MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by Naac with A Grade & TCS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

AL3391 - ARTIFICIAL INTELLIGENCE

UNIT I

UNIT 1- INTELLIGENT AGENTS

Introduction to AI – Agents and Environments –concept of rationality – nature of environments –structure of agents. Problem solving agents – search algorithms – uninformed search strategies.

PART-A

1. What is Artificial Intelligence?

Define AI and write the applications of AI. Nov 2023

- Artificial intelligence is a wide-ranging branch of computer science concerned with building smart machines capable of performing tasks that typically require human intelligence.
- Artificial Intelligence is the process of building intelligent machines from vast volumes of data.
- Systems learn from past learning and experiences and perform humanlike tasks.
- It enhances the speed, precision, and effectiveness of human efforts.

Applications

- Transportation: AI is being used to develop self-driving cars and improve traffic management.
- Energy: AI is being used to improve energy efficiency and predict energy demand.
- Government: AI is being used to improve public safety, detect crime, and provide citizen services.

2. How Artificial Intelligence is defined? List the Types of Approaches for Artificial Intelligence.

- Thinking humanly - mimicking thought based on the human mind.

- Thinking rationally - mimicking thought based on logical reasoning.
- Acting humanly - acting in a manner that mimics human behavior.
- Acting rationally - acting in a manner that is meant to achieve a particular goal.

3. What Are the Four Types of Artificial Intelligence?

- Reactive machines - able to perceive and react to the world in front of it as it performs limited tasks.
- Limited memory - able to store past data and predictions to inform predictions of what may come next.
- Theory of mind - able to make decisions based on its perceptions of how others feel and make decisions.
- Self-awareness - able to operate with human-level consciousness and understand its own existence.

4. List the types of Artificial Intelligence-based on capabilities.

- Narrow AI
- General AI
- Super AI

5. List the types of Artificial Intelligence-based on functionalities -

- Purely Reactive or Reactive Machines
- Limited Memory
- Theory of Mind
- Self-Awareness

6. What are the 4 functionalities of AI?

Reactive Machines	Limited Memory	Theory of Mind	Self-Awareness
Simple classification and pattern recognition tasks	Complex classification tasks	Understands human reasoning and motives	Human-level intelligence that can by-pass human intelligence too
Great when all parameters are known	Uses historical data to make predictions	Needs fewer examples to learn because it understands motives	Sense of self-consciousness
Can't deal with imperfect information	Current state of AI	Next milestone for the evolution of AI	Does not exist yet

6. List the Four possible goals to pursue in artificial intelligence.

- Systems that think like humans.
- Systems that think rationally.

- Systems that act like humans.
- Systems that act rationally.

7. Point out the subfields of Artificial Intelligence.

- Machine Learning.
- Deep Learning.
- Neural Networks.
- Cognitive Computing.
- Computer Vision.
- Natural Language Processing.

8. What are the advantages of Artificial Intelligence?

- Reduced human error
- Risk avoidance
- Replacing repetitive jobs
- Digital assistance

9. List the limitations of Artificial Intelligence

- High cost of creation.
- No emotions.
- Box thinking.
- Can‘t think for itself.

10. Compare Data Science and Artificial Intelligence

- Data science combines statistical tools, methods, and technology to generate meaning from data.
- Artificial Intelligence takes this one step further and uses the data to solve cognitive problems commonly associated with human intelligence, such as learning, pattern recognition, and human-like expression.

11. List the various Applications of Artificial Intelligence or AI Applications.

AI Application in E-Commerce

- Personalized Shopping
- AI-powered Assistants
- Fraud Prevention

AI Application in Education

- Administrative Tasks Automated to Aid Educators
- Creating Smart Content
- Voice Assistants

- Personalized Learning

12. What are the Goals of Artificial Intelligence?

Main goals of Artificial intelligence. They are,

1. Replicate human intelligence.
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:

   a. Proving a theorem

   b. Playing chess

   c. Plan some surgical operation

   d. Driving a car in traffic

Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

13. Define Rational Agent.

- A rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.
- Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

14. What is the Structure of an AI Agent?

- The task of AI is to design an agent program which implements the agent function.
- The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

a. **Architecture:** Architecture is machinery that an AI agent executes on.

b. **Agent Function:** Agent function is used to map a percept to an action

   i. **$f:P^* \rightarrow A$**

c. **Agent program:** Agent program is an implementation of agent function.

15. Define PEAS Representation.

- PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

  P: Performance measure

  E: Environment

  A: Actuators

  S: Sensors

- Here performance measure is the objective for the success of an agent's behavior.

16. Define an omniscient agent.

- An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.

17. Define Goal based Agents.

- An Agent knows the description of current state as well as goal state.
- The action matches with the current state is selected depends on the goal state.

18. Give the component of learning Agent.

- A learning agent can be divided into four conceptual components:
- Learning element.
- performance element.
- Critic.
- Problem generator.

19. Define an Agent and list its types.

Agent

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

Types of Agents

- **Simple reflex agents** respond directly to percepts.
- **Model-based reflex** agents maintain internal state to track aspects of the world that are not evident in the current percept.

- **Goal-based agents** act to achieve their goals, and
- **Utility-based agents** try to maximize their own expected "happiness".

20. Define an environment and task environment.

Environment - The environment could be everything—the entire universe.

Task Environment - A task environment specification includes the performance measure, the external environment, the actuators, and the sensors.

21. List the steps in designing an agent

- In designing an agent, the first step must always be to specify the task environment as fully as possible.
- The agent program implements the agent function.
- There exists a variety of basic agent program designs reflecting the kind of information made explicit and used in the decision process.
- The designs vary in efficiency, compactness, and flexibility.
- The appropriate design of the agent program depends on the nature of the environment.
- All agents can improve their performance through learning.

22. Define Problem Solving Agent. List the steps involved in Problem Solving Agent.

Problem Solving Agent

- Problem-solving agent is a goal-based agent that focuses on goals using a group of algorithms and techniques to solve a well- defined problem.
- An agent may need to plan a sequence of actions that form a path to a goal state.
- Such an agent is called a problem-solving agent, and the computational process it undertakes is called search.

Steps performed by Problem-solving agent

- Goal Formulation
- Problem Formulation
- Search
- Solution
- Execution

23. List the Components in problem formulation.

- Initial State: It is the starting state or initial step of the agent towards its goal.
- Actions: It is the description of the possible actions available to the agent.
- Transition Model: It describes what each action does.
- Goal Test: It determines if the given state is a goal state.

- Path cost: It assigns a numeric cost to each path that follows the goal.

24. Point out the subfields of Artificial Intelligence.

- Machine Learning
- Deep Learning
- Neural Networks
- Cognitive Computing
- Computer Vision

25. Define search algorithms and list the types of search algorithm.

- A search algorithm takes a search problem as input and returns a solution.

Types of Search Algorithm

Uninformed Search Algorithms

- **o** Depth First Search
- **o** Depth Limited Search
- **o** Breadth First Search
- **o** Iterative Deepening Search
- **o** Uniform Cost Search
- **o** Bidirectional Search

Informed (Heuristic) Search Strategies

- **o** Greedy Search
- **o** A* Tree Search

26. Define Depth First Search. List the advantages and disadvantages of DFS.

- Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures.
- The algorithm starts at the root node and explores as far as possible along each branch before backtracking.
- It uses last in-first-out strategy and hence it is implemented using a stack.

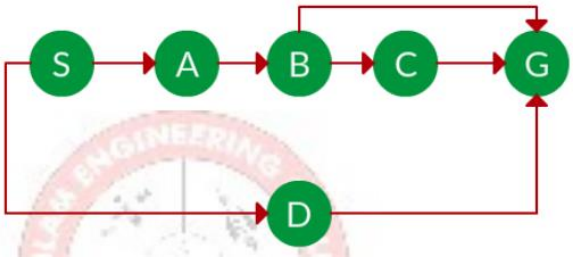
Advantage of Depth-first Search:

- DFS uses extremely little memory.
- It takes less time to reach the goal node than the BFS method.

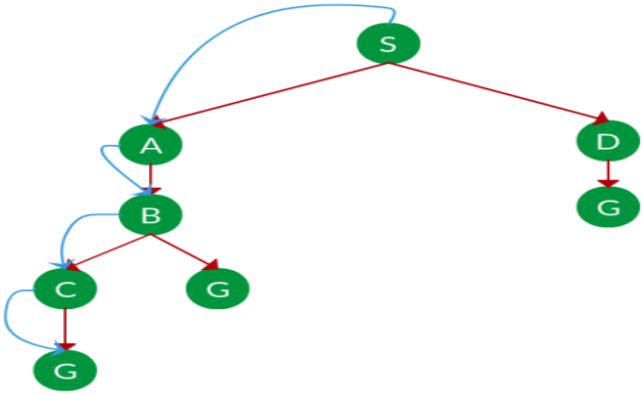
Disadvantage of Depth-first Search:

- There's a chance that many states will recur, and there's no certainty that a solution will be found.
- The DFS algorithm performs deep searching and may occasionally enter an infinite cycle.

27. Which solution would DFS find to move from node S to node G if Run on the graph below?



Solution



Path: S -> A -> B -> C -> G

28. Define Depth Limited Search. List its advantages and disadvantages.

- A depth-limited search algorithm is similar to depth-first search with a predetermined limit.
- Depth-limited search can solve the drawback of the infinite path in the Depth-first search.

Advantage

- Depth-limited search is Memory efficient.

Disadvantage

- Depth-limited search also has a disadvantage of incompleteness.
- It may not be optimal if the problem has more than one solution.

29. Define Breadth-first search (BFS). List its advantages and disadvantages.

- Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

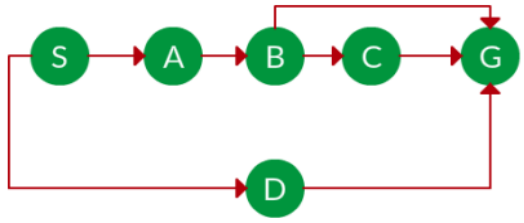
Advantages of Breadth-first Search

- If a solution is available, BFS will provide it.
- If there are multiple answers to a problem, BFS will present the simplest solution with the fewest steps.

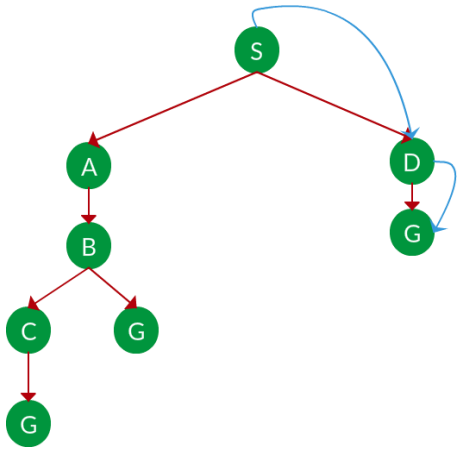
Disadvantages of Breadth-first Search

- It necessitates a large amount of memory.
- If the solution is located far from the root node, BFS will take a long time.

30. Which solution would BFS find to move from node S to node G if run on the graph below?



Solution



Path: S -> D -> G

31. Define iterative deepening depth-first search/ iterative deepening search

- This search is a combination of BFS and DFS, as BFS guarantees to reach the goal node and DFS occupies less memory space.
- Therefore, iterative deepening search combines these two advantages of BFS and DFS to reach the goal node.
- It gradually increases the depth-limit from 0,1,2 and so on and reach the goal node.

32. Define Uniform Cost Search. List its advantages and disadvantages.

- A searching algorithm for traversing a weighted tree or graph is uniform-cost search.
- When a separate cost is provided for each edge, this algorithm is used.
- The uniform-cost search's main purpose is to discover the shortest path to the goal node with the lowest cumulative cost.
- Cost of a node is defined as:

cost(node) = cumulative cost of all nodes from root

cost(root) = 0

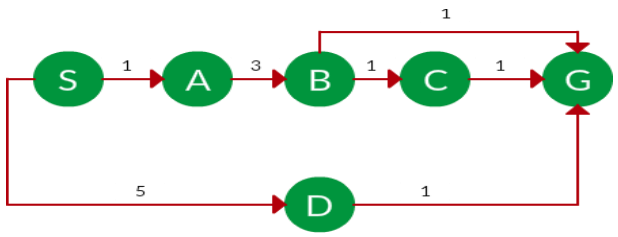
Advantages of Uniform-cost Search

- The path with the lowest cost is chosen at each state.
- UCS is complete only if states are finite and there should be no loop with zero weight.
- UCS is optimal only if there is no negative cost.

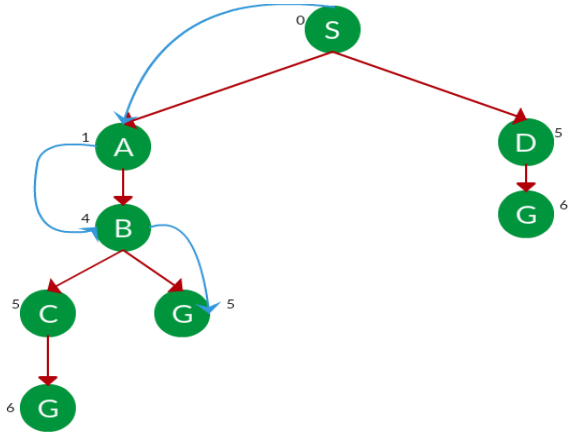
Disadvantages of Uniform-cost Search

- It is just concerned with the expense of the path.

33. Which solution would UCS find to move from node S to node G if run on the graph below?



Solution



Path: S -> A -> B -> G

Cost: 5

34. Define Bidirectional Search.

- The strategy behind the bidirectional search is to run two searches simultaneously--**one forward search from the initial state and other from the backside of the goal**--hoping that both searches will meet in the middle.

35. Define Agent Environment in AI.

- An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself.
- An environment can be described as a situation in which an agent is present.
- The environment is where agent lives, operate and provide the agent with something to sense and act upon it.

36. What are the Features of Environment?

- Environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable.
2. Static vs Dynamic.
3. Discrete vs Continuous.
4. Deterministic vs Stochastic.
5. Single-agent vs Multi-agent.
6. Episodic vs sequential.
7. Known vs Unknown.
8. Accessible vs Inaccessible.

37. What is Goal-based agents?

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.

38. Define Utility-based agent.

- Utility-based agent are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.

39. What are the Properties of Search Algorithms?

- Completeness.
- Optimality.
- Time Complexity.
- Space Complexity.

40. List down the characteristics of intelligent agent.

- The IA must learn and improve through interaction with the environment.
- The IA must adapt online and in the real time situation.
- The IA must accommodate new problem-solving rules incrementally.
- The IA must have memory which must exhibit storage and retrieval capabilities.

41. What is Model-based reflex agent?

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
  a. **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
  b. **Internal State:** It is a representation of the current state based on percept history.

42. Write a function for the table driven agent. (Or) Define basic agent programs. (MAY 2013)

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  static: percepts, a sequence, initially empty
          table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

43. What are learning agents?

- A learning agent can be divided into four conceptual components. The most important distinction is between the learning element, which is responsible for making improvements, and the performance element, which is responsible for selecting external actions.
- The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
- The learning element uses CRITIC feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.

44. There are well-known classes of problems that are intractably difficult for computers and other classes that are provably undecidable. Does this mean that AI is impossible? [Apr 2024]

- No , AI is always possible. The progress made so far can be executed on computers, in which some tasks perform better than humans and some perform worse than human beings.
- The social impact of AI is considered to be smaller in 1963 to 2013 than the technical advances between 1890 and 1940. The comparison is listed in table given below: S. No. Social Impact Technical Advances
  1. Thousands of workers lost their jobs because of automated machines which were considered to be a part of AI. The work became easier to do as the work load now was to be handled by machines mostly.
  2. Large industries quickly adopted robotic technologies and thus the production in the market increased. Robotic technologies were highly efficient in giving accurate information without producing any error.

46. Formulate PEAS for an automated Bill paying system. [Apr 2024]

PEAS stands for Performance measures, Environment, Actuators, and Sensors. We shall see what these terms mean individually.

- **Performance measures:** These are the parameters used to measure the performance of the agent. How well the agent is carrying out a particular assigned task.
- **Environment:** It is the task environment of the agent. The agent interacts with its environment. It takes perceptual input from the environment and acts on the environment using actuators.
- **Actuators**: These are the means of performing calculated actions on the environment. For a human agent; hands and legs are the actuators.

- **Sensors:** These are the means of taking the input from the environment. For a human agent; ears, eyes, and nose are the sensors.

47. List the steps involved in simple problem solving technique. Nov 2023

There are basically three types of problem in artificial intelligence:

- Ignorable: In which solution steps can be ignored.
- Recoverable: In which solution steps can be undone.
- Irrecoverable: Solution steps cannot be undo.

Steps problem-solving in AI:

- The problem of AI is directly associated with the nature of humans and their activities. So we need a number of finite steps to solve a problem which makes human easy works.

These are the following steps which require to solve a problem:

- Problem definition: Detailed specification of inputs and acceptable system solutions.
- Problem analysis: Analyse the problem thoroughly.
- Knowledge Representation: collect detailed information about the problem and define all possible techniques.
- Problem-solving: Selection of best techniques.

PART B

1. Explain in detail about Introduction to Artificial Intelligence.

Introduction to AI:

What is Artificial Intelligence?

- In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.
- Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines.The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.
- AI is one of the fascinating and universal fields of Computer science which has a great scope in
- Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines *"man-made,"* and intelligence defines *"thinking power"*, hence AI means *"a man-made thinking power."*
- So, we can define AI as:
- "It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."
- Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems
- With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.
- It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

Why Artificial Intelligence?

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing,

traffic issues, etc.

- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Goals of Artificial Intelligence

Following are the main goals of Artificial Intelligence:

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:
    - Proving a theorem
    - Playing chess
    - Plan some surgical operation
    - Driving a car in traffic
5. Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

What Comprises to Artificial Intelligence?

- Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it.
- To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc**.

To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:

- Mathematics

- Biology
- Psychology
- Sociology
- Computer Science
- Neurons Study
- Statistics

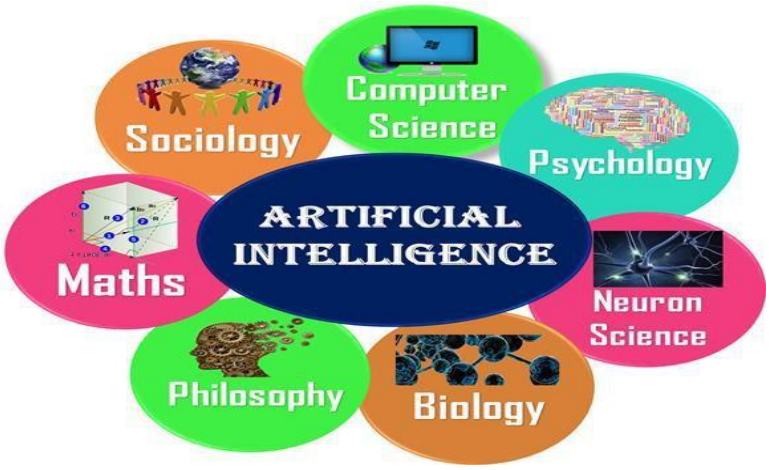


Fig.1.1 AI Discipline

Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- **High Accuracy with less errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for

security purpose, Natural language processing to communicate with the human in human- language, etc.

Disadvantages of Artificial Intelligence

Every technology has some disadvantages, and thesame goes for Artificial intelligence. Being so advantageous technology still, it has some disadvantages which we need to keep in our mind while creating an AI system. Following are the disadvantages of AI:

- **High Cost:** The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.
- **Can't think out of the box:** Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- **No feelings and emotions:** AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.
- **Increase dependency on machines:** With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.
- **No Original Creativity:** As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

Prerequisite

Before learning about Artificial Intelligence, you must have the fundamental knowledge of following so that you can understand the concepts easily:

- Any computer language such as C, C++, Java, Python, etc.(knowledge of Python will be an advantage)
- Knowledge of essential Mathematics such as derivatives, probability theory, etc.

Future of Artificial Intelligence:

1. Health Care Industries

- India is 17.7% of the worlds' population that makes it the second-largest country in terms of China's population. Health care facilities are not available to all individuals living in the country. It is because of the lack of good doctors, not having good infrastructure, etc. Still, there are people who couldn't reach to doctors/ hospitals. AI

has the ability to provide the facility to detect disease based on symptoms; even if you don't go to the doctor, AI would read the data from Fitness band/medical history of an individual to analyze the pattern and suggest proper medication and even deliver it on one's fingertips just through cell-phone.

- As mentioned earlier Google's deep mind has already beaten doctors in detecting fatal diseases like breast cancer. It's not far away when AI will be detecting common disease as well as providing proper suggestions for medication. The consequences of this could be: no need for doctors in the long term result in JOB reduction.

2. AI in Education

- The development of a country depends on the quality of education youth is getting. Right now, we can see there are lots of courses are available on AI. But in the future AI is going to transform the classical way of education. Now the world doesn't need skilled labourers for manufacturing industries, which is mostly replaced by robots and automation. The education system could be quite effective and can be according to the individual's personality and ability. It would give chance brighter students to shine and to imbecile a better way to cop up.
- Right Education can enhance the power of individuals/nations; on the other hand, misuse of the same could lead to devastating results.

3. AI in Finance

- Quantification of growth for any country is directly related to its economic and financial condition. As AI has enormous scope in almost every field, it has great potential to boost individuals' economic health and a nation. Nowadays, the AI algorithm is being used in managing equity funds.
- An AI system could take a lot number of parameters while figuring out the best way to manage funds. It would perform better than a human manager. AI-driven strategies in the field of finance are going to change the classical way of trading and investing. It could be devastating for some fund managing firms who cannot afford such facilities and could affect business on a large scale, as the decision would be quick and abrupt. The competition would be tough and on edge all the time.

4. AI in Military and Cybersecurity

- AI-assisted Military technologies have built autonomous weapon systems, which won't need humans at all hence building the safest way to enhance the security of a nation. We could see robot Military in the near future, which is as intelligent as a

soldier/ commando and will be able to perform some tasks.

- AI-assisted strategies would enhance mission effectiveness and will provide the safest way to execute it. The concerning part with AI-assisted system is that how it performs algorithm is not quite explainable. The deep neural networks learn faster and continuously keep learning the main problem here would be explainable AI. It could possess devastating results when it reaches in the wrong hands or makes wrong decisions on its own.

2. Explain in detail about Intelligent Agent.

List the basic kinds of intelligent agents and explain any two agents with neat schematic diagram. [Nov 2023]

What is an Agent? How does it interact with environment? Explain. [Nov 2023]

Agents in Artificial Intelligence

- An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

- An agent can be anything that perceive its environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of perceiving, thinking, and acting.

An agent can be:

- **Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- **Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- **Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

Before moving forward, we should first know about sensors, effectors, and actuators.

- **Sensor:** Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.
- **Actuators**: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.
- **Effectors:** Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.

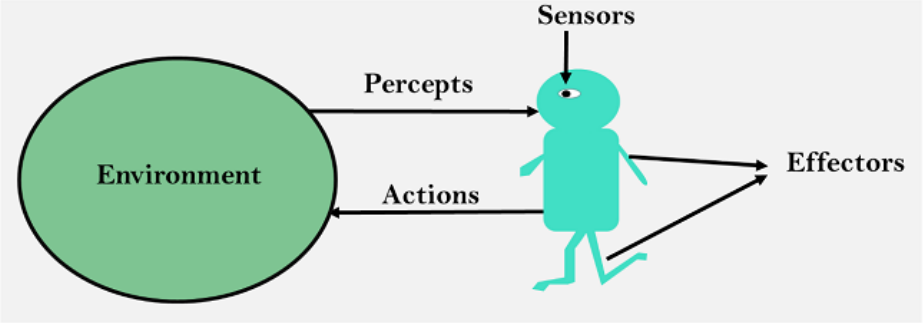


Fig.1.1 Agents

Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

Rule 1: An AI agent must have the ability to perceive the environment.

Rule 2: The observation must be used to make decisions.

Rule 3: Decision should result in an action.

Rule 4: The action taken by an AI agent must be a rational action.

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning

algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

3. Explain in detail about Rationality.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.
- The sequence of percepts.

Types of AI Agents or (Structure of Agents):

Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

- Simple Reflex Agent
- Model-based reflex agent
- Goal-based agents
- Utility-based agent
- Learning agent

1. Simple Reflex agent:

- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.

- Problems for the simple reflex agent design approach:
  - They have very limited intelligence
  - They do not have knowledge of non-perceptual parts of the current state
  - Mostly too big to generate and to store.
  - Not adaptive to changes in the environment

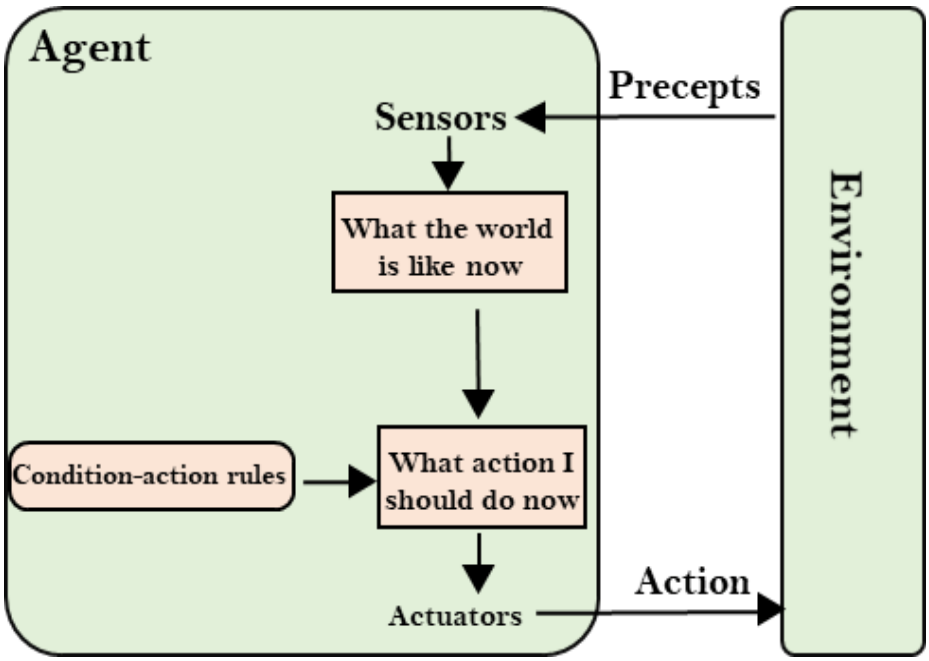


Fig.1.2 Simplex Reflex Agent

2. Model-based reflex agent

- The Model-based agent can work in a partially observable environment, and track the situation.
- A model-based agent has two important factors:
  - **Model:** It is knowledge about "how things happen in the world," so it is called a Model-based agent.
  - **Internal State:** It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
  a. How the world evolves

b. How the agent's action affects the world.

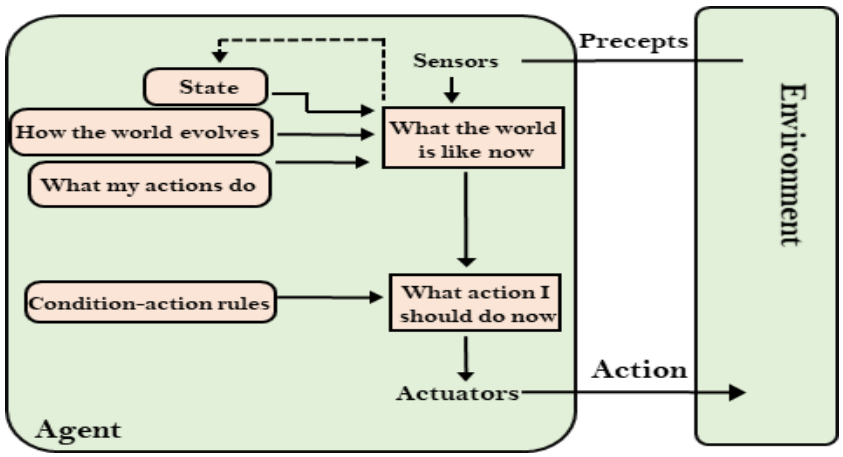


Fig.1.3 Model Based Reflex Agent

3. Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.
- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching and planning, which makes an agent proactive.

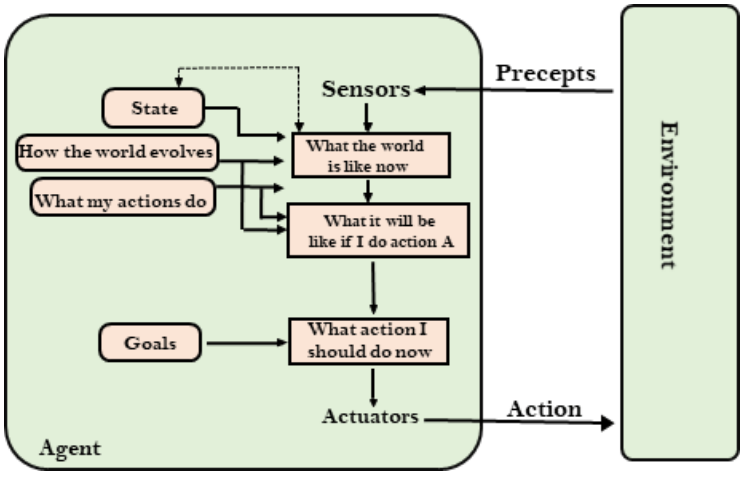


Fig.1.4 Goal Based Agent

4. Utility-based agents

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

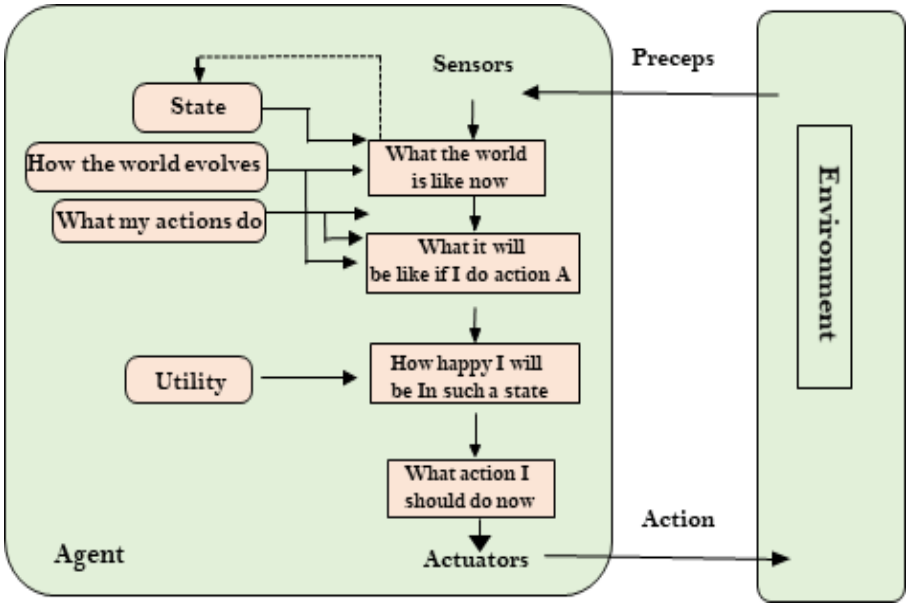


Fig.1.5 Utility Based Agent

5. Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:
    a. **Learning element:** It is responsible for making improvements by learning from environment

b. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.

c. **Performance element:** It is responsible for selecting external action

d. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.

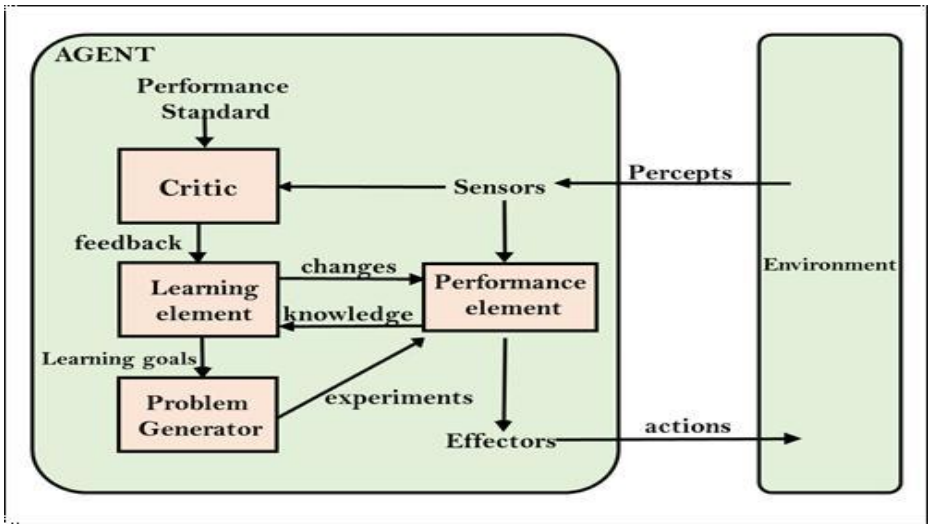


Fig.1.6 Learning Agent

4. Explain in detail about Nature of Environments.

Nature of Environments:

The environment is the **Task Environment (problem)** for which the **Rational Agent is the solution.** Any task environment is characterised on the basis of PEAS.

1. **Performance** – What is the performance characteristic which would either make the agent successful or not. For example, as per the previous example clean floor, optimal energy consumption might be performance measures.
2. **Environment** – Physical characteristics and constraints expected. For example, wood floors, furniture in the way etc
3. **Actuators** – The physical or logical constructs which would take action. For example for the vacuum cleaner, these are the suction pumps.
4. **Sensors** – Again physical or logical constructs which would sense the environment

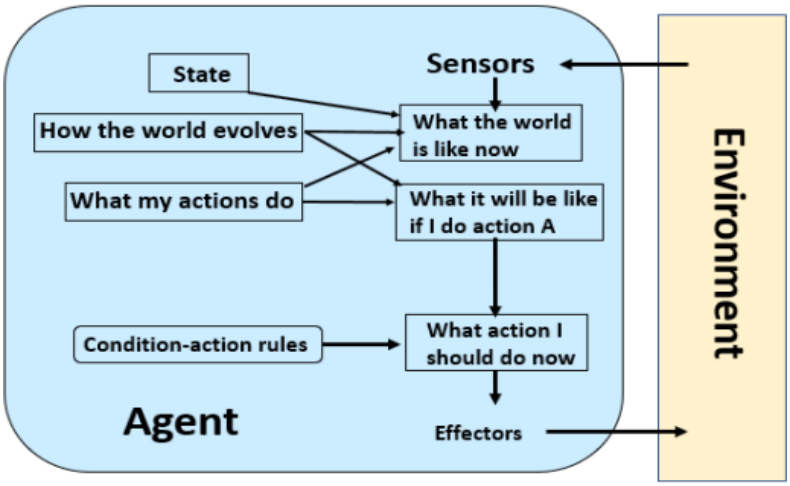


Fig. 1.7 Nature of Agents

- Rational Agents could be physical agents like the one described above or it could also be a program that operates in a non-physical environment like an operating system. Imagine a both web site operator designed to scan Internet news sources and show the interesting items to its users, while selling advertising space to generate revenue.
- As another example, consider an online tutoring system

Agent	Performance	Environment	Actuator	Sensor
Math E learning system	SLA defined score on the test	Student, Teacher, parents	Computer display system for exercises, corrections, feedback	Keyboard, Mouse

Environments can further be classified into various buckets. This would help determine the intelligence which would need to be built in the agent. These are

- **Observable** – Full or Partial? If the agents sensors get full access then they do not need to pre- store any information. Partial may be due to inaccuracy of sensors or incomplete information about an environment, like limited access to enemy territory
- **Number of Agents** – For the vacuum cleaner, it works in a single agent environment but for driver-less taxis, every driver-less taxi is a separate agent and hence multi agent environment
- **Deterministic** – The number of unknowns in the environment which affect the predictability of the environment. For example, floor space for cleaning is mostly deterministic, the furniture is where it is most of the time but taxi driving on a road is non-deterministic.
- **Discrete** – Does the agent respond when needed or does it have to continuously scan the environment. Driver-less is continuous, online tutor is discrete

- **Static** – How often does the environment change. Can the agent learn about the environment and always do the same thing?
- **Episodic** – If the response to a certain precept is not dependent on the previous one i.e. it is stateless (static methods in Java) then it is discrete. If the decision taken now influences the future decisions then it is a sequential environment.

Agents in Artificial Intelligence

An AI system can be defined as the study of the rational agent and its environment. The agents sense the environment through sensors and act on their environment through actuators. An AI agent can have mental properties such as knowledge, belief, intention, etc.

What is an Agent?

An agent can be anything that perceiveits environment through sensors and act upon that environment through actuators. An Agent runs in the cycle of **perceiving**, **thinking**, and **acting**. An agent can be:

- **Human-Agent:** A human agent has eyes, ears, and other organs which work for sensors and hand, legs, vocal tract work for actuators.
- **Robotic Agent:** A robotic agent can have cameras, infrared range finder, NLP for sensors and various motors for actuators.
- **Software Agent:** Software agent can have keystrokes, file contents as sensory input and act on those inputs and display output on the screen.

Hence the world around us is full of agents such as thermostat, cellphone, camera, and even we are also agents.

Before moving forward, we should first know about sensors, effectors, and actuators.

Sensor: Sensor is a device which detects the change in the environment and sends the information to other electronic devices. An agent observes its environment through sensors.

Actuators: Actuators are the component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system. An actuator can be an electric motor, gears, rails, etc.

Effectors: Effectors are the devices which affect the environment. Effectors can be legs, wheels, arms, fingers, wings, fins, and display screen.

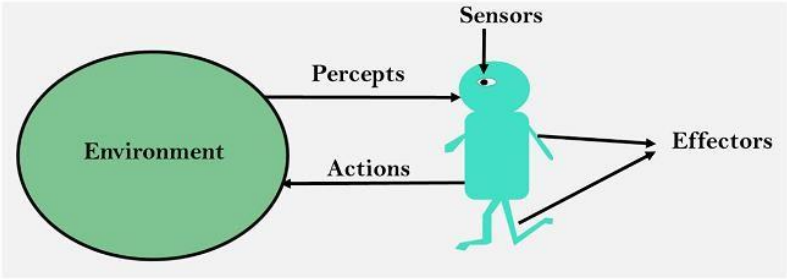


Fig.1.8 Agents

Intelligent Agents:

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- **Rule 1:** An AI agent must have the ability to perceive the environment.
- **Rule 2:** The observation must be used to make decisions.
- **Rule 3:** Decision should result in an action.
- **Rule 4:** The action taken by an AI agent must be a rational action.

Rational Agent:

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Note: Rational agents in AI are very similar to intelligent agents.

Rationality:

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.

- The sequence of percepts.

Note: Rationality differs from Omniscience because an Omniscient agent knows the actual outcome of its action and act accordingly, which is not possible in reality.

5. Explain in detail about Rationality.

Rational Agent in AI

- In artificial intelligence and machine learning, there's a concept called the "rational agent." It's a theoretical entity that considers realistic models of how people think, with preferences for advantageous outcomes and an ability to learn. In other words, it's what most people would call "you."
- The rational agent is used in game theory and decision theory to help us apply artificial intelligence to various real-world scenarios.
- We can use it to understand how we make decisions, allowing us to develop artificial intelligence that can mimic human behavior to solve problems or make decisions.

How Does a Rational Agent Work?

- A rational agent is a mathematical model that tries to represent the behavior of an intelligent being, like a person or animal. It uses a set of rules to determine the best course of action for a given situation.
- These agents usually work by comparing their current state with their previous state and then choosing an action based on how much better or worse they feel about their position now compared to before.
- For example, if you're hungry, you might want to eat something. If you're not hungry any longer, you might stop eating. A rational agent will do this repeatedly until it reaches some goal or decides it's time for bed (or both).
- The most basic form of this type of agent is called a reinforcement learning agent, which gets its name from the fact that it learns from experience as it goes along—it tries things out and then rates them based on how well they worked out in the past.

Examples of Rational Agents in AI

Some rational agents in AI include:

- Self-driving cars make decisions based on sensor data and optimize for safety and efficiency.
- Game-playing AI, such as AlphaGo, makes decisions based on the game's rules and the board's current state to maximize the chances of winning.
- Virtual personal assistants, such as Siri or Alexa, understand natural language commands and take appropriate actions based on the user's request.
- Stock trading algorithms make buy and sell decisions based on market data and predictions about future performance.
- Robotics, such as industrial robots, performs task based on programmed instructions and sensor inputs.

Rational Agent Real-World Applications

Rational agents are used in many real-world applications. Here are a few examples:

1. Autonomous systems: Self-driving cars, drones, and robots use rational agents to make decisions, plan their actions and optimize their behavior to achieve their goals, such as safely transporting passengers or completing a task.
2. Finance: Rational agents are used in financial services to make investment decisions, risk management, and trading. They can analyze market data, predict future trends, and optimize their behavior to maximize returns.
3. Healthcare: Rational agents make medical diagnoses, plan treatment, and monitor patients' progress. They can analyze medical data, predict the progression of diseases, and optimize the treatment plan.
4. Manufacturing: Rational agents are used in manufacturing to control production processes, plan logistics, and optimize the use of resources.
5. Transportation: Rational agents are used in transportation to plan routes, schedule vehicles, and optimize the use of resources.
6. Customer service: Rational agents interacting with customers, respond to their queries, and provide recommendations.
7. Social media: Rational agents are used to recommending content, filter spam, and moderate content.

Types of Agents

Agents are the computer programs we use to interact with the world. The idea of an agent is based on the idea that we can have computers act on our behalf, just like humans send other humans on their behalf.

Agents come in many forms, and you can group them into five classes based on their degree of perceived intelligence and capability:

- The simplest form of the agent is a reflex agent. Reflex agents simply react to stimuli without accurate understanding or memory of what has happened. They are the most basic form of AI and are used for simple tasks like controlling drones or autonomous cars.
- A model-based reflex agent is similar to a simple reflex agent but uses models to predict future states based on current state data. It allows it to learn from past experiences and make better decisions in the future.
- A goal-based agent uses logic to determine how best to achieve its goals, whether by moving forward or avoiding obstacles. Goal-based agents are used in applications like robot navigation systems or automated driving systems that need to plan their routes before taking action.
- Utility-based agents use utility functions (like rewards) as motivation towards achieving specific goals set by human users--this means that they have been programmed with particular behaviors.
- A learning agent is an agent that can learn from its experiences. It means that it can change its behavior based on previous experiences. It is not necessarily intelligent, but it does have a limited form of intelligence or the ability to learn from experience.

