

ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028



A T M E
College of Engineering

DEPARTMENT OF COMPUTER SCIENCE AND DESIGN



LECTURE NOTES

COURSE: MULTIMEDIA SYSTEM AND DESIGN

COURSE CODE: BCS515B

SEMESTER: V

COURSE COORDINATOR: Mrs. DIVYA N

(ACADEMIC YEAR 2025-26)

INSTITUTIONAL VISION AND MISSION

Vision

Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

DEPARTMENT VISION AND MISSION

Vision

To be a global leader in Computer Science and Design Engineering striving for design excellence, cultural awareness, a profound commitment to environmental stewardship, and societal responsibility.

Mission

M1: To establish a technology-enabled experiential learning environment, prioritizing and cultivating problem-solving and design thinking skills among students.

M2: To foster collaboration with industries, research and development organizations, jointly addressing socially relevant challenges in Computer Science with a core emphasis on design.

Program Educational Objective (PEO's)

- Demonstrate the proficiency and adaptability needed to seamlessly integrate with the latest advancements and changes in Computer Science and Design.
- Our graduates will possess the adaptability to explore various career paths, such as Research & Development, Manufacturing, and Entrepreneurship with independent design firms, Design Services, and Consultancies.
- Our students will be well-prepared to excel in collaborative, multi-disciplinary environments, showcasing their versatility and skills.

Program Specific Outcomes (PSO's)

- To develop stand-alone, embedded, and web-based solutions with easy-to-operate interfaces using software engineering practices and contemporary programming languages.
- Design and develop computer-based systems in various areas of Multimedia, Graphics data visualization and computer vision.

MODULE - 3

3.5 INFORMED (HEURISTIC) SEARCH STRATEGIES

A Heuristic technique helps in solving problems, even though there is no guarantee that it will never lead in the wrong direction. There are heuristics of every general applicability as well as domain specific. The strategies are general purpose heuristics. In order to use them in a specific domain they are coupled with some domain specific heuristics. There are two major ways in which domain - specific, heuristic information can be incorporated into rule-based search procedure. A heuristic function is a function that maps from problem state description to measures desirability, usually represented as number weights. The value of a heuristic function at a given node in the search process gives a good estimate of that node being on the desired path to solution.

3.5.1 Greedy best-first search

Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function:

$$f(n) = h(n)$$

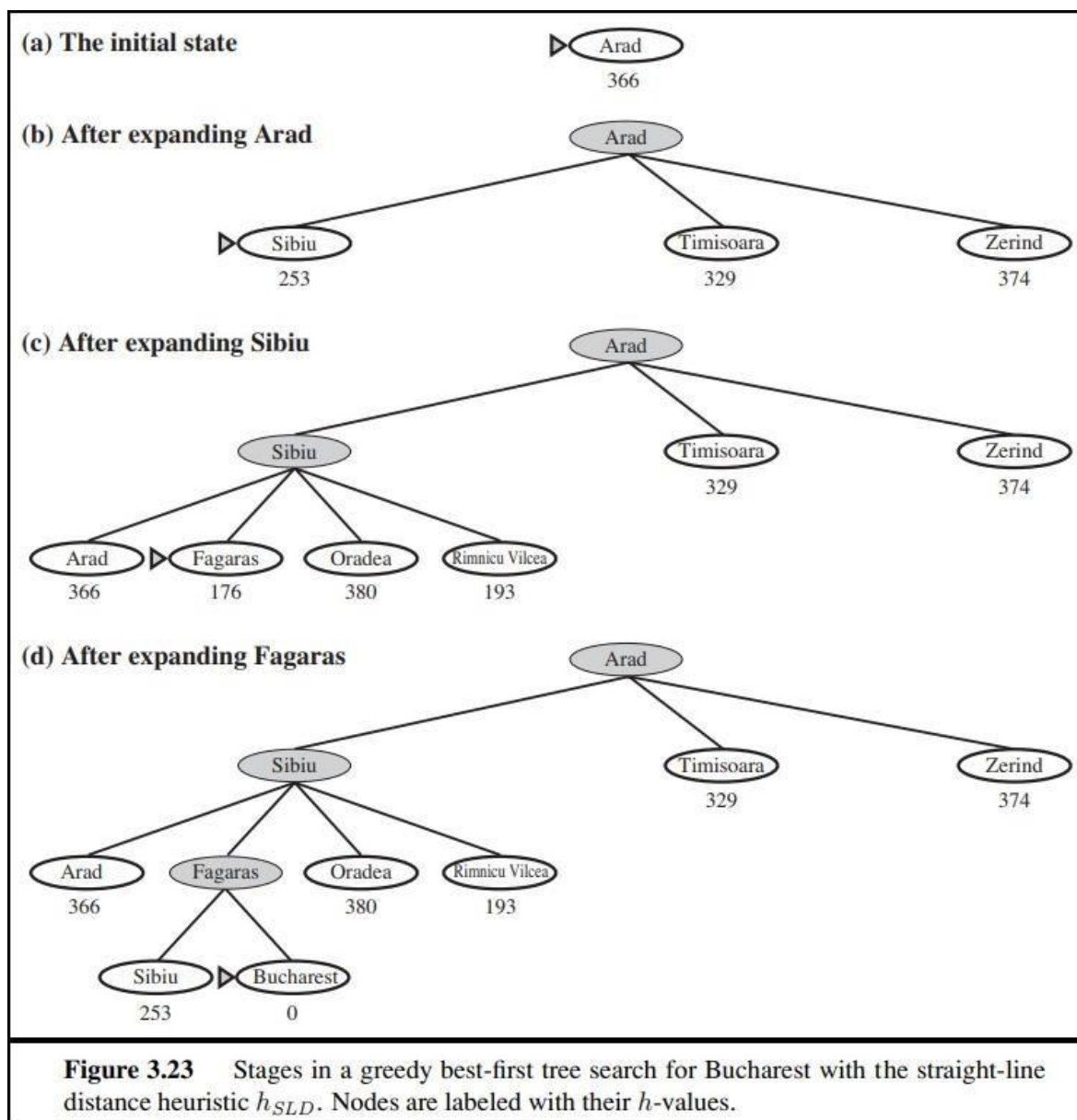
Taking the example of Route-finding problems in Romania, the goal is to reach Bucharest starting from the city Arad. We need to know the straight-line distances to Bucharest from various cities as shown in Figure 3.22. For example, the initial state is In (Arad), and the straight line distance heuristic h_{SLD} (In (Arad)) is found to be 366. Using the straight-line distance heuristic h_{SLD} , the goal state can be reached faster.

