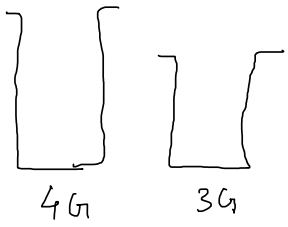
Water Jug Problem

Monday, January 31, 2022 8:56 AM

* AGENTS / ENVIRONMENT

* Water Jug Problem :→



Fill 4G Jug with 2G of water

I.S. (0,0)

(4,0) → (1,3) → (1,0) → (0,1) → (4,1) → (2,3)

(0,0) → (0,3) → (3,0) → (3,3) → (4,2) → (0,2) → (2,0)

(0,0) → (0,3) → (4,3) → (4,0) → (1,3) → (1,0) → (0,1) → (4,1) → (2,3)

Non optimum

Optimum soln → Intelligence

G.S.

* State space Problem

→ Steps to solve AI problems

1. State description
2. Define Initial state & Goal state
3. List all the actions required to solve the problem
4. Find the soln.

1. State
2. I.S. & G.S
3. Actions / Operators

* State discription :-

Ordered pair of two integers (x, y)

where x = 0, 1, 2, 3, 4 i.e. amount of water in 4G Jug

y = 0, 1, 2, 3 i.e. amount of water in 3G Jug

* Initial state = (0,0)

Goal state = (2, n)

L → R

* List of actions : [Condition action Rule]

1. Fill 3G jug: (x, y) → (x, 3), if (y < 3)

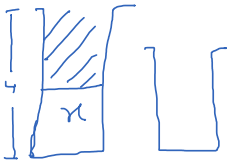
(2) Fill 4G jug $(x,y) \rightarrow (4,y)$, if $(x<4)$

(3) Empty 3G jug $(x,y) \rightarrow (x,0)$, if $(y>0)$

(4) Empty 4G jug $(x,y) \rightarrow (0,y)$, if $(x>0)$

(5) Transfer all water from 3G jug to 4G jug $\}$ $(x,y) \rightarrow (x+y,0)$ if $(x+y \leq 4)$ & $(y>0)$

(6) Transfer all the water from 4G jug to 3G jug $\}$ $(x,y) \rightarrow (0,x+y)$ if $(x+y \leq 3)$ & $(x>0)$

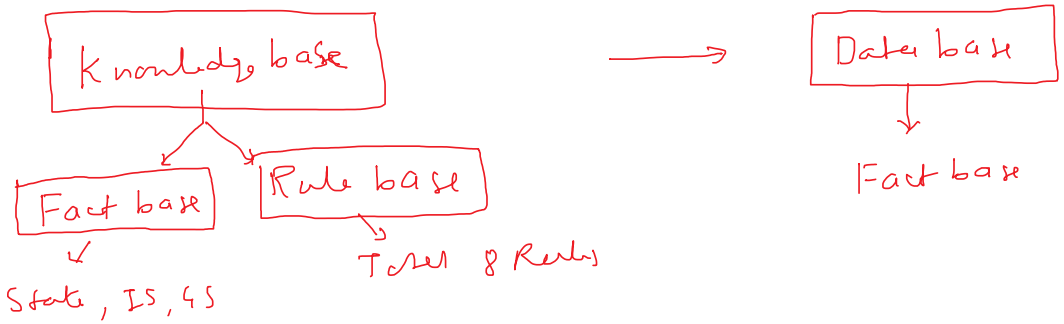


(7) Transfer some water from 3G jug to 4G jug until 4G jug is ~~full~~ full. $\}$ $(x,y) \rightarrow (4, y-(4-x))$ if $(x+y>4)$ & $(y>0)$

(1), (3)

3, 3

(8) Transfer some water from 4G jug to 3G jug until 3G jug is full $\}$ $(x,y) \rightarrow (x-(3-y), 3)$ if $(x+y>3)$ & $(x>0)$



* Solution

(0,0) ← I.S.

↓ Rule 1

(0,3)

↓ Rule 5

(3,0)

↓ Rule 1

(3,3)

↓ Rule 7

(4,2)

↓ Rule 4

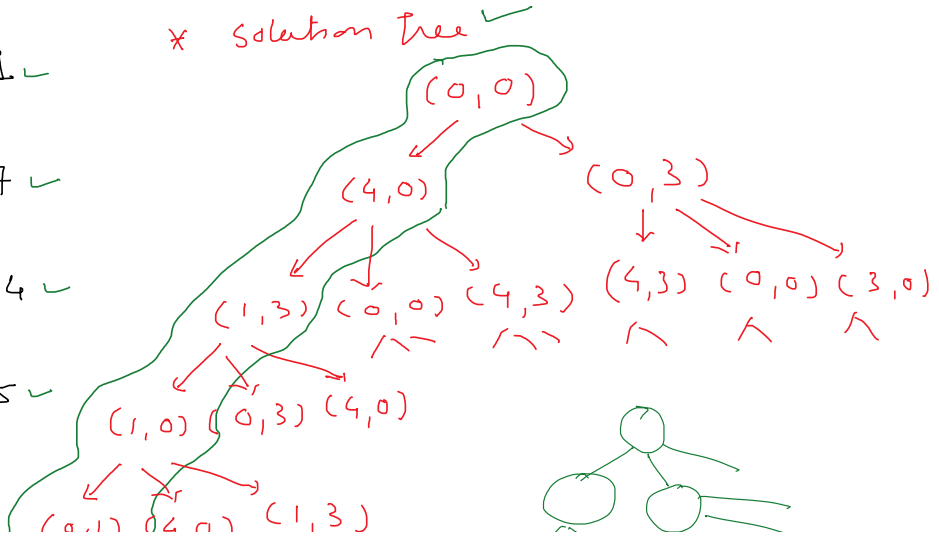
(0,2)

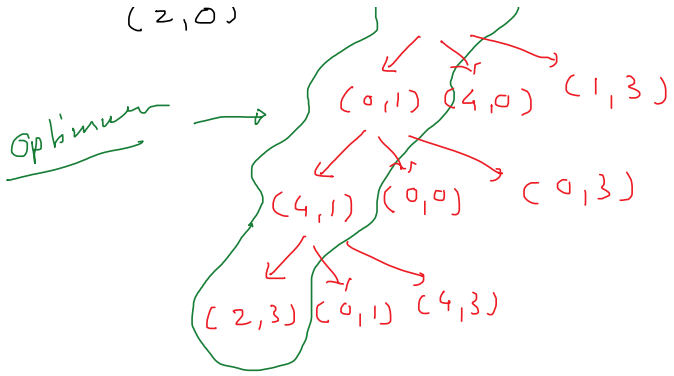
↓ Rule 5

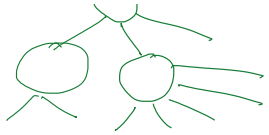
(2,0)

cost = 6

* Solution Tree







$x = 0, 1, 2, 3, 4$

$y = 0, 1, 2, 3$

* State Space : The no: of all possible states in a problem space.

↑ Water jug problem :-

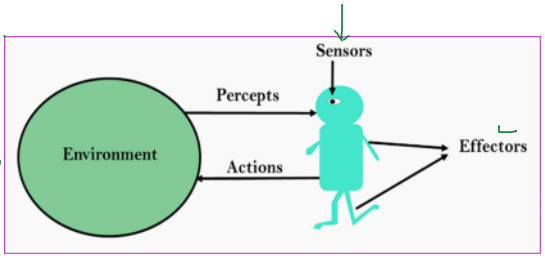
(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
(0,1)	(1,1)	↓	↓	↓
(0,2)	(1,2)	↓	↓	↓
(0,3)	(1,3)	↓	↓	↓

$= 20$

Agents

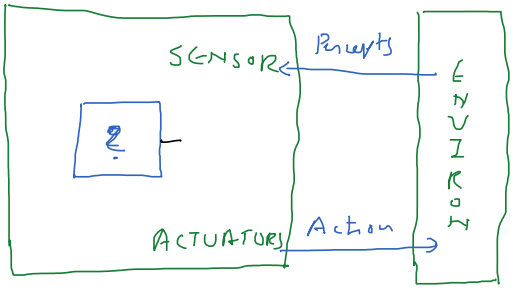
Monday, January 31, 2022 10:55 AM

IS → Agent



Domain →

AGENT



Goal

- Sensors
- Actuators / Effectors

An agent can be:

- Human-Agent:
  - Sensors : Eyes, Ears & Other Organs.
  - Actuators : Hand, Legs, Vocal tract.
- Robotic Agent:
  - Sensors : Cameras, Infrared range finder, NLP
  - Actuators : Various motors
- Software Agent:
  - Sensors : Keystrokes, File contents
  - Actuators : Screen

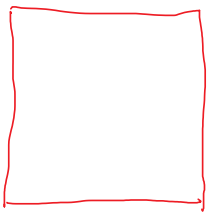
An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals.

* Types of Agents

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

Complexity of the environment

① Fully observable
② Partially observable

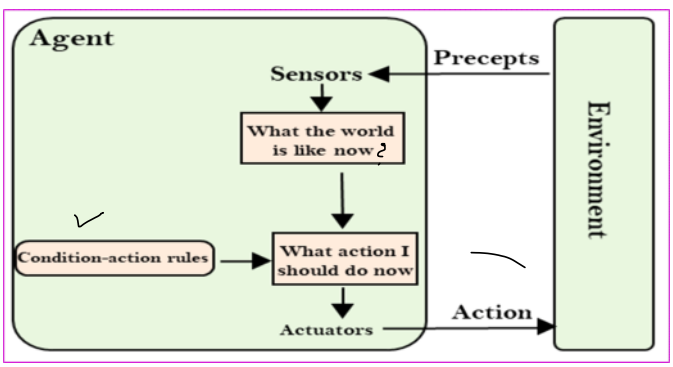


8x8

- 4 sensors (Giving you information about the tiles)
- 4 sensors (Not all four)

[water-jug problem]

* simple reflex agent



→ knowledge

→ state

?

(0,0) → Percept

↓ Rule 2

(0,3) → Percept

↓ Rule 5

↓ Rules

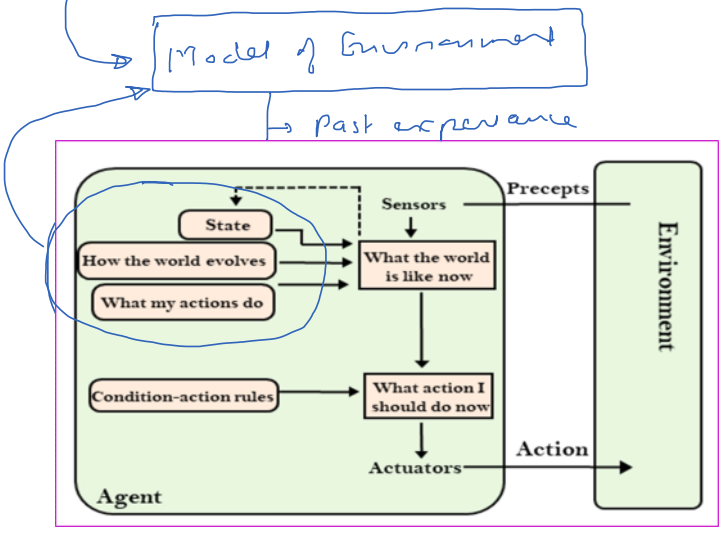
(3,0)

① Operates in Fully observable environment

② It can map percepts directly to the actions

* Model based Agent :-

① It is suitable for partially observable environment



State :-

→ How the world evolves ?

→ What my actions do ?

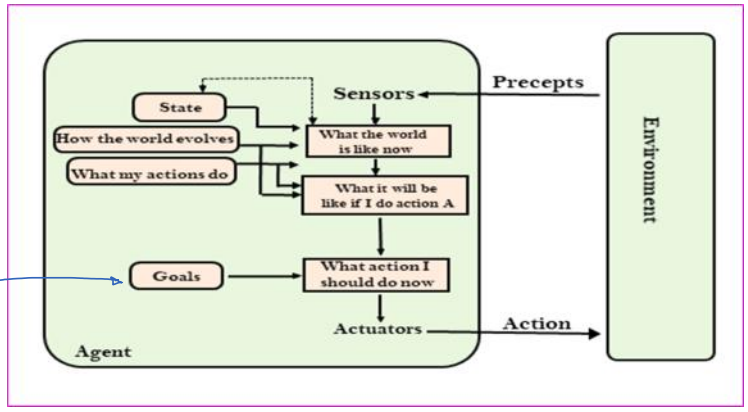
↓

(0,0)

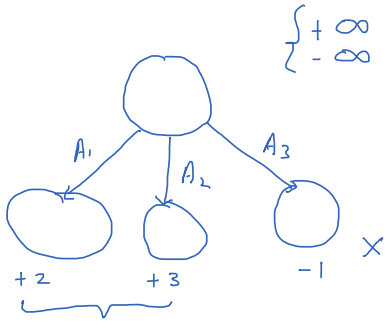
↓

(0,3)

* Goal based agent :-



Goal description



$A_1 = +2$

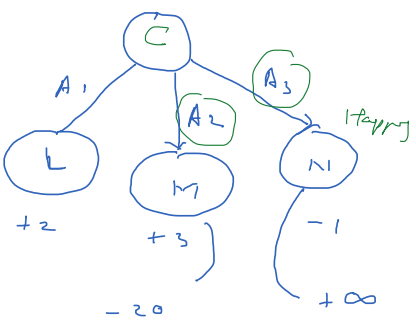
$A_2 = +3$

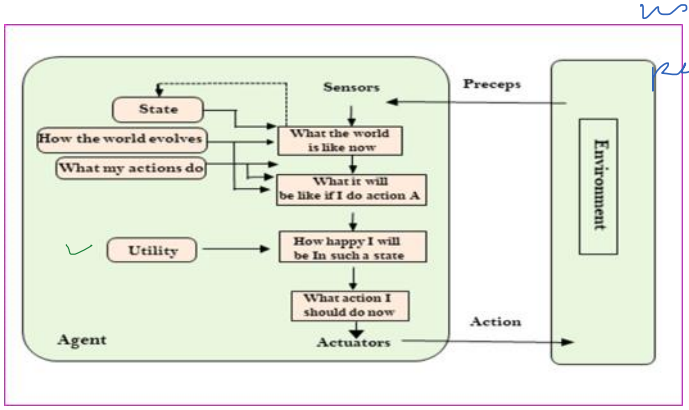
$A_3 = -1$

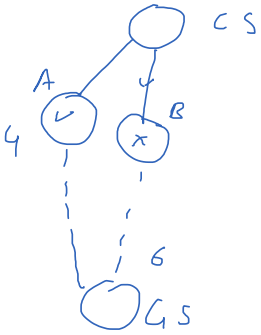
* Utility based agent :-

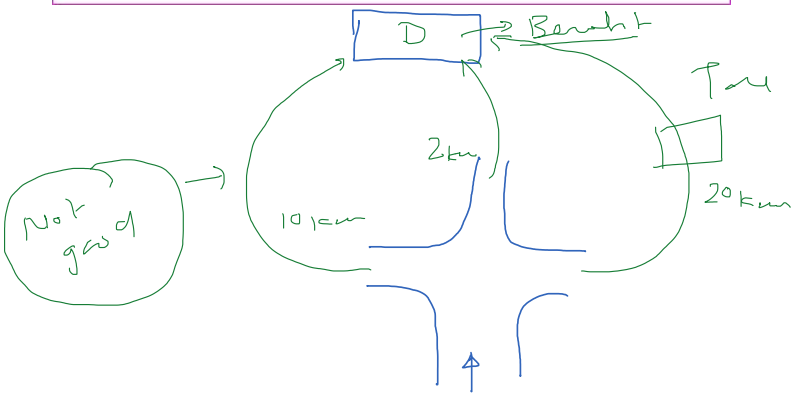
Utility =

How happy I will be in a

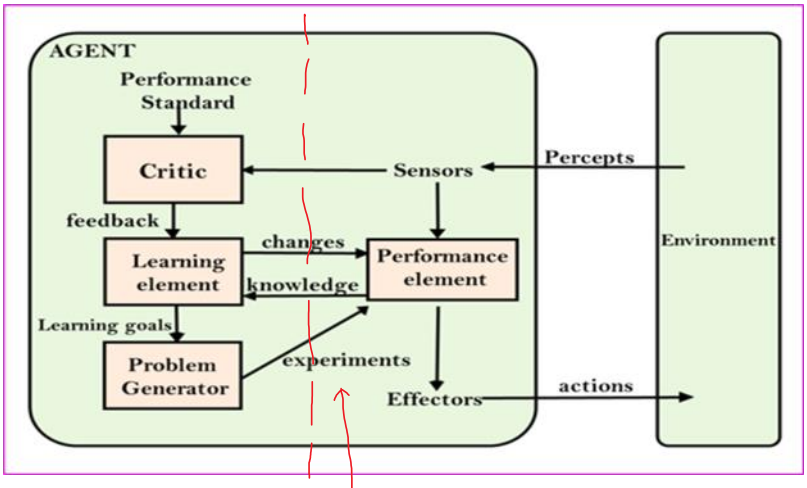








- Minimum distance
- Minimum time
- Safety



Suggestions.

$P \rightarrow Q$

$P \rightarrow \neg Q$

Knowledge base

editors

$S_1 \rightarrow [A_1]$

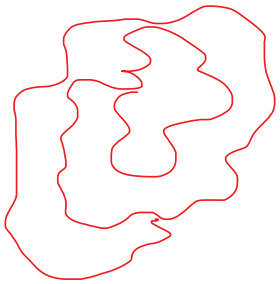
$S_1 \rightarrow [A_3]$

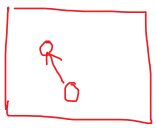
* Environment :→ Task environment

↳ Domain

↓

Features





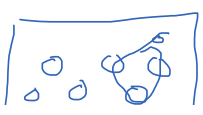
① Fully observable / Partially observable

② Static / Dynamic

③ Deterministic / Stochastic [Non deterministic]

④ Discrete / continuous

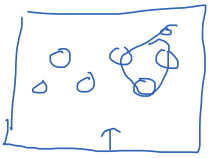
↳ states ∞ ↳ states

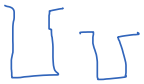


State space
20 states

→ States
→ Actions

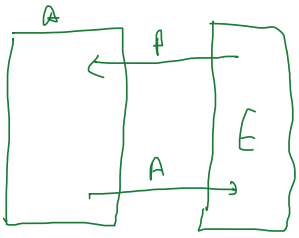
→ States
→ Action



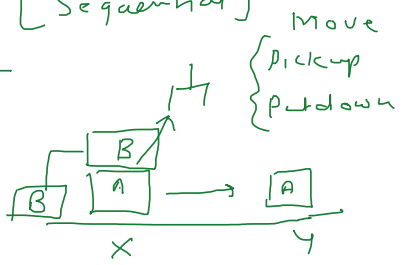


(16) × (16)

(5) Episodic / Non-episodic [Sequential]



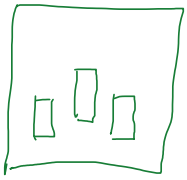
A → q_1 q_2 q_3 q_4



Move object A from X to Y

8-queens

(6) Single agent / Multi agent



* PEAS [Describe PEAS for an agent/environment]

Performance Measure
Environment
Actuators
Sensors

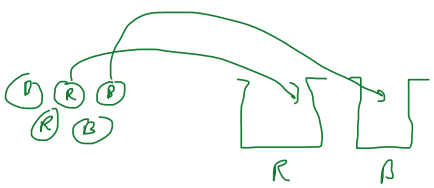
PEAS for self-driving cars:

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.

