

Artificial Intelligence

Course Outcomes	
CO1	Understanding about concept and definition AI along with its agents , its structure and environments (L2)
CO2	Apply various AI search algorithms (uninformed, informed, heuristic, constraint satisfaction, genetic algorithms). (L3)
CO3	Apply the reinforcement learning and natural language processing in AI. (L3)
CO4	Apply the natural language for communication and understand the perception of image formation, reconstruction of 3D world etc., (L3, L2)
CO5	Understand the Robotic perception like planning to move etc., and to understand the Philosophical foundation of AI. (L2)

ARTIFICIAL INTELLIGENCE

UNIT-I

What is artificial intelligence?

1. Artificial Intelligence is the branch of computer science concerned with making computers behave like humans.
2. Major AI textbooks define artificial intelligence as "the study and design of Intelligent agents," where an intelligent agent is a system that perceives its environment and takes actions which maximize its chances of success.
3. John McCarthy, who coined the term in 1956, defines it as "the science and engineering of making intelligent machines, especially intelligent computer programs."
4. The definitions of AI according to some text books are categorized into approaches and are summarized in the table below :

Systems that think like humans "The exciting new effort to make computers think ... machines with minds, in the full and literal sense."(Haugeland,1985)	Systems that think rationally "The study of mental faculties through the use of computer models." (Charniak and McDermont,1985)
Systems that act like humans The art of creating machines that performs functions that require intelligence when performed by people."(Kurzweil,1990)	Systems that act rationally "Computational intelligence is the study of the design of intelligent agents."(Poole et al.,1998)

Goals of AI

- To Create Expert Systems – The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- To Implement Human Intelligence in Machines – Creating systems that understand, think, learn, and behave like humans.

Applications of Artificial Intelligence:

1. **Gaming** – AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.

2. **Natural Language Processing** – It is possible to interact with the computer that understands natural language spoken by humans.
3. **Expert Systems** – There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
4. **Vision Systems** – These systems understand, interpret, and comprehend visual input on the computer. For example,
 5. A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
 6. Doctors use clinical expert system to diagnose the patient.
 7. Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
8. **Speech Recognition** – Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
9. **Handwriting Recognition** – The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
10. **Intelligent Robots** – Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

History of AI

Maturation of Artificial Intelligence (1943-1952)

1. **Year 1943:** The first work which is now recognized as AI was done by Warren McCulloch and Walter Pitts in 1943. They proposed a model of **artificial neurons**.
2. **Year 1949:** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
3. **Year 1950:** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes "**Computing Machinery and Intelligence**" in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.

The birth of Artificial Intelligence (1952-1956)

1. **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program" which was named as "**Logic Theorist**". This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
2. **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

The golden years-Early enthusiasm (1956-1974)

1. **Year 1966:** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first chatbot in 1966, which was named as ELIZA.
2. **Year 1972:** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.

The first AI winter (1974-1980)

1. The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
2. During AI winters, an interest of publicity on artificial intelligence was decreased.

A boom of AI (1980-1987)

1. **Year 1980:** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
2. In the Year 1980, the first national conference of the American Association of Artificial Intelligence **was held at Stanford University**.

The second AI winter (1987-1993)

1. The duration between the years 1987 to 1993 was the second AI Winter duration.
2. Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The emergence of intelligent agents (1993-2011)

1. **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
2. **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
3. **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

Deep learning, big data and artificial general intelligence (2011-present)

1. **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
2. **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
3. **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
4. **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
5. Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.

Agents and environments:

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

1. A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
2. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
3. A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

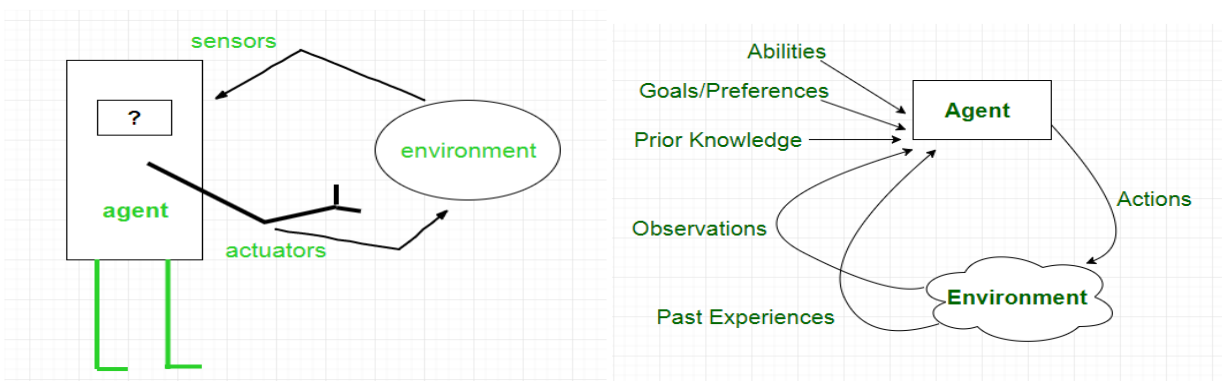


FIG:AGENT WORKING

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

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

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

Percept

We use the term **percept** to refer to the agent's perceptual inputs at any given instant.
Percept Sequence

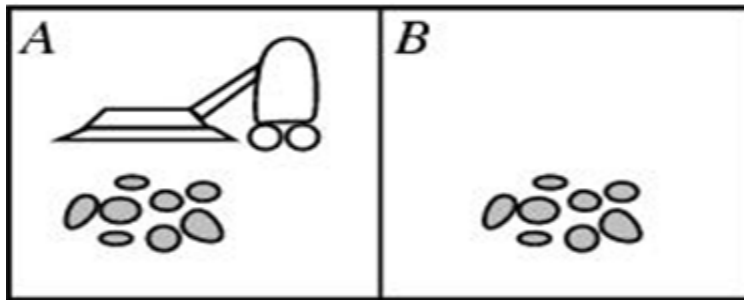
An agent's **percept sequence** is the complete history of everything the agent has ever perceived.

Agent function

Mathematically speaking, we say that an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.

Agent program

1. The agent function for an artificial agent will be implemented by an agent program.
2. It is important to keep these two ideas distinct.
3. The agent function is an abstract mathematical description;
4. the agent program is a concrete implementation, running on the agent architecture.
5. To illustrate these ideas, we will use a very simple example-the vacuum-cleaner world shown in Figure.
6. This particular world has just two locations: squares A and B.
7. The vacuum agent perceives which square it is in and whether there is dirt in square.
8. It can choose to move left, move right, suck up the dirt, or do nothing.
9. One very simple agent function is the following:
10. if the current square is dirty, then suck, otherwise,
11. it move to the other square.
12. A partial tabulation of this agent function is shown in Figure.



Agent function

Percept Sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
.....

Agent program

```

function Reflex-VACUUM-AGENT ([locations, status]) returns an
action if status = Dirty then return Suck
else if location = A then return Right
elseif location = B then return Left
    
```

Intelligent Agents:

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

Following are the main four rules for an AI agent:

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

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:

$$\text{Agent} = \text{Architecture} + \text{Agent program}$$

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.

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

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.

Good Behavior: The concept of Rationality

Rationality:

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

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

Performance measures

1. A performance measure embodies the criterion for success of an agent's behavior.
2. When an agent is plunked down in an environment, it generates a sequence of ~~acts~~ actions according to the percepts it receives.
3. This sequence of actions causes the environment to go through a sequence of states.
4. If the sequence is desirable, then the agent has performed well.

PEAS Representation

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

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

Examples:

1) Self Driving car

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.