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 3.23 shows the progress of greedy best-first search using h_{SLD} to find a path from Arad to Bucharest. The first node to be expanded from Arad will be Sibiu, because it is closer to Bucharest than either Zerind or Timisoara. The next node to be expanded will be Fagaras,

because it is closest. Fagaras in turn generates Bucharest, which is the goal.



Evaluation Criterion of Greedy Search

- Complete: NO [can get stuck in loops, e.g., Complete in finite space with repeated state checking]
- Time Complexity: $O(bm)$ [but a good heuristic can give dramatic improvement]
- Space Complexity: $O(bm)$ [keeps all nodes in memory]
- Optimal: NO

Greedy best-first search is not optimal, and it is incomplete. The worst-case time and space complexity is $O(bm)$, where m is the maximum depth of the search space.

3.5.2 A* search: Minimizing the total estimated solution cost:

The most widely known form of best-first search is called A* search (pronounced “A-star * search”). It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal: $f(n) = g(n) + h(n)$. Since $g(n)$ gives the path cost from the start node to node n , and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have $f(n)$ = estimated cost of the cheapest solution through n .

Conditions for optimality: Admissibility and consistency:

The first condition we require for optimality is that $h(n)$ be an admissible heuristic. An admissible heuristic is one that never overestimates the cost to reach the goal. Because $g(n)$ is the actual cost to reach n along the current path, and $f(n) = g(n) + h(n)$, we have as an immediate consequence that $f(n)$ never overestimates the true cost of a solution along the current path through n .

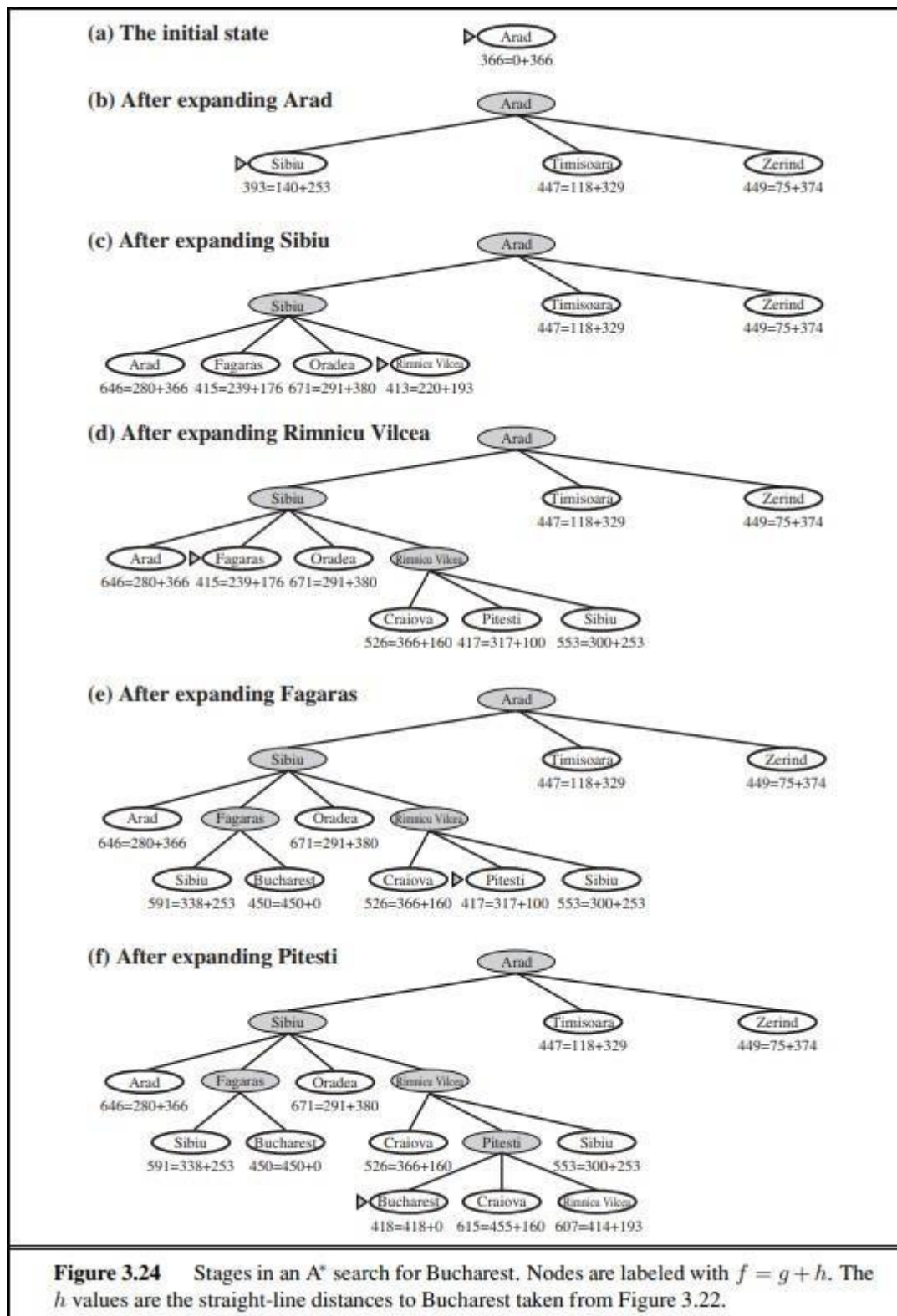
In Figure 3.24, we show the progress of an A* tree search for Bucharest. Notice in particular that Bucharest first appears on the frontier at step (e), but it is not selected for expansion because its f -cost (450) is higher than that of Pitesti (417). Another way to say this is that there might be a solution through Pitesti whose cost is as low as 417, so the algorithm will not settle for a solution that costs 450.