Intelligent Agent vs. Rational Agent

	Intelligent Agent	**Rational Agent**
Definition	An Intelligent Agent is a system that can perceive its environment and take actions to achieve a specific goal.	A Rational Agent is an Intelligent Agent that makes decisions based on logical reasoning and optimizes its behavior to achieve a specific goal.
Perception	An Intelligent Agent can perceive its environment through various sensors or inputs.	A Rational Agent's perception is based on the information available to it and logical reasoning.
Decision-making	It can make decisions based on a set of rules or a pre-defined algorithm.	It makes decisions based on logical reasoning and optimizes its behavior to achieve its goals.
Learning	An Intelligent Agent can learn from its environment and adapt its behavior.	A Rational Agent can also learn from its environment and adapt its behavior, but it does so based on logical reasoning.
Autonomy	It can operate independently of human intervention.	It can also operate independently of human intervention, but it does so based on logical reasoning.

Goals	An Intelligent Agent can be designed to achieve a specific goal.	A Rational Agent has a specific goal and optimizes its behavior.
Examples	An Intelligent Agent can be a self-driving car, a virtual personal assistant, or a recommendation system	A Rational Agent can be a financial advisor, a chess-playing program, or a logistics planner.

6. Explain in detail about Structure of an AI Agent.

Structure of an AI Agent

- The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

  Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on. **Agent Function:** Agent function is used to map a percept to an action f:P* → A

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- **P:** Performance measure
- **E:** Environment
- **A:** Actuators
- **S:** Sensors

Here performance measure is the objective for the success of an agent's behavior. PEAS for self-driving cars:

Let's suppose a self-driving car then PEAS representation will be:

Performance: Safety, time, legal drive, comfort **Environment:** Roads, other vehicles, road signs, pedestrian **Actuators:** Steering, accelerator, brake, signal, horn

Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

Example of Agents with their PEAS representation

Agent	Performance measure	Environment	Actuators	Sensors
1.Medical Diagnose	○ Healthy patient ○ Minimized cost	○ Patient ○ Hospital ○ Staff	○ Tests ○ Treatments	○ Keyboard ○ (Entry of symptoms)
2.Vacuum Cleaner	○ Cleanness ○ Efficiency ○ Battery life ○ Security	○ Room ○ Table ○ Wood floor ○ Carpet ○ Various obstacles	○ Wheels ○ Brushes ○ Vacuum Extractor	○ Camera ○ Dirt detection sensor ○ Cliff sensor ○ Bump Sensor ○ Infrared Wall Sensor
3. Part - picking Robot	○ Percentage of parts in correct bins.	○ Conveyor belt with parts, ○ Bins	○ Jointed Arms ○ Hand	○ Camera ○ Joint angle sensors.

Agent Environment in AI:

- An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent

is present.

- The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

Features of Environment

Environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable:

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is **a fully observable** environment, else it is **partially observable**.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

4. Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

5. Static vs Dynamic:

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
- Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- However for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

7. Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.

- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
- It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

8. Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.
- Information about an event on earth is an example of Inaccessible environment.

7. Explain in detail about Search Algorithms in Artificial Intelligence.

Search Algorithms in Artificial Intelligence:

- Search algorithms are one of the most important areas of Artificial Intelligence. Problem-solving agents:
- In Artificial Intelligence, Search techniques are universal problem-solving methods. **Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation. In this topic, we will learn various problem-solving search algorithms.

Search Algorithm Terminologies:

- **Search:** Searchingis a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
  a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
  b. **Start State:** It is a state from where agent begins **the search**.
  c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search tree: A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

Actions: It gives the description of all the available actions to the agent.

Transition model: A description of what each action do, can be represented as a transition model.

Path Cost: It is a function which assigns a numeric cost to each path.

Solution: It is an action sequence which leads from the start node to the goal node.

Optimal Solution: If a solution has the lowest cost among all solutions. Properties of Search Algorithms:

Following are the four essential properties of search algorithms to compare the efficiency of these algorithms:

Completeness: A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.

Optimality: If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.

Time Complexity: Time complexity is a measure of time for an algorithm to complete its task.

Space Complexity: It is the maximum storage space required at any point during the search, as the complexity of the problem.

Types of search algorithms

Based on the search problems we can classify the search algorithms into uninformed (Blind search) search and informed search (Heuristic search) algorithms.

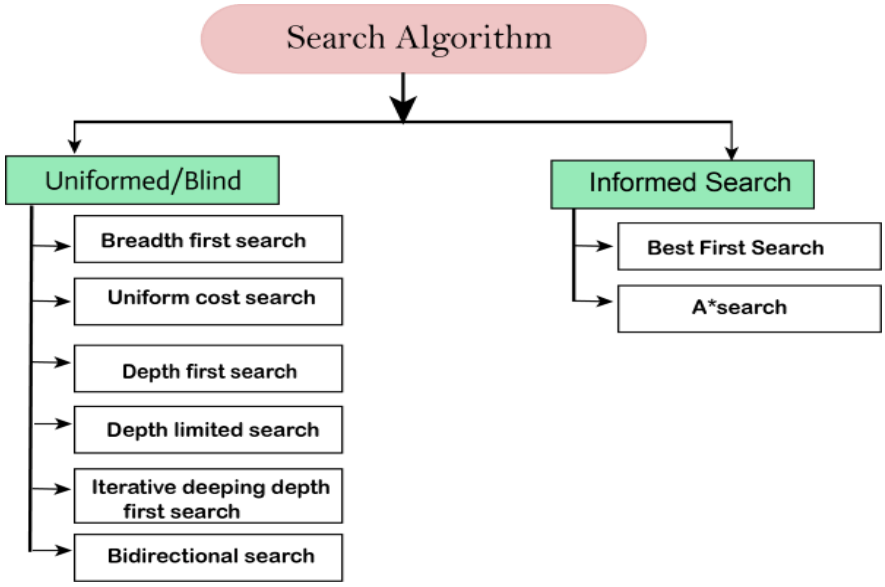


Fig. 1.9 Types of Search Algorithms

8. Explain in detail about Uninformed Search Algorithms.

Uninformed Search Algorithms:

Uninformed search is a class of general-purpose search algorithms which operates in brute force- way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. **Breadth-first Search**
2. **Depth-first Search**
3. **Depth-limited Search**
4. **Iterative deepening depth-first search**
5. **Uniform cost search**
6. **Bidirectional Search**

1. **Breadth-first Search:**

- Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.
- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.

- Breadth-first search implemented using FIFO queue data structure.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

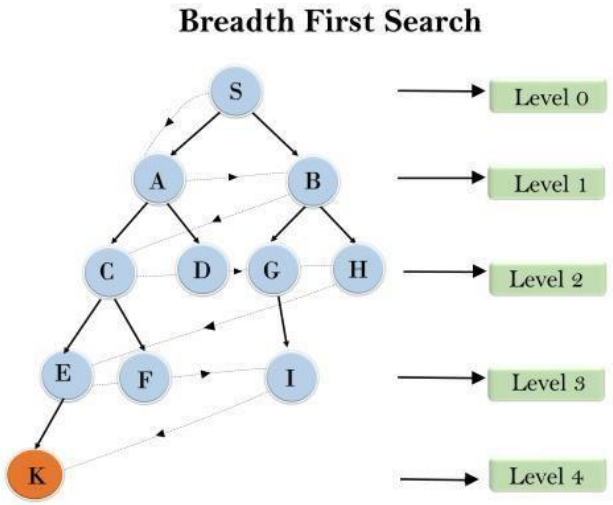
Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I >K



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

$T(b) = 1+b^2+b^3+\ldots. + b^d = O(b^d)$

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of

frontier which is $O(b^d)$.

Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth-first Search

- Depth-first search isa recursive algorithm for traversing a tree or graph data structure.
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation.
- The process of the DFS algorithm is similar to the BFS algorithm.

Note: Backtracking is an algorithm technique for finding all possible solutions using recursion.

Advantage:

- DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.
- It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage:

- There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.
- DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

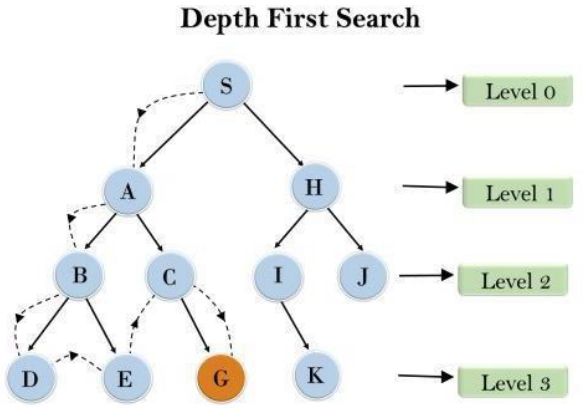
Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node-- > right node.

It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found.

After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.



Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$T(n)= 1+ n^2+ n^3 + ...+ n^m=O(n^m)$

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is **O(bm)**.

Optimal: DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

3. Depth-Limited Search Algorithm:

A depth-limited search algorithm is similar to depth-first search with a predetermined limit. Depth-limited search can solve the drawback of the infinite path in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

Depth-limited search can be terminated with two Conditions of failure:

- Standard failure value: It indicates that problem does not have any solution.
- Cutoff failure value: It defines no solution for the problem within a given depth limit.

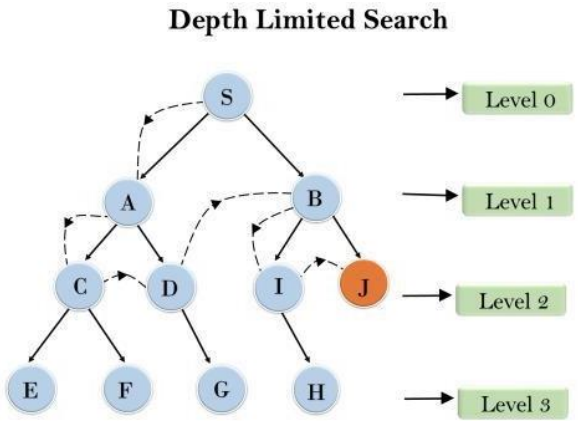
Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

- Depth-limited search also has a disadvantage of incompleteness.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is **O(b$^{\ell}$)**. **Space Complexity:** Space complexity of DLS algorithm is O**(b×ℓ)**.

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

4. Uniform-cost Search Algorithm:

- Uniform-cost search is a searching algorithm used for traversing a weighted tree or graph. This algorithm comes into play when a different cost is available for each edge. The primary goal of the uniform-cost search is to find a path to the goal node which has the lowest cumulative cost.
- Uniform-cost search expands nodes according to their path costs form the root node. It can be used to solve any graph/tree where the optimal cost is in demand. A uniform-cost search algorithm is implemented by the priority queue. It gives maximum priority to the lowest cumulative cost. Uniform cost search is equivalent to BFS algorithm if the path cost of all edges is the same.

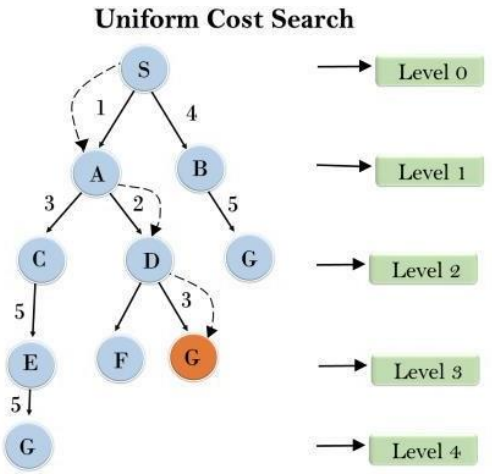
Advantages:

- Uniform cost search is optimal because at every state the path with the least cost is chosen.

Disadvantages:

- It does not care about the number of steps involve in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

Example:



Completeness:

Uniform-cost search is complete, such as if there is a solution, UCS will find it.

Time Complexity:

Let C* **is Cost of the optimal solution**, and **ε** is each step to get closer to the goal node. Then the number of steps is = C*/ε+1. Here we have taken +1, as we start from state 0 and end to C*/ε.

Hence, the worst-case time complexity of Uniform-cost search is$\mathbf{O(b^{1 + [C^*/\varepsilon]})/}$.

Space Complexity:

The same logic is for space complexity so, the worst-case space complexity of Uniform-cost search is $\mathbf{O(b^{1 + [C^*/\varepsilon]})}$.

Optimal:

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

5. Iterative deepening depth-first Search:

The iterative deepening algorithm is a combination of DFS and BFS algorithms. This search algorithm finds out the best depth limit and does it by gradually increasing the limit until a goal is found.

This algorithm performs depth-first search up to a certain "depth limit", and it keeps increasing the depth limit after each iteration until the goal node is found.

This Search algorithm combines the benefits of Breadth-first search's fast search and depth-first search's memory efficiency.

The iterative search algorithm is useful uninformed search when search space is large, and depth of goal node is unknown.

Advantages:

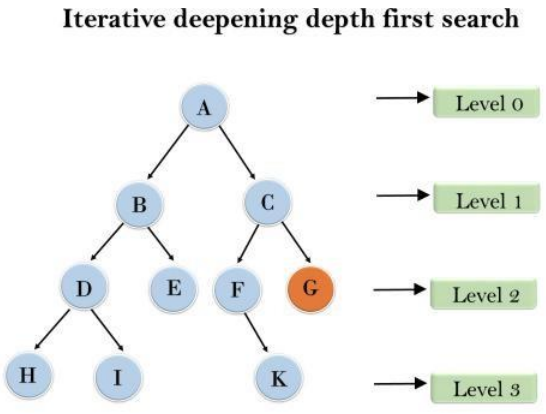
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:



1'st Iteration --> A

2'nd Iteration -> A, B, C

3'rd Iteration --- >A, B, D, E, C, F, G

4'th Iteration --- >A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Completeness:

This algorithm is complete is ifthe branching factor is finite.

Time Complexity:

Let's suppose b is the branching factor and depth is d then the worst-case time complexity is **$O(b^d)$**.

Space Complexity:

The space complexity of IDDFS will be **O(bd)**. **Optimal:**

IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.

6. Bidirectional Search Algorithm:

Bidirectional search algorithm runs two simultaneous searches, one form initial state called as forward-search and other from goal node called as backward-search, to find the goal node. Bidirectional search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex and other starts from goal vertex. The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Advantages:

- Bidirectional search is fast.
- Bidirectional search requires less memory

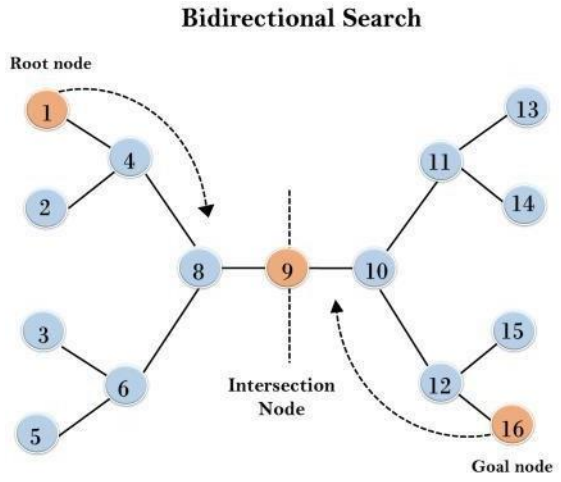
Disadvantages:

- Implementation of the bidirectional search tree is difficult.
- **In bidirectional search, one should know the goal state in advance.**

Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction.

The algorithm terminates at node 9 where two searches meet.



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is **$O(b^d)$**.

Space Complexity: Space complexity of bidirectional search is **$O(b^d)$**.

Optimal: Bidirectional search is Optimal.

9. Explain in detail about Informed Search Algorithms.

What are informed search techniques? Explain any one. Nov 2023

Informed Search Algorithms:

- So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.

Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

h(n) <= h*(n)

Pure Heuristic Search:

- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.
- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

1. Best First Search Algorithm(Greedy search)
2. A* Search Algorithm

1.) Best-first Search Algorithm (Greedy Search):

- Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

  f(n)= g(n).

- Were, h(n)= estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.

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.

Advantages:

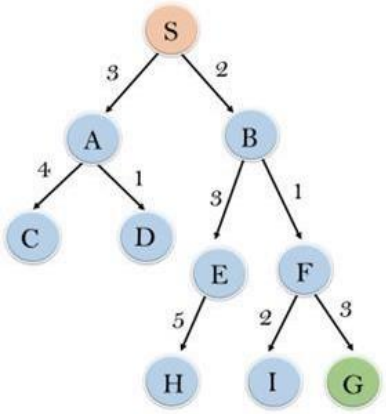
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

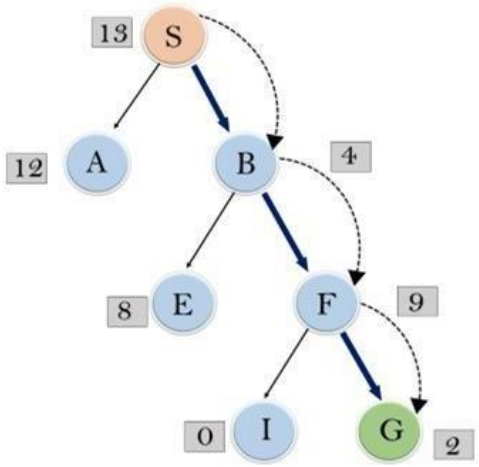
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



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

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]: Open [I, E, A], Closed [S, B, F, G]

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

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

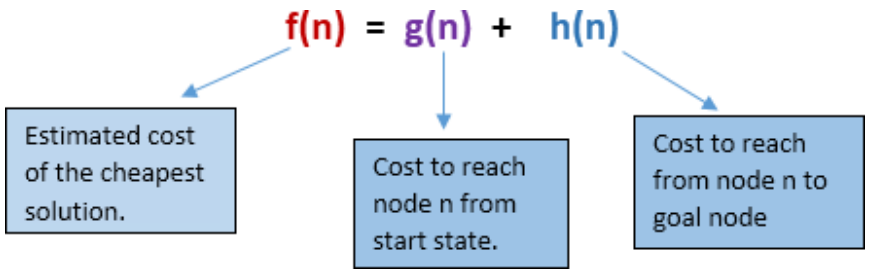
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

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

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



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**. Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.

- This algorithm can solve very complex problems.

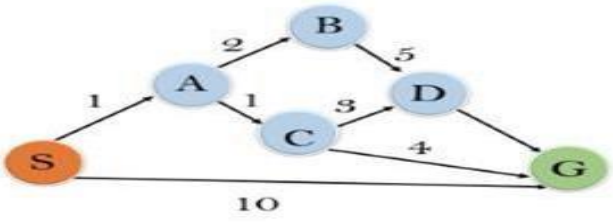
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

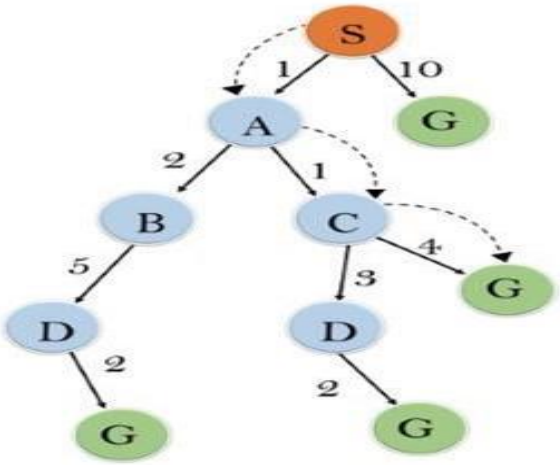
In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



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

Solution:



Initialization: {(S, 5)}

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

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

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

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

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

10. List some environmental properties that are important for intelligent agents.

[Apr 2024]

- Common characteristics of intelligent agents are adaptation based on experience, real-time problem-solving, analysis of error or success rates, and the use of memory-based storage and retrieval.

Features of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

1. Fully observable vs Partially Observable
2. Static vs Dynamic
3. Discrete vs Continuous
4. Deterministic vs Stochastic
5. Single-agent vs Multi-agent
6. Episodic vs sequential
7. Known vs Unknown
8. Accessible vs Inaccessible

1. Fully observable vs Partially Observable

- If an agent sensor can sense or access the complete state of an environment at each point in time then it is a fully observable environment, it is partially observable. For reference, Imagine a chess-playing agent. In this case, the agent can fully observe the state of the chessboard at all times. Its sensors (in this case, vision or the ability to access the board's state) provide complete information about the current position of all pieces. This is a fully observable environment because the agent has perfect information about the state of the world.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track of the history of the world. For reference, Consider a self-driving car navigating a busy city. While the car has sensors like cameras, lidar, and radar, it can't see everything at all times. Buildings, other vehicles, and pedestrians can obstruct its sensors. In this scenario, the car's environment is partially observable because it doesn't have complete and constant access to all relevant information. It needs to maintain an internal state and history to make informed decisions even when some information is temporarily unavailable.
- An agent with no sensors in all environments then such an environment is called unobservable. For reference, think about an agent designed to predict earthquakes but placed in a sealed, windowless room with no sensors or access to external data. In this situation, the environment is unobservable because the agent has no way to gather information about the outside world. It can't sense any aspect of its environment, making it completely unobservable.

2. Deterministic vs Stochastic:

- If an agent's current state and selected action can completely determine the next state of the environment, then such an environment is called a deterministic environment. For reference, Chess is a classic example of a deterministic environment. In chess, the rules are well-defined, and each move made by a player has a clear and predictable outcome based on those rules. If you move a pawn from one square to another, the resulting state of the chessboard is entirely determined by that action, as is your opponent's response. There's no randomness or uncertainty in the outcomes of chess moves because they follow strict rules. In a deterministic environment like chess, knowing the current state and the actions taken allows you to completely determine the next state.
- A stochastic environment is random and cannot be determined completely by an agent. For reference, The stock market is an example of a stochastic environment. It's highly influenced by a multitude of unpredictable factors, including economic events, investor sentiment, and news. While there are patterns and trends, the exact behavior of stock prices is inherently random and cannot be completely determined by any individual or agent. Even with access to extensive data and analysis tools, stock market movements can exhibit a high degree of unpredictability. Random events and market sentiment play significant roles, introducing uncertainty.
- In a deterministic, fully observable environment, an agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action. For example, Tic-Tac-Toe is a classic example of an episodic environment. In this game, two players take turns placing their symbols (X or O) on a 3x3 grid. Each move by a player is independent of previous moves, and the goal is to form a line of three symbols horizontally, vertically, or diagonally. The game consists of a series of one-shot actions where the current state of the board is the only thing that matters for the next move. There's no need for the players to remember past moves because they don't affect the current move. The game is self-contained and episodic.
- However, in a Sequential environment, an agent requires memory of past actions to determine the next best actions. For example, Chess is an example of a sequential environment. Unlike Tic-Tac-Toe, chess is a complex game where the outcome of each move depends on a sequence of previous moves. In chess, players must consider the history of the game, as the current position of pieces, previous moves, and potential future moves all influence the best course of action. To play chess effectively, players need to maintain a

memory of past actions, anticipate future moves, and plan their strategies accordingly. It's a sequential environment because the sequence of actions and the history of the game significantly impact decision-making.

4. Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called a single-agent environment. For example, Solitaire is a classic example of a single-agent environment. When you play Solitaire, you're the only agent involved. You make all the decisions and actions to achieve a goal, which is to arrange a deck of cards in a specific way. There are no other agents or players interacting with you. It's a solitary game where the outcome depends solely on your decisions and moves. In this single-agent environment, the agent doesn't need to consider the actions or decisions of other entities.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment. For reference, A soccer match is an example of a multi-agent environment. In a soccer game, there are two teams, each consisting of multiple players (agents). These players work together to achieve common goals (scoring goals and preventing the opposing team from scoring). Each player has their own set of actions and decisions, and they interact with both their teammates and the opposing team. The outcome of the game depends on the coordinated actions and strategies of all the agents on the field. It's a multi-agent environment because there are multiple autonomous entities (players) interacting in a shared environment.
- The agent design problems in the multi-agent environment are different from single-agent environments.

5. Static vs Dynamic

- If the environment can change itself while an agent is deliberating then such an environment is called a dynamic environment it is called a static environment.
- Static environments are easy to deal with because an agent does not need to continue looking at the world while deciding on an action. For reference, A crossword puzzle is an example of a static environment. When you work on a crossword puzzle, the puzzle itself doesn't change while you're thinking about your next move. The arrangement of clues and empty squares remains constant throughout your problem-solving process. You can take your time to deliberate and find the best word to fill in each blank, and the puzzle's state remains unaltered during this process. It's a static environment because there are no

changes in the puzzle based on your deliberations.

- However, for a dynamic environment, agents need to keep looking at the world at each action. For reference, Taxi driving is an example of a dynamic environment. When you're driving a taxi, the environment is constantly changing. The road conditions, traffic, pedestrians, and other vehicles all contribute to the dynamic nature of this environment. As a taxi driver, you need to keep a constant watch on the road and adapt your actions in real time based on the changing circumstances. The environment can change rapidly, requiring your continuous attention and decision-making. It's a dynamic environment because it evolves while you're deliberating and taking action.

6. Discrete vs Continuous

- If in an environment, there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment it is called a continuous environment.
- Chess is an example of a discrete environment. In chess, there are a finite number of distinct chess pieces (e.g., pawns, rooks, knights) and a finite number of squares on the chessboard. The rules of chess define clear, discrete moves that a player can make. Each piece can be in a specific location on the board, and players take turns making individual, well-defined moves. The state of the chessboard is discrete and can be described by the positions of the pieces on the board.
- Controlling a robotic arm to perform precise movements in a factory setting is an example of a continuous environment. In this context, the robot arm's position and orientation can exist along a continuous spectrum. There are virtually infinite possible positions and orientations for the robotic arm within its workspace. The control inputs to move the arm, such as adjusting joint angles or applying forces, can also vary continuously. Agents in this environment must operate within a continuous state and action space, and they need to make precise, continuous adjustments to achieve their goals.

7. Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results of all actions are known to the agent. While in an unknown environment, an agent needs to learn how it works in order to perform an action.
- It is quite possible for a known environment to be partially observable and an Unknown environment to be fully observable.

- The opening theory in chess can be considered as a known environment for experienced chess players. Chess has a vast body of knowledge regarding opening moves, strategies, and responses. Experienced players are familiar with established openings, and they have studied various sequences of moves and their outcomes. When they make their initial moves in a game, they have a good understanding of the potential consequences based on their knowledge of known openings.
- Imagine a scenario where a rover or drone is sent to explore an alien planet with no prior knowledge or maps of the terrain. In this unknown environment, the agent (rover or drone) has to explore and learn about the terrain as it goes along. It doesn't have prior knowledge of the landscape, potential hazards, or valuable resources. The agent needs to use sensors and data it collects during exploration to build a map and understand how the terrain works. It operates in an unknown environment because the results and consequences of its actions are not initially known, and it must learn from its experiences.

8. Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- For example, Imagine an empty room equipped with highly accurate temperature sensors. These sensors can provide real-time temperature measurements at any point within the room. An agent placed in this room can obtain complete and accurate information about the temperature at different locations. It can access this information at any time, allowing it to make decisions based on the precise temperature data. This environment is accessible because the agent can acquire complete and accurate information about the state of the room, specifically its temperature.
- For example, Consider a scenario where a satellite in space is tasked with monitoring a specific event taking place on Earth, such as a natural disaster or a remote area's condition. While the satellite can capture images and data from space, it cannot access fine-grained information about the event's details. For example, it may see a forest fire occurring but cannot determine the exact temperature at specific locations within the fire or identify individual objects on the ground. The satellite's observations provide valuable data, but the environment it is monitoring (Earth) is vast and complex, making it impossible to access complete and detailed information about all aspects of the event. In this case, the Earth's surface is an inaccessible environment for obtaining fine-grained information about specific events.

11. Describe the properties of the environment for the Mars Rover. [Apr 2024]

- rover: a vehicle for exploring the surface of a planet or moon
- lander: a type of spacecraft that touches down on a planet or moon; a lander does not move around
- Mars is very cold. The average temperature on Mars is minus 80 degrees Fahrenheit — way below freezing! Its surface is rocky, with canyons, volcanoes, dry lake beds and craters all over it. Red dust covers most of its surface.
- Mars has clouds and wind just like Earth. Sometimes the wind blows the red dust into a dust storm. Tiny dust storms can look like tornados, and large ones can be seen from Earth. Mars' large storms sometimes cover the entire planet.
- Mars has about one-third the gravity of Earth. A rock dropped on Mars would fall more slowly than a rock falls on Earth. A person who weighs 100 pounds on Earth would only weigh about 38 pounds on Mars because of the reduced gravity.
- The atmosphere of Mars is much thinner than Earth's. The Red Planet's atmosphere contains more than 95% carbon dioxide and much less than 1% oxygen. People would not be able to breathe the air on Mars.
- Mars is the fourth planet from the Sun and the next planet beyond Earth. It is, on average, more than 142 million miles from the Sun. Mars turns on its axis more slowly than Earth does. So, a day on Mars is 24.6 hours. Since this planet is farther from the Sun than Earth, one revolution of Mars around the Sun is a longer trip. So, a year on Mars is 687 Earth days. Mars is about half the size of Earth. Mars is known as the Red Planet because the iron oxide chemicals in its soil looks like rust.
- Mars is named for the ancient Roman god of war. The Greeks called the planet Ares (pronounced Air-EEZ). The Romans and Greeks associated the planet with war because its color resembles the color of blood.
- Mars has two small moons. Their names are Phobos (FOE-bohs) and Deimos (DEE-mohs). They are named for the sons of Ares, the Greek god of war. Phobos means "fear," and Deimos means "panic."

12. Consider the search graph shown in fig. S is the start state and G is the goal state. All edges

are bidirectional. For each of the following search strategies, give the path that would be returned and the pesudocode for the same. [Apr 2024]

(i) Depth first graph search

(ii) Breadth first graph search

(iii) Uniform cost graph search

To solve this problem, we need to analyze the search graph and apply each search strategy to determine the path that would be returned. Since the graph is not provided, I will assume a generic graph structure and apply the search strategies based on typical behaviors of each algorithm.

Step 1: Define the Graph Structure

Let's assume the graph has the following structure:

S connects to A and B.

A connects to C and D.

B connects to D and E.

C connects to G.

D connects to G.

E connects to G.

Step 2: Apply Depth-first Graph Search

Depth-first search (DFS) explores as far as possible along each branch before retracing steps. Starting from S, DFS would go to A or B, then deeper into the graph. Assuming it goes to A first, it would reach G via C.

Step 3: Apply Breadth-first Graph Search

Breadth-first search (BFS) explores all the neighboring nodes at the present depth before moving on to nodes at the next depth level. Starting from S, BFS would visit A and B, then C, D, and E, and finally G.

Step 4: Apply Uniform Cost Graph Search

Uniform cost search (UCS) explores nodes based on the path cost from the start node. Assuming all edges have equal cost, the path would be similar to BFS.

Step 5: Apply Greedy Graph Search

Greedy search chooses the node that appears to be closest to the goal. Starting from S, it would choose A or B based on which looks closer to G. Assuming it chooses A, it would reach G via D or

C.

Step 6: Apply A Graph Search*

A* combines features of UCS and greedy best-first search, using a heuristic to estimate the cost to reach the goal from a given node. Assuming a heuristic that prefers nodes closer to G, it would likely choose a path similar to greedy search but optimized based on actual path costs.

Final Answer

(a) Path: S -> A -> C -> G

(b) Path: S -> A -> B -> C -> D -> E -> G

(c) Path: S -> A -> C -> G (assuming equal edge costs)

(d) Path: S -> A -> C -> G (assuming A looks closer)

(e) Path: S -> A -> C -> G (assuming a heuristic that prefers closer nodes)

MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by Naac with A Grade & TCS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

II YEAR / III SEM

AL3391 ARTIFICIAL INTELLIGENCE

UNIT II

Heuristic search strategies – heuristic functions. Local search and optimization problems – local search in continuous space – search with non-deterministic actions – search in partially observable environments – online search agents and unknown environments.

PART-A

1. Define informed (heuristic) search strategies and mention its types.

- Here, the algorithms have information on the goal state, which helps in more efficient searching.
- This information is obtained by something called a heuristic.

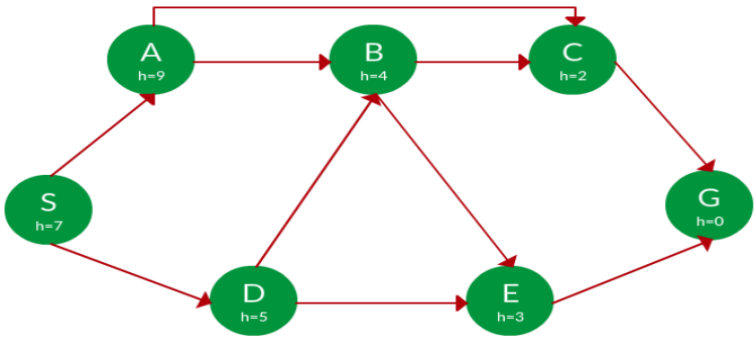
Search Heuristics:

- In an informed search, a heuristic is a function h(n) estimates how close a state is to the goal state.
- h(n) = estimated cost of the cheapest path from the state at node n to a goal state.

Types of Informed search algorithms.

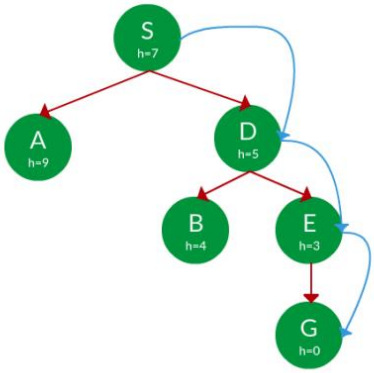
- Greedy Search
- A* Tree Search
- A* Graph Search

2. Find the path from S to G using greedy search. The heuristic values h of each node below the name of the node.



Solution

Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with a lower heuristic cost. Finally, from E, we go to G(h=0). This entire traversal is shown in the search tree below, in blue.

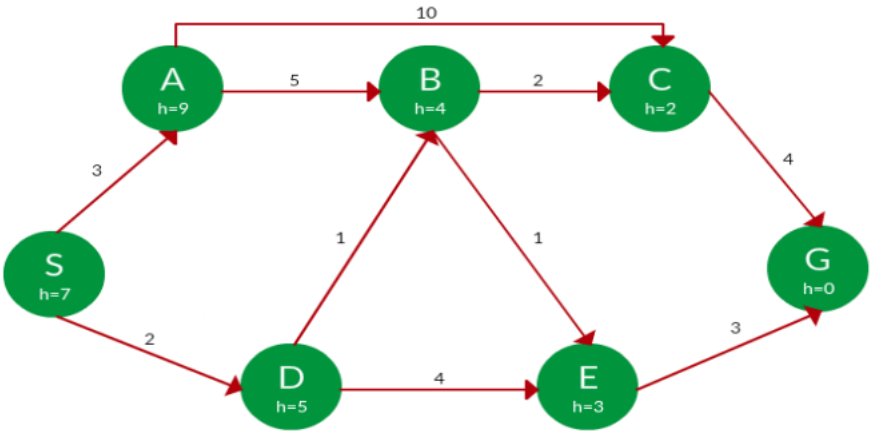


Path: S -> D -> E -> G

3. Define A* tree search.

- A* Tree Search, or simply known as A* Search, combines the strengths of uniform-cost search and greedy search.
- In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in the greedy search, denoted by h(x). The summed cost is denoted by f(x).

4. Find the path to reach from S to G using A* search.



Solution

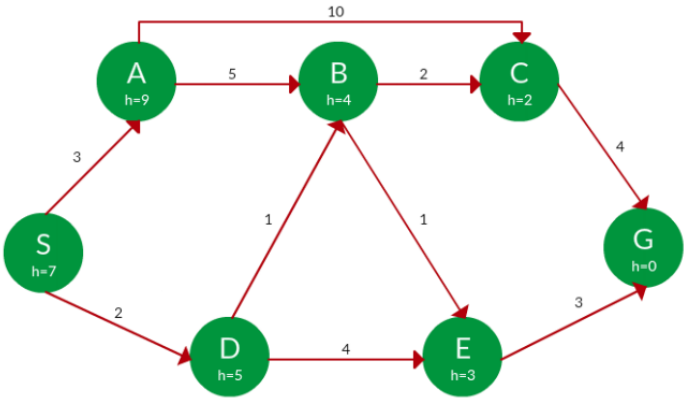
- Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum.
- The entire work is shown in the table below.

Path	h(x)	g(x)	f(x)
S	7	0	7
S -> A	9	3	12
S -> D ✓	5	2	7
S -> D -> B ✓	4	2 + 1 = 3	7
S -> D -> E	3	2 + 4 = 6	9
S -> D -> B -> C ✓	2	3 + 2 = 5	7
S -> D -> B -> E ✓	3	3 + 1 = 4	7
S -> D -> B -> C -> G	0	5 + 4 = 9	9
S -> D -> B -> E -> G ✓	0	4 + 3 = 7	7

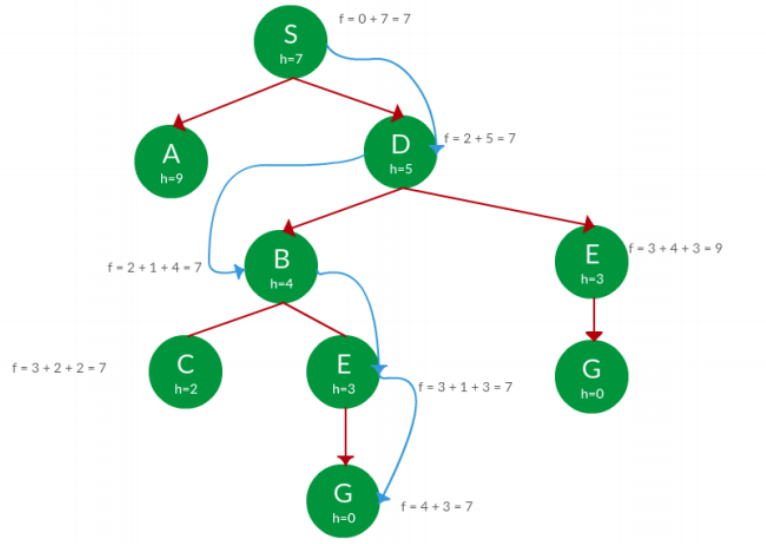
Path: S -> D -> B -> E -> G

Cost: 7

5. Use graph searches to find paths from S to G in the following graph.



Solution



Path: S -> D -> B -> E -> G
Cost: 7

6. Define Local Search Algorithm. List its types.

Local Search Algorithm - Definition

- **Local search** algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, or the set of states that have been reached.
- **"Local search algorithms" where the path cost does not matters, and only focus on solution-state needed to reach the goal node.**

It has two key advantages:

- use very little memory;
- Often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

Different types of local searches:

- Hill-climbing Search

Types of Hill climbing search algorithm

- Simple hill climbing
- Steepest-ascent hill climbing
- Stochastic hill climbing
- Random-restart hill climbing
- Simulated Annealing
- Local Beam Search

7. What are the Limitations of Hill climbing algorithm.

- **Local Maxima:** It is that peak of the mountain which is highest than all its neighboring states but lower than the global maxima. It is not the goal peak because there is another peak higher than it.
- **Plateau:** It is a flat surface area where no uphill exists. It becomes difficult for the climber to decide that in which direction he should move to reach the goal point.
- **Ridges:** It is a challenging problem where the person finds two or more local maxima of the same height commonly. It becomes difficult for the person to navigate the right point and stuck to that point itself.

8. How will you measure the problem-solving performance?

Problem solving performance is measured with 4 factors.

1. Completeness - Does the algorithm (solving procedure) surely finds solution
2. If really the solution exists.

   Optimality – If multiple solutions exits then do the algorithm returns optimal amongst them.
3. Time requirement.
4. Space requirement.

9. Mention how the search strategies are evaluated?

Search strategies are evaluated on following four criteria

- Completeness
- Time complexity
- Space complexity
- Optimality

10. What are the types of Informed search algorithms.

- Greedy Search (best-first search)
- A* Tree Search
- A* Graph Search

11. Define GREEDY SEARCH - (BEST-FIRST SEARCH).

- In greedy search, we expand the node closest to the goal node. The closeness is estimated by a heuristic h(x).
- **Heuristic:** A heuristic h is defined as-

  h(x) = Estimate of distance of node x from the goal node. Lower the value of h(x), closer is the node from the goal.
- **Strategy:** Expand the node closest to the goal state, i.e., expand the node with a lower h value.

12. Write the Working of a Local search algorithm.

Consider the below state-space landscape having both:

- **Location:** It is defined by the state.
- **Elevation:** It is defined by the value of the objective function or heuristic cost function.

13. Write simple Simple hill climbing Algorithm.

1. Create a CURRENT node, NEIGHBOUR node, and a GOAL node.
2. If the CURRENT node=GOAL node, return GOAL and terminate the search.
3. Else CURRENT node<= NEIGHBOUR node, move ahead.
4. Loop until the goal is not reached or a point is not found.

14. Write down the Algorithm of A*.

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n)  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

15. Define Genetic Algorithms.

- Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics.
- GAs are a subset of a much larger branch of computation Known as Evolutionary Computation.
- Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics.
- GAs are a subset of a much larger branch of computation known as Evolutionary Computation.

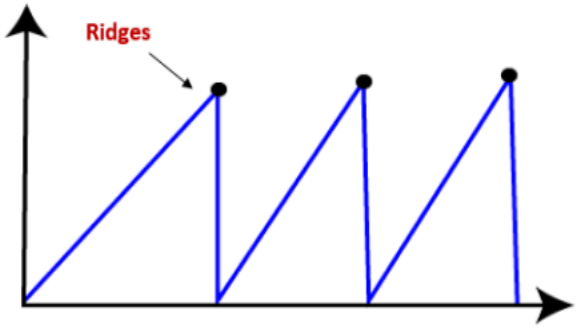
17. How can we avoid ridge and plateau in hill climbing. (NOV 2012)

It can be avoided by means of random-restart hill climbing will find a good solution very quickly

1. **Ridges:** Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
2. **Plateau:** A plateau is an area of the state space landscape where the evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which it is possible to make progress.

18. What is Ridges? (APR/MAY 2016)

- A "ridges" which is an area in the search that is higher than the surrounding areas, but cannot be searched in a simple move.



19. What are the 2 types of memory bounded heuristic algorithms?

- Recursive Best First Search (RBFS)
- Memory bounded A*(MA*)

20. Define Global minimum.

- If elevation corresponds to cost, then the aim is to find the lowest valley is called global minimum.

21. Define Global Maximum.

- If elevation corresponds to an objective function, then the aim is to find the highest peak is called global maximum.

22. Define Local maxima.

- A local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum**.**

23. Define Online Search agent.

- Agent operates by interleaving computation and action (i.e.) first it takes an action, and then it observes the environment and computes the next action.

24. What are the things that agent knows in online search problems?

a. Actions(s)

b. Step cost function C(s, a, s')

c. Goal TEST(s)

25. Difference between Uninformed search (Blind Search) and Informed Search.

S.NO	Uninformed search or Blind Search	Informed or Heuristic Search
1	No additional information beyond that provided in the problem definition.	Uses problem-specific knowledge beyond the definition of the problem itself.
2	Not effective.	More effective.
3	No information about number of Steps or path cost.	Information is provided.
4	The search should proceed in a Systematic way by exploring in Some predefined order by simply selecting the nodes at random. Eg. DFS, BFS, Bidirectional search.	The search space is thus Constrained making search efficient. They use and depend upon heuristic information.