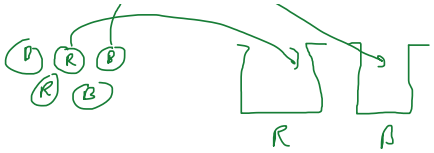
Example of Agents with their PEAS representation

Agent	Performance	Environment	Actuators	Sensors



Example of Agents with their PEAS representation

Agent	Performance measure	Environment	Actuators	Sensors
Medical Diagnose	◦ Healthy patient ◦ Minimized cost	◦ Patient ◦ Hospital ◦ Staff	◦ Tests ◦ Treatments	◦ Keyboard (Entry of symptoms)
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
Part - picking Robot	◦ Percentage of parts in correct bins.	◦ Conveyor belt with parts, ◦ Bins	◦ Jointed Arms ◦ Hand	◦ Camera ◦ Joint angle sensors.



Problem Formulation

Components involved in **problem formulation**:

I. **Initial State**: It is the starting state or initial step of the agent towards its goal.

II. **Actions**: It is the description of the possible actions available to the agent.

III. **Transition Model**: It describes what each action does.

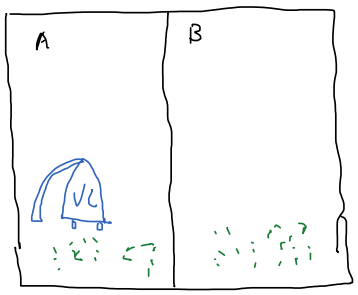
IV. **Goal Test**: It determines if the given state is a goal state.

V. **Path cost**: It assigns a numeric cost to each path that follows the goal. An optimal solution has the lowest path cost among all the solutions.

Fill 4G jug
(x, y) → (4, y)

( )
↓
( ) → goal
↓

* Vacuum Cleaner Problem :-



Goal : To clear Rooms A & B

(*) State discription: [Pair of three integers]
(x, y, z)

VC, A, B
↳ Room A ↳ Room is clean
Room B ↳ Room is having dirt

VC A B
↓ ↓ ↓
(x, y, z)
↳ 1 (A) ↳ 3 (C) ↳ 5 (C)
2 (B) ↳ 4 (NC) ↳ 6 (NC)

(*) Initial state : (1, 4, 6) / (2, 4, 6)
Goal state : (1, 3, 5) / (2, 3, 5)

* list of actions :-
1. Right :- Move from room A to room B
2. Left :- ——→ B to ——— A
3. Clean :- Clean the dirt

* Solution :-
(1, 4, 6) ← I.S.
(1) ↓ clean
Path cost = 3

(1, 3, 6) → Goal test

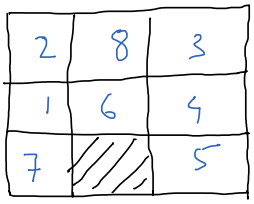
(1) ↓ Right

(2, 3, 6) → Goal test

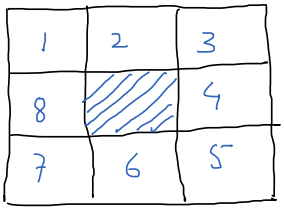
(1) ↓ Clean

(2, 3, 5) ← G.S. ✓

* 8-puzzle problem :-



I.S.

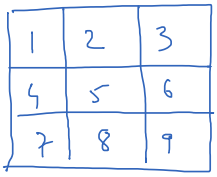


G.S.

Tiles - (8)

Positions - (9)

↳ Blank



Board Position

* State discription :- Position of each tile & the blank on the board.

Blank can be represented by integer 9

∴ State is a list of 9 integers :-

* Initial state : [2, 8, 3, 1, 6, 4, 7, 9, 5]

Goal state : [1, 2, 3, 8, 9, 4, 7, 6, 5]

* List of actions :- [w.r.t. the movement of blank]

(1) Right :- Swap the blank with tile to the right of the blank.

(2) Left :- ——— Left ———

(3) Up :- ——— with upper tile

(4) down :- ——— with lower tile

* Solution :-

[2, 8, 3, 1, 6, 4, 7, 9, 5]

↓ up

[2, 8, 3, 1, 9, 4, 7, 6, 5]

↓ up

[2, 9, 3, 1, 8, 4, 7, 6, 5]

[2, 9, 3, 1, 8, 4, 7, 6, 5]

↓ Left

[9, 2, 3, 1, 8, 4, 7, 6, 5]

↓ down

[1, 2, 3, 9, 8, 4, 7, 6, 5]

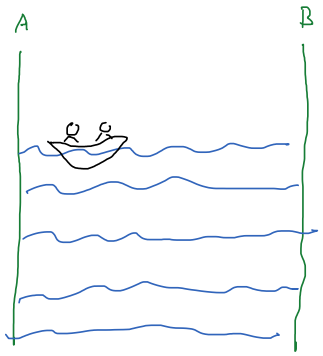
↓ Right

[1, 2, 3, 8, 9, 4, 7, 6, 5] ← G.S.

Path cost :- 5

— x — x — x —

* Missionaries & cannibals problem :-



M_1, M_2, M_3
C_1, C_2, C_3

Boat capacity = 2

- Bank A
- Bank B
- Boat

{ A, Boat, B }
↳ A
↳ B

I.S. = { [$M_1, M_2, M_3, C_1, C_2, C_3$], [A], [ ] }

G.S. : { [ ], [B], [$M_1, M_2, M_3, C_1, C_2, C_3$] }

Module-I

Introduction to Artificial Intelligence (AI) and Soft Computing (SC)

Agenda

- Introduction to AI
- Introduction to Agent and their types
- Properties of environment
- PEAS for Agent

Artificial Intelligence

Definition of AI:

- "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

Why Artificial Intelligence?

- 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:

- Replicate human intelligence
- Solve Knowledge-intensive tasks
- An intelligent connection of perception and action
- 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
- Creating some system which can exhibit intelligent behavior, learn new things by itself, demonstrate, explain, and can advise to its user.

What Comprises to Intelligence?

Intelligence is an intangible part of our brain which is a combination of:

- Reasoning
- Learning
- problem-solving
- Perception
- language understanding, etc.

Advantages & Disadvantages of AI

Advantages	Disadvantages
• High Accuracy with less errors	• High Cost
• High-Speed	• Can't think out of the box
• High reliability	• No feelings and emotions
• Useful for risky areas	• Increase dependency on machines
• Digital Assistant	• No Original Creativity
• Useful as a public utility	

Advantages..

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…

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.

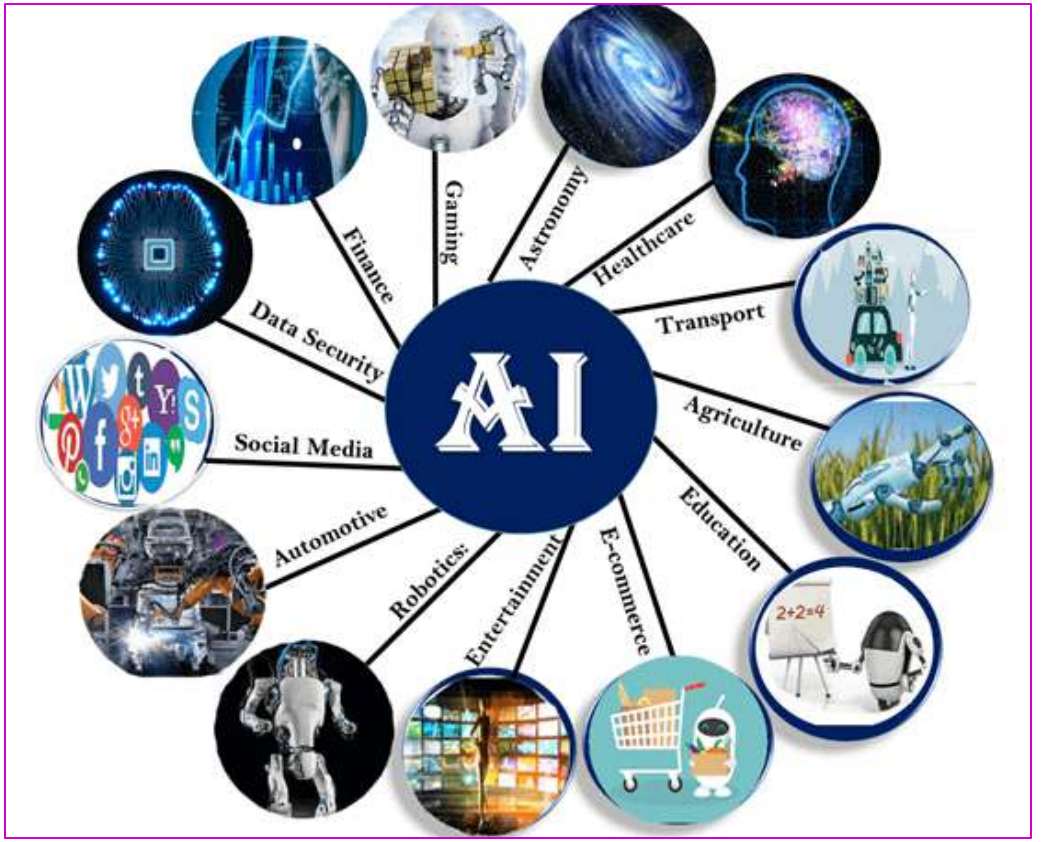
Increase dependency on machines: With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.

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

Application of AI

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time. The Various sectors:

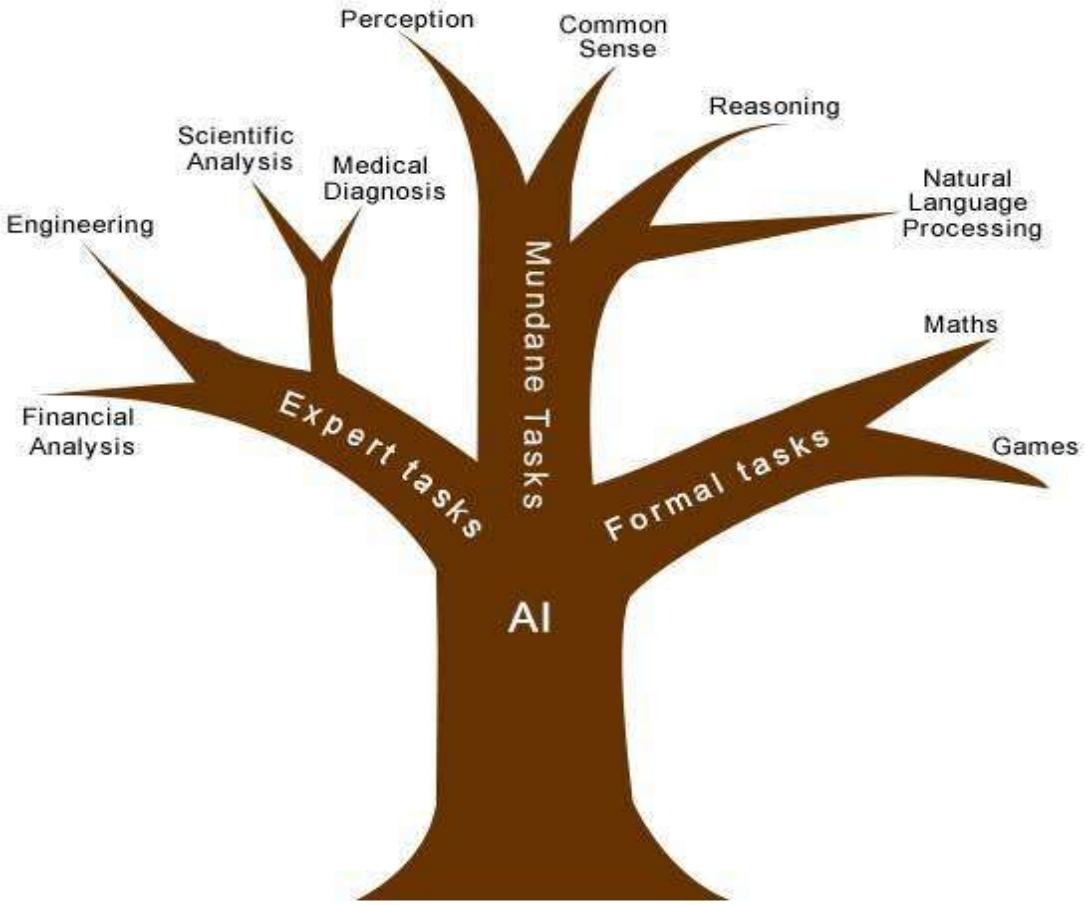
- Astronomy
- Healthcare
- Gaming
- Finance
- Data Security
- Social Media
- Travel & Transport
- Automotive Industry
- Robotics
- Entertainment
- Agriculture
- E-commerce
- Education



AI Problems (Tasks)

AI Problems can be classified as:

1. Mundane tasks
2. Formal tasks
3. Expert tasks



Agents

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:

  Sensors : Eyes, Ears & Other Organs.

  Actuators : Hand, Legs, Vocal tract.

- Robotic Agent:

  Sensors : Cameras, Infrared range finder, NLP

  Actuators : Various motors

- Software Agent:

  Sensors : Keystrokes, File contents

  Actuators : Screen

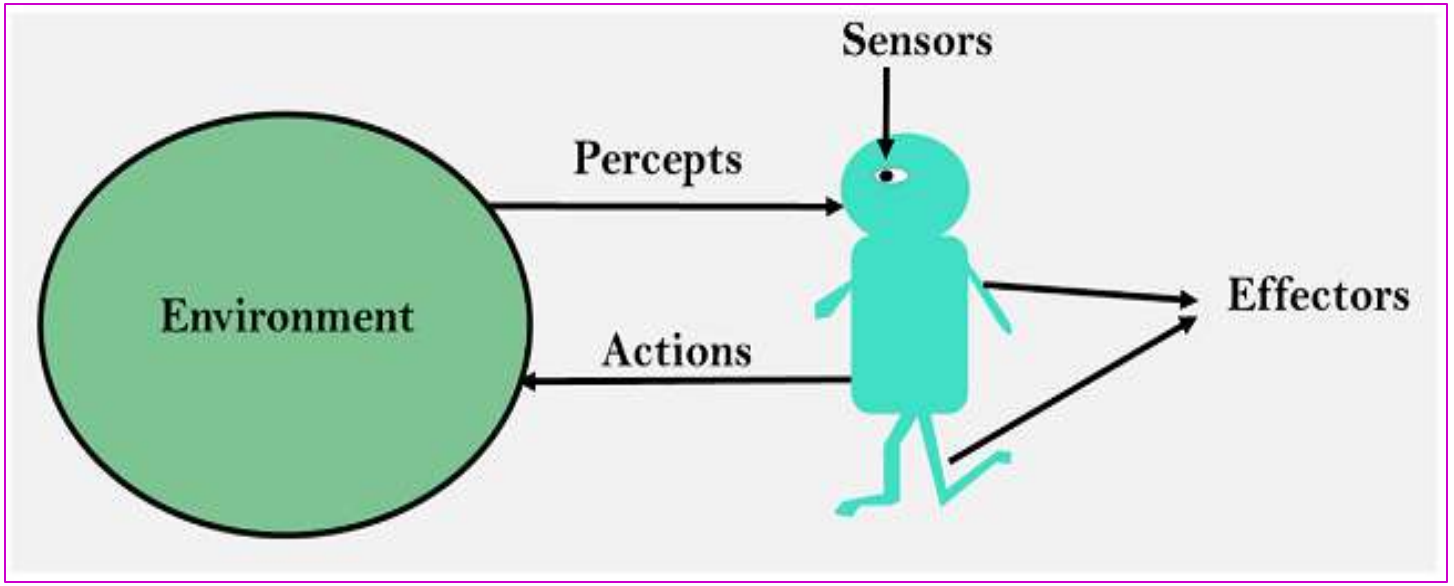
An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals.

Agent

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

Actuators: These are component of machines that converts energy into motion. The actuators are only responsible for moving and controlling a system.

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



Intelligent Agents:

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.

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.

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.

