AI UNIT - 1 CONCISE

UNIT 1:**** Introduction to Artificial Intelligence & Problem Solving

1. What is AI Technique? `[PYQ]`

- Real-world knowledge properties:
  - Huge volume, next to unimaginable.
  - Not well-organized or well-formatted.
  - Keeps changing constantly.
- AI Technique: A method to organize & use knowledge efficiently such that:
  - It should be perceivable by the people who provide it.
  - It should be easily modifiable to correct errors.
  - It should be useful in many situations though it is incomplete or inaccurate.
- Elevates speed of execution for complex programs it is equipped with.

2. Applications of AI `[PYQ]`

- **Gaming:** `[PYQ]`
  - Crucial role in strategic games (chess, poker, tic-tac-toe, etc.).
  - Machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing (NLP):** `[PYQ]`
  - Possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems:** `[PYQ]`
  - Integrate machine, software, and special information to impart reasoning and advising.
  - Provide explanation and advice to the users.
- **Vision Systems:**
  - Understand, interpret, and comprehend visual input on the computer.
  - Examples:
    - Spying airplane takes photographs for spatial information or maps.
    - Doctors use clinical expert system to diagnose patients.
    - Police use software to recognize criminal faces from forensic portraits.
- **Speech Recognition:**

    - Systems capable of hearing and comprehending language (sentences, meanings).
    - Can handle different accents, slang, background noise, etc.
- **Handwriting Recognition:**
    - Reads text written on paper/screen by pen/stylus.
    - Recognizes letter shapes and converts to editable text.
- **Intelligent Robots:**
    - Able to perform tasks given by a human.
    - Have sensors for physical data (light, heat, temperature, movement, sound, bump, pressure).
    - Efficient processors, multiple sensors, huge memory.
    - Capable of learning from mistakes and adapting to new environments.
- **Astronomy:**
    - Solve complex universe problems (how it works, origin, etc.).
- **Healthcare:** `[PYQ]`
    - Better and faster diagnosis than humans.
    - Can inform when patients are worsening for timely medical help.
- **Finance:** `[PYQ]`
    - Automation, chatbot, adaptive intelligence, algorithm trading, machine learning in financial processes.
- **Data Security:**
    - Make data more safe and secure.
    - Examples (AEG bot, AI2 Platform) determine software bugs and cyber-attacks.
- **Social Media:**
    - Manage billions of user profiles efficiently.
    - Analyze data for latest trends, hashtags, user requirements.
- **Travel & Transport:**
    - Travel arrangements, suggesting hotels, flights, best routes.
    - AI-powered chatbots for human-like customer interaction.
- **Automotive Industry:** `[PYQ]`
    - Virtual assistants for better performance (e.g., TeslaBot).
    - Developing self-driven cars for safer journeys.
- **Robotics:** `[PYQ]`
    - Create intelligent robots that perform tasks with own experiences, not just pre-programmed.
    - E.g., Humanoid robots (Erica, Sophia) that talk and behave like humans.

- **Entertainment:**
  - AI-based applications (Netflix, Amazon) using ML/AI for recommendations.
- **Agriculture:**
  - Agriculture robotics, solid and crop monitoring, predictive analysis.
- **E-commerce:**
  - Competitive edge; helps shoppers discover products with recommended size, color, brand.
- **Education:** `[PYQ]`
  - Automate grading, allowing tutors more teaching time.
  - AI chatbot as teaching assistant.
  - Personal virtual tutor accessible anytime, anywhere.

3. What is Artificial Intelligence (AI)? `[PYQ]`

- A branch of computer science to create intelligent machines which can behave like humans, think like humans, and make decisions. `[PYQ]`
- Composed of two words: "Artificial" (man-made) and "Intelligence" (thinking power).
- Hence AI means "a man-made thinking power."
- AI exists when a machine can have human-based skills such as learning, reasoning, and solving problems. `[PYQ]`
- The goal is for machines to work with their own intelligence, not just programmed algorithms.

4. Why Artificial Intelligence?

- Create software/devices to solve real-world problems easily and accurately (health issues, marketing, traffic issues, etc.).
- Create personal virtual assistants (e.g., Cortana, Google Assistant, Siri, Alexa).
- Build robots to work in environments where human survival is at risk.
- Opens a path for other new technologies, new devices, and new Opportunities.

5. Goals of Artificial Intelligence

- Replicate human intelligence. `[PYQ]`
- Solve Knowledge-intensive tasks.
- An intelligent connection of perception and action.
- Building a machine which can perform tasks that require human intelligence such as: `[PYQ]`
  - Proving a theorem.
  - Playing chess.
  - Plan some surgical operation.

    - Driving a car in traffic.
- Creating systems that can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and advise its user.

6. What Comprises Artificial Intelligence? / What is Intelligence Composed of? `[PYQ]`

- Intelligence is an intangible part of our brain.
- It is a combination of:
    - **Reasoning:** The set of processes that enables us to provide basis for judgment, making decisions, and prediction.
    - **Learning:** The activity of gaining knowledge or skill by studying, practicing, being taught, or experiencing something.
    - **Problem Solving:** `[PYQ]` The process in which one perceives and tries to arrive at a desired solution. Includes **decision making** (selecting the best alternative).
    - **Perception:** The process of acquiring, interpreting, selecting, and organizing sensory information. (Presumes sensing).
    - **Linguistic Intelligence:** One's ability to use, comprehend, speak, and write verbal and written language.
- Disciplines required to achieve AI factors for a machine/software: Mathematics, Biology, Psychology, Sociology, Computer Science, Neurons Study, Statistics.

7. Advantages of Artificial Intelligence

- **High Accuracy with less errors:** AI machines are prone to fewer errors as decisions are based on pre-experience or information.
- **High-Speed:** AI systems can be very high-speed and enable fast decision-making (e.g., beating chess champions).
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful for risky areas:** Helpful in situations like defusing bombs, exploring the ocean floor.
- **Digital Assistant:** Useful for providing digital assistance (e.g., E-commerce product suggestions).
- **Useful as a public utility:** Self-driving cars for safer journeys, facial recognition for security, NLP for communication.

8. Disadvantages of Artificial Intelligence

- **High Cost:** Hardware and software requirements are very costly; requires maintenance.
- **Can't think out of the box:** Robots only do work for which they are trained/programmed.
- **No feelings and emotions:** AI machines cannot make emotional attachments; may be harmful if proper care is not taken.

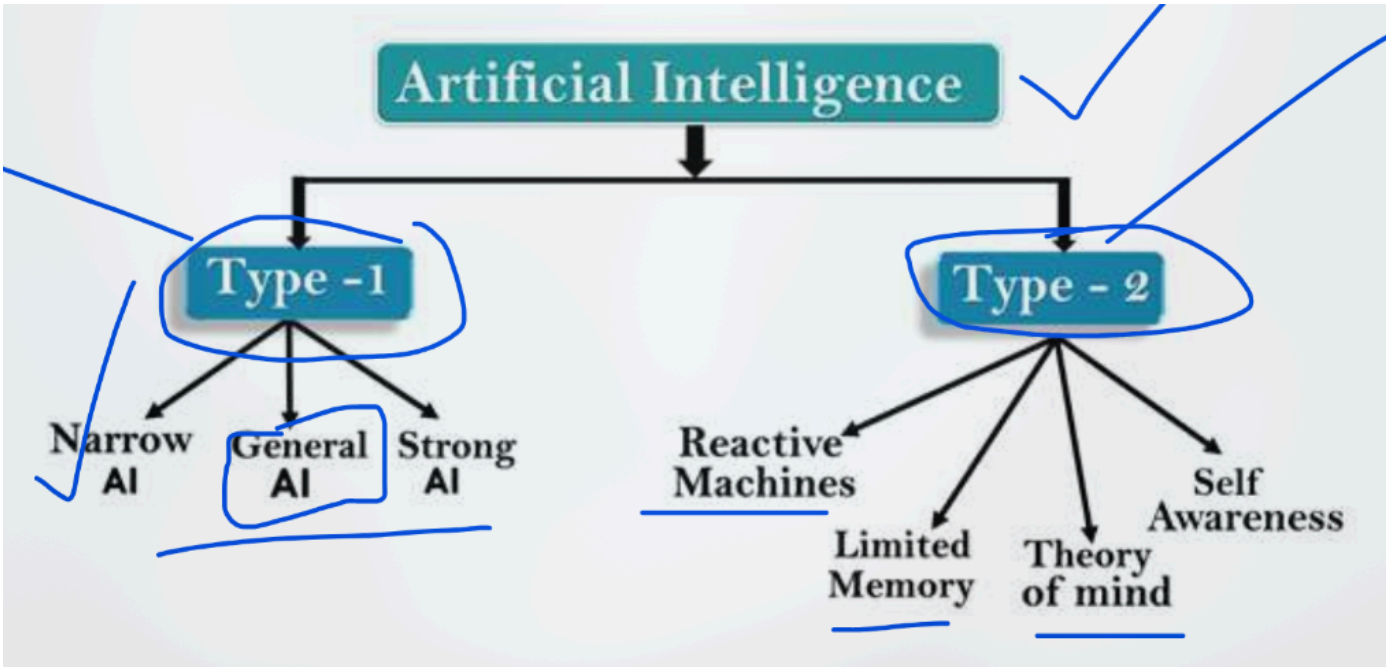
- **Increase dependency on machines:** People become more dependent, losing mental capabilities.
- **No Original Creativity:** AI machines cannot beat human intelligence in creativity and imagination.

9. History of Artificial Intelligence `[PYQ]`

- **Maturation of Artificial Intelligence (1943-1952):**
  - **Year 1943:** `[PYQ]` Warren McCulloch & Walter Pitts proposed a model of **artificial neurons**.
  - **Year 1949:** Donald Hebb demonstrated **Hebbian learning** (modifying connection strength between neurons).
  - **Year 1950:** `[PYQ]` Alan Turing published "Computing Machinery and Intelligence," proposing the **Turing test**. `[FIG]`
- **The birth of Artificial Intelligence (1952-1956):** `[PYQ]`
  - **Year 1955:** `[PYQ]` Allen Newell & Herbert A. Simon created the **"Logic Theorist,"** the first AI program.
  - **Year 1956:** `[PYQ]` John McCarthy coined "Artificial Intelligence" at the Dartmouth Conference. High-level languages (FORTRAN, LISP, COBOL) invented.
- **The golden years-Early enthusiasm (1956-1974):**
  - **Year 1966:** `[PYQ]` Joseph Weizenbaum created **ELIZA**, the first chatbot.
  - **Year 1972: WABOT-1**, the first intelligent humanoid robot, built in Japan.
- **The first AI winter (1974-1980):** Severe shortage of funding, decreased publicity.
- **A boom of AI (1980-1987):**
  - **Year 1980:** `[PYQ]` AI came back with **"Expert System."** First national AAAI conference at Stanford University.
  - **Year 1986:** NETtalk (neural network for reading and pronouncing words).
- **The second AI winter (1987-1993):** Investors stopped funding due to high cost and inefficient results (though XCON was cost-effective).
- **The emergence of intelligent agents (1993-2011):**
  - **Year 1997:** `[PYQ]` IBM Deep Blue beat world chess champion Gary Kasparov.
  - **Year 2002:** AI entered homes with Roomba (vacuum cleaner).
  - **Year 2006:** AI in Business world (Facebook, Twitter, Netflix).
- **Deep learning, big data and artificial general intelligence (2011-present):** `[PYQ]`
  - **Year 2011:** `[PYQ]` IBM's Watson won Jeopardy. Apple's Siri launched.
  - **Year 2012:** Google launched "Google now."
  - **Year 2014:** Chatbot "Eugene Goostman" won a Turing test competition variant. Microsoft Cortana.
  - **Year 2015:** Amazon Echo with Alexa.