A second, slightly stronger condition called consistency (or sometimes monotonicity) is required only for applications of A* to graph search.⁹ A heuristic $h(n)$ is consistent if, for every node n and every successor n^1 of n generated by any action a , the estimated cost of reaching the goal from n is no greater than the step cost of getting to n^1 plus the estimated cost of reaching the goal from n^1 : $h(n) \leq c(n, a, n^1) + h(n^1)$. This is a form of the general triangle inequality, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides. Here, the triangle is formed by n , n^1 , and the goal G_n closest to n .

3.5.3 Memory-bounded heuristic search

The simplest way to reduce memory requirements for A* is to adapt the idea of iterative deepening to the heuristic search context, resulting in the iterative-deepening A* (IDA*) algorithm. The main difference between IDA* and standard iterative deepening is that the cutoff used is the f -cost ($g + h$) rather than the depth; at each iteration, the cutoff value is the smallest f -cost of any node that exceeded the cutoff on the previous iteration.

Recursive best-first search (RBFS) is a simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only linear space. The algorithm is shown in Figure 3.26.



Its structure is similar to that of a recursive depth-first search, but rather than continuing indefinitely down the current path, it uses the f limit variable to keep track of the f -value of the best alternative path available from any ancestor of the current node. If the current node exceeds this limit, the recursion unwinds back to the alternative path.

```

function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
  return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE),  $\infty$ )

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors  $\leftarrow$  []
  for each action in problem.ACTIONS(node.STATE) do
    add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure,  $\infty$ 
  for each s in successors do /* update f with value from previous search, if any */
    s.f  $\leftarrow$  max(s.g + s.h, node.f)
  loop do
    best  $\leftarrow$  the lowest f-value node in successors
    if best.f > f_limit then return failure, best.f
    alternative  $\leftarrow$  the second-lowest f-value among successors
    result, best.f  $\leftarrow$  RBFS(problem, best, min(f_limit, alternative))
    if result  $\neq$  failure then return result

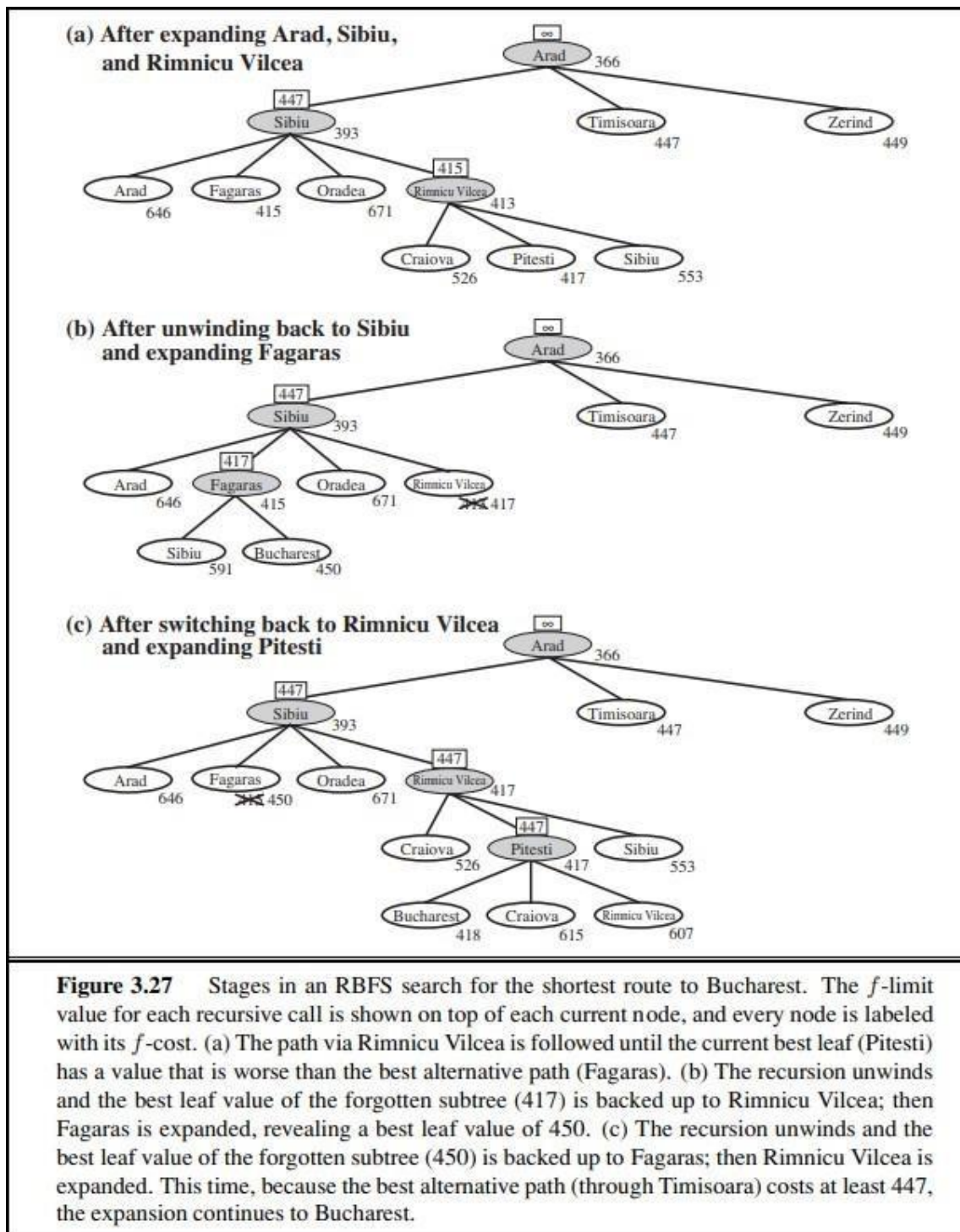
```

Figure 3.26 The algorithm for recursive best-first search.