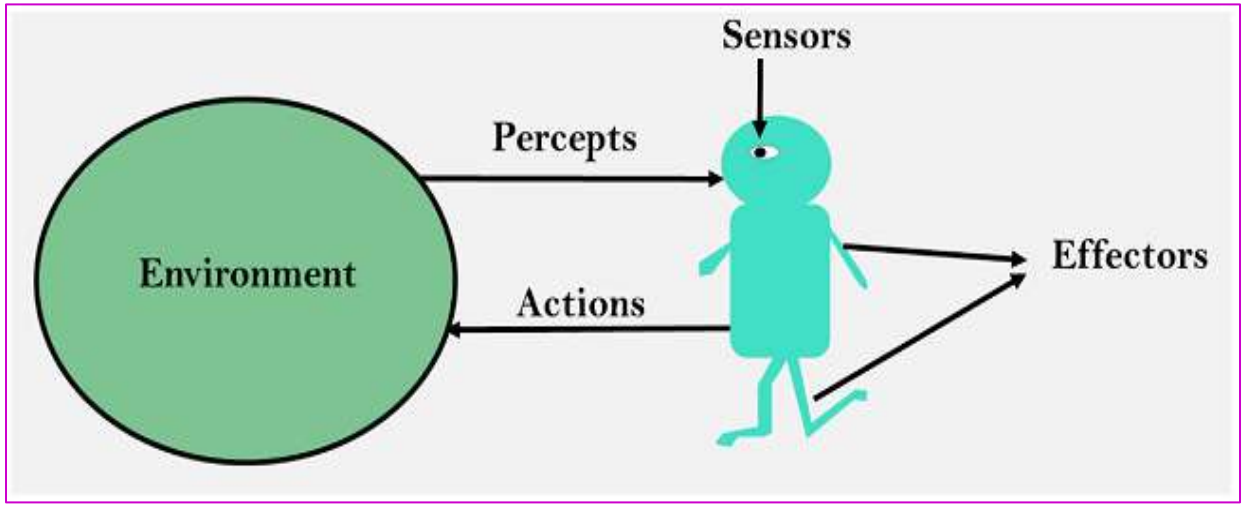
$$\mathbf{f : P^* \rightarrow A}$$

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

Types of AI Agents

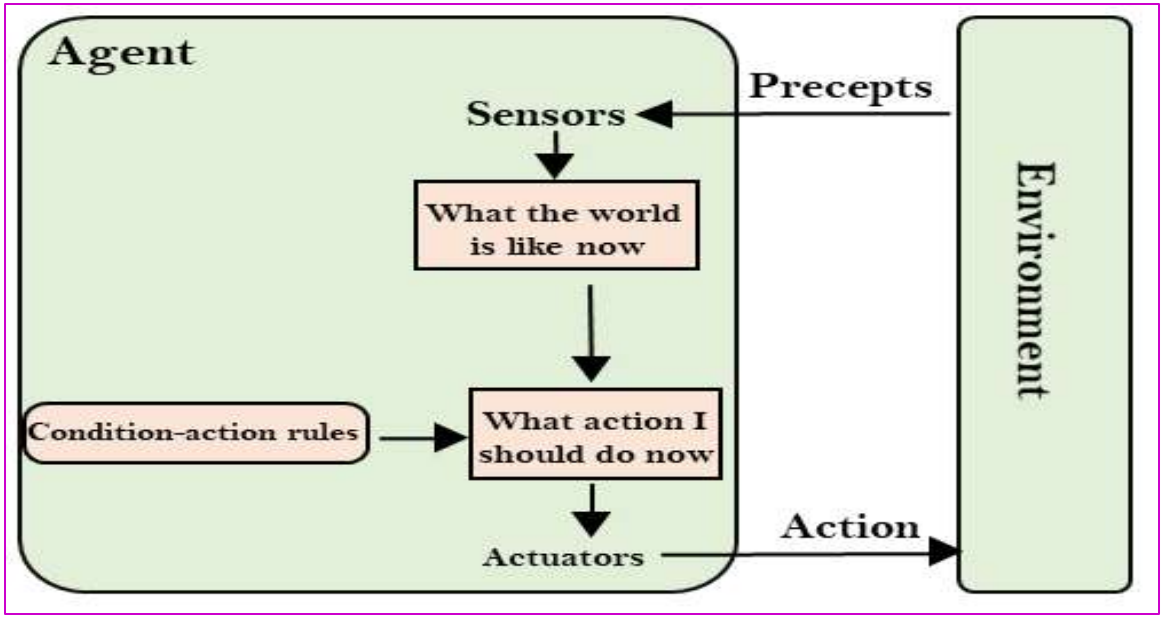
Agents can be grouped into **five classes** based on their degree of perceived intelligence and capability. These are given below:

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



1. Simple Reflex Agent:

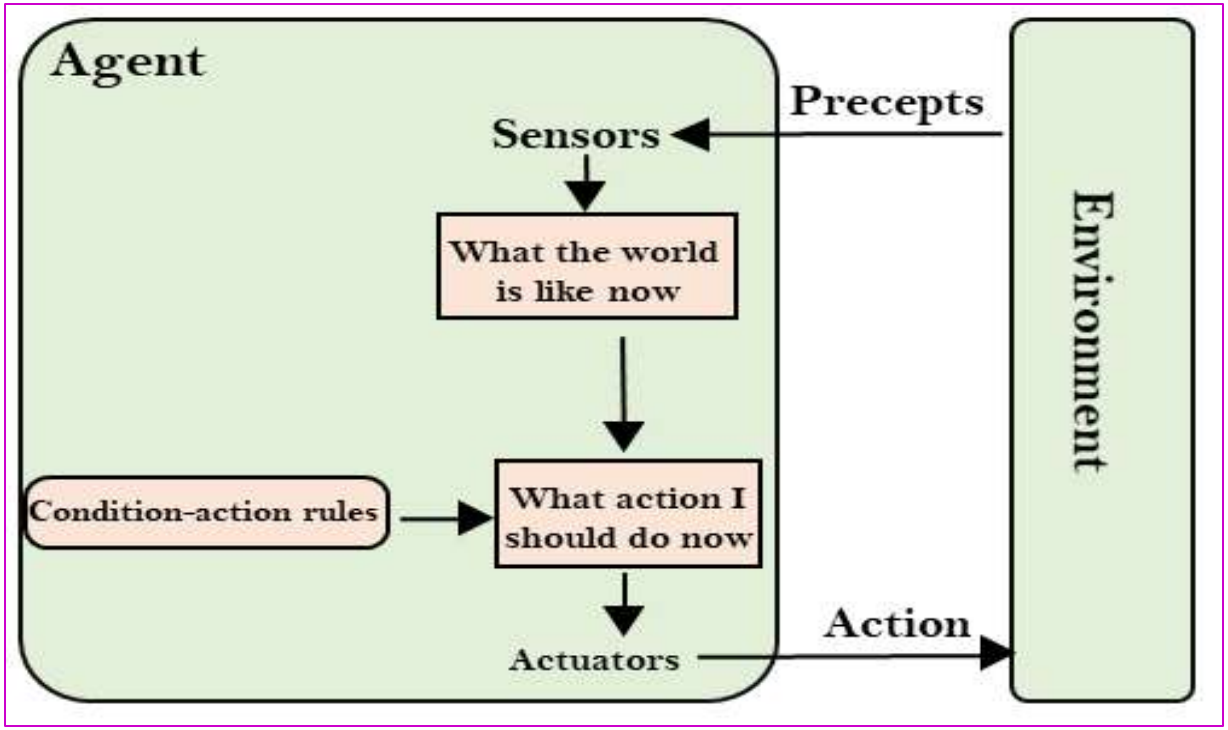
- Succeed in the fully observable environment.
- Take decisions on the basis of the current percepts and ignore the rest of the percept history.
- Do not consider any part of percepts history during their decision and action process.
- Works on Condition-action rule, which means it maps the current state to action. Eg. Room Cleaner agent



1. Simple Reflex Agent:

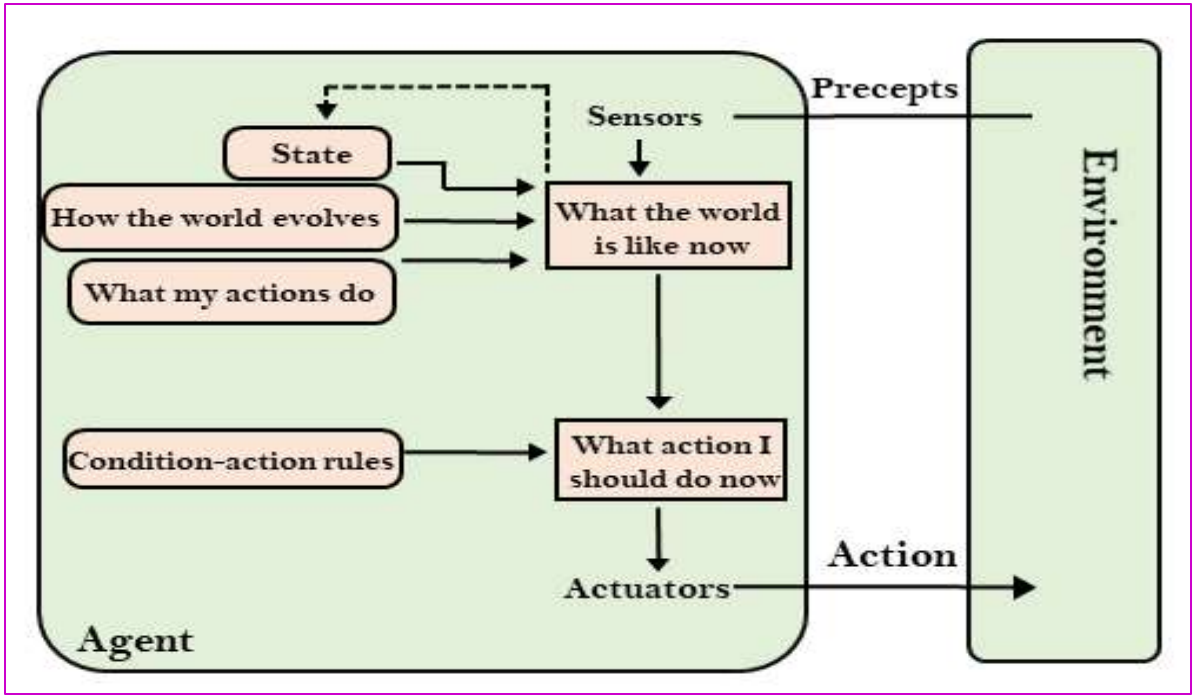
Problems for the simple reflex agent design approach:

- Limited intelligence
- Don’t 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.



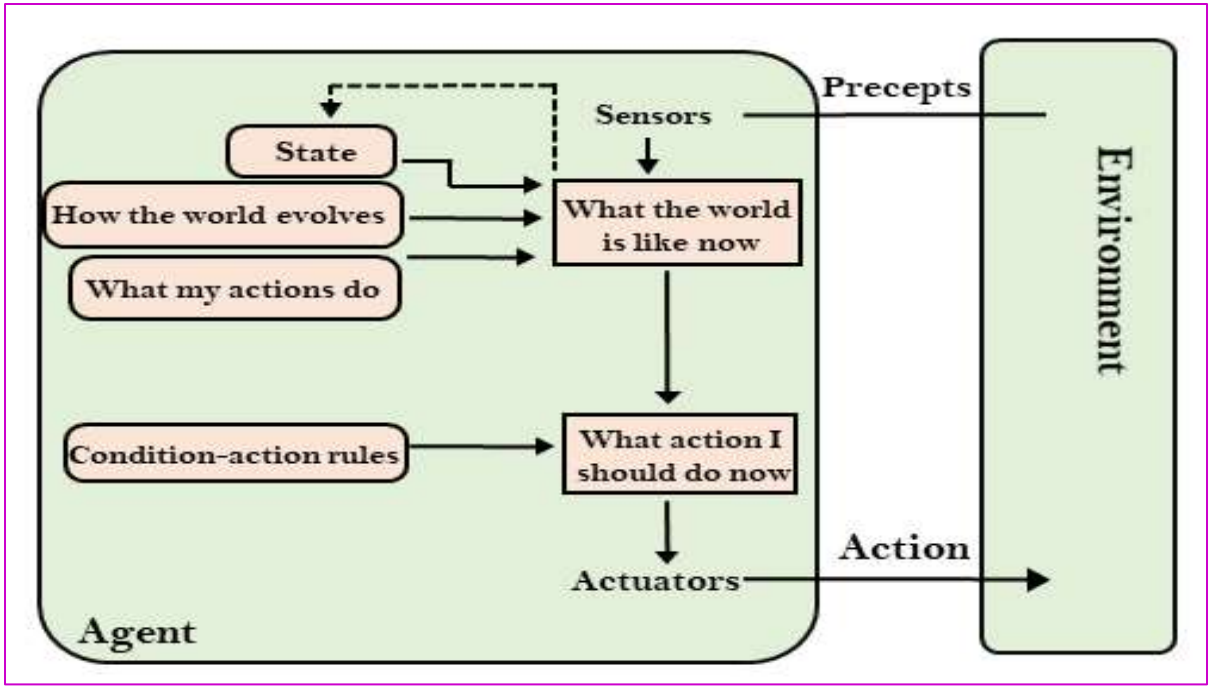
2. Model-based reflex agent

- It 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.



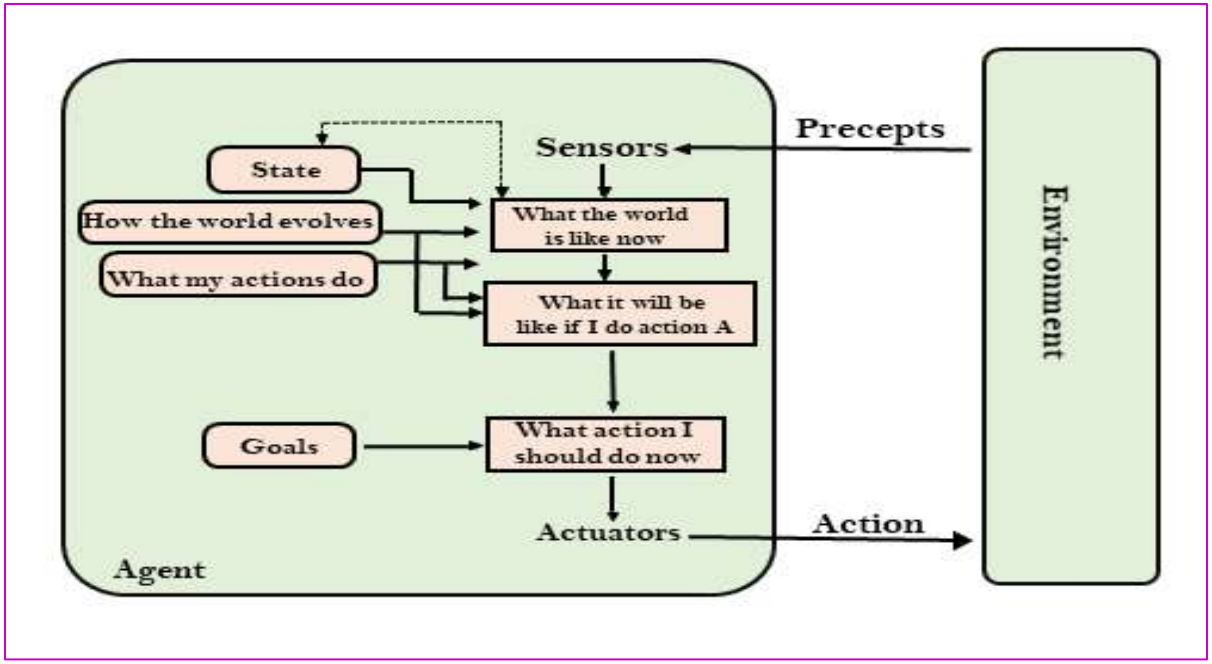
2. Model-based reflex agent

- 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:
  - How the world evolves?
  - How the agent's action affects the world?



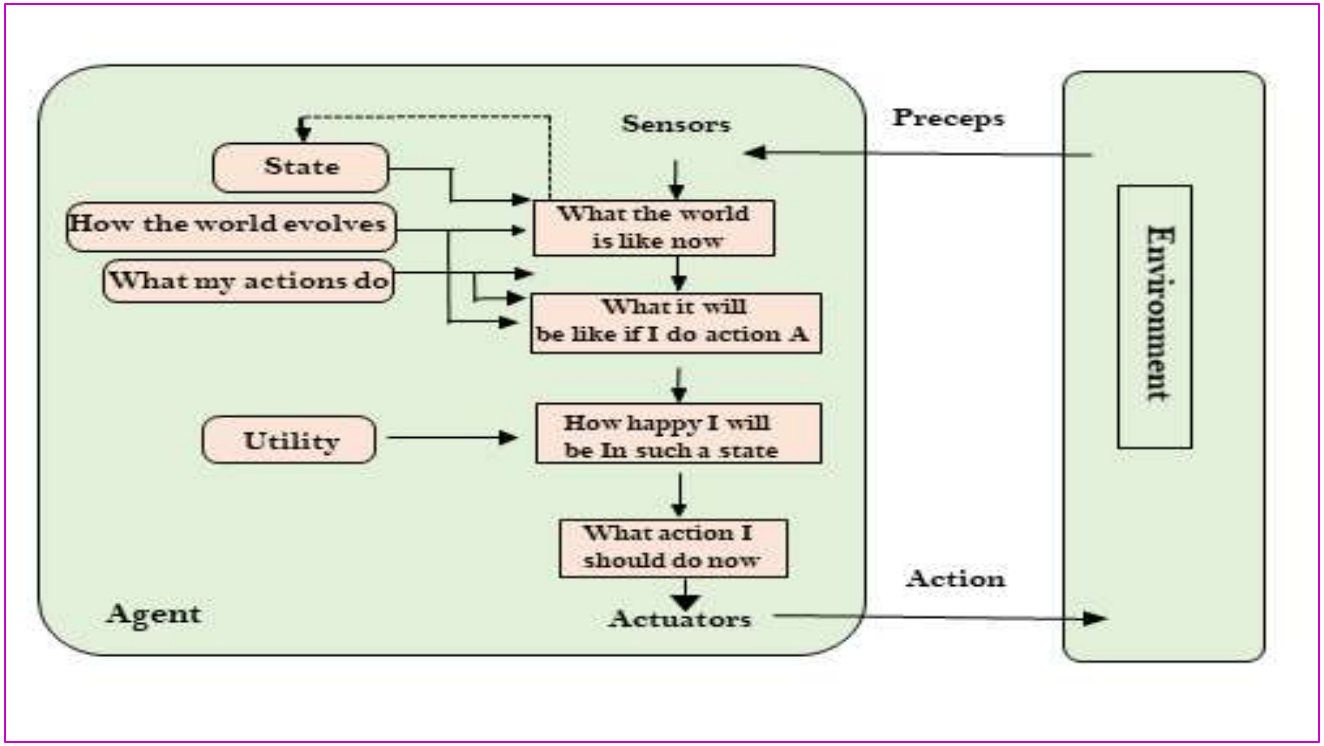
3. Goal-based agents

- 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.
- It 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.