26. What is Admissible Heuristics? [Apr 2024]

- An admissible heuristic is one that never overestimates the cost to reach the goal, making it a valuable tool in optimizing the efficiency of AI processes.

27. Give two examples of partially observable environments. [Apr 2024]

- An example of a partially observable system would be a card game in which some of the cards are discarded into a pile face down. In this case the observer is only able to view their own cards and potentially those of the dealer.

28. What is Heuristic Function? [Nov 2023]

- A heuristic function, also simply called a heuristic, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution.

29. What are the things that agent knows in online search problems? [Nov 2023]

- Online search is a necessary idea for an exploration problem where the states and actions are unknown to the agent. An online search problem can be solved by an agent executing actions, so these functions are necessary.

PART B

1. Explain in detail about Heuristic Search Strategies.

What is Heuristic search technique in AI? How does heuristics search works? Explain its advantages and disadvantages. [Nov 2023]

Heuristic Search Strategies:

Heuristics

- A heuristic is a technique that is used to solve a problem faster than the classic methods. These techniques are used to find the approximate solution of a problem when classical methods do not.
- Heuristics are said to be the problem-solving techniques that result in practical and quick solutions. **Optimal,** but those solutions are sufficient in a given limited timeframe.

Need of Heuristics

- Heuristics are used in situations in which there is the requirement of a short-term solution. On facing complex situations with limited resources and time, Heuristics can help the companies to make quick decisions by shortcuts and approximated calculations. Most of the heuristic methods involve mental shortcuts to make decisions on past experiences.
- The heuristic method might not always provide us the finest solution, but it is assured that it helps us find a good solution in a reasonable time.
- Based on context, there can be different heuristic methods that correlate with the problem's scope. The most common heuristic methods are - trial and error, guesswork, the process of elimination, historical data analysis. These methods involve simply available information that is not particular to the problem but is most appropriate. They can include representative, affect, and availability heuristics.

Heuristic search techniques in AI (Artificial Intelligence)

Fig. 2.1 Heuristic Search in AI

We can perform the Heuristic techniques into two categories:

Direct Heuristic Search techniques in AI

- It includes Blind Search, Uninformed Search, and Blind control strategy. These search techniques are not always possible as they require much memory and time. These techniques search the complete space for a solution and use the arbitrary ordering of operations.
- The examples of Direct Heuristic search techniques include Breadth-First Search (BFS) and Depth First Search (DFS).

Weak Heuristic Search techniques in AI

- It includes Informed Search, Heuristic Search, and Heuristic control strategy. These techniques are helpful when they are applied properly to the right types of tasks. They usually require domain-specific information.
- The examples of Weak Heuristic search techniques include Best First Search (BFS) and A*. Before describing certain heuristic techniques, let's see some of the techniques listed below:
  - Bidirectional Search
  - A* search
  - Simulated Annealing
  - Hill Climbing
  - Best First search
  - Beam search

Hill Climbing Algorithm

- It is a technique for optimizing the mathematical problems. Hill Climbing is

widely used when a good heuristic is available.

- It is a local search algorithm that continuously moves in the direction of increasing elevation/value to find the mountain's peak or the best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value. Traveling-salesman Problem is one of the widely discussed examples of the Hill climbing algorithm, in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that. The steps of a simple hill-climbing algorithm are listed below:

  Step 1: Evaluate the initial state. If it is the goal state, then return success and Stop.

  Step 2: Loop Until a solution is found or there is no new operator left to apply.

  Step 3: Select and apply an operator to the current state.

  Step 4: Check new state:

  If it is a goal state, then return to success and quit.

  Else if it is better than the current state, then assign a new state as a current state. Else if not better than the current state, then return to step2. **Step 5:** Exit.

Best first search (BFS)

- This algorithm always chooses the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It lets us to take the benefit of both algorithms. It uses the heuristic function and search. With the help of the best-first search, at each step, we can choose the most promising node.

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 the goal node, then return success and stop the search, else continue to next step.

Step 6: For each successor node, the 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 lists, then add it to the OPEN list. **Step 7:** Return to Step 2.

A* Search Algorithm

- A* search is the most commonly known form of best-first search. It uses the 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. It finds the shortest path through the search space using the heuristic function. This search algorithm expands fewer search tree and gives optimal results faster.

Algorithm of A* search:

Step 1: 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 the evaluation function (g+h). If node n is the 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 the evaluation function for n' and place it into the 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.

Examples of heuristics in everyday life

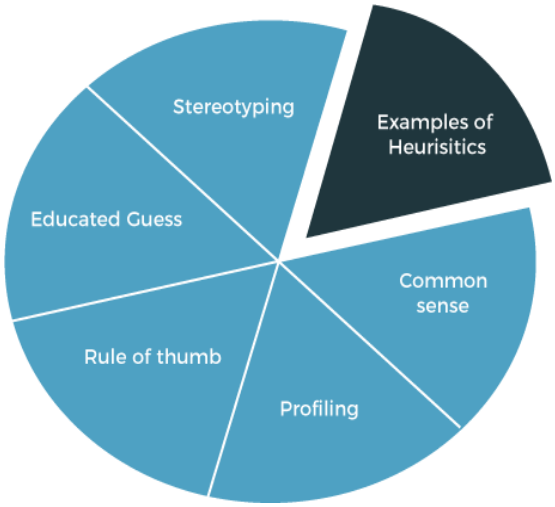


Fig.2.2 Heuristic in Every day life

Some of the real-life examples of heuristics that people use as a way to solve a problem:

- **Common sense:** It is a heuristic that is used to solve a problem based on the observation of an individual.
- **Rule of thumb:** In heuristics, we also use a term rule of thumb. This heuristic allows an individual to make an approximation without doing an exhaustive search.
- **Working backward:** It lets an individual solve a problem by assuming that the problem is already being solved by them and working backward in their minds to see how much a solution has been reached.
- **Availability heuristic:** It allows a person to judge a situation based on the examples of similar situations that come to mind.
- **Familiarity heuristic:** It allows a person to approach a problem on the fact that an individual is familiar with the same situation, so one should act similarly as he/she acted in the same situation before.
- **Educated guess:** It allows a person to reach a conclusion without doing an exhaustive search. Using it, a person considers what they have observed in the past and applies that history to the situation where there is not any definite answer has decided yet.

Types of heuristics

- There are various types of heuristics, including the availability heuristic, affect

heuristic and representative heuristic. Each heuristic type plays a role in decision-making. Let's discuss about the Availability heuristic, affect heuristic, and Representative heuristic.

Availability heuristic

- Availability heuristic is said to be the judgment that people make regarding the likelihood of an event based on information that quickly comes into mind. On making decisions, people typically rely on the past knowledge or experience of an event. It allows a person to judge a situation based on the examples of similar situations that come to mind.

Representative heuristic

- It occurs when we evaluate an event's probability on the basis of its similarity with another event.

Example:

- We can understand the representative heuristic by the example of product packaging, as consumers tend to associate the products quality with the external packaging of a product. If a company packages its products that remind you of a high quality and well-known product, then consumers will relate that product as having the same quality as the branded product.
- So, instead of evaluating the product based on its quality, customers correlate the products quality based on the similarity in packaging.

Affect heuristic

- It is based on the negative and positive feelings that are linked with a certain stimulus. It includes quick feelings that are based on past beliefs. Its theory is one's emotional response to a stimulus that can affect the decisions taken by an individual.
- When people take a little time to evaluate a situation carefully, they might base their decisions based on their emotional response.

Example:

- The affect heuristic can be understood by the example of advertisements. Advertisements can influence the emotions of consumers, so it affects the purchasing decision of a consumer. The most common examples of advertisements are the ads of fast food. When fast- food companies run the

advertisement, they hope to obtain a positive emotional response that pushes you to positively view their products.

- If someone carefully analyzes the benefits and risks of consuming fast food, they might decide that fast food is unhealthy. But people rarely take time to evaluate everything they see and generally make decisions based on their automatic emotional response. So, Fast food companies present advertisements that rely on such type of Affect heuristic for generating a positive emotional response which results in sales.

Limitation of heuristics

Along with the benefits, heuristic also has some limitations.

- Although heuristics speed up our decision-making process and also help us to solve problems, they can also introduce errors just because something has worked accurately in the past, so it does not mean that it will work again.
- It will hard to find alternative solutions or ideas if we always rely on the existing solutions or heuristics.

2. Explain in detail about Heuristic Functions in Artificial Intelligence.

Heuristic Functions in AI:

- 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.*
- Some toy problems, such as 8-puzzle, 8-queen, tic-tac-toe, etc., can be solved more efficiently with the help of a heuristic function.

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.

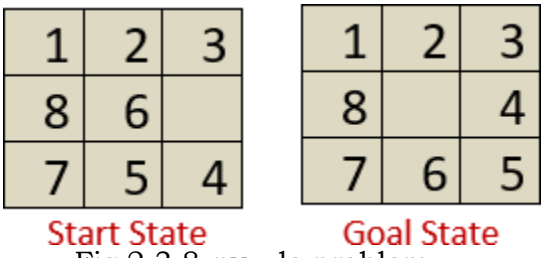


Fig.2.3 8-puzzle problem

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:

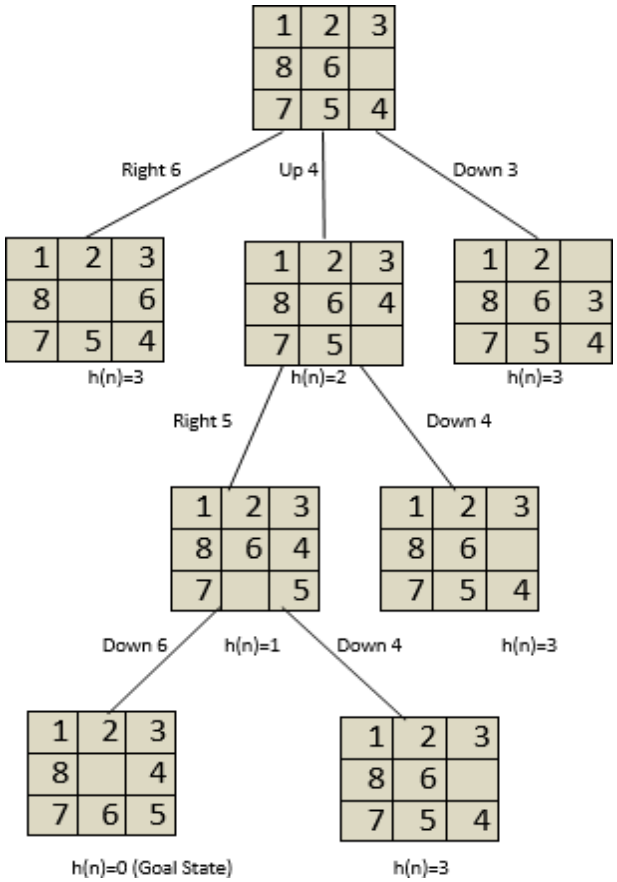


Fig.2.4 8-puzzle problem Solution

- 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

reqirement. 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 characterstics,** etc., which is expressed as a heuristic function.

3. Explain in detail about Local Search Algorithms and Optimization Problem.

Describe the local search algorithm with neat sketch. [Nov 2023]

Discuss on any two local search algorithms with examples. [Apr 2024]

Hill climbing

- Hill climbing is a straightforward local search algorithm that starts with an initial solution and iteratively moves to the best neighboring solution that improves the objective function.

Working

Initialization: Begin with an initial solution, often generated randomly or using a heuristic method.

Evaluation: Calculate the quality of the initial solution using an objective function or fitness measure.

Neighbor Generation: Generate neighboring solutions by making small changes (moves) to the current solution.

Selection: Choose the neighboring solution that results in the most significant improvement in the objective function.

Termination: Continue this process until a termination condition is met (e.g., reaching a maximum number of iterations or finding a satisfactory solution).

- Hill climbing has a limitation in that it can get stuck in local optima, which are solutions that are better than their neighbors but not necessarily the best overall solution. To overcome this limitation, variations of hill climbing algorithms have been developed, such as stochastic hill climbing and simulated annealing.

Local beam search

- Local beam search represents a parallelized adaptation of hill climbing, designed specifically to counteract the challenge of becoming ensnared in local optima. Instead

of starting with a single initial solution, local beam search begins with multiple solutions, maintaining a fixed number (the "beam width") simultaneously. The algorithm explores the neighbors of all these solutions and selects the best solutions among them.

Working

Initialization: Start with multiple initial solutions.

Evaluation: Evaluate the quality of each initial solution.

Neighbor Generation: Generate neighboring solutions for all the current solutions.

Selection: Choose the top solutions based on the improvement in the objective function.

Termination: Continue iterating until a termination condition is met.

- Local beam search effectively avoids local optima because it maintains diversity in the solutions it explores. However, it requires more memory to store multiple solutions in memory simultaneously.

Simulated Annealing

- Simulated annealing is a probabilistic local search algorithm inspired by the annealing process in metallurgy. It allows the algorithm to accept worse solutions with a certain probability, which decreases over time. This randomness introduces exploration into the search process, helping the algorithm escape local optima and potentially find global optima.

Working

Initialization: Start with an initial solution.

Evaluation: Evaluate the quality of the initial solution.

Neighbor Generation: Generate neighboring solutions.

Selection: Choose a neighboring solution based on the improvement in the objective function and the probability of acceptance.

Termination: Continue iterating until a termination condition is met.

- The key to simulated annealing's success is the "temperature" parameter, which controls the likelihood of accepting worse solutions. Initially, the temperature is high, allowing for more exploration. As the algorithm progresses, the temperature

decreases, reducing the acceptance probability and allowing the search to converge towards a better solution.

Travelling Salesman Problem

- The Travelling Salesman Problem (TSP) is a classic optimization problem in which a salesman is tasked with finding the shortest possible route that visits a set of cities exactly once and returns to the starting city. TSP is NP-hard, meaning that finding an exact solution for large instances becomes computationally infeasible.
- Local search algorithms, including hill climbing and simulated annealing, are often used to find approximate solutions to the TSP. In this context, the cities and their connections form the solution space, and the objective function is to minimize the total distance traveled.
- These algorithms iteratively explore different routes, making incremental changes to improve the tour's length. While they may not guarantee the absolute optimal solution, they often find high-quality solutions in a reasonable amount of time, making them practical for solving TSP instances.

4. Explain in detail about types of Hill Climbing Search Algorithm.

Types of Hill climbing search algorithm

There are following types of Hill-Climbing Search:

a) Simple hill climbing

b) Steepest-ascent hill climbing

c) Stochastic hill climbing

d) Random-restart hill climbing

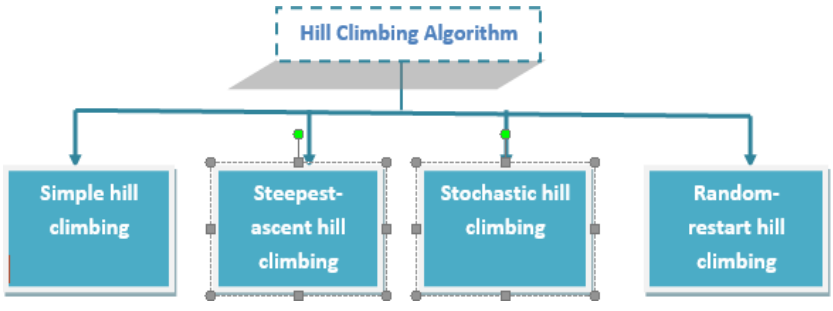


Fig.2.5 Types of Hill Climbing Search

a) Simple hill climbing search

- Simple hill climbing is the simplest technique to climb a hill. The task is to reach the highest peak of the mountain. Here, the movement of the climber depends on his move/steps. If he finds his next step better than the previous one, he continues to move else remain in the same state.

This search focus only on his previous and next step.

Simple hill climbing Algorithm

1. Create a **CURRENT** node, **NEIGHBOUR** node, and a **GOAL** node.
2. If the **CURRENT node=GOAL node**, return **GOAL** and terminate the search.
3. Else **CURRENT node<= NEIGHBOUR node,** move ahead.
4. Loop until the goal is not reached or a point is not found.

b) Steepest-ascent hill climbing

- Steepest-ascent hill climbing is different from simple hill climbing search. Unlike simple hill climbing search, It considers all the successive nodes, compares them, and choose the node which is closest to the solution. Steepest hill climbing search is similar to **best-first search** because it focuses on each node instead of one.

Note: Both simple, as well as steepest-ascent hill climbing search, fails when there is no closer node.

Steepest-ascent hill climbing algorithm

1. Create a **CURRENT** node and a **GOAL** node.
2. If the **CURRENT node=GOAL** node, return **GOAL** and terminate the search.
3. Loop until a better node is not found to reach the solution.
4. If there is any better successor node present, expand it.
5. When the **GOAL** is attained, return **GOAL** and terminate.

c) Stochastic hill climbing

- Stochastic hill climbing does not focus on all the nodes. It selects one node at random and decides whether it should be expanded or search for a better one.

d) Random-restart hill climbing

- Random-restart algorithm is based on **try and try strategy**. It iteratively searches the node and selects the best one at each step until the goal is not found. The success depends most commonly on the shape of the hill. If there are few plateaus, local maxima, and ridges, it becomes easy to reach the destination.

Limitations of Hill climbing algorithm

- Hill climbing algorithm is a fast and furious approach. It finds the solution state rapidly because it is quite easy to improve a bad state.

But, there are following limitations of this search:

- **Local Maxima:** It is that peak of the mountain which is highest than all its neighboring states but lower than the global maxima. It is not the goal peak because there is another peak higher than it.

Fig.2.6 Local Maxima

- **Plateau:** It is a flat surface area where no uphill exists. It becomes difficult for the climber to decide that in which direction he should move to reach the goal point. Sometimes, the person gets lost in the flat are

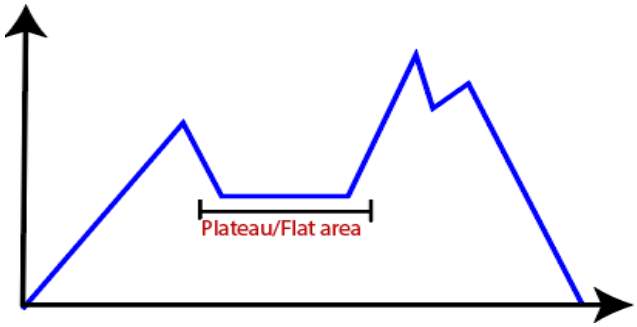


Fig.2.7 Plateau

- **Ridges:** It is a challenging problem where the person finds two or more local maxima of the same height commonly. It becomes difficult for the person to navigate the right point and stuck to that point itself.

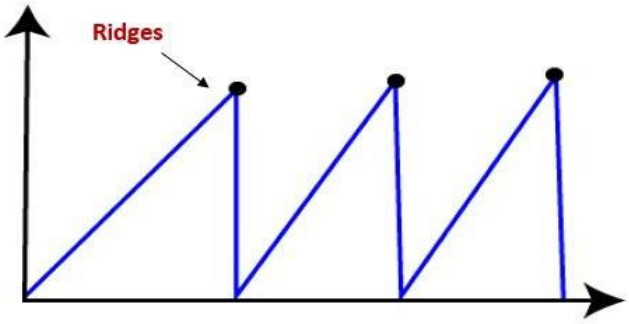


Fig.2.8 Ridges

5. Explain in detail about Local search in continuous space.

- A local search is first conducted in the continuous space until a local optimum is reached. It then switches to a discrete space that represents a Discretization of a continuous model to find an improved solution from there.
- The process continues switching between two problem formulations until no further improvement can be found in either.
- To perform local search in continuous state space we need techniques from Calculus.
- The main technique to find a minimum is called gradient descent.

Gradient Descent

- A gradient measures how much the output of a function changes if you change the inputs a little bit.
- In machine learning, a gradient is a derivative of a function that has more than one input variable known as the slope of a function in mathematical terms, the gradient simply measures the change in all weights with regard to the change in error.
- Gradient descent is an iterative optimization algorithm to find the minimum of a function.
- Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used in machine learning to find the values of a functions parameters (coefficients) that minimize a cost function as far as possible.

Linear Regression

- In statistics, linear regression is a linear approach to modelling the relationship between a dependent variable and one or more independent variables.

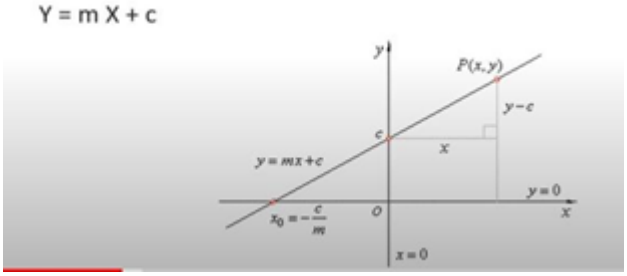


Fig.2.9 Linear Regression

Loss Function

The loss is the error in our predicted value of m and c. Our goal is to minimize this error to obtain the most accurate value of m and c.

Mean squared Error function to calculate the loss

Step1: Find the difference between the actual y and predicted y value(y=mx+c), for a given x.

Step2: square this difference

Step3: Find the mean of the squares for every value in x.

Mean Squared Error Equation

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

Substituting the value of

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

Gradient Descent Algorithm

- Applying gradient descent to m and c and approach it step by step
- Step1: Initially let m=0 and c=0. Let L be our learning rate. This controls how much the value of m changes with each step. L could be a small value like 0.0001 for good accuracy.
- Step2: Calculate the partial derivative of the loss function with respect to m, and plug in the current values of x,y,m and c in it to obtain the derivative value D

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

- Step3: Now we update the current value of m and c using the following equation:

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

- Step4: We repeat this process until our loss function is a very small value or ideally 0(which means 0 error or 100% accuracy). The value of m and c that we are left with now will be the optimum values.

6. Explain in detail about Search with non-deterministic actions.

- In **Deterministic environment** the agent knows that a particular action will take it from state S1 to another state S2.
- In **Non- deterministic environment** the agent is not sure that a particular action would take from state S1 to S2.
- **Fully Observable Environment** is the one in which the agent has sensors which gives the complete information about the environment to the agent.
- **Partially Observable Environment** is the one in which the agent has incomplete information of the environment, due to faulty sensors or noisy sensors.

The Erratic Vacuum World

- This is an example of Non deterministic Action
- We consider the vacuum world having an error in the suck operation, there are two erratic operations with respect to suck operation

The Suck action works as follows

- If the tile is dirty, the suck action cleans up the tile and sometimes also clean up dirt in adjacent tiles
- If the tile is clean, the suck action sometimes deposits dirt on the carpet.

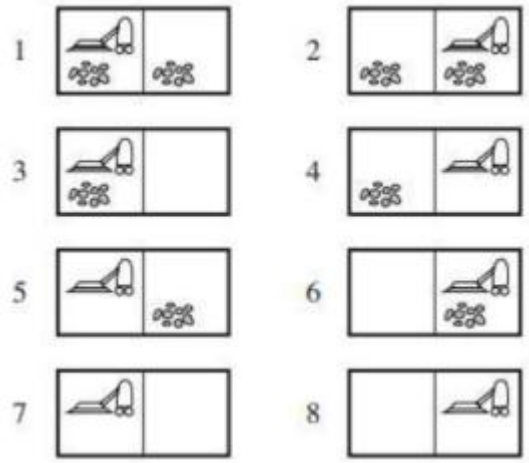


Fig.2.10 Vacuum cleaner works

- With non-deterministic Action, if the agent takes the Suck action in state 1 then the agent can be in either in state 5 or state 7
- There must be a contingency plan to get the goal, as a sequence of actions will not be sufficient.

AND-OR search trees

- In a deterministic environment, the branching depends only on agents choice of action, such nodes are called as OR node (In Vacuum world the possible actions are Left OR Right OR suck
- In a non-deterministic environment, branching also depends on the environments choice of the outcome, such nodes are called as AND nodes
- Hence we have AND-OR search tree in such cases

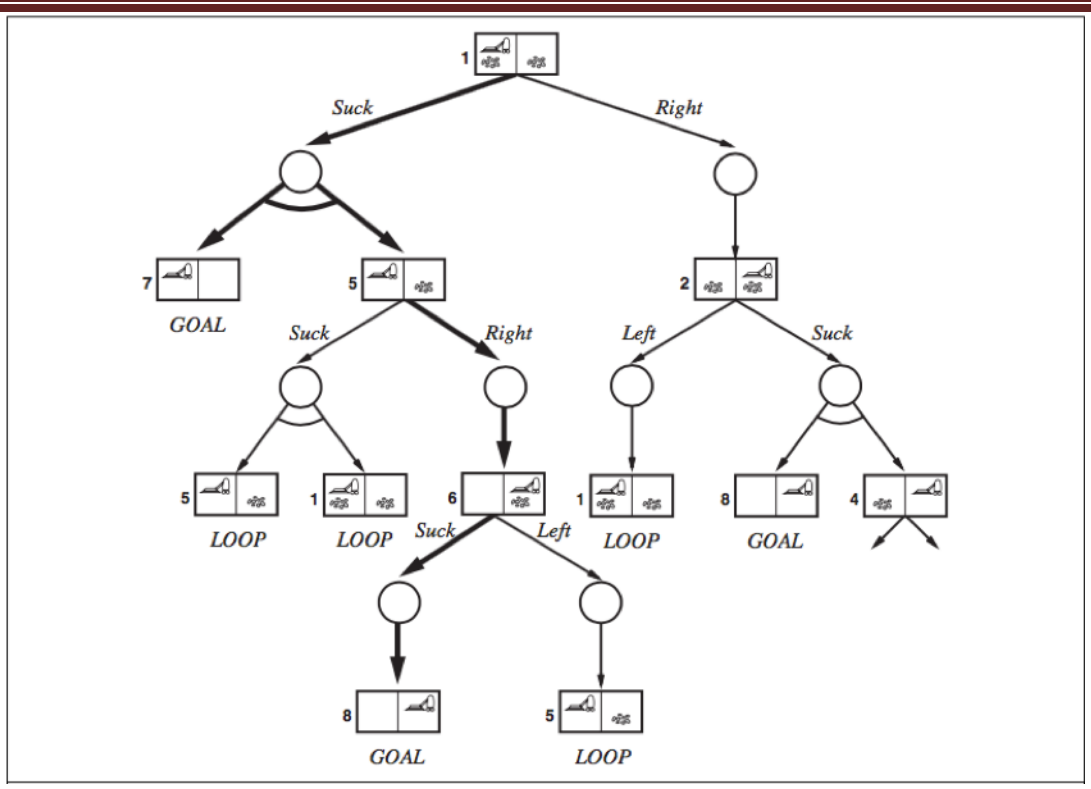


Fig.2.11 AND-OR search trees for Vacuum cleaner

A solution for an AND-OR search problem is a subtree that

1. Has goal node at every leaf

2. Specifies one action at each of its OR node

3. Includes every outcome branch at each of its AND nodes.

AND-OR search can be explored using recursive Depth-first search or by Breadth first or Best first methods.

Slippery Vacuum World

- Another case of Non Deterministic action in which the Right action fails sometimes. When the agent is in the Left region, and takes the Right action, it may go to the right region or may remain in the Left region if the operation fails

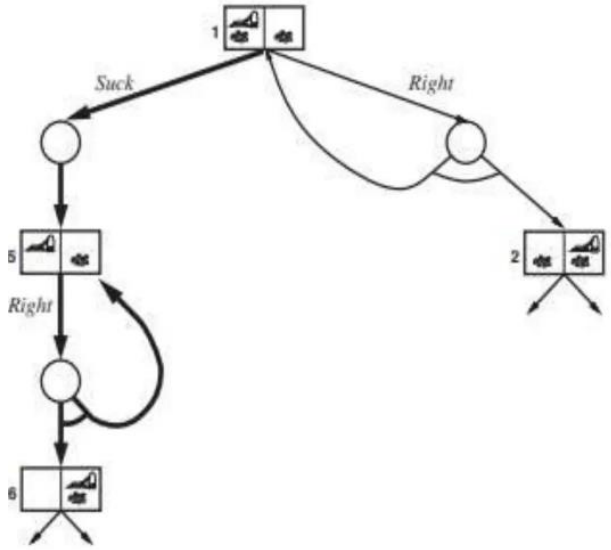


Fig.2.12 Slippery Vacuum world

7. Explain in detail about Search in partial observable environments.

Partial Observability:

- The agents percepts do not suffice to pin down the exact state. In such environment an action may lead to one of several possible outcomes-even if the environment is deterministic.
- To solve partially observable problems the concept of the "belief state" is utilized. The belief state represents the agents current belief about the possible physical states it might be in, given the sequence of actions and percepts up to that point.

The belief state concept can be studied in two different scenarios.

- Searching with no Observations
- Searching with Observations

Sensorless or conformant agents:

- Percepts provide no information at all
- They are surprisingly useful, because they don't rely on sensors working properly
- Sensorless vacuum world (geography known, not location & dirt distribution)
    - Initial state could be any element of the set{1,2,3,4,5,6,7,8}
    - Action right will cause it to be in one of the states {2,4,6,8}

- The action sequence [Right,Suck] will always end up in one of the states {4,8}
- Sequence[Right,Suck,Left,Suck] is guaranteed to reach the goal state 7 irrespective of the start state.

- To solve sensorless problems, space of belief-states is searched rather than physical
- Believe-state space is fully observable as that as it is always known to the agent
- Suppose the underlying physical problem P is defined by ACTIONp, RESULTp,GOAL-TESTp, and STEP-COSTp.
- **Belief states:** If P has N states, then the sensorless problem has upto 2^N states
- **Initial state:** Typically the set of all states in P
- **Actions:** If a belief state b={s1,s2}, and ACTIONp(s1)≠ACTIONSP(s2); legal actions?
  - Take union of all the actions, given that illegal actions has no effect on the environment
  - Otherwise intersection, i.e the set of actions legal in all the states (illegal action might be the end of the world)
- **Transition model:**
  - With deterministic action b'=RESULT(b,a)=(s':s'=RESULT(s,a) and s€b} sometimes called prediction
  - With non deterministic actions b'=RESULT{s':s'€RESULTp(s,a) and s€b}}=[RESULTp(s,a) union of all
- **Goal Test:** a belief state is a goal state if all physical states in it satisfy GOAL-TESTp.
- **Path cost:** Same action can have different costs in different states of a belief state, an assumption can make the formulation easy, i.e the cost of an action is the same in all states

Searching with Observation:

- In partial observations, usually several states could have produced any given percept

- The percept [a,Dirty] is produced by state 3 as well as by state 1 in local-sensing vacuum (where only a position sensor and a local dirt sensor but no global dirt sensor)
- In such case the initial belief state state for the local sensing vacuum world will be {1,3}
- ACTIONS,STEP-COST, and GOAL-TEST are same as for sensorless problems
- Transit model is a bit different and consists of three stages
    - Prediction: given the action a in belief state b, b'=PREDICT(b,a), same as for sensor less
    - Observation Prediction: set of percepts that could be observed in the predicted belief state POSSIBLE-PERCEPTS(b')={o:o=PERCEPT(s) and s€b'}
    - Update: for each possible percept, the belief ststae that would result from the percept bo=UPDATE(b,o)={s:o=PERCEPT(s) and s€b} updated belief state bo can be no larger than the predicted belief state n'.

Searching with Observation

- Putting these three stages together, to get the transition model
- RESULTS(b,a)={bo:bo=UPDATE(PREDICT(b,a),o),o) and

  POSSIBLE-PERCEPTS(PREDICT(b,a))}
- The next belief state is achieved a given action a the subsequent possible percepts
- Figure a, Shows transition model for local sensor vacuum world
- Figure b shows transition model for slippery vacuum world

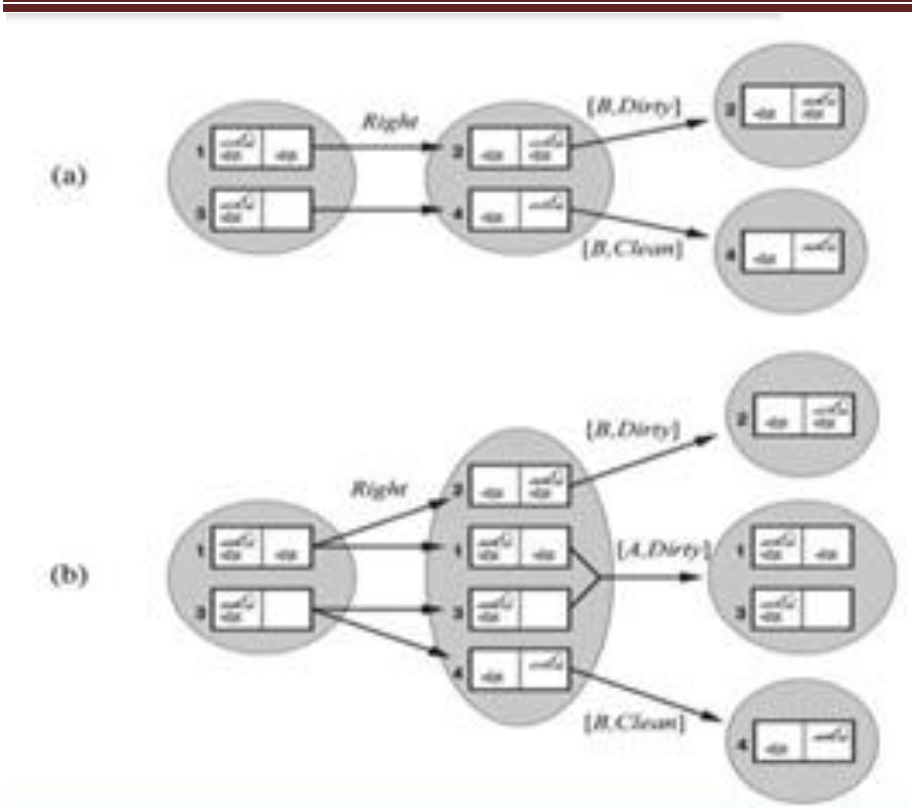


Fig.2.13 (a) Transition model for Local sensor vacuum

(b) Transition model for slippery vacuum world

Solving partially observable problems

- Given the formulation discussed previously, the AND-OR search algorithm can be applied directly to derive a solution
- The solution is the conditional plan that tests the belief state rather than the actual state
- The first level of the AND-OR search tree for a problem in the local sensing vacuum is shown in Fig.

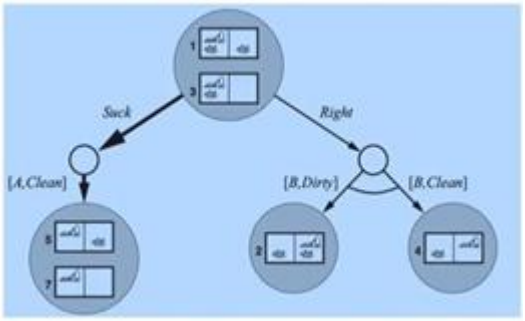


Fig.2.14 First Level of AND-OR search tree

Online search problems

- The offline search algorithms compute a complete solution before setting foot in the real world and then execute the solution
- Online search takes an action, then it observes the environment and computes the next
- Suitable for dynamic or semidynamic domains where there is a penalty for sitting around and computing too long

Also useful for

- Unknown environments
- Non-deterministic domains

Examples

- Web search
- Autonomous Vehicle

8. Explain in detail about Online Search Agents and Unknown Environments.

Discuss on online search agents with example. [Apr 2024]

1) The online search agent works on the concept of interleaving of two tasks namely computation and action. In the real-world, the agent perform an action and it observes the environment and then work out on the next action.
2) If an environment is dynamic or semidynamic or stochastic the online search work in best manner.
3) An agent needs to pay a penalty for lengthy computation and for sitting around.
4) An offline search is costly. For offline search we need to have proper planning which consider all related information. In online search there is no need of planning just see what happens and act accordingly.

   Example - Chess moves
5) In online search the states and actions are unknown to the agent. Thus it has an exploration problem. (It needs to explore the world).
6) In an online search the actions are used to predict next states and actions.
7) For predicting next states and actions computations are performed.

For example

- Consider example of robot who builds a map between 2 locations say LOC 1 and LOC 2. For building a map robot needs to explore the world so as to reach to LOC 2 from LOC 1. Parallely it will build the map.
- A natural real world example of exploration problem and online search is a baby trying to understand the world around it. It is baby's online search process. A baby tries an action and go in certain state. It keeps on doing this, gathering new experience and learning from it.

Solving Online Search Problems

1) The purely computation process cannot handle an online search problem properly. Agent needs to execute actions for solving problem.
2) The agent has knowledge about -
    a. ACTIONS (s) which returns a list of actions allowed in state s.
    b. The step-cost function cost (s1, a, s') (note that cost function cannot be used until the agent knows that s' is the outcome).
    c. GOAL-TEST (s)
3) It is not possible for agent to access the successors of a state. In fact the agent try related actions, in that state.
4) As basic objective of an agent is to minimize cost, the goal state must be found with minimum cost.
5) The another possible goal can be to explore the entire environment. The total path cost is cost of path across which the agent travels.
6) In the online algorithm we can find the competitive ratio means shortest path. The competitive ratio must be very small. But in reality many a time competitive ratio is infinite.
7) If the online search is with irreversible action then it will reach to a dead-end state and from that state it can't achieve goal state.
8) We must build an algorithm which does not explore dead-end path. But accidently or in reality an algorithm can avoid dead ends in all state spaces.
9) Consider two goal state space as shown in Fig. 4.4.1.

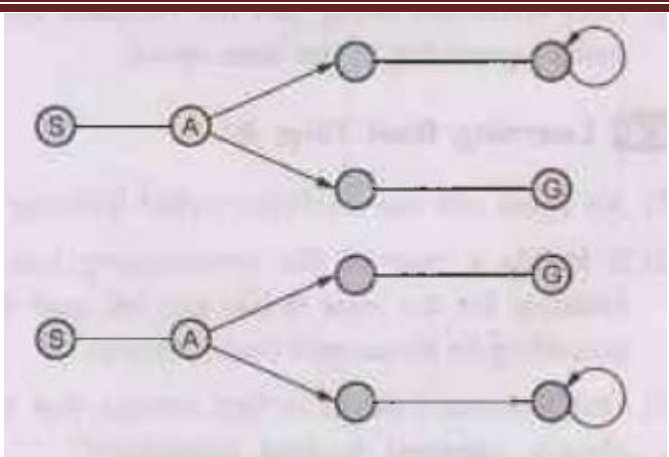


Fig.2.15 Two goal state space which may lead to a dead end

- In the Fig.2.15 online search algorithm visit states S and A, both the state spaces are identical so it must be explored in both cases. One link is to goal state (G) and other lead to dead end in both states spaces. It is an example of an adversary arguments. The exploration of state space is in an adversary manner. One can put the goals and dead ends likewise.

Online Search Agents

- **Online search agents** are algorithms or systems designed to explore and find solutions in environments where the complete information about the state space is not known beforehand. Unlike offline search, where the entire problem space is available in advance, online search agents must make decisions and adapt as they gather new information.

Characteristics of Online Search Agents

1. **Partial Information**: They operate with incomplete knowledge of the environment and must make decisions based on the information available at each step.
2. **Exploration vs. Exploitation**: They often face the challenge of balancing exploration (discovering new areas) with exploitation (using known information to make the best decisions).
3. **Adaptation**: They need to adapt their strategy as they learn more about the environment. This can involve changing their search strategy or modifying their goals.
4. **Real-Time Decision Making**: Online search agents typically need to make decisions in real time, without the luxury of extensive computation time.

Unknown Environments

An **unknown environment** is a scenario where the agent has no prior knowledge about the layout or structure of the environment. This can include:

- **Dynamic Environments**: Where the environment changes over time, making it challenging to develop a fixed strategy.
- **Stochastic Environments**: Where there are probabilistic elements affecting outcomes, adding uncertainty to decision-making.
- **Adversarial Environments**: Where other agents or entities may act in ways that are not predictable or friendly.

Strategies for Online Search in Unknown Environments

1. **Exploration Strategies**:
    - **Random Exploration**: Moving randomly to gather information, which can be useful when little is known.
    - **Heuristic Exploration**: Using heuristics or rules of thumb to guide exploration based on partial information.
2. **Adaptive Strategies**:
    - **Learning-Based Approaches**: Employing machine learning to improve decision-making based on experience and gathered data.
    - **Reinforcement Learning**: Learning through rewards and penalties to improve performance over time.
3. **Planning and Decision-Making**:
    - **Model-Free Methods**: Techniques like Q-learning, where the agent learns to make decisions based on rewards without needing a model of the environment.
    - **Model-Based Methods**: Creating and updating a model of the environment to plan and make better-informed decisions.
4. **Dynamic Adjustments**:
    - **Online Planning**: Continuously planning based on the latest information rather than relying on a precomputed plan.
    - **Reactive Strategies**: Responding to changes and new information in real time, often used in conjunction with planning.

Applications

Online search agents in unknown environments are used in various fields, including:

- **Robotics**: For robots navigating unknown terrains or performing tasks in dynamic settings.
- **Autonomous Vehicles**: For self-driving cars that must adapt to new road conditions and obstacles.
- **Game AI**: For creating intelligent behaviors in games where the environment can change or is not fully known.

Challenges

1. **Computational Efficiency**: Making real-time decisions with limited computational resources.
2. **Robustness**: Ensuring that the agent can handle unexpected changes or uncertainties effectively.
3. **Scalability**: Adapting strategies to work in increasingly complex or larger environments.

- In summary, online search agents in unknown environments are designed to handle the unpredictability and incomplete information typical of real-world scenarios. They leverage various strategies to explore, learn, and make decisions dynamically, often using techniques from reinforcement learning and adaptive planning.

9. Explain the steps involved in formulating problems with example.

Problem Solving in AI

- The reflex agent of AI directly maps states into action. Whenever these agents fail to operate in an environment where the state of mapping is too large and not easily performed by the agent, then the stated problem dissolves and sent to a problem-solving domain which breaks the large stored problem into the smaller storage area and resolves one by one. The final integrated action will be the desired outcomes.
- On the basis of the problem and their working domain, different types of problem-solving agent defined and use at an atomic level without any internal state visible with a problem-solving algorithm. The problem-solving agent performs precisely by defining problems and several solutions. So we can say that problem solving is a part of artificial intelligence that encompasses a number of techniques such as a tree, B-tree, heuristic algorithms to solve a problem.

- We can also say that a problem-solving agent is a result-driven agent and always focuses on satisfying the goals.
    - There are basically three types of problem in artificial intelligence:
    - Ignorable: In which solution steps can be ignored.
    - Recoverable: In which solution steps can be undone.
    - Irrecoverable: Solution steps cannot be undo.