As the recursion unwinds, RBFS replaces the *f*-value of each node along the path with a backed-up value—the best *f*-value of its children. In this way, RBFS remembers the *f*-value of the best leaf in the forgotten subtree and can therefore decide whether it's worth reexpanding the subtree at some later time. Figure 3.27 shows how RBFS reaches Bucharest. RBFS is somewhat more efficient than IDA*, but still suffers from excessive node regeneration. In the example in Figure 3.27, RBFS follows the path via Rimnicu Vilcea, the “changes its mind” and tries Fagaras, and then changes its mind back again.

Like A* tree search, RBFS is an optimal algorithm if the heuristic function $h(n)$ is admissible. Its space complexity is linear in the depth of the deepest optimal solution, but its time complexity is rather difficult to characterize: it depends both on the accuracy of the heuristic function and on how often the best path changes as nodes are expanded. IDA* and RBFS suffer from using too little memory. Between iterations, IDA* retains only a single number: the current *f*-cost limit. RBFS retains more information in memory, but it uses only linear space: even if more memory were available, RBFS has no way to make use of it.

On very hard problems, however, it will often be the case that SMA* is forced to switch back and forth continually among many candidate solution paths, only a small subset of which can fit in memory. That is to say, memory limitations can make a problem intractable from the point of view of computation time. Although no current theory explains the tradeoff between time and memory, it seems that this is an inescapable problem. The only way out is to drop the optimality requirement.



3.5.4 Learning to search better

Each state in a metalevel state space captures the internal (computational) state of a program that is searching in an object-level state space such as Romania. For example, the internal state of the A* algorithm consists of the current search tree. Each action in the metalevel state space is a computation step that alters the internal state; for example, each computation step in A* expands a leaf node and adds its successors to the tree.

3.6 Heuristic functions

- Heuristic functions in order to reach the goal node in a more prominent way.
- Therefore, there are several pathways in a search tree to reach the goal node from the current node.
- The selection of a good heuristic function matters certainly.
- A good heuristic function is determined by its efficiency.
- More is the information about the problem, more is the processing time.

Example:

- Consider the following 8-puzzle problem where we have a start state and a goal state.
- Our task is to slide the tiles of the current/start state and place it in an order followed in the goal state. There can be four moves either left, right, up, or down.
- There can be several ways to convert the current/start state to the goal state, but we can use a heuristic function $h(n)$ to solve the problem more efficiently.

1	2	3
8	6	
7	5	4

Start State

1	2	3
8		4
7	6	5

Goal State

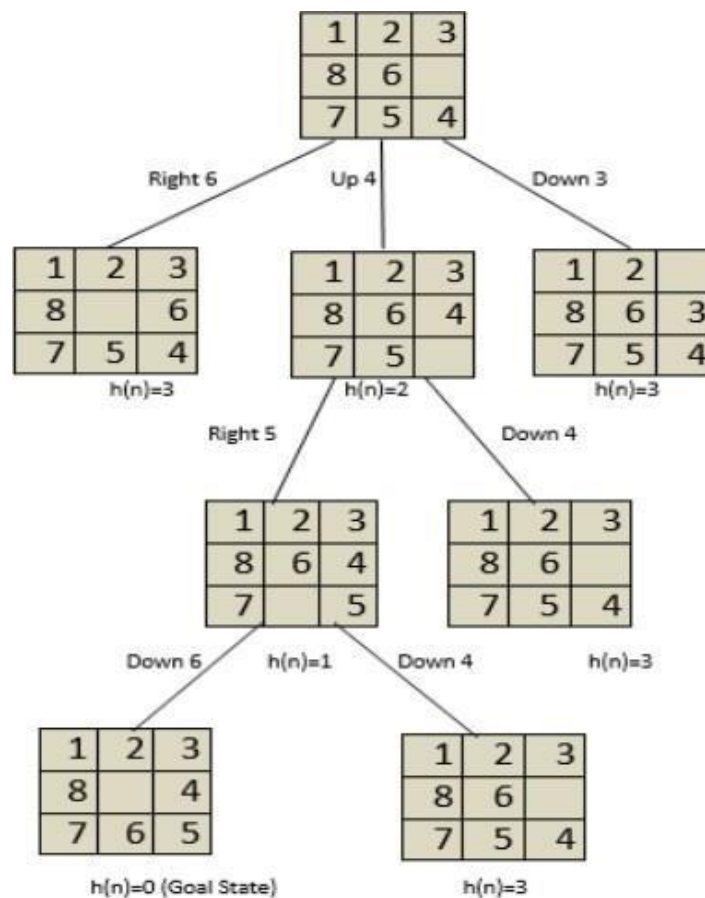
A heuristic function for the 8-puzzle problem is defined below:

$h(n)$ = Number of tiles out of position.

So, there is total of three tiles out of position i.e., 6, 5 and 4. Do not count the empty tile present in the goal state). i.e. $h(n)=3$. Now, we require to minimize the value of $h(n) = 0$.

We can construct a state-space tree to minimize the $h(n)$ value to 0, as shown below:

- It is seen from the above state space tree that the goal state is minimized from $h(n)=3$ to $h(n)=0$. However, we can create and use several heuristic functions as per the requirement.
- It is also clear from the above example that a heuristic function $h(n)$ can be defined as the information required to solve a given problem more efficiently.
- The information can be related to the nature of the state, cost of transforming from one state to another, goal node characteristics, etc., which is expressed as a heuristic function.