2) Medical Diagnose

Performance: Healthy patient, Minimized cost

Environment: Patient, Hospital, Staff

Actuators: Tests, Treatments

Sensors: Keyboard (Entry of symptoms)

3) Vacuum Cleaner

Performance: Cleanness, Efficiency, Battery life, Security

Environment: Patient, Hospital, Staff

Actuators: Wheels, Brushes, Vacuum, Extractor

Sensors: Camera, Dirt detection sensor, Cliff sensor, Bump Sensor, Infrared Wall Sensor

4) Part -picking Robot

Performance: Percentage of parts in correct bins.

Environment: Conveyor belt with parts, Bins

Actuators: Jointed Arms, Hand

Sensors: Camera, Joint angle sensors.

Types of Agents

Agents can be grouped into four classes based on their degree of perceived intelligence and capability :

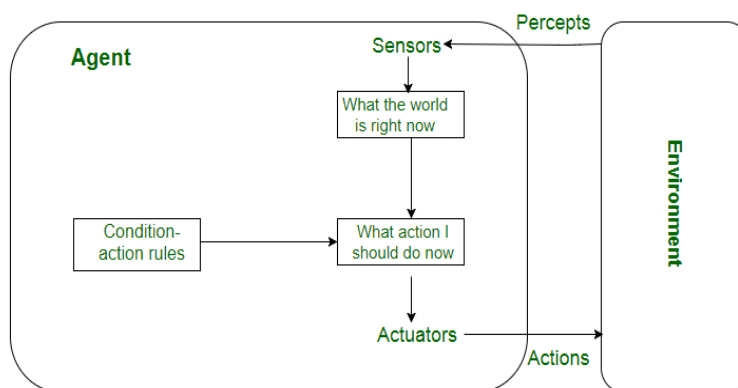
- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents
- Learning Agent

Simple reflex agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the **current percept**. Percept history is the history of all that an agent has perceived to date. The agent function is based on the **condition-action rule**. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions.

Problems with Simple reflex agents are :

- Very limited intelligence.
- No knowledge of non-perceptual parts of the state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.



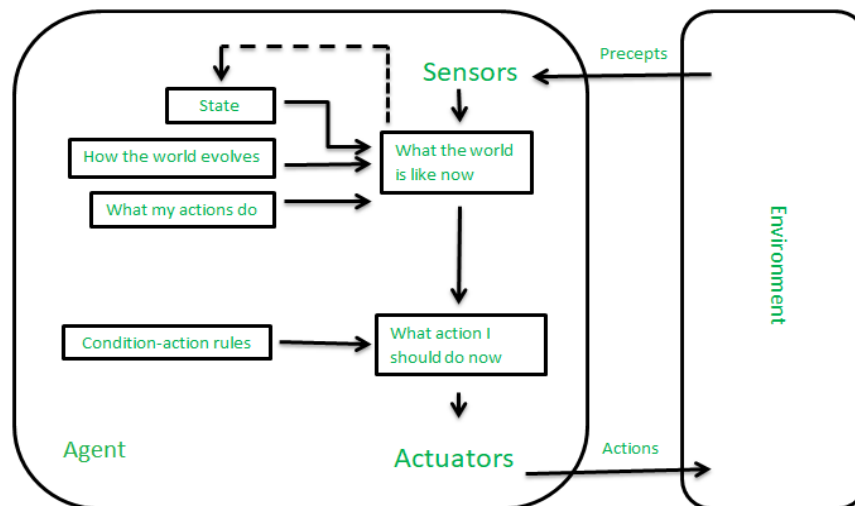
Model-based reflex agents

It works by finding a rule whose condition matches the current situation. A model-based agent can handle **partially observable environments** by the use of a model about the world. The agent has to keep track

of the **internal state** which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen.

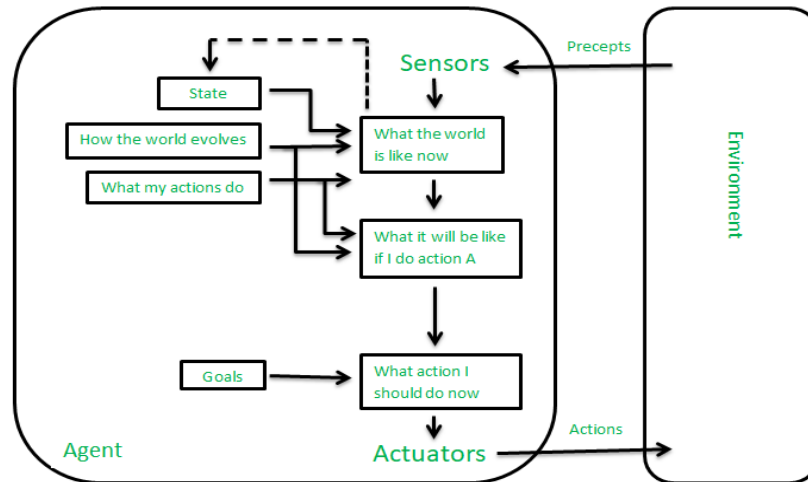
Updating the state requires information about :

- how the world evolves independently from the agent, and
- how the agent's actions affect the world.



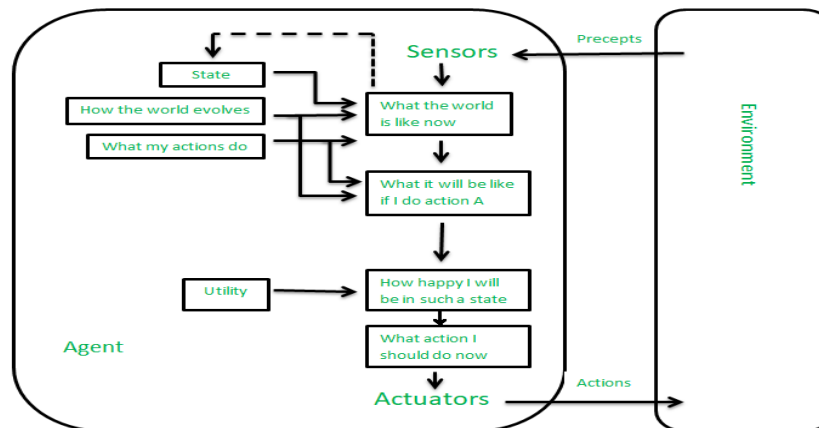
Goal-based agents

These kinds of agents take decisions based on how far they are currently from their **goal**(description of desirable situations). Their every action is intended to reduce its distance from the goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning. The goal-based agent's behavior can easily be changed.



Utility-based agents

The agents which are developed having their end uses as building blocks are called utility-based agents. When there are multiple possible alternatives, then to decide which one is best, utility-based agents are used. They choose actions based on a **preference (utility)** for each state. Sometimes achieving the desired goal is not enough. We may look for a quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how **“happy”** the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.