- **Year 2018:** IBM's "Project Debater." Google's "Duplex" virtual assistant.

10. Types of Artificial Intelligence `[PYQ]` `[FIG]`



- **Type-1: Based on Capabilities**
  - **1. Weak AI or Narrow AI:** `[PYQ]`
    - Able to perform a *dedicated task* with intelligence. Most common type.
    - Cannot perform beyond its field or limitations; trained for one specific task.
    - Examples: Apple Siri, IBM's Watson, playing chess, e-commerce suggestions, self-driving cars, speech recognition, image recognition.
  - **2. General AI:** `[PYQ]`
    - Could perform any intellectual task with efficiency like a human.
    - System would be smarter and think like a human by its own.
    - Currently, no such system exists; still under research.
  - **3. Super AI:** `[PYQ]`
    - Level of Intelligence where machines could surpass human intelligence.
    - Can perform any task better than humans with cognitive properties. An outcome of general AI.
    - Characteristics: ability to think, reason, solve puzzles, make judgments, plan, learn, communicate by its own.
    - Hypothetical concept. `[FIG]` (showing progression: Narrow -> General -> Super)
- **Type-2: Based on Functionality** `[PYQ]`
  - **1. Reactive Machines:**
    - Most basic types. Do not store memories or past experiences for future actions.

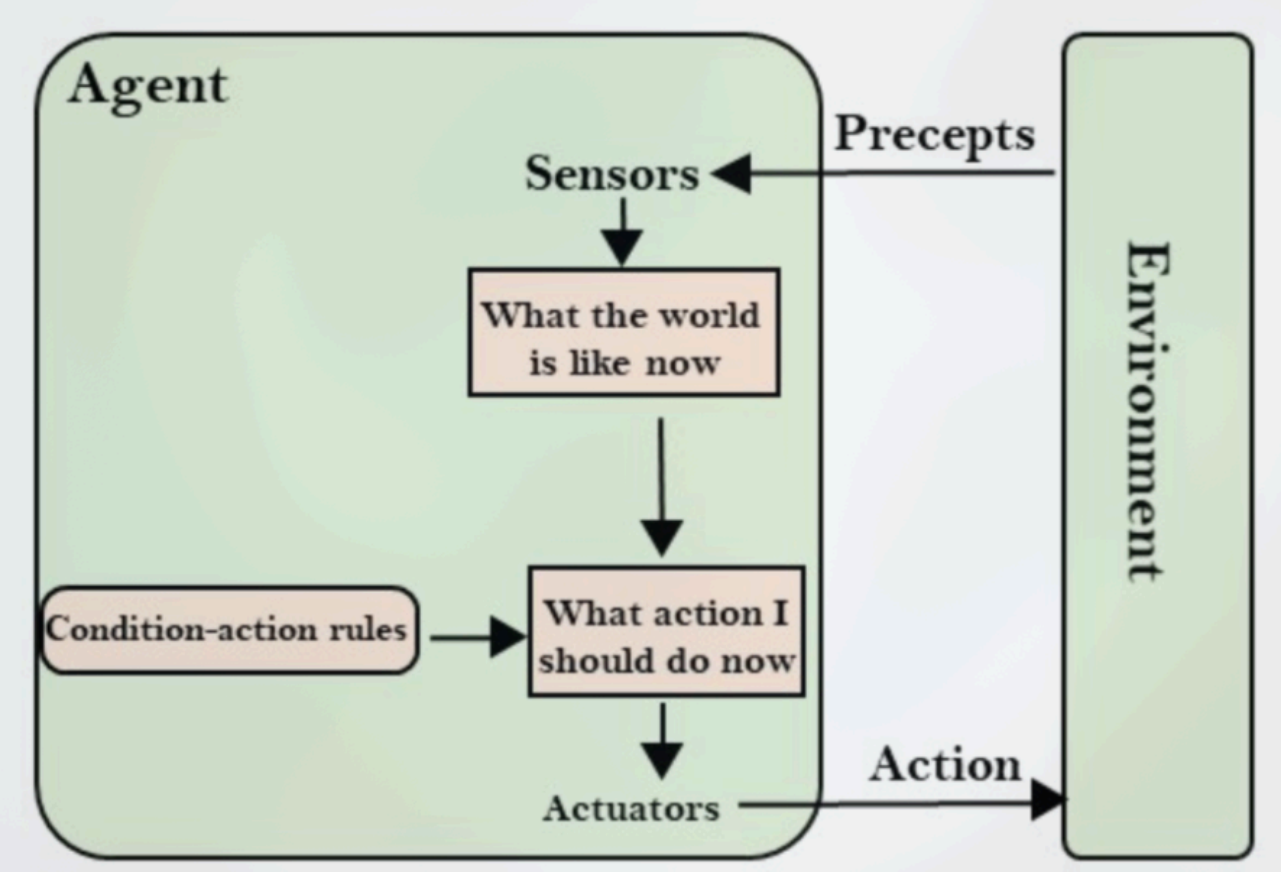
        - Only focus on current scenarios and react based on possible best action.
        - Examples: IBM's Deep Blue, Google's AlphaGo.
    - **2. Limited Memory:** `[PYQ]`
        - Can store past experiences or some data for a *short period of time*.
        - Can use stored data for a limited time period only.
        - Examples: Self-driving cars (store recent speed of nearby cars, distance of other cars, speed limit, etc.).
    - **3. Theory of Mind:**
        - Should understand human emotions, people, beliefs, and be able to interact socially like humans.
        - This type of AI machine is still not developed; researchers are making efforts.
    - **4. Self-Awareness:**
        - Future of Artificial Intelligence. Machines will be super intelligent, with their own consciousness, sentiments, and self-awareness.
        - These machines will be smarter than human mind.
        - Does not exist in reality yet; a hypothetical concept.

11. AI Agents `[PYQ]`

- **Definition:** Anything that perceives its environment through *sensors* and acts upon that environment through *actuators*. `[PYQ]` `[FIG]`
- Agent runs in the cycle of perceiving, thinking, and acting.
- **Types of Agents (Examples):**
    - **Human-Agent:** Eyes, ears (sensors); hands, legs, vocal tract (actuators).
    - **Robotic Agent:** Cameras, infrared range finder, NLP (sensors); various motors (actuators).
    - **Software Agent:** Keystrokes, file contents (sensory input); display output on screen (actions).
- **Agent Terminology (PEAS Components):** `[PYQ]` (as in "What is PEAS?")
    - **P - Performance Measure:** Criteria for success of an agent's behavior.
    - **E - Environment:** The world in which the agent operates.
    - **A - Actuators:** Devices agent uses to make changes in the environment (e.g., motors, display screen).
    - **S - Sensors:** Devices agent uses to perceive its environment (e.g., camera, keyboard).
- **PEAS Representation:** `[PYQ]` `[FIG]`
    - A way to describe an AI agent.
    - Example for self-driving cars:
        - **Performance:** Safety, time, legal drive, comfort.

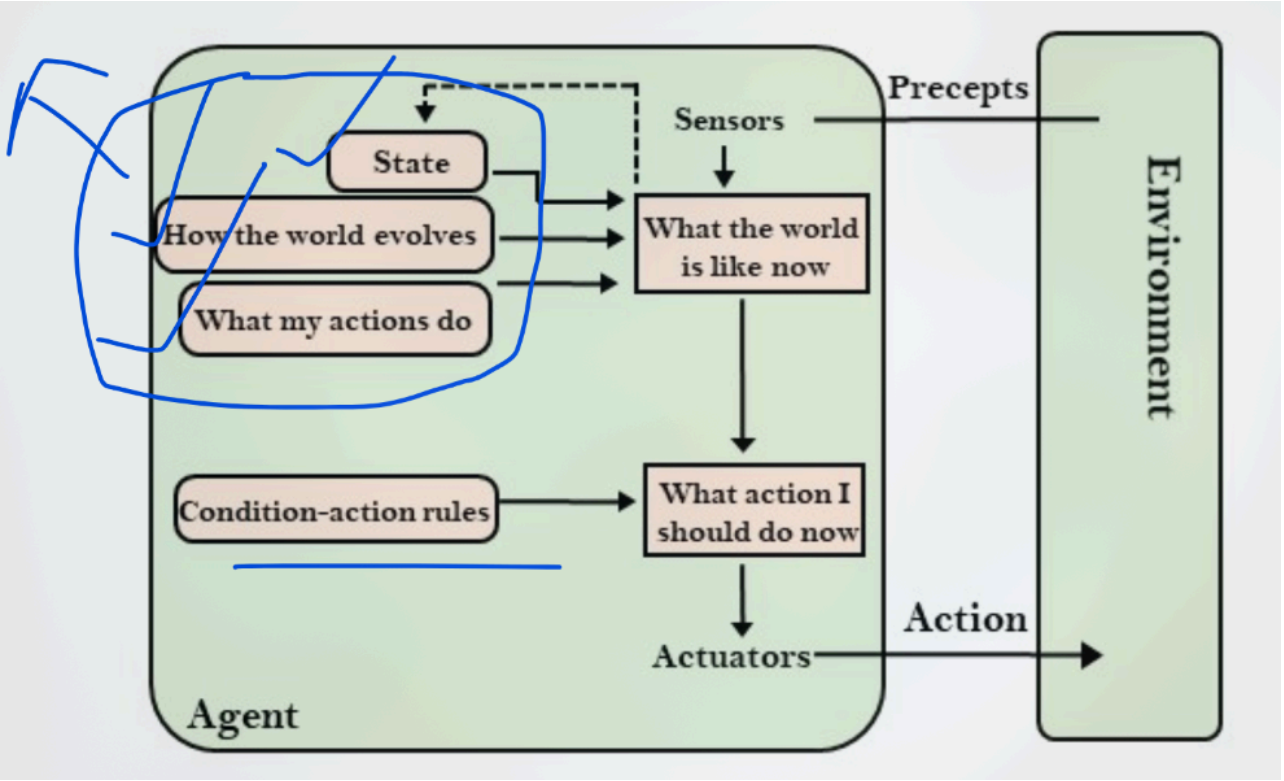
        - **Environment:** Roads, other vehicles, road signs, pedestrians.
        - **Actuators:** Steering, accelerator, brake, signal, horn.
        - **Sensors:** Camera, GPS, speedometer, odometer, accelerometer, sonar.
- **Rational Agent:** `[PYQ]`
    - An agent which has clear preference, models uncertainty, and acts to maximize its performance measure with all possible actions.
    - Performs the "right thing."
    - Rationality depends on: Performance measure, Agent's prior knowledge, Actions agent can perform, Percept sequence.
- **Intelligent Agent vs. Rational Agent:**
    - **Intelligent Agent:** A system that can perceive its environment and take actions to achieve a specific goal.
    - **Rational Agent:** An Intelligent Agent that makes decisions based on logical reasoning and optimizes its behavior to achieve a specific goal.
- **Structure of an AI Agent:** `[FIG]`
    - Agent = Architecture + Agent program.
    - **Architecture:** Machinery that an AI agent executes on.
    - **Agent Function:** Maps a percept to an action (f: P* → A).
    - **Agent program:** Implementation of agent function, executes on physical architecture.
- **Types of AI Agents (Program Structure):** `[PYQ]` `[FIG]` for each type

- **Simple Reflex Agent:** `[FIG]`



  - Acts based only on the current percept, ignoring percept history.
  - Works on Condition-action rules.
  - Succeeds only in fully observable environments.