Properties of a Heuristic search Algorithm

Use of heuristic function in a heuristic search algorithm leads to following properties of a heuristic search algorithm:

- **Admissible Condition:** An algorithm is said to be admissible, if it returns an optimal solution.
- **Completeness:** An algorithm is said to be complete, if it terminates with a solution (if the solution exists).
- **Dominance Property:** If there are two admissible heuristic algorithms A1 and A2 having h_1 and h_2 heuristic functions, then A1 is said to dominate A2 if h_1 is better than h_2 for all the values of node n .
- **Optimality Property:** If an algorithm is complete, admissible, and dominating other algorithms, it will be the best one and will definitely give an optimal solution

Chapter 7:

Logical Agents

7.1 Knowledge-Based Agent in Artificial intelligence

An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.

Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge**, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently.

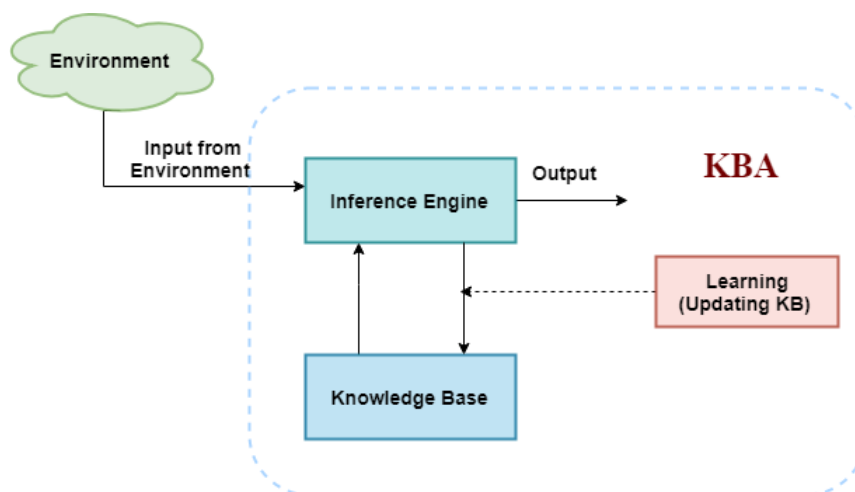
Knowledge-based agents are composed of two main parts:

- **Knowledge-base and**
- **Inference system.**

A knowledge-based agent must be able to do the following:

- An agent should be able to represent states, actions, etc.
- An agent should be able to incorporate new percepts.
- An agent can update the internal representation of the world
- An agent can deduce the internal representation of the world
- An agent can deduce appropriate actions.

The architecture of knowledge-based agent:



The above diagram is representing a generalized architecture for a knowledge-based agent. The knowledge-based agent (KBA) take input from the environment by perceiving the environment. The input is taken by the inference engine of the agent and which also communicate with KB to decide as per the knowledge store in KB. The learning element of KBA regularly updates the KB by learning new knowledge.

Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world.

Why use a knowledge base?

Knowledge-base is required for updating knowledge for an agent to learn with experiences and take action as per the knowledge.

Inference system:

Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.

Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:

- o Forward chaining
- o Backward chaining

Operations Performed by KBA

Following are three operations which are performed by KBA in order to show the intelligent behavior:

1. **TELL:** This operation tells the knowledge base what it perceives from the environment.
2. **ASK:** This operation asks the knowledge base what action it should perform.
3. **Perform:** It performs the selected action.

A generic knowledge-based agent:

Following is the structure outline of a generic knowledge-based agents program:

```
function KB-AGENT (percept) :  
    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
```

The knowledge-based agent takes percept as input and returns an action as output. The agent maintains the knowledge base, KB, and it initially has some background knowledge of the real world. It also has a counter to indicate the time for the whole process, and this counter is initialized with zero.

Each time when the function is called, it performs its three operations:

- Firstly it TELLS the KB what it perceives.
- Secondly, it asks KB what action it should take
- Third agent program TELLS the KB that which action was chosen.

The MAKE-PERCEPT-SENTENCE generates a sentence as setting that the agent perceived the given percept at the given time.

The MAKE-ACTION-QUERY generates a sentence to ask which action should be done at the current time.

MAKE-ACTION-SENTENCE generates a sentence which asserts that the chosen action was executed.

Various levels of knowledge-based agent:

A knowledge-based agent can be viewed at different levels which are given below:

1. Knowledge level

Knowledge level is the first level of knowledge-based agent, and in this level, we need to

specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior. For example, suppose an automated taxi agent needs to go from a station A to station B, and he knows the way from A to B, so this comes at the knowledge level.

2. Logical level:

At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs. At the logical level we can expect to the automated taxi agent to reach to the destination B.

3. Implementation level:

This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.

Approaches to designing a knowledge-based agent:

There are mainly two approaches to build a knowledge-based agent:

1. Declarative approach: We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.

2. Procedural approach: In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

What is knowledge representation?

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning.** Hence we can describe Knowledge representation as following:

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to

intelligent behavior of agents.

- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge- based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

Types of knowledge

- Following are the various types of knowledge:

**1. Declarative Knowledge:**

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta- knowledge.