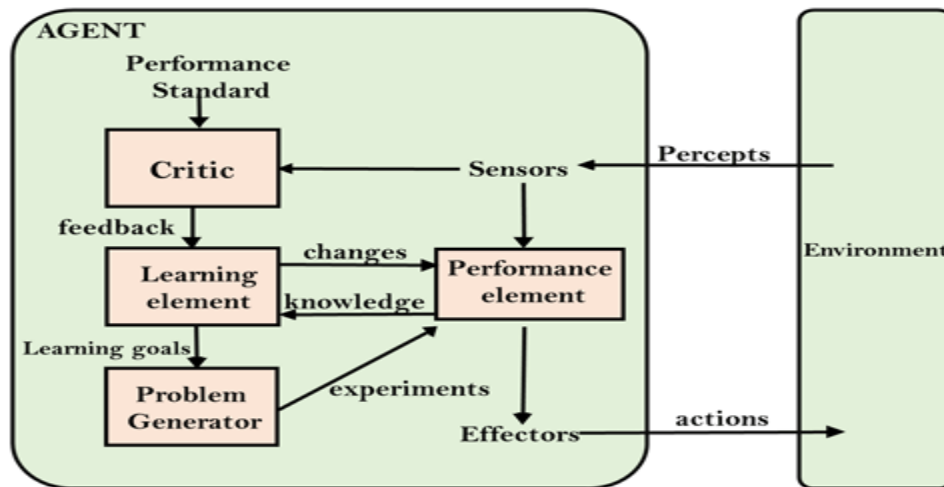
Learning Agent :

A learning agent in AI is the type of agent that can learn from its past experiences or it has learning capabilities. It starts to act with basic knowledge and then is able to act and adapt automatically through learning.

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

1. **Learning element:** It is responsible for making improvements by learning from the environment

2. **Critic:** The learning element takes feedback from critics which describes 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.



The Nature of Environments

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

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

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

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

Features of Environment

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

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

1. Fully observable vs Partially Observable:

1. 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**.
2. A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
3. An agent with no sensors in all environments then such an environment is called as **unobservable**.

2. Deterministic vs Stochastic:

1. 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.
2. A stochastic environment is random in nature and cannot be determined completely by an agent.
3. In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

3. Episodic vs Sequential:

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

4. Single-agent vs Multi-agent

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

5. Static vs Dynamic:

1. 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.
2. Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
3. However for dynamic environment, agents need to keep looking at the world at each action.
4. Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

6. Discrete vs Continuous:

1. 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.
2. A chess game comes under discrete environment as there is a finite number of moves that can be performed.
3. A self-driving car is an example of a continuous environment.

7. Known vs Unknown

1. Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
2. 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.
3. It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

8. Accessible vs Inaccessible

1. 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.
2. An empty room whose state can be defined by its temperature is an example of an accessible environment.
3. Information about an event on earth is an example of Inaccessible environment.

1.4 THE STATE OF THE ART

What can AI do today? A concise answer is difficult because there are so many activities in so many subfields. Here we sample a few applications; others appear throughout the book.

Robotic vehicles: A driverless robotic car named STANLEY sped through the rough terrain of the Mojave dessert at 22 mph, finishing the 132-mile course first to win the 2005 DARPA Grand Challenge. STANLEY is a Volkswagen Touareg outfitted with cameras, radar, and laser rangefinders to sense the environment and onboard software to command the steering, braking, and acceleration (Thrun, 2006). The following year CMU's BOSS won the Urban Challenge, safely driving in traffic through the streets of a closed Air Force base, obeying traffic rules and avoiding pedestrians and other vehicles.

Speech recognition: A traveler calling United Airlines to book a flight can have the entire conversation guided by an automated speech recognition and dialog management system.

Autonomous planning and scheduling: A hundred million miles from Earth, NASA's Remote Agent program became the first on-board autonomous planning program to control the scheduling of operations for a spacecraft (Jonsson *et al.*, 2000). REMOTE AGENT generated plans from high-level goals specified from the ground and monitored the execution of those plans—detecting, diagnosing, and recovering from problems as they occurred. Successor program MAPGEN (Al-Chang *et al.*, 2004) plans the daily operations for NASA's Mars Exploration Rovers, and MEXAR2 (Cesta *et al.*, 2007) did mission planning—both logistics and science planning—for the European Space Agency's Mars Express mission in 2008.

Game playing: IBM's DEEP BLUE became the first computer program to defeat the world champion in a chess match when it bested Garry Kasparov by a score of 3.5 to 2.5 in an exhibition match (Goodman and Keene, 1997). Kasparov said that he felt a "new kind of intelligence" across the board from him. *Newsweek* magazine described the match as "The brain's last stand." The value of IBM's stock increased by \$18 billion. Human champions studied Kasparov's loss and were able to draw a few matches in subsequent years, but the most recent human-computer matches have been won convincingly by the computer.

Spam fighting: Each day, learning algorithms classify over a billion messages as spam, saving the recipient from having to waste time deleting what, for many users, could comprise 80% or 90% of all messages, if not classified away by algorithms. Because the spammers are continually updating their tactics, it is difficult for a static programmed approach to keep up, and learning algorithms work best (Sahami *et al.*, 1998; Goodman and Heckerman, 2004).

Logistics planning: During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, and conflict resolution among all parameters. The AI planning techniques generated in hours a plan that would have taken weeks with older methods. The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

Robotics: The iRobot Corporation has sold over two million Roomba robotic vacuum cleaners for home use. The company also deploys the more rugged PackBot to Iraq and Afghanistan, where it is used to handle hazardous materials, clear explosives, and identify the location of snipers.

Machine Translation: A computer program automatically translates from Arabic to English, allowing an English speaker to see the headline "Ardogan Confirms That Turkey Would Not Accept Any Pressure, Urging Them to Recognize Cyprus." The program uses a statistical model built from examples of Arabic-to-English translations and from examples of English text totaling two trillion words (Brants *et al.*, 2007). None of the computer scientists on the team speak Arabic, but they do understand statistics and machine learning algorithms.

These are just a few examples of artificial intelligence systems that exist today. Not magic or science fiction—but rather science, engineering, and mathematics, to which this book provides an introduction.

UNIT 2

SOLVING PROBLEMS BY SEARCHING

Problem Solving Agents

Problem-solving agents:

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

Search algorithm

A search algorithm is **the step-by-step procedure used to locate specific data among a collection of data**. It is considered a fundamental procedure in computing. In computer science, when searching for data, the difference between a fast application and a slower one often lies in the use of the proper search algorithm

Search Algorithm Terminologies

1. **Search:** Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:
 - a. **Search Space:** Search space represents a set of possible solutions, which a system may have.
 - b. **Start State:** It is a state from where agent begins **the search**.
 - c. **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.
2. **Search tree:** A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.
3. **Actions:** It gives the description of all the available actions to the agent.
4. **Transition model:** A description of what each action do, can be represented as a transition model.
5. **Path Cost:** It is a function which assigns a numeric cost to each path.
6. **Solution:** It is an action sequence which leads from the start node to the goal node.
7. **Optimal Solution:** If a solution has the lowest cost among all solutions.

Properties of Search Algorithms:

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

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

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

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

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

Steps problem-solving in AI

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

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

- **Goal Formulation:** This one is the first and simple step in problem-solving. It organizes finite steps to formulate a target/goals which require some action to achieve the goal. Today the formulation of the goal is based on AI agents.
- **Problem formulation:** It is one of the core steps of problem-solving which decides what action should be taken to achieve the formulated goal. In AI this core part is dependent upon software agent which consisted of the following components to formulate the associated problem.