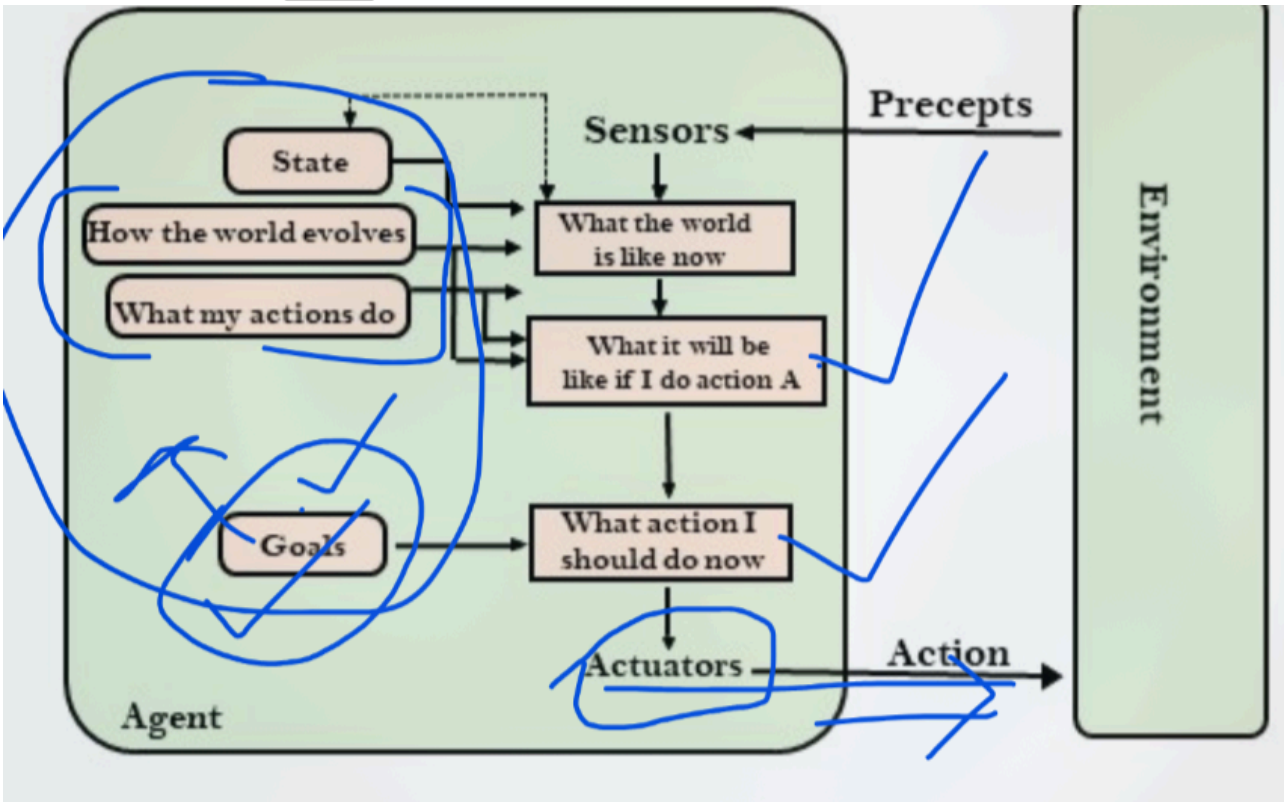
- **Model-Based Reflex Agent:** `[FIG]`



  - Maintains an internal state to track the part of the world it cannot currently see.
  - Uses a model of how the world evolves and how its actions affect the world.

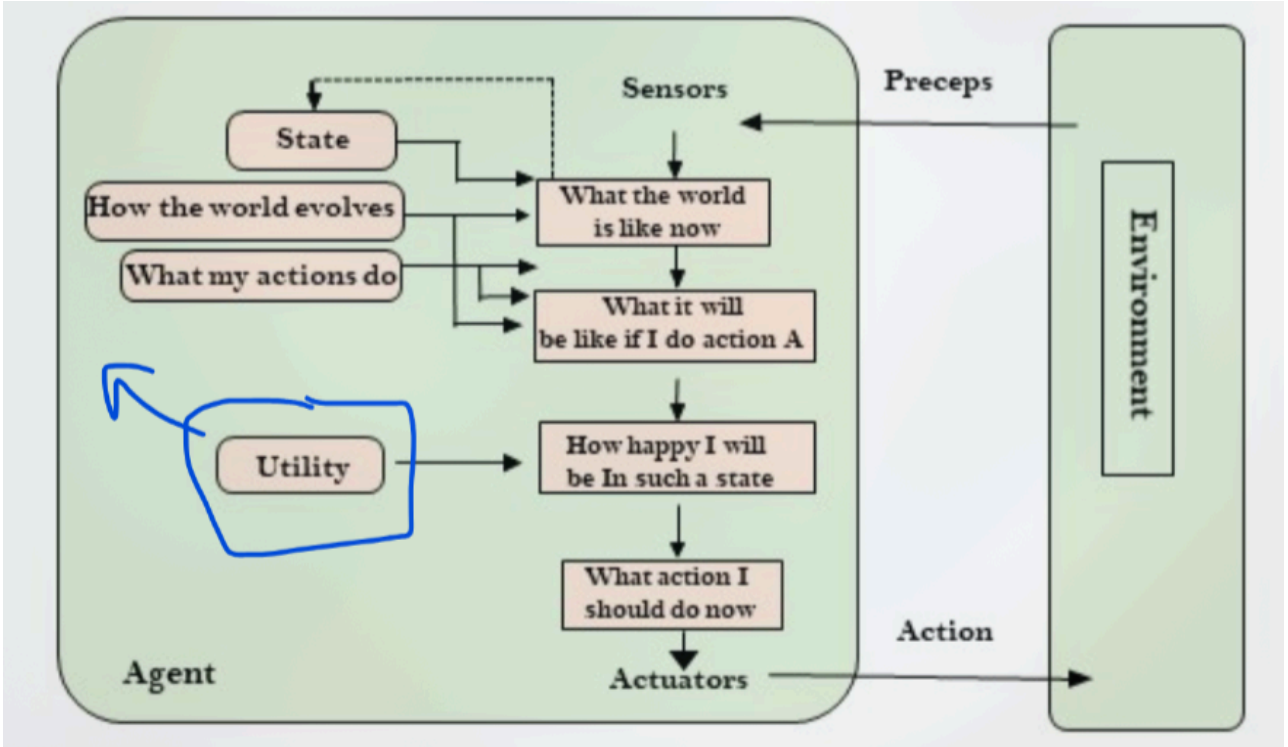
- Can handle partially observable environments.

- **Goal-Based Agent:** `[FIG]`



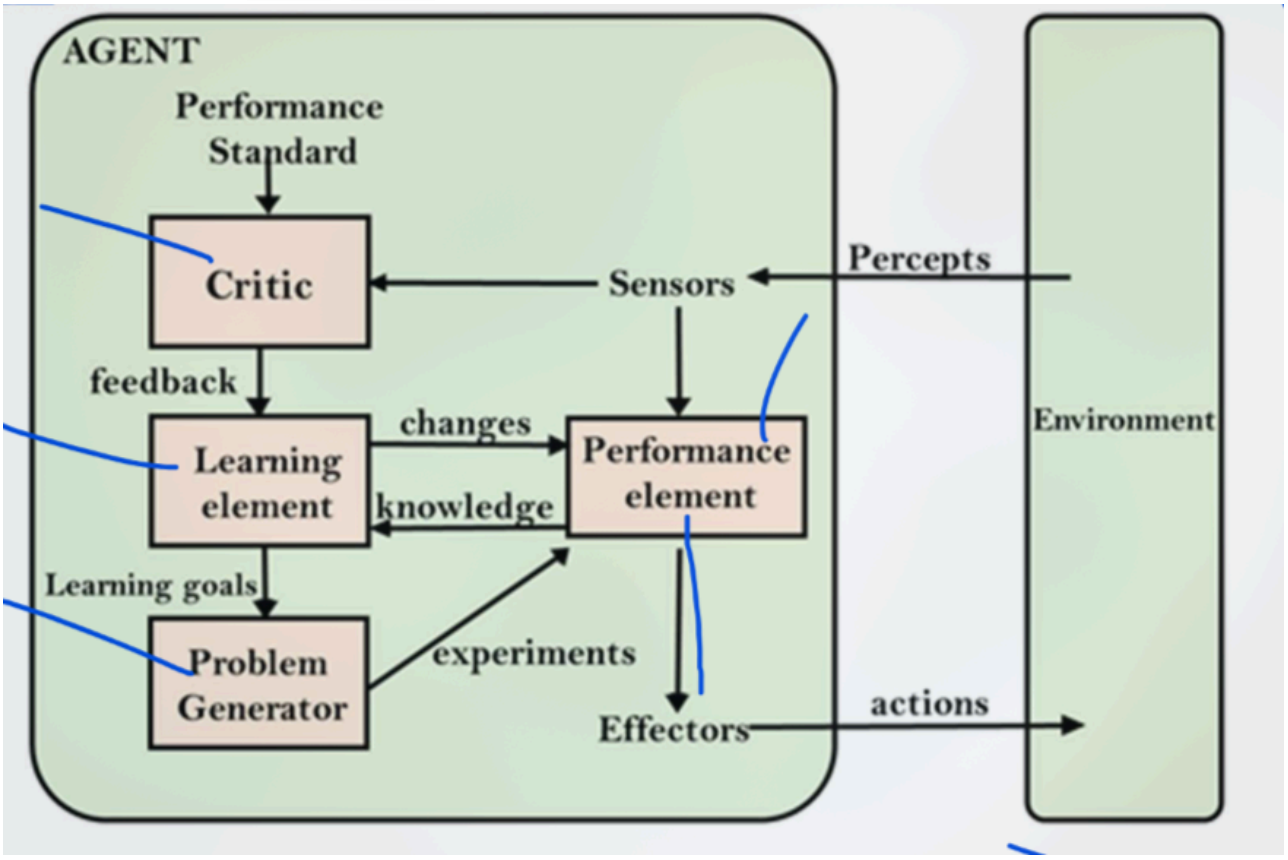
- Acts to achieve explicit goals. Knowledge of goals describes desirable situations.
- May require search and planning. More flexible.

- **Utility-Based Agent:** `[FIG]`



- Acts to maximize expected utility (a measure of "happiness" or success at a given state).
- Useful with multiple conflicting goals or uncertainty.