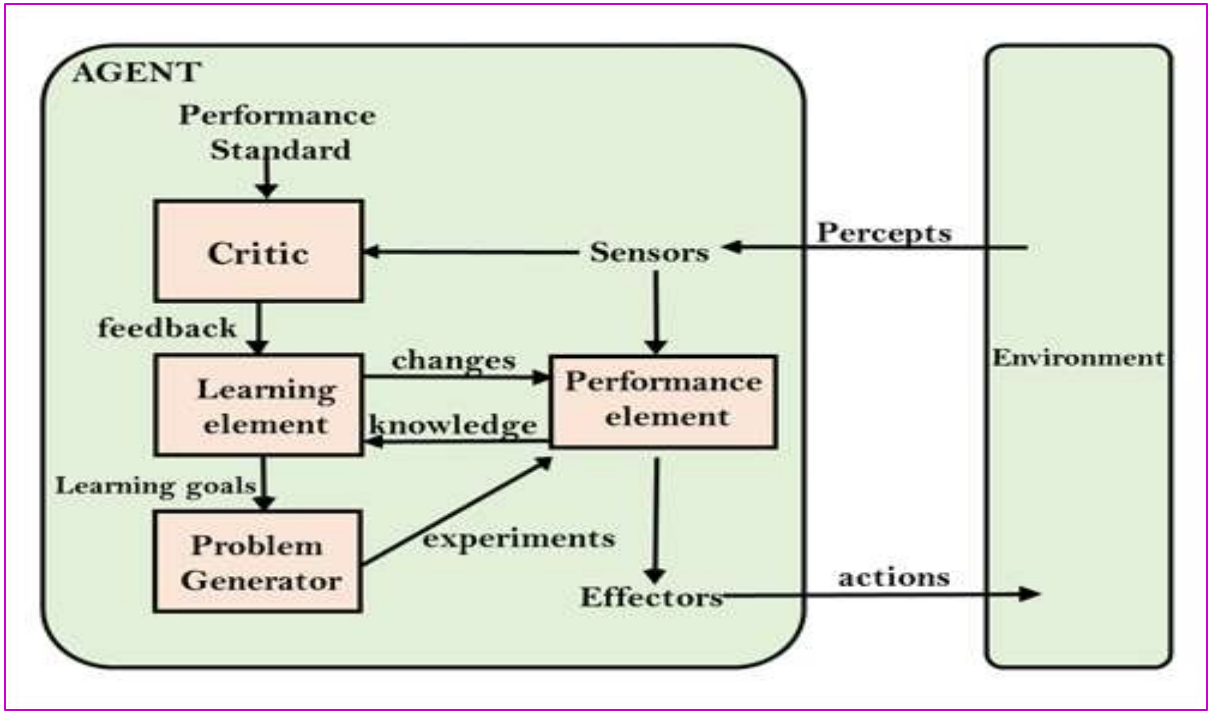
4. Utility-based agents

- These agents provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Its act based not only goals but also the best way to achieve the goal.
- It is useful when there are multiple possible alternatives, and an agent must 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.



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.



5. Learning Agents

- A learning agent has mainly four conceptual components, which are:

1. **Learning element:** It is responsible for making improvements by learning from environment.
2. **Critic:** Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
3. **Performance element:** It is responsible for selecting external action
4. **Problem generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.

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

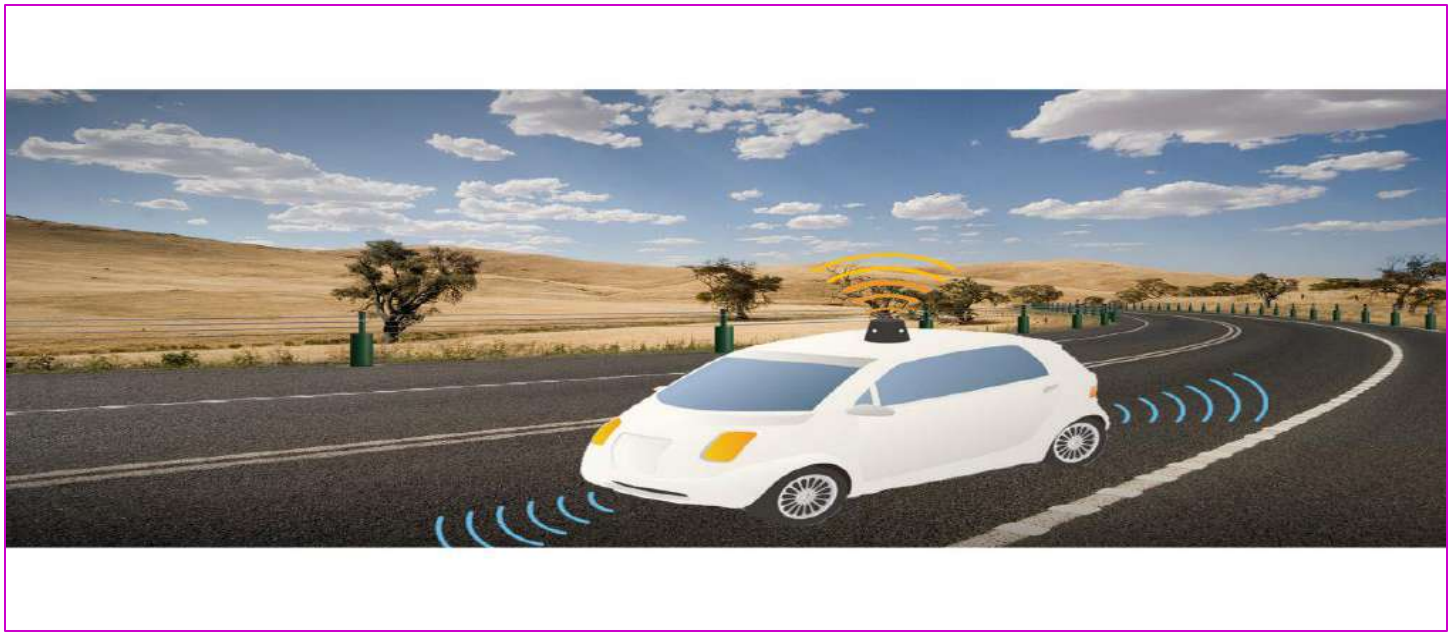
PEAS Representation

PEAS is a type of model on which an AI agent works upon. It is made up of **four** words:

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

Note: 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
Medical Diagnose	◦ Healthy patient ◦ Minimized cost	◦ Patient ◦ Hospital ◦ Staff	◦ Tests ◦ Treatments	◦ Keyboard (Entry of symptoms)
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
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 can be described as a situation in which an agent is present.
- An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself.
- The environment is where agent lives, operate and provide the agent with something to sense and act upon it.

Features of Environment

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

Features of Environment

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**.

Features of Environment

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.

Features of 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 game** comes 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.

Features of 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.

QUIZ

1. The term Artificial Intelligence was coined by_____

   a) Bill Gates b) Larry Page

   c) Steve Jobs d) John McCarthy

2. The domain of AI is classified into ____ , ____ and ____ tasks

   a) Formal, Mundane, and Expert b) Input , Processing and Output

   c) Intercept, Interpret and Infer d) None of the mentioned

3. Common Sense , Reasoning , Perception and NLP belongs to ___

   a) Formal Tasks b) Mundane Tasks

   c) Expert Tasks d) None of the mentioned

QUIZ

4. Which instruments are used for perceiving and acting upon the environment?
   a) Sensors and Actuators
   b) Sensors
   c) Perceiver
   d) None of the mentioned

5. What is the expansion of PEAS in task environment?
   a) Peer, Environment, Actuators, Sense
   b) Perceiving, Environment, Actuators, Sensors
   c) Performance, Environment, Actuators, Sensors
   d) None of the mentioned

6. Sophia Robot is developed at _____
   a) Boston Dynamics
   b) Apple
   c) Hanson Robotics
   d) Sony Electronics

QUIZ

7. The rational agent is one which _____

   a) Knows Everything
   b) Does the right things
   c) acts like human
   d) Thinks like human

8. What is the rule of simple reflex agent?

   a) Simple-action rule
   b) Condition-action rule
   c) Both a & b
   d) None of the mentioned

9. In which agent does the problem generator is present?

   a) Learning agent
   b) Observing agent
   c) Reflex agent
   d) None of the mentioned

10. Which agent deals with happy and unhappy states?

    a) Simple reflex agent
    b) Model based agent
    c) Learning agent
    d) Utility based agent

QUIZ

11. 4. What kind of environment is crossword puzzle?
    a) Partially Observable b) Fully Observable
    c) Deterministic d) both b and c

12. 5. What provides the feedback to the learning element?
    a) Critic b) Actuators
    c) Sensor d) None of the mentioned

Introduction
to
Soft Computing

What is Soft Computing?

The idea behind soft computing is to model **cognitive behavior** of human mind.

Soft computing is foundation of **conceptual intelligence** in machines.

Soft computing is defined as a collection of techniques spanning many fields that fall under various categories in computational intelligence.

Soft computing has main branches:

- Fuzzy Systems
- Evolutionary computation
- Artificial neural computing
- Machine Learning (ML)
- Probabilistic Reasoning (PR)
- Belief networks
- Parts of learning theory
- Wisdom based Expert System (WES), etc.

Soft and Hard Computing

Hard Computing	Soft computing
Conventional computing requires a precisely stated analytical model.	Soft computing is tolerant of imprecision.
Often requires a lot of computation time.	Can solve some real-world problems in reasonably less time.
Not suited for real world problems for which ideal model is not present.	Suitable for real world problems.
It requires full truth	Can work with partial truth
It is precise and accurate.	Imprecise
High cost for solution.	Low cost for solution.

Soft Computing

Soft Computing is an approach for constructing systems which are **computationally intelligent**, possess human like **expertise** in particular domain, can **adapt** to the changing environment, can **learn** to do better and can **explain** their decisions

Components of Soft Computing include:

- Fuzzy Logic (FL)
- Evolutionary Computation (EC)
- Genetic Algorithm
- Swarm Intelligence
- Neural Network (NN)
- Machine Learning (ML)

Techniques In Soft Computing:

1. Neural Networks (NN)

- An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way **biological nervous systems,** such as the brain, process information.
- The key element of this paradigm is the **novel structure** of the information processing system.
- It is composed of a large number of highly interconnected **processing elements** (**neurons**) working in unison to solve specific problems.
- ANNs, like people, **learn** by example.
- An ANN is configured for a specific application, such as **pattern recognition** or **data classification**, through a **learning process**.
- Learning in biological systems involves **adjustments** to the synaptic connections that exist between the neurons.
- This is true for ANNs as well.

Techniques In Soft Computing:

2. Fuzzy Logic (FL)

- FL is a **problem-solving control system methodology** that lends itself to implementation in systems **ranging** from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation- based **data acquisition** and **control systems**.
- It can be **implemented** in hardware, software, or a combination of both.
- FL provides a simple way to **arrive** at a **definite conclusion** based upon vague, ambiguous, imprecise, noisy, or missing input information.
- FL's approach to control problems **mimics** how a person would make **decisions**, only **much faster**.

Techniques In Soft Computing:

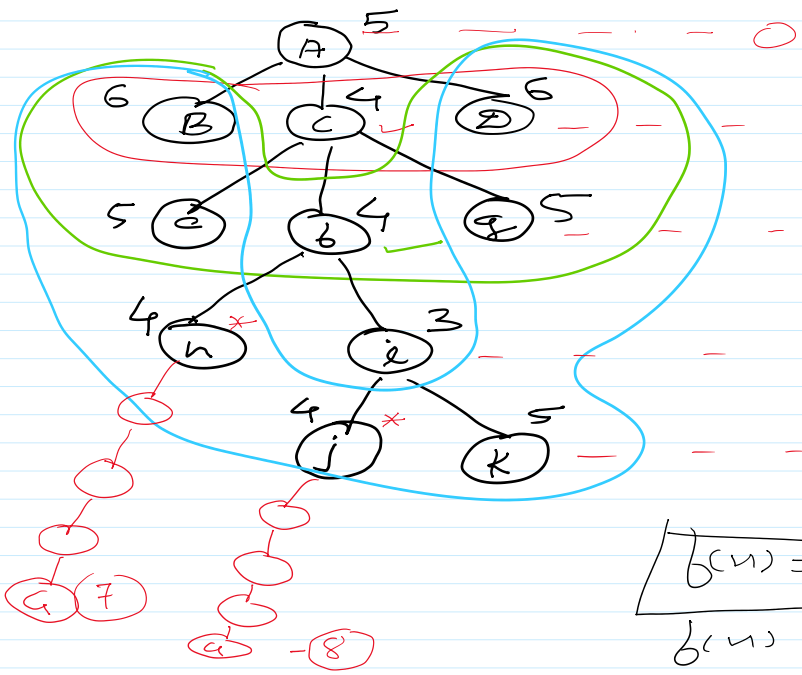
3. Genetic Algorithms in Evolutionary Computation

- A genetic or evolutionary algorithm applies the **principles of evolution** found in nature to the problem of finding an **optimal solution** to a Solver problem.
- In a "genetic algorithm." the problem is encoded in a **series of bit strings** that are manipulated by the algorithm.
- In an "evolutionary algorithm," the **decision variables** and **problem functions** are used directly.
- Most commercial Solver products are based on evolutionary algorithms.

Applications of Soft Computing

- Handwriting Recognition
- Image Processing and Data Compression
- Automotive Systems and Manufacturing
- Soft Computing to Architecture
- Decision-support Systems
- Soft Computing to Power Systems
- Neuro Fuzzy systems
- Fuzzy Logic Control
- Machine Learning Applications
- Speech and Vision Recognition Systems
- Process Control and So On

A* Search

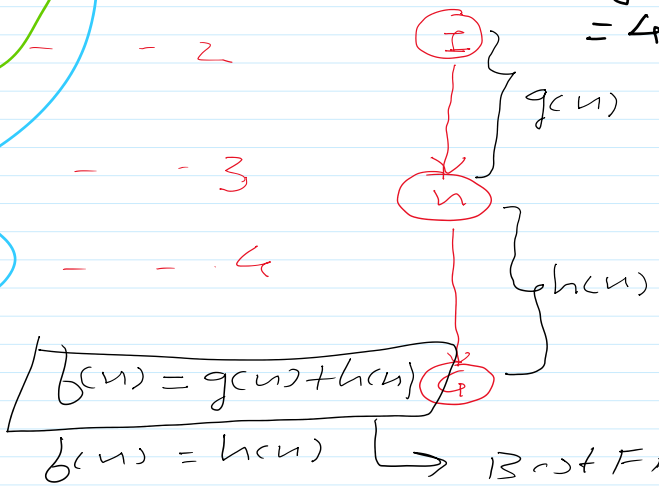


$f(n) = g(n) + h(n)$

$= 3 + 4 = 7$

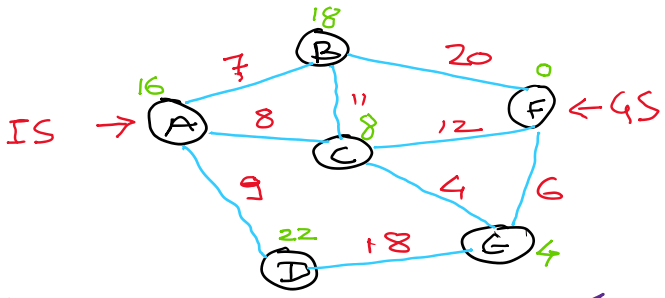
$f(j) = g(j) + h(j)$

$= 4 + 4 = 8$

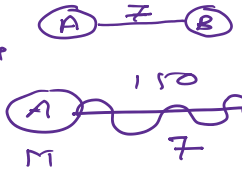


A - Admissible
- Complete
- Consistent

Route Finding Problem :



h(n) < ≠



Actual Distance > Straight Line distance

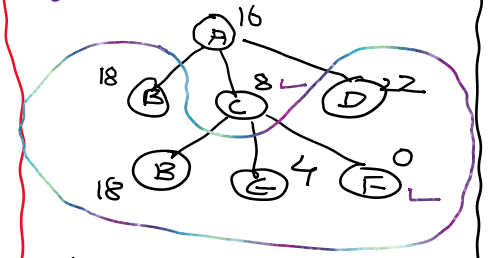
SDL = AD - 2

SDL(A) = 18 - 2
= 16

h(A) = 16
h(B) = 18

Best First Search

$f(n) = h(n)$

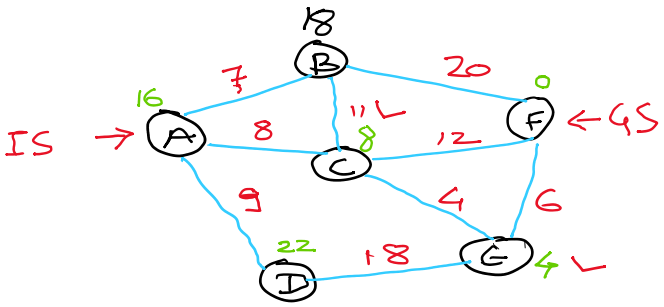


Soln :-

= 20

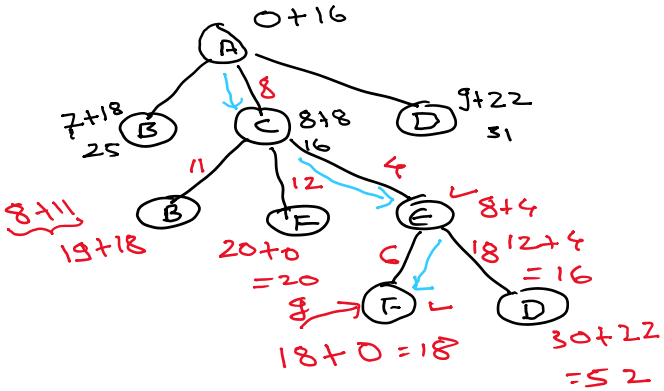
A* Search

$f(n) = g(n) + h(n)$



A*

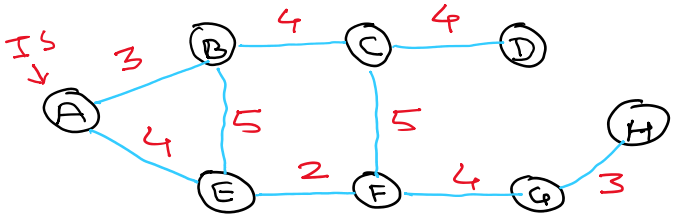
$f(n) = g(n) + h(n)$



Solution Path

= A 8 C 4 E 6 F

= 18



h(B) = 14 h(F) = 10
h(C) = 10 h(G) = 10

- Best First Search
- A* Search

h(D) = 8 h(H) = 0
h(E) = 12 h(A) = 15

gives a heuristic value of 3. There is one minor irritation: the set cover problem is NP-hard. A simple greedy set-covering algorithm is guaranteed to return a value that is within a factor of $\log n$ of the true minimum value, where n is the number of literals in the goal, and usually works much better than this in practice. Unfortunately, the greedy algorithm loses the guarantee of admissibility for the heuristic.

It is also possible to generate relaxed problems by removing negative effects without removing preconditions. That is, if an action has the effect $A \wedge \neg B$ in the original problem, it will have the effect A in the relaxed problem. This means that we need not worry about negative interactions between subplans, because no action can delete the literals achieved by another action. The solution cost of the resulting relaxed problem gives what is called the

EMPTY-DELETE-LIST

empty-delete-list heuristic. The heuristic is quite accurate, but computing it involves actually running a (simple) planning algorithm. In practice, the search in the relaxed problem is often fast enough that the cost is worthwhile.