Steps problem-solving in AI:

The problem of AI is directly associated with the nature of humans and their activities. So we need a number of finite steps to solve a problem which makes human easy works.

These are the following steps which require to solve a problem :

- Problem definition: Detailed specification of inputs and acceptable system solutions.
- Problem analysis: Analyse the problem thoroughly.
- Knowledge Representation: collect detailed information about the problem and define all possible techniques.
- Problem-solving: Selection of best techniques.

Components to formulate the associated problem:

- Initial State: This state requires an initial state for the problem which starts the AI agent towards a specified goal. In this state new methods also initialize problem domain solving by a specific class.
- Action: This stage of problem formulation works with function with a specific class taken from the initial state and all possible actions done in this stage.
- Transition: This stage of problem formulation integrates the actual action done by the previous action stage and collects the final stage to forward it to their next stage.
- Goal test: This stage determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determines the cost to achieve the goal.
- Path costing: This component of problem-solving numerical assigned what will be the cost to achieve the goal. It requires all hardware software and human working cost.

10. Write short notes on genetic algorithm. [Nov 2023]

- Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space.

They are commonly used to generate high-quality solutions for optimization problems and search problems.

- **Genetic algorithms simulate the process of natural selection** which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate "survival of the fittest" among individuals of consecutive generations to solve a problem.
- **Each generation consists of a population of individuals** and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

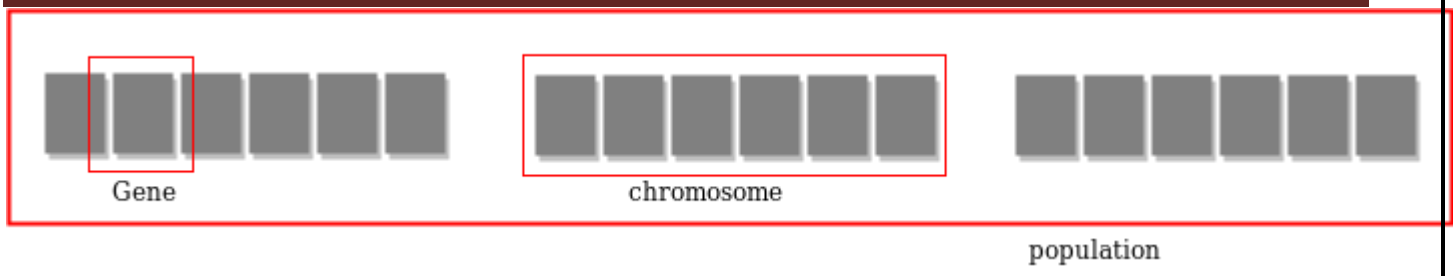
Foundation of Genetic Algorithms

Genetic algorithms are based on an analogy with the genetic structure and behavior of chromosomes of the population. Following is the foundation of GAs based on this analogy –

1. Individuals in the population compete for resources and mate
2. Those individuals who are successful (fittest) then mate to create more offspring than others
3. Genes from the "fittest" parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
4. Thus each successive generation is more suited for their environment.

Search space

- The population of individuals are maintained within search space. Each individual represents a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).



Fitness Score

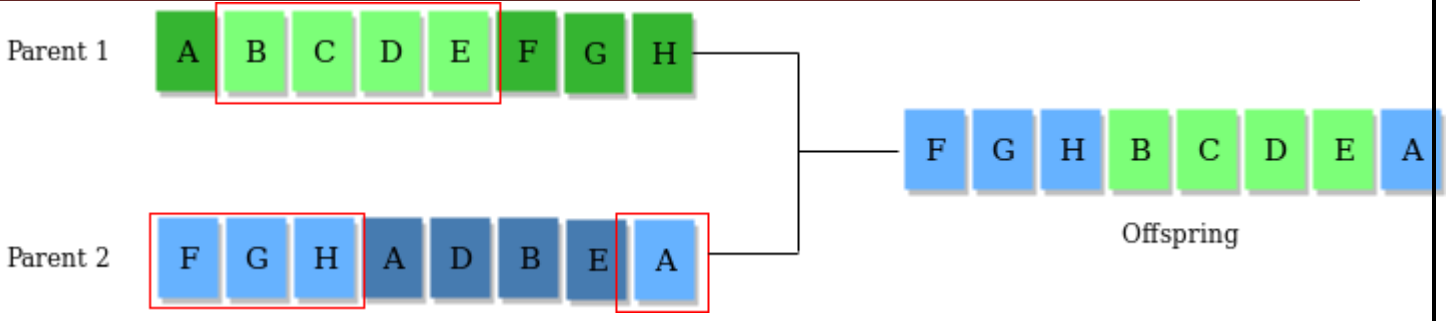
- A Fitness Score is given to each individual which **shows the ability of an individual to "compete"**. The individual having optimal fitness score (or near optimal) are sought.
- The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness scores.The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce **better offspring** by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die.
- Each new generation has on average more "better genes" than the individual (solution) of previous generations. Thus each new generations have better **"partial solutions"** than previous generations. Once the offspring produced having no significant difference from offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

Operators of Genetic Algorithms

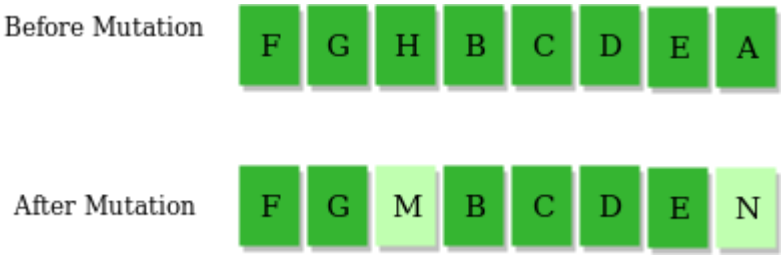
Once the initial generation is created, the algorithm evolves the generation using following operators –

1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.

2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring). For example –



3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence. For example –



The whole algorithm can be summarized as –

1. Randomly initialize populations p
2. Determine fitness of population
3. Until convergence repeat:
   - Select parents from population
   - Crossover and generate new population
   - Perform mutation on new population
   - Calculate fitness for new population

Example problem and solution using Genetic Algorithms

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9, and other special symbols are considered as genes
- A string generated by these characters is considered as chromosome/solution/Individual

Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.

Why use Genetic Algorithms

- They are Robust
- Provide optimisation over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

Application of Genetic Algorithms

Genetic algorithms have many applications, some of them are –

- Recurrent Neural Network
- Mutation testing
- Code breaking
- Filtering and signal processing
- Learning fuzzy rule base etc

MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by Naac with A Grade & TCS

AL3391 ARTIFICIAL INTELLIGENCE

UNIT III

Game theory – optimal decisions in games – alpha-beta search – monte-carlo tree search – stochastic games – partially observable games. Constraint satisfaction problems – constraint propagation – backtracking search for CSP – local search for CSP – structure of CSP.

PART A

1. What are the components of Game software?

Write the components of a game. [Nov 2023]

- Key components of games are goals, rules, challenge, and interactivity. Games generally involve mental or physical stimulation, and sometimes both. Many games help develop practical skills, serve as a form of exercise, or otherwise perform an educational, simulational or psychological role.

2. Define alpha & beta values in a game tree.

- A modified strategy in game trees is called alpha-hero pruning. It requires the maintenance of two threshold values, one representing a lower bound on the value that a maximizing node may ultimately be assigned (we call this alpha) and a value representing an upper bound on the value that a minimizing node may be assigned (we call this beta).

3. Define Constraint Satisfaction Problem. (CSP).

- A constraint satisfaction problem is a special kind of problem satisfies some additional structural properties beyond the basic requirements for problem in general. In a CSP, the states are defined by the values of a set of variables and the goal test specifies a set of constraint that the value must obey.

4. Define backtracking search in CSP.

- The term **backtracking search** is used for depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

5. Define Constraint propagation (Inference in CSP) (or) List out the Classification of CSP with respect to constraints.

What do you mean by constraint propagation? [Nov 2023]

- Although forward checking detects many inconsistencies, it does not detect all of them.

Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables

- Node consistency
- Arc consistency
- Path consistency
- K-consistency
- Global constraints

6. Define Alpha-Beta pruning.

- Problem with minimax search is that number of games state it has to examine is exponential in depth of tree.
- Trick is that it is possible to compute correct minimax decision without looking at every node in game tree.
- We can borrow idea of pruning to eliminate large parts of tree from consideration .
- Particular technique we examine is called alpha-beta pruning.
- When applied to a standard minimax tree, it returns same move as minimax would, but away branches that cannot possibly influence final decision.

7. What do you meant by Stochastic Games?

- In real life, many unpredictable external events can put us into unforeseen situations. Many games mirror this unpredictability by including a random element, such as the throwing of dice. We call these **stochastic games**. Backgammon is a typical game that combines luck and skill.

8. List the ways to formulate Constraint satisfaction problem.

A problem to be converted to CSP requires the following steps:

- **Step 1:** Create a variable set.
- **Step 2:** Create a domain set.
- **Step 3:** Create a constraint set with variables and domains (if possible) after considering the constraints.
- **Step 4:** Find an optimal solution.

9. What are the Elements of Game Playing search?

- **S0:** It is the initial state from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves to be used in a state.

- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s)**: It defines that the game has ended and returns true.
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**.
- The price which the winner will get i.e.

  (-1): If the PLAYER loses.

  (+1): If the PLAYER wins.

  (0): If there is a draw between the PLAYERS.

10. List the types of algorithms in adversarial search.

- Minimax Algorithm
- Alpha-beta Pruning

11. Define a Game Tree.

- **A game tree** is where the nodes represent the states of the game and edges represent the moves made by the players in the game.
- Players will be two namely:
- **MIN:** Decrease the chances to win the game.
- **MAX:** Increases his chances of winning the game.
- MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

12. List the CSP Algorithms and Problems.

- Crypt Arithmetic (Coding alphabets to numbers.)
- N-Queen (In an n-queen problem, n queens should be placed in an nXn matrix such that no queen shares the same row, column or diagonal.)
- Map Coloring (coloring different regions of map, ensuring no adjacent regions have the same color)
- Crossword (everyday puzzles appearing in newspapers)
- Sudoku (a number grid)
- Latin Square Problem.

13. List the Types of Domains in CSP.

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example,** a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

14. List the Constraint Types in CSP.

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.

- **Binary Constraints:** It is the constraint type which relates two variables. A value **x2** will contain a value which lies between **x1** and **x3**.
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

15. Define Map Coloring / Graph Coloring Problem.

- It is a map of **Australia**; it consists of states and territories
- The task will be coloring each region with the colors red,green or blue.
- In such case, no neighboring regions will have the same color.

16. Define Game theory.

- **Game theory** is basically a branch of mathematics that is used to typical strategic interaction between different players (agents), all of which are equally rational, in a context with predefined rules (of playing or maneuvering) and outcomes.
- A GAME can be defined as a set of players, actions, strategies, and a final playoff for which all the players are competing.
- Game Theory has now become a describing factor for both Machine Learning algorithms and many daily life situations.

17. Define Nash Equilibrium.

- Nash equilibrium can be considered the essence of Game Theory. It is basically a state, a point of equilibrium of collaboration of multiple players in a game. Nash Equilibrium guarantees maximum profit to each player. Let us try to understand this with the help of Generative Adversarial Networks (GANs).

18. What is GAN?

- A GAN consists of two neural networks, **namely the generator and the discriminator**, which are trained simultaneously through a competitive process.
- Here's how they work:
  1. **Generator:** The generator network takes random noise as input and generates data (e.g., images, text, audio) that should resemble the real data from a specific dataset. It tries to create data that is indistinguishable from real data.
  2. **Discriminator:** The discriminator network, on the other hand, receives both real data from the dataset and fake data generated by the generator. Its job is to distinguish between the real and fake data. It assigns a probability score to each piece of data, indicating how likely it is to be real.

19. What are types of Game?

1. Zero-Sum and Non-Zero Sum Games
2. Simultaneous and Sequential Games
3. Asymmetric and Symmetric Games

4. Co-operative and Non-Co-operative Games

20. What is Zero-Sum and Non-Zero Sum Games?

- In non-zero-sum games, there are multiple players and all of them have the option to gain a benefit due to any move by another player. In zero-sum games, however, if one player earns something, the other players are bound to lose a key playoff.

21. What is Simultaneous and Sequential Games?

- Sequential games are the more popular games where every player is aware of the movement of another player. Simultaneous games are more difficult as in them, the players are involved in a concurrent game. BOARD GAMES are the perfect example of sequential games and are also referred to as turn-based or extensive-form games.

22. What is Asymmetric and Symmetric Games?

- Asymmetric games are those win in which each player has a different and usually conflicting final goal. Symmetric games are those in which all players have the same ultimate goal but the strategy being used by each is completely different.

23. What is Co-operative and Non-Co-operative Games?

- In non-co-operative games, every player plays for himself while in co- operative games, players form alliances in order to achieve the final goal.

24. Define Monte Carlo Tree Search (MCTS.

- Monte Carlo Tree Search (MCTS) is a search technique in the field of Artificial Intelligence (AI).
- It is a probabilistic and heuristic driven search algorithm that combines the classic tree search implementations alongside machine learning principles of reinforcement learning.
- In tree search, there's always the possibility that the current best action is actually not the most optimal action.

25. How can of Monte Carlo Tree Search can be broken down and List out that?

- The process of Monte Carlo Tree Search can be broken down into four distinct steps, viz.,
  - Selection.
  - Expansion.
  - Simulation.
  - Back propagation

26. Define K-Consistency.

A graph is K-consistent if the following is true:

- Choose values of any K-1 variables that satisfy all the constraints among these variables and choose any Kth variable. Then there exists a value for this Kth variable that satisfies all the constraints among these K variables.

27. Define Arc Consistency.

- The pair (X, Y) of constraint variables is arc consistent if for each value x ∈ D X there exists a value y ∈ D y such that the assignments X = x and Y = y satisfy all binary constraints between X and Y. A CSP is arc consistent if all variable pairs are arc consistent.

28. What are the advantages and disadvantages?

Advantages

- MCTS is a simple algorithm to implement.
- Monte Carlo Tree Search is a heuristic algorithm. MCTS can operate effectively without any knowledge in the particular domain, apart from the rules and end conditions, and can find its own moves and learn from them by playing random playouts.
- The MCTS can be saved in any intermediate state and that state can be used in future use cases whenever required.

Disadvantages

- It requires a huge amount of memory.
- There is a bit of a reliability issue with Monte Carlo Tree Search.
- MCTS algorithm needs a huge number of iterations to be able to effectively decide the most efficient path. So, there is a bit of a speed issue there.

29. Give two jugs of capacities 5 litres and 3 litres with no measuring markers on them. Assume that there is endless supply of water. What is the minimum number of states to measure 4 litres of water? [Apr 2024]

- To measure exactly 4 litres using a 5-litre jug and a 3-litre jug, you can follow these steps:
    - Fill the 3-litre jug completely with water.
    - Pour the water from the 3-litre jug into the 5-litre jug. This will leave you with 2 litres of water in the 3-litre jug.
    - Empty the water from the 5-litre jug.
    - Pour the remaining 2 litres of water from the 3-litre jug into the 5-litre jug.
    - Fill the 3-litre jug again with water.
    - Pour the water from the 3-litre jug into the 5-litre jug until the 5-litre jug is full. This will take 1 litre of water from the 3-litre jug, leaving you with exactly 4 litres of water in the 5-litre jug.
    - By following these steps, you can measure exactly 4 litres of water using a 5-litre jug and a 3-litre jug.

30. What for Monte-Carlo tree search is used? [Apr 2024]

- Monte Carlo tree search is a heuristic search algorithm that relies on intelligent tree search to make decisions. It's most often used to perform game simulations, but it can also be utilized in cybersecurity, robotics and text generation.

PART B

1. Explain in detail about Game theory.

- **Game theory** is basically a branch of mathematics that is used to typical strategic interaction between different players (agents), all of which are equally rational, in a context with predefined rules (of playing or maneuvering) and outcomes.

- Every player or agent is a rational entity who is selfish and tries to maximize the reward to be obtained using a particular strategy. All the players abide by certain rules in order to receive a predefined playoff- a reward after a certain outcome.
- Hence, a Game can be defined as a set of players, strategies, and a final playoff for which all the players are competing. Game theory has now become a describing factor for both Machine Learning algorithms and many daily life situations.
- Consider the SVM (Support Vector Machine) for instance. According to Game Theory, the SVM is a game between 2 players where one player challenges the other to find the best hyper- plane after providing the most difficult points for classification. The final playoff of this game is a solution that will be a trade-off between the strategic abilities of both players competing.

Nash equilibrium:

- Nash equilibrium can be considered the essence of Game Theory. It is basically a state, a point of equilibrium of collaboration of multiple players in a game. Nash Equilibrium guarantees maximum profit to each player.

Generative Adversarial Networks (GANs).

GAN

- It is a combination of two neural networks: the Discriminator and the Generator. The Generator Neural Network is fed input images which it analyzes and then produces new sample images, which are made to represent the actual input images as close as possible.
- Once the images have been produced, they are sent to the Discriminator Neural Network. This neural network judges the images sent to it and classifies them as generated images and actual input images. If the image is classified as the original image, the DNN changes its parameters of judging. If the image is classified as a generated image, the image is rejected and returned to the GNN.
- The GNN then alters its parameters in order to improve the quality of the image produced.
- This is a competitive process which goes on until both neural networks do not require to make any changes in their parameters and there can be no further improvement in both neural networks.
- This state of no further improvement is known as NASH EQUILIBRIUM. In other

words, GAN is a 2-player competitive game where both players are continuously optimizing themselves to find a Nash Equilibrium.

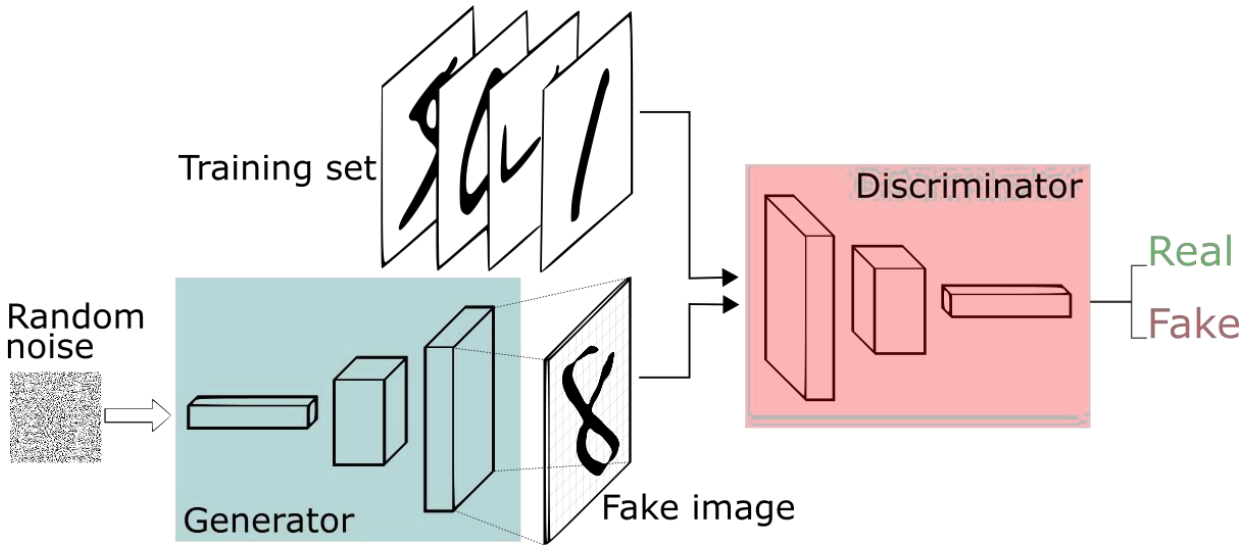


Fig.3.1 Generative Adversarial Networks

But how do we know if the game has reached Nash Equilibrium?

- In any game, one of the agents is required to disclose their strategy in front of the other agents. After the revelation, if none of the players changes their strategies, it is understood that the game has reached Nash Equilibrium.
- Now that we are aware of the basics of Game Theory, let us try to understand how Nash Equilibrium is attained in a simultaneous game. There are many examples but the most famous is the Prisoner's Dilemma. There are some more examples such as the Closed-bag exchange Game, the Friend or For Game, and the iterated Snowdrift Game.
- In all these games, two players are involved and the final playoff is a result of a decision that has to be made by both players. Both players have to make a choice between defection and co- operation. If both players cooperate, the final playoff will turn out to be positive for both. However, if both defect, the final playoff will be negative for both players. If there is a combination of one player defecting and the other co-operating, the final playoff will be positive for one and negative for another.
- Here, Nash Equilibrium plays an important role. Only if both players jot out a strategy that benefits each other and provide both with a positive playoff, the solution to this problem will be optimal.
- There are many more real examples and a number of pieces of code that try to solve

this dilemma. The basic essence, however, is the attainment of the Nash Equilibrium in an uncomfortable situation.

Where is GAME THEORY now?

- Game Theory is increasingly becoming a part of the real-world in its various applications in areas like public health services, public safety, and wildlife. Currently, game theory is being used in adversary training in GANs, multi-agent systems, and imitation and reinforcement learning. In the case of perfect information and symmetric games, many Machine Learning and
- Deep Learning techniques are applicable. The real challenge lies in the development of techniques to handle incomplete information games, such as Poker. The complexity of the game lies in the fact that there are too many combinations of cards and the uncertainty of the cards being held by the various players.

Types of Games:

Currently, there are about 5 types of classification of games. They are as follows:

1. Zero-Sum and Non-Zero Sum Games: In non-zero-sum games, there are multiple players and all of them have the option to gain a benefit due to any move by another player. In zero- sum games, however, if one player earns something, the other players are bound to lose a key playoff.
2. Simultaneous and Sequential Games: Sequential games are the more popular games where every player is aware of the movement of another player. Simultaneous games are more difficult as in them, the players are involved in a concurrent game. BOARD GAMES are the perfect example of sequential games and are also referred to as turn-based or extensive-form games.
3. Perfect Information game.
4. Asymmetric and Symmetric Games: Asymmetric games are those win in which each player has a different and usually conflicting final goal. Symmetric games are those in which all players have the same ultimate goal but the strategy being used by each is completely different.
5. Co-operative and Non-Co-operative Games: In non-co-operative games, every player plays for himself while in co-operative games, players form alliances in order to achieve the final goal.

2. Explain in detail about Optimal Decisions in Games.

- Humans' intellectual capacities have been engaged by games for as long as civilization has existed, sometimes to an alarming degree. Games are an intriguing subject for AI researchers because of their abstract character.
- A game's state is simple to depict, and actors are usually limited to a small number of actions with predetermined results. Physical games, such as croquet and ice hockey, contain significantly more intricate descriptions, a much wider variety of possible actions, and rather ambiguous regulations defining the legality of activities. With the exception of robot soccer, these physical games have not piqued the AI community's interest.
- Games are usually intriguing because they are difficult to solve. Chess, for example, has an average branching factor of around 35, and games frequently stretch to 50 moves per player, therefore the search tree has roughly 35100 or 10154 nodes (despite the search graph having "only" about 1040 unique nodes). As a result, games, like the real world, necessitate the ability to make some sort of decision even when calculating the best option is impossible.
- Inefficiency is also heavily punished in games. Whereas a half-efficient implementation of A search will merely take twice as long to complete, a chess software that is half as efficient in utilizing its available time will almost certainly be beaten to death, all other factors being equal. As a result of this research, a number of intriguing suggestions for making the most use of time have emerged.

Optimal Decision Making in Games

- Let us start with games with two players, whom we'll refer to as MAX and MIN for obvious reasons.
- MAX is the first to move, and then they take turns until the game is finished. At the conclusion of the game, the victorious player receives points, while the loser receives penalties. A game can be formalized as a type of search problem that has the following elements:
  - S0: The initial state of the game, which describes how it is set up at the start.
  - Player (s): Defines which player in a state has the move.
  - Actions (s): Returns a state's set of legal moves.

- Result (s, a): A transition model that defines a move's outcome.
- Terminal-Test (s): A terminal test that returns true if the game is over but false otherwise. Terminal states are those in which the game has come to a conclusion.
- Utility (s, p): A utility function (also known as a payout function or objective function) determines the final numeric value for a game that concludes in the terminal state s for player p.

- The result in chess is a win, a loss, or a draw, with values of +1, 0, or 1/2. Backgammon's payoffs range from 0 to +192, but certain games have a greater range of possible outcomes.
- A zero-sum game is defined (confusingly) as one in which the total reward to all players is the same for each game instance. Chess is a zero-sum game because each game has a payoff of 0 + 1, 1 + 0, or 1/2 + 1/2.
- "Constant-sum" would have been a preferable name, 22 but zero-sum is the usual term and makes sense if each participant is charged 1.
- The game tree for the game is defined by the beginning state, ACTIONS function, and RESULT function—a tree in which the nodes are game states and the edges represent movements.
- The **figure3.2** depicts a portion of the tic-tac-toe game tree (noughts and crosses).
- MAX may make nine different maneuvers from his starting position. The game alternates between MAXs setting an X and MINs placing an O until we reach leaf nodes corresponding to terminal states, such as one player having three in a row or all of the squares being filled.
- The utility value of the terminal state from the perspective of MAX is shown by the number on each leaf node; high values are thought to be beneficial for MAX and bad for MIN
- The game tree for tic-tac-toe is relatively short, with just 9! = 362,880 terminal nodes. However, because there are over 1040 nodes in chess, the game tree is better viewed as a theoretical construct that cannot be realized in the actual world. But, no matter how big the game tree is, MAX's goal is to find a solid move. A tree that is superimposed on the whole game tree and examines enough nodes to allow a player to identify what move to make is referred to as

a search tree.

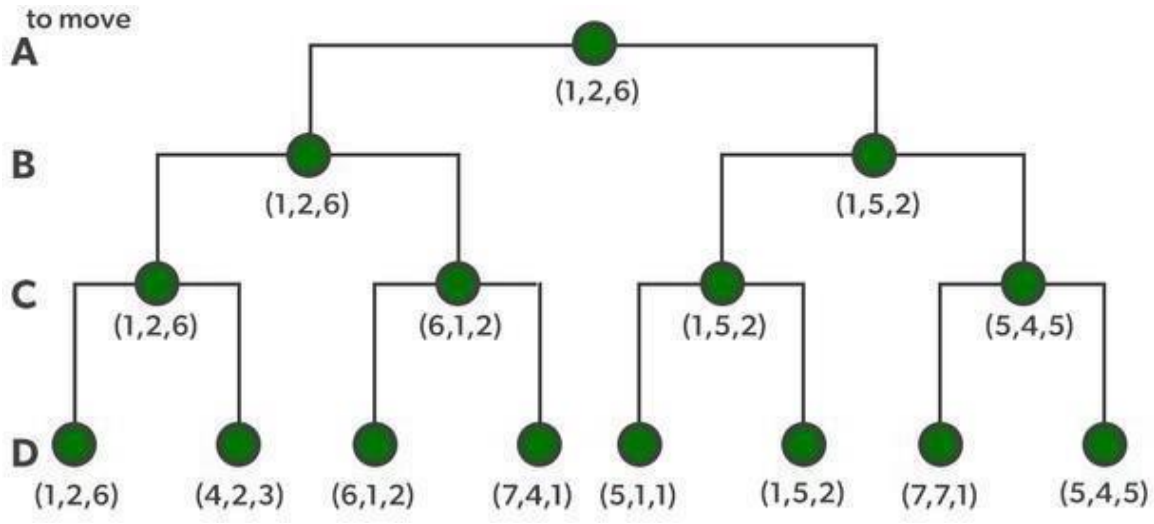


Fig. 3.2 tic-tac-toe game tree

- A sequence of actions leading to a goal state—a terminal state that is a win—would be the best solution in a typical search problem. MIN has something to say about it in an adversarial search.
- MAX must therefore devise a contingent strategy that specifies M A X's initial state move, then MAX's movements in the states resulting from every conceivable MIN response, then MAX's moves in the states resulting from every possible MIN reaction to those moves, and so on. This is quite similar to the AND-OR search method, with MAX acting as OR and MIN acting as AND.
- When playing an infallible opponent, an optimal strategy produces results that are as least as excellent as any other plan. We'll start by demonstrating how to find the best plan.
- We'll move to the trivial game in the figure 3.3 since even a simple game like tic-tac-toe is too complex for us to draw the full game tree on one page. MAX's root node moves are designated by the letters a1, a2, and a3.
- MIN's probable answers to a1 are b1, b2, b3, and so on. This game is over after MAX and MIN each make one move. (In game terms, this tree consists of two half-moves and is one move deep, each of which is referred to as a ply.)
- The terminal states in this game have utility values ranging from 2 to 14.

Game's Utility Function

- The optimal strategy can be found from the minimax value of each node, which we express as MINIMAX, given a game tree (n).
- Assuming that both players play optimally from there through the finish of the game, the utility (for MAX) of being in the corresponding state is the node's minimax value.
- The usefulness of a terminal state is obviously its minimax value. Furthermore, if given the option, MAX prefers to shift to a maximum value state, whereas MIN wants to move to a minimum value state. So here's what we've got:

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

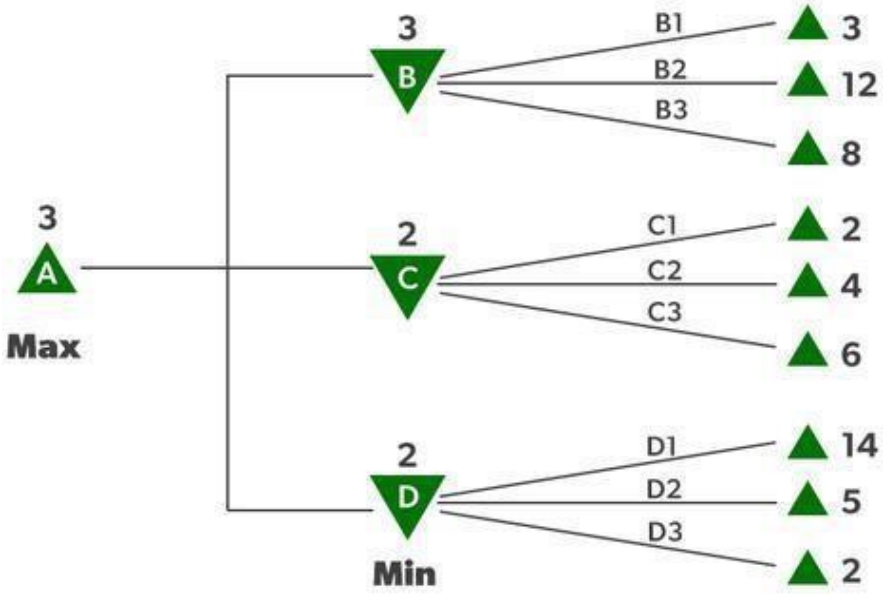


Fig. 3.3 Games Utility Function

- Let's use these definitions to analyze the game tree shown in the figure 3.3. The game's UTILITY function provides utility values to the terminal nodes on the bottom level.
- Because the first MIN node, B, has three successor states with values of 3, 12, and 8, its minimax value is 3. Minimax value 2 is also used by the other two MIN nodes. The root node is a MAX node, with minimax values of 3, 2, and 2, resulting in a minimax value of 3. We can also find the root of the minimax decision: action a1 is the best option for MAX since it leads to the highest minimax value.
- This concept of optimal MAX play requires that MIN plays optimally as well—it maximizes MAX's worst-case outcome. What happens if MIN isn't performing at its

best? Then it's a simple matter of demonstrating that MAX can perform even better. Other strategies may outperform the minimax method against suboptimal opponents, but they will always outperform optimal opponents.

3. Explain in detail about Alpha-Beta Pruning Search.

Describe how the minmax and alpha-beta algorithms change for two player, non zero sum games in which each player has a distinct utility function and both utility functions are known to both players. If there are no constraints on the two terminal utilities, is it possible for any node to be pruned by alpha-beta? What if the players utility functions on any state differ by at most a constant k, making the game almost cooperative? [Apr 2024]

How does Alpha Beta search algorithm differ from Minimax algorithm. Analyse. [Nov 2023]

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- In the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning.**
- This involves two threshold parameter Alpha and beta for future expansion, so it is called alpha-beta pruning. It is also called as **Alpha-Beta Algorithm.**
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    a. Alpha: The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
    b. Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow.

- Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning α>=β

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning

Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.

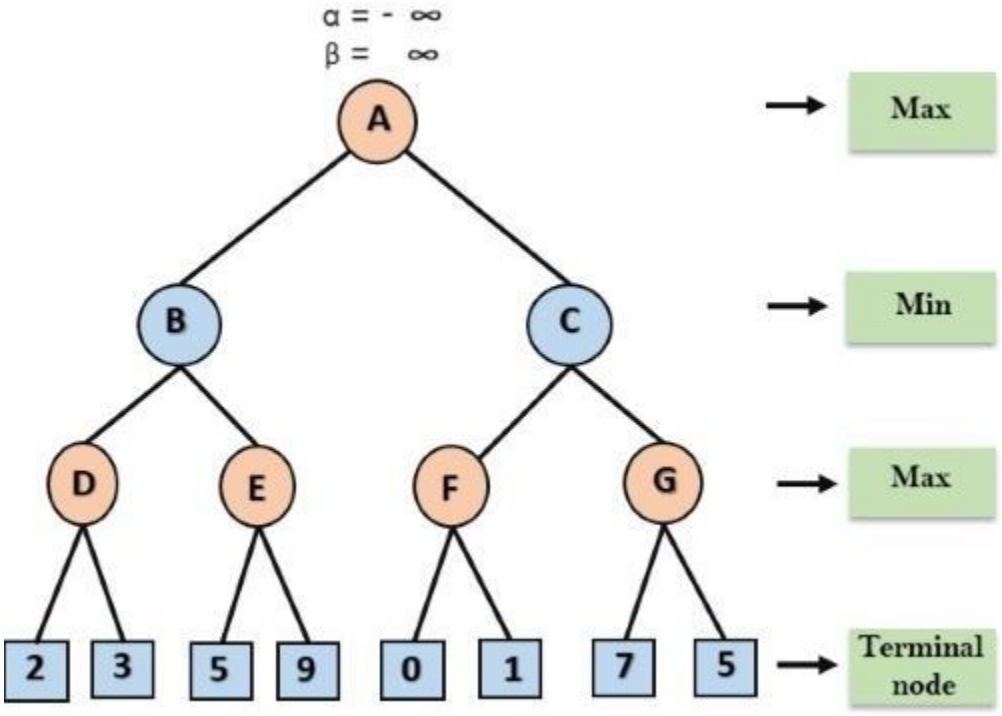


Fig.3.4 Max player –First Move -Alpha-Beta Pruning

Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α

is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.

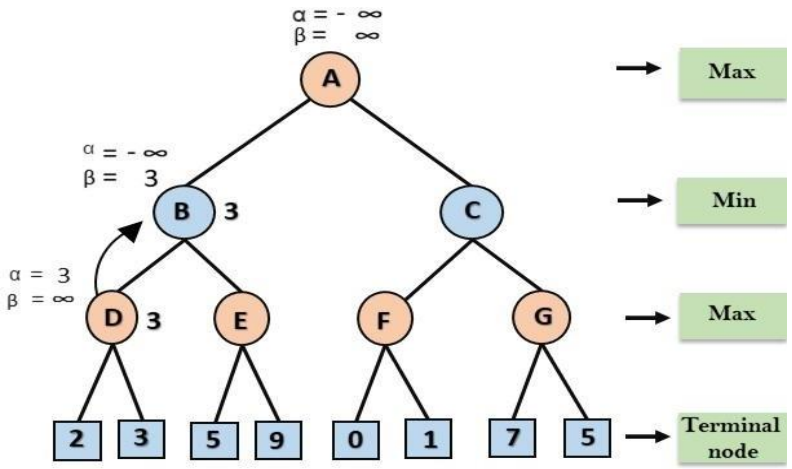


Fig.3.5 Min player – First move -Alpha-Beta Pruning

In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.

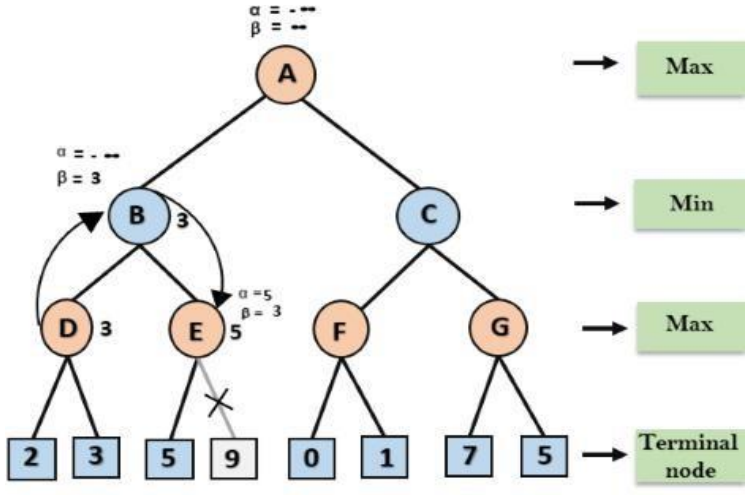


Fig.3.6 Max player –Second Move -Alpha-Beta Pruning

Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-

∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.

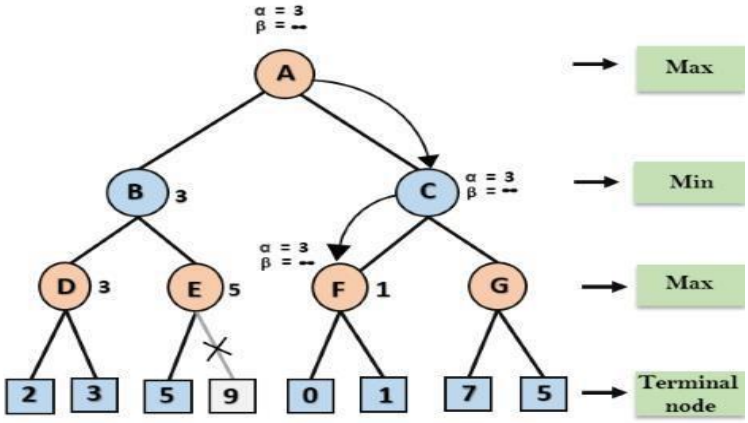


Fig.3.7 Min Player – Second Move -Alpha-Beta Pruning

Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.

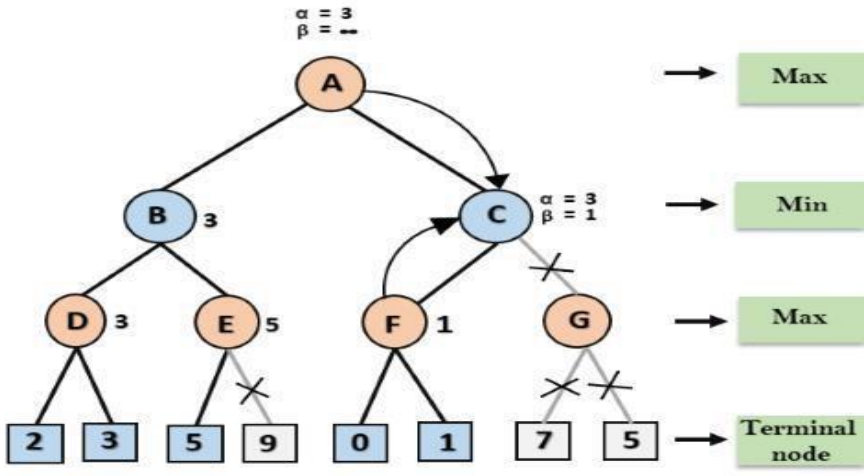


Fig.3.8 Apply Alpha-Beta Pruning Methods

Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which

has never computed. Hence the optimal value for the maximizer is 3 for this example.

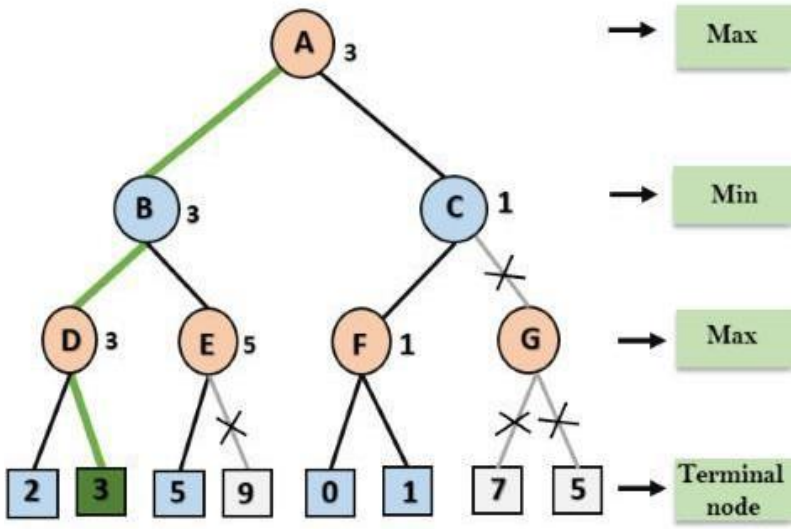


Fig.3.9 Apply Alpha-Beta Pruning Methods

Move Ordering in Alpha-Beta pruning:

- The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

Worst ordering: In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is O(bm).

Ideal ordering: The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is O(bm/2).

Rules to find good ordering:

Following are some rules to find good ordering in alpha-beta pruning:

- Occur the best move from the shallowest node.
- Order the nodes in the tree such that the best nodes are checked first.
- Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.
- We can bookkeep the states, as there is a possibility that states may repeat.

4. Explain in detail about Monte Carlo Tree Search (MCTS).

Write short notes on Monte-Carlo Search. [Nov 2023]

- Monte Carlo Tree Search (MCTS) is a search technique in the field of Artificial Intelligence (AI). It is a probabilistic and heuristic driven search algorithm that combines the classic tree search implementations alongside machine learning principles of reinforcement learning. In tree search, there's always the possibility that the current best action is actually not the most optimal action.
- In such cases, MCTS algorithm becomes useful as it continues to evaluate other alternatives periodically during the learning phase by executing them, instead of the current perceived optimal strategy. This is known as the" exploration-exploitation trade-off ".
- It exploits the actions and strategies that is found to be the best till now but also must continue to explore the local space of alternative decisions and find out if they could replace the current best.
- Exploration helps in exploring and discovering the unexplored parts of the tree, which could result in finding a more optimal path. In other words, we can say that exploration expands the tree's breadth more than its depth. Exploration can be useful to ensure that MCTS is not overlooking any potentially better paths.
- But it quickly becomes inefficient in situations with large number of steps or repetitions. In order to avoid that, it is balanced out by exploitation. Exploitation sticks to a single path that has the greatest estimated value. This is a greedy approach and this will extend the tree's depth more than its breadth.
- In simple words, UCB formula applied to trees helps to balance the exploration-exploitation trade-off by periodically exploring relatively unexplored nodes of the tree and discovering potentially more optimal paths than the one it is currently exploiting. For this characteristic, MCTS becomes particularly useful in making optimal decisions in Artificial Intelligence (AI) problems.

