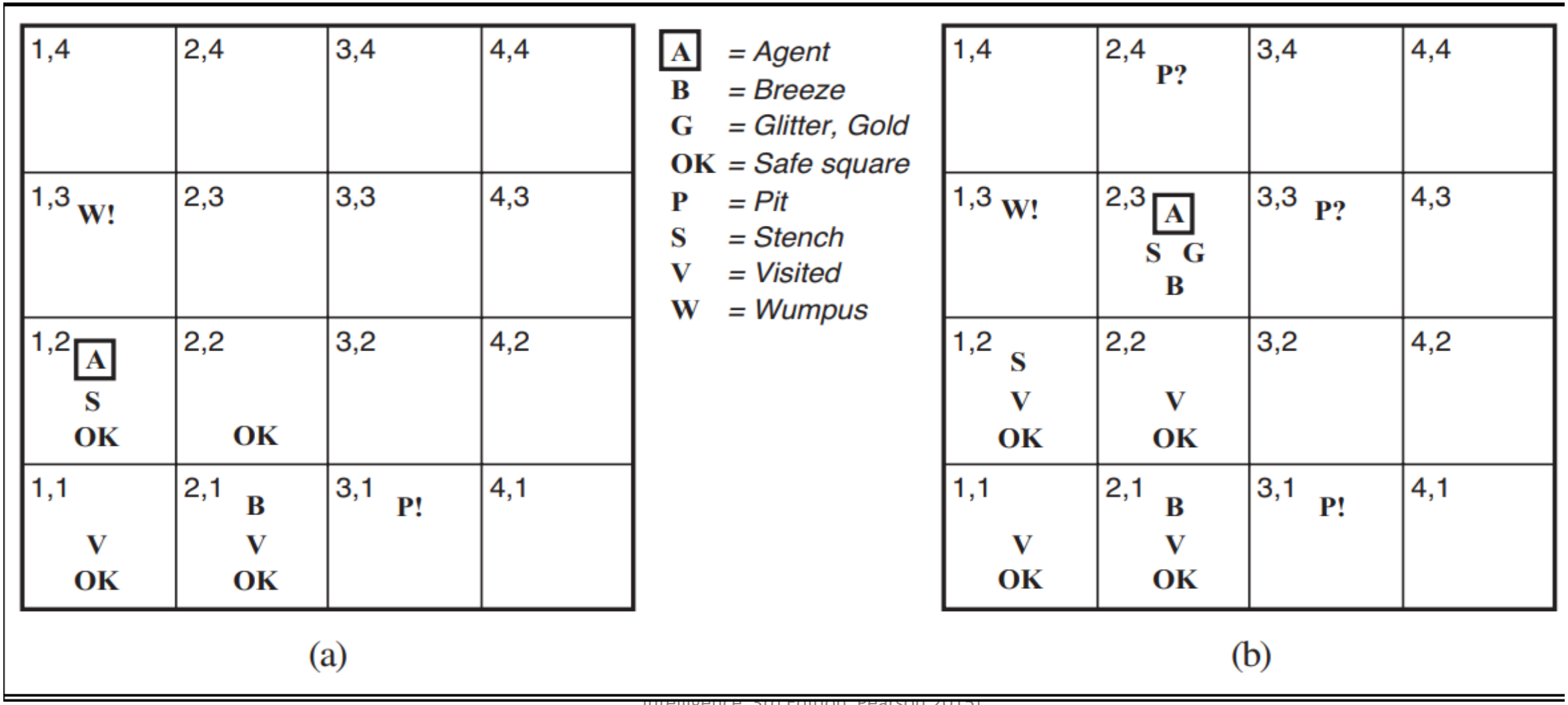
5 Percept Symbols : [Stench, Breeze, Glitter, Bump, Scream].

The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a **stench** and a **breeze**, but no **glitter**, **bump**, or **scream**, the agent program will get **[Stench, Breeze, None, None, None].**

The first step taken by the agent in the wumpus world.
(a) The initial situation, after percept [None, None, None, None, None].
(b) After one move, with percept [None, Breeze, None, None, None]



Two later stages in the progress of the agent.
(a) After the third move, with percept [Stench, None, None, None, None].
(b) After the fifth move, with percept [Stench, Breeze, Glitter , None, None].