Components to formulate the associated problem:

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

EXAMPLE PROBLEMS

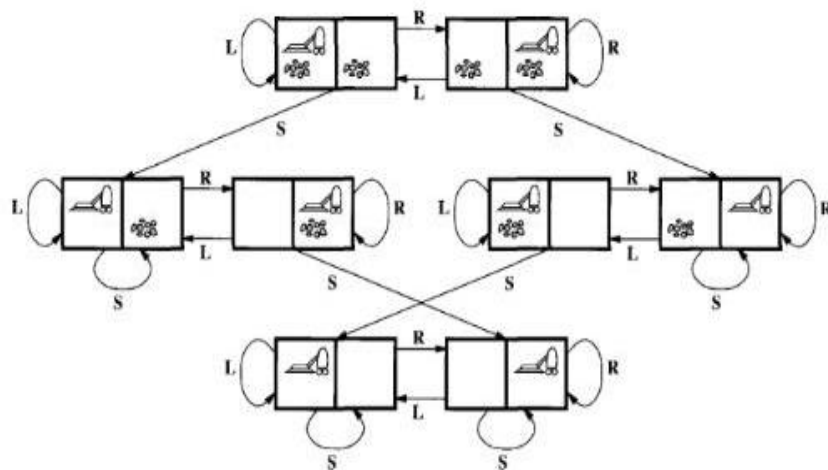
- a. The problem solving approach has been applied to a vast array of task environments.
 - i. Some best known problems are summarized below.
- b. They are distinguished as toy or real-world problems
 1. A **Toy problem** is intended to illustrate various problem solving ~~meth~~ It can be easily used by different researchers to compare the performance of algorithms.
 2. A **Real world problem** is one whose solutions people actually care about.

TOY PROBLEMS

Vacuum World Example

1. **States:** The agent is in one of two locations.,each of which might or might not contain dirt. Thus there are $2 \times 2^2 = 8$ possible world states.
2. **Initial state:** Any state can be designated as initial state.
3. **Successor function :** This generates the legal states that results from trying the three actions (left, right, suck). The complete state space is shown in figure 2.3
4. **Goal Test :** This tests whether all the squares are clean.
5. **Path test :** Each step costs one ,so that the the path cost is the number of steps in the path.

Vacuum World State Space

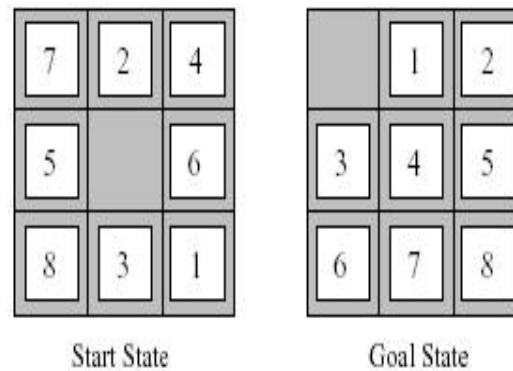


The state space for the vacuum world.
Arcs denote actions: L = Left,R = Right,S = Suck

8-puzzle:

1. An 8-puzzle consists of a 3x3 board with eight numbered tiles and a blank space.
2. A tile adjacent to the blank space can slide into the space. The object is to reach the specific goal state ,as shown in figure

Example: The 8-puzzle

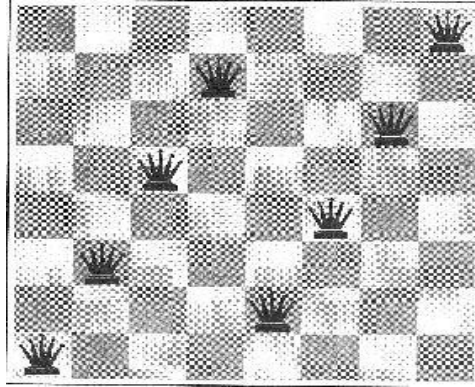


The problem formulation is as follows :

- a. **States** : A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- b. **Initial state** : Any state can be designated as the initial state. It can be noted that any given goal can be reached from exactly half of the possible initial states.
- c. **Successor function** : This generates the legal states that result from trying the four actions (blank moves Left, Right, Up or down).
- d. **Goal Test** : This checks whether the state matches the goal configuration shown in figure 2.4. (Other goal configurations are possible)
- e. **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.
- f. **The 8-puzzle** belongs to the family of **sliding-block puzzles**, which are often used as
 - i. test problems for new search algorithms in AI.
- g. This general class is known as NP-complete.
- h. The **8-puzzle** has $9!/2 = 181,440$ reachable states and is easily solved.
- i. The **15 puzzle** (4×4 board) has around 1.3 trillion states, and the random instances can be solved optimally in few milli seconds by the best search algorithms.
- j. The **24-puzzle** (on a 5×5 board) has around 10^{25} states, and random instances are still quite difficult to solve optimally with current machines and algorithms.

8-queens problem

1. The goal of 8-queens problem is to place 8 queens on the chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal).
2. The following figure shows an attempted solution that fails: the queen in the right most column is attacked by the queen at the top left.
3. An **Incremental formulation** involves operators that augments the state description, starting with an empty state. for 8-queens problem, this means each action adds a queen to the state.
4. A **complete-state formulation** starts with all 8 queens on the board and move them around.
5. In either case the path cost is of no interest because only the final state counts.



8-queens problem

The first incremental formulation one might try is the following :

- a. **States** : Any arrangement of 0 to 8 queens on board is a state.
- b. **Initial state** : No queen on the board.
- c. **Successor function** : Add a queen to any empty square.
- d. **Goal Test** : 8 queens are on the board, none attacked.

In this formulation, we have $64 \cdot 63 \cdot \dots \cdot 57 = 3 \times 10^{14}$ possible sequences to investigate.

A better formulation would prohibit placing a queen in any square that is already attacked. :

- a. **States** : Arrangements of n queens ($0 \leq n \leq 8$), one per column in the left most columns, with no queen attacking another are states.
- b. **Successor function** : Add a queen to any square in the left most empty column such that it is not attacked by any other queen.

This formulation reduces the 8-queen state space from 3×10^{14} to just 2057, and solutions are easy to find.

- a. For the 100 queens the initial formulation has roughly 10^{400} states whereas the improved formulation has about 10^{52} states.
- b. This is a huge reduction, but the improved state space is still too big for the algorithms to handle.

REAL WORLD PROBLEMS

1. A real world problem is one whose solutions people actually care about.
2. They tend not to have a single agreed upon description, but attempt is made to give general flavor of their formulation,

The following are the some real world problems,

- a. Route Finding Problem
- b. Touring Problems
- c. Travelling Salesman Problem
- d. Robot Navigation

ROUTE-FINDING PROBLEM

1. Route-finding problem is defined in terms of specified locations and transitions along links between them.
2. Route-finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning, and air line travel planning systems.

AIRLINE TRAVEL PROBLEM

The **airline travel problem** is specifies as follows :

1. **States** : Each is represented by a location(e.g.,an airport) and the current time.
2. **Initial state** : This is specified by the problem.
3. **Successor function** : This returns the states resulting from taking any scheduled flight(further specified by seat class and location),leaving later than the current time plus the within-airport transit time,from the current airport to another.
4. **Goal Test** : Are we at the destination by some prespecified time?
5. **Path cost** : This depends upon the monetary cost,waiting time,flight time,customs and immigration procedures,seat quality,time of dat,type of air plane,frequent-flyer mileage awards, and so on.

TOURING PROBLEMS

1. **Touring problems** are closely related to route-finding problems,but with an important difference.
2. Consider for example, the problem, "Visit every city at least once" as shown in Romania map.
3. As with route-finding the actions correspond to trips between adjacent cities. The state space, however,is quite different.
4. **Initial state** would be "In Bucharest; visited{ Bucharest}".
5. **Intermediate state** would be "In Vaslui; visited
a. { Bucharest,Vrziceni,Vaslui}".
6. **Goal test** would check whether the agent is in Bucharest and all cities have been visited.