Monte Carlo Tree Search (MCTS) algorithm:

- In MCTS, nodes are the building blocks of the search tree. These nodes are formed based on the outcome of a number of simulations. The process of Monte Carlo Tree Search can be broken down into four distinct steps, viz., selection, expansion, simulation and backpropagation.

Each of these steps is explained in details below:

- **Selection:**
- In this process, the MCTS algorithm traverses the current tree from the root node using a specific strategy. The strategy uses an evaluation function to optimally select nodes with the highest estimated value.
- MCTS uses the Upper Confidence Bound (UCB) formula applied to trees as the strategy in the selection process to traverse the tree. It balances the exploration-exploitation trade-off. During tree traversal, a node is selected based on some parameters that return the maximum value.
- The parameters are characterized by the formula that is typically used for this purpose is given below.

$$S_i = x_i + C\sqrt{\frac{\ln(t)}{n_i}}$$

where;

S_i = value of a node i

x_i = empirical mean of a node i C = a constant

t = total number of simulations

- When traversing a tree during the selection process, the child node that returns the greatest value from the above equation will be one that will get selected. During traversal, once a child node is found which is also a leaf node, the MCTS jumps into the expansion step.

- **Expansion:** In this process, a new child node is added to the tree to that node which was optimally reached during the selection process.
- **Simulation:** In this process, a simulation is performed by choosing moves or strategies until a result or predefined state is achieved.
- **Backpropagation:** After determining the value of the newly added node, the remaining tree must be updated. So, the backpropagation process is performed, where it backpropagates from the new node to the root node. During the process, the number of simulation stored in each node is incremented. Also, if the new node's simulation results in a win, then the number of wins is also incremented.

The above steps can be visually understood by the diagram given below:

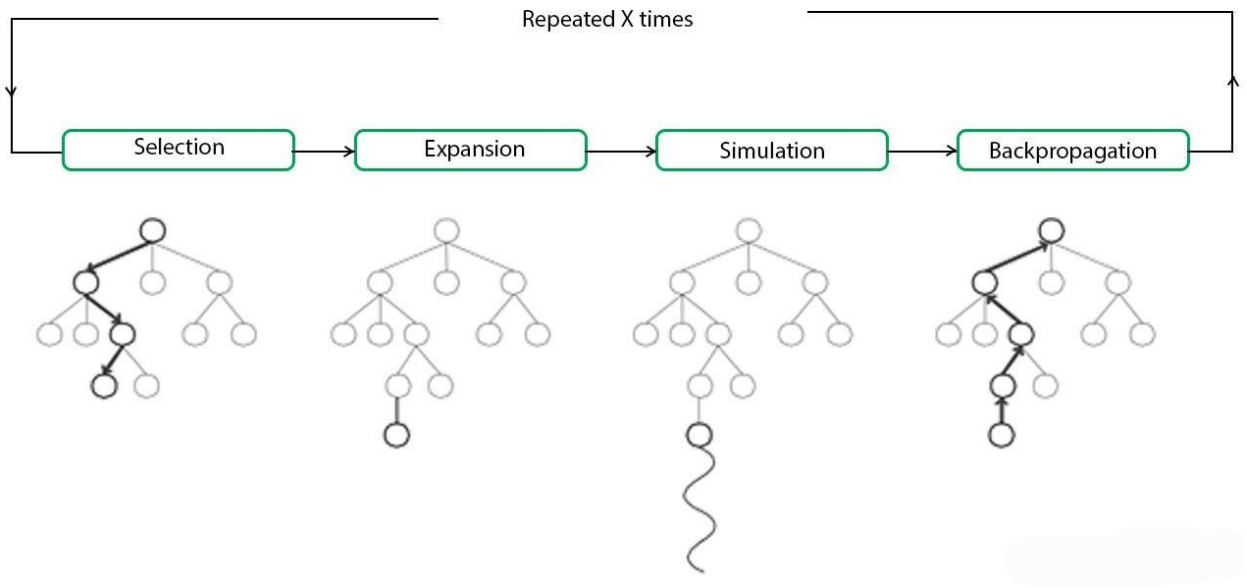


Fig. 3.10 Monte Carlo Tree Search (MCTS) algorithm Steps

- These types of algorithms are particularly useful in turn based games where there is no element of chance in the game mechanics, such as Tic Tac Toe, Connect 4, Checkers, Chess, Go, etc. This has recently been used by Artificial Intelligence Programs like AlphaGo, to play against the world's top Go players. But, its application is not limited to games only. It can be used in any situation which is described by state-action pairs and simulations used to forecast outcomes.
- As we can see, the MCTS algorithm reduces to a very few set of functions which we can use any choice of games or in any optimizing strategy.

Advantages of Monte Carlo Tree Search:

1. MCTS is a simple algorithm to implement.
2. Monte Carlo Tree Search is a heuristic algorithm. MCTS can operate effectively without any knowledge in the particular domain, apart from the rules and end conditions, and can find its own moves and learn from them by playing random playouts.
3. The MCTS can be saved in any intermediate state and that state can be used in future use cases whenever required.
4. MCTS supports asymmetric expansion of the search tree based on the circumstances in which it is operating.

Disadvantages of Monte Carlo Tree Search:

1. As the tree growth becomes rapid after a few iterations, it requires a huge amount of memory.

2. There is a bit of a reliability issue with Monte Carlo Tree Search. In certain scenarios, there might be a single branch or path, that might lead to loss against the opposition when implemented for those turn-based games. This is mainly due to the vast amount of combinations and each of the nodes might not be visited enough number of times to understand its result or outcome in the long run.
3. MCTS algorithm needs a huge number of iterations to be able to effectively decide the most efficient path. So, there is a bit of a speed issue there.

5. Explain in detail about Stochastic games.

- Many unforeseeable external occurrences can place us in unforeseen circumstances in real life. Many games, such as dice tossing, have a random element to reflect this unpredictability. These are known as stochastic games. Backgammon is a classic game that mixes skill and luck.
- The legal moves are determined by rolling dice at the start of each player's turn white, for example, has rolled a 6–5 and has four alternative moves in the backgammon scenario shown in the figure below.

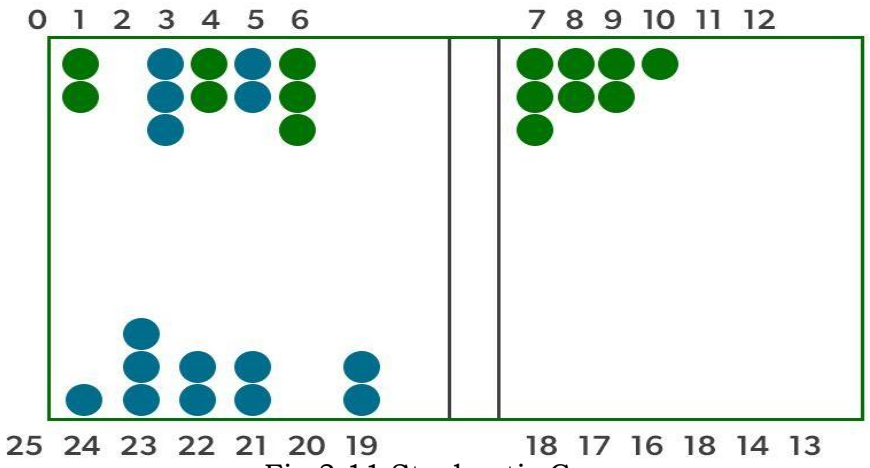


Fig.3.11 Stochastic Games

- This is a standard backgammon position. The object of the game is to get all of one's pieces off the board as quickly as possible. White moves in a clockwise direction toward 25, while Black moves in a counterclockwise direction toward 0.
- Unless there are many opponent pieces, a piece can advance to any position; if there is only one opponent, it is caught and must start over. White has rolled a 6–5 and must pick between four valid moves: (5–10,5–11), (5–11,19– 24), (5–10,10–16), and (5–11,11–16), where the notation (5–11,11–16) denotes moving one piece from position 5 to 11 and then another from 11 to 16. Stochastic game tree

for a backgammon position. White knows his or her own legal moves, but he or she has no idea how Black will roll, and thus has no idea what Black's legal moves will be. That means White won't be able to build a normal game tree-like in chess or tic-tac-toe.

- In backgammon, in addition to M A X and M I N nodes, a game tree must include chance nodes. The figure 3.12 depicts chance nodes as circles.
- The possible dice rolls are indicated by the branches leading from each chance node; each branch is labelled with the roll and its probability.
- There are 36 different ways to roll two dice, each equally likely, yet there are only 21 distinct rolls because a 6–5 is the same as a 5–6.

P (1–1) = 1/36 because each of the six doubles (1–1 through 6–6) has a probability of 1/36. Each of the other 15 rolls has a 1/18 chance of happening.

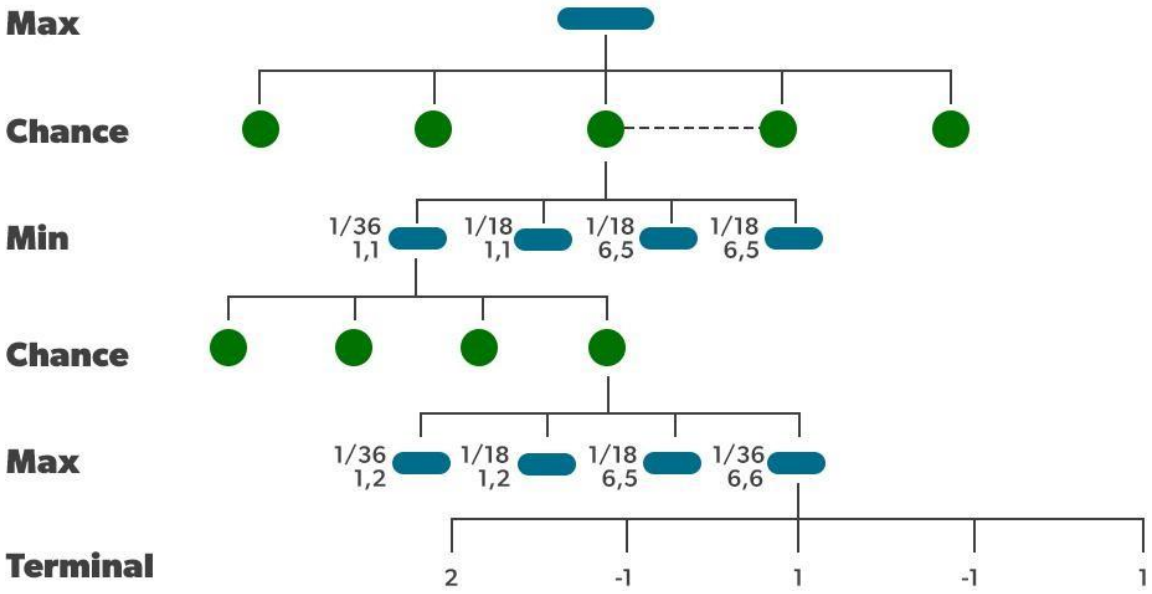


Fig.3.12 Dice possibilities

- The following phase is to learn how to make good decisions. Obviously, we want to choose the move that will put us in the best position. Positions, on the other hand, do not have specific minimum and maximum values. Instead, we can only compute a position's anticipated value, which is the average of all potential outcomes of the chance nodes.
- As a result, we can generalize the deterministic minimax value to an expected-minimax value for games with chance nodes. Terminal nodes, MAX and MIN nodes (for which the dice roll is known), and MAX and MIN nodes (for which the dice roll is unknown) all function as before. We compute the expected value for chance nodes, which is the sum of all outcomes, weighted by the probability of each chance

action.

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\,\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

where r is a possible dice roll (or other random events) and RESULT(s,r) denotes the same state as s, but with the addition that the dice roll's result is r.

6. Explain in detail about Partially observable games.

- A partially observable system is one in which the entire state of the system is not fully visible to an external sensor. In a partially observable system the observer may utilise a memory system in order to add information to the observer's understanding of the system.
- An example of a partially observable system would be a card game in which some of the cards are discarded into a pile face down. In this case the observer is only able to view their own cards and potentially those of the dealer. They are not able to view the face-down (used) cards, nor the cards that will be dealt at some stage in the future. A memory system can be used to remember the previously dealt cards that are now on the used pile. This adds to the total sum of knowledge that the observer can use to make decisions.
- In contrast, a fully observable system would be that of chess. In chess (apart from the 'who is moving next' state, and minor subtleties such as whether a side has castled, which may not be clear) the full state of the system is observable at any point in time.
- Partially observable is a term used in a variety of mathematical settings, including that of artificial intelligence and partially observable Markov decision processes.

7. Explain in detail about Constraint satisfaction problems.

- Techniques like Local search, Adversarial search to solve different problems. The objective of every problem-solving technique is one, i.e., to find a solution to reach the goal.

- Although, in adversarial search and local search, there were no constraints on the agents while solving the problems and reaching to its solutions.
- Constraint satisfaction technique. By the name, it is understood that constraint satisfaction means solving a problem under certain constraints or rules.
- Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem. Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.
- Constraint satisfaction depends on three components, namely:
  - X: It is a set of variables.
  - D: It is a set of domains where the variables reside. There is a specific domain for each variable.
  - C: It is a set of constraints which are followed by the set of variables.
- In constraint satisfaction, domains are the spaces where the variables reside, following the problem specific constraints. These are the three main elements of a constraint satisfaction technique. The constraint value consists of a pair of {scope, rel}. The scope is a tuple of variables which participate in the constraint and rel is a relation which includes a list of values which the variables can take to satisfy the constraints of the problem.

Solving Constraint Satisfaction Problems

The requirements to solve a constraint satisfaction problem (CSP) is:

- A state-space
- The notion of the solution.

A state in state-space is defined by assigning values to some or all variables such as

{X1=v1, X2=v2, and so on…}.

An assignment of values to a variable can be done in three ways:

- Consistent or Legal Assignment: An assignment which does not violate any constraint or rule is called Consistent or legal assignment.
- Complete Assignment: An assignment where every variable is assigned with a value, and the solution to the CSP remains consistent. Such assignment is known as Complete assignment.
- Partial Assignment: An assignment which assigns values to some of the variables

only. Such type of assignments are called Partial assignments.

Types of Domains in CSP

There are following two types of domains which are used by the variables :

- Discrete Domain: It is an infinite domain which can have one state for multiple variables. For example, a start state can be allocated infinite times for each variable.
- Finite Domain: It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

Constraint Types in CSP

With respect to the variables, basically there are following types of constraints:

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value x2 will contain a value which lies between x1 and x3.
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.
- Some special types of solution algorithms are used to solve the following types of constraints:
    - **Linear Constraints**: These type of constraints are commonly used in linear programming where each variable containing an integer value exists in linear form only.
    - **Non-linear Constraints:** These type of constraints are used in non-linear programming where each variable (an integer value) exists in a non-linear form.

Note: A special constraint which works in real-world is known as Preference constraint.

Constraint Propagation

In local state-spaces, the choice is only one, i.e., to search for a solution. But in CSP, we have two choices either:

- We can search for a solution or
- We can perform a special type of inference called constraint propagation.

- Constraint propagation is a special type of inference which helps in reducing the legal number of values for the variables. The idea behind constraint propagation is local consistency.

Define Local Consistency. What are the different types of local consistency? Explain any two. [Nov 2023]

- In local consistency, variables are treated as nodes, and each binary constraint is treated as an arc in the given problem.

There are following local consistencies:

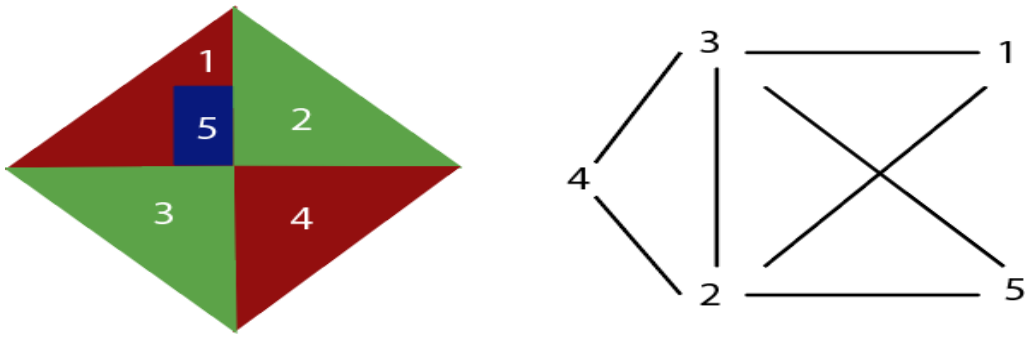
- **Node Consistency:** A single variable is said to be node consistent if all the values in the variable's domain satisfy the unary constraints on the variables.
- **Arc Consistency:** A variable is arc consistent if every value in its domain satisfies the binary constraints of the variables.
- **Path Consistency:** When the evaluation of a set of two variable with respect to a third variable can be extended over another variable, satisfying all the binary constraints. It is similar to arc consistency.
- **k-consistency:** This type of consistency is used to define the notion of stronger forms of propagation. Here, we examine the k-consistency of the variables.

CSP Problems

Constraint satisfaction includes those problems which contains some constraints while solving the problem.

CSP includes the following problems:

- **Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.



Graph Coloring

Fig.3.13 Graph Coloring

- **Sudoku Playing:** The gameplay where the constraint is that no number from 0-9 can be repeated in the same row or column.

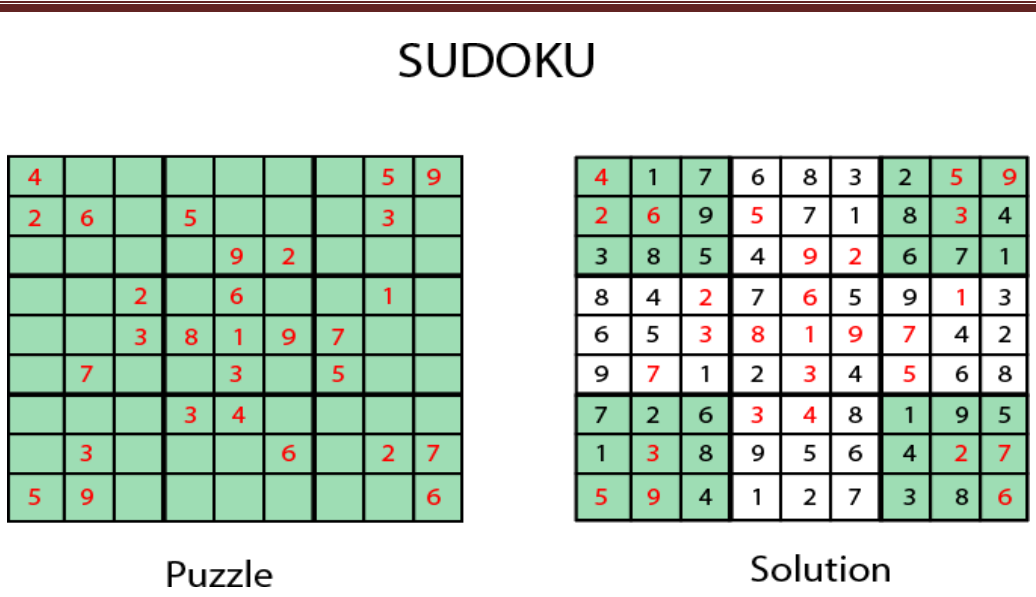


Fig.3.14 Sudoku Game solution

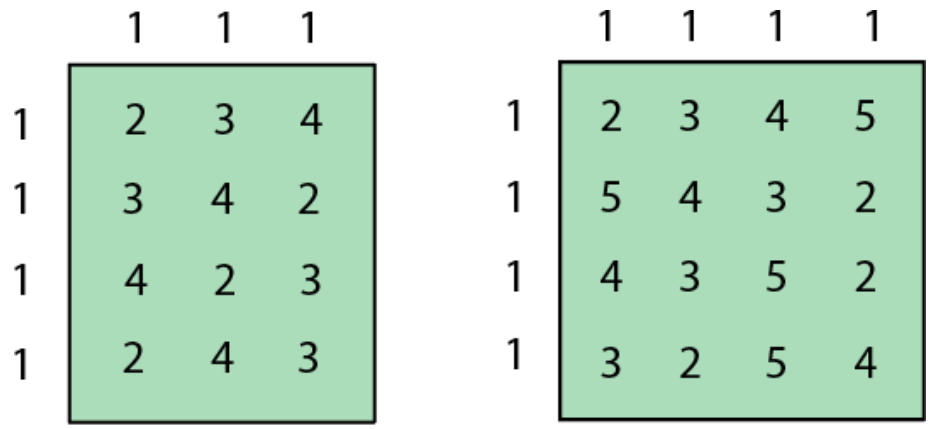
- **n-queen problem:** In n-queen problem, the constraint is that no queen should be placed either diagonally, in the same row or column.

  Note: The n-queen problem is already discussed in Problem-solving in AI section.

- **Crossword:** In crossword problem, the constraint is that there should be the correct formation of the words, and it should be meaningful.

Fig.3.15 Crossword Game

- **Latin square Problem:** In this game, the task is to search the pattern which is occurring several times in the game. They may be shuffled but will contain the same digits.



Latin Squence Problem

Fig.3.16 Latin Sequence problem

Consider the so-called crypt-arithmetic problem shown below

SAVE+

MORE

MONEY

The problem is to figure out the digits (0to9) that should be assigned to these letters so that the indicated arithmetic operation is valid. Your job is to formulate this as a search problem. Clearly indicate a definition of state, initial state, goal state, operators and define a suitable path cost. Estimate the size of the search tree and draw a small segment of the search tree. [Apr 2024]

Cryptarithmetic Problem: This problem has one most important constraint that is, we cannot assign a different digit to the same character. All digits should contain a unique alphabet.

Crypt arithmetic Problem

Cryptarithmetic Problem is a type of constraint satisfaction problem where the game is about digits and its unique replacement either with alphabets or other symbols. In **cryptarithmetic problem,** the digits (0-9) get substituted by some possible alphabets or symbols. The task in cryptarithmetic problem is to substitute each digit with an alphabet to get the result arithmetically correct.

We can perform all the arithmetic operations on a given cryptarithmetic problem. The rules or constraints on a cryptarithmetic problem are as follows:

- There should be a unique digit to be replaced with a unique alphabet.
- The result should satisfy the predefined arithmetic rules, i.e., 2+2 =4, nothing else.
- Digits should be from **0-9** only.
- There should be only one carry forward, while performing the addition operation on a problem.
- The problem can be solved from both sides, i.e., **lefthand side (L.H.S), or righthand side (R.H.S)**

  Let's understand the cryptarithmetic problem as well its constraints better with the help of an example:

- Given a cryptarithmetic problem, i.e., S E N D + M O R E = M O N E Y

```
  SEND
+ MORE
------
 MONEY
------
```

- In this example, add both terms **S E N D** and **M O R E** to bring **M O N E Y** as a result.

Follow the below steps to understand the given problem by breaking it into its subparts:

- Starting from the left hand side (L.H.S) , the terms are S and M. Assign a digit which could give a satisfactory result. Let's assign S->9 and M->1.

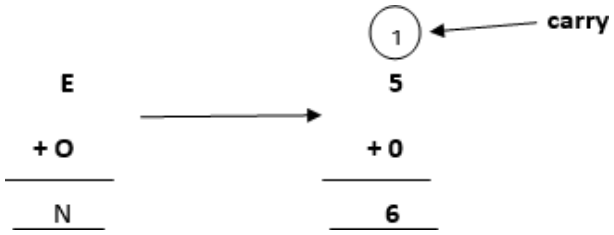
```
  S          9
+ M   ---->  + 1
-----      -----
 MO         10
-----      -----
```

Hence, we get a satisfactory result by adding up the terms and got an assignment for O as O->0 as well.

- Now, move ahead to the next terms E and O to get N as its output.

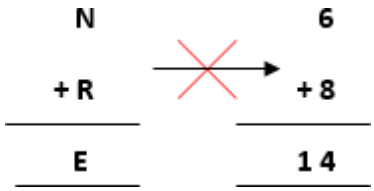
```
  E          5
+ O   ---X->  + 0
-----      -----
 N          5
-----      -----
```

Adding E and O, which means 5+0=0, which is not possible because according to cryptarithmetic constraints, we cannot assign the same digit to two letters. So, we need to think more and assign some other value.



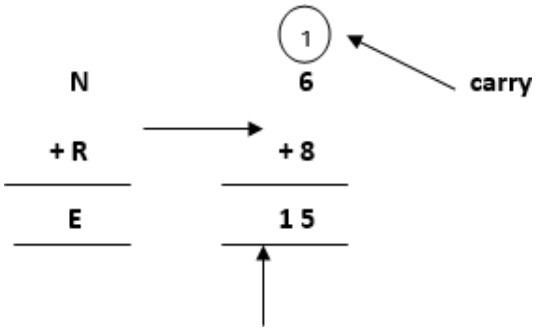
Note: When we will solve further, we will get one carry, so after applying it, the answer will be satisfied.

- Further, adding the next two terms N and R w



But, we have already assigned **E->5**. Thus, the above result does not satisfy the values because we are getting a different value for **E.** So, we need to think more.

Again, after solving the whole problem, we will get a carryover on this term, so our answer will be satisfied.



where 1 will be carry forward to the above term

Let's move ahead.

- Again, on adding the last two terms, i.e., the rightmost terms D and E, we get Y as its result.

where 1 will be carry forward to the above term

- Keeping all the constraints in mind, the final resultant is as follows:

```
  SEND
+MORE
-----
MONEY
-----
```

- Below is the representation of the assignment of the digits to the alphabets.

S	9
E	5
N	6
D	7
M	1
O	0
R	8
y	2

More examples of cryptarithmatic problems can be:

1.

```
  BASE
+BALL
-----
GAMES
-----
```

→

B	7
A	4
S	8
E	3
L	5
G	1
M	9

2.

```
  YOUR
+YOU
-----
HEART
-----
```

→

Y	9
O	4
U	2
R	6
H	1
E	0
A	3
T	8

8. Explain in detail about Backtracking search for CSP.

Backtracking search, a form of depth-first search, is commonly used for solving CSPs. Inference can be interwoven with search.

Commutativity: CSPs are all commutative. A problem is commutative if the order of application of any given set of actions has no effect on the outcome.

Backtracking search: A depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. Backtracking algorithm repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

There is no need to supply BACKTRACKING-SEARCH with a domain-specific initial state, action function, transition model, or goal test.

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
    remove {var = value} and inferences from assignment
  return failure
```

Figure 3.17 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INFERENCE can optionally be used to impose arc-, path-, or *k*-consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are removed from the current assignment and a new value is tried.

BACKTRACKING-SARCH keeps only a single representation of a state and alters that representation rather than creating a new ones.

To solve CSPs efficiently without domain-specific knowledge, address following questions:

1) function SELECT-UNASSIGNED-VARIABLE: which variable should be assigned next?

   function ORDER-DOMAIN-VALUES: in what order should its values be tried?

2) function INFERENCE: what inferences should be performed at each step in the search?

3) When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

1. Variable and value ordering

Select Unassigned Variable Variable selection—fail-first

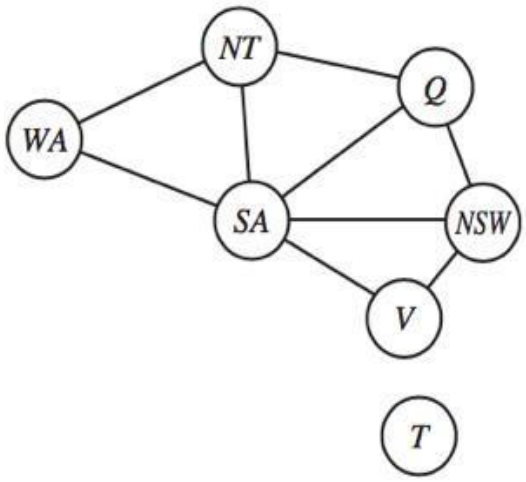


Fig.3.18 Example Backtracking

Minimum-remaining-values (MRV) heuristic:

- The idea of choosing the variable with the fewest "legal" value. A.k.a. "most constrained variable" or "fail-first" heuristic, it picks a variable that is most likely to cause a failure soon thereby pruning the search tree. If some variable X has no legal values left, the MRV heuristic will select X and failure will be detected immediately—avoiding pointless searches through other variables.
- E.g. After the assignment for WA=red and NT=green, there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assigning Q.

[Powerful guide]

Degree heuristic: The degree heuristic attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables. **[useful tie-breaker]**

e.g. SA is the variable with highest degree 5; the other variables have degree 2 or 3; T has degree 0.

ORDER-DOMAIN-VALUES

Value selection—fail-last

If we are trying to find all the solution to a problem (not just the first one), then the ordering does not matter.

Least-constraining-value heuristic: prefers the value that rules out the fewest choice for the neighboring variables in the constraint graph. **(Try to leave the maximum flexibility for subsequent variable assignments.)**

e.g. We have generated the partial assignment with WA=red and NT=green and that our next choice is for Q. Blue would be a bad choice because it eliminates the last legal value left for Q's neighbor, SA, therefore prefers red to blue.

2. Interleaving search and inference INFERENCE

Forward checking: [One of the simplest forms of inference.]

Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it: for each unassigned variable Y that is connected to X by a constraint, delete from Y's domain any value that is inconsistent with the value chosen for X.

There is no reason to do forward checking if we have already done arc consistency as a preprocessing step.

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	(R)	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	(R)	B	(G)	R B	R G B	B	R G B
After *V=blue*	(R)	B	(G)	R	(B)		R G B

Figure 3.19 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$ is assigned, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$ is assigned, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Advantage: For many problems the search will be more effective if we combine the MRV heuristic with forward checking.

Disadvantage: Forward checking only makes the current variable arc-consistent, but doesn't look ahead and make all the other variables arc-consistent.

MAC (Maintaining Arc Consistency) algorithm:

[More powerful than forward checking, detect this inconsistency.] After a variable Xi is assigned a value, the INFERENCE procedure calls AC-3, but instead of a queue of all arcs in the CSP, we start with only the arcs(Xj, Xi) for all Xj that are unassigned variables that are neighbors of Xi. From there, AC-3 does constraint propagation in the usual way, and if any variable has its domain reduced to the empty set, the call to AC-3 fails and we know

to backtrack immediately.

Intelligent backtracking

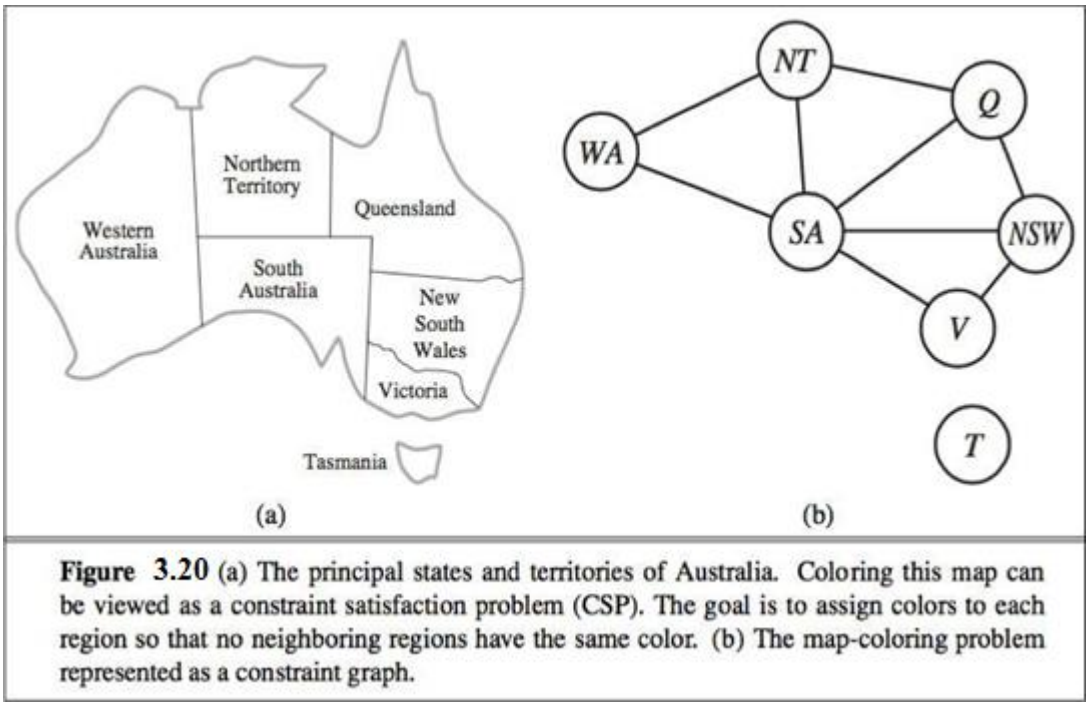


Figure 3.20 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

Chronological backtracking: The BACKGRACKING-SEARCH in Fig 6.5. When a branch of the search fails, back up to the preceding variable and try a different value for it. (The most recent decision point is revisited.)

e.g. Suppose we have generated the partial assignment {Q=red, NSW=green, V=blue, T=red}. When we try the next variable SA, we see every value violates a constraint.

We back up to T and try a new color, it cannot resolve the problem.

Intelligent backtracking: Backtrack to a variable that was responsible for making one of the possible values of the next variable (e.g. SA) impossible.

Conflict set for a variable: A set of assignments that are in conflict with some value for that variable.

(e.g. The set {Q=red, NSW=green, V=blue} is the conflict set for SA.)

backjumping method: Backtracks to the most recent assignment in the conflict set. (e.g. backjumping would jump over T and try a new value for V.)

Forward checking can supply the conflict set with no extra work.

Whenever forward checking based on an assignment X=x deletes a value from Y's domain, add X=x to Y's conflict set;

If the last value is deleted from Y's domain, the assignment in the conflict set of Y are added to the conflict set of X.

In fact, every branch pruned by backjumping is also pruned by forward checking. Hence simple backjumping is redundant in a forward-checking search or in a search that uses stronger consistency checking (such as MAC).

Conflict-directed backjumping:

- e.g. Consider the partial assignment which is proved to be inconsistent: {WA=red, NSW=red}. We try T=red next and then assign NT, Q, V, SA, no assignment can work for these last 4 variables.
- Eventually we run out of value to try at NT, but simple backjumping cannot work because NT doesn't have a complete conflict set of preceding variables that caused to fail.
- The set {WA, NSW} is a deeper notion of the conflict set for NT, caused NT together with any subsequent variables to have no consistent solution. So the algorithm should backtrack to NSW and skip over T.
- A backjumping algorithm that uses conflict sets defined in this way is called conflict-direct backjumping.

How to Compute:

- When a variable's domain becomes empty, the "terminal" failure occurs, that variable has a standard conflict set.
- Let Xj be the current variable, let conf(Xj) be its conflict set. If every possible value for Xj fails, backjump to the most recent variable Xi in conf(Xj), and set
- conf(Xi) ← conf(Xi)∪ conf(Xj) – {Xi}.
- The conflict set for an variable means, there is no solution from that variable onward, given the preceding assignment to the conflict set.
- e.g. assign WA, NSW, T, NT, Q, V, SA.
- SA fails, and its conflict set is {WA, NT, Q}. (standard conflict set)
- Backjump to Q, its conflict set is {NT, NSW}∪{WA,NT,Q}-{Q} = {WA, NT, NSW}.

Backtrack to NT, its conflict set is {WA}∪{WA,NT,NSW}-{NT} = {WA, NSW}.

- Hence the algorithm backjump to NSW. (over T)
- After backjumping from a contradiction, how to avoid running into the same problem again:

Constraint learning:

- The idea of finding a minimum set of variables from the conflict set that causes the problem. This set of variables, along with their corresponding values, is called a **no-good**. We then record the no-good, either by adding a new constraint to the CSP or by keeping a separate cache of no-goods.
- Backtracking occurs when no legal assignment can be found for a variable. **Conflict-directed backjumping** backtracks directly to the source of the problem.

8. Explain in detail about Local search for CSP.

Local search algorithms for CSPs use a complete-state formulation: the initial state assigns a value to every variable, and the search change the value of one variable at a time.

The min-conflicts heuristic:

In choosing a new value for a variable, select the value that results in the minimum number of conflicts with other variables.

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen conflicted variable from csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
  return failure
```

Figure 3.22 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

Local search techniques in Section 4.1 can be used in local search for CSPs.

The landscape of a CSP under the mini-conflicts heuristic usually has a series of plateau. Simulated annealing and Plateau search (i.e. allowing sideways moves to another state with the same score) can help local search find its way off the plateau. This wandering on the plateau can be directed with **tabu search**: keeping a small list of recently visited states and forbidding the algorithm to return to those tates.

Constraint weighting:

- A technique that can help concentrate the search on the important constraints.
- Each constraint is given a numeric weight Wi, initially all 1.
- At each step, the algorithm chooses a variable/value pair to change that will result in the lowest total weight of all violated constraints.
- The weights are then adjusted by incrementing the weight of each constraint that is violated by the current assignment.
- Local search can be used in an online setting when the problem changes, this is particularly important in scheduling problems.

The structure of problem

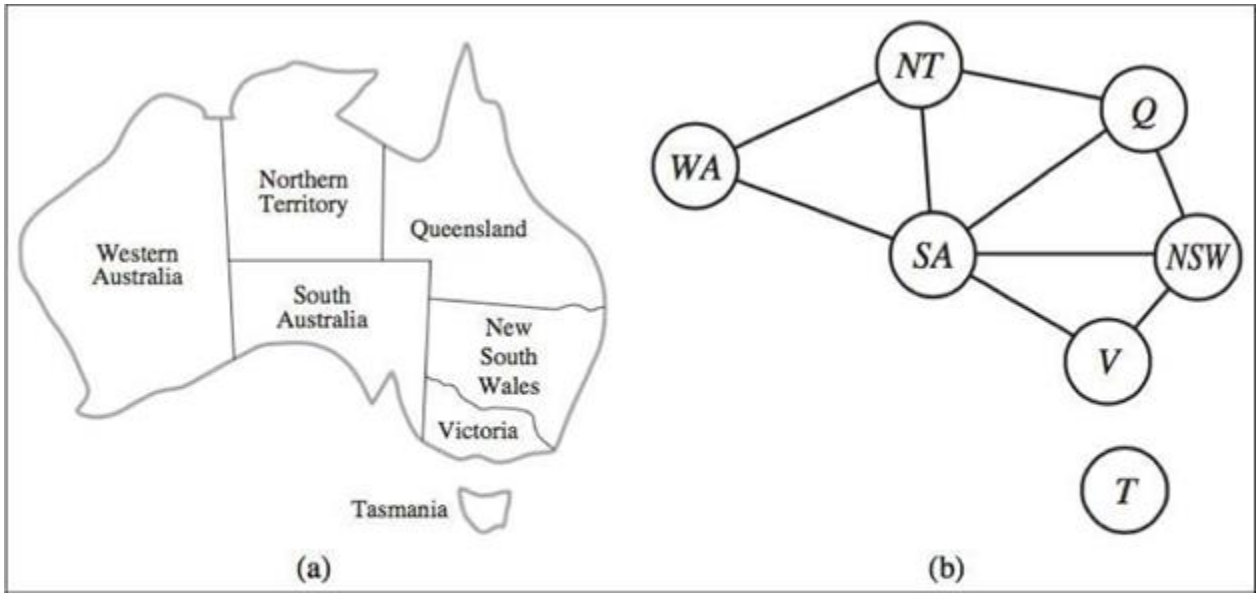


Figure 3.23 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

1. The structure of constraint graph

- The structure of the problem as represented by the constraint graph can be used to find solution quickly.
- e.g. The problem can be decomposed into 2 **independent subproblems**: Coloring T and coloring the mainland.

- **Tree:** A constraint graph is a tree when any two varyiable are connected by only one path.

Directed arc consistency (DAC):

- A CSP is defined to be directed arc-consistent under an ordering of variables X1, X2, … , Xn if and only if every Xi is arc-consistent with each Xj for j>i. By using DAC, any tree-structured CSP can be solved in time linear in the number of variables.

How to solve a tree-structure CSP:

- Pick any variable to be the root of the tree;
- Choose an ordering of the variable such that each variable appears after its parent in the tree. (topological sort)
- Any tree with n nodes has n-1 arcs, so we can make this graph directed arc-consistent in O(n) steps, each of which must compare up to d possible domain values for 2 variables, for a total time of O(nd2).
- Once we have a directed arc-consistent graph, we can just march down the list of variables and choose any remaining value.
- Since each link from a parent to its child is arc consistent, we won't have to backtrack, and can move linearly through the variables.

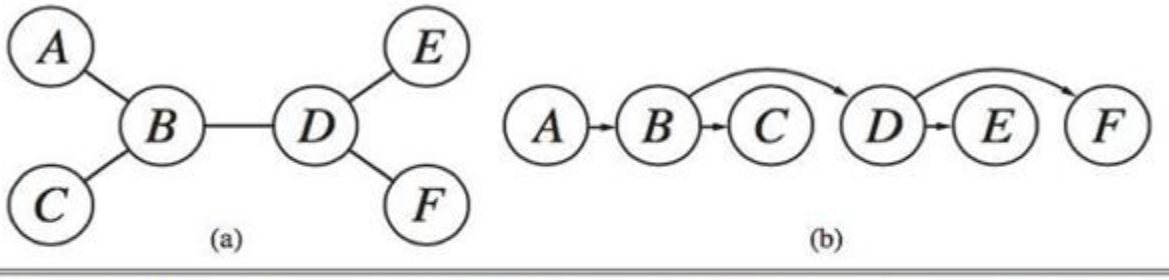


Figure 3.24 (a) The constraint graph of a tree-structured CSP. (b) A linear ordering of the variables consistent with the tree with A as the root. This is known as a **topological sort** of the variables.

function TREE-CSP-SOLVER(*csp*) **returns** a solution, or failure
inputs: *csp*, a CSP with components X, D, C

$n \leftarrow$ number of variables in X
assignment $\leftarrow$ an empty assignment
root $\leftarrow$ any variable in X
$X \leftarrow$ TOPOLOGICALSORT(X, *root*)
for $j = n$ **down to** 2 **do**
  MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
  if it cannot be made consistent **then return** *failure*
for $i = 1$ **to** n **do**
  assignment[X_i] $\leftarrow$ any consistent value from D_i
  if there is no consistent value **then return** *failure*
return *assignment*

Figure 3.25 The TREE-CSP-SOLVER algorithm for solving tree-structured CSPs. If the CSP has a solution, we will find it in linear time; if not, we will detect a contradiction.