- The agent's initial knowledge base contains the rules of the environment, as described previously; in particular, it knows that it is in [1,1] and that [1,1] is a safe square; we denote that with an "A" and "OK," respectively, in square [1,1].
- The first percept is [None, None, None, None, None], from which the agent can conclude that its neighboring squares, [1,2] and [2,1], are free of dangers—they are OK. Figure 7.3(a) shows the agent's state of knowledge at this point.
- A cautious agent will move only into a square that it knows to be OK. Let us suppose the agent decides to move forward to [2,1]. The agent perceives a breeze (denoted by "B") in [2,1], so there must be a pit in a neighboring square. The pit cannot be in [1,1], by the rules of the game, so there must be a pit in [2,2] or [3,1] or both. The notation "P?" in Figure 7.3(b) indicates a possible pit in those squares. At this point, there is only one known square that is OK and that has not yet been visited. So the prudent agent will turn around, go back to [1,1], and then proceed to [1,2].

- The agent perceives a stench in [1,2], resulting in the state of knowledge shown in Figure 7.4(a). The stench in [1,2] means that there must be a wumpus nearby. But the wumpus cannot be in [1,1], by the rules of the game, and it cannot be in [2,2] (or the agent would have detected a stench when it was in [2,1]). Therefore, the agent can infer that the wumpus is in [1,3]. The notation W! indicates this inference. Moreover, the lack of a breeze in [1,2] implies that there is no pit in [2,2]. Yet the agent has already inferred that there must be a pit in either [2,2] or [3,1], so this means it must be in [3,1]. This is a fairly difficult inference, because it combines knowledge gained at different times in different places and relies on the lack of a percept to make one crucial step.

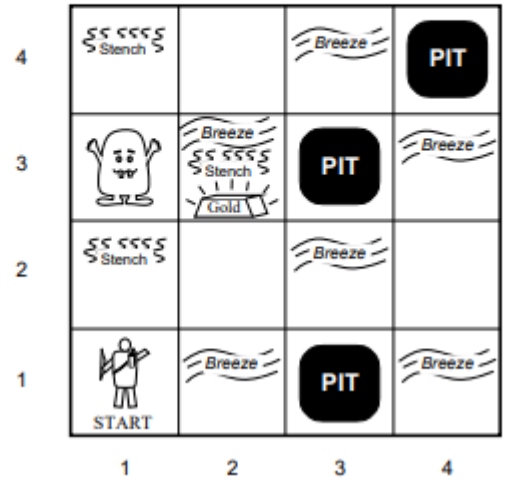
- The agent has now proved to itself that there is neither a pit nor a wumpus in [2,2], so it is OK to move there. We do not show the agent's state of knowledge at [2,2]; we just assume that the agent turns and moves to [2,3], giving us Figure 7.4(b). In [2,3], the agent detects a glitter, so it should grab the gold and then return home.

Wumpus World PAGE description

Percepts Breeze, Glitter, Smell

Actions Left turn, Right turn,
Forward, Grab, Release, Shoot

Goals Get gold back to start
without entering pit or wumpus square



Environment

Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter if and only if gold is in the same square
Shooting kills the wumpus if you are facing it
Shooting uses up the only arrow
Grabbing picks up the gold if in the same square
Releasing drops the gold in the same square

Wumpus world characterization

Is the world deterministic?? Yes—outcomes exactly specified

Is the world fully accessible?? No—only local perception

Is the world static?? Yes—Wumpus and Pits do not move

Is the world discrete?? Yes

2.3 Logic

- Logic is a fundamental component of artificial intelligence (AI). It enables machines to understand and represent data and knowledge in a reasoning way.
- Logical reasoning is a process of inferring a conclusion based on observations or data.
- It is concerned with the principles of reasoning and how conclusions can be drawn from given premises.
- Logic provides the theoretical foundation for reasoning.

Types of logic

Logics are characterized by what they commit to as "primitives"

Ontological commitment: what exists—facts? objects? time? beliefs?

Epistemological commitment: what states of knowledge?