THE TRAVELLING SALESPERSON PROBLEM (TSP)

1. TSP is a touring problem in which each city must be visited exactly once.
2. The aim is to find the shortest tour. The problem is known to be NP-hard.
3. Enormous efforts have been expended to improve the capabilities of TSP algorithms.
4. These algorithms are also used in tasks such as planning movements of **automatic circuit-board drills** and of **stocking machines** on shop floors.

VLSI layout

A **VLSI layout** problem requires positioning millions of components and connections on a chip to minimize area ,minimize circuit delays,minimize stray capacitances,and maximize manufacturing yield. The layout problem is split into two parts : **cell layout** and **channel routing**.

ROBOT NAVIGATION

ROBOT navigation is a generalization of the route-finding problem. Rather than a discrete set of routes, a robot can move in a continuous space with an infinite set of possible actions and states. For a circular Robot moving on a flat surface, the space is essentially two-dimensional.

When the robot has arms and legs or wheels that also must be controlled, the search space becomes multi-dimensional. Advanced techniques are required to make the search space finite.

AUTOMATIC ASSEMBLY SEQUENCING

The example includes assembly of intricate objects such as electric motors. The aim in assembly problems is to find the order in which to assemble the parts of some objects. If the wrong order is chosen, there will be no way to add some part later without undoing some work already done.

Another important assembly problem is protein design, in which the goal is to find a sequence of Amino acids that will be fold into a three-dimensional protein with the right properties to cure some disease.

INTERNET SEARCHING

In recent years there has been increased demand for software robots that perform Internet searching, looking for answers to questions, for related information, or for shopping deals.

The searching techniques consider internet as a graph of nodes (pages) connected by links.

MEASURING PROBLEM-SOLVING PERFORMANCE

The output of problem-solving algorithm is either failure or a solution. (Some algorithms might stuck in an infinite loop and never return an output.)

The algorithm's performance can be measured in four ways :

- a. **Completeness:** Is the algorithm guaranteed to find a solution when there is one?
- b. **Optimality :** Does the strategy find the optimal solution
- c. **Time complexity:** How long does it take to find a solution?
- d. **Space complexity:** How much memory is needed to perform the search

SEARCHING FOR SOLUTIONS

State Spaces

One general formulation of intelligent action is in terms of a state space. A state contains all of the information necessary to predict the effects of an action and to determine whether a state satisfies the goal.

Assumptions of State-space searching :

1. The agent has perfect knowledge of the state space and is planning for the case where it observes what state it is in: there is full observability.
2. The agent has a set of actions that have known deterministic effects.
3. The agent can determine whether a state satisfies the goal

A **state-space problem** consists of

1. A set of states
2. A distinguished state called the **start state**
3. For each state, a set of actions available to the agent in that state
4. an **action function** that, given a state and an action, returns a new state
5. A **goal** specified as a Boolean function, $goal(s)$, that is true when state ss satisfies the goal, in which case we say that ss is a **goal state**
6. A criterion that specifies the quality of an acceptable solution. For example, any sequence of actions that gets the agent to the goal state may be acceptable, or there may be costs associated with actions and the agent may be required to find a sequence that has minimal total cost. A solution that is best according to some criterion is called an **optimal solution**. We do not always need an optimal solution, for example, we may be satisfied with any solution that is within 10% of optimal.

Graph Searching

1. The problem of finding a sequence of actions to achieve a goal is abstracted as searching for paths in directed graphs.
2. To solve a problem, first define the underlying search space and then apply a search algorithm to that search space.
3. Many problem-solving tasks are transformable into the problem of finding a path in a graph.
4. Searching in graphs provides an appropriate abstract model of problem solving independent of a particular domain.

A Generic Searching Algorithm

A generic algorithm to search for a solution path in a graph. The algorithm calls procedures that can be coded to implement various search strategies.

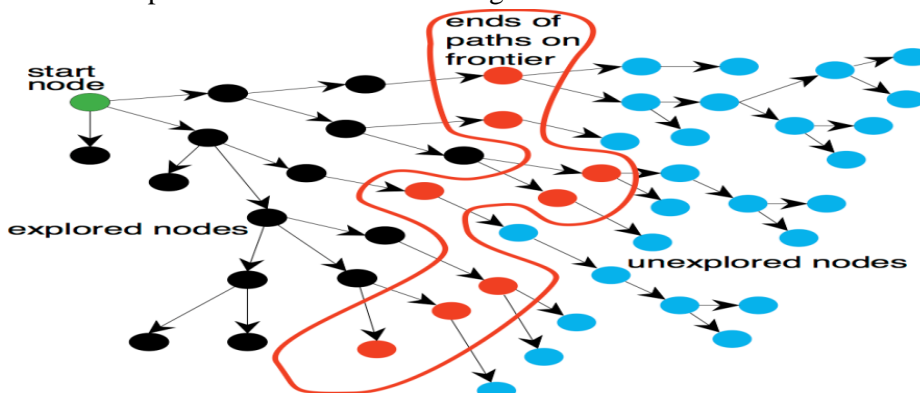


FIG: Problem solving by graph searching

The intuitive idea behind the generic search algorithm, given a graph, a start node, and a goal predicate, is to explore paths incrementally from the start node. This is done by maintaining a **frontier** (or **fringe**) of paths from the start node. The frontier contains all of the paths that could form initial segments of paths from the start node to a goal node. Initially, the frontier contains the trivial path containing just the start node, and no arcs. As the search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered. Different search strategies are obtained by providing an appropriate implementation of the frontier.

Generic graph searching algorithm

```

1. procedure Search(G,S,goalG,S,goal)
2:   Inputs
3:     GG: graph with nodes NN and arcs AA
4:     ss: start node
5:     goal: Boolean function of nodes
6:   Output
7:     path from ss to a node for which goal is true
8:     or  $\perp$  if there are no solution paths
9:   Local
10:    Frontier: set of paths
11:    Frontier:= $\{\langle s \rangle\}$  Frontier:= $\{\langle s \rangle\}$ 
12:    while Frontier $\neq \{\}$  Frontier $\neq \{\}$  do
13:      select and remove  $\langle n_0, \dots, n_k \rangle$  from Frontier
14:      if goal( $n_k$ )goal( $n_k$ ) then
15:        return  $\langle n_0, \dots, n_k \rangle$ 
16:      Frontier:=Frontier $\cup \{\langle n_0, \dots, n_k, n \rangle : \langle n_k, n \rangle \in A\}$  Frontier:=Frontier $\cup \{\langle n_0, \dots, n_k, n \rangle : \langle n_k, n \rangle \in A\}$ 
17:    return  $\perp$ 

```

The frontier is a set of paths. Initially, the frontier contains the path of zero cost consisting of just the start node. At each step, the algorithm removes a path $\langle n_0, \dots, n_k \rangle$ from the frontier.

If goal(n_k) is true (i.e., n_k is a goal node), it has **found a solution** and returns the path $\langle n_0, \dots, n_k \rangle$. Otherwise, the path is extended by one more arc by finding the neighbors of n_k . For every neighbor n of n_k , the path $\langle n_0, \dots, n_k, n \rangle$ is added to the frontier. This step is known as **expanding** the path $\langle n_0, \dots, n_k \rangle$.

This algorithm has a few features that should be noted:

1. Which path is selected at line 13 defines the search strategy. The selection of a path can affect the efficiency; see the box for more details on the use of “select”.
2. It is useful to think of the *return* at line 15 as a temporary return, where a caller can **retry** the search to get another answer by continuing the while loop. This can be implemented by having a class that keeps the state of the search and a search() method that returns the next solution.