There are 2 primary ways to reduce more general constraint graphs to trees:

1. Based on removing nodes;

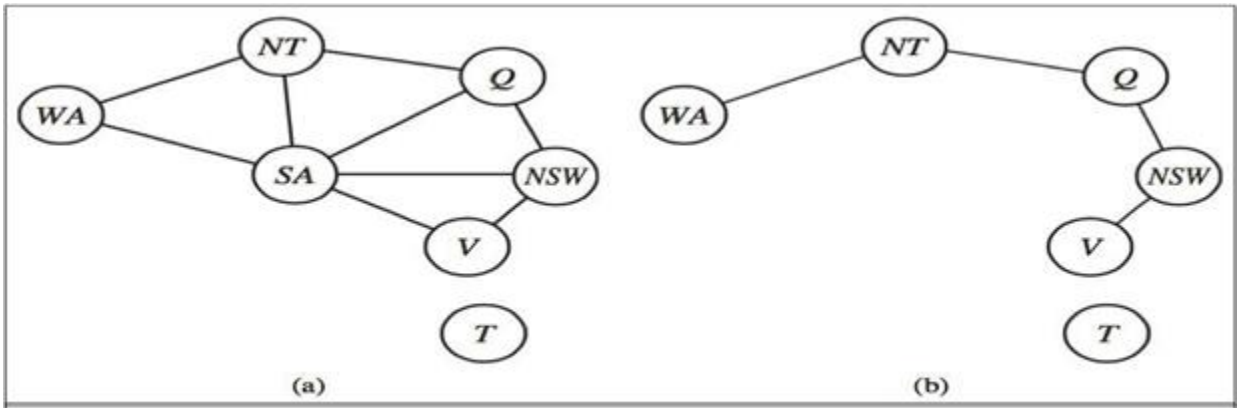


Figure 3.26 (a) The original constraint graph from Figure 6.1. (b) The constraint graph after the removal of *SA*.

e.g. We can delete SA from the graph by fixing a value for SA and deleting from the domains of other variables any values that are inconsistent with the value chosen for SA.

The general algorithm:

Choose a subset S of the CSP's variables such that the constraint graph becomes a tree after removal of S. S is called a **cycle cutset**.

For each possible assignment to the variables in S that satisfies all constraints on S,

(a) remove from the domain of the remaining variables any values that are inconsistent with the assignment for S, and

(b) If the remaining CSP has a solution, return it together with the assignment for S.

Time complexity: O(dc·(n-c)d2), c is the size of the cycle cut set.

Cutset conditioning: The overall algorithmic approach of efficient approximation algorithms to find the smallest cycle cutset.

2. Based on collapsing nodes together

Tree decomposition: construct a **tree decomposition** of the constraint graph into a set of connected subproblems, each subproblem is solved independently, and the resulting solutions are then combined.

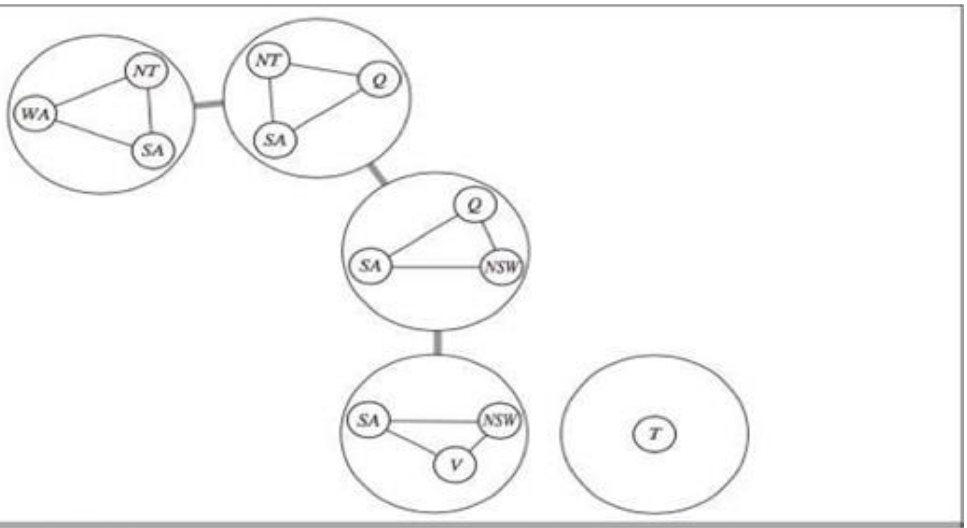


Figure 3.27 A tree decomposition of the constraint graph in Figure 6.12(a).

A tree decomposition must satisfy 3 requirements:

1. Every variable in the original problem appears in at least one of the subproblems.
2. If 2 variables are connected by a constraint in the original problem, they must

appear together (along with the constraint) in at least one of the subproblems.

3. If a variable appears in 2 subproblems in the tree, it must appear in every subproblem along the path connecting those those subproblems.

- We solve each subproblem independently.
- If any one has no solution, the entire problem has no solution.
- If we can solve all the subproblems, then construct a global solution as follows:
- First, view each subproblem as a "mega-variable" whose domain is the set of all solutions for the subproblem.
- Then, solve the constraints connecting the subproblems using the efficient algorithm for trees.

A given constraint graph admits many tree decomposition;

In choosing a decomposition, the aim is to make the subproblems as small as possible.

Tree width:

The tree width of a tree decomposition of a graph is one less than the size of the largest subproblems.

The tree width of the graph itself is the minimum tree width among all its tree decompositions. Time complexity: O(ndw+1), w is the tree width of the graph.

2. The structure in the values of variables

- By introducing a **symmetry-breaking constraint**, we can break the **value symmetry** and reduce the search space by a factor of n!.
- e.g. Consider the map-coloring problems with n colors, for every consistent solution, there is actually a set of n! solutions formed by permuting the color names.(value symmetry)
- On the Australia map, WA, NT and SA must all have different colors, so there are 3!=6 ways to assign.
- We can impose an arbitrary ordering constraint NT<SA<WA that requires the 3 values to be in alphabetical order. This constraint ensures that only one of the n! solution is possible: {NT=blue, SA=green, WA=red}. (symmetry-breaking constraint)

9. Explain the constraint satisfaction problem and the variations on constraint satisfaction problem with example. [Nov 2023]

- Finding a solution that meets a set of constraints is the goal of constraint satisfaction problems (CSPs), a type of AI issue. Finding values for a group of variables that fulfill a set of restrictions or rules is the aim of constraint satisfaction problems. For tasks including resource allocation, planning, scheduling, and decision-making, CSPs are frequently employed in AI.

There are mainly three basic components in the constraint satisfaction problem:

- Variables: The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints. Boolean, integer, and categorical variables are just a few examples of the various types of variables, for instance, could stand in for the many puzzle cells that need to be filled with numbers in a sudoku puzzle.

- Domains: The range of potential values that a variable can have is represented by domains. Depending on the issue, a domain may be finite or limitless. For instance, in Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable representing a problem cell.
- Constraints: The guidelines that control how variables relate to one another are known as constraints. Constraints in a CSP define the ranges of possible values for variables. Unary constraints, binary constraints, and higher-order constraints are only a few examples of the various sorts of constraints. For instance, in a sudoku problem, the restrictions might be that each row, column, and 3×3 box can only have one instance of each number from 1 to 9.

Constraint Satisfaction Problems (CSP) representation:

- The finite set of variables V1, V2, V3 ……Vn.
- Non-empty domain for every single variable D1, D2, D3 ……Dn.
- The finite set of constraints C1, C2 ……, Cm.
- where each constraint Ci restricts the possible values for variables,
- e.g., V1 ≠ V2
- Each constraint Ci is a pair <scope, relation>

- Example: <(V1, V2), V1 not equal to V2>
- Scope = set of variables that participate in constraint.
- Relation = list of valid variable value combinations.
- There might be a clear list of permitted combinations. Perhaps a relation that is abstract and that allows for membership testing and listing.

Constraint Satisfaction Problems (CSP) algorithms:

- The backtracking algorithm is a depth-first search algorithm that methodically investigates the search space of potential solutions up until a solution is discovered that satisfies all the restrictions. The method begins by choosing a variable and giving it a value before repeatedly attempting to give values to the other variables. The method returns to the prior variable and tries a different value if at any time a variable cannot be given a value that fulfills the requirements. Once all assignments have been tried or a solution that satisfies all constraints has been discovered, the algorithm ends.
- The forward-checking algorithm is a variation of the backtracking algorithm that condenses the search space using a type of local consistency. For each unassigned variable, the method keeps a list of remaining values and applies local constraints to eliminate inconsistent values from these sets. The algorithm examines a variable's neighbors after it is given a value to see whether any of its remaining values become inconsistent and removes them from the sets if they do. The algorithm goes backward if, after forward checking, a variable has no more values.
- Algorithms for propagating constraints are a class that uses local consistency and inference to condense the search space. These algorithms operate by propagating restrictions between variables and removing inconsistent values from the variable domains using the information obtained.

Real-world Constraint Satisfaction Problems (CSP):

- Scheduling: A fundamental CSP problem is how to efficiently and effectively schedule resources like personnel, equipment, and facilities. The constraints in this domain specify the availability and capacity of each resource, whereas the variables indicate the time slots or resources.
- Vehicle routing: Another example of a CSP problem is the issue of minimizing travel time or distance by optimizing a fleet of vehicles' routes. In this domain, the

constraints specify each vehicle's capacity, delivery locations, and time windows, while the variables indicate the routes taken by the vehicles.

- Assignment: Another typical CSP issue is how to optimally assign assignments or jobs to humans or machines. In this field, the variables stand in for the tasks, while the constraints specify the knowledge, capacity, and workload of each person or machine.
- Sudoku: The well-known puzzle game Sudoku can be modeled as a CSP problem, where the variables stand in for the grid's cells and the constraints specify the game's rules, such as prohibiting the repetition of the same number in a row, column, or area.
- Constraint-based image segmentation: The segmentation of an image into areas with various qualities (such as color, texture, or shape) can be treated as a CSP issue in computer vision, where the variables represent the regions and the constraints specify how similar or unlike neighboring regions are to one another.

Constraint Satisfaction Problems (CSP) benefits:

- conventional representation patterns
- generic successor and goal functions
- Standard heuristics (no domain-specific expertise).

MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by Naac with A Grade & TCS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

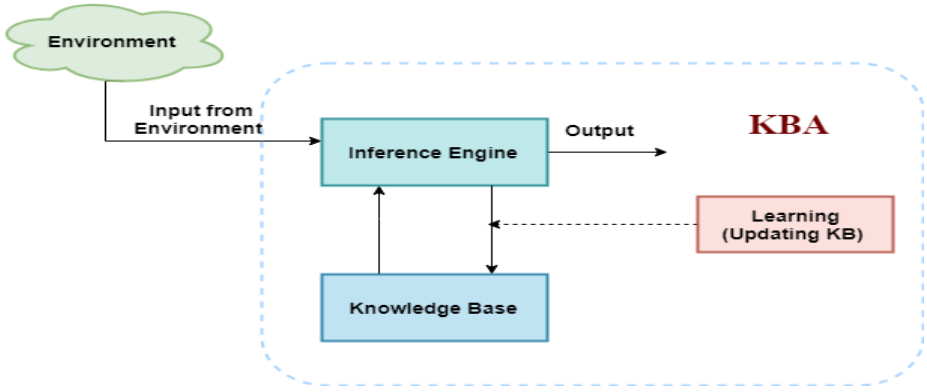
II YEAR / III SEM -AL3391-ARTIFICIAL INTELLIGENCE

UNIT IV LOGICAL REASONING

Knowledge-based agents – propositional logic – propositional theorem proving – propositional model checking – agents based on propositional logic. First-order logic – syntax and semantics – knowledge representation and engineering – inferences in first-order logic

PART-A

1. **Define a Knowledge-based agents.**
   - 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.
2. **List out the parts of Knowledge-based agents.**
   - Knowledge-based agents are composed of two main parts:
     - Knowledge-base and
     - Inference system.
3. **Give the structure of an architecture of knowledge-based agent.**



4. Define an 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.

5. List out the rules of Inference system.

- An inference system works mainly in two rules which are given as:
    - Forward chaining.
    - Backward chaining.

6. What are the Operations Performed by KBA?

- There are three operations which are performed by KBA in order to show the intelligent behaviour.

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.

7. Write a function for the knowledge-based agents program.

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
```

8. List the operations performed by knowledge-based agent.

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.

9. List the Various levels of knowledge-based agent.

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

1. Knowledge level
2. Logical level
3. Implementation level

10. Define a 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.

11. Define a 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 behaviour.
- 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.

12. Define an 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.

13. List out the approaches to designing a knowledge-based agent.

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

1. Declarative approach.
2. Procedural approach.

14. Define a 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.

15. Define an 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.

16. What is Propositional logic and give some examples?

- 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:

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

17. What are the Basic facts about propositional logic?

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- Propositional logic consists of an object, relations or function, and logical connectives.
- 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.
- 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.

18. Why does Knowledge require a different representation model than the data?

- Knowledge representation is not just storing data into some database ,but it also enables an intelligent machine to learn from that knowledge and experience so that it can behave intelligently like human.

19. In which situation do we use the concept Existential instantiation?

- The rule of Existential Instantiation replaces an existentially quantified variable with a single *new constant symbol*.
- The formal statement is as follows: for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,

$$\frac{\exists v \ \alpha}{SUBST(\{v/k\}, \alpha)}$$

For example, from the sentence

- $\exists x \ Crown(x) \wedge OnHead(x, John)$
- we can infer the sentence
- $Crown(C1) \wedge OnHead(C1, John)$
- As long as $C1$ does not appear elsewhere in the knowledge base. Basically, the existential sentence says there is some object satisfying a condition, and applying the existential instantiation rule just gives a name to that object.

20. P->Q->¬PVQ Construct a truth table to show that this equivalence holds (MAY 2014).

¬P	Q	¬PVQ
T	F	T
T	T	T
F	F	F
F	T	T

21. Define the first order definite clause. (NOV 2013).

- **First-order definite clauses** closely resemble propositional definite clauses, they are disjunctions of literals of which *exactly one is positive.*
- *A* definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. The following are first-order definite clauses:

King (x) A Greedy (x)

Evil (x)

Kzng(John)

Greedy(y)

22. Represent the following sentence in predicate from "All the children likes sweet".

∀x Likes (children, sweet)

23. List the Procedure for resolution.

- Convert given proposition in to clausal form
- Convert the negation of the sentence to be proved into clausal form
- Combine the clauses into a set
- Iteratively apply resolution to the set and add the resolvent to the set
- Continue until no further resolvents can be obtained or a null clause is obtained.

24. Differentiate the forward and backward chaining. (APR 2021) (NOV 2022).

Forward Chaining	Backward Chaining
Planning, Monitoring, control	Diagnosis
Data-driven	Goal-driven(hypothesis)
Bottom-up processing	Top-down processing
Find possible Conclusions supported by given facts	Find facts that support a given hypothesis
Similar to breadth-first search	Similar depth first search
CLIPS	PROLOG

25. What are the techniques used to represent knowledge?

- Tables
- Rules
- Semantic Networks
- Frames
- Conceptual Dependency
- Scripts

26. What are the agents of Propositional Logic?

Two kinds of agents:

- Inference-based agents: Use inference algorithms to keep track of the world and deduce hidden properties.
- Circuit-based agents: represent propositions as bits in registers and update them

using signal propagation in logical circuits.

27. What is forward chaining?

- A deduction to reach a conclusion from a set of antecedents is called forward chaining.
- In other words, the system starts from a set of facts, and a set of rules, and tries to find the way of using these rules and facts to deduce a conclusion or come up with a suitable course of action.

28. What is backward chaining?

- In backward chaining, we start from a conclusion, which is the hypothesis we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the data base.

29. What are the types of Quantifiers?

The types of Quantifiers are,

- Universal Quantifiers
- Existential Quantifiers

30. What is Existential quantification?

- (x)P(x) means that P holds for some value of x in the domain associated with that variable b.

  E.g., (x) mammal(x) ^ lays-eggs(x)

- Permits one to make a statement about some object without naming it Connections between All and Exists.

31. What is Universal Quantification?

Universal quantification

(x) P(x) means that P holds for all values of x in the domain associated with that variable b.

E.g., (x) dolphin(x) => mammal(x)

32. What is resolution /refutation?

- Resolution is a procedure used in proving that arguments which are expressible in predicate logic are correct**.**
- Resolution is a procedure that produces proofs by refutation or contradiction.
- Resolution lead to refute a theorem-proving technique for sentences in propositional logic and first order logic.

- Resolution is a rule of inference.
- Resolution is computerized theorem prover.
- -Resolution is so far only defined for propositional logic. The strategy is that the Resolution techniques of propositional logic be adopted in predicate logic.

33. Define OR-Introduction rule in propositonal logic.

- OR-Introduction rule states that from, a sentence, we can infer its disjunction with anything.

34. List some of the rules of inference.

The sound rules are:

- Modus ponens (or implication-elimination) - From an implication and the premise of the implication you can infer the conclusion. If there is an axiom E F and an axiom E, then F follows logically.
- Modus tolens - If there is an axiom E F and an axiom F, then E follows logically.
- Resolution - If there is an axiom E F and an axiom F G then E G follows logically. In fact, resolution can subsume both modus ponens and modus tolens.

  It can also be generalized so that there can be any number of disjuncts in either of the two resolving expressions, including just one. The only requirement is that one expression contains the negation of one disjunct from the other.

35. Define AND-Introduction rule in propositional logic.

- AND-Introduction rule states that from a list of sentences we can infer their conjunctions.

36. Define AND –Elimination rule in propositional logic.

- AND elimination rule states that from a given conjunction it is possible to inference any of the conjuncts.

$$\frac{\alpha \wedge \beta}{\alpha}$$

37. What are the basic Components of propositional logic?

The basic Components of propositional logic

- Logical Constants (True, False)

- Propositional symbols (P, Q)
- Logical Connectives (^,=,∨ , ¬)

38. Define a Proof.

- A sequence of application of inference rules is called a proof.
- Finding proof is exactly finding solution to search problems.
- If the successor function is defined to generate all possible applications of inference rules then the search algorithms can be applied to find proofs.

39. Write the generalized modus ponens rule (MAY 2014)

- Modus ponens (or implication-elimination) - From an implication and the premise of the implication you can infer the conclusion. If there is an axiom E F and an axiom E, then F follows logically.

$$\frac{\alpha \Rightarrow \beta, \qquad \alpha}{\beta}$$

39. Define Validity of a sentence.

- A sentence is valid or necessarily true if and only if it is true under all possible interpretation in all posssible world.

40. Define Satisfiability of a sentence.

- A sentence is satisfiable if and only if there is some interpretation in some world for which it is true.

41. What are nested quantifiers?

- Two quantifiers are nested if one is within the scope of the other. Example-1: ∀x ∃y (x+y=5) Here '∃' (read as-there exists) and '∀' (read as-for all) are quantifiers for variables x and y.

42. Differentiate prepositional & predicate logic. (NOV 2011) (NOV 2021)

- **Propositional Logic** is a tool for reasoning. It provide basic concepts used in many computer science fields and also used in many medical applications. The components are:
  - Proposition Basic
  - Operators logic

- Truth table
- Boolean Algebra

- **Predicate Logic** provides a formalism for performing this analysis of prepositions and additional methods for reasoning with quantified expressions. The term predicate logic derives from the fact that we analyze prepositions into predicate - argument compositions.

43. What are the three levels in describing knowledge based agent? [Nov 2023]

The three levels in describing knowledge based agent

- 1. Logical level;
- 2. Implementation level;
- 3. Knowledge level or epistemological level.

44. What is Logic?

Logic is one which consist of

- A formal system for describing states of affairs, consisting of Syntax b) Semantics;
- Proof Theory – a set of rules for deducing the entailment of set sentences.

45. Explain the connection between ∀ and ∃

- "Everyone likes icecream " is equivalent "there is no one who does not like ice cream" This can be expressed as:

  ∀x Likes(x,IceCream) is equivalent to

  ∃ Likes(x,IceCream)

46. State Unification in first order logic. Nov 2023

- Unification in First-Order Logic deals with finding a common substitution for variables in different terms to make them match. The goal of unification is to make two expressions identical by assigning values to variables in a way that preserves their meanings.

47. How do you represent "All Dogs have Tails" in First Order Logic? Apr 2024

- We represent the statement in mathematical logic taking 'x 'as Dog and which has tail. We cannot represent two variable x, y for the same object Dog that has tail. The symbol "∀ "represent all.

48. What are knowledge based agents? Name its parts. Apr 2024

- Knowledge-based agents are artificial intelligence systems that use knowledge to perform

their tasks.

- They are composed of two main parts, a knowledge base, and an inference system, and they operate by making deductions, decisions, and conclusions based on the available knowledge.

PART B

1. **Explain in detail about 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 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.

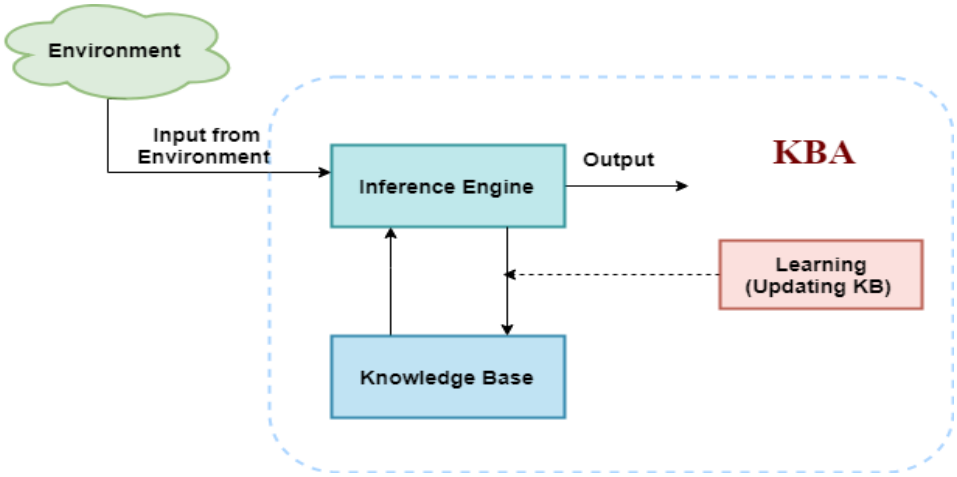


Fig.4.1 Architecture of Knowledge based Agent

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:
    - **Forward chaining**
    - **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:**

- Knowledge level 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. **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. **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.

However, in the real world, a successful agent can be built by combining both declarative and procedural approaches, and declarative knowledge can often be compiled into more efficient procedural code.

2. Explain in detail about Propositional logic.

What are logical connectives? Explain in detail. [Nov 2023]

- 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:

a) It is Sunday.

b) The Sun rises from West (False proposition)

c) 3+3= 7(False proposition)

d) 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.

- **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 ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has ∧ connective such as, **P ∧ Q** is called a conjunction.

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

   P= Rohan is intelligent,

   Q= Rohan is hardworking. → P∧ Q.
3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ 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 ∨ Q**.
4. **Implication:** A sentence such as P → 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 → Q
5. **Biconditional:** A sentence such as **P⇔ 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 ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ 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 Negation:

P	¬ P
True	**False**
False	**True**

For Conjunction:

P	Q	P∧ Q
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

For disjunction:

P	Q	P ∨ Q.
True	True	**True**
False	True	**True**
True	False	**True**
False	False	**False**

For Implication:

P	Q	P→ Q
True	True	**True**
True	False	**False**
False	True	**True**
False	False	**True**

For Biconditional:

P	Q	P⇔ Q
True	True	**True**
True	False	**False**
False	True	**False**
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	¬R	Pv Q	PvQ→¬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 ¬Rv Q, It can be interpreted as (¬R) v 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⇔B. In below truth table we can see that column for ¬A∨ B and A→B, are identical hence A is Equivalent to B

A	B	¬A	¬A∨ B	A→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∧ Q= Q ∧ P, or
  - P ∨ Q = Q ∨ P.
- **Associativity:**
  - (P ∧ Q) ∧ R= P ∧ (Q ∧ R),
  - (P ∨ Q) ∨ R= P ∨ (Q ∨ R)
- **Identity element:**
  - P ∧ True = P,
  - P ∨ True= True.
- **Distributive:**
  - P∧ (Q ∨ R) = (P ∧ Q) ∨ (P ∧ R).
  - P ∨ (Q ∧ R) = (P ∨ Q) ∧ (P ∨ R).
- **DE Morgan's Law:**
  - ¬ (P ∧ Q) = (¬P) ∨ (¬Q)
  - ¬ (P ∨ Q) = (¬ P) ∧ (¬Q).
- **Double-negation elimination:**
  - ¬ (¬P) = P.

Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  a. **All the girls are intelligent.**

b. **Some apples are sweet.**

Propositional logic has limited expressive power.

In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

3. Explain in detail about Propositional theorem proving.

Theorem proving

- Applying rules of inference directly to the sentences in our knowledge base to construct a proof of the desired sentence without consulting models.
- **Inference rules** are patterns of sound inference that can be used to find proofs. The **resolution** rule yields a complete inference algorithm for knowledge bases that are expressed in **conjunctive normal form**.
- **Forward chaining** and **backward chaining** are very natural reasoning algorithms for knowledge bases in **Horn form**.

Logical equivalence

- Two sentences a and β are logically equivalent if they are true in the same set of models. (write as $a \equiv \beta$). Also: $a \equiv \beta$ if and only if $a \subset \beta$ and $\beta \subset a$.

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Figure 4.2 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

Validity: A sentence is valid if it is true in all models.

Valid sentences are also known as **tautologies**—they are necessarily true. Every valid sentence is logically equivalent to *True*.

The **deduction theorem**: *For any sentence aand β, a* ⊂ *β if and only if the sentence (a* ⇒ *β) is valid.* **Satisfiability:** A sentence is satisfiable if it is true in, or satisfied by, some

model. Satisfiability can be checked by enumerating the possible models until one is found that satisfies the sentence.

The SAT problem: The problem of determining the satisfiability of sentences in propositional logic.

Validity and satisfiability are connected:

a is valid iff $\neg a$ is unsatisfiable;

a is satisfiable iff $\neg a$ is not valid;

$a \models \beta$ if and only if the sentence $(a \wedge \neg \beta)$ is unsatisfiable.

Proving β from a by checking the unsatisfiability of $(a \wedge \neg \beta)$ corresponds to **proof by refutation / proof by contradiction**.

Inference and proofs

Inferences rules (such as Modus Ponens and And-Elimination) can be applied to derived to a **proof**.

Modus Ponens:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}.$$

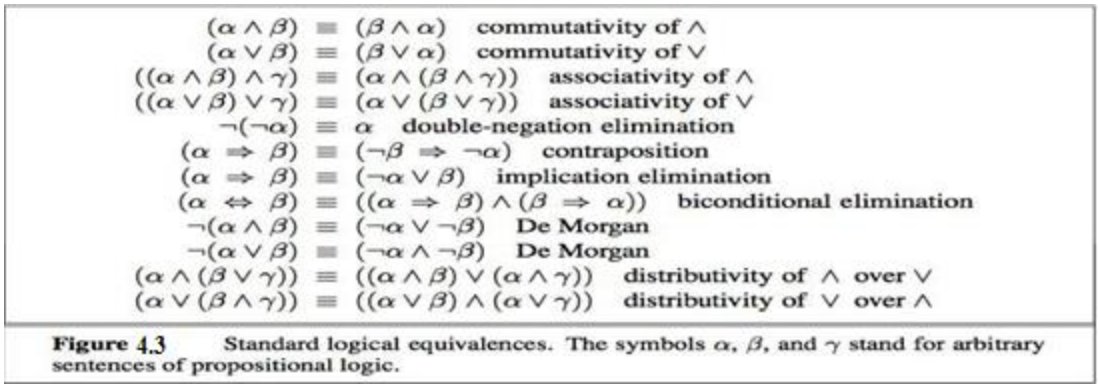
Whenever any sentences of the form $a \Rightarrow \beta$ and a are given, then the sentence β can be inferred.

And-Elimination:

$$\frac{\alpha \wedge \beta}{\alpha}.$$

From a conjunction, any of the conjuncts can be inferred.

All of **logical equivalence** (in Figure 7.11) can be used as inference rules.



$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$ commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$ commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$ associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$ associativity of $\vee$
$\neg(\neg\alpha) \equiv \alpha$ double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$ contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$ implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$ De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$ De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$ distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$ distributivity of $\vee$ over $\wedge$

Figure 4.3 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

e.g. The equivalence for biconditional elimination yields 2 inference rules:

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{and} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}.$$

De Morgan's rule

We can apply any of the search algorithms in Chapter 3 to find a sequence of steps that constitutes a proof. We just need to define a proof problem as follows:

- INITIAL STATE: the initial knowledge base;
- ACTION: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule.
- RESULT: the result of an action is to add the sentence in the bottom half of the inference rule.
- GOAL: the goal is a state that contains the sentence we are trying to prove.

In many practical cases, finding a proof can be more efficient than enumerating models, because the proof can ignore irrelevant propositions, no matter how many of them they are.

Monotonicity: A property of logical system, says that the set of entailed sentences can only increased as information is added to the knowledge base.

For any sentences a and β, If KB c $a$then KB $\wedge\beta$ c a.

Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base, what else in the knowledge base cannot invalidate any conclusion already inferred.

Proof by resolution

Resolution: An inference rule that yields a complete inference algorithm when coupled with any complete search algorithm.

Clause: A disjunction of literals. (e.g. A∨ B). A single literal can be viewed as a **unit clause** (a disjunction of one literal).

Unit resolution inference rule: Takes a clause and a literal and produces a new clause.

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

where each *l* is a literal, *li* and *m* are complementary literals **(one is the negation of the other).**

Full resolution rule: Takes 2 clauses and produces a new clause.

$$\frac{l_1 \vee \cdots \vee l_k, \quad m_1 \vee \cdots \vee m_n}{l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee vm_n}$$

where *li* and mj are complementary literals.

Notice: The resulting clause should contain only one copy of each literal. The removal of multiple copies of literal is called **factoring**.

e.g. resolve(A∨ B) with (A∨ ¬B), obtain(A∨ A) and reduce it to just A. The resolution rule is sound and complete.

Conjunctive normal form

Conjunctive normal form (CNF): A sentence expressed as a conjunction of clauses is said to be in CNF.

Every sentence of propositional logic is logically equivalent to a conjunction of clauses, after converting a sentence into CNF, it can be used as input to a resolution procedure.

A resolution algorithm

```
function PL-RESOLUTION(KB, α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  loop do
    for each pair of clauses C_i, C_j in clauses do
      resolvents ← PL-RESOLVE(C_i, C_j)
      if resolvents contains the empty clause then return true
      new ← new ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```

Figure 4.4 A simple resolution algorithm for propositional logic. The function PL-RESOLVE returns the set of all possible clauses obtained by resolving its two inputs.

e.g. KB = (B1,1⇔(P1,2∨P2,1))∧¬B1,1

α = ¬P1,2

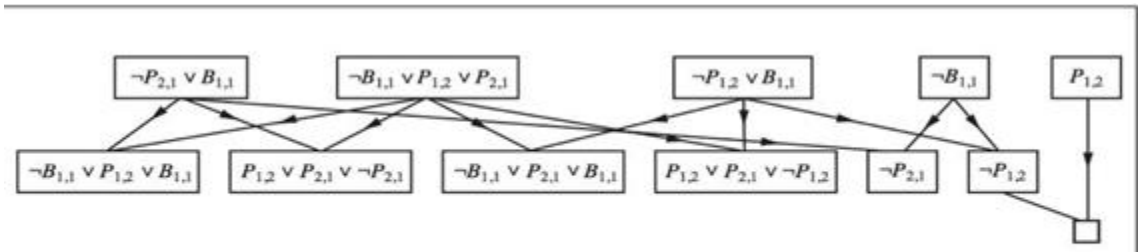


Figure 4.5 Partial application of PL-RESOLUTION to a simple inference in the wumpus world. $\neg P_{1,2}$ is shown to follow from the first four clauses in the top row.

Notice: Any clause in which two complementary literals appear can be discarded, because it is always equivalent to *True*.

e.g. B1,1∨¬B1,1∨P1,2 = True∨P1,2 = True. PL-RESOLUTION is complete.

$$
\begin{aligned}
\textit{CNFSentence} &\rightarrow \textit{Clause}_1 \wedge \cdots \wedge \textit{Clause}_n \\
\textit{Clause} &\rightarrow \textit{Literal}_1 \vee \cdots \vee \textit{Literal}_m \\
\textit{Literal} &\rightarrow \textit{Symbol} \mid \neg \textit{Symbol} \\
\textit{Symbol} &\rightarrow P \mid Q \mid R \mid \ldots \\
\textit{HornClauseForm} &\rightarrow \textit{DefiniteClauseForm} \mid \textit{GoalClauseForm} \\
\textit{DefiniteClauseForm} &\rightarrow (\textit{Symbol}_1 \wedge \cdots \wedge \textit{Symbol}_l) \Rightarrow \textit{Symbol} \\
\textit{GoalClauseForm} &\rightarrow (\textit{Symbol}_1 \wedge \cdots \wedge \textit{Symbol}_l) \Rightarrow \textit{False}
\end{aligned}
$$

Figure 4.6 A grammar for conjunctive normal form, Horn clauses, and definite clauses. A clause such as $A \wedge B \Rightarrow C$ is still a definite clause when it is written as $\neg A \vee \neg B \vee C$, but only the former is considered the canonical form for definite clauses. One more class is the k-CNF sentence, which is a CNF sentence where each clause has at most k literals.

Horn clauses and definite clauses

- **Definite clause:** A disjunction of literals of which exactly one is positive. (e.g. ¬ L1,1∨¬Breeze∨B1,1)

  Every definite clause can be written as an implication, whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.

- **Horn clause:** A disjunction of literals of which at most one is positive. (All definite clauses are Horn clauses.) In Horn form, the premise is called the **body** and the conclusion is called the **head**.

- Inference with horn clauses can be done through the **forward-chaining** and **backward-chaining** algorithms. Deciding entailment with Horn clauses can be done in time that is linear in the size of the knowledge base.

- **Goal clause:** A clause with no positive literals.

Forward and backward chaining

Forward-chaining algorithm: PL-FC-ENTAILS?(KB, q) (runs in linear time) Forward chaining is sound and complete.

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
    p ← POP(agenda)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

Figure 4.7 The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

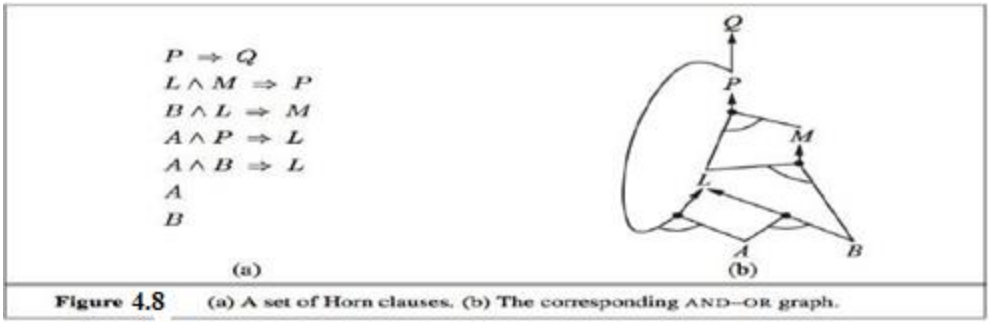


Figure 4.8 (a) A set of Horn clauses. (b) The corresponding AND–OR graph.

e.g. A knowledge base of horn clauses with A and B as known facts.

Fixed point: The algorithm reaches a fixed point where no new inferences are possible.

Data-driven reasoning: Reasoning in which the focus of attention starts with the known data. It can be used within an agent to derive conclusions from incoming percept, often without a specific query in mind. (forward chaining is an example)

Backward-chaining algorithm: works backward rom the query. If the query q is known to be true, no work is needed;

Otherwise the algorithm finds those implications in the KB whose conclusion is q. If all the premises of one of those implications can be proved true (by backward chaining), then q is true. (runs in linear time) in the corresponding AND-OR graph: it works back down the graph until it reaches a set of known facts. (Backward-chaining algorithm is essentially identical to the AND-OR-GRAPH-SEARCH algorithm.) Backward-chaining is a form of **goal-directed reasoning**.

4. Explain in detail about Propositional model checking.

- The set of possible models, given a fixed propositional vocabulary, is finite, so entailment can be checked by enumerating models. Efficient **model-checking** inference algorithms for propositional logic include backtracking and local search methods and can often solve large problems quickly.

2 families of algorithms for the SAT problem based on model checking:

- based on backtracking
- based on local hill-climbing search

1. A complete backtracking algorithm David-Putnam algorithm (DPLL):

```
function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false

  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

Figure 4.9 The DPLL algorithm for checking satisfiability of a sentence in propositional logic. The ideas behind FIND-PURE-SYMBOL and FIND-UNIT-CLAUSE are described in the text; each returns a symbol (or null) and the truth value to assign to that symbol. Like TT-ENTAILS?, DPLL operates over partial models.

- DPLL embodies 3 improvements over the scheme of TT-ENTAILS?: Early termination, pure symbol heuristic, unit clause heuristic.
- Tricks that enable SAT solvers to scale up to large problems:
- Component analysis, ordering, intelligent backtracking, random restarts, clever indexing.

Local search algorithms

- Local search algorithms can be applied directly to the SAT problem, provided that choose the right evaluation function. (We can choose an evaluation function that counts the number of unsatisfied clauses.)
- These algorithms take steps in the space of complete assignments, flipping the truth value of one symbol at a time. The space usually contains many local minima, to escape from which various forms of randomness are required.
- **Local search** methods such as WALKSAT can be used to find solutions. Such algorithm are sound but not complete.
- **WALKSAT:** one of the simplest and most effective algorithms.

```
function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a "random walk" move, typically around 0.5
          max_flips, number of flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for i = 1 to max_flips do
      if model satisfies clauses then return model
      clause ← a randomly selected clause from clauses that is false in model
      with probability p flip the value in model of a randomly selected symbol from clause
      else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```

Figure 4.10 The WALKSAT algorithm for checking satisfiability by randomly flipping the values of variables. Many versions of the algorithm exist.

The landscape of random SAT problems

- **Underconstrained problem:** When we look at satisfiability problems in CNF, an underconstrained problem is one with relatively few clauses constraining the variables.
- An **overconstrained problem** has many clauses relative to the number of variables and is likely to have no solutions.
- The notation *CNFk(m, n)* denotes a *k*-CNF sentence with *m* clauses and *n* symbols. (with *n* variables and *k* literals per clause).
- Given a source of random sentences, where the clauses are chosen uniformly, independently and without replacement from among all clauses with k different literals, which are positive or negative at random.
- **Hardness:** problems right at the threshold > overconstrained problems > underconstrained problems

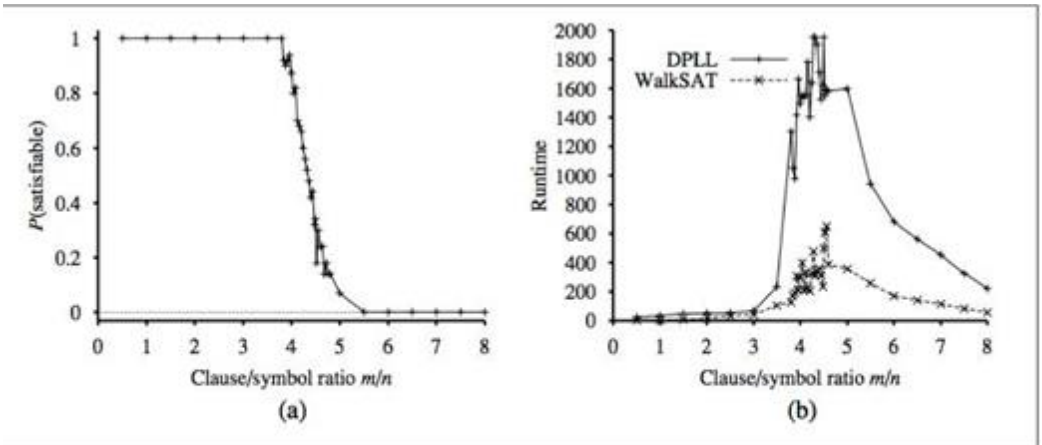


Figure 4.11 (a) Graph showing the probability that a random 3-CNF sentence with $n = 50$ symbols is satisfiable, as a function of the clause/symbol ratio m/n. (b) Graph of the median run time (measured in number of recursive calls to DPLL, a good proxy) on random 3-CNF sentences. The most difficult problems have a clause/symbol ratio of about 4.3.

Satifiability threshold conjecture: A theory says that for every k≥3, there is a threshold ratio r_k, such that as n goes to infinity, the probability that $CNF_k(n, rn)$ is satisfiable becomes 1 for all values or r below the threshold, and 0 for all values above. (remains unproven)

5. Explain in detail about Agents based on propositional logic.

1. The current state of the world

We can associate proposition with timestamp to avoid contradiction.

e.g. $\neg Stench^3$, $Stench^4$

Fluent: refer an aspect of the world that changes. (E.g. $L^t x,y$)

Temporal variables: Symbols associated with permanent aspects of the world do not need a time superscript.

Effect axioms: specify the outcome of an action at the next time step.

Frame problem: some information lost because the effect axioms fails to state what remains unchanged as the result of an action.

Solution: add frame axioms explicity asserting all the propositions that remain the same.

Representation frame problem: The proliferation of frame axioms is inefficient, the set of frame axioms will be O(mn) in a world with m different actions and n fluents.

Solution: because the world exhibits **locaility** (for humans each action typically changes no more than some number k of those fluents.) Define the transition model with a set of axioms of size O(mk) rather than size O(mn).

Inferential frame problem: The problem of projecting forward the results of a t step lan of action in time O(kt) rather than O(nt).

Solution: change one's focus from writing axioms about actions to writing axioms about fluents.

For each fluent F, we will have an axiom that defines the truth value of F^{t+1} in terms of fluents at time t and the action that may have occurred at time t.

The truth value of F^{t+1} can be set in one of 2 ways:

Either a. The action at time t cause F to be true at t+1

Or b. F was already true at time t and the action at time t does not cause it to be false. An axiom of this form is called a **successor-state axiom** and has this schema:

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t) .$$

Qualification problem: specifying all unusual exceptions that could cause the action to fail.

2. A hybrid agent

- **Hybrid agent:** combines the ability to deduce various aspect of the state of the world with condition-action rules, and with problem-solving algorithms.
- The agent maintains and update KB as a current plan.
- The initial KB contains the atemporal axioms. (don't depend on t)
- At each time step, the new percept sentence is added along with all the axioms that depend on t (such as the successor-state axioms).
- Then the agent use logical inference by ASKING questions of the KB (to work out which squares are safe and which have yet to be visited).