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief 0...1
Fuzzy logic	degree of truth	degree of belief 0...1

Logic in general

Logics are formal languages for representing information
such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the "meaning" of sentences;
i.e., define truth of a sentence in a world

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; $x2 + y >$ is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where $x = 7,\ y = 1$
$x + 2 \geq y$ is false in a world where $x = 0,\ y = 6$

[Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015]

Model or Possible World in logic

- The semantics defines the truth of each sentence with respect to each possible world.
- In standard logics, every sentence must be either **true or false** in each possible world—there is no “in between.”
- When we need to be precise, we use the term model in place of “possible world.”
- If a sentence α is true in model m, we say that m satisfies α or sometimes m is a model of α. We use the notation M(α) to mean the set of all models of α

Logical entailment between sentences

- A **sentence follows logically from another sentence**. In mathematical notation, we write $\alpha \models \beta$
- Entailment in logic refers to a relationship between propositions where the truth of one proposition necessarily guarantees the truth of another.
- If proposition **A entails proposition B**, it means that whenever A is true, B must also be true. In formal logic, this relationship is often represented as $A \models B$.

Formal Definition

The formal definition of entailment is this:

- **α |= β if and only if**, in every model in which **α is true, β** is also true.

Using the notation just introduced, we can write

- **α |= β if and only if** $M(\alpha) \subseteq M(\beta)$.
- **Example** :
  - The relation of entailment is familiar from **arithmetic**; the idea that the **sentence x = 0** entails the sentence **xy = 0**.
  - Obviously, in any model where **x is zero**, it is the case that **xy is zero** (**regardless of the value of y**)