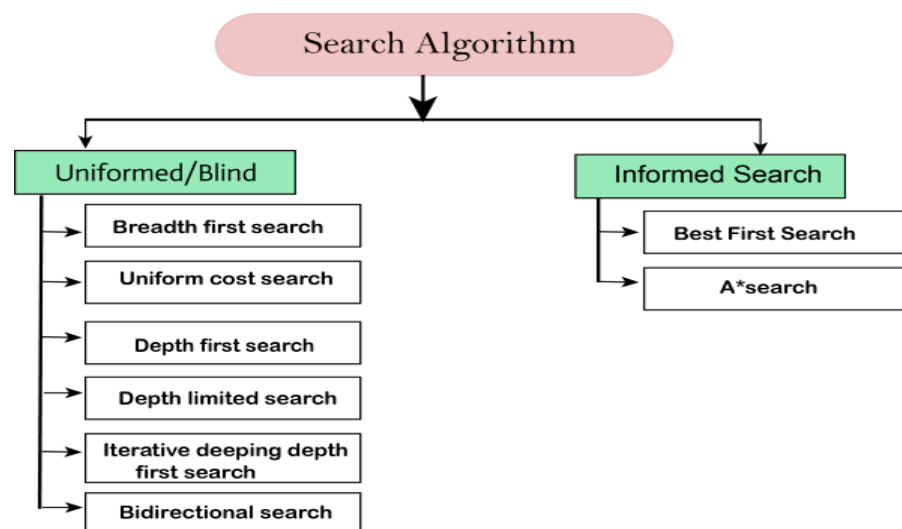
3.If the procedure returns \perp (“**bottom**”), there are no solutions, or no remaining solutions if the search has been retried.

4.The algorithm only tests if a path ends in a goal node *after* the path has been selected from the frontier, not when it is added to the frontier. There are two important reasons for this. There could be a costly arc from a node on the frontier to a goal node. The search should not always return the path with this arc, because a lower-cost solution may exist. This is crucial when the lowest-cost path is required. A second reason is that it may be expensive to determine whether a node is a goal node, and so this should be delayed in case the computation is not necessary.

If the node at the end of the selected path is not a goal node and it has no neighbors, then extending the path means removing the path from the frontier. This outcome is reasonable because this path could not be part of a path from the start node to a goal node.

Types of search algorithms

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



Uninformed/Blind Search:

The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes. Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.

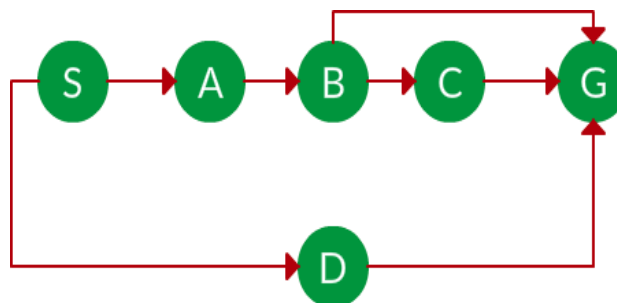
It can be divided into five main types:

- Breadth-first search
- Uniform cost search
- Depth-first search
- Iterative deepening depth-first search
- Bidirectional Search

Breadth-first search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

BFS expands the leaf node with the lowest path cost so far, and keeps going until a goal node is generated. If the path cost simply equals the number of links, we can implement this as a simple queue ("first in, first out").



Depth-first Search

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

Advantage:

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

Disadvantage:

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

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

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

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

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

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

Depth-Limited Search Algorithm:

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

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

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

Advantages:

Depth-limited search is Memory efficient.

Disadvantages:

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

Bidirectional search

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

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

Advantages:

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

Disadvantages:

Implementation of the bidirectional search tree is difficult.

In bidirectional search, one should know the goal state in advance.

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

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

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

Optimal: Bidirectional search is Optimal.

Uniform-cost Search Algorithm:

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

Advantages:

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

Disadvantages:

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

Best first search

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

Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.
- **Step 4:** Expand the node n , and generate the successors of node n .
- **Step 5:** Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

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

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

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

Optimal: Greedy best first search algorithm is not optimal.

Iterative deepening algorithm

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

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

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

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

Completeness: This algorithm is complete if the branching factor is finite.

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

Space Complexity: The space complexity of IDDFS will be $O(bd)$.

Informed Search

Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function:

Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it is guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value $h(n)$. It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues until a goal state is found.

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

- **Best First Search Algorithm(Greedy search)**
- **A* Search Algorithm**

Best First Search Algorithm(Greedy search)

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first

search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

$$f(n) = g(n).$$

Where, $h(n)$ = estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and places it in the CLOSED list.
- **Step 4:** Expand the node n , and generate the successors of node n .
- **Step 5:** Check each successor of node n , and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function $f(n)$, and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

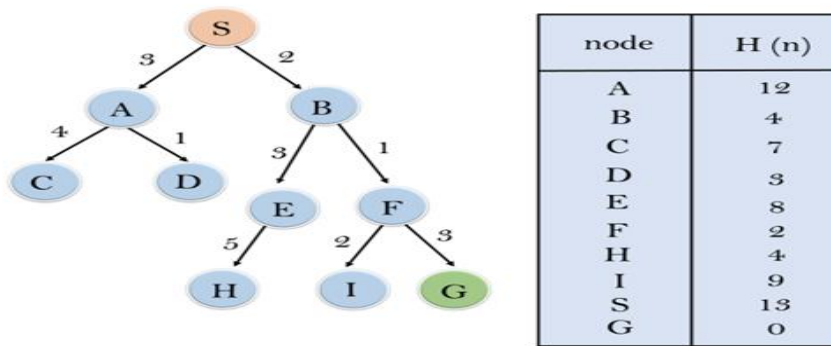
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

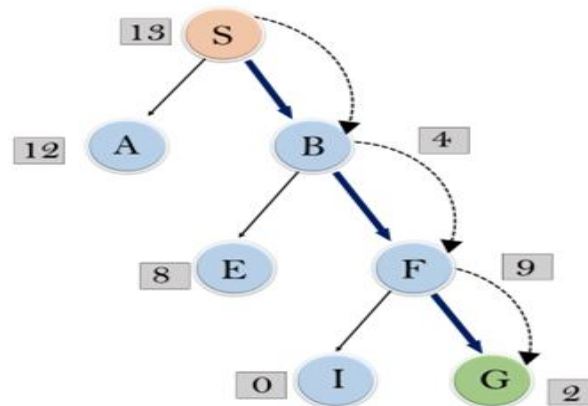
- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

Example:

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



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



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

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

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

Iteration3: Open[I,G,E,A],Closed[S,B,F]
: Open [I, E, A], Closed [S, B, F, G]

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

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

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

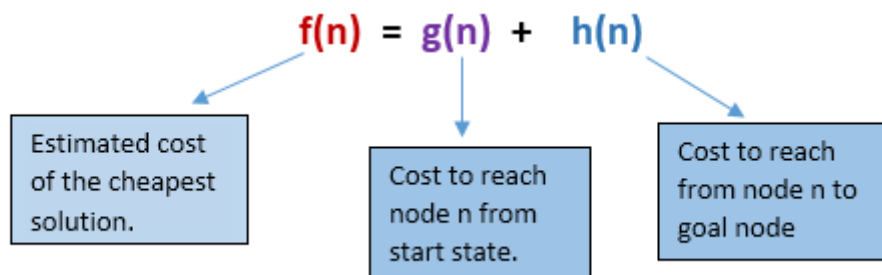
Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

2.) A* Search Algorithm:

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

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



Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function ($g+h$), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n' , check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.

Step 6: Return to **Step 2**.