The heuristics described here can be used in either the progression or the regression direction. At the time of writing, progression planners using the empty-delete-list heuristic hold the lead. That is likely to change as new heuristics and new search techniques are explored. Since planning is exponentially hard,[5] no algorithm will be efficient for all problems, but many practical problems can be solved with the heuristic methods in this chapter—far more than could be solved just a few years ago.

11.3 PARTIAL-ORDER PLANNING

Forward and backward state-space search are particular forms of *totally ordered* plan search. They explore only strictly linear sequences of actions directly connected to the start or goal. This means that they cannot take advantage of problem decomposition. Rather than work on each subproblem separately, they must always make decisions about how to sequence actions from all the subproblems. We would prefer an approach that works on several subgoals independently, solves them with several subplans, and then combines the subplans.

Such an approach also has the advantage of flexibility in the order in which it *constructs* the plan. That is, the planner can work on "obvious" or "important" decisions first, rather than being forced to work on steps in chronological order. For example, a planning agent that is in Berkeley and wishes to be in Monte Carlo might first try to find a flight from San Francisco to Paris; given information about the departure and arrival times, it can then work on ways to get to and from the airports.

LEAST COMMITMENT

The general strategy of delaying a choice during search is called a **least commitment** strategy. There is no formal definition of least commitment, and clearly some degree of commitment is necessary, lest the search would make no progress. Despite the informality, least commitment is a useful concept for analyzing when decisions should be made in any search problem.

[5] Technically, STRIPS-style planning is PSPACE-complete unless actions have only positive preconditions and only one effect literal (Bylander, 1994).

Our first concrete example will be much simpler than planning a vacation. Consider the simple problem of putting on a pair of shoes. We can describe this as a formal planning problem as follows:

$$Goal(RightShoeOn \wedge LeftShoeOn)$$
$$Init()$$
$$Action(RightShoe, \text{PRECOND:}RightSockOn, \text{EFFECT:}RightShoeOn)$$
$$Action(RightSock, \text{EFFECT:}RightSockOn)$$
$$Action(LeftShoe, \text{PRECOND:}LeftSockOn, \text{EFFECT:}LeftShoeOn)$$
$$Action(LeftSock, \text{EFFECT:}LeftSockOn)\ .$$

A planner should be able to come up with the two-action sequence *RightSock* followed by *RightShoe* to achieve the first conjunct of the goal and the sequence *LeftSock* followed by *LeftShoe* for the second conjunct. Then the two sequences can be combined to yield the final plan. In doing this, the planner will be manipulating the two subsequences independently, without committing to whether an action in one sequence is before or after an action in the other. Any planning algorithm that can place two actions into a plan without specifying which comes first is called a **partial-order planner**. Figure 11.6 shows the partial-order plan that is the solution to the shoes and socks problem. Note that the solution is represented as a *graph* of actions, not a sequence. Note also the "dummy" actions called *Start* and *Finish*, which mark the beginning and end of the plan. Calling them actions symplifies things, because now every step of a plan is an action. The partial-order solution corresponds to six possible total-order plans; each of these is called a **linearization** of the partial-order plan.

PARTIAL-ORDER PLANNER

LINEARIZATION

Partial-order planning can be implemented as a search in the space of partial-order plans. (From now on, we will just call them "plans.") That is, we start with an empty plan. Then we consider ways of refining the plan until we come up with a complete plan that solves the problem. The actions in this search are not actions in the world, but actions on plans: adding a step to the plan, imposing an ordering that puts one action before another, and so on.

We will define the POP algorithm for partial-order planning. It is traditional to write out the POP algorithm as a stand-alone program, but we will instead formulate partial-order planning as an instance of a search problem. This allows us to focus on the plan refinement steps that can be applied, rather than worrying about how the algorithm explores the space. In fact, a wide variety of uninformed or heuristic search methods can be applied once the search problem is formulated.

Remember that the states of our search problem will be (mostly unfinished) plans. To avoid confusion with the states of the world, we will talk about plans rather than states. Each plan has the following four components, where the first two define the steps of the plan and the last two serve a bookkeeping function to determine how plans can be extended:

- A set of **actions** that make up the steps of the plan. These are taken from the set of actions in the planning problem. The "empty" plan contains just the *Start* and *Finish* actions. *Start* has no preconditions and has as its effect all the literals in the initial state of the planning problem. *Finish* has no effects and has as its preconditions the goal literals of the planning problem.

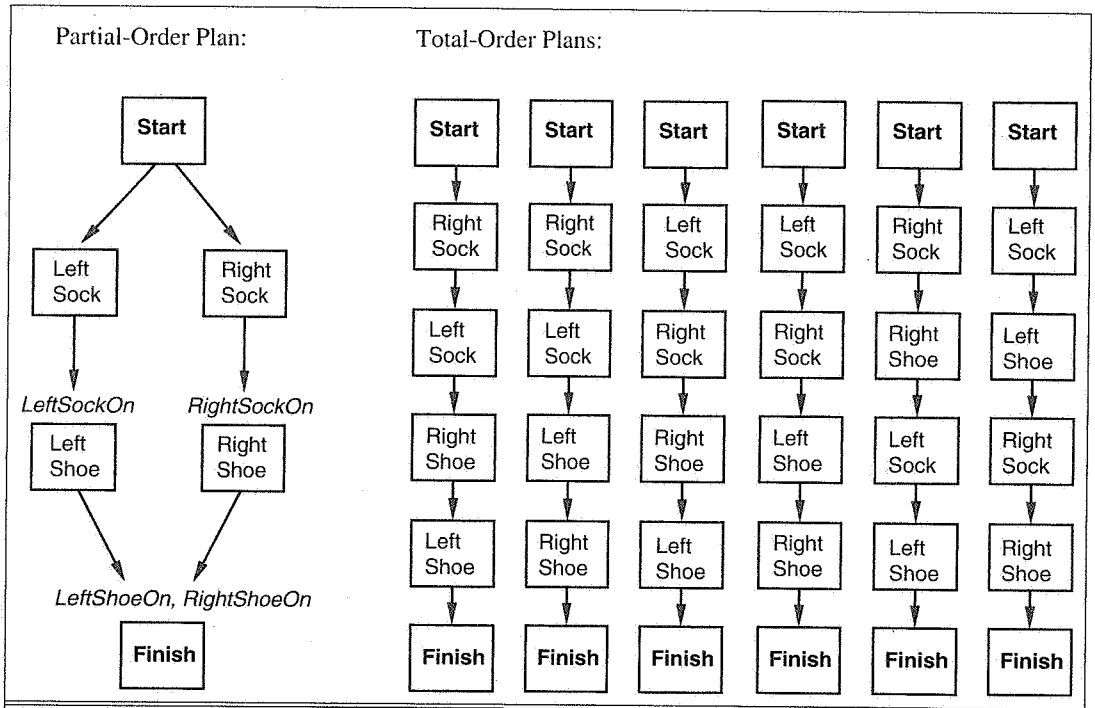


Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

ORDERING CONSTRAINTS

- A set of **ordering constraints**. Each ordering constraint is of the form $A \prec B$, which is read as "A before B" and means that action A must be executed sometime before action B, but not necessarily immediately before. The ordering constraints must describe a proper partial order. Any cycle—such as $A \prec B$ and $B \prec A$—represents a contradiction, so an ordering constraint cannot be added to the plan if it creates a cycle.

CAUSAL LINKS
ACHIEVES

- A set of **causal links**. A causal link between two actions A and B in the plan is written as $A \xrightarrow{p} B$ and is read as "A **achieves** p for B." For example, the causal link

  $$RightSock \xrightarrow{RightSockOn} RightShoe$$

  asserts that $RightSockOn$ is an effect of the $RightSock$ action and a precondition of $RightShoe$. It also asserts that $RightSockOn$ must remain true from the time of action $RightSock$ to the time of action $RightShoe$. In other words, the plan may not be extended by adding a new action C that **conflicts** with the causal link. An action C conflicts with $A \xrightarrow{p} B$ if C has the effect $\neg p$ and if C could (according to the ordering constraints) come after A and before B. Some authors call causal links **protection intervals**, because the link $A \xrightarrow{p} B$ protects p from being negated over the interval from A to B.

CONFLICTS

OPEN PRECONDITIONS

- A set of **open preconditions**. A precondition is open if it is not achieved by some action in the plan. Planners will work to reduce the set of open preconditions to the empty set, without introducing a contradiction.

For example, the final plan in Figure 11.6 has the following components (not shown are the ordering constraints that put every other action after *Start* and before *Finish*):

Actions:$\{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$
Orderings:$\{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$
Links:$\{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,$
$RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoe \xrightarrow{LeftShoeOn} Finish\}$
Open Preconditions:$\{\}$.

CONSISTENT PLAN

We define a **consistent plan** as a plan in which there are no cycles in the ordering constraints and no conflicts with the causal links. A consistent plan with no open preconditions is a **solution**. A moment's thought should convince the reader of the following fact: *every linearization of a partial-order solution is a total-order solution whose execution from the initial state will reach a goal state.* This means that we can extend the notion of "executing a plan" from total-order to partial-order plans. A partial-order plan is executed by repeatedly choosing *any* of the possible next actions. We will see in Chapter 12 that the flexibility available to the agent as it executes the plan can be very useful when the world fails to cooperate. The flexible ordering also makes it easier to combine smaller plans into larger ones, because each of the small plans can reorder its actions to avoid conflict with the other plans.

Now we are ready to formulate the search problem that POP solves. We will begin with a formulation suitable for propositional planning problems, leaving the first-order complications for later. As usual, the definition includes the initial state, actions, and goal test.

- The initial plan contains *Start* and *Finish*, the ordering constraint $Start \prec Finish$, and no causal links and has all the preconditions in *Finish* as open preconditions.
- The successor function arbitrarily picks one open precondition p on an action B and generates a successor plan for every possible consistent way of choosing an action A that achieves p. Consistency is enforced as follows:
  1. The causal link $A \xrightarrow{p} B$ and the ordering constraint $A \prec B$ are added to the plan. Action A may be an existing action in the plan or a new one. If it is new, add it to the plan and also add $Start \prec A$ and $A \prec Finish$.
  2. We resolve conflicts between the new causal link and all existing actions and between the action A (if it is new) and all existing causal links. A conflict between $A \xrightarrow{p} B$ and C is resolved by making C occur at some time outside the protection interval, either by adding $B \prec C$ or $C \prec A$. We add successor states for either or both if they result in consistent plans.
- The goal test checks whether a plan is a solution to the original planning problem. Because only consistent plans are generated, the goal test just needs to check that there are no open preconditions.

Remember that the actions considered by the search algorithms under this formulation are plan refinement steps rather than the real actions from the domain itself. The path cost is therefore irrelevant, strictly speaking, because the only thing that matters is the total cost of the real actions in the plan to which the path leads. Nonetheless, it *is* possible to specify a path cost function that reflects the real plan costs: we charge 1 for each real action added to

the plan and 0 for all other refinement steps. In this way, $g(n)$, where n is a plan, will be equal to the number of real actions in the plan. A heuristic estimate $h(n)$ can also be used.

At first glance, one might think that the successor function should include successors for *every* open p, not just for one of them. This would be redundant and inefficient, however, for the same reason that constraint satisfaction algorithms don't include successors for every possible variable: the order in which we consider open preconditions (like the order in which we consider CSP variables) is commutative. (See page 141.) Thus, we can choose an arbitrary ordering and still have a complete algorithm. Choosing the right ordering can lead to a faster search, but all orderings end up with the same set of candidate solutions.

A partial-order planning example

Now let's look at how POP solves the spare tire problem from Section 11.1. The problem description is repeated in Figure 11.7.

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
  PRECOND: $At(Spare, Trunk)$
  EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
  PRECOND: $At(Flat, Axle)$
  EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
  PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
  EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
  $\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

Figure 11.7 The simple flat tire problem description.

The search for a solution begins with the initial plan, containing a *Start* action with the effect $At(Spare, Trunk) \wedge At(Flat, Axle)$ and a *Finish* action with the sole precondition $At(Spare, Axle)$. Then we generate successors by picking an open precondition to work on (irrevocably) and choosing among the possible actions to achieve it. For now, we will not worry about a heuristic function to help with these decisions; we will make seemingly arbitrary choices. The sequence of events is as follows:

1. Pick the only open precondition, $At(Spare, Axle)$ of *Finish*. Choose the only applicable action, $PutOn(Spare, Axle)$.
2. Pick the $At(Spare, Ground)$ precondition of $PutOn(Spare, Axle)$. Choose the only applicable action, $Remove(Spare, Trunk)$ to achieve it. The resulting plan is shown in Figure 11.8.

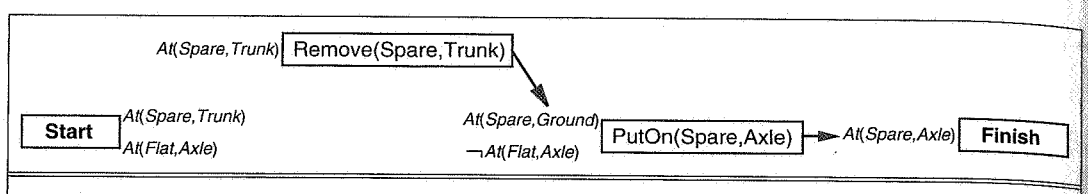


Figure 11.8 The incomplete partial-order plan for the tire problem, after choosing actions for the first two open preconditions. Boxes represent actions, with preconditions on the left and effects on the right. (Effects are omitted, except for that of the *Start* action.) Dark arrows represent causal links protecting the proposition at the head of the arrow.

3. Pick the $\neg At(Flat, Axle)$ precondition of $PutOn(Spare, Axle)$. Just to be contrary, choose the $LeaveOvernight$ action rather than the $Remove(Flat, Axle)$ action. Notice that $LeaveOvernight$ also has the effect $\neg At(Spare, Ground)$, which means it conflicts with the causal link

$$Remove(Spare, Trunk) \overset{At(Spare,Ground)}{\longrightarrow} PutOn(Spare, Axle) .$$

To resolve the conflict we add an ordering constraint putting $LeaveOvernight$ before $Remove(Spare, Trunk)$. The resulting plan is shown in Figure 11.9. (Why does this resolve the conflict, and why is there no other way to resolve it?)

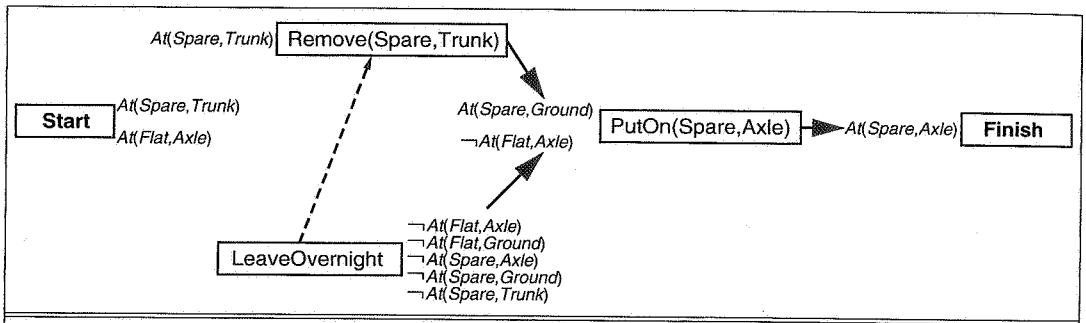


Figure 11.9 The plan after choosing $LeaveOvernight$ as the action for achieving $\neg At(Flat, Axle)$. To avoid a conflict with the causal link from $Remove(Spare, Trunk)$ that protects $At(Spare, Ground)$, $LeaveOvernight$ is constrained to occur before $Remove(Spare, Trunk)$, as shown by the dashed arrow.

4. The only remaining open precondition at this point is the $At(Spare, Trunk)$ precondition of the action $Remove(Spare, Trunk)$. The only action that can achieve it is the existing *Start* action, but the causal link from *Start* to $Remove(Spare, Trunk)$ is in conflict with the $\neg At(Spare, Trunk)$ effect of $LeaveOvernight$. This time there is no way to resolve the conflict with $LeaveOvernight$: we cannot order it before *Start* (because nothing can come before *Start*), and we cannot order it after $Remove(Spare, Trunk)$ (because there is already a constraint ordering it before $Remove(Spare, Trunk)$). So we are forced to back up, remove the $LeaveOvernight$ action and the last two causal links, and return to the state in Figure 11.8. In essence, the planner has proved that $LeaveOvernight$ doesn't work as a way to change a tire.

5. Consider again the $\neg At(Flat, Axle)$ precondition of $PutOn(Spare, Axle)$. This time, we choose $Remove(Flat, Axle)$.
6. Once again, pick the $At(Spare, Trunk)$ precondition of $Remove(Spare, Trunk)$ and choose $Start$ to achieve it. This time there are no conflicts.
7. Pick the $At(Flat, Axle)$ precondition of $Remove(Flat, Axle)$, and choose $Start$ to achieve it. This gives us a complete, consistent plan—in other words a solution—as shown in Figure 11.10.

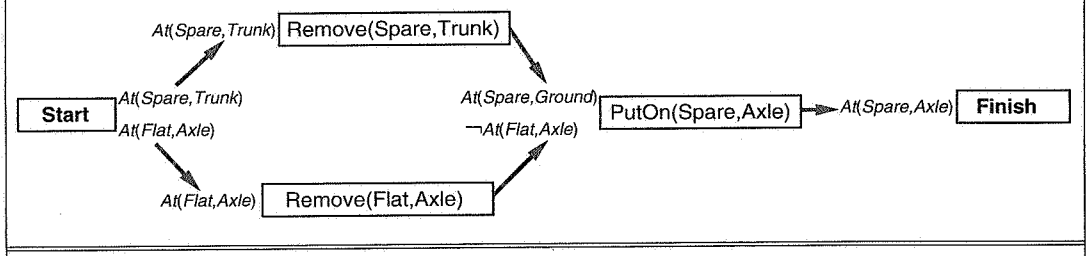


Figure 11.10 The final solution to the tire problem. Note that $Remove(Spare, Trunk)$ and $Remove(Flat, Axle)$ can be done in either order, as long as they are completed before the $PutOn(Spare, Axle)$ action.

Although this example is very simple, it illustrates some of the strengths of partial-order planning. First, the causal links lead to early pruning of portions of the search space that, because of irresolvable conflicts, contain no solutions. Second, the solution in Figure 11.10 is a partial-order plan. In this case the advantage is small, because there are only two possible linearizations; nonetheless, an agent might welcome the flexibility—for example, if the tire has to be changed in the middle of heavy traffic.