The main body of the agent program constructs a plan based on a decreasing priority of goals:

1. If there is a glitter, construct a plan to grab the gold, follow a route back to the initial location and climb out of the cave;
2. Otherwise if there is no current plan, plan a route (with A* search) to the closest safe square unvisited yet, making sure the route goes through only safe squares;
3. If there are no safe squares to explore, if still has an arrow, try to make a safe square by shooting at one of the possible wumpus locations.
4. If this fails, look for a square to explore that is not provably unsafe.
5. If there is no such square, the mission is impossible, then retreat to the initial location and climb out of the cave.

```
function HYBRID-WUMPUS-AGENT(percept) returns an action
  inputs: percept, a list, [stench,breeze,glitter,bump,scream]
  persistent: KB, a knowledge base, initially the atemporal "wumpus physics"
              t, a counter, initially 0, indicating time
              plan, an action sequence, initially empty

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  TELL the KB the temporal "physics" sentences for time t
  safe ← {[x, y] : ASK(KB, OK^t_{x,y}) = true}
  if ASK(KB, Glitter^t) = true then
    plan ← [Grab] + PLAN-ROUTE(current, {[1,1]}, safe) + [Climb]
  if plan is empty then
    unvisited ← {[x, y] : ASK(KB, L^{t'}_{x,y}) = false for all t' ≤ t}
    plan ← PLAN-ROUTE(current, unvisited ∩ safe, safe)
  if plan is empty and ASK(KB, HaveArrow^t) = true then
    possible_wumpus ← {[x, y] : ASK(KB, ¬ W_{x,y}) = false}
    plan ← PLAN-SHOT(current, possible_wumpus, safe)
  if plan is empty then  // no choice but to take a risk
    not_unsafe ← {[x, y] : ASK(KB, ¬ OK^t_{x,y}) = false}
    plan ← PLAN-ROUTE(current, unvisited ∩ not_unsafe, safe)
  if plan is empty then
    plan ← PLAN-ROUTE(current, {[1, 1]}, safe) + [Climb]
  action ← POP(plan)
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action

function PLAN-ROUTE(current,goals,allowed) returns an action sequence
  inputs: current, the agent's current position
          goals, a set of squares; try to plan a route to one of them
          allowed, a set of squares that can form part of the route

  problem ← ROUTE-PROBLEM(current, goals, allowed)
  return A*-GRAPH-SEARCH(problem)
```

Figure 4.12 A hybrid agent program for the wumpus world. It uses a propositional knowledge base to infer the state of the world, and a combination of problem-solving search and domain-specific code to decide what actions to take.

Weakness: The computational expense goes up as time goes by.

3. Logical state estimation

To get a constant update time, we need to **cache** the result of inference.

Belief state: Some representation of the set of all possible current state of the world. (used to replace the past history of percepts and all their ramifications)

$$WumpusAlive^1 \wedge L^1_{2,1} \wedge B_{2,1} \wedge (P_{3,1} \vee P_{2,2})$$

We use a logical sentence involving the proposition symbols associated with the current time step and the temporal symbols.

Logical **state estimation** involves maintaining a logical sentence that describes the set

of possible states consistent with the observation history.

Each update step requires inference using the transition model of the environment, which is built from **successor-state axioms** that specify how each **fluent** changes.

State estimation: The process of updating the belief state as new percepts arrive.

Exact state estimation may require logical formulas whose size is exponential in the number of symbols.

One common scheme for approximate state estimation: to represent belief state as conjunctions of literals (1-CNF formulas).

The agent simply tries to prove X^t and $\neg X^t$ for each symbol X^t, given the belief state at t-1.

The conjunction of provable literals becomes the new belief state, and the previous belief state is discarded. (This scheme may lose some information as time goes along.)

The set of possible states represented by the 1-CNF belief state includes all states that are in fact possible given the full percept history. The 1-CNF belief state acts as a simple outer envelope, or **conservative approximation**.

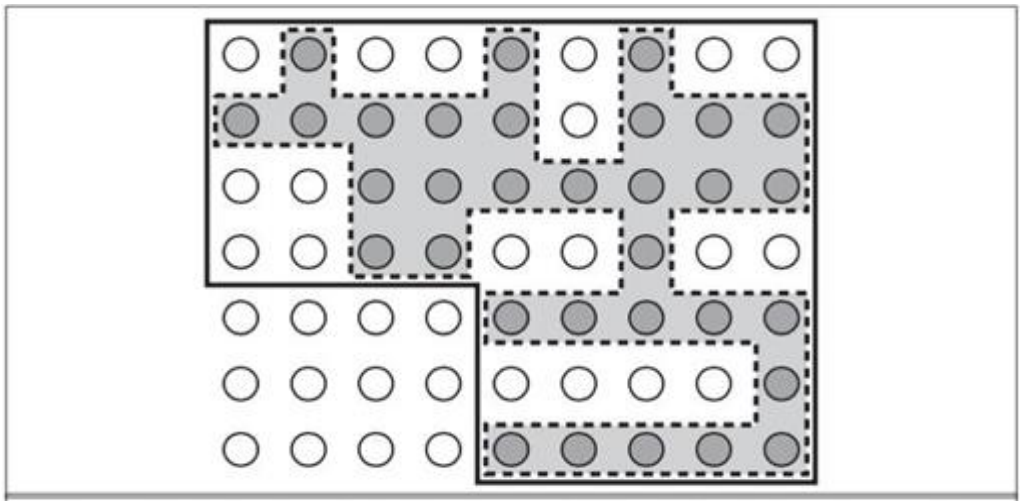


Figure 4.13 Depiction of a 1-CNF belief state (bold outline) as a simply representable, conservative approximation to the exact (wiggly) belief state (shaded region with dashed outline). Each possible world is shown as a circle; the shaded ones are consistent with all the percepts.

Describe an algorithm for general propositional inference based on model checking.

[Nov 2023]

4. Making plans by propositional inference

We can make plans by logical inference instead of A* search in Figure 7.20. Basic idea:

1. Construct a sentence that includes:

- $Init^0$: a collection of assertions about the initial state;

- Transition^1, …, Transition^t: The successor-state axioms for all possible actions at each time up to some maximum time t;
- HaveGold^t∧ClimbedOut^t: The assertion that the goal is achieved at time t.

2. Present the whole sentence to a SAT solver. If the solver finds a satisfying model, the goal is achievable; else the planning is impossible.

3. Assuming a model is found, extract from the model those variables that represent actions and are assigned true.

Together they represent a plan to achieve the goals.

Decisions within a logical agent can be made by SAT solving: finding possible models specifying future action sequences that reach the goal. This approach works only for fully observable or sensorless environment.

5. SATPLAN: A propositional planning. (Cannot be used in a partially observable environment)

SATPLAN finds models for a sentence containing the initial sate, the goal, the successor-state axioms, and the action exclusion axioms.

(Because the agent does not know how many steps it will take to reach the goal, the algorithm tries each possible number of steps t up to some maximum conceivable plan length T_{max}.)

```
function SATPLAN(init, transition, goal, T_max) returns solution or failure
  inputs: init, transition, goal, constitute a description of the problem
          T_max, an upper limit for plan length

  for t = 0 to T_max do
    cnf ← TRANSLATE-TO-SAT(init, transition, goal, t)
    model ← SAT-SOLVER(cnf)
    if model is not null then
      return EXTRACT-SOLUTION(model)
  return failure
```

Figure 4.14 The SATPLAN algorithm. The planning problem is translated into a CNF sentence in which the goal is asserted to hold at a fixed time step t and axioms are included for each time step up to t. If the satisfiability algorithm finds a model, then a plan is extracted by looking at those proposition symbols that refer to actions and are assigned *true* in the model. If no model exists, then the process is repeated with the goal moved one step later.

Precondition axioms: stating that an action occurrence requires the preconditions to be satisfied, added to avoid generating plans with illegal actions.

Action exclusion axioms: added to avoid the creation of plans with multiple simultaneous actions that interfere with each other.

Propositional logic does not scale to environments of unbounded size because it lacks

the expressive power to deal concisely with time, space and universal patterns of relationshipgs among objects.

6. Explain in detail about First-order logic.

First-Order Logic in Artificial intelligence

- In the topic of Propositional logic, we have seen that how to represent statements using propositional logic.
- But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements.
- The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.
    - "**Some humans are intelligent", or**
    - "**Sachin likes cricket.**"
- To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first- order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
    - **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, ...
    - **Relations: It can be unary relation such as:** red, round, is adjacent, **or n-any relation such as:** the sister of, brother of, has color, comes between

- **Function:** Father of, best friend, third inning of, end of, ...

- As a natural language, first-order logic also has two main parts:

  a. **Syntax**

  b. **Semantics**

Syntax of First-Order logic:

- The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ,term n)**.

Example:

Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).

Chinky is a cat: => cat (Chinky).

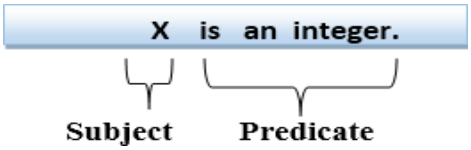
Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
    a. **Universal Quantifier, (for all, everyone, everything)**
    b. **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

If x is a variable, then ∀x is read as:

- **For all x**

- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:

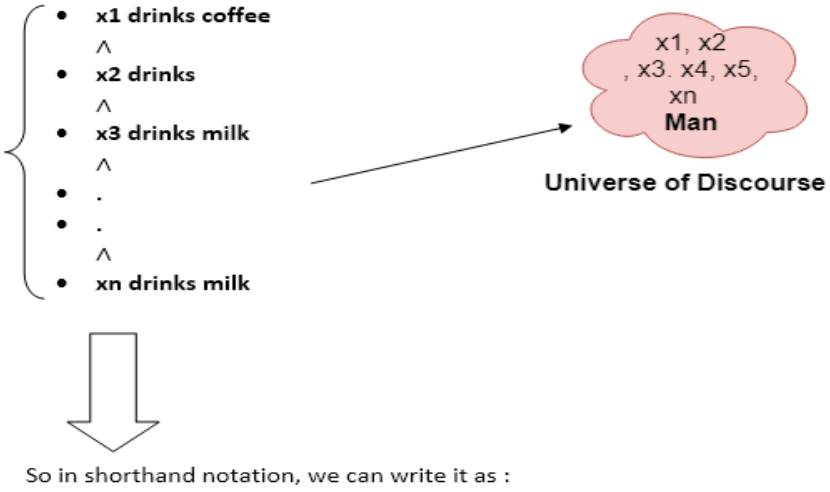


Fig.4.15 Flow of Universal Quantifier

∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

- Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.
- It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.

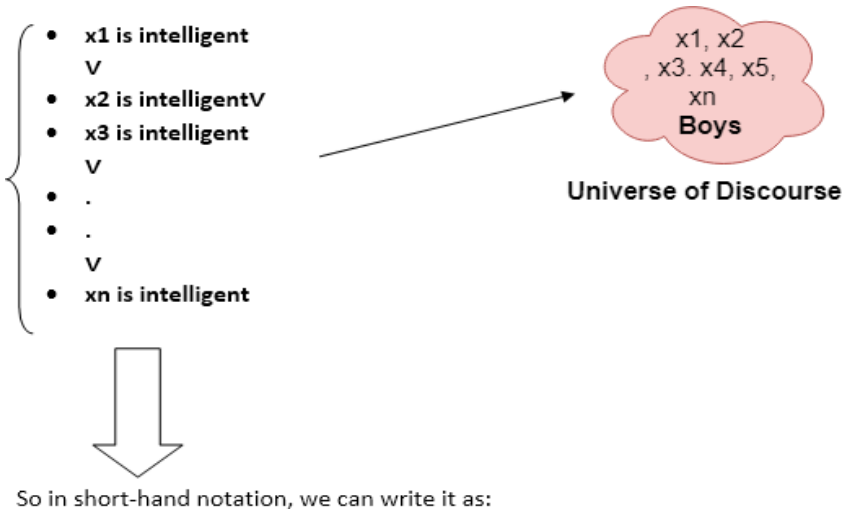


Fig.4.16 Flow of Universal Quantifier

∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier ∀ is implication →.
- The main connective for existential quantifier ∃ is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. **All birds fly.**

In this question the predicate is "**fly(bird)**."

And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. **Every man respects his parent.**

In this question, the predicate is "**respect(x, y),**" **where x=man, and y= parent**. Since there is every man so will use

∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. **Some boys play cricket.**

In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:

∃x boys(x) → play(x, cricket).

4. **Not all students like both Mathematics and Science.**

In this question, the predicate is "**like(x, y)," where x= student, and y= subject**.

Since there are not all students, so we will use **∀ with negation, so** following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. **Only one student failed in Mathematics.**

In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.

Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) → ¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, *z*)], where *z* is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: ∀x [A (x) B(y)], here x and y are the bound variables.

7. Explain in detail about Knowledge representation and engineering.

Discuss on Knowledge representation and engineering. [Apr 2024]

Discuss the knowledge Engineering Process with proper illustration. Depict the concept of forward chaining. [Nov 2023]

Knowledge Engineering in First-order logic

What is knowledge-engineering?

- The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering. In **knowledge- engineering**, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as **knowledge engineer**.
- In this topic, we will understand the Knowledge engineering process in an electronic circuit domain, which is already familiar. This approach is mainly suitable for creating **special-purpose knowledge base**.

The knowledge-engineering process:

- Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (**One-bit full adder**) which is given below

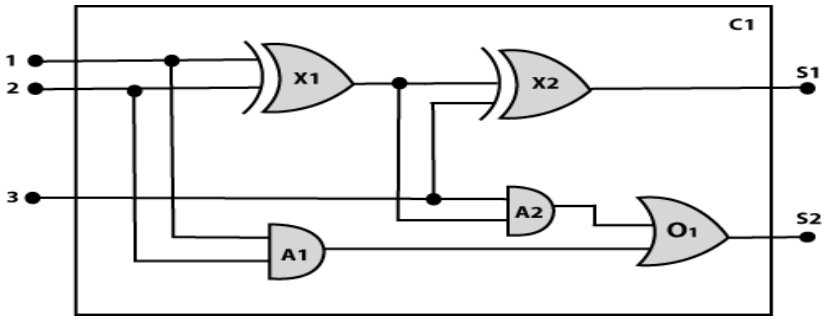


Fig. 4.17 Knowledge Engineering Process

1. **Identify the task:**

- The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks At the first level or highest level, we will examine the functionality of the circuit:
  - **Does the circuit add properly?**
  - **What will be the output of gate A2, if all the inputs are high?**

At the second level, we will examine the circuit structure details such as:

- **Which gate is connected to the first input terminal?**
- **Does the circuit have feedback loops?**

2. Assemble the relevant knowledge:

- In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:
- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.
- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

3. Decide on vocabulary:

- The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates. Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**.
- The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**. Circuits will be identified by a predicate: **Circuit (C1)**.
- For the terminal, we will use predicate: **Terminal(x)**.
- For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.
- The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.
- The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**. We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

4. Encode general knowledge about the domain:

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

$\forall$ t1, t2 Terminal (t1) $\wedge$ Terminal (t2) $\wedge$ Connect (t1, t2) $\rightarrow$ Signal (t1) = Signal (2).

- Signal at every terminal will have either value 0 or 1, it will be represented as:

∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.

- Connect predicates are commutative:

∀ t1, t2 Connect(t1, t2) → Connect (t2, t1).

- Representation of types of gates:

∀ g Gate(g) ∧ r = Type(g) → r = OR ∨r = AND ∨r = XOR ∨r = NOT.

- Output of AND gate will be zero if and only if any of its input is zero.

∀ g Gate(g) ∧ Type(g) = AND →Signal (Out(1, g))= 0 ⇔ ∃n Signal (In(n, g))= 0.

- Output of OR gate is 1 if and only if any of its input is 1:

∀ g Gate(g) ∧ Type(g) = OR → Signal (Out(1, g))= 1 ⇔ ∃n Signal (In(n, g))= 1

- Output of XOR gate is 1 if and only if its inputs are different:

∀ g Gate(g) ∧ Type(g) = XOR → Signal (Out(1, g)) = 1 ⇔ Signal (In(1, g)) ≠ Signal (In(2, g)).

- Output of NOT gate is invert of its input:

∀ g Gate(g) ∧ Type(g) = NOT → Signal (In(1, g)) ≠ Signal (Out(1, g)).

- All the gates in the above circuit have two inputs and one output (except NOT gate).

∀ g Gate(g) ∧ Type(g) = NOT → Arity(g, 1, 1)

∀ g Gate(g) ∧ r =Type(g) ∧ (r= AND ∨r= OR ∨r= XOR) → Arity (g, 2, 1).

- All gates are logic circuits:

∀ g Gate(g) → Circuit (g).

5. Encode a description of the problem instance:

Now we encode problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if ontology about the problem is already thought. This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology.

For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

For XOR gate: Type(x1)=

XOR, Type(X2) = XOR For
AND gate: Type(A1) = AND,
Type(A2)= AND For OR
gate: Type (O1) = OR.

And then represent the connections between all the gates.

6. Pose queries to the inference procedure and get answers:

In this step, we will find all the possible set of values of all the terminal for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3

∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like 1 ≠ 0.

8. Explain in detail about Inferences in first-order logic.

Inference in First-Order Logic

- Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

- Substitution is a fundamental operation performed on terms and formulas. It occurs in all inference systems in first- order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality:

- First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can

use **equality symbols** which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

- As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x ≠y.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

1. Universal Generalization
2. Universal Instantiation
3. Existential Instantiation
4. Existential introduction

1. Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as $\frac{P(c)}{\forall x\, P(x)}$.
- It can be represented as: .
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

Example: Let's represent, P(c): "**A byte contains 8 bits**", so for ∀ **x P(x)** "**All bytes contain 8 bits**.", it will also be true.

2. Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.

- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from ∀ **x P(x) for any object in the universe of discourse**.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$.
- **Example:1.**
- IF "Every person like ice-cream"=> ∀x P(x) so we can infer that "John likes ice-cream" => P(c)
- **Example: 2.**
- Let's take a famous example,
- "All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

∀x king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

- **King(John) ∧ Greedy (John) → Evil (John),**
- **King(Richard) ∧ Greedy (Richard) → Evil (Richard),**
- **King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),**

3. **Existential Instantiation:**

Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.

- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.

- It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$

Example:

From the given sentence: ∃**x Crown(x)** ∧ **OnHead(x, John),**

So we can infer: **Crown(K)** ∧ **OnHead(K, John),** as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$
- **Example: Let's say that,**

  "Priyanka got good marks in English." "Therefore, someone got good marks in English."

Generalized Modus Ponens Rule:

- For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.
- Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."
- According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',)**

= SUBST(θ, pi), it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)}{SUBST(\theta, q)}$$

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

Here let say, p1' is king(John)

p1 is king(x)

p2' is Greedy(y)

p2 is Greedy(x)

θ is {x/John, y/John} q is evil(x)

SUBST(θ,q).

9. Explain in detail about Forward chaining and Backward chaining.

Forward Chaining and backward chaining in AI

- In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

- The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

    A. Forward chaining

    B. Backward chaining Horn Clause and Definite clause

- Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

- **Definite clause:** A clause which is a disjunction of literals with **exactly one**

positive literal is known as a definite clause or strict horn clause.

- **Horn clause:** A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.
- **Example: (¬ p V ¬ q V k)**. It has only one positive literal k. It is equivalent to p ∧ q → k.

A. Forward Chaining

- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Consider the following famous example which we will use in both approaches:

Example:

"**As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen.**"

Prove that "**Robert is criminal.**"

- To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p)...(1)

- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A, T1)……. (2)

  Missile(T1)…….. (3)

- All of the missiles were sold to country A by Robert.

  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) (4)

- Missiles are weapons.

  Missile(p) → Weapons (p)…. **(5)**

- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p)..**(6)**

- Country A is an enemy of America.

  Enemy (A, America) ….. (7)

- Robert is American

  American(Robert)……..(8)

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1)**. All these facts will be represented as below.

Step-2:

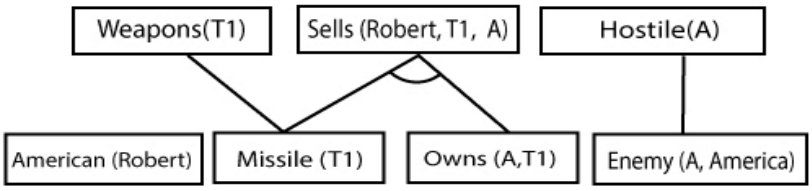
At the second step, we will see those facts which infer from available facts and with satisfied premises. Rule-(1) does not satisfy premises, so it will not be added in the first

iteration.

Rule-(2) and (3) are already added.

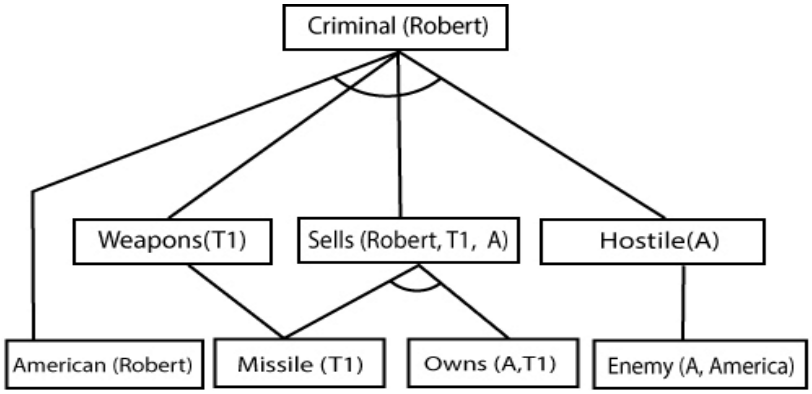
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:

- At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

B. Backward Chaining:

- Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.

- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**

  Owns(A, T1)..... (2)
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) (4)**
- **Missile(p) → Weapons (p)**.....**(5)**
- **Enemy(p, America) →Hostile(p)**.... **(6)**
- **Enemy (A, America) ..... (7)**
- **American(Robert)..... (8)**

Backward-Chaining proof:

- In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

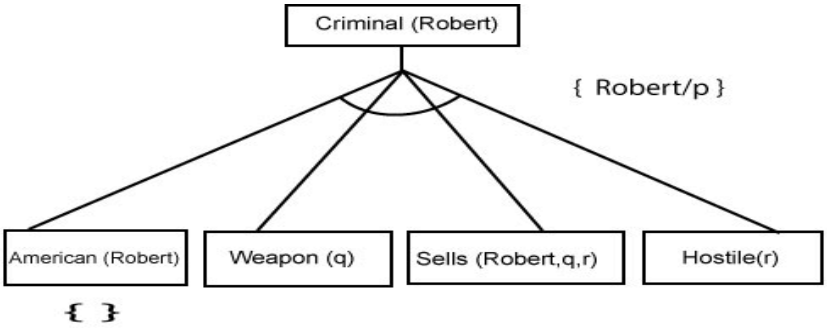
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.

Criminal (Robert)

Step-2:

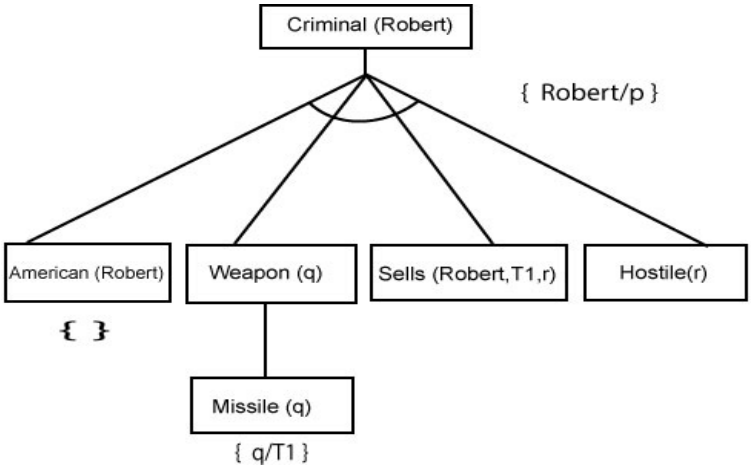
- At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.



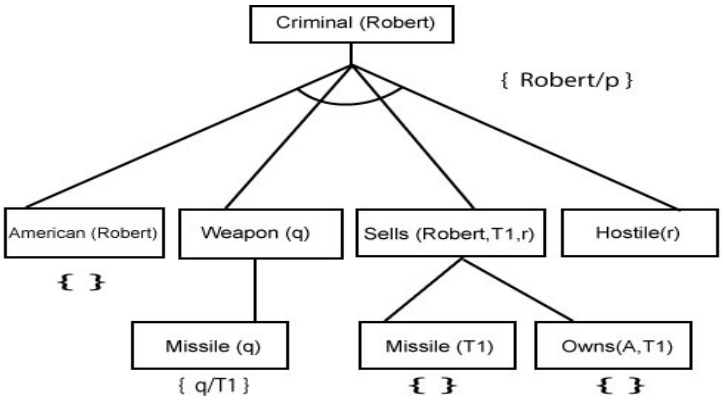
Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon

(q) is also true with the substitution of a constant T1 at q.



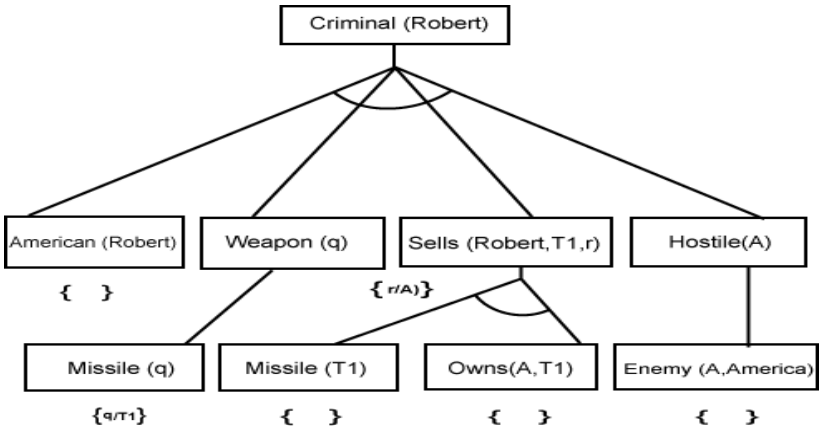
Step-4:

At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



10. Difference between backward chaining and forward chaining.

Difference between backward chaining and forward chaining with example. [Apr 2024]

Following is the difference between the forward chaining and backward chaining:

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal- driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top- down** approach.
- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both applies **Modus ponens** inference rule.
- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge

base, whereas in backward chaining there can be fewer ASK questions.

- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

SNO	**Forward Chaining**	**Backward Chaining**
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data- driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

11. Explain in detail about Resolution.

Resolution in FOL

Resolution

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.
- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

The resolution inference rule:

- The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$\frac{l_1 \vee \ldots\ldots \vee l_{k,} \qquad m_1 \vee \ldots\ldots \vee m_n}{\text{SUBST}(\theta, l_1 \vee \ldots\ldots\ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots\ldots \vee l_k \vee m_1 \vee \ldots\ldots\ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots\ldots \vee m_n)}$$

Where **li** and **mj** are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

- We can resolve two clauses which are given below:
- **[Animal (g(x) V Loves (f(x), x)] and [¬ Loves(a, b) V ¬Kills(a, b)]**
- Where two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**
- These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:

[Animal (g(x) V ¬ Kills(f(x), x)].

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

- John likes all kind of food.
- Apple and vegetable are food
- Anything anyone eats and not killed is food.
- Anil eats peanuts and still alive

a. Harry eats everything that Anil eats. Prove by resolution that:

b. John likes peanuts.

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

a. ∀x: food(x) → likes(John, x)
b. food(Apple) ∧ food(vegetables)
c. ∀x ∀y: eats(x, y) ∧ ¬ killed(x) → food(y)
d. eats (Anil, Peanuts) ∧ alive(Anil).
e. ∀x : eats(Anil, x) → eats(Harry, x)
f. ∀x: ¬ killed(x) → alive(x) } **added predicates.**
g. ∀x: alive(x) →¬ killed(x) }
h. likes(John, Peanuts)

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form

makes easier for resolution proofs.

- **Eliminate all implication (→) and rewrite**

  a. ∀x ¬ food(x) V likes(John, x)

  b. food(Apple) Λ food(vegetables)

  c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)

  d. eats (Anil, Peanuts) Λ alive(Anil)

  e. ∀x ¬ eats(Anil, x) V eats(Harry, x)

  f. ∀x¬ [¬ killed(x)] V alive(x)

  g. ∀x ¬ alive(x) V ¬ killed(x)

  h. likes(John, Peanuts).

- **Move negation (¬)inwards and rewrite**

. ∀x ¬ food(x) V likes(John, x)

  a. food(Apple) Λ food(vegetables)

  b. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)

  c. eats (Anil, Peanuts) Λ alive(Anil)

  d. ∀x ¬ eats(Anil, x) V eats(Harry, x)

  e. ∀x ¬killed(x)] V alive(x)

  f. ∀x ¬ alive(x) V ¬ killed(x)

  g. likes(John, Peanuts).

- **Rename variables or standardize variables**

. ∀x ¬ food(x) V likes(John, x)

  a. food(Apple) Λ food(vegetables)

  b. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)

  c. eats (Anil, Peanuts) Λ alive(Anil)

  d. ∀w¬ eats(Anil, w) V eats(Harry, w)

  e. ∀g ¬killed(g)] V alive(g)

  f. ∀k ¬ alive(k) V ¬ killed(k)

  g. likes(John, Peanuts).

- **Eliminate existential instantiation quantifier by elimination.**
- In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.
- **Drop Universal quantifiers.**
- In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

  a. ¬ food(x) V likes(John, x)

  b. food(Apple)

  c. food(vegetables)

  d. ¬ eats(y, z) V killed(y) V food(z)

  e. eats (Anil, Peanuts)

  f. alive(Anil)

  g. ¬ eats(Anil, w) V eats(Harry, w)

  h. killed(g) V alive(g)

  i. ¬ alive(k) V ¬ killed(k)

  j. likes(John, Peanuts).

- **Distribute conjunction ∧ over disjunction ¬.**

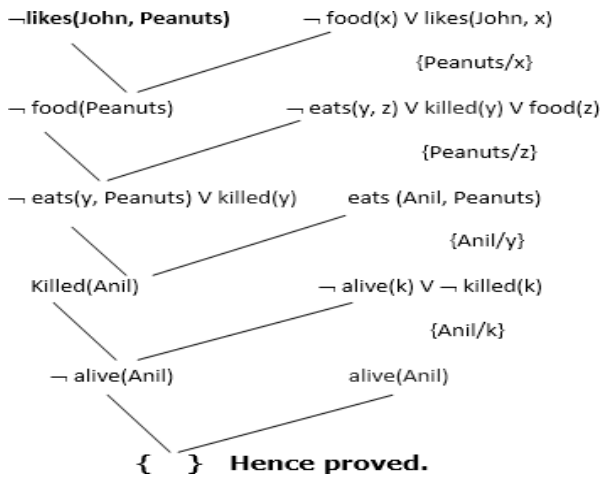
  This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Step-4: Draw Resolution graph:

Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- In the first step of resolution graph, **¬likes(John, Peanuts)** , and **likes(John, x)** get resolved(canceled) by substitution of **{Peanuts/x}**, and we are left with **¬ food(Peanuts)**
- In the second step of the resolution graph, **¬ food(Peanuts)** , and **food(z)** get resolved (canceled) by substitution of **{ Peanuts/z}**, and we are left with **¬ eats(y, Peanuts) V killed(y)** .
- In the third step of the resolution graph, **¬ eats(y, Peanuts)** and **eats (Anil, Peanuts)** get resolved by substitution **{Anil/y}**, and we are left with **Killed(Anil)** In the fourth step of the resolution graph, **Killed(Anil)** and **¬ killed(k)** get resolve by substitution **{Anil/k}**, and we are left with **¬ alive(Anil)** .
- In the last step of the resolution graph **¬ alive(Anil)** and **alive(Anil)** get resolved.

12. What is significance of knowledge representation? Mention the desirable properties of knowledge representation. [Nov 2023]

- Knowledge representation makes complex software easier to define and maintain than procedural code and can be used in expert systems. For example, talking to experts in terms of business rules rather than code lessens the semantic gap between users and developers and makes development of complex systems more practical.
- The properties of a good representation are representational adequacy, inferential adequacy, inferential efficiency and efficiency in the acquisition/modification. Several different

formalisms can follow those recommendations; among them, the more common way is based on logics.

13. What is Conjuctive normal form? Illustrate and explain the procedure to convert sentences into conjuctive normal form with a neat example. Depict ral time images where it could be applied. [Nov 2023]

Conjunctive normal form

The conjunctive normal form states that a formula is in CNF if it is a conjunction of one or more than one clause, where each clause is a disjunction of literals. In other words, it is a product of sums where $\wedge$ symbol occurs between the clauses and the $\vee$ symbol occurs in the clauses.

Steps to convert a formula into CNF

- We eliminate all the occurrences of $\oplus$ (XOR operator), $\rightarrow$ (conditional), and $\leftrightarrow$ (biconditional) from the formula. We convert it into its equivalent formula containing $\vee$, $\wedge$, and $\neg$ symbol. We use the following logical equivalences:
  - $A \oplus B \equiv (A \vee B) \wedge \neg(A \wedge B)$
  - $A \rightarrow B \equiv \neg A \vee B$
  - $A \leftrightarrow B \equiv (\neg A \vee B) \wedge (A \vee \neg B)$
  - $A \leftrightarrow B \equiv (A \wedge B) \vee (\neg A \wedge \neg B)$
- We move all the negations inwards to appear only as a part of the literal. The $\neg$ symbol can only precede a propositional variable or a predicate symbol. To accomplish this, we use the following logical equivalences:
  - $\neg\neg A \equiv A$
  - De Morgan's law
- Some common equivalences that we use for the conversion are:
  - **Commutativity for disjunction**: $A \vee B \equiv B \vee A$
  - **Commutativity for conjunction**: $A \wedge B \equiv B \wedge A$
  - **Associativity for disjunction**: $(A \vee B) \vee C \equiv A \vee (B \vee C)$
  - **Associativity for conjunction**: $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
  - **Distribution over disjunction**: $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
  - **Distribution over conjunction**: $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

MAILAM Engineering College

Approved by AICTE, New Delhi, affiliated to Anna University, Chennai, Accredited by Naac with A Grade & TCS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

II YEAR / III SEM

UNIT V PROBABILISTIC REASONING

Acting under uncertainty – Bayesian inference – naïve Bayes models. Probabilistic reasoning –Bayesian networks – exact inference in BN – approximate inference in BN – causal networks.

PART A

1. **Define uncertainty. Give an example?**
   - Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates.
   - With this knowledge representation, we might write A→B, which means if A istrue then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.
   - So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

2. **Define Belief state.**
   - Problem-solving agents and Logical agents designed to handle uncertainty by keeping track of a **belief state—**a representation of the set of all possible world states that it might be in—and generating a contingency plan that handles every possible eventuality that its sensors may report during execution.

3. **Define Bayes rule.**
   - Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(B),and P(A).
   - This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one.
   - Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, **then the Bayes' rule becomes:**

$$P(cause|effect) = \frac{P(effect|cause)\,P(cause)}{P(effect)}$$

4. Define Bayesian network.

- Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty.
- We can define a Bayesian network as: "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."
- It is also called a Bayes network, belief network, decision network, or Bayesian model.
- Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection.

5. List out the components of Bayesian network.

- The Bayesian network has mainly two components:
  - Causal Component
  - Actual numbers
- Each node in the Bayesian network has condition probability distribution P(Xi |Parent(Xi)), which determines the effect of the parent on that node.
- Bayesian network is based on Joint probability distribution and conditional probability.
- So let's first understand the joint probability distribution:
  - Joint Probability – It is the measure of two events happening at the same time. It can be written as P(A∩B)
  - Conditional Probability– It is the measure of the probability of an event occurring given that another event has occurred. In other terms, the conditional probability of an event X is the probability that the event will occur given that event Y has already occurred.
  - P(X|Y): Probability of event X occurring given that event Y already occurred.
  - If X and Y are dependent events, then P(X|Y) = P(X∩Y)/P(Y) If X and Y are independent events, then P(X∩Y) = 0So, P(X|Y) = P(X).

6. Write an Applications of Bayesian network.

1. **Healthcare Industry**

   The Bayesian network is used in the healthcare industry for the detection and prevention of diseases.

2. **Web Search**

   Bayesian Network modes can be used for search accuracy based on user intent.

3. **Mail Spam Filtering**

Gmail is using Bayesian Models to filter the mails by reading or understanding the context of mail.

4. **Bio monitoring**

   Bayesian Models are used to quantify the concentration of chemicals in blood and human tissues.

5. **Information Retrieval**

   Models can be used for retrieving information from the database.

7. Define causal networks. Give an example?

- Causal reasoning is a crucial element to how humans understand, explain, and make decisions about the world.
- Causal AI means automating causal reasoning with machine learning.
- Today's learning machines have superhuman prediction ability but aren't particularly good at causal reasoning, even when we train them on obscenely large amounts of data.
- In this book, you will learn how to write algorithms that capture causal reasoning in the context of machine learning and automated data science.

8. Define exact inference. Give an example?

- It is the term used when inference is performed exactly (subject to standard numerical rounding errors).
- Exact inference is applicable to a large range of problems, but may not be possible when combinations/paths get large.

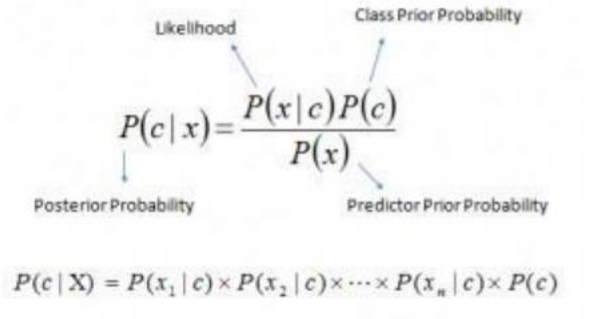
9. What is approximate Inference? Give an example?

- Wider class of problems.
- Non deterministic.
- No guarantee of correct answer.

10. Define Naïve Bayes model.

- It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.
- For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter.
- Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

- Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.
    - Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c).
    - Look at the equation below:



Above,

- P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

11. List out some leading causes of uncertainty to occur in the real world.

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors.
3. Equipment fault.
4. Temperature variation.
5. Climate change.

12. In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

13. Two students A and B have registered for a certain course. Student A attends the class 80% of the time. Students B attends the class 60% of the time. Assuming their absences are independent, what is the probability that both neither show up to class on any given day? Apr 2024

Step 1: Given information

A and B, two individuals, are both enrolled in the same course. Student A attends class 80\% of the time, and student B attends class 60\% of the time, and the two individuals' absences are independent.

Step 2: Defining events

Let:A= The event that student A attends class on a given day

B= The event that student B attends class on a given day

Thus, the given information can be summarized as follows:

$P(A)=0.80$

$P(B)=0.60$

Step 3: (a) Computing the probability in part a

The Additional rule of probability results in the probability of appearance of either of the events A or B. Mathematically, it is given by:

$P(A \cup B)=P(A)+P(B)-P(A \cap B)$

It is given that the absences of the two students are independent. This indicates that the presences of the two students are also independent. Thus,

$P(A \cap B)=P(A) \times P(B)$

P(atleast one of the two students will be in class)

$=P(A \cup B)$

$=P(A)+P(B)-P(A \cap B)$

$=P(A)+P(B)-P(A) \times P(B)$

Substituting the values,

P(atleast one of the two students will be in class)

$=0.80+0.60-0.80 \times 0.60$

$=1.4-0.48$

$=0.92$

Therefore, the required probability is 0.92.

Step 4: (b) Computing the probability in part b

The Conditional probability of an event A given some other event B has appeared is given by:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

By the definition of conditional probability,

If at least one of the two students will be in class on a given day, the probability that A is in class that day is obtained as:

$$P[A|(A \cup B)] = \frac{P[A \cap (A \cup B)]}{P(A \cup B)}$$
$$= \frac{P(A)}{P(A \cup B)} \quad \{A \subseteq (A \cup B)\}$$
$$= \frac{0.80}{0.92}$$
$$\approx 0.8696$$

Therefore, the required probability is approximately 0.8696.

14. What are Casual Networks? [Apr 2024]

- A directed network which illustrates the causal dependencies of all the components in the network. A causal relationship exists when one variable in a data set has a direct influence on another variable.

15. Why does uncertainty arise? [Nov 2023]

- Uncertainty in AI Perception: AI systems perceive their environment through sensors and cameras, which can be subject to noise, occlusion, or other forms of interference.
- This can lead to uncertainty in the accuracy of the data used to train AI models or the effectiveness of AI systems in real-world applications.

16. What is Bayes rule? Mention its use. [Nov 2023]

- Bayes' Theorem in AI, also known as Bayes' rule or Bayes' law, is a fundamental concept in probability theory and statistics. It provides a way to update our beliefs or the probability of an event occurring based on new evidence or information.

PART B

1. Explain in detail about Acting under uncertainty.

1. Uncertainty:

- Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.
- So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change.

2. Explain in detail about Bayesian inference.

Discuss the exact inference in Bayesian networks. [Nov 2023]

How does Exact interface differ from approximate Inference? Give example. [Apr 2024]

Bayes' theorem in Artificial intelligence

Bayes' theorem:

- Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.
- In probability theory, it relates the conditional probability and marginal probabilities of two random events.

- Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.
- It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).
- Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example:

- If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.
- Bayes' theorem can be derived using product rule and conditional probability of event A with known event B: As from product rule we can write:

P(A ∧ B)= P(A|B) P(B) or

Similarly, the probability of event B with known event A:

P(A ∧ B)= P(B|A) P(A)

- Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

- The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.
- It shows the simple relationship between joint and conditional probabilities. Here,
- P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.
- P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.
- P(A) is called the **prior probability**, probability of hypothesis before considering the evidence P(B) is called **marginal probability**, pure probability of an evidence.
- In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes'

rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,. , A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

- Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A).
- This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one.
- Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$\mathbf{P(cause|effect)} = \frac{P(effect|cause)\,P(cause)}{P(effect)}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck? Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8 P(b) = 1/30000 P(a)= .02

$$\mathbf{P(b|a)} = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

Question:

From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$\mathbf{P(king|face)} = \frac{P(\text{Face}|\text{king}) * P(\text{King})}{P(\text{Face})} \quad \text{.....(i)}$$

P(king): probability that the card is King= 4/52= 1/13 P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1 Putting all values in equation (i) we will get:

$$P(\text{king}|\text{face}) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = \mathbf{1/3,}$$ **it is a probability that a face card is a king card.**

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

3. Explain in detail about Probabilistic reasoning.

Probabilistic reasoning:

- Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

- We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
- In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

Need of probabilistic reasoning in AI:

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**
- As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability:

- Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$0 \le P(A) \le 1$, where P(A) is the probability of an event A. P(A) = 0, indicates total uncertainty in an event A.

P(A) =1, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- P(¬A) = probability of a not happening event.
- P(¬A) + P(A) = 1.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects in the real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred,

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

"the probability of A under the conditions of B", it can be written as:

Where P(*A*∧*B*)= Joint probability of a and B P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:

$$P(B|A) = \frac{P(A \wedge B)}{P(A)}$$

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of **P(A∧*B*) by P(B)**.