- **Learning Agent:** `[PYQ]` `[FIG]`



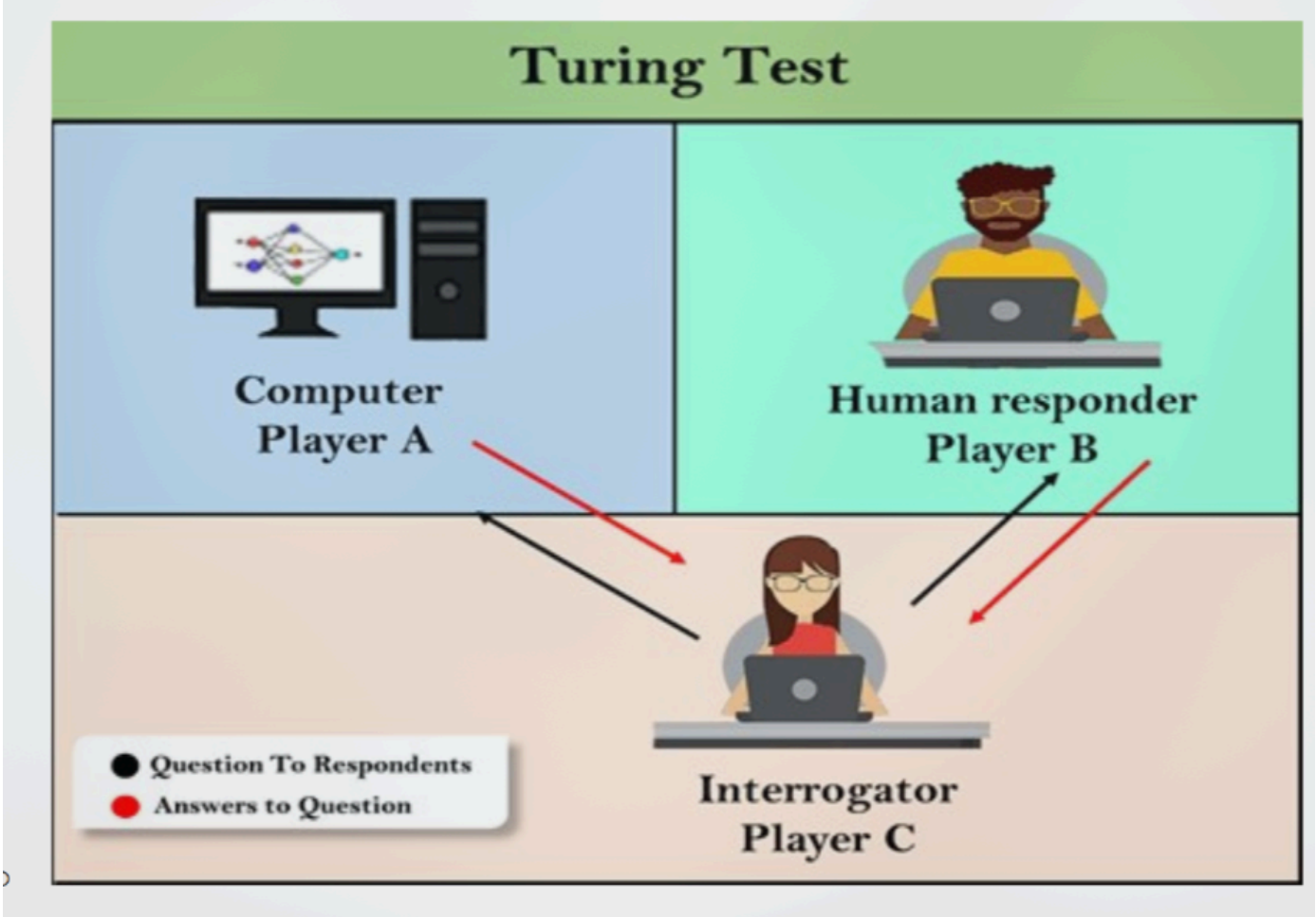
  - Can learn from past experiences to improve performance.
  - Starts with basic knowledge, adapts automatically through learning.
  - Components: Learning element, Critic, Performance element, Problem generator.

12. Agent Environment `[PYQ]`

- Everything in the world which surrounds the agent, but not part of the agent itself.
- Where the agent lives, operates, and gets something to sense and act upon.
- **Properties/Features of Environments:** `[PYQ]`
  - **Fully observable vs. Partially Observable:** Agent's sensors give access to complete state vs. not.
  - **Deterministic vs. Stochastic:** `[PYQ]` Next state completely determined by current state and action vs. randomness involved.
  - **Episodic vs. Sequential:** `[PYQ]` Agent's experience is a series of one-shot actions (current percept for action) vs. agent requires memory of past actions.
  - **Static vs. Dynamic:** `[PYQ]` Environment cannot change while agent is deliberating vs. it can change.
  - **Discrete vs. Continuous:** Finite number of percepts/actions vs. continuous values.
  - **Single-agent vs. Multi-agent:** `[PYQ]` Only one agent involved vs. multiple agents operating (can be cooperative or competitive).
  - **Known vs. Unknown:** Outcomes for all actions are known to the agent vs. agent needs to learn how the environment works.

- **Accessible vs. Inaccessible:** Agent can obtain complete and accurate information about the state vs. not (can be subtle, often related to observability).

13. Turing Test `[PYQ]` `[FIG]`



- Proposed by Alan Turing (1950) to check if a machine can think like a human.
- Involves three players: Computer, Human responder, Human Interrogator.
- Interrogator, isolated, tries to identify the machine based on text conversation.
- If the machine's responses are indistinguishable from a human's, it passes.
- **Features required for a machine to pass the Turing test:** `[PYQ]`
  - **Natural language processing (NLP):** To communicate.
  - **Knowledge representation:** To store information.
  - **Automated reasoning:** To use stored information for answers.
  - **Machine learning:** To adapt and detect generalized patterns.
  - **Vision (For total Turing test):** To recognize interrogator actions/objects.
  - **Motor Control (For total Turing test):** To act upon objects if requested.
- Chatbots that attempted the test: ELIZA, Parry, Eugene Goostman.

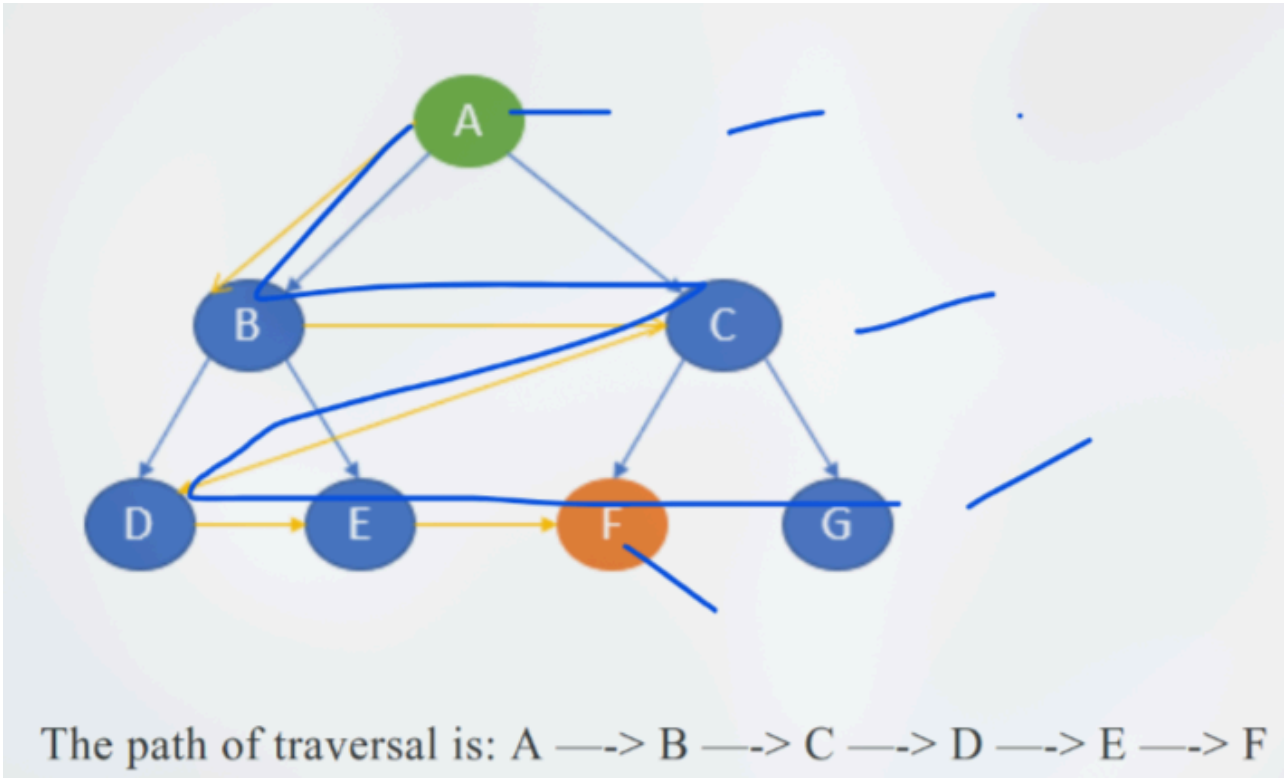
14. Problem Solving Agents & State Space Search `[PYQ]`

- **Problem-solving agents:** Goal-based agents that use algorithms to search for a sequence of actions leading to a goal.
- **Search Problem Components:** `[PYQ]`
  - **Initial State:** `[PYQ]` The state from where the search begins.
  - **Actions:** `[PYQ]` Set of possible operations applicable to a state.
  - **Transition Model:** `[PYQ]` Describes the result of performing an action in a state.
  - **Goal Test:** `[PYQ]` Determines if a given state is a goal state.
  - **Path Costing:** `[PYQ]` Assigns a numerical cost to a path.
- **State Space:** `[PYQ]` `[FIG]` Set of all possible states reachable from the initial state by any sequence of actions.
- **State Space Search:** `[FIG]` (general search process diagram) Process of finding a path from an initial state to a goal state.
- **Types of Problems in AI (based on solution step recoverability):**
  - **Ignorable:** Solution steps can be ignored (e.g., theorem proving).
  - **Recoverable:** Solution steps can be undone (e.g., 8-puzzle).
  - **Irrecoverable:** Solution steps cannot be undone (e.g., chess).
- **Problem Formulation:** Defining states, initial state, actions, transition model, goal test, path cost.
  - Example: Vacuum World `[FIG]`
    - **States:** Agent location and dirt location (e.g., 2 rooms, 2^2 dirt configs = 8 states).
    - **Initial State:** Any state can be assigned.
    - **Actions:** Move Left, Move Right, Suck.
    - **Goal Test:** All squares are clean.
    - **Path Cost:** Each step costs 1.
- **Properties of Search Algorithms:** `[PYQ]`
  - **Completeness:** Does the algorithm guarantee finding a solution if one exists?
  - **Optimality:** `[PYQ]` (e.g. A*) Does it find the best (least-cost) solution?
  - **Time Complexity:** How long does it take as a function of problem size?
  - **Space Complexity:** How much memory does it require?