Advantages:

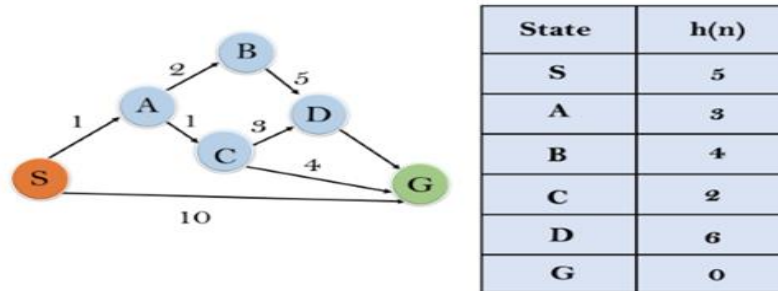
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

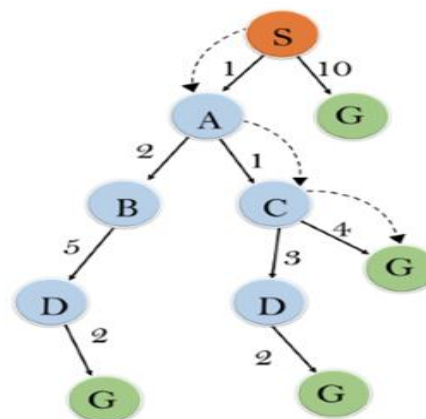
- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:

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



Solution:



Initialization: $\{(S, 5)\}$

Iteration1: $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

Iteration2: $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration3: $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration 4 will give the final result, as $S \rightarrow A \rightarrow C \rightarrow G$ it provides the optimal path with cost 6.

Points to remember:

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

Complete: A* algorithm is complete as long as:

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

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

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

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

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

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

Recursive Best-first Search (RBFS)

1. Recursive best-first search is a simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only linear space.
2. The algorithm is shown in below figure.
3. Its structure is similar to that of recursive depth-first search, but rather than continuing indefinitely down the current path, it keeps track of the f -value of the best alternative path available from any ancestor of the current node.
4. If the current node exceeds this limit, the recursion unwinds back to the alternative path. As the recursion unwinds, RBFS replaces the f -value of each node along the path with the best f -value of its children.

function RECURSIVE-BEST-FIRST-SEARCH(problem) **return** a solution or failure
return RFBS(problem, MAKE-NODE(INITIAL-STATE[problem]), ∞)

function RFBS(problem, node, f_limit) **return** a solution or failure and a new f -cost limit

if GOAL-TEST[problem](STATE[node]) **then return** node successors

□ EXPAND(node, problem)

if successors is empty **then return** failure, ∞

for each s **in** successors **do**

$f[s]$ □ $\max(g(s) + h(s), f[node])$

repeat

best □ the lowest f -value node in successors

if $f[best] > f_limit$ **then return** failure, $f[best]$

alternative □ the second lowest f -value among successors result, $f[best]$

□ RBFS(problem, best, $\min(f_limit, alternative)$) **if** result □ failure **then return** result

The algorithm for recursive best-first search

RBFS Evaluation:

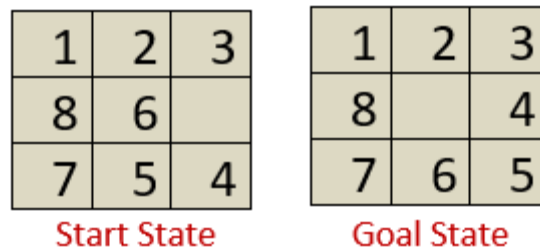
- a. RBFS is a bit more efficient than IDA*
 - i. Still excessive node generation (mind changes)
- b. Like A*, optimal if $h(n)$ is admissible
- c. Space complexity is $O(bd)$.
 - i. IDA* retains only one single number (the current f-cost limit)
- d. Time complexity difficult to characterize
 - i. Depends on accuracy of $h(n)$ and how often best path changes.
- e. IDA* and RBFS suffer from *too little* memory.

Heuristic Functions

A **heuristic function** or simply a heuristic is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search

Example: 8 Puzzle

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

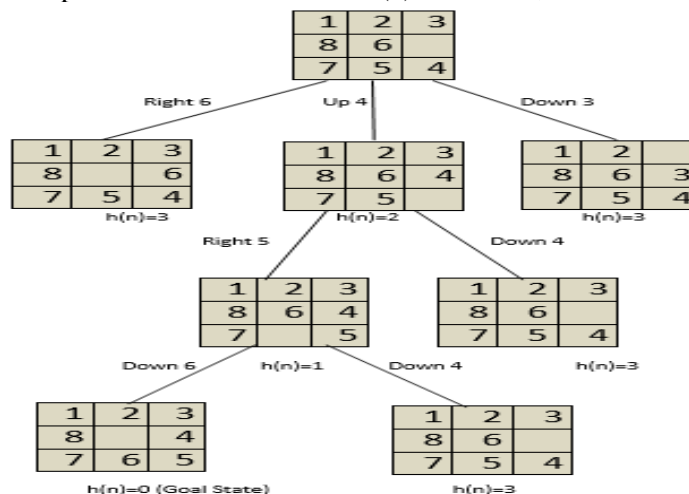


A heuristic function for the 8-puzzle problem is defined below:

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

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

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



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

Properties of a Heuristic search Algorithm

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

- **Admissible Condition:** An algorithm is said to be admissible, if it returns an optimal solution.
- **Completeness:** An algorithm is said to be complete, if it terminates with a solution (if the solution exists).
- **Dominance Property:** If there are two admissible heuristic algorithms **A1** and **A2** having **h1** and **h2** heuristic functions, then **A1** is said to dominate **A2** if **h1** is better than **h2** for all the values of node **n**.
- **Optimality Property:** If an algorithm is **complete, admissible, and dominating** other algorithms, it will be the best one and will definitely give an optimal solution.

The Effective Branching factor

- a. One way to characterize the **quality of a heuristic** is the **effective branching factor b^*** .

If the total number of nodes generated by A^* for a particular problem is N , and the **solution depth** is d , then b^* is the branching factor that a uniform tree of depth d would have to have in order to contain $N+1$ nodes. Thus,

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

- b. For example, if A^* finds a solution at depth 5 using 52 nodes, then effective branching factor is 1.92.
- c. A well designed heuristic would have a value of b^* close to 1, allowing fairly large problems to be solved.
- d. To test the heuristic functions h_1 and h_2 , 1200 random problems were generated with solution lengths from 2 to 24 and solved them with iterative deepening search and with A^* search using both h_1 and h_2 .
- e. The following table gives the average number of nodes expanded by each strategy and
 - i. the effective branching factor.
- f. The results suggest that h_2 is better than h_1 , and is far better than using iterative deepening search.
- g. For a solution length of 14, A^* with h_2 is 30,000 times more efficient than uninformed iterative deepening search.

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	92	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Comparison of search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A^* Algorithms with h_1 and h_2 . Data are average over 100 instances of the 8-puzzle, for various solution lengths.