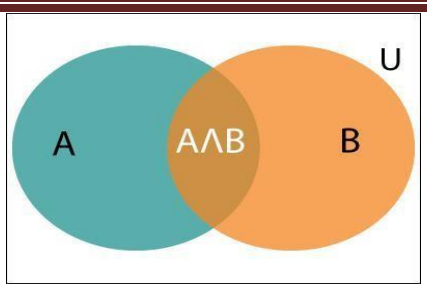


Fig.5.1 Venn Diagram

Example:

- In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

Hence, 57% are the students who like English also like Mathematics.

4. Explain in detail about Bayesian networks or Belief networks.

What is Bayesian network? Explain the method for constructing Bayesian networks. [Nov 2023]

Bayesian Belief Network in artificial intelligence

- Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:
- "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."
- It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.
- Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and

anomaly detection.

- Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- Directed Acyclic Graph
- Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:

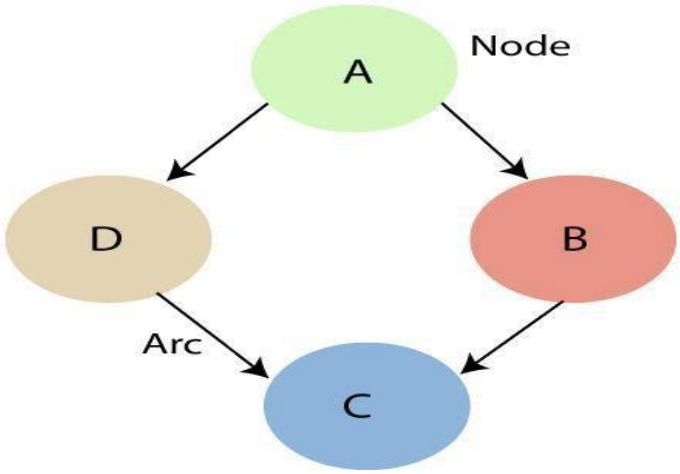


Fig.5.2 Influence Diagram

- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
    - In fig.5.2, A, B, C, and D are random variables represented by the nodes

of the network graph.

- If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
- Node C is independent of node A.

Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a directed acyclic graph or DAG

The Bayesian network has mainly two components:

- Causal Component
- Actual numbers

Each node in the Bayesian network has condition probability distribution

$P(X_i \mid Parent(X_i))$, which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability.

So let's first understand the joint probability distribution:

Joint probability distribution:

If we have variables x1, x2, x3, , xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

$P[x1, x2, x3, , xn]$, it can be written as the following way in terms of the joint probability distribution.

$= P[x1 \mid x2, x3, \ldots, xn]P[x2, x3, , xn]$

$= P[x1 \mid x2, x3, \ldots, xn]P[x2 \mid x3, \ldots, xn]. \; P[xn-1 \mid xn]P[xn].$

In general for each variable Xi, we can write the equation as:

$P(Xi \mid Xi-1, \ldots\ldots, X1) = P(Xi \mid Parents(Xi))$

Explanation of Bayesian network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

Example:

- Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes.
- Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm.
- David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too.
- On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains 2^K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

List of all events occurring in this network:

- Burglary (B)

- Earthquake(E)
- Alarm(A)
- David Calls(D)
- Sophia calls(S)

We can write the events of problem statement in the form of probability:

P[D, S, A, B, E], can rewrite the above probability statement using joint probability distribution:

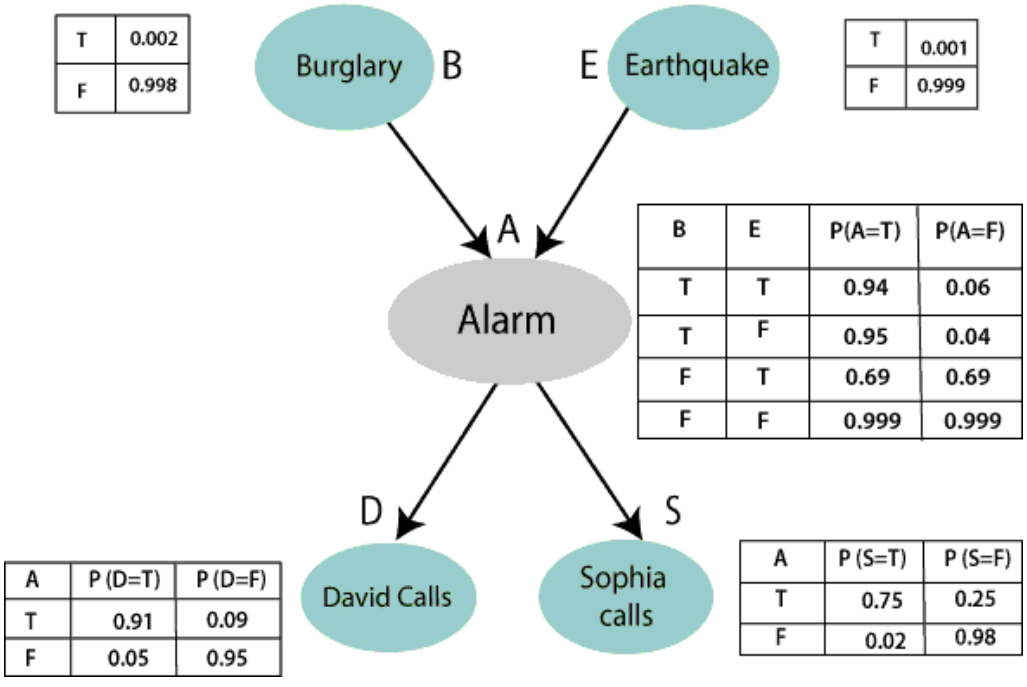
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]

= P [D| A]. P [S| A, B, E]. P[A, B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B, E]

= P[D | A]. P[S | A]. P[A| B, E]. P[B |E]. P[E]



T	0.002
F	0.998

T	0.001
F	0.999

B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Fig. 5.3 Bayesian Network

Let's take the observed probability for the Burglary and earthquake component:

P(B= True) = 0.002, which is the probability of burglary.

P(B= False)= 0.998, which is the probability of no burglary.

P(E= True)= 0.001, which is the probability of a minor earthquake

P(E= False)= 0.999, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

Conditional probability table for Alarm A:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

- The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

P(S, D, A, ¬B, ¬E) = P (S|A) *P (D|A)*P (A|¬B ^ ¬E) *P (¬B) *P (¬E).

= 0.75* 0.91* 0.001* 0.998*0.999

= 0.00068045.

Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

What are the ways to understand the semantics of Bayesian Networks?

[Nov 2023]

The semantics of Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

1. To understand the network as the representation of the Joint probability distribution. It is helpful to understand how to construct the network.
2. To understand the network as an encoding of a collection of conditional independence statements. It is helpful in designing inference procedure.

1. Inference in Bayesian Networks

1. Exact inference
2. Approximate inference

1. Exact inference:

- In exact inference, we analytically compute the conditional probability distribution over the variables of interest.
- But sometimes, that's too hard to do, in which case we can use approximation techniques based on statistical sampling
- Given a Bayesian network, what questions might we want to ask?
    - Conditional probability query: P(x | e)
    - Maximum a posteriori probability: What value of x maximizes P(x|e) ?
- General question: What's the whole probability distribution over variable X given evidence e, P(X |e)?
- In our discrete probability situation, the only way to answer a MAP query is to compute the probability of x given e for all possible values of x and see

which one is greatest

- So, in general, we'd like to be able to compute a whole probability distribution over some variable or variables X, given instantiations of a set of variables e

Using the joint distribution

- To answer any query involving a conjunction of variables, sum over the variables not involved in the query
- Given the joint distribution over the variables, we can easily answer any question about the value of a single variable by summing (or marginalizing) over the other variables.

$$\Pr(d) = \sum_{ABC} \Pr(a,b,c,d)$$
$$= \sum_{a \in \mathrm{dom}(A)} \sum_{b \in \mathrm{dom}(B)} \sum_{c \in \mathrm{dom}(C)} \Pr(A = a \wedge B = b \wedge C = c)$$

- So, in a domain with four variables, A, B, C, and D, the probability that variable D has value d is the sum over all possible combinations of values of the other three variables of the joint probability of all four values.
- This is exactly the same as the procedure we went through in the last lecture, where to compute the probability of cavity, we added up the probability of cavity and toothache and the probability of cavity and not toothache.
- In general, we'll use the first notation, with a single summation indexed by a list of variable names, and a joint probability expression that mentions values of those variables. But here we can see the completely written-out definition, just so we all know what the shorthand is supposed to mean.

$$\Pr(d \mid b) = \frac{\Pr(b,d)}{\Pr(b)} = \frac{\sum_{AC} \Pr(a,b,c,d)}{\sum_{ACD} \Pr(a,b,c,d)}$$

- To compute a conditional probability, we reduce it to a ratio of conjunctive queries using the definition of conditional probability, and then answer each of those queries by marginalizing out the variables not mentioned.

- In the numerator, here, you can see that we're only summing over variables A and C, because b and d are instantiated in the query.

Simple Case

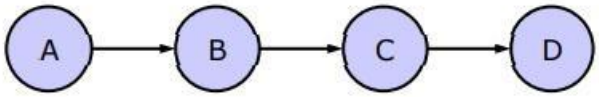
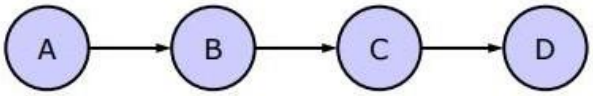


Fig.5.4 Simple Case

- We're going to learn a general purpose algorithm for answering these joint queries fairly efficiently. We'll start by looking at a very simple case to build up our intuitions, then we'll write down the algorithm, then we'll apply it to a more complex case.
- Here's our very simple case. It's a bayes net with four nodes, arranged in a chain.

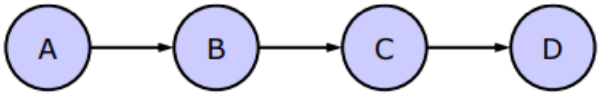


$$\Pr(d) = \sum_{ABC} \Pr(a,b,c,d)$$
$$= \sum_{ABC} \Pr(d \mid c)\Pr(c \mid b)\Pr(b \mid a)\Pr(a)$$
$$= \sum_{C}\sum_{B}\sum_{A} \Pr(d \mid c)\Pr(c \mid b)\Pr(b \mid a)\Pr(a)$$
$$= \sum_{C} \Pr(d \mid c) \sum_{B} \Pr(c \mid b) \sum_{A} \Pr(b \mid a)\Pr(a)$$

- So, we know from before that the probability that variable D has some value little d is the sum over A, B, and C of the joint distribution, with d fixed.
- Now, using the chain rule of Bayesian networks, we can write down the joint probability as a product over the nodes of the probability of each node's value given the values of its parents. So, in this case, we get P(d|c) times P(c|b) times P(b|a) times P(a).
- This expression gives us a method for answering the query, given the conditional probabilities that are stored in the net. And this method can be applied directly to

any other bayes net. But there's a problem with it: it requires enumerating all possible combinations of assignments to A, B, and C, and then, for each one, multiplying the factors for each node. That's an enormous amount of work and we'd like to avoid it if at all possible.

- So, we'll try rewriting the expression into something that might be more efficient to evaluate. First, we can make our summation into three separate summations, one over each variable.
- Then, by distributivity of addition over multiplication, we can push the summations in, so that the sum over A includes all the terms that mention A, but no others, and so on. It's pretty clear that this expression is the same as the previous one in value, but it can be evaluated more efficiently. We're still, eventually, enumerating all assignments to the three variables, but we're doing somewhat fewer multiplications than before. So this is still not completely satisfactory.



$$\Pr(d) = \sum_C \Pr(d \mid c) \sum_B \Pr(c \mid b) \underbrace{\sum_A \Pr(b \mid a) \Pr(a)}_{\begin{bmatrix} \Pr(b_1 \mid a_1)\Pr(a_1) & \Pr(b_1 \mid a_2)\Pr(a_2) \\ \Pr(b_2 \mid a_1)\Pr(a_1) & \Pr(b_2 \mid a_2)\Pr(a_2) \end{bmatrix}}$$

- If you look, for a minute, at the terms inside the summation over A, you'll see that we're doing these multiplications over for each value of C, which isn't necessary, because they're independent of C. Our idea, here, is to do the multiplications once and store them for later use. So, first, for each value of A and B, we can compute the product, generating a two dimensional matrix.

$$\Pr(d) = \sum_C \Pr(d \mid c) \sum_B \Pr(c \mid b) \underbrace{\sum_A \Pr(b \mid a) \Pr(a)}_{\begin{bmatrix} \sum_A \Pr(b_1 \mid a)\Pr(a) \\ \sum_A \Pr(b_2 \mid a)\Pr(a) \end{bmatrix}}$$

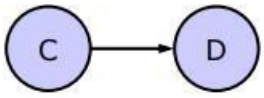
- Then, we can sum over the rows of the matrix, yielding one value of the sum for each possible value of b.

$$\Pr(d) = \sum_C \Pr(d \mid c) \sum_B \Pr(c \mid b) \underbrace{\sum_A \Pr(b \mid a) \Pr(a)}_{f_1(b)}$$

We'll call this set of values, which depends on b, f1 of b.

$$\Pr(d) = \sum_C \Pr(d \mid c) \underbrace{\sum_B \Pr(c \mid b) f_1(b)}_{f_2(c)}$$

- Now, we can substitute f1 of b in for the sum over A in our previous expression. And, effectively, we can remove node A from our diagram. Now, we express the contribution of b, which takes the contribution of a into account, as f_1 of b.
- We can continue the process in basically the same way. We can look at the summation over b and see that the only other variable it involves is c. We can summarize those products as a set of factors, one for each value of c. We'll call those factors f_2 of c.



$$\Pr(d) = \sum_C \Pr(d \mid c) f_2(c)$$

- We substitute f_2 of c into the formula, remove node b from the diagram, and now we're down to a simple expression in which d is known and we have to sum over values of c.

Variable Elimination Algorithm

Given a Bayesian network, and an elimination order for the non-query variables , compute

$$\sum_{X_1}\sum_{X_2}\cdots\sum_{X_m}\prod_{j}\Pr(x_j \mid Pa(x_j))$$

For i = m downto 1

- remove all the factors that mention Xi
- multiply those factors, getting a value for each combination of mentioned variables
- sum over Xi
- put this new factor into the factor set

- That was a simple special case. Now we can look at the algorithm in the general case. Let's assume that we're given a Bayesian network and an ordering on the variables that aren't fixed in the query. We'll come back later to the question of the influence of the order, and how we might find a good one.
- We can express the probability of the query variables as a sum over each value of each of the non- query variables of a product over each node in the network, of the probability that that variable has the given value given the values of its parents.
- So, we'll eliminate the variables from the inside out. Starting with variable Xm and finishing with variable X1.
- To eliminate variable Xi, we start by gathering up all of the factors that mention Xi, and removing them from our set of factors. Let's say there are k such factors.
- Now, we make a k+1 dimensional table, indexed by Xi as well as each of the other variables that is mentioned in our set of factors.
- We then sum the table over the Xi dimension, resulting in a k-dimensional table. This table is our new factor, and we put a term for it back into our set of factors. Once we've eliminated all the summations, we have the desired value.

One more example

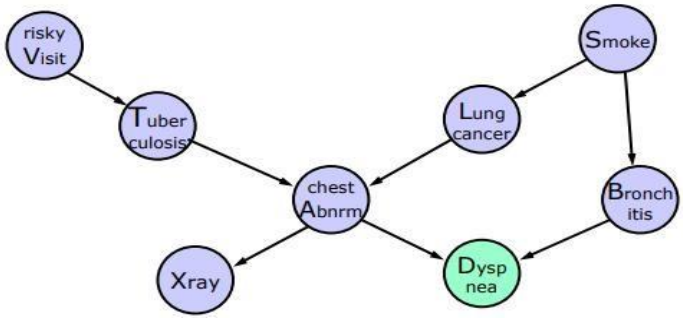


Fig.5.5 Variable Elimination - Example

$$\Pr(d) = \sum_{A,B,L,T,S,X,V} \frac{\Pr(d \mid a,b)\Pr(a \mid t,l)\Pr(b \mid s)\Pr(l \mid s)\Pr(s)}{\Pr(x \mid a)\Pr(t \mid v)\Pr(v)}$$

$$\Pr(d) = \sum_{A,B,L,T,S,X} \frac{\Pr(d \mid a,b)\Pr(a \mid t,l)\Pr(b \mid s)\Pr(l \mid s)\Pr(s)}{\Pr(x \mid a)\underbrace{\sum_{V}\Pr(t \mid v)\Pr(v)}_{f_1(t)}}$$

$$\Pr(d) = \sum_{A,B,L,T,S,X} \frac{\Pr(d \mid a,b)\Pr(a \mid t,l)\Pr(b \mid s)\Pr(l \mid s)\Pr(s)}{\Pr(x \mid a) f_1(t)}$$

$$\Pr(d) = \sum_{A,B,L,T,S} \frac{\Pr(d \mid a,b)\Pr(a \mid t,l)\Pr(b \mid s)\Pr(l \mid s)\Pr(s) f_1(t)}{\underbrace{\sum_{X}\Pr(x \mid a)}_{1}}$$

$$\Pr(d) = \sum_{A,B,L,T,S}\Pr(d \mid a,b)\Pr(a \mid t,l)\Pr(b \mid s)\Pr(l \mid s)\Pr(s) f_1(t)$$

$$\Pr(d) = \sum_{A,B,L,T} \Pr(d \mid a,b)\Pr(a \mid t,l) f_1(t)\underbrace{\sum_{S}\Pr(b \mid s)\Pr(l \mid s)\Pr(s)}$$

$$\Pr(d) = \sum_{A,B,L,T} \Pr(d \mid a,b)\Pr(a \mid t,l) f_1(t)\underbrace{\sum_{S}\Pr(b \mid s)\Pr(l \mid s)\Pr(s)}_{f_2(b,l)}$$

$$\Pr(d) = \sum_{A,B,L,T}\Pr(d \mid a,b)\Pr(a \mid t,l) f_1(t) f_2(b,l)$$

$$\Pr(d) = \sum_{A,B,L} \Pr(d \mid a,b) f_1(b,l) \underbrace{\sum_{T} \Pr(a \mid t,l) f_1(t)}_{f_3(a,l)}$$

$$\Pr(d) = \sum_{A,B,L} \Pr(d \mid a,b) f_2(b,l) f_3(a,l)$$

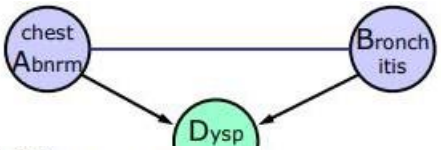
- Here's a more complicated example, to illustrate the variable elimination algorithm in a more general case. We have this big network that encodes a domain for diagnosing lung disease. (Dyspnea, as I understand it, is shortness of breath).
- We'll do variable elimination on this graph using elimination order A, B, L, T, S, X, V.
- So, we start by eliminating V. We gather the two terms that mention V and see that they also involve variable T. So, we compute the product for each value of T, and summarize those in the factor f1 of T.
- Now we can substitute that factor in for the summation, and remove the node from the network.
- The next variable to be eliminated is X. There is actually only one term involving X, and it also involves variable A. So, for each value of A, we compute the sum over X of P(x|a). But wait! We know what this value is! If we fix a and sum over x, these probabilities have to add up to 1.
- So, rather than adding another factor to our expression, we can just remove the whole sum. In general, the only nodes that will have an influence on the probability of D are its ancestors.
- Now, it's time to eliminate S. We find that there are three terms involving S, and we gather them into the sum. These three terms involve two other variables, B and L. So we have to make a factor that specifies, for each value of B and L, the value of the sum of products.
- We'll call that factor f_2 of b and l.
- Now we can substitute that factor back into our expression. We can also eliminate node S. But in eliminating S, we've added a direct dependency between L and B (they used to be dependent via S, but now the dependency is encoded explicitly in f2(b). We'll show that in the graph by drawing a line between the two nodes. It's not exactly a standard directed conditional dependence, but it's still useful to show that they're

coupled.

- Now we eliminate T. It involves two terms, which themselves involve variables A and L. So we make a new factor f3 of A and L.
- We can substitute in that factor and eliminate T. We're getting close!

$$\Pr(d) = \sum_{A,B} \Pr(d \mid a,b) \underbrace{\sum_{L} f_2(b,l) f_3(a,l)}_{f_4(a,b)}$$

- Next we eliminate L. It involves these two factors, which depend on variables A and B. So we make a new factor, f4 of A and B, and substitute it in. We remove node L, but couple A and B.



$$\Pr(d) = \sum_{A,B} \underbrace{\sum_{B} \Pr(d \mid a,b) f_4(a,b)}_{f_5(a)}$$

- At this point, we could just do the summations over A and B and be done. But to finish out the algorithm the way a computer would, it's time to eliminate variable B.
- It involves both of our remaining terms, and it seems to depend on variables A and D. However, in this case, we're interested in the probability of a particular value, little d of D, and so the variable d is instantiated. Thus, we can treat it as a constant in this expression, and we only need to generate a factor over a, which we'll call f5 of a. And we can now, in some sense, remove D from our network as well (because we've already factored it into our answer).
- Finally, to get the probability that variable D has value little d, we simply sum factor f5 over all values of a. Yay! We did it.

$$\Pr(d) = \sum_{A} f_5(a)$$

Properties of Variable Elimination

Let's see how the variable elimination algorithm performs, both in theory and in practice.

- Time is exponential in size of largest factor
- Bad elimination order can generate huge factors
- NP Hard to find the best elimination order
- Even the best elimination order may generate large factors
- There are reasonable heuristics for picking an elimination order (such as choosing the variable that results in the smallest next factor)
- Inference in polytrees (nets with no cycles) is linear in size of the network (the largest CPT)
- Many problems with very large nets have only small factors, and thus efficient inference

- First of all, it's pretty easy to see that it runs in time exponential in the number of variables involved in the largest factor. Creating a factor with k variables involves making a k+1 dimensional table. If you have b values per variable, that's a table of size b^(k+1). To make each entry, you have to multiply at most n numbers, where n is the number of nodes. We have to do this for each variable to be eliminated (which is usually close to n). So we have something like time = O(n^2 b^k).
- How big the factors are depends on the elimination order. You'll see in one of the recitation exercises just how dramatic the difference in factor sizes can be. A bad elimination order can generate huge factors.
- So, we'd like to use the elimination order that generates the smallest factors. Unfortunately, it turns out to be NP hard to find the best elimination order.
- At least, there are some fairly reasonable heuristics for choosing an elimination order. It's usually done dynamically. So, rather than fixing the elimination order in advance, as we suggested in the algorithm description, you can pick the next variable to be eliminated depending on the situation. In particular, one reasonable heuristic is to pick the variable to eliminate next that will result in the smallest factor. This greedy approach won't always be optimal, but it's not usually too bad.
- There is one case where Bayes net inference in general, and the variable elimination

algorithm in particular is fairly efficient, and that's when the network is a polytree. A polytree is a network with no cycles. That is, a network in which, for any two nodes, there is only one path between them. In a polytree, inference is linear in the size of the network, where the size of the network is defined to be the size of the largest conditional probability table (or exponential in the maximum number of parents of any node). In a polytree, the optimal elimination order is to start at the root nodes, and work downwards, always eliminating a variable that no longer has any parents. In doing so, we never introduce additional connections into the network.

- So, inference in polytrees is efficient, and even in many large non-polytree networks, it's possible to keep the factors small, and therefore to do inference relatively efficiently.
- When the network is such that the factors are, of necessity, large, we'll have to turn to a different class of methods.

2. Approximate inference:

Sampling

To get approximate answer we can do stochastic simulation (sampling).

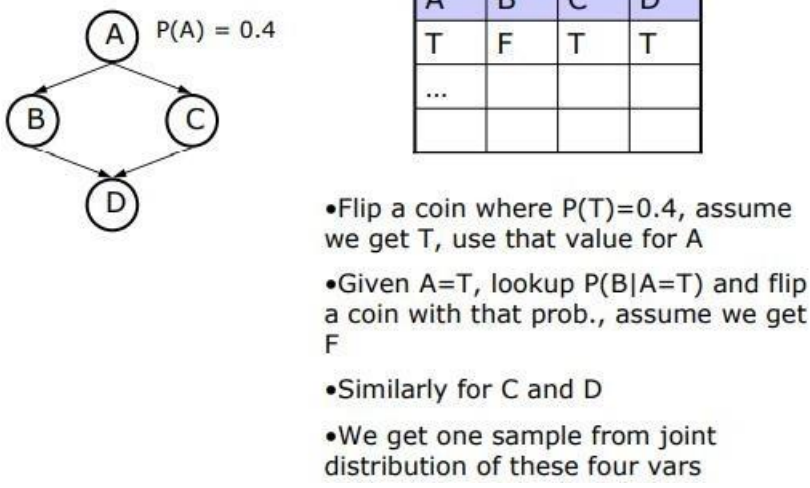


Fig.5.6 Approximate Inference

- Another strategy, which is a theme that comes up also more and more in AI actually, is to say, well, we didn't really want the right answer anyway. Let's try to do an approximation. And you can also show that it's computationally hard to get an approximation that's within epsilon of the answer that you want, but again

that doesn't keep us from trying.

- So, the other thing that we can do is the stochastic simulation or sampling. In sampling, what we do is we look at the root nodes of our graph, and attached to this root node is some probability that A is going to be true, right? Maybe it's .4. So we flip a coin that comes up heads with probability .4 and see if we get true or false.
- We flip our coin, let's say, and we get true for A -- this time. And now, given the assignment of true to A, we look in the conditional probability table for B given A = true, and that gives us a probability for B.
- Now, we flip a coin with that probability. Say we get False. We enter that into the table.
- We do the same thing for C, and let's say we get True.
- Now, we look in the CPT for D given B and C, for the case where B is false and C is true, and we flip a coin with that probability, in order to get a value for D.
- So, there's one sample from the joint distribution of these four variables. And you can just keep doing this, all day and all night, and generate a big pile of samples, using that algorithm. And now you can ask various questions.

Estimate:

P*(D|A) = #D,A / #A

- Let's say you want to know the probability of D given A. How would you answer -- given all the examples -- what would you do to compute the probability of D given A? You would just count. You'd count the number of cases in which A and D were true, and you'd divide that by the number of cases in
- which A was true, and that would give you an unbiased estimate of the probability of D given A. The more samples, the more confidence you'd have that the estimated probability is close to the true one.

Estimation

- Some probabilities are easier than others to estimate
- In generating the table, the rare events will not be well represented
- P(Disease| spots-on-your-tongue, sore toe)
- If spots-on-your-tongue and sore toe are not root nodes, you would generate a

huge table but the cases of interest would be very sparse in the table

- Importance sampling lets you focus on the set of cases that are important to answering your question
- It's going to turn out that some probabilities are easier than other ones to estimate.
- Exactly because of the process we're using to generate the samples, the majority of them will be the typical cases. Oh, it's someone with a cold, someone with a cold, someone with a cold, someone with a cold, someone with a cold, someone with malaria, someone with a cold, someone with a cold. So the rare results are not going to come up very often. And so doing this sampling naively can make it really hard to estimate the probability of a rare event. If it's something that happens one in ten thousand times, well, you know for sure you're going to need, some number of tens of thousands of samples to get even a reasonable estimate of that probability.
- Imagine that you want to estimate the probability of some disease given -- oh, I don't know -- spots on your tongue and a sore toe. Somebody walks in and they have a really peculiar set of symptoms, and you want to know what's the probability that they have some disease.
- Well, if the symptoms are root nodes, it's easy. If the symptoms were root nodes, you could just assign the root nodes to have their observed values and then simulate the rest of the network as before.
- But if the symptoms aren't root nodes then if you do naïve sampling, you would generate a giant table of samples, and you'd have to go and look and say, gosh, how many cases do I have where somebody has spots on their tongue and a sore toe; and the answer would be, well, maybe zero or not very many.
- There's a technique called importance sampling, which allows you to draw examples from a distribution that's going to be more helpful and then reweight them so that you can still get an unbiased estimate of the desired conditional probability. It's a bit beyond the scope of this class to get into the details, but it's an important and effective idea.

Recitation Problem

- Do the variable elimination algorithm on the net below using the elimination order A,B,C (that is, eliminate node C first). In computing P(D=d), what factors do you get?
- What if you wanted to compute the whole marginal distribution P(D)?

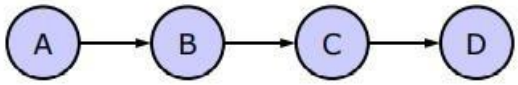


Fig.5.7 Recitation Problem Example

- Here's the network we started with. We used elimination order C, B, A (we eliminated A first). Now we're going to explore what happens when we eliminate the variables in the opposite order. First, work on the case we did, where we're trying to calculate the probability that node D takes on a particular value, little d. Remember that little d is a constant in this case. Now, do the case where we're trying to find the whole distribution over D, so we don't know a particular value for little d.

Another Recitation Problem

- Find an elimination order that keeps the factors small for the net below, or show that there is no such order.

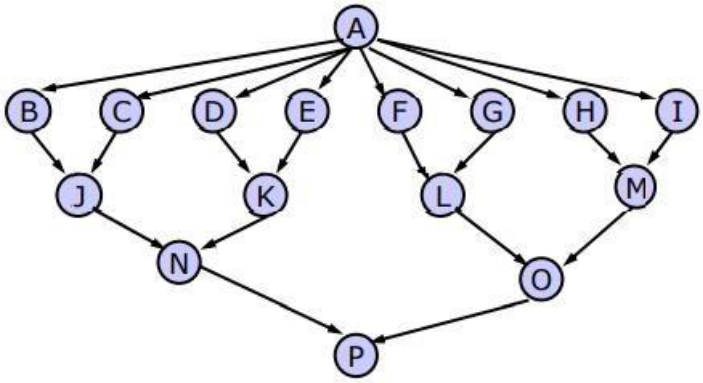


Fig.5.8 Recitation Problem Example

- Here's a pretty complicated graph. But notice that no node has more than 2 parents, so none of the CPTs are huge. The question is, is this graph hard for variable elimination? More concretely, can you find an elimination order that results only in fairly small factors? Is there an elimination order that generates a huge factor?

The Last Recitation Problem

- Bayesian networks (or related models) are often used in computer vision, but they almost always require sampling. What happens when you try to do variable elimination on a model like the grid below?

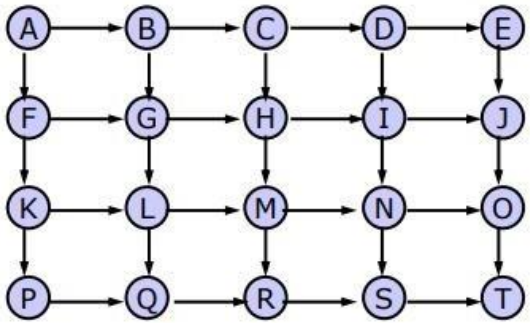


Fig.5.9 Recitation Problem Example

6. Explain in detail about Casual Networks.

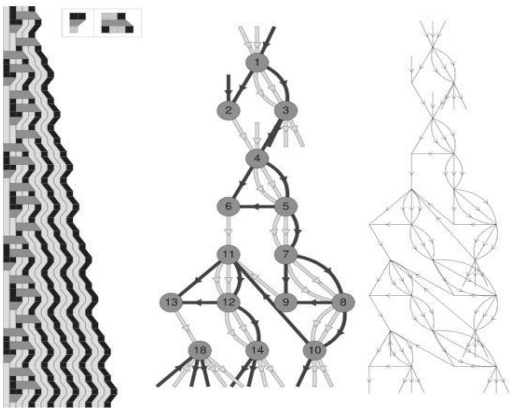


Fig.5.10 Casual Networks

- A causal network is an acyclic digraph arising from an evolution of a substitution system, and representing its history. The illustration above shows a causal network corresponding to the rules

$\{B\,B \to A,\ A\,A\,B \to B\,A\,A\,B\}$ (applied in a left-to-right scan) and initial condition $A\,B\,A\,A\,B$.

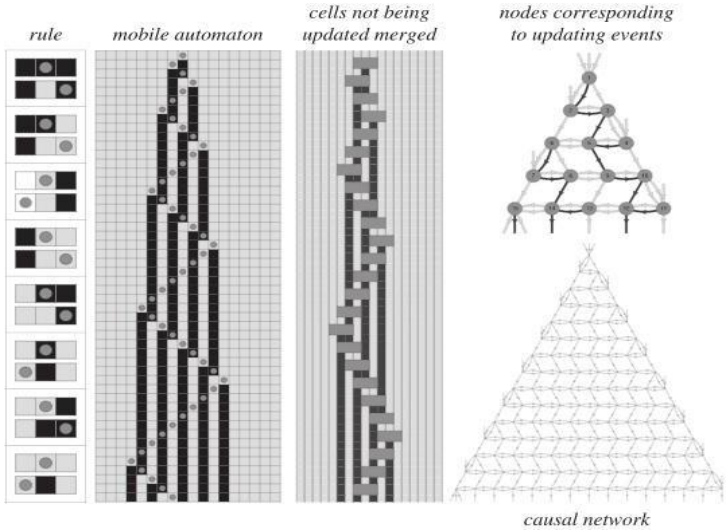


Fig.5.11 Casual Network

- The figure5.11 shows the procedure for diagrammatically creating a causal network from a mobile automaton.
- In an evolution of a multiway system, each substitution event is a vertex in a causal network. Two events which are related by causal dependence, meaning one occurs just before the other, have an edge between the corresponding vertices in the causal network. More precisely, the edge is a directed edge leading from the past event to the future event.
- Some causal networks are independent of the choice of evolution, and these are called causally invariant.

7. We have a bag of three biased coins a,b, and c with probabilities of coming up heads of 30%,60%, and 75% respectively. One coin is drawn randomly from the bag (with equal likelihood of drawing each of the three coins) and then the coin is flipped three times to generate the outcomes X1,X2, and X3.

(i) Draw the Bayesian network corresponding to this setup and define the necessary Conditinal Probability Tables.

(ii) Calculate which coin was most likely to have been drawn from the bag if the observed flips come out heads twice and tails once. [Apr 2024]

a. With the random variable C denoting which coin $\{a, b, c\}$ we drew, the network has C at the root and X_1, X_2, and X_3 as children.

The CPT for C is:

C	$P(C)$
a	1/3
b	1/3
c	1/3

The CPT for X_i given C are the same, and equal to:

C	X_1	$P(C)$
a	*heads*	0.2
b	*heads*	0.6
c	*heads*	0.8

b. The coin most likely to have been drawn from the bag given this sequence is the value of C with greatest posterior probability $P(C|2 \text{ heads}, 1 \text{ tails})$. Now,

$$\begin{aligned} P(C|2 \text{ heads}, 1 \text{ tails}) &= P(2 \text{ heads}, 1 \text{ tails}|C)P(C)/P(2 \text{ heads}, 1 \text{ tails}) \\ &\propto P(2 \text{ heads}, 1 \text{ tails}|C)P(C) \\ &\propto P(2 \text{ heads}, 1 \text{ tails}|C) \end{aligned}$$

where in the second line we observe that the constant of proportionality $1/P(2 \text{ heads}, 1 \text{ tails})$ is independent of C, and in the last we observe that $P(C)$ is also independent of the value of C since it is, by hypothesis, equal to 1/3.

From the Bayesian network we can see that X_1, X_2, and X_3 are conditionally independent given C, so for example

$$\begin{aligned} &P(X_1 = tails, X_2 = heads, X_3 = heads|C = a) \\ &= P(X_1 = tails|C = a)P(X_2 = heads|C = a)P(X_3 = heads|C = a) \\ &= 0.8 \times 0.2 \times 0.2 = 0.032 \end{aligned}$$

Note that since the CPTs for each coin are the same, we would get the same probability above for any ordering of 2 heads and 1 tails. Since there are three such orderings, we have

$$P(2heads, 1tails|C = a) = 3 \times 0.032 = 0.096.$$

Similar calculations to the above find that

$$\begin{aligned} P(2heads, 1tails|C = b) &= 0.432 \\ P(2heads, 1tails|C = c) &= 0.384 \end{aligned}$$

showing that coin b is most likely to have been drawn.

Alternatively, one could directly compute the value of $P(C|2 \text{ heads}, 1 \text{ tails})$.

The traveling salesperson problem can be solved with the Minimum Spanning Tree heuristic, which estimates the cost of completing a tour, given that a partial tour has already been constructed. The MST cost of a set of cities is the smallest sum of the link costs of any tree connects all the cities

(i) show how this heuristic can be derived from a relaxed version of the TSP

(ii) Show that the MST heuritis dominates straight line distance

(iii) Write a problem generator for instances of the TSP where cities are represented by random points in the unit square.

(iv) Find an efficient algorithm for constructing the MST and use it with a graph search to solve instances of the TSP. [Apr 2024]

- If we relax the constraint for TSP such that each city can be visited more than one time (that is, there may be some cities visited more than one time) and the cost for repeated edges are not count, we can get a solution for MST problem) Show how this heuristic dominates straight-line distance.
- MST dominates straight-line distance because the shortest distance from any two cities u, v is their straight-line distance. Moreover, letV ,´ E GMST ¿) be the solution of MST problem given the instance G. The cost from u to v for MST will be equal to the straight-line distance only if (u,v)ϵ´ E
- . Thus, MST heuristic dominates straight-line distance because it is based on the minimum connection distance, making it an admissible heuristic.

We have five planes: A<B,C,D, and E and two runways: international and domestic. We have like to scheduke a time slot and runway for each aircraft to either land or take off. We have four time slts: {1,2,3,4} for each runway, during which we can schedule a landing or take off of a plane. We must find an assignment that meets the following constraints.

Plain B has lost an engine and must land in time slot1

Plain D can only arrive at the airport to land during or after time slot 3

Plane A is running low on fuel but can last until at most time slot 2

Plane D must land before plane C takes off, because some passengers must transfer from D to C

No two aircrafts can reserve the same time slot for the same runway.

Brief on CSP and complete the formulation of this problem as a CSP in terms of variables, domains and constraints (both unary and binary). Constraints should be expressed implicitly using mathematical or logical notation rather than with words.

[Apr 2024]

(a) Complete the formulation of this problem as a CSP in terms of variables, domains, and constraints (both unary and binary). Constraints should be expressed implicitly using mathematical or logical notation rather than with words. Make sure to specify variables, domains, and constraints.

Variables: A, B, C, D, E for each plane.
Domains: a tuple (*runway type, time slot*) for runway type $\in$ {international, domestic} and time slot $\in \{1,2,3,4\}$.
Constraints:

$$B[1] = 1$$

$$D[1] \geq 3$$

$$A[1] \leq 2$$

$$D[1] < C[1]$$

$$A \neq B \neq C \neq D \neq E$$

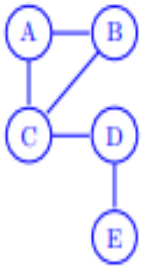
Note here we use B[1] to denote the second value of the tuple assigned to variable B. the time slot value, which is a number in $\{1,2,3,4\}$

For the following parts, we add the following two constraints:

- Planes A, B, and C cater to international flights and can only use the international runway.
- Planes D and E cater to domestic flights and can only use the domestic runway.

(b) The addition of the two constraints above alters the CSP. Specifically, the domain does not need to include the runway type since this information is carried by the variable, and the binary constraints have changed. Determine the new domain and complete the constraint graph for this problem given the original constraints and the two added ones.

Variables: A, B, C, D, E for each plane.
Domain: $\{1,2,3,4\}$
Constraint Graph:



Explanation of Constraints Graph: We can now encode the runway information into the identity of the variable, since each runway has more than enough time slots for the planes it serves. We represent the non-colliding time slot constraint as a binary constraint between the planes that use the same runways.

(c) What are the domains of the variables after enforcing arc consistency? Begin by enforcing unary constraints. (Cross out values that are no longer in the domain.)

Enforcing arc consistency with AC-3, we have the following domain as a result:

A	~~1~~	2	~~3~~	~~4~~
B	1	~~2~~	~~3~~	~~4~~
C	~~1~~	~~2~~	~~3~~	4
D	~~1~~	~~2~~	3	~~4~~
E	1	2	~~3~~	4

(explanation of process below)
Enforcing unary constraints (in an arbitrary order) first,

1. We cross out 2, 3, 4 from B's domain, adding arcs A → B and C → B to the queue.
2. We cross out 3, 4 from A's domain, adding arcs B → A and C → A to the queue.
3. We cross out 1, 2 from D's domain, adding arcs C → D and E → D to the queue.

Enforcing A → B, we cross out 1 from A's domain; add arcs B → A and C → A to the queue.
Enforcing C → B, we cross out 1 from C's domain; add arcs A → C, B → C, and D → C to the queue.
Enforcing B → A, no domain changes are necessary (all values remaining in B's domain have a consistent corresponding value in A's domain); no arcs are added.
Enforcing C → A, we cross out 2 from C's domain; add arcs A → C, B → C, and D → C to the queue.
Enforcing C → D, we cross out 3 from C's domain; add arcs A → C, B → C, and D → C to the queue.
Enforcing E → D, no domain changes are necessary.
Enforcing B → A, no domain changes are necessary.
Enforcing C → A, no domain changes are necessary.
Enforcing A → C, no domain changes are necessary.
Enforcing B → C, no domain changes are necessary.
Enforcing D → C, we cross out 4 from D's domain (there is no c in C's domain such that c > 4); add arcs C → D and E → D to the queue.
Enforcing A → C, no domain changes are necessary.
Enforcing B → C, no domain changes are necessary.
Enforcing D → C, no domain changes are necessary.
Enforcing C → D, no domain changes are necessary.
Enforcing E → D, we cross out 3 from E's domain; add arc D → E to the queue.
Enforcing D → E, no domain changes are necessary.
(phew!)

Note: For a general binary CSP, to enforce arc consistency before assigning any variables, you should add all arcs to the initial queue. For this problem, it can be easily seen that if there are no unary constraints, all the arcs will be consistent before any variable is assigned a value. As a result, we can start with the unary constraints and add arcs only for the related variables after enforcing the unary constraints.

(d) Arc-consistency can be rather expensive to enforce, and we believe that we can obtain faster solutions using only forward-checking on our variable assignments. Using the Minimum Remaining Values heuristic, perform backtracking search on the graph, breaking ties by picking lower values and characters first. List the (variable, assignment) pairs in the order they occur (including the assignments that are reverted upon reaching a dead end). Enforce unary constraints before starting the search.

List of (variable, assignment) pairs:

(You don't have to use this table)

A	1	2	3	4
B	1	2	3	4
C	1	2	3	4
D	1	2	3	4
E	1	2	3	4

Answer: (B, 1), (A, 2), (C, 3), (C, 4), (D, 3), (E, 1)