4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a filed or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness

of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

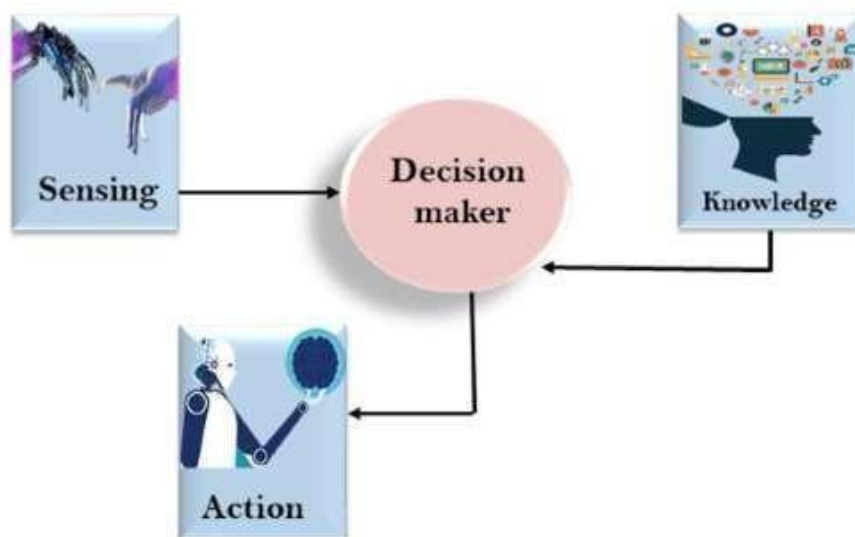
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will be able to act on that. The same thing applies to the intelligent behavior of the agents.

As we can see in below diagram, there is one decision maker which acts by sensing the environment and using knowledge. But if the knowledge part will not be present then, it cannot display intelligent behavior.



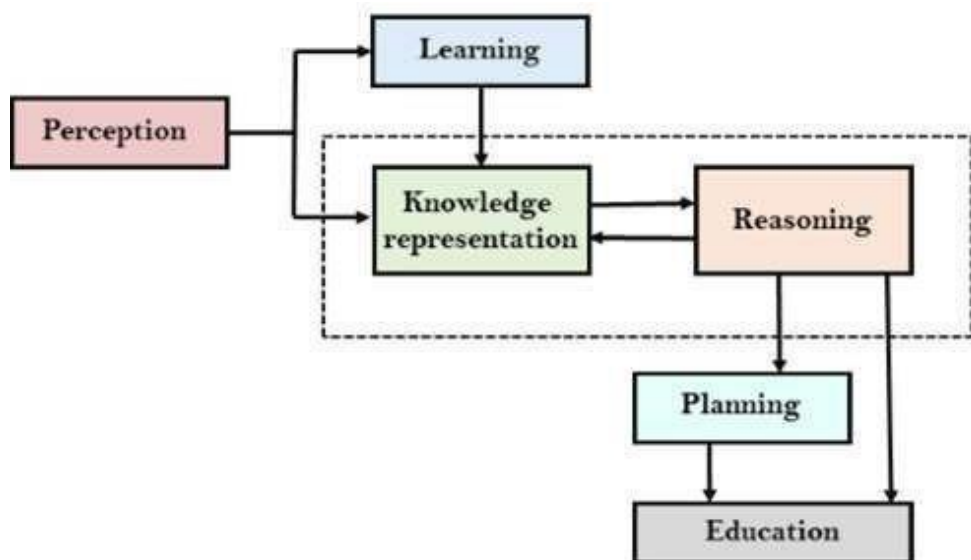
AI knowledge cycle:

An Artificial intelligence system has the following components for displaying

intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution

The below diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.



Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

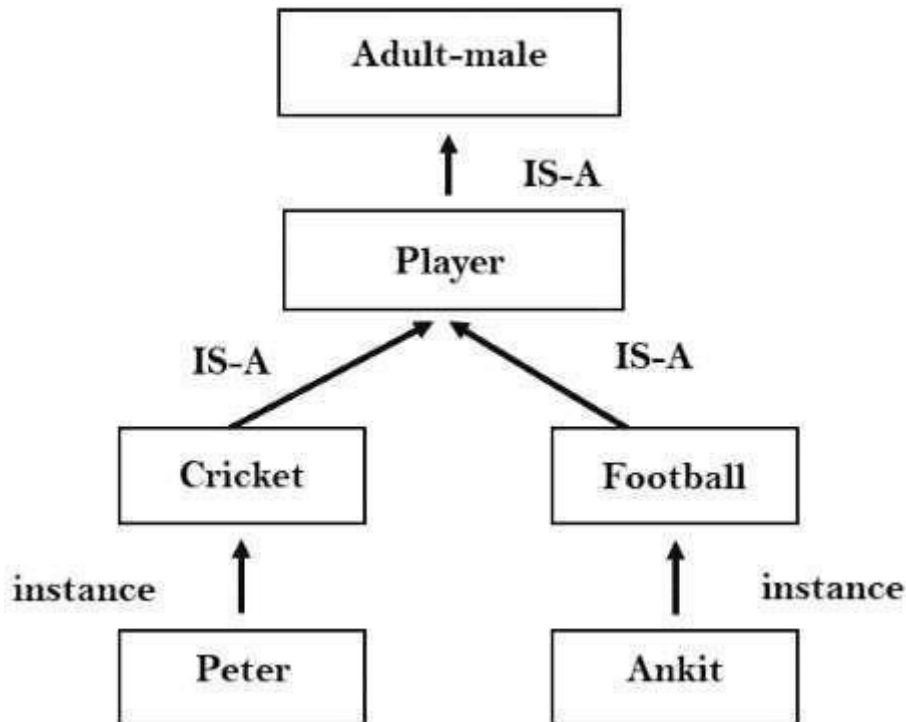
Example: The following is the simple relational knowledge representation.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.

Example:



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
 - a. Marcus is a man
 - b. All men are mortal
 Then it can represent as;

man(Marcus)

$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

1. Representational Accuracy:

KR system should have the ability to represent all kind of required knowledge.

2. Inferential Adequacy:

KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.

3. Inferential Efficiency:

The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.