The example also points to some possible improvements that could be made. For example, there is duplication of effort: $Start$ is linked to $Remove(Spare, Trunk)$ before the conflict causes a backtrack and is then unlinked by backtracking even though it is not involved in the conflict. It is then relinked as the search continues. This is typical of chronological backtracking and might be mitigated by dependency-directed backtracking.

Partial-order planning with unbound variables

In this section, we consider the complications that can arise when POP is used with first-order action representations that include variables. Suppose we have a blocks world problem (Figure 11.4) with the open precondition $On(A, B)$ and the action

$$Action(Move(b, x, y),$$
$$\text{PRECOND}: On(b, x) \wedge Clear(b) \wedge Clear(y),$$
$$\text{EFFECT}: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)) .$$

This action achieves $On(A, B)$ because the effect $On(b, y)$ unifies with $On(A, B)$ with the substitution $\{b/A, y/B\}$. We then apply this substitution to the action, yielding

$$Action(Move(A, x, B),$$
$$\text{PRECOND:}On(A, x) \wedge Clear(A) \wedge Clear(B),$$
$$\text{EFFECT:}On(A, B) \wedge Clear(x) \wedge \neg On(A, x) \wedge \neg Clear(B)) .$$

This leaves the variable x unbound. That is, the action says to move block A from *somewhere*, without yet saying whence. This is another example of the least commitment principle: we can delay making the choice until some other step in the plan makes it for us. For example, suppose we have $On(A, D)$ in the initial state. Then the *Start* action can be used to achieve $On(A, x)$, binding x to D. This strategy of waiting for more information before choosing x is often more efficient than trying every possible value of x and backtracking for each one that fails.

The presence of variables in preconditions and actions complicates the process of detecting and resolving conflicts. For example, when $Move(A, x, B)$ is added to the plan, we will need a causal link

$$Move(A, x, B) \xrightarrow{On(A,B)} Finish .$$

If there is another action M_2 with effect $\neg On(A, z)$, then M_2 conflicts only if z is B. To accommodate this possibility, we extend the representation of plans to include a set of **inequality constraints** of the form $z \neq X$ where z is a variable and X is either another variable or a constant symbol. In this case, we would resolve the conflict by adding $z \neq B$, which means that future extensions to the plan can instantiate z to any value except B. Anytime we apply a substitution to a plan, we must check that the inequalities do not contradict the substitution. For example, a substitution that includes x/y conflicts with the inequality constraint $x \neq y$. Such conflicts cannot be resolved, so the planner must backtrack.

INEQUALITY CONSTRAINTS

A more extensive example of POP planning with variables in the blocks world is given in Section 12.6.

Heuristics for partial-order planning

Compared with total-order planning, partial-order planning has a clear advantage in being able to decompose problems into subproblems. It also has a disadvantage in that it does not represent states directly, so it is harder to estimate how far a partial-order plan is from achieving a goal. At present, there is less understanding of how to compute accurate heuristics for partial-order planning than for total-order planning.

The most obvious heuristic is to count the number of distinct open preconditions. This can be improved by subtracting the number of open preconditions that match literals in the *Start* state. As in the total-order case, this overestimates the cost when there are actions that achieve multiple goals and underestimates the cost when there are negative interactions between plan steps. The next section presents an approach that allows us to get much more accurate heuristics from a relaxed problem.

The heuristic function is used to choose which plan to refine. Given this choice, the algorithm generates successors based on the selection of a single open precondition to work

on. As in the case of variable selection on constraint satisfaction algorithms, this selection has a large impact on efficiency. The **most-constrained-variable** heuristic from CSPs can be adapted for planning algorithms and seems to work well. The idea is to select the open condition that can be satisfied in the *fewest* number of ways. There are two special cases of this heuristic. First, if an open condition cannot be achieved by any action, the heuristic will select it; this is a good idea because early detection of impossibility can save a great deal of work. Second, if an open condition can be achieved in only one way, then it should be selected because the decision is unavoidable and could provide additional constraints on other choices still to be made. Although full computation of the number of ways to satisfy each open condition is expensive and not always worthwhile, experiments show that handling the two special cases provides very substantial speedups.

11.4 PLANNING GRAPHS

PLANNING GRAPH

All of the heuristics we have suggested for total-order and partial-order planning can suffer from inaccuracies. This section shows how a special data structure called a **planning graph** can be used to give better heuristic estimates. These heuristics can be applied to any of the search techniques we have seen so far. Alternatively, we can extract a solution directly from the planning graph, using a specialized algorithm such as the one called GRAPHPLAN.

LEVELS

A planning graph consists of a sequence of **levels** that correspond to time steps in the plan, where level 0 is the initial state. Each level contains a set of literals and a set of actions. Roughly speaking, the literals are all those that *could* be true at that time step, depending on the actions executed at preceding time steps. Also roughly speaking, the actions are all those actions that *could* have their preconditions satisfied at that time step, depending on which of the literals actually hold. We say "roughly speaking" because the planning graph records only a restricted subset of the possible negative interactions among actions; therefore, it might be optimistic about the minimum number of time steps required for a literal to become true. Nonetheless, this number of steps in the planning graph provides a good estimate of how difficult it is to achieve a given literal from the initial state. More importantly, the planning graph is defined in such a way that it can be constructed very efficiently.

Planning graphs work only for propositional planning problems—ones with no variables. As we mentioned in Section 11.1, both STRIPS and ADL representations can be propositionalized. For problems with large numbers of objects, this could result in a very substantial blowup in the number of action schemata. Despite this, planning graphs have proved to be effective tools for solving hard planning problems.

We will illustrate planning graphs with a simple example. (More complex examples lead to graphs that won't fit on the page.) Figure 11.11 shows a problem, and Figure 11.12 shows its planning graph. We start with state level S_0, which represents the problem's initial state. We follow that with action level A_0, in which we place all the actions whose preconditions are satisfied in the previous level. Each action is connected to its preconditions in S_0 and its effects in S_1, in this case introducing new literals into S_1 that were not in S_0.

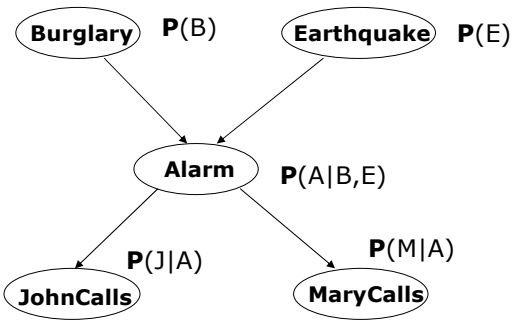
CS 2740 Knowledge representation
Lecture 20

Bayesian belief networks: Inference

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Bayesian belief network.

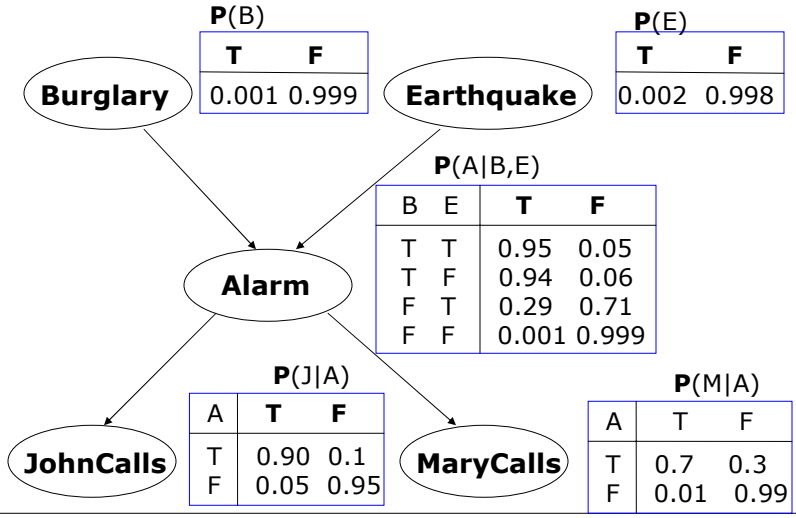
1. **Directed acyclic graph**
   - **Nodes** = random variables
   - **Links** = missing links encode independences.



Bayesian belief network

2. Local conditional distributions

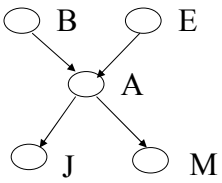
- relate variables and their parents



Full joint distribution in BBNs

Full joint distribution is defined in terms of local conditional distributions (obtained via the chain rule):

$$\mathbf{P}(X_1, X_2, .., X_n) = \prod_{i=1,..n} \mathbf{P}(X_i \mid pa(X_i))$$



Example:

Assume the following assignment of values to random variables

$B = T, E = T, A = T, J = T, M = F$

Then its probability is:

$$P(B=T, E=T, A=T, J=T, M=F) =$$
$$P(B=T)P(E=T)P(A=T \mid B=T, E=T)P(J=T \mid A=T)P(M=F \mid A=T)$$

Parameter complexity problem

- In the BBN the **full joint distribution** is defined as:

$$\mathbf{P}(X_1, X_2, .., X_n) = \prod_{i=1,..n} \mathbf{P}(X_i \mid pa(X_i))$$

- **What did we save?**

Alarm example: 5 binary (True, False) variables

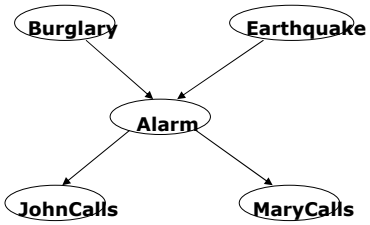
# of parameters of the full joint:

$$2^5 = 32$$

One parameter is for free:

$$2^5 - 1 = 31$$

# of parameters of the BBN: ?



Parameter complexity problem

- In the BBN the **full joint distribution** is defined as:

$$\mathbf{P}(X_1, X_2, .., X_n) = \prod_{i=1,..n} \mathbf{P}(X_i \mid pa(X_i))$$

- **What did we save?**

Alarm example: 5 binary (True, False) variables

# of parameters of the full joint:

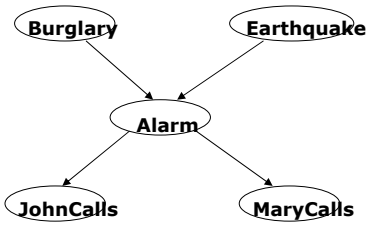
$$2^5 = 32$$

One parameter is for free:

$$2^5 - 1 = 31$$

# of parameters of the BBN:

$$2^3 + 2(2^2) + 2(2) = 20$$



One parameter in every conditional is for free:

?

Parameter complexity problem

- In the BBN the **full joint distribution** is defined as:

$$\mathbf{P}(X_1, X_2, .., X_n) = \prod_{i=1,..n} \mathbf{P}(X_i \mid pa(X_i))$$

- **What did we save?**

Alarm example: 5 binary (True, False) variables

# of parameters of the full joint:

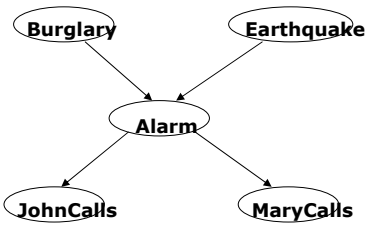
$$2^5 = 32$$

One parameter is for free:

$$2^5 - 1 = 31$$

# of parameters of the BBN:

$$2^3 + 2(2^2) + 2(2) = 20$$



One parameter in every conditional is for free:

$$2^2 + 2(2) + 2(1) = 10$$

Inference in Bayesian networks

- BBN models compactly the full joint distribution by taking advantage of existing independences between variables
  - Smaller number of parameters
- But we are interested in solving various **inference tasks**:
  - **Diagnostic task. (from effect to cause)**

$$\mathbf{P}(Burglary \mid JohnCalls = T)$$

  - **Prediction task. (from cause to effect)**

$$\mathbf{P}(JohnCalls \mid Burglary = T)$$

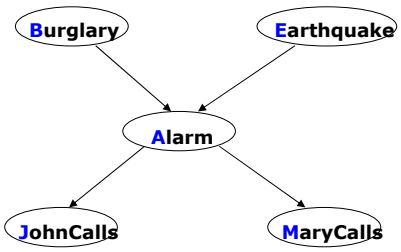
  - **Other probabilistic queries** (queries on joint distributions).

$$\mathbf{P}(Alarm)$$

- **Question:** Can we take advantage of independences to construct special algorithms and speedup the inference?

Inference in Bayesian network

- **Bad news:**
  - Exact inference problem in BBNs is NP-hard (Cooper)
  - Approximate inference is NP-hard (Dagum, Luby)
- **But** very often we can achieve significant improvements
- Assume our Alarm network



- Assume we want to compute: $P(J = T)$

Inference in Bayesian networks

Computing: $P(J = T)$

Approach 1. Blind approach.

- Sum out all un-instantiated variables from the full joint,
- express the joint distribution as a product of conditionals

$$P(J = T) =$$

$$= \sum_{b \in T,F} \sum_{e \in T,F} \sum_{a \in T,F} \sum_{m \in T,F} P(B = b, E = e, A = a, J = T, M = m)$$

$$= \sum_{b \in T,F} \sum_{e \in T,F} \sum_{a \in T,F} \sum_{m \in T,F} P(J = T \mid A = a) P(M = m \mid A = a) P(A = a \mid B = b, E = e) P(B = b) P(E = e)$$

Computational cost:

Number of additions: ?

Number of products: ?

Inference in Bayesian networks

Computing: $P(J = T)$

Approach 1. Blind approach.

- Sum out all un-instantiated variables from the full joint,
- express the joint distribution as a product of conditionals

$$P(J = T) =$$

$$= \sum_{b \in T,F} \sum_{e \in T,F} \sum_{a \in T,F} \sum_{m \in T,F} P(B = b, E = e, A = a, J = T, M = m)$$

$$= \sum_{b \in T,F} \sum_{e \in T,F} \sum_{a \in T,F} \sum_{m \in T,F} P(J = T \mid A = a) P(M = m \mid A = a) P(A = a \mid B = b, E = e) P(B = b) P(E = e)$$

Computational cost:

Number of additions: 15

Number of products: ?

Inference in Bayesian networks

Computing: $P(J = T)$

Approach 1. Blind approach.

- Sum out all un-instantiated variables from the full joint,
- express the joint distribution as a product of conditionals

$$P(J = T) =$$

$$= \sum_{b \in T,F} \sum_{e \in T,F} \sum_{a \in T,F} \sum_{m \in T,F} P(B = b, E = e, A = a, J = T, M = m)$$

$$= \sum_{b \in T,F} \sum_{e \in T,F} \sum_{a \in T,F} \sum_{m \in T,F} P(J = T \mid A = a) P(M = m \mid A = a) P(A = a \mid B = b, E = e) P(B = b) P(E = e)$$

Computational cost:

Number of additions: 15

Number of products: 16*4=64

Inference in Bayesian networks

Approach 2. Interleave sums and products

- Combines sums and product in a smart way (multiplications by constants can be taken out of the sum)

$$P(J=T)=$$

$$=\sum_{b\in T,F}\sum_{e\in T,F}\sum_{a\in T,F}\sum_{m\in T,F}P(J=T\mid A=a)P(M=m\mid A=a)P(A=a\mid B=b,E=e)P(B=b)P(E=e)$$

$$=\sum_{b\in T,F}\sum_{a\in T.F}\sum_{m\in T,F}P(J=T\mid A=a)P(M=m\mid A=a)P(B=b)[\sum_{e\in T,F}P(A=a\mid B=b,E=e)P(E=e)]$$

$$=\sum_{a\in T,F}P(J=T\mid A=a)[\sum_{m\in T,F}P(M=m\mid A=a)][\sum_{b\in T,F}P(B=b)[\sum_{e\in T,F}P(A=a\mid B=b,E=e)P(E=e)]$$

Computational cost:

Number of additions: 1+2*[1+1+2*1]=**?**

Number of products: 2*[2+2*(1+2*1)]=?

Inference in Bayesian networks

Approach 2. Interleave sums and products

- Combines sums and product in a smart way (multiplications by constants can be taken out of the sum)

$$P(J=T)=$$

$$=\sum_{b\in T,F}\sum_{e\in T,F}\sum_{a\in T,F}\sum_{m\in T,F}P(J=T\mid A=a)P(M=m\mid A=a)P(A=a\mid B=b,E=e)P(B=b)P(E=e)$$

$$=\sum_{b\in T,F}\sum_{a\in T.F}\sum_{m\in T,F}P(J=T\mid A=a)P(M=m\mid A=a)P(B=b)[\sum_{e\in T,F}P(A=a\mid B=b,E=e)P(E=e)]$$

$$=\sum_{a\in T,F}P(J=T\mid A=a)[\sum_{m\in T,F}P(M=m\mid A=a)][\sum_{b\in T,F}P(B=b)[\sum_{e\in T,F}P(A=a\mid B=b,E=e)P(E=e)]$$

Computational cost:

Number of additions: 1+2*[1+1+2*1]=**9**

Number of products: 2*[2+2*(1+2*1)]=?