15. Uninformed Search (Blind Search) Algorithms `[PYQ]`

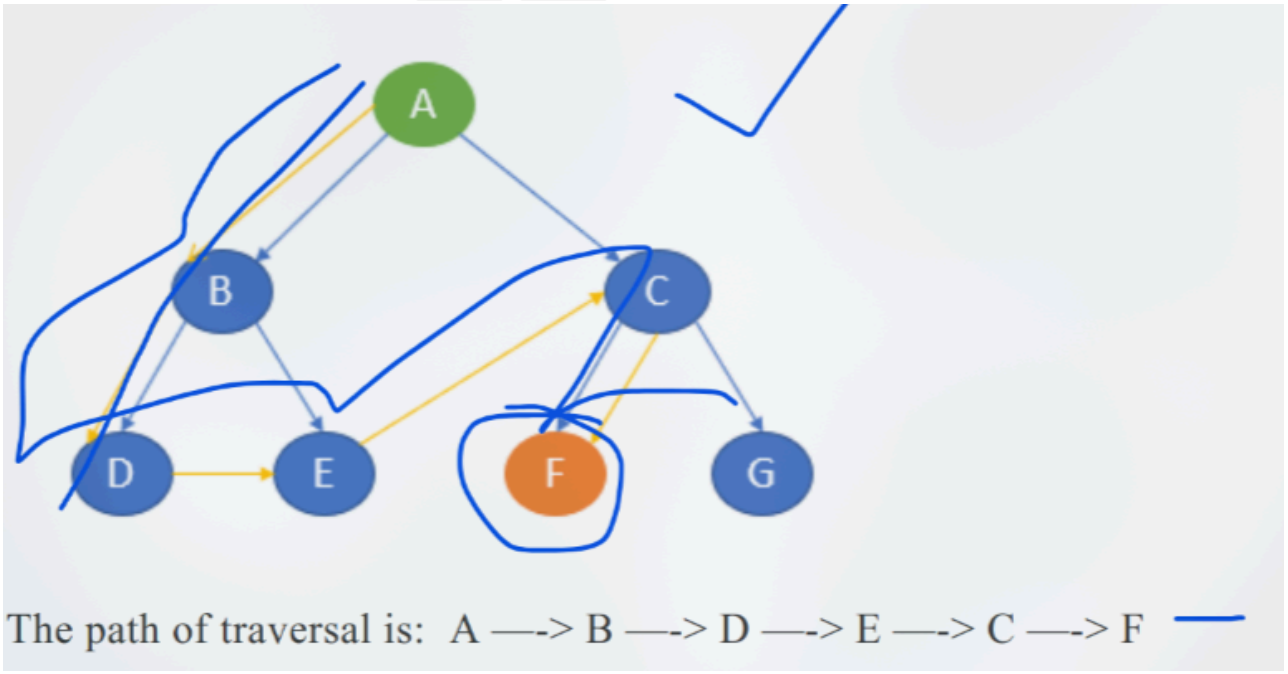
- No domain-specific knowledge beyond problem definition. Explores in a brute-force way.
- **Types:**

- **1. Breadth-First Search (BFS):** `[PYQ]` `[FIG]`



  - Explores level by level (shallowest nodes first). Uses FIFO queue.
  - **Completeness:** Yes.
  - **Optimality:** Yes (if path cost is a non-decreasing function of depth, e.g., all steps cost 1).
  - **Time Complexity:** O(b^d) (b=branching factor, d=depth of shallowest goal).
  - **Space Complexity:** O(b^d) (stores all nodes at current depth frontier).

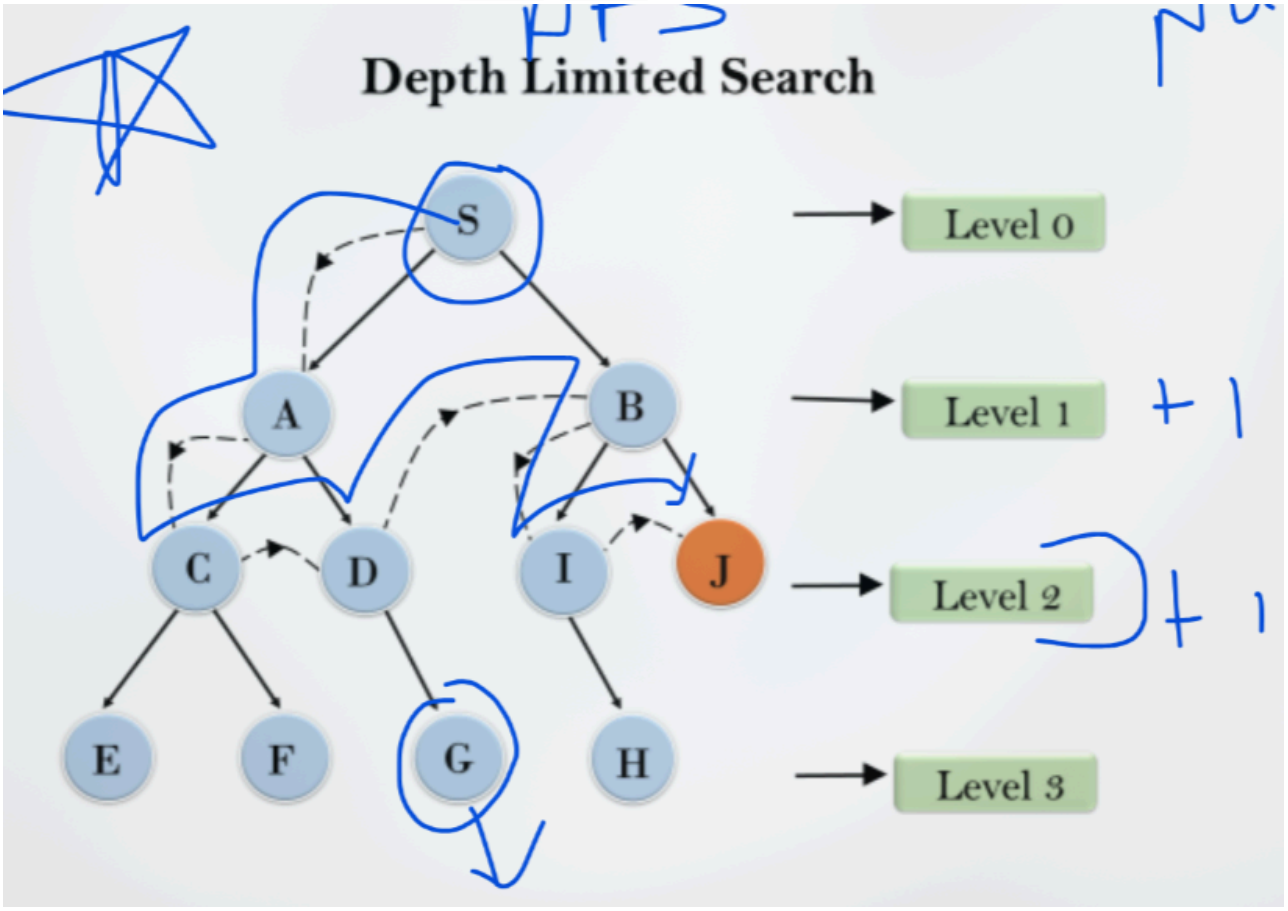
- **2. Depth-First Search (DFS):** `[PYQ]` `[FIG]`



  - Explores one branch of the tree as far as possible before backtracking. Uses LIFO stack (often implemented via recursion).
  - **Completeness:** No (can get stuck in infinite loops on infinite graphs); Yes for finite graphs or if loop detection is used.

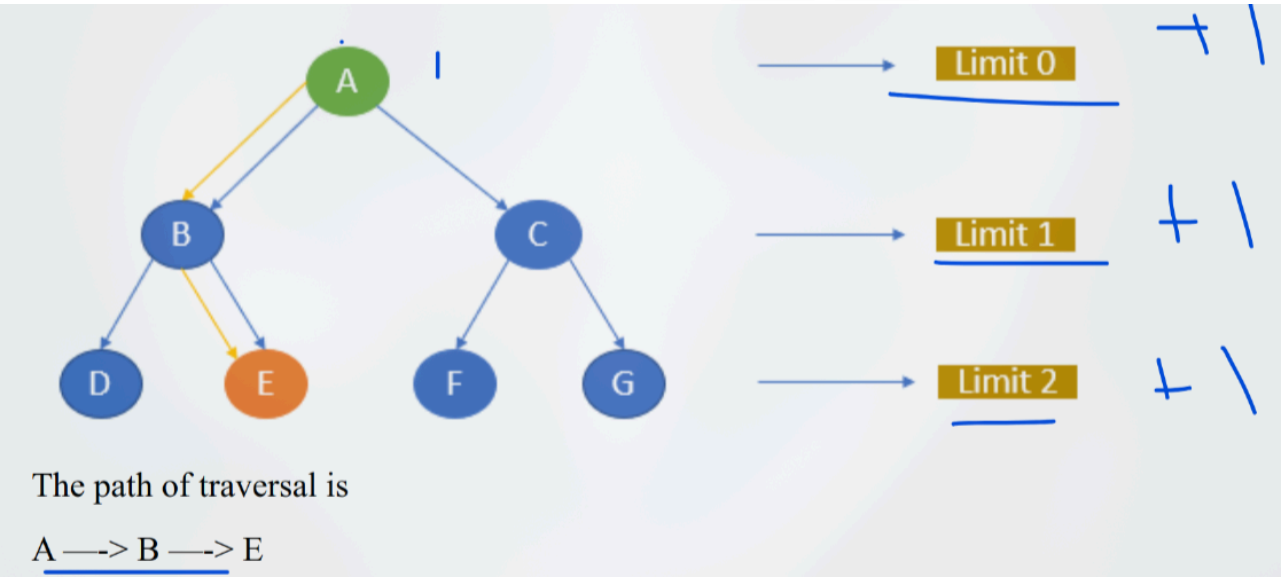
- **Optimality:** No.
- **Time Complexity:** O(b^m) (m=maximum depth of the state space).
- **Space Complexity:** O(b*m) (stores only nodes on the current path).