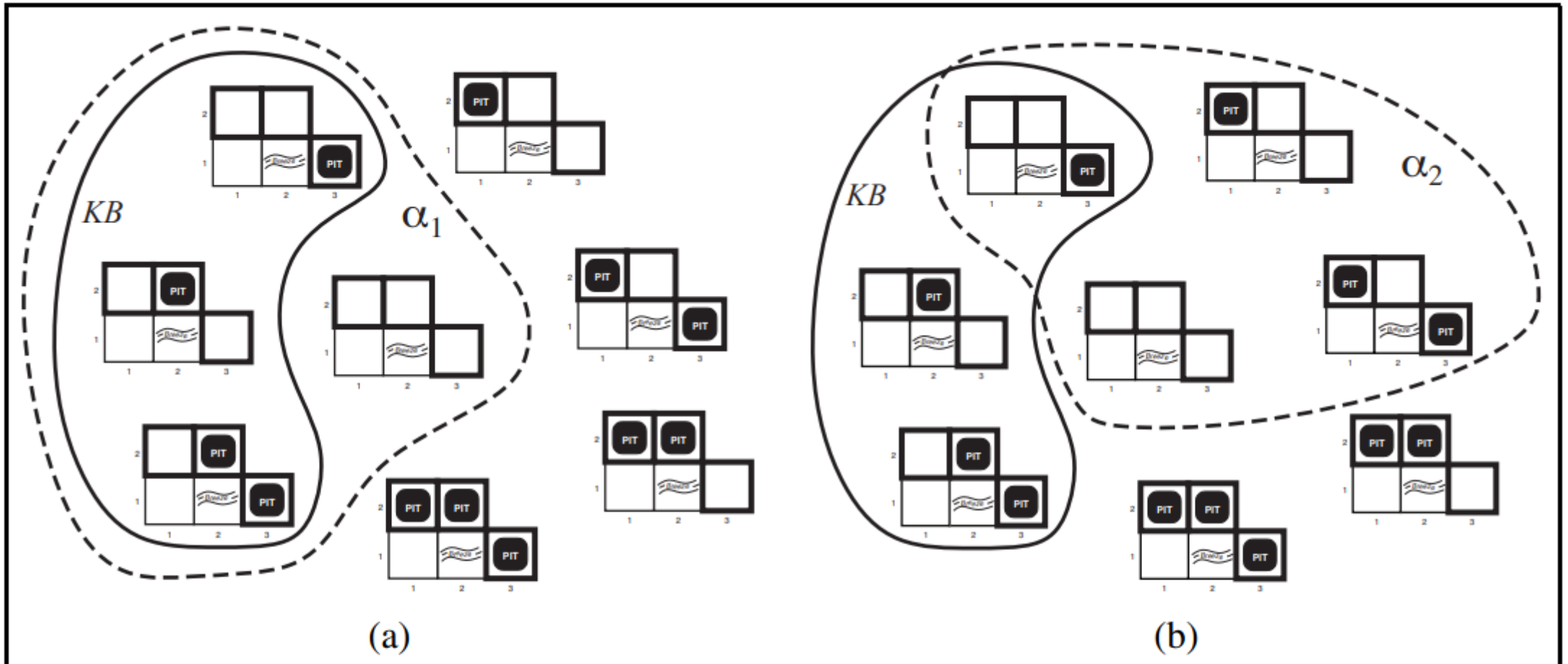
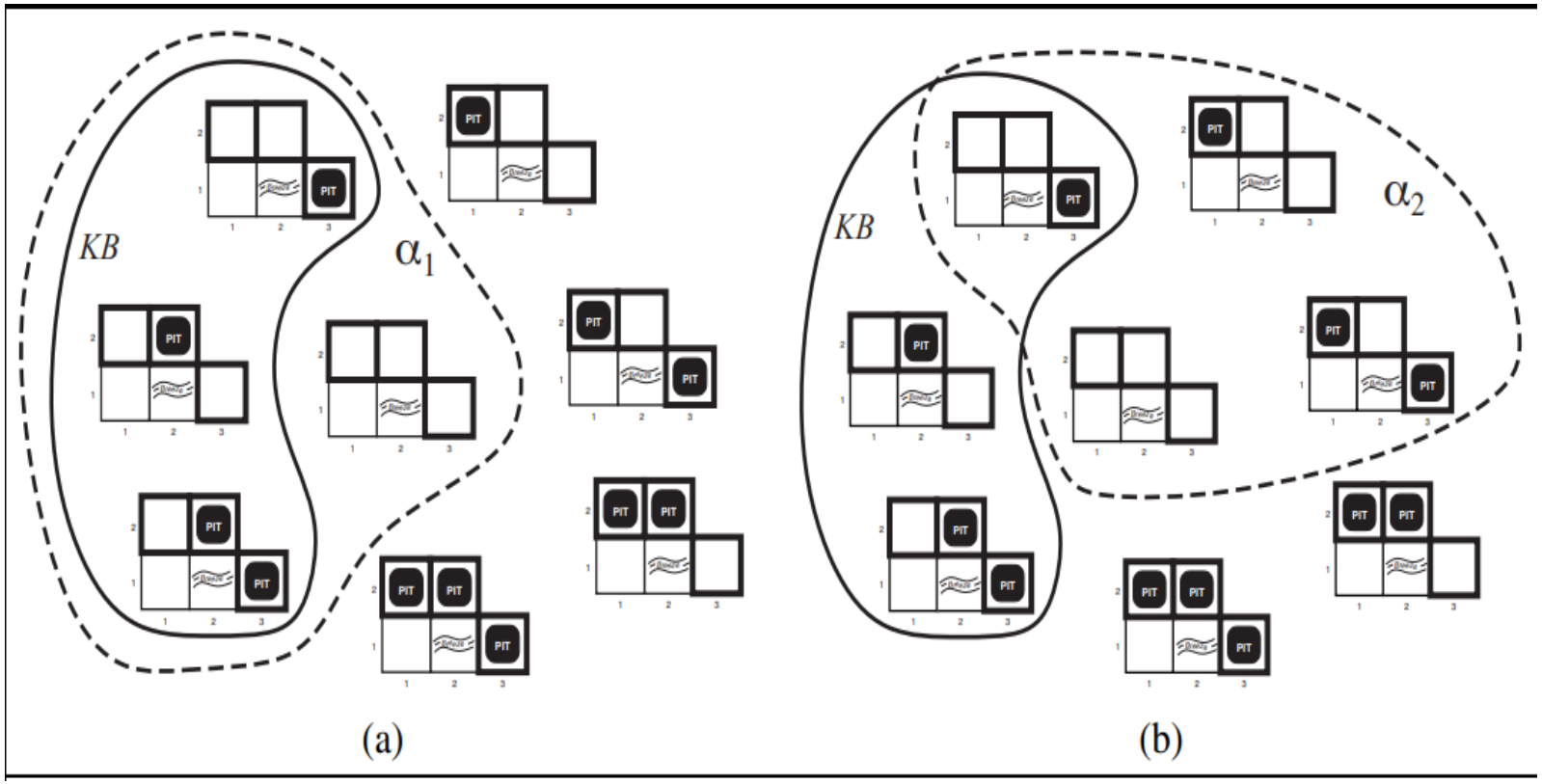