4. Acquisitional efficiency- The ability to acquire the new knowledge easily using automatic methods.

The Wumpus World in Artificial intelligence

Wumpus world:

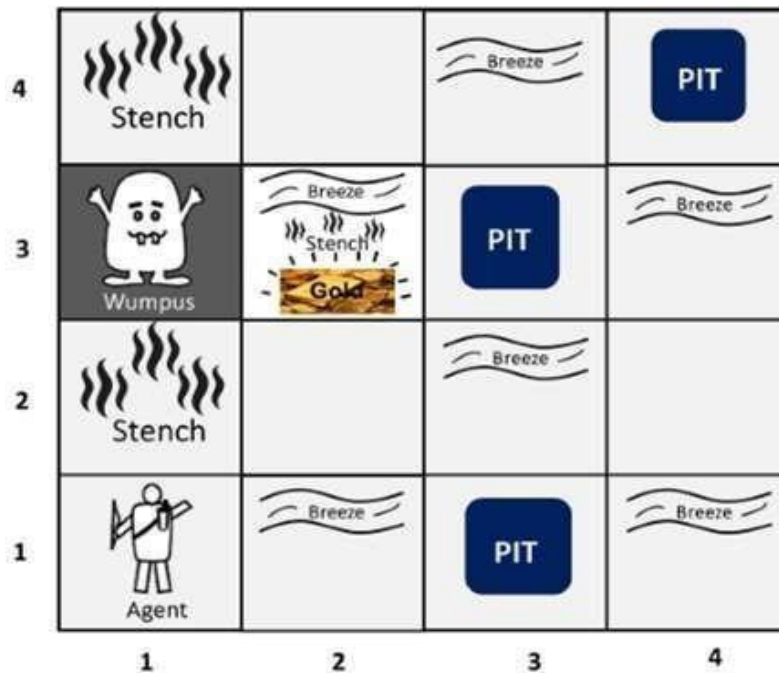
The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game **Hunt the Wumpus** by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

Note: Here Wumpus is static and cannot move.

Following is a sample diagram for representing the Wumpus world. It is showing some

rooms with Pits, one room with Wumpus and one agent at (1, 1) square location of the world.



There are also some components which can help the agent to navigate the cave. These components are given as follows:

- The rooms adjacent to the Wumpus room are smelly, so that it would have some stench.
- The room adjacent to PITs has a breeze, so if the agent reaches near to PIT, then he will perceive the breeze.
- There will be glitter in the room if and only if the room has gold.
- The Wumpus can be killed by the agent if the agent is facing to it, and Wumpus will emit a horrible scream which can be heard anywhere in the cave.

PEAS description of Wumpus world:

To explain the Wumpus world we have given PEAS description as below

Performance measure:

- +1000 reward points if the agent comes out of the cave with the gold.
- -1000 points penalty for being eaten by the Wumpus or falling into the pit.
- -1 for each action, and -10 for using an arrow.
- The game ends if either agent dies or came out of the cave.

Environment:

- A 4*4 grid of rooms.
- The agent initially in room square [1, 1], facing toward the right.
- Location of Wumpus and gold are chosen randomly except the first square

[1,1].

- Each square of the cave can be a pit with probability 0.2 except the first square.

Actuators:

- Left turn,
- Right turn
- Move forward
- Grab
- Release
- Shoot.

Sensors:

- The agent will perceive the **stench** if he is in the room adjacent to the Wumpus. (Not diagonally).
- The agent will perceive **breeze** if he is in the room directly adjacent to the Pit.
- The agent will perceive the **glitter** in the room where the gold is present.
- The agent will perceive the **bump** if he walks into a wall.
- When the Wumpus is shot, it emits a horrible **scream** which can be perceived anywhere in the cave
- These percepts can be represented as five element list, in which we will have different indicators for each sensor.
- Example if agent perceives stench, breeze, but no glitter, no bump, and no scream then it can be represented as:
[Stench, Breeze, None, None, None].

The Wumpus world Properties:

- **Partially observable:** The Wumpus world is partially observable because the agent can only perceive the close environment such as an adjacent room.
- **Deterministic:** It is deterministic, as the result and outcome of the world are already known.
- **Sequential:** The order is important, so it is sequential.
- **Static:** It is static as Wumpus and Pits are not moving.
- **Discrete:** The environment is discrete.
- **One agent:** The environment is a single agent as we have one agent only and Wumpus is not considered as an agent.

Exploring the Wumpus world:

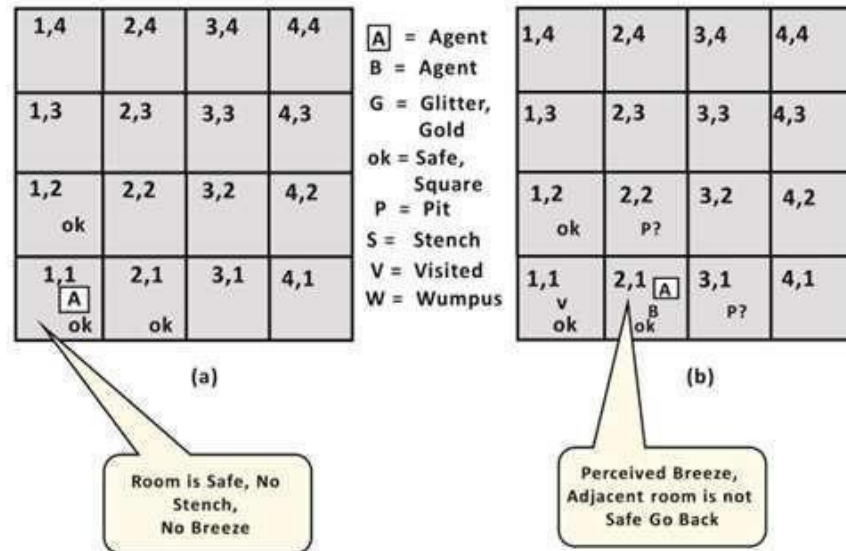
Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

Agent's First step:

Initially, the agent is in the first room or on the square [1,1], and we already know that this

room is safe for the agent, so to represent on the below diagram (a) that room is safe we will add symbol OK. Symbol A is used to represent agent, symbol B for the breeze, G for Glitter or gold, V for the visited room, P for pits, W for Wumpus.