- **3. Depth-Limited Search (DLS):** `[PYQ]` `[FIG]`



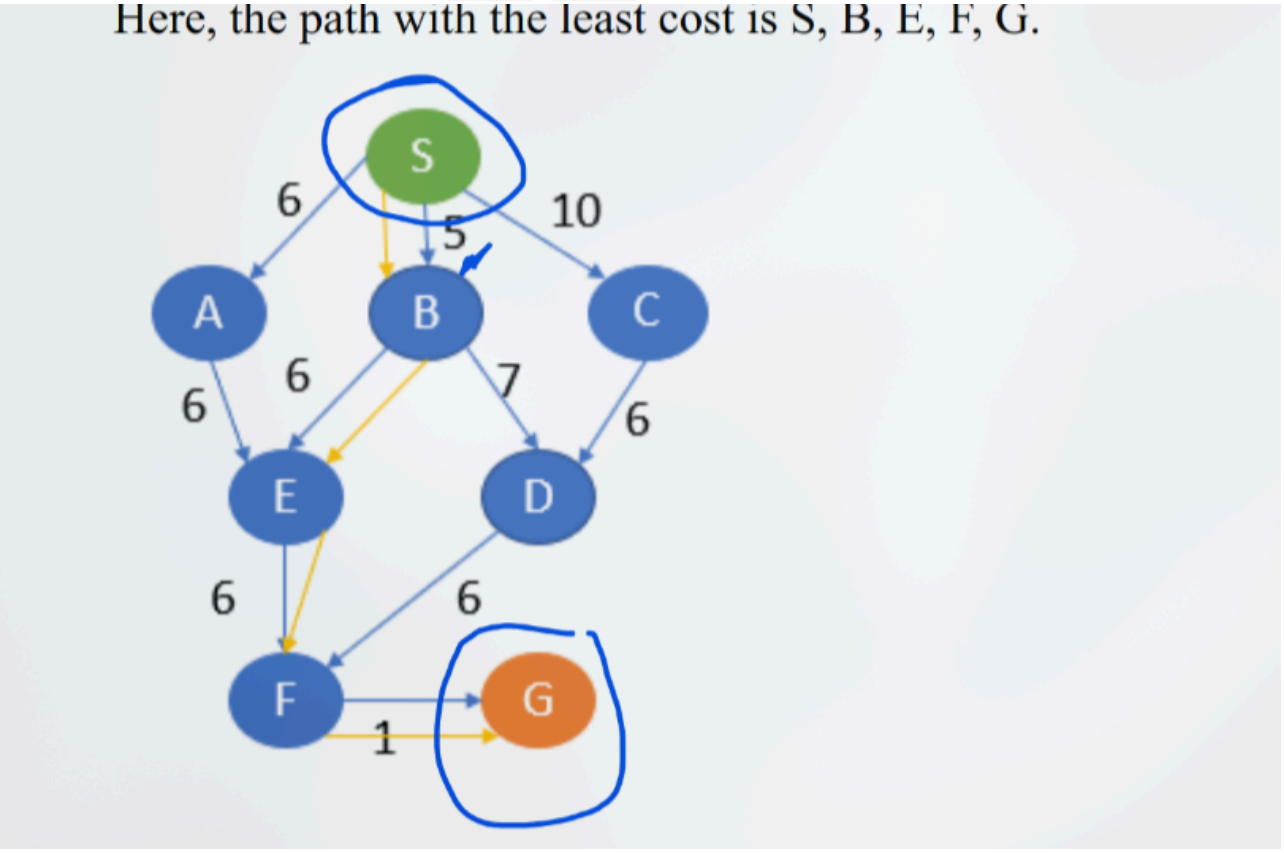
- DFS with a pre-defined depth limit (l). Solves the infinite loop problem of DFS in infinite state spaces.
- **Completeness:** Yes, if l >= d. No, if l < d (goal is deeper than limit).
- **Optimality:** No.
- **Time Complexity:** O(b^l).
- **Space Complexity:** O(b*l).

- **4. Iterative Deepening Depth-First Search (IDDFS):** `[PYQ]` `[FIG]`



  - Combines benefits of BFS (completeness, optimality for unit costs) and DFS (space efficiency).
  - Performs DLS repeatedly with increasing depth limits (0, 1, 2, ..., d).
  - **Completeness:** Yes.
  - **Optimality:** Yes (if step costs are uniform).
  - **Time Complexity:** O(b^d).
  - **Space Complexity:** O(b*d).

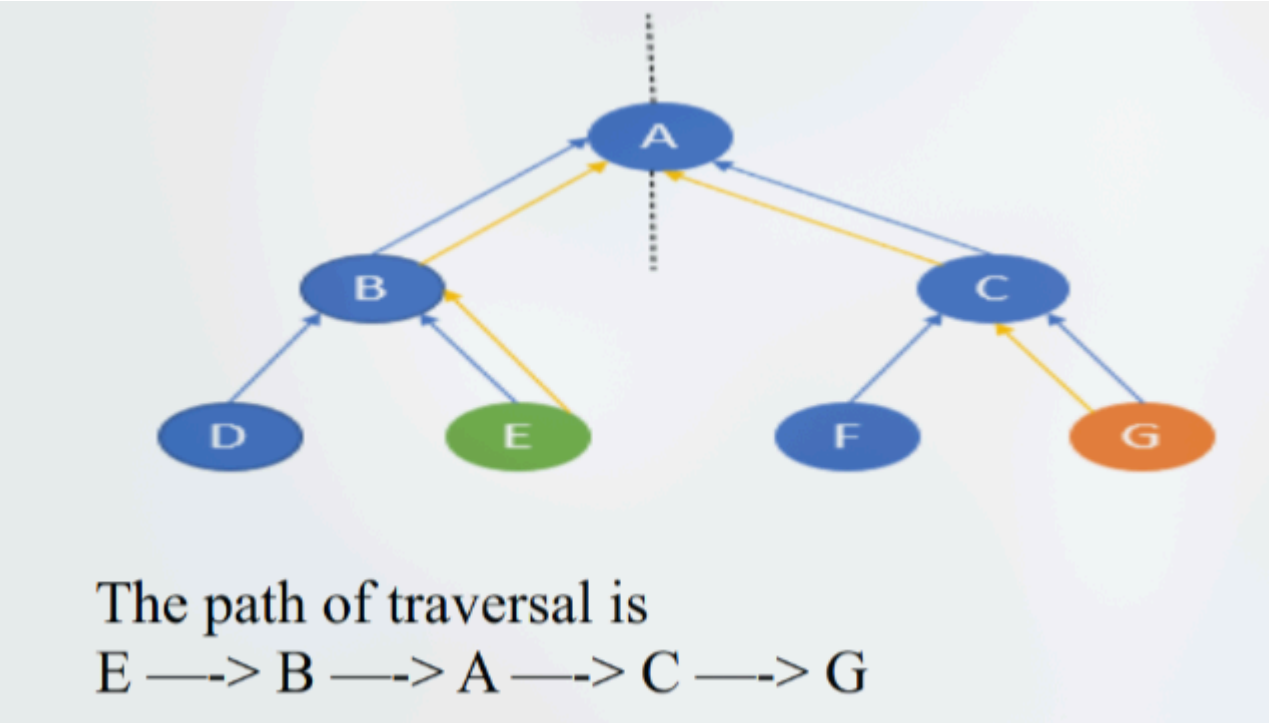
- **5. Uniform Cost Search (UCS):** `[PYQ]` `[FIG]`



  - Expands the node with the lowest path cost (g(n)) from the start. Uses a priority queue.
  - Finds the cheapest path in graphs where edge costs can vary and are non-negative.

- **Completeness:** Yes (if step costs > 0 or a very small positive ε).
- **Optimality:** Yes.
- **Time Complexity:** O(b^(1 + C*/ε)) (C*=cost of optimal solution, ε=minimum step cost).
- **Space Complexity:** O(b^(1 + C*/ε)).

- **6. Bidirectional Search:** `[FIG]`

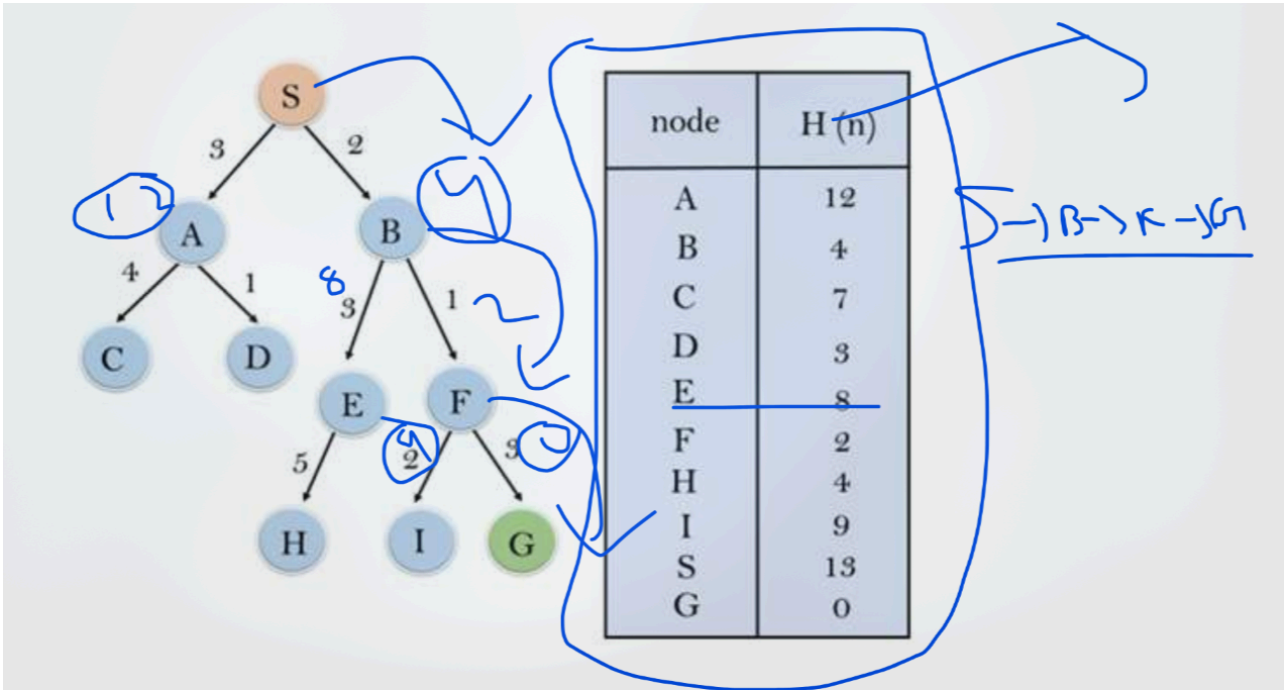


- Runs two simultaneous searches: one forward from the initial state, one backward from the goal state.
- Stops when the two searches meet in the middle.
- Reduces search complexity (roughly 2 * b^(d/2) instead of b^d).
- Requires an explicit goal state and ability to reverse actions.

16. Informed (Heuristic) Search Algorithms `[PYQ]`

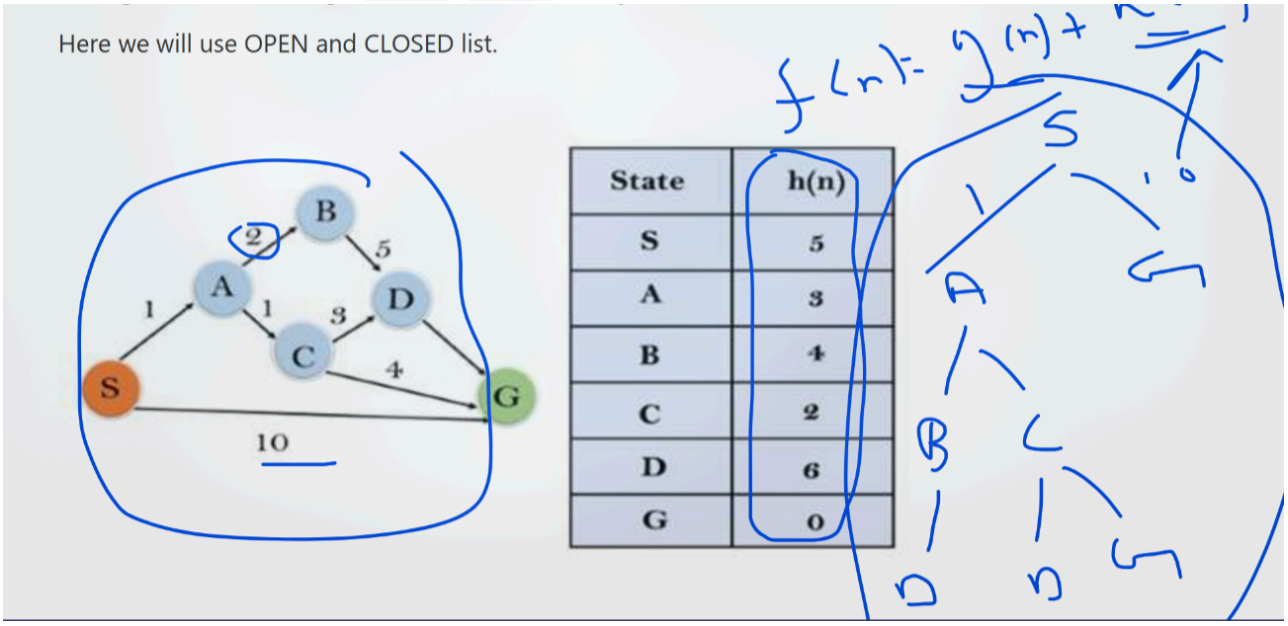
- Use problem-specific knowledge (heuristics) to guide the search more efficiently.
- **Heuristic function (h(n)):** `[PYQ]` An estimate of the cost from the current state 'n' to the nearest goal state.
  - Value is always positive or zero (h(goal)=0).
  - **Admissibility:** `[PYQ]` h(n) <= h*(n) (where h*(n) is the true cost to the nearest goal). An admissible heuristic never overestimates the cost.
  - **Consistency (Monotonicity):** For every node n and any successor n' generated by an action a, h(n) <= cost(n,a,n') + h(n'). This is a stronger condition than admissibility. If h is consistent, f(n) is non-decreasing along any path.
- **Pure Heuristic Search:** Expands the node with the lowest h(n) value. It's essentially Greedy Best-First Search. Not optimal or complete.
- **Types:**

- **1. Greedy Best-First Search [Best-first Search]:** `[PYQ]` `[FIG]`



  - Expands the node that *appears* to be closest to the goal, based solely on the heuristic h(n).
  - Uses a priority queue, ordering nodes by h(n). Evaluation function f(n) = h(n).
  - **Completeness:** No (can get stuck in loops, similar to DFS).
  - **Optimality:** No.
  - **Time Complexity:** O(b^m) in the worst case.
  - **Space Complexity:** O(b^m) in the worst case (keeps all nodes in memory).