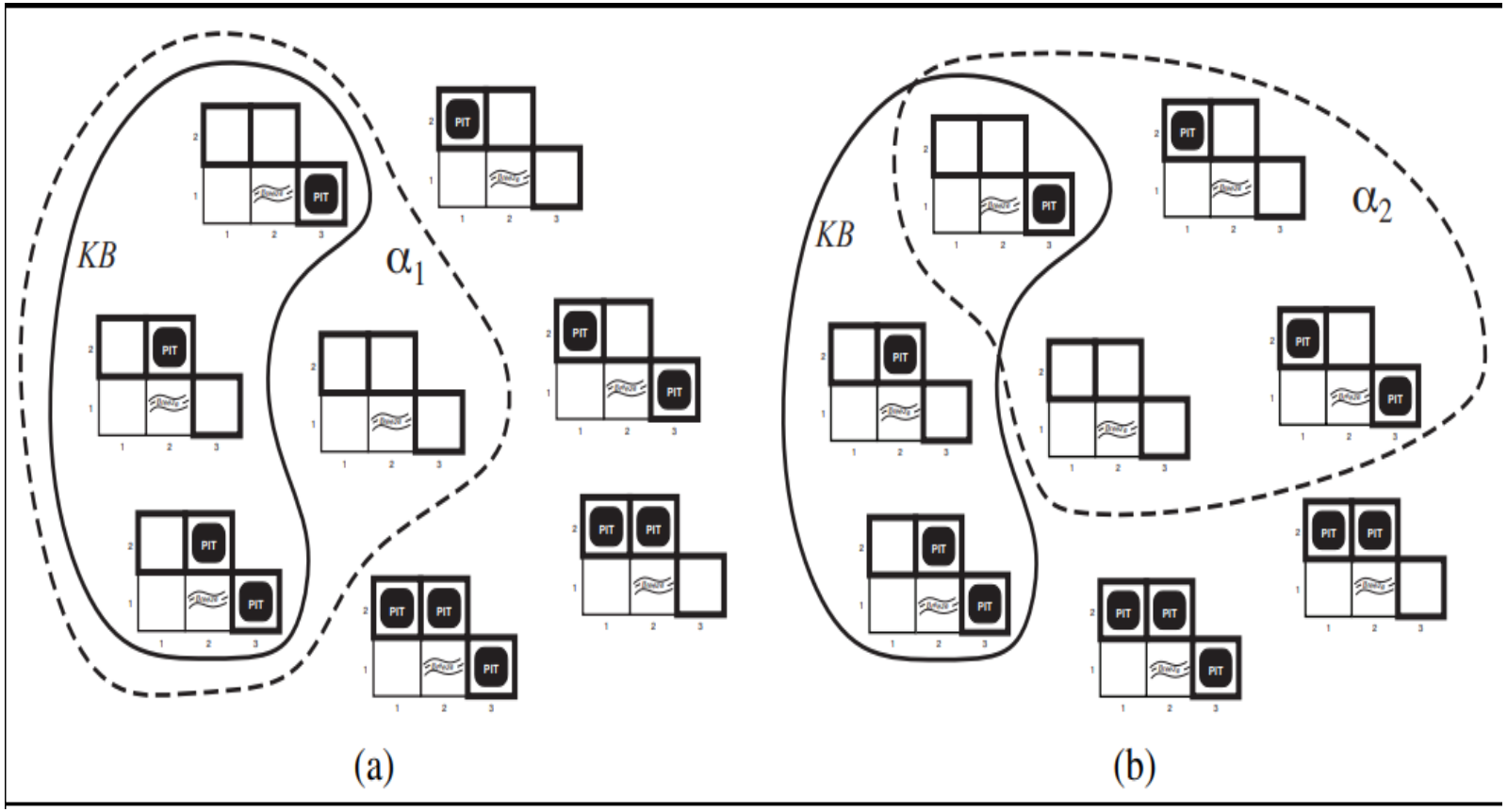