At Room [1,1] agent does not feel any breeze or any Stench which means the adjacent squares are also OK.



Agent's second Step:

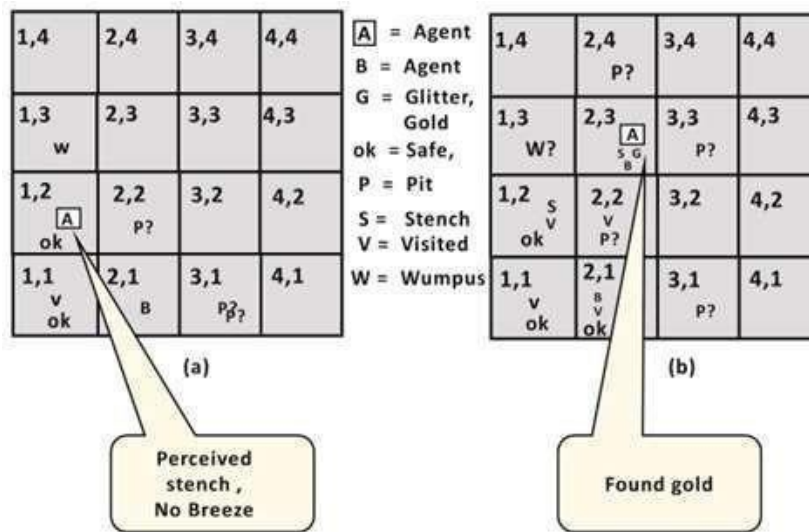
Now agent needs to move forward, so it will either move to [1, 2], or [2,1]. Let's suppose agent moves to the room [2, 1], at this room agent perceives some breeze which means Pit is around this room. The pit can be in [3, 1], or [2,2], so we will add symbol P? to say that, is this Pit room?

Now agent will stop and think and will not make any harmful move. The agent will go back to the [1, 1] room. The room [1,1], and [2,1] are visited by the agent, so we will use symbol V to represent the visited squares.

Agent's third step:

At the third step, now agent will move to the room [1,2] which is OK. In the room [1,2] agent perceives a stench which means there must be a Wumpus nearby. But Wumpus cannot be in the room [1,1] as by rules of the game, and also not in [2,2] (Agent had not detected any stench when he was at [2,1]). Therefore agent infers that Wumpus is in the room [1,3], and

in current state, there is no breeze which means in [2,2] there is no Pit and no Wumpus. So it is safe, and we will mark it OK, and the agent moves further in [2,2].



Agent's fourth step:

At room [2,2], here no stench and no breezes present so let's suppose agent decides to move to [2,3]. At room [2,3] agent perceives glitter, so it should grab the gold and climb out of the cave.

Propositional logic in Artificial intelligence

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Example:

- It is Sunday.
- The Sun rises from West (False proposition)
- $3+3=7$ (False proposition)
- 5 is a prime number.

Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can

use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.

- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

a. **Atomic Propositions**

b. **Compound propositions**

. **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

a) $2+2$ is 4, it is an atomic proposition as it is a **true** fact.

b) "The Sun is cold" is also a proposition as it is a **false** fact.

a. **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Example:

a) "It is raining today, and street is wet."

b) "Ankit is a doctor, and his clinic is in Mumbai."

Logical Connectives:

Logical connectives are used to connect two simpler propositions or representing a

sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as $\neg P$ is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has $\wedge$ connective such as, $P \wedge Q$ is called a conjunction.

Example: Rohan is intelligent and hardworking. It can be written as,

P= Rohan is intelligent,

Q= Rohan is hardworking. $\rightarrow PA Q$.

3. **Disjunction:** A sentence which has $\vee$ connective, such as $P \vee Q$. is called disjunction, where P and Q are the propositions.

Example: "Ritika is a doctor or Engineer",

Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as $P \vee Q$.

4. **Implication:** A sentence such as $P \rightarrow Q$, is called an implication.

Implications are also known as if-then rules. It can be represented as

If it is raining, then the street is wet.

Let P= It is raining, and Q= Street is wet, so it is represented as $P \rightarrow Q$

5. **Biconditional:** A sentence such as $P \Leftrightarrow Q$ is a Biconditional sentence, example If I am breathing, then I am alive

P= I am breathing, Q= I am alive, it can be represented as $P \Leftrightarrow Q$.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\Leftrightarrow$	If and only if	Biconditional	$A \Leftrightarrow B$
$\neg$ or $\sim$	Not	Negation	$\neg A$ or $\neg B$

Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called Truth table. Following are the truth table for all logical connectives:

For Biconditional:

P	Q	$P \leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

For Negation:

P	$\neg P$
True	False
False	True

For Conjunction:

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

For disjunction:

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

For Implication:

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem.

Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication
Six Precedence	Biconditional

Note: For better understanding use parenthesis to make sure of the correct interpretations. Such as $\neg R \vee Q$, It can be interpreted as $(\neg R) \vee Q$.

Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as $A \Leftrightarrow B$. In

below truth table we can see that column for $\neg A \vee B$ and $A \rightarrow B$, are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Properties of Operators:

Commutativity:

$P \wedge Q = Q \wedge P$, or $P \vee Q = Q \vee P$.

Associativity:

$(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$, $(P \vee Q) \vee R = P \vee (Q \vee R)$

Identity element:

$P \wedge \text{True} = P$,

$P \vee \text{True} = \text{True}$.

Distributive:

$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$. $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$. DE Morgan's Law:

$\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$

$\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$.

Double-negation elimination:

$\neg (\neg P) = P$.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.

Example:

- All the girls are intelligent.
 - Some apples are sweet.
- b. Propositional logic has limited expressive power.
- c. In propositional logic, we cannot describe statements in terms of their properties or logical relationships.