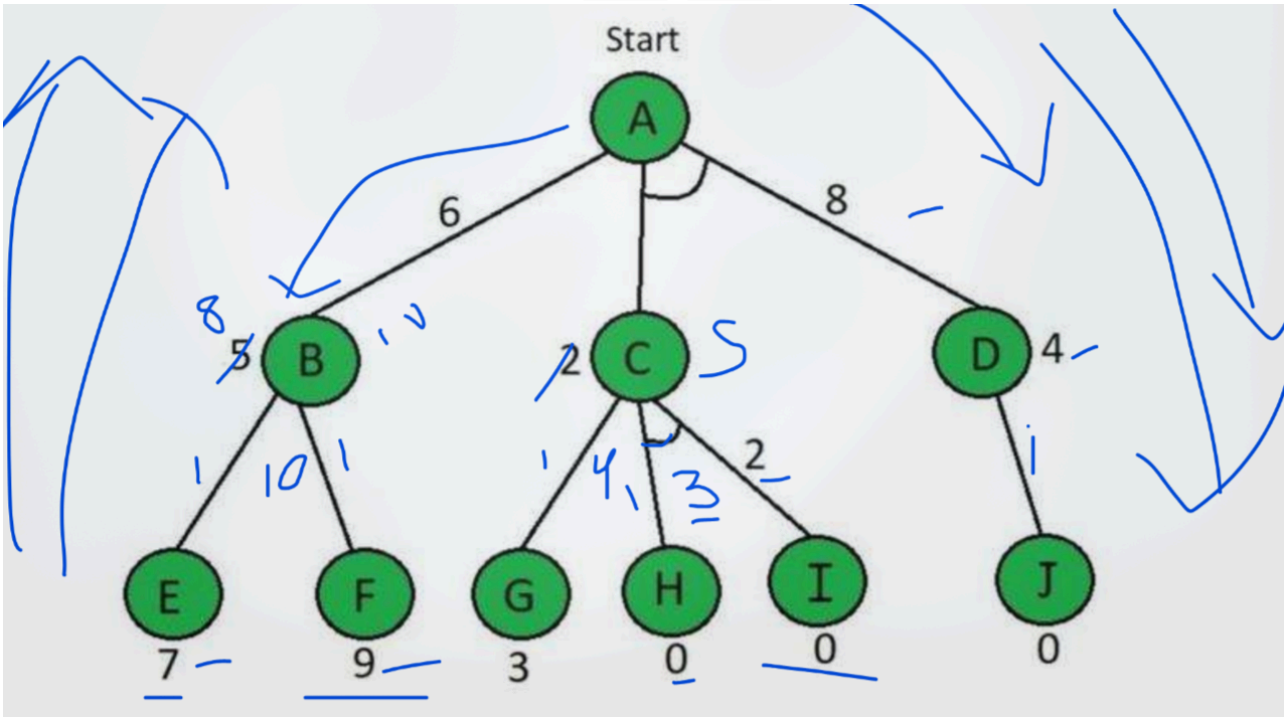
- **2. A'*' Search Algorithm:** `[PYQ]` `[FIG]`



  - Combines Uniform Cost Search (g(n) - actual cost from start to node n) and Greedy Best-First Search (h(n) - estimated cost from n to goal).
  - Evaluation function: **f(n) = g(n) + h(n)**. `[PYQ]`
  - Expands the node with the lowest f(n) value from the OPEN list (priority queue).
  - **Completeness:** Yes.

- **Optimality:** `[PYQ]` Yes, if h(n) is admissible (for tree search) or consistent (for graph search, which is more common in practice).
- **Time Complexity:** Exponential in the worst case, but can be much better with a good heuristic.
- **Space Complexity:** Exponential (keeps all generated nodes in memory, which is its main drawback).

- **3. AO'*' Algorithm (for AND-OR graphs):** `[PYQ]` `[FIG]`

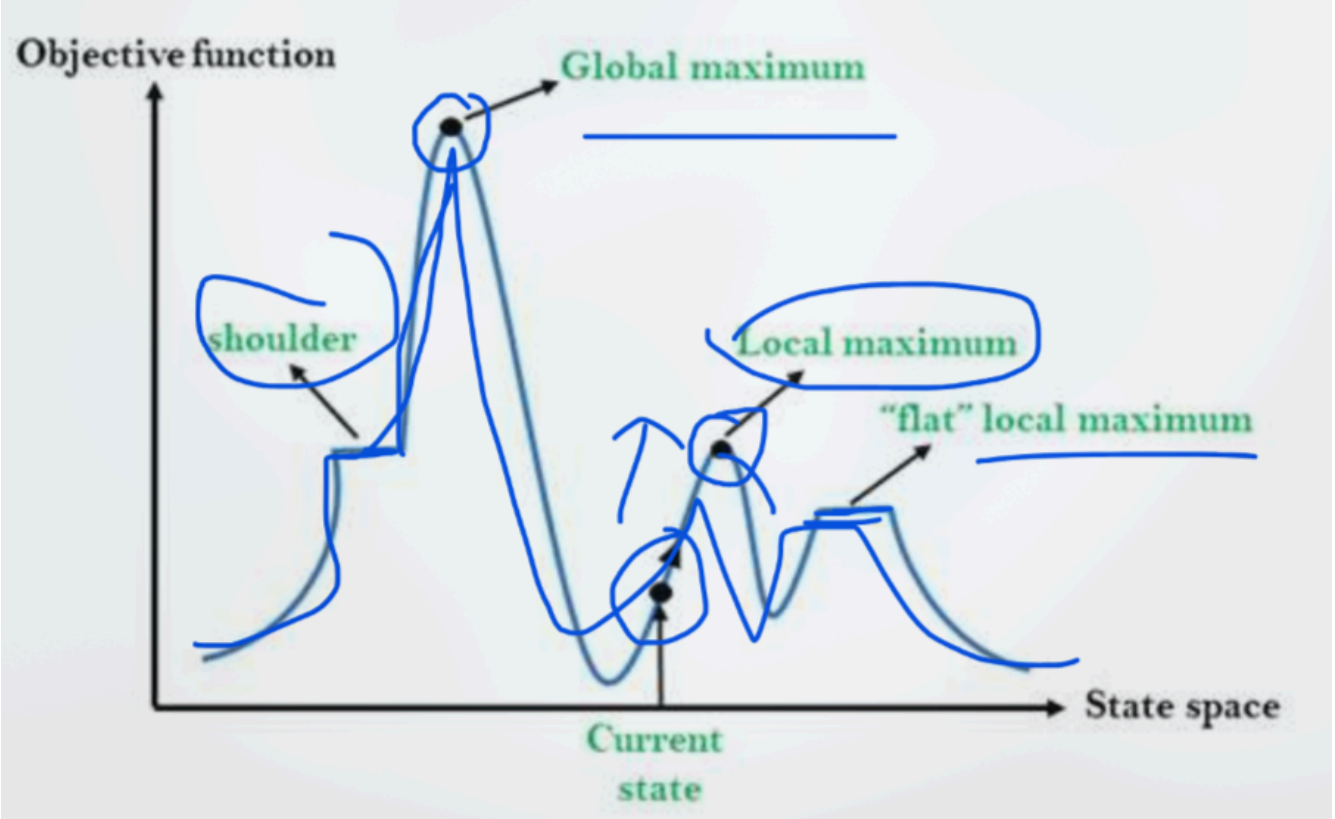


- Used for problems that can be decomposed into subproblems.
- **AND nodes:** All subproblems must be solved.
- **OR nodes:** Any one of the alternative subproblems can be solved.
- A best-first search algorithm applied to AND-OR graphs.
- Divides a problem into smaller subproblems and uses heuristics. Evaluation f(n) = g(n) + h(n) (cost definitions can be complex for AND-OR).
- More effective in searching AND-OR trees than A* (A* is for state-space graphs).
- Doesn't guarantee optimality in the same way A* does for standard state spaces, can use less memory, designed to avoid endless loops in cyclic AND-OR graphs.
- Example: "Want to buy a car"
  - OR node: [Steal car], [Buy with own money]
  - If [Buy with own money] is chosen: AND node: [Get money], [Find a car to buy]

17. Hill Climbing Algorithm `[PYQ]` `[FIG]`

- A local search algorithm that continuously moves in the direction of increasing value (uphill) to find the peak of the mountain or the best solution to the problem.
- Terminates when it reaches a peak value where no neighbor has a higher value.