Figure 7.5 Possible models for the presence of pits in squares [1,2], [2,2], and [3,1]. The KB corresponding to the observations of nothing in [1,1] and a breeze in [2,1] is shown by the solid line. (a) Dotted line shows models of α_1 (no pit in [1,2]). (b) Dotted line shows models of α_2 (no pit in [2,2]).

[Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015]



The KB is false in models that contradict what the agent knows — for example, the **KB is false** in any model in which **[1,2] contains a pit**,



Now let us consider two possible conclusions:

α1 = “There is no pit in [1,2].”

α2 = “There is no pit in [2,2].”

By inspection, we see the following

KB |= α1

$$KB \not\models \alpha_2$$

Entailment

$KB \models \alpha$

Knowledge base KB entails sentence α
if and only if
α is true in all worlds where KB is true

E.g., the KB containing "the Giants won" and "the Reds won"
entails "Either the Giants won or the Reds won"

Inference

- The inference algorithm illustrated in Figure is called model checking, because it enumerates all possible models to check that α is true in all models in which KB is true, that is, that M(KB) ⊆ M(α).
- Entailment is comparable to finding the needle within the haystack, while inference is akin to the process of **locating it**.
- This distinction is represented in formal notation:
  - if an inference algorithm "**i**" can derive **α** from **KB**, it is denoted as **KB ⊢i α**, which can be read as :
  - "**α is derived from KB by i**" or "**i derives α from KB**."

Note :

- An inference algorithm that derives only **entailed sentences** is called **sound or truth preserving**.
- An **inference algorithm** is complete if it can derive any sentence that is entailed.
- *if KB is true in the real world, then any sentence α derived from KB by a sound inference procedure is also true in the real world*.
- grounding—the connection between logical reasoning processes and the real environment in which the agent exists