CS 2750 Machine Learning

Inference in Bayesian networks

Approach 2. Interleave sums and products

- Combines sums and product in a smart way (multiplications by constants can be taken out of the sum)

$$P(J=T) =$$

$$= \sum_{b\in T,F}\sum_{e\in T,F}\sum_{a\in T,F}\sum_{m\in T,F} P(J=T\mid A=a)P(M=m\mid A=a)P(A=a\mid B=b,E=e)P(B=b)P(E=e)$$

$$= \sum_{b\in T,F}\sum_{a\in T.F}\sum_{m\in T,F} P(J=T\mid A=a)P(M=m\mid A=a)P(B=b)[\sum_{e\in T,F} P(A=a\mid B=b,E=e)P(E=e)]$$

$$= \sum_{a\in T,F} P(J=T\mid A=a)[\sum_{m\in T,F} P(M=m\mid A=a)][\sum_{b\in T,F} P(B=b)[\sum_{e\in T,F} P(A=a\mid B=b,E=e)P(E=e)]$$

Computational cost:

Number of additions: 1+2*[1+1+2*1]=**9**

Number of products: 2*[2+2*(1+2*1)]=16

Inference in Bayesian networks

- The smart interleaving of sums and products can help us to speed up the computation of joint probability queries
- What if we want to compute: $P(B=T, J=T)$

$$P(B=T, J=T) =$$

$$= \sum_{a\in T,F} P(J=T\mid A=a)[\sum_{m\in T,F} P(M=m\mid A=a)]\left[P(B=T)\left[\sum_{e\in T,F} P(A=a\mid B=T,E=e)P(E=e)\right]\right]$$

$$P(J=T) =$$

$$= \sum_{a\in T,F} P(J=T\mid A=a)[\sum_{m\in T,F} P(M=m\mid A=a)]\left[\sum_{b\in T,F} P(B=b)\left[\sum_{e\in T,F} P(A=a\mid B=b,E=e)P(E=e)\right]\right]$$

- A lot of shared computation
  - Smart cashing of results can save the time for more queries

Inference in Bayesian network

- **Exact inference algorithms:**
  - **Variable elimination**
  - Recursive decomposition (Cooper, Darwiche)
  - Belief propagation algorithm (Pearl)
  - Arc reversal (Olmsted, Schachter)

- **Approximate inference algorithms:**
  - **Monte Carlo methods:**
    - Forward sampling, Likelihood sampling
  - Variational methods

Variable elimination

- **Variable elimination:**
  - Interleave sum and products one variable at the time during the inference
  - Typically relies on a special structure (called **joint tree**) that groups together multiple variables
  - E.g. Query $P(J=T)$ requires to eliminate A,B,E,M and this can be done in different order

$$P(J=T) =$$

$$= \sum_{b\in T,F} \sum_{e\in T,F} \sum_{a\in T,F} \sum_{m\in T,F} P(J=T \mid A=a)P(M=m \mid A=a)P(A=a \mid B=b, E=e)P(B=b)P(E=e)$$

Variable elimination

Assume the elimination order: M, E, B,A to calculate $P(J=T)$

$$= \sum_{b\in T,F}\sum_{e\in T,F}\sum_{a\in T,F}\sum_{m\in T,F} P(J=T\mid A=a)P(M=m\mid A=a)P(A=a\mid B=b,E=e)P(B=b)P(E=e)$$

$$= \sum_{b\in T,F}\sum_{e\in T,F}\sum_{a\in T,F} P(J=T\mid A=a)P(A=a\mid B=b,E=e)P(B=b)P(E=e)\left[\sum_{m\in T,F} P(M=m\mid A=a)\right]$$

$$= \sum_{b\in T,F}\sum_{e\in T,F}\sum_{a\in T,F} P(J=T\mid A=a)P(A=a\mid B=b,E=e)P(B=b)P(E=e)\quad 1$$

$$= \sum_{a\in T,F}\sum_{b\in T,F} P(J=T\mid A=a)P(B=b)\left[\sum_{e\in T,F} P(A=a\mid B=b,E=e)P(E=e)\right]$$

$$= \sum_{a\in T,F}\sum_{b\in T,F} P(J=T\mid A=a)P(B=b)\,\tau_1(A=a,B=b)$$

$$= \sum_{a\in T,F} P(J=T\mid A=a)\left[\sum_{e\in T,F} P(B=b)\,\tau_1(A=a,B=b)\right]$$

$$= \sum_{a\in T,F} P(J=T\mid A=a)\quad \tau_2(A=a)$$

Factors

- **Factor:** is a function that maps value assignments for a subset of random variables to $\Re$ (reals)
- **The scope of the factor:**
  - a set of variables defining the factor
- **Example:**
  - Assume discrete random variables x (with values a1,a2, a3) and y (with values b1 and b2)
  - Factor:

$\phi(x, y)$ →

a1	b1	0.5
a1	b2	0.2
a2	b1	0.1
a2	b2	0.3
a3	b1	0.2
a3	b2	0.4

  - Scope of the factor:

$\{x, y\}$

Factor Product

$$\phi_1(x, y)\phi_2(y, z) = \tau(x, y, z)$$

a1	b1	0.5
a1	b2	0.2
a2	b1	0.1
a2	b2	0.3
a3	b1	0.2
a3	b2	0.4

•

b1	c1	0.1
b1	c2	0.6
b2	c1	0.3
b2	c2	0.4

=

a1	b1	c1	0.5*0.1
a1	b1	c2	0.5*0.6
a1	b2	c1	0.2*0.3
a1	b2	c2	0.2*0.4
a2	b1	c1	0.1*0.1
a2	b1	c2	0.1*0.6
a2	b2	c1	0.3*0.3
a2	b2	c2	0.3*0.4
a3	b1	c1	0.2*0.1
a3	b1	c2	0.2*0.6
a3	b2	c1	0.4*0.3
a3	b2	c2	0.4*0.4

Factor Product

$$\phi_1(x, y)\phi_2(y, z) = \tau(x, y, z)$$

a1	b1	0.5
a1	b2	0.2
a2	b1	0.1
a2	b2	0.3
a3	b1	0.2
a3	b2	0.4

•

b1	c1	0.1
b1	c2	0.6
b2	c1	0.3
b2	c2	0.4

=

a1	b1	c1	0.5*0.1
a1	b1	c2	0.5*0.6
a1	b2	c1	0.2*0.3
a1	b2	c2	0.2*0.4
a2	b1	c1	0.1*0.1
a2	b1	c2	0.1*0.6
a2	b2	c1	0.3*0.3
a2	b2	c2	0.3*0.4
a3	b1	c1	0.2*0.1
a3	b1	c2	0.2*0.6
a3	b2	c1	0.4*0.3
a3	b2	c2	0.4*0.4

CS 2750 Machine Learning

Factor Sum (marginalization)

$$\sum_{y} \phi(x, y, z) = \tau(x, z)$$

$\sum$

a1	b1	c1	0.2
a1	b1	c2	0.35
a1	b2	c1	0.4
a1	b2	c2	0.15
a2	b1	c1	0.5
a2	b1	c2	0.1
a2	b2	c1	0.3
a2	b2	c2	0.2
a3	b1	c1	0.25
a3	b1	c2	0.45
a3	b2	c1	0.15
a3	b2	c2	0.25

=

a1	c1	0.6
a1	c2	0.5
a2	c1	0.8
a2	c2	0.3
a3	c1	0.4
a3	c2	0.7

Factor Sum (marginalization)

$$\sum_{y} \phi(x, y, z) = \tau(x, z)$$

$\sum$

a1	b1	c1	0.2
a1	b1	c2	0.35
a1	b2	c1	0.4
a1	b2	c2	0.15
a2	b1	c1	0.5
a2	b1	c2	0.1
a2	b2	c1	0.3
a2	b2	c2	0.2
a3	b1	c1	0.25
a3	b1	c2	0.45
a3	b2	c1	0.15
a3	b2	c2	0.25

=

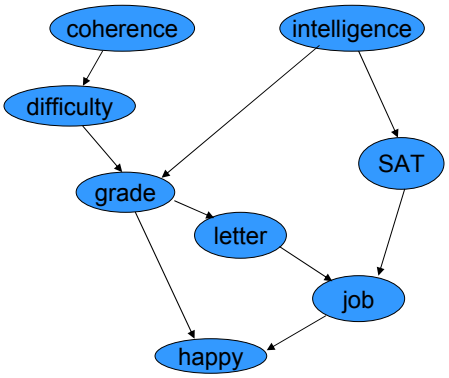
a1	c1	0.6
a1	c2	0.5
a2	c1	0.8
a2	c2	0.3
a3	c1	0.4
a3	c2	0.7

CS 2750 Machine Learning

Variable elimination

The order in which variables are eliminated may effect the efficiency of the variable elimination process

Assume the following BBN and calculation of *P(Job)*:



Variable elimination

Calculations performed in terms of factors:

$$p(J) = \sum_{L,S,G,H,I,D,C} \phi(c)\phi(i)\phi(d,c)\phi(g,i,d)\phi(s,i)\phi(l,g)\phi(j,l,s)\phi(h,g,j)$$

$$= \sum_{L,S,G,H,I,D} \phi(i)\phi(g,i,d)\phi(s,i)\phi(l,g)\phi(j,l,s)\phi(h,g,j)\sum_{C} \phi(c)\phi(d,c)$$

$$= \sum_{L,S,G,H,I,D} \phi(i)\phi(g,i,d)\phi(s,i)\phi(l,g)\phi(j,l,s)\phi(h,g,j)\tau(d)$$

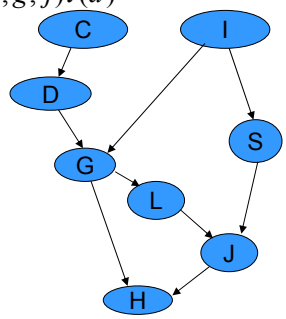
...

$$= \sum_{L,S} \phi(j,l,s)\sum_{G} \phi(l,g)\tau(s,g)\tau(g,j)$$

$$= \sum_{L,S} \phi(j,l,s)\tau(l,s,j)$$

$$= \sum_{L} \tau(l,j)$$

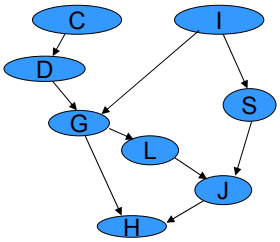
$$= \tau(j)$$



Variable elimination

Trace 1:

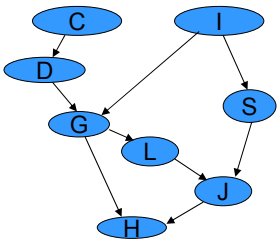
Step	Var	Factors Used	New Factor
1	C	$\phi_c(C), \phi_D(D,C)$	$\tau_1(D)$
2	D	$\phi_G(G,I,D), \tau_1(D)$	$\tau_2(G,I)$
3	I	$\phi_I(I), \phi_S(S,I), \tau_2(G,I)$	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	$\tau_4(G,J)$
5	G	$\tau_4(G,J), \tau_3(G,S), \phi_L(L,G)$	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S), \phi_J(J,L,S)$	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	$\tau_7(J)$



Variable elimination

Trace 1:

Step	Var	Factors Used	New Factor
1	C	$\phi_c(C), \phi_D(D,C)$	$\tau_1(D)$
2	D	$\phi_G(G,I,D), \tau_1(D)$	$\tau_2(G,I)$
3	I	$\phi_I(I), \phi_S(S,I), \tau_2(G,I)$	$\tau_3(G,S)$
4	H	$\phi_H(H,G,J)$	$\tau_4(G,J)$
5	G	$\tau_4(G,J), \tau_3(G,S), \phi_L(L,G)$	$\tau_5(J,L,S)$
6	S	$\tau_5(J,L,S), \phi_J(J,L,S)$	$\tau_6(J,L)$
7	L	$\tau_6(J,L)$	$\tau_7(J)$

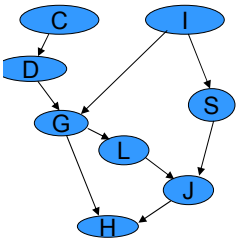


Complexity: 4 variables – 1 summed away

Variable elimination

Trace 2:

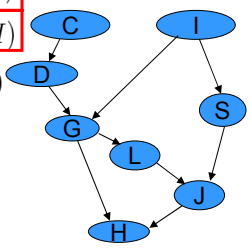
Step	Var	Factors Used	New Factor
1	G	$\phi_G(G,I,D), \phi_L(L,G)\phi_H(H,G,J)$	$\tau_1(I,D,L,J,H)$
2	I	$\phi_I(I), \phi_S(S,I)\tau_1(I,D,L,J,H)$	$\tau_2(D,L,S,J,H)$
3	S	$\phi_J(J,L,S), \tau_2(D,L,S,J,H)$	$\tau_3(D,L,J,H)$
4	L	$\tau_3(D,L,J,H)$	$\tau_4(D,J,H)$
5	H	$\tau_4(D,J,H)$	$\tau_5(D,J)$
6	C	$\tau_5(D,J), \phi_D(D,C)$	$\tau_6(D,J)$
7	D	$\tau_6(D,J)$	$\tau_7(J)$



Variable elimination

Trace 2:

Step	Var	Factors Used	New Factor
1	G	$\phi_G(G,I,D), \phi_L(L,G)\phi_H(H,G,J)$	$\tau_1(I,D,L,J,H)$
2	I	$\phi_I(I), \phi_S(S,I)\tau_1(I,D,L,J,H)$	$\tau_2(D,L,S,J,H)$
3	S	$\phi_J(J,L,S), \tau_2(D,L,S,J,H)$	$\tau_3(D,L,J,H)$
4	L	$\tau_3(D,L,J,H)$	$\tau_4(D,J,H)$
5	H	$\tau_4(D,J,H)$	$\tau_5(D,J)$
6	C	$\tau_5(D,J), \phi_D(D,C)$	$\tau_6(D,J)$
7	D	$\tau_6(D,J)$	$\tau_7(J)$



Complexity: 6 variables used – 1 summed out

Types of Machine Learning

Last Updated : 29 Nov, 2023

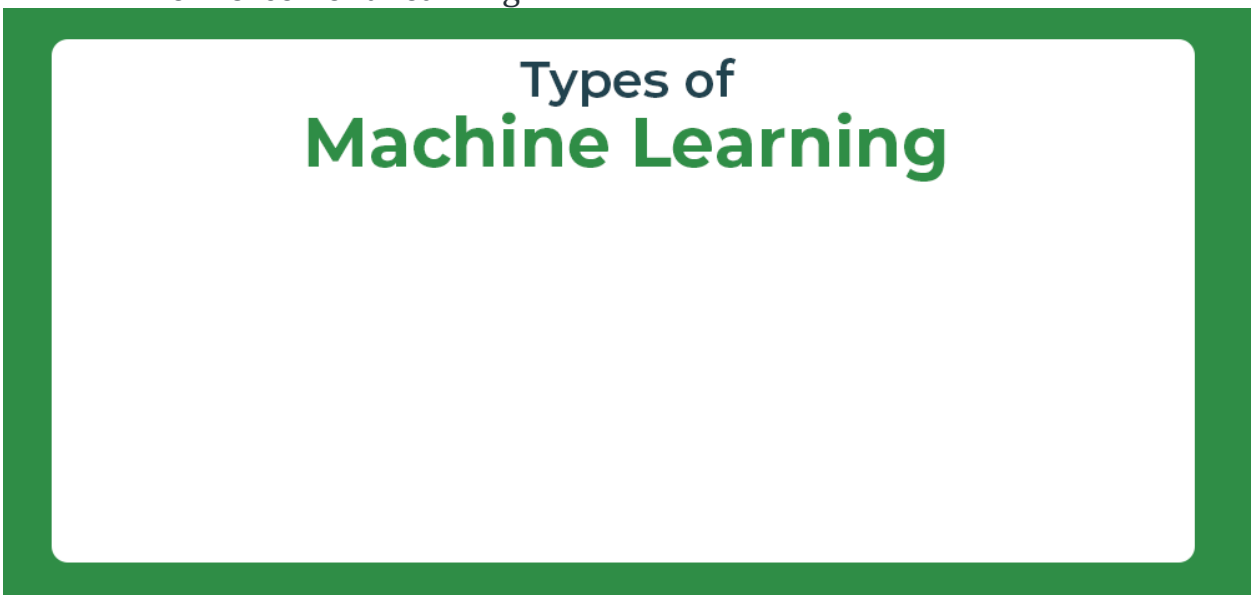
Machine learning is the branch of Artificial Intelligence that focuses on developing models and algorithms that let computers learn from data and improve from previous experience without being explicitly programmed for every task. In simple words, ML teaches the systems to think and understand like humans by learning from the data.

In this article, we will explore the various **types of machine learning algorithms** that are important for future requirements. **Machine learning** is generally a training system to learn from past experiences and improve performance over time. Machine learning helps to predict massive amounts of data. It helps to deliver fast and accurate results to get profitable opportunities.

Types of Machine Learning

There are several types of machine learning, each with special characteristics and applications. Some of the main types of machine learning algorithms are as follows:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Machine Learning
4. Reinforcement Learning

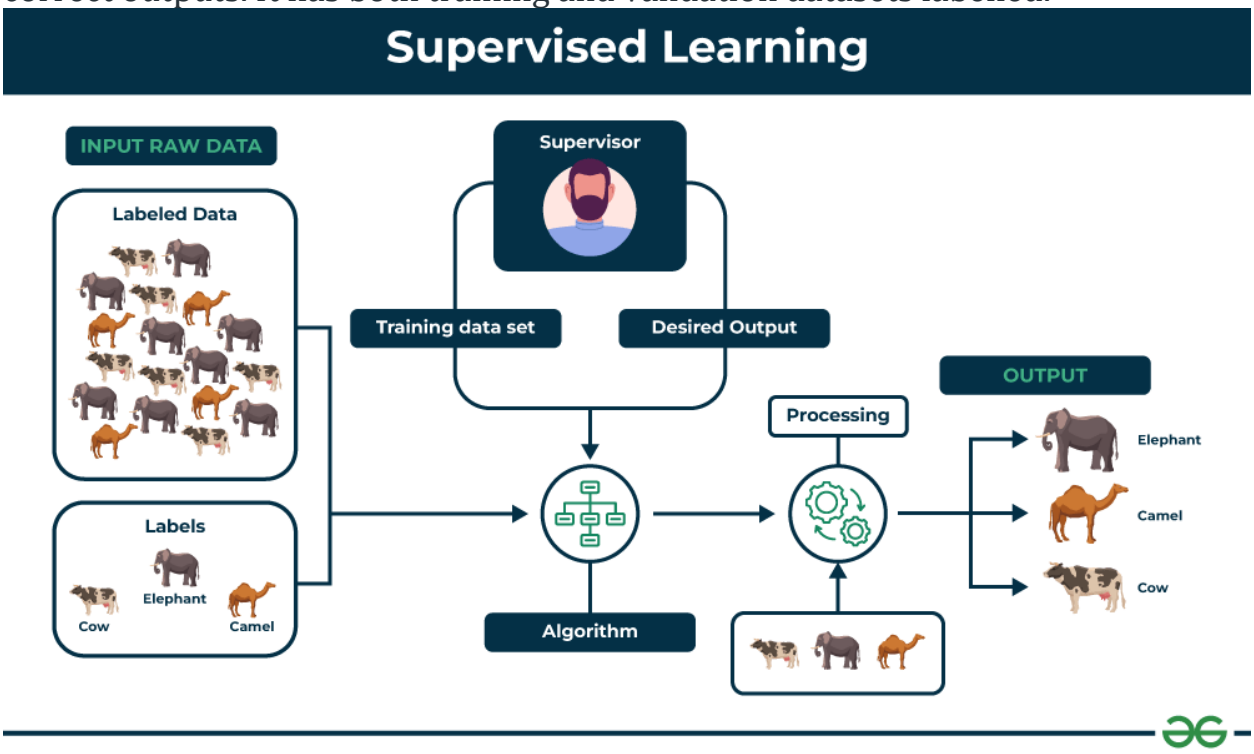


Types of Machine Learning

1. Supervised Machine Learning

Supervised learning is defined as when a model gets trained on a **"Labelled Dataset"**. Labelled datasets have both input and output parameters.