- It's a greedy approach; it only looks at its immediate neighbors. No backtracking.
- Keeps only a single current state in memory.
- **State-space Diagram for Hill Climbing:** `[FIG]`

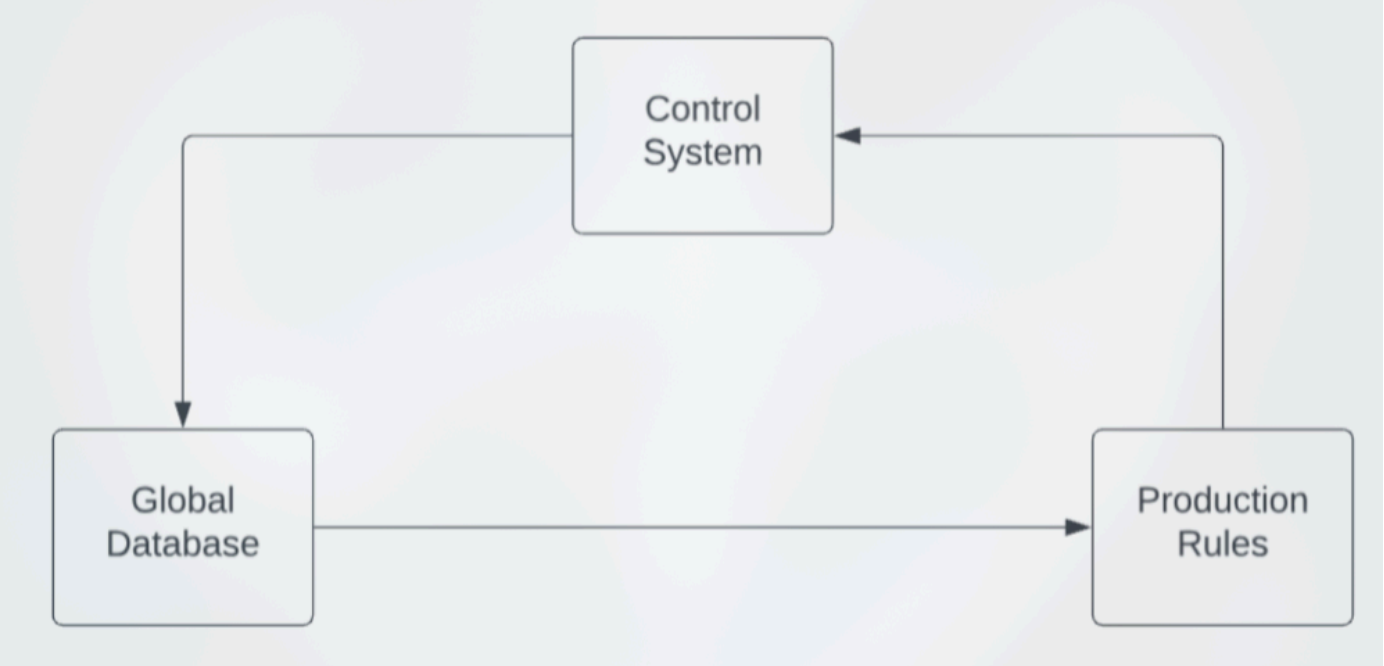


- A graphical representation showing states vs. Objective function/Cost.
  - **Local Maximum:** A state better than all its neighbors but not the best overall.
  - **Global Maximum:** The best possible state in the landscape.
  - **Flat local maximum (Plateau):** A flat region where all neighbors have the same value as the current state.
  - **Shoulder:** A plateau region that has an uphill edge.
  - **Ridge:** A path where movement along it might be uphill, but any move off the ridge is downhill.
- **Types of Hill Climbing:**
  - **Simple Hill Climbing:** Evaluates neighbor nodes one by one and selects the first one that is better than the current state.
  - **Steepest-Ascent Hill Climbing:** `[PYQ]` Examines all neighboring nodes of the current state and selects the one that is closest to the goal state (i.e., the best successor). Consumes more time.
  - **Stochastic Hill Climbing:** Does not examine all neighbors. Randomly selects one neighbor and decides whether to choose it or examine another.
- **Problems in Hill Climbing:** `[PYQ]`
  - **Local Maximum:** `[PYQ]` Algorithm gets stuck on a peak that isn't the global maximum. Solution: Backtracking, random restart hill climbing.

- **Plateau:** `[PYQ]` All neighbors have the same value, making it hard to choose a direction. Solution: Make a big step/random jump.
- **Ridge:** `[PYQ]` A narrow path of higher elevation. Steps off the ridge are downhill. Solution: Use bidirectional search or allow multiple moves before re-evaluating.

- **Simulated Annealing:** `[PYQ]`
  - A variation that attempts to overcome local maxima.
  - Allows occasional "bad" moves (to a state with lower value) with a probability that decreases over time (controlled by a "temperature" parameter).
  - Aims for both efficiency and completeness (finding the global optimum if cooled slowly enough).

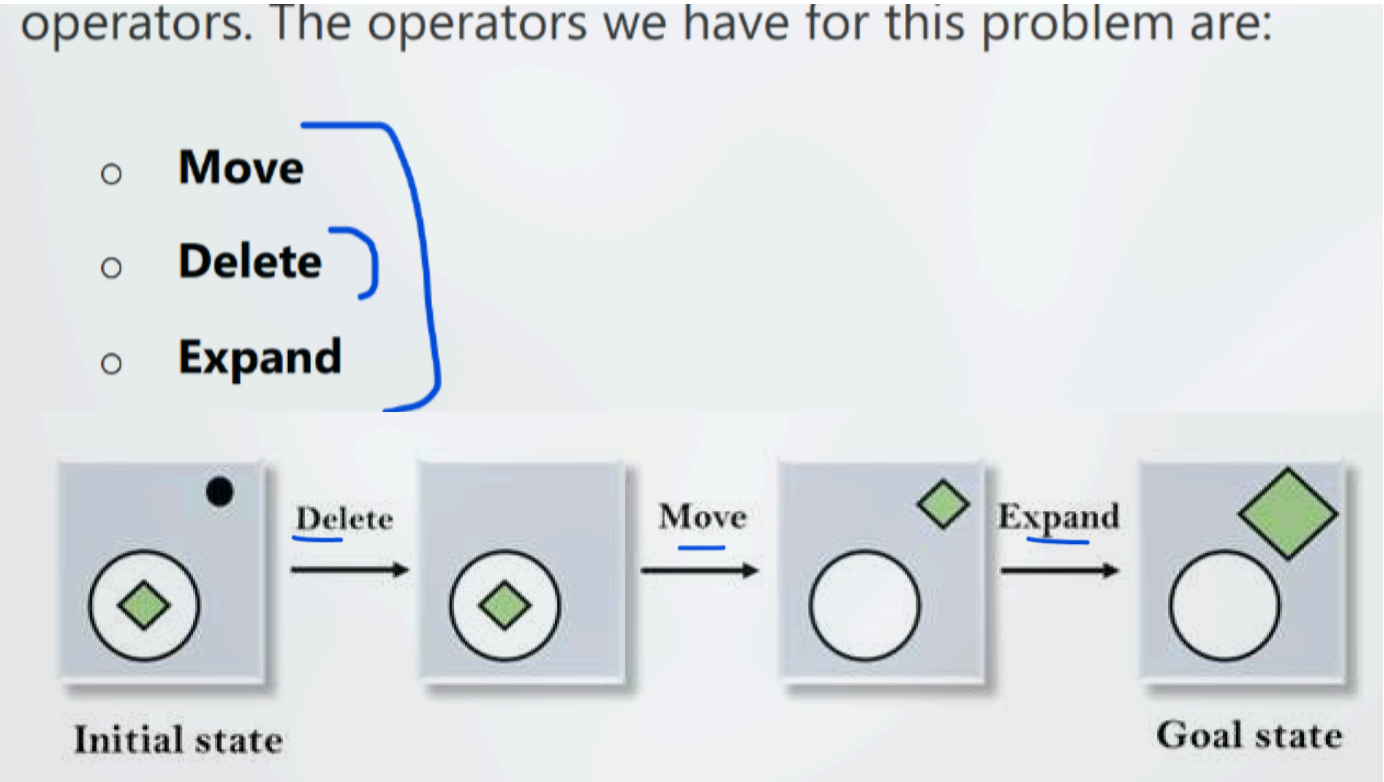
18. Production Systems (Rule-Based Systems) `[PYQ]` `[FIG]`



- Based on a set of rules about behavior, typically in IF-THEN format.
- Used in expert systems, automated planning, and action selection.
- **Components:** `[PYQ]` `[FIG]`
  - **Global Database (Working Memory):** Central data structure containing facts about the current state of the world.
  - **Set of Production Rules:** Rules that operate on the global database. Each rule has a precondition (IF part) and an action (THEN part). If the precondition is satisfied by the database, the rule can be applied, changing the database.
  - **Control System (Inference Engine):**
    - Chooses which applicable rule to "fire" if multiple rules match (conflict resolution).
    - Determines when to stop computation (e.g., when a goal is reached or no more rules apply).
- **Types of Inference Rules:** `[FIG]`

    - **Deductive Inference Rules:** Logic used to draw specific conclusions from general premises or facts. (e.g., Modus Ponens).
    - **Abductive Inference Rules:** Used to make educated guesses or hypotheses based on observed data; generating plausible explanations.
- **Features of Production Systems:**
    - **Simplicity:** Uniform IF-THEN structure for knowledge representation.
    - **Modularity:** Knowledge is coded in discrete pieces (rules), easy to add/delete.
    - **Modifiability:** Facility for modifying rules.
    - **Knowledge-intensive:** Knowledge base stores pure knowledge, separate from control.
- **Classes of Production Systems:** `[PYQ]`
    - **Monotonic Production System:** Application of a rule never prevents the later application of another rule that could also have been applied. Once a fact is true, it remains true.
    - **Non-Monotonic Production System:** `[PYQ]` Application of a rule may prevent later application of another. Facts can be retracted. More dynamic and adaptive.
    - **Partially Commutative Production System:** Rules can be applied with some flexibility in order, but the order might still matter for efficiency or intermediate states.
    - **Commutative Production System:** The order of rule application does not affect the final outcome.
- **Applications:** Customer support chatbots, fraud detection systems, medical diagnosis systems, traffic management systems. `[PYQ]`
- **Water Jug Problem as a Production System Example:** `[PYQ]` `[FIG]`
    - **Goal:** Obtain a specific amount of water (e.g., 4 liters) in a designated jug (e.g., 5-liter jug), given jugs of fixed capacities (e.g., 5l and 3l) and a water source.
    - **Rules (Operators):** Fill jug X, Empty jug X, Pour water from jug X to jug Y until Y is full or X is empty.
    - **Global Database:** Current amount of water in each jug (e.g., [x, y]).

19. Means-Ends Analysis (MEA) `[PYQ]` `[FIG]`



- A problem-solving technique that combines forward and backward reasoning.
- Focuses on reducing the "difference" between the current state and the goal state.
- **Steps:**
  1. Compare CURRENT state to GOAL state. If no differences, return Success.
  2. Else, select the most significant difference.
  3. Select an operator O relevant to reducing this difference. If no such operator, signal failure.
  4. Attempt to apply operator O:
     - **Operator Subgoaling:** If O's preconditions are not met in CURRENT, set a sub-goal to reach a state (O-START) where they are met. Recursively call MEA(CURRENT, O-START).
     - If successful, apply O to get O-RESULT.
     - Recursively call MEA(O-RESULT, GOAL) to solve the remaining problem.
     - If both recursive calls are successful, return the combined plan.
- Example: Planning a trip. Difference: Current location vs. Destination. Operator: Fly. Precondition: Have ticket. Sub-goal: Buy ticket.

20. Constraint Satisfaction Problems (CSP) `[PYQ]`

- Defined by:
  - A set of **Variables** (X1, ..., Xn).
  - For each variable Xi, a **Domain** Di of possible values.
  - A set of **Constraints** C that specify allowable combinations of values for subsets of variables.
- **Goal:** To find an assignment of a value to each variable from its domain such that all constraints are satisfied.

- **Examples:** `[PYQ]`
  - Map Coloring: Assign colors to regions so no adjacent regions have the same color.
  - N-Queens Problem: Place N queens on an N×N chessboard so no two queens attack each other.
  - Sudoku.
  - Cryptarithmetic Puzzles (e.g., SEND + MORE = MONEY). `[PYQ]`
- **Solution Methods:** Typically involve search (e.g., backtracking search) often enhanced by:
  - Heuristics (e.g., most constrained variable, least constraining value).
  - Inference/Constraint Propagation (e.g., forward checking, arc consistency) to prune the search space.