Correspondence between world and representation

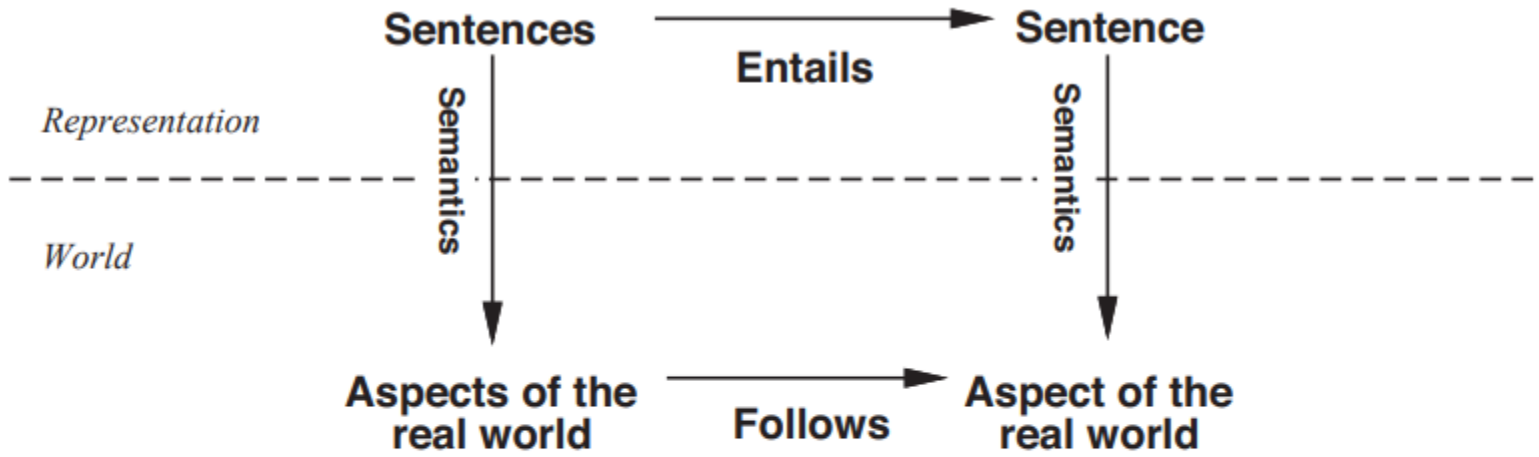


Figure 7.6 Sentences are physical configurations of the agent, and reasoning is a process of constructing new physical configurations from old ones. Logical reasoning should ensure that the new configurations represent aspects of the world that actually follow from the aspects that the old configurations represent.

Inference

$KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i

Soundness: i is sound if

whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$

Completeness: i is complete if

whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

Propositional Logic

- Propositional logic, also known as **sentential logic or propositional calculus**, is a branch of formal logic that deals with the logical relationships between **propositions** (statements or sentences) without considering the internal structure of the propositions.
- In propositional logic, **propositions are considered as atomic units**, and logical operations are applied to these propositions to form more complex statements.

Some key elements and concepts in propositional logic:

1. **Propositions:** These are statements that can be either true or false. Propositions are represented by variables, typically denoted by letters (p, q, r, etc.).
2. **Logical Connectives:** These are symbols that combine propositions to form more complex statements. The main logical connectives in propositional logic include:
    1. **Conjunction (∧):** Represents "and." The compound proposition "p ∧ q" is true only when both p and q are true.
    2. **Disjunction (∨):** Represents "or." The compound proposition "p ∨ q" is true when at least one of p or q is true.
    3. **Negation (¬):** Represents "not." The compound proposition "¬p" is true when p is false.
    4. **Implication (→):** Represents "if...then." The compound proposition "p → q" is false only when p is true and q is false.
    5. **Biconditional (↔):** Represents "if and only if." The compound proposition "p ↔ q" is true when p and q have the same truth value.
3. **Truth Tables:** Truth tables are used to systematically list all possible truth values for a compound proposition based on the truth values of its constituent propositions. Truth tables help determine the truth conditions for complex statements.
4. **Logical Equivalence:** Two propositions are logically equivalent if they have the same truth values for all possible combinations of truth values of their component propositions.

Syntax

- The syntax of propositional logic defines the **allowable sentences**.
- The atomic sentences consist of a **single proposition symbol**. Each such symbol stands for a proposition that can be true or false.
- We use symbols that start with an **uppercase letter and may contain other letters or subscripts**, for example: P, Q, R, $W_{1,3}$ and North.
- **$W_{1,3}$** to stand for the proposition that the wumpus is in [1,3]
- **Complex sentences** are constructed from simpler sentences, using parentheses and logical connectives.

There are five connectives in common use

All sentences are constructed from atomic sentences and the five connectives

1. NEGATION ¬ (not).
2. CONJUNCTION ∧ (and).
3. DISJUNCTION ∨ (or).
4. IMPLICATION ⇒ (implies).
5. BICONDITIONAL ⇔ (if and only if)

$$
\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} &\rightarrow (\ \textit{Sentence}\ ) \mid [\ \textit{Sentence}\ ] \\
&\mid \neg\ \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence}
\end{aligned}
$$

OPERATOR PRECEDENCE : $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Figure 7.7 A BNF (Backus–Naur Form) grammar of sentences in propositional logic, along with operator precedences, from highest to lowest.

Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence

If S_1 and S_2 is a sentence, $S_1 \wedge S_2$ is a sentence

If S_1 and S_2 is a sentence, $S_1 \vee S_2$ is a sentence

If S_1 and S_2 is a sentence, $S_1 \Rightarrow S_2$ is a sentence

If S_1 and S_2 is a sentence, $S_1 \Leftrightarrow S_2$ is a sentence

Semantics

- The semantics defines the rules for determining the truth of a sentence with respect to a particular model.
- In propositional logic, a model simply **fixes the truth value—true or false—for every proposition symbol**.
- For example, if the sentences in the knowledge base make use of the proposition symbols $P_{1,2}$, $P_{2,2}$, and $P_{3,1}$, then one possible model is
- $m1 = \{P_{1,2} = \text{false}, P_{2,2} = \text{false}, P_{3,1} = \text{true}\}$

Rule for Atomic Sentences

- **True** is true in every model and **False** is false in every model.