In **Supervised Learning** algorithms learn to map points between inputs and correct outputs. It has both training and validation datasets labelled.



Supervised Learning

Let's understand it with the help of an example.

Example: Consider a scenario where you have to build an image classifier to differentiate between cats and dogs. If you feed the datasets of dogs and cats labelled images to the algorithm, the machine will learn to classify between a dog or a cat from these labeled images. When we input new dog or cat images that it has never seen before, it will use the learned algorithms and predict whether it is a dog or a cat. This is how **supervised learning** works, and this is particularly an image classification.

There are two main categories of supervised learning that are mentioned below:

- Classification
- Regression

Classification

Classification deals with predicting **categorical** target variables, which represent discrete classes or labels. For instance, classifying emails as spam or not spam, or predicting whether a patient has a high risk of heart disease. Classification algorithms learn to map the input features to one of the predefined classes.

Here are some classification algorithms:

- **Logistic Regression**

- **Support Vector Machine**
- **Random Forest**
- **Decision Tree**
- **K-Nearest Neighbors (KNN)**
- **Naive Bayes**

Regression

Regression, on the other hand, deals with predicting **continuous** target variables, which represent numerical values. For example, predicting the price of a house based on its size, location, and amenities, or forecasting the sales of a product. Regression algorithms learn to map the input features to a continuous numerical value.

Here are some regression algorithms:

- **Linear Regression**
- **Polynomial Regression**
- **Ridge Regression**
- **Lasso Regression**
- **Decision tree**
- **Random Forest**

Advantages of Supervised Machine Learning

- **Supervised Learning** models can have high accuracy as they are trained on **labelled data**.
- The process of decision-making in supervised learning models is often interpretable.
- It can often be used in pre-trained models which saves time and resources when developing new models from scratch.

Disadvantages of Supervised Machine Learning

- It has limitations in knowing patterns and may struggle with unseen or unexpected patterns that are not present in the training data.
- It can be time-consuming and costly as it relies on **labeled** data only.
- It may lead to poor generalizations based on new data.

Applications of Supervised Learning

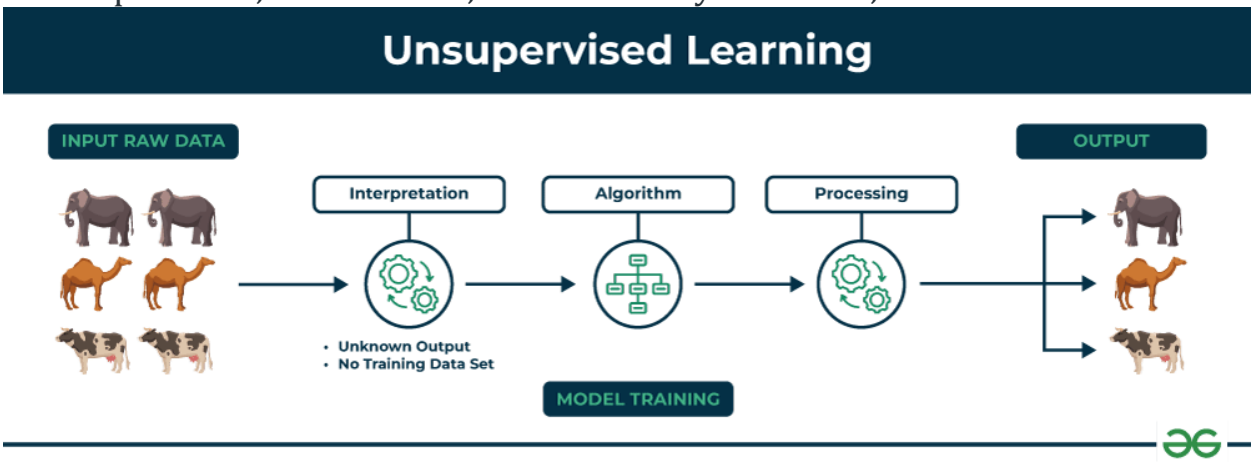
Supervised learning is used in a wide variety of applications, including:

- **Image classification**: Identify objects, faces, and other features in images.
- **Natural language processing:** Extract information from text, such as sentiment, entities, and relationships.
- **Speech recognition**: Convert spoken language into text.
- **Recommendation systems**: Make personalized recommendations to users.
- **Predictive analytics**: Predict outcomes, such as sales, customer churn, and stock prices.

- **Medical diagnosis**: Detect diseases and other medical conditions.
- **Fraud detection**: Identify fraudulent transactions.
- **Autonomous vehicles**: Recognize and respond to objects in the environment.
- **Email spam detection**: Classify emails as spam or not spam.
- **Quality control in manufacturing**: Inspect products for defects.
- **Credit scoring**: Assess the risk of a borrower defaulting on a loan.
- **Gaming**: Recognize characters, analyze player behavior, and create NPCs.
- **Customer support**: Automate customer support tasks.
- **Weather forecasting**: Make predictions for temperature, precipitation, and other meteorological parameters.
- **Sports analytics**: Analyze player performance, make game predictions, and optimize strategies.

2. Unsupervised Machine Learning

Unsupervised Learning Unsupervised learning is a type of machine learning technique in which an algorithm discovers patterns and relationships using unlabeled data. Unlike supervised learning, unsupervised learning doesn't involve providing the algorithm with labeled target outputs. The primary goal of Unsupervised learning is often to discover hidden patterns, similarities, or clusters within the data, which can then be used for various purposes, such as data exploration, visualization, dimensionality reduction, and more.



Unsupervised Learning

Let's understand it with the help of an example.

Example: Consider that you have a dataset that contains information about the purchases you made from the shop. Through clustering, the algorithm can group the same purchasing behavior among you and other customers, which reveals potential customers without predefined labels. This type of information can help businesses get target customers as well as identify outliers.

There are two main categories of unsupervised learning that are mentioned below:

- Clustering
- Association

Clustering

Clustering is the process of grouping data points into clusters based on their similarity. This technique is useful for identifying patterns and relationships in data without the need for labeled examples.

Here are some clustering algorithms:

- **K-Means Clustering algorithm**
- **Mean-shift algorithm**
- **DBSCAN Algorithm**
- **Principal Component Analysis**
- **Independent Component Analysis**

Association

Association rule learning is a technique for discovering relationships between items in a dataset. It identifies rules that indicate the presence of one item implies the presence of another item with a specific probability.

Here are some association rule learning algorithms:

- **Apriori Algorithm**
- **Eclat**
- **FP-growth Algorithm**

Advantages of Unsupervised Machine Learning

- It helps to discover hidden patterns and various relationships between the data.
- Used for tasks such as **customer segmentation, anomaly detection,** and **data exploration**.
- It does not require labeled data and reduces the effort of data labeling.

Disadvantages of Unsupervised Machine Learning

- Without using labels, it may be difficult to predict the quality of the model's output.
- Cluster Interpretability may not be clear and may not have meaningful interpretations.
- It has techniques such as autoencoders and dimensionality reduction that can be used to extract meaningful features from raw data.

Applications of Unsupervised Learning

Here are some common applications of unsupervised learning:

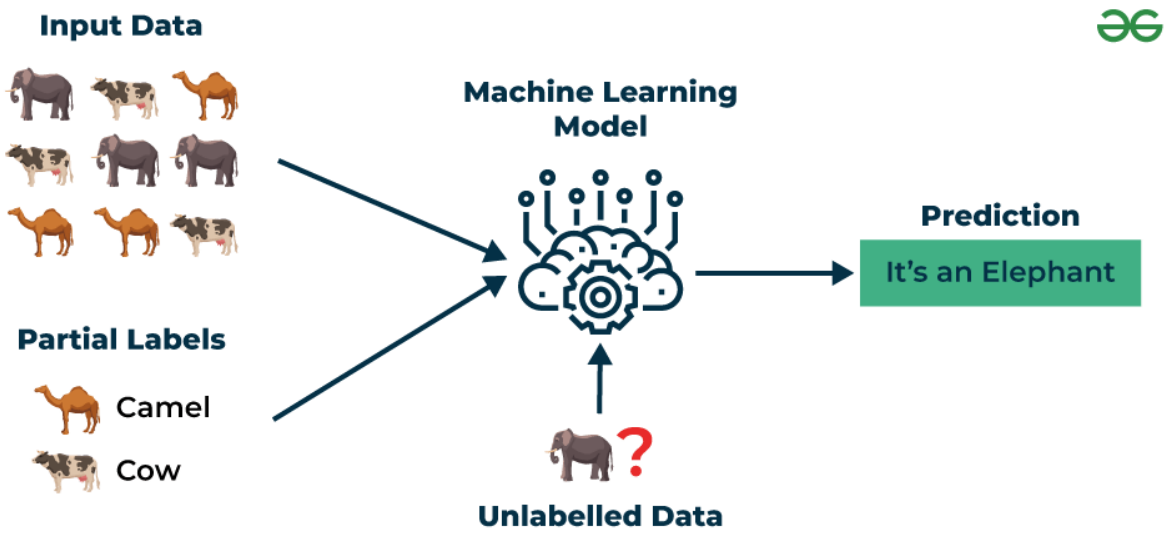
- **Clustering**: Group similar data points into clusters.
- **Anomaly detection**: Identify outliers or anomalies in data.

- **Dimensionality reduction**: Reduce the dimensionality of data while preserving its essential information.
- **Recommendation systems**: Suggest products, movies, or content to users based on their historical behavior or preferences.
- **Topic modeling**: Discover latent topics within a collection of documents.
- **Density estimation**: Estimate the probability density function of data.
- **Image and video compression**: Reduce the amount of storage required for multimedia content.
- **Data preprocessing**: Help with data preprocessing tasks such as data cleaning, imputation of missing values, and data scaling.
- **Market basket analysis**: Discover associations between products.
- **Genomic data analysis**: Identify patterns or group genes with similar expression profiles.
- **Image segmentation**: Segment images into meaningful regions.
- **Community detection in social networks**: Identify communities or groups of individuals with similar interests or connections.
- **Customer behavior analysis**: Uncover patterns and insights for better marketing and product recommendations.
- **Content recommendation**: Classify and tag content to make it easier to recommend similar items to users.
- **Exploratory data analysis (EDA)**: Explore data and gain insights before defining specific tasks.

3. Semi-Supervised Learning

Semi-Supervised learning is a machine learning algorithm that works between the supervised and unsupervised learning so it uses both **labelled and unlabelled** data. It's particularly useful when obtaining labeled data is costly, time-consuming, or resource-intensive. This approach is useful when the dataset is expensive and time-consuming. Semi-supervised learning is chosen when labeled data requires skills and relevant resources in order to train or learn from it.

We use these techniques when we are dealing with data that is a little bit labeled and the rest large portion of it is unlabeled. We can use the unsupervised techniques to predict labels and then feed these labels to supervised techniques. This technique is mostly applicable in the case of image data sets where usually all images are not labeled.



Semi-Supervised Learning

Let's understand it with the help of an example.
Example: Consider that we are building a language translation model, having labeled translations for every sentence pair can be resources intensive. It allows the models to learn from labeled and unlabeled sentence pairs, making them more accurate. This technique has led to significant improvements in the quality of machine translation services.

Types of Semi-Supervised Learning Methods

There are a number of different semi-supervised learning methods each with its own characteristics. Some of the most common ones include:

- **Graph-based semi-supervised learning:** This approach uses a graph to represent the relationships between the data points. The graph is then used to propagate labels from the labeled data points to the unlabeled data points.
- **Label propagation:** This approach iteratively propagates labels from the labeled data points to the unlabeled data points, based on the similarities between the data points.
- **Co-training:** This approach trains two different machine learning models on different subsets of the unlabeled data. The two models are then used to label each other's predictions.
- **Self-training:** This approach trains a machine learning model on the labeled data and then uses the model to predict labels for the unlabeled data. The model is then retrained on the labeled data and the predicted labels for the unlabeled data.
- **Generative adversarial networks (GANs):** GANs are a type of deep learning algorithm that can be used to generate synthetic data. GANs

can be used to generate unlabeled data for semi-supervised learning by training two neural networks, a generator and a discriminator.

Advantages of Semi- Supervised Machine Learning

- It leads to better generalization as compared to **supervised learning,** as it takes both labeled and unlabeled data.
- Can be applied to a wide range of data.

Disadvantages of Semi- Supervised Machine Learning

- **Semi-supervised** methods can be more complex to implement compared to other approaches.
- It still requires some **labeled data** that might not always be available or easy to obtain.
- The unlabeled data can impact the model performance accordingly.

Applications of Semi-Supervised Learning

Here are some common applications of semi-supervised learning:

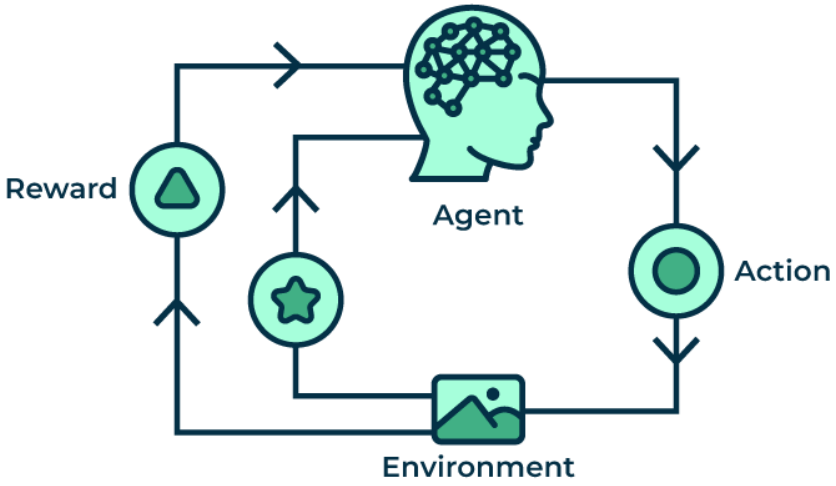
- **Image Classification and Object Recognition**: Improve the accuracy of models by combining a small set of labeled images with a larger set of unlabeled images.
- **Natural Language Processing (NLP)**: Enhance the performance of language models and classifiers by combining a small set of labeled text data with a vast amount of unlabeled text.
- **Speech Recognition:** Improve the accuracy of speech recognition by leveraging a limited amount of transcribed speech data and a more extensive set of unlabeled audio.
- **Recommendation Systems**: Improve the accuracy of personalized recommendations by supplementing a sparse set of user-item interactions (labeled data) with a wealth of unlabeled user behavior data.
- **Healthcare and Medical Imaging**: Enhance medical image analysis by utilizing a small set of labeled medical images alongside a larger set of unlabeled images.

4. Reinforcement Machine Learning

Reinforcement machine learning algorithm is a learning method that interacts with the environment by producing actions and discovering errors. **Trial, error, and delay** are the most relevant characteristics of reinforcement learning. In this technique, the model keeps on increasing its performance using Reward Feedback to learn the behavior or pattern. These algorithms are specific to a particular problem e.g. Google Self Driving car, AlphaGo where a bot competes with humans and even itself to get better and better performers in Go Game. Each time we feed in data, they learn and add the data to their knowledge which is training data. So, the more it learns the better it gets trained and hence experienced.

Here are some of most common reinforcement learning algorithms:

- **Q-learning:** Q-learning is a model-free RL algorithm that learns a Q-function, which maps states to actions. The Q-function estimates the expected reward of taking a particular action in a given state.
- **SARSA (State-Action-Reward-State-Action):** SARSA is another model-free RL algorithm that learns a Q-function. However, unlike Q-learning, SARSA updates the Q-function for the action that was actually taken, rather than the optimal action.
- **Deep Q-learning:** Deep Q-learning is a combination of Q-learning and deep learning. Deep Q-learning uses a neural network to represent the Q-function, which allows it to learn complex relationships between states and actions.



Reinforcement Machine Learning

Let's understand it with the help of examples.

Example: Consider that you are training an AI agent to play a game like chess. The agent explores different moves and receives positive or negative feedback based on the outcome. Reinforcement Learning also finds applications in which they learn to perform tasks by interacting with their surroundings.

Types of Reinforcement Machine Learning

There are two main types of reinforcement learning:

Positive reinforcement

- Rewards the agent for taking a desired action.
- Encourages the agent to repeat the behavior.
- Examples: Giving a treat to a dog for sitting, providing a point in a game for a correct answer.

Negative reinforcement

- Removes an undesirable stimulus to encourage a desired behavior.
- Discourages the agent from repeating the behavior.
- Examples: Turning off a loud buzzer when a lever is pressed, avoiding a penalty by completing a task.

Advantages of Reinforcement Machine Learning

- It has autonomous decision-making that is well-suited for tasks and that can learn to make a sequence of decisions, like robotics and game-playing.
- This technique is preferred to achieve long-term results that are very difficult to achieve.
- It is used to solve a complex problems that cannot be solved by conventional techniques.

Disadvantages of Reinforcement Machine Learning

- Training Reinforcement Learning agents can be computationally expensive and time-consuming.
- Reinforcement learning is not preferable to solving simple problems.
- It needs a lot of data and a lot of computation, which makes it impractical and costly.

Applications of Reinforcement Machine Learning

Here are some applications of reinforcement learning:

- **Game Playing**: RL can teach agents to play games, even complex ones.
- **Robotics**: RL can teach robots to perform tasks autonomously.
- **Autonomous Vehicles**: RL can help self-driving cars navigate and make decisions.
- **Recommendation Systems**: RL can enhance recommendation algorithms by learning user preferences.
- **Healthcare**: RL can be used to optimize treatment plans and drug discovery.
- **Natural Language Processing (NLP)**: RL can be used in dialogue systems and chatbots.
- **Finance and Trading**: RL can be used for algorithmic trading.
- **Supply Chain and Inventory Management**: RL can be used to optimize supply chain operations.
- **Energy Management**: RL can be used to optimize energy consumption.
- **Game AI**: RL can be used to create more intelligent and adaptive NPCs in video games.
- **Adaptive Personal Assistants**: RL can be used to improve personal assistants.
- **Virtual Reality (VR) and Augmented Reality (AR):** RL can be used to create immersive and interactive experiences.
- **Industrial Control**: RL can be used to optimize industrial processes.

- **Education**: RL can be used to create adaptive learning systems.
- **Agriculture**: RL can be used to optimize agricultural operations.

***Must check, our detailed article on**: Machine Learning Algorithms*

Conclusion

In conclusion, each type of machine learning serves its own purpose and contributes to the overall role in development of enhanced data prediction capabilities, and it has the potential to change various industries like Data Science. It helps deal with massive data production and management of the datasets.