For complex sentences, we have five rules, which hold for any subsentences P and Q in any model m (here "iff" means "if and only if"):

- **¬P** is true iff P is false in m.
- **P ∧ Q** is true iff both **P and Q** are true in m.
- **P ∨ Q** is true iff either **P or Q** is true in m.
- **P ⇒ Q** is true unless **P is true and Q** is false in m.
- **P ⇔ Q** is true iff P and Q are both true or both false in m.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *true*.

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g.

A	B	C
$True$	$True$	$False$

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false				
$S_1 \wedge S_2$	is true iff	S_1	is true and	S_2	is true		
$S_1 \vee S_2$	is true iff	S_1	is true or	S_2	is true		
$S_1 \Rightarrow S_2$	is true iff	S_1	is false or	S_2	is true		
i.e.,	is false iff	S_1	is true and	S_2	is false		
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true and	$S_2 \Rightarrow S_1$	is true		

A simple knowledge base : Wumpus World

Symbols for each [x, y] location:

- $P_{x,y}$ is true if there is a pit in [x, y].
- $W_{x,y}$ is true if there is a wumpus in [x, y], dead or alive.
- $B_{x,y}$ is true if the agent perceives a breeze in [x, y].
- $S_{x,y}$ is true if the agent perceives a stench in [x, y].

A simple knowledge base : Wumpus World

- There is no pit in [1,1]:

  $R_1 : \neg P_{1,1}$.

- A square is breezy if and only if there is a pit in a neighboring square. This has to be stated for each square; for now, we include just the relevant squares:

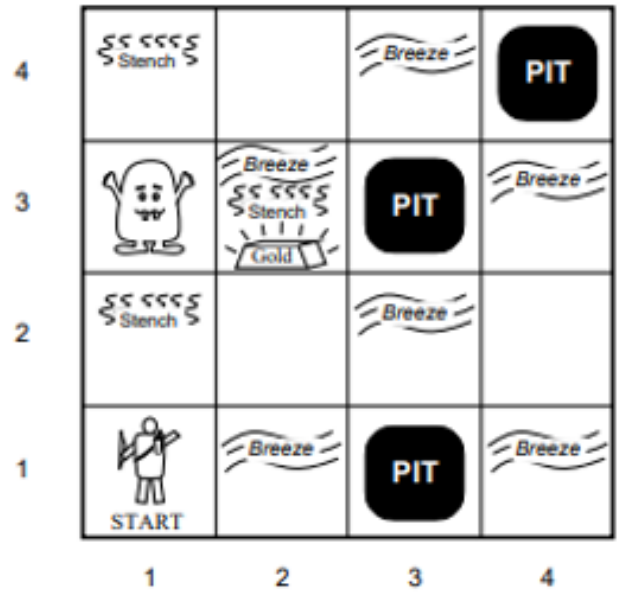
  R2 : $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.

  R3 : $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$.

- The preceding sentences are true in all Wumpus worlds**:**

  R4 : $\neg B_{1,1}$

  R5 : $B_{2,1}$.



Propositional inference: Enumeration method

Let $\alpha = A \vee B$ and $KB = (A \vee C) \wedge (B \vee \neg C)$

Is it the case that $KB \models \alpha$?
Check all possible models—α must be true wherever KB is true

A	B	C	$A \vee C$	$B \vee \neg C$	KB	α
$False$	$False$	$False$				
$False$	$False$	$True$				
$False$	$True$	$False$				
$False$	$True$	$True$				
$True$	$False$	$False$				
$True$	$False$	$True$				
$True$	$True$	$False$				
$True$	$True$	$True$				

Dr.Thyagaraju G S , Professor,Dept of CSE,SDMIT,Ujire -574240
[Source Book: Stuart J. Russell and Peter Norvig, Artificial Intelligence, 3rd Edition, Pearson,2015]