Hill Climbing Algorithm in Artificial Intelligence

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

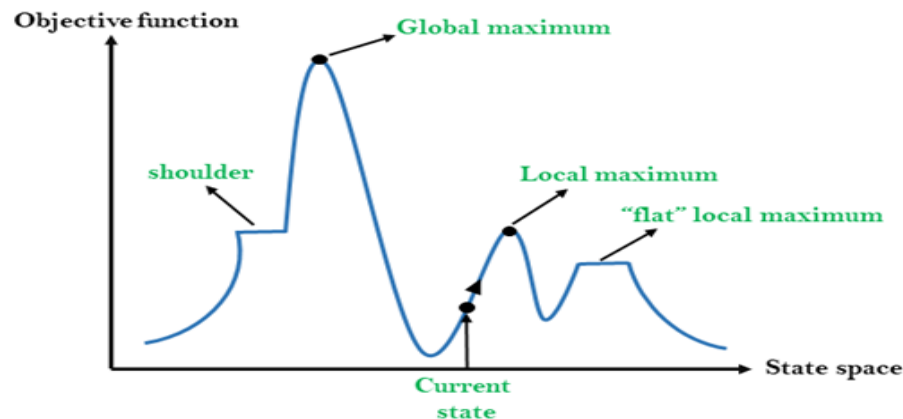
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state.** It only checks its one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming

- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
 1. If it is goal state, then return success and quit.
 2. Else if it is better than the current state then assign new state as a current state.
 3. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
 1. Let SUCC be a state such that any successor of the current state will be better than it.
 2. For each operator that applies to the current state:
 - I. Apply the new operator and generate a new state.
 - II. Evaluate the new state.
 - III. If it is goal state, then return it and quit, else compare it to the SUCC.
 - IV. If it is better than SUCC, then set new state as SUCC.
 - V. If the SUCC is better than the current state, then set current state to SUCC.
- **Step 5:** Exit.

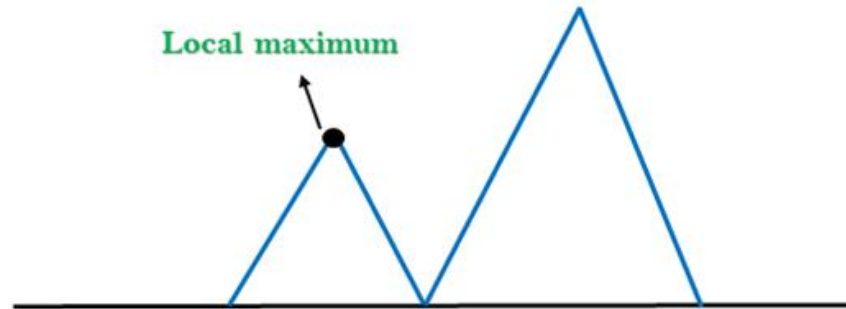
3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm:

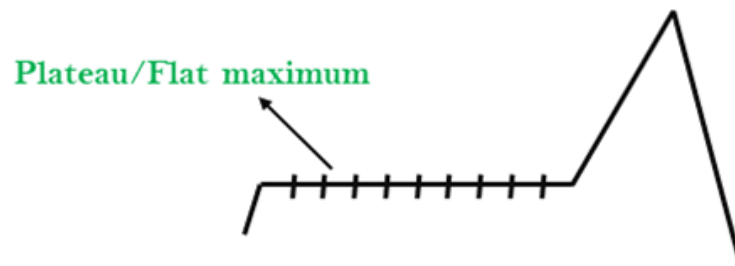
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.



Simulated Annealing:

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient. **Simulated Annealing** is an algorithm which yields both efficiency and completeness.

In mechanical term **Annealing** is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

Inventing admissible heuristic functions

Relaxed problems

- A problem with fewer restrictions on the actions is called a **relaxed problem**
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to *any adjacent square*, then $h_2(n)$ gives the shortest solution

Local Search Algorithms and Optimization Problem

The informed and uninformed search expands the nodes systematically in two ways:

- keeping different paths in the memory and
- selecting the best suitable path,

Which leads to a solution state required to reach the goal node. But beyond these “classical search algorithms,” we have some “local search algorithms” where the path cost does not matters, and only focus on solution-state needed to reach the goal node.

A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbors of that node generally.

Although local search algorithms are not systematic, still they have the following two advantages:

- Local search algorithms use a very little or constant amount of memory as they operate only on a single path.
- Most often, they find a reasonable solution in large or infinite state spaces where the classical or systematic algorithms do not work.

PURE OPTIMIZED PROBLEM

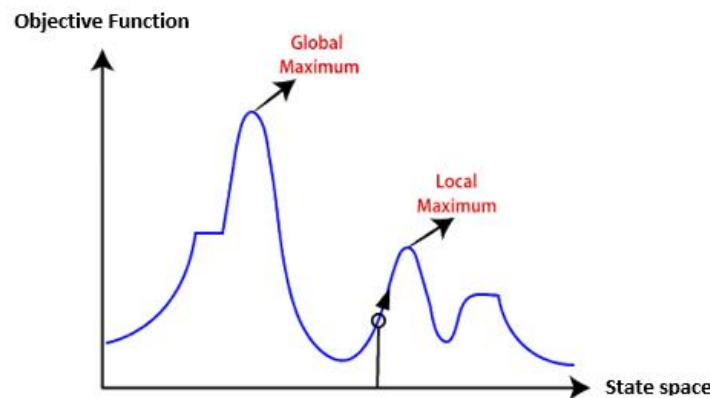
A pure optimization problem is one where all the nodes can give a solution. But the target is to find the best state out of all according to the **objective function**. Unfortunately, the pure optimization problem fails to find high-quality solutions to reach the goal state from the current state.

An objective function is a function whose value is either minimized or maximized in different contexts of the optimization problems. In the case of search algorithms, an objective function can be the path cost for reaching the goal node, etc.

Working of a Local search algorithm

Consider the below state-space landscape having both:

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



A one-dimensional state-space landscape in which elevation corresponds to the objective function

The local search algorithm explores the above landscape by finding the following two points:

- **Global Minimum:** If the elevation corresponds to the cost, then the task is to find the lowest valley, which is known as **Global Minimum**.
- **Global Maxima:** If the elevation corresponds to an objective function, then it finds the highest peak which is called as **Global Maxima**. It is the highest point in the valley.

We will understand the working of these points better in Hill-climbing search.

Below are some different types of local searches:

- Hill-climbing Search
- Simulated Annealing
- Local Beam Search

Online search agents

Online search is a necessary idea for unknown environments, where the agent does not know what states exist or what its actions do. In this state of ignorance, the agent faces an exploration problem and must use its actions as experiments in order to learn enough to make deliberation

The canonical example of online search is a robot that is placed in a new building and must explore it to build a map that it can use for getting from A to B. Methods for escaping from labyrinths—required knowledge for aspiring heroes of antiquity—are also examples of online search algorithms

Spatial exploration is not the only form of exploration, however. Consider a newborn baby: it has many possible actions but knows the outcomes of none of them, and it has experienced only a few of the possible states that it can reach. The baby's gradual discovery of how the world works is, in part, an online search process.

Online search problems

An online search problem must be solved by an agent executing actions, rather than by pure computation. We assume a deterministic and fully observable environment (Chapter 17 relaxes these assumptions), but we stipulate that the agent knows only the following:

- $ACTIONS(s)$, which returns a list of actions allowed in state s ;
- The step-cost function $c(s, a, s')$ —note that this cannot be used until the agent knows that s' is the outcome; and
- $GOAL-TEST(s)$.

Note in particular that the agent cannot determine $RESULT(s, a)$ except by actually being in s and doing a . For example, in the maze problem shown in Figure 4.19, the agent does not know that going Up from (1,1) leads to (1,2); nor, having done that, does it know that going Down will take it back to (1,1). This degree of ignorance can be reduced in some applications—for example, a robot explorer might know how its movement actions work and be ignorant only of the locations of obstacles

UNIT 3

REINFORCEMENT LEARNING

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.

Since there is no labeled data, so the agent is bound to learn by its experience only.

RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.

The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.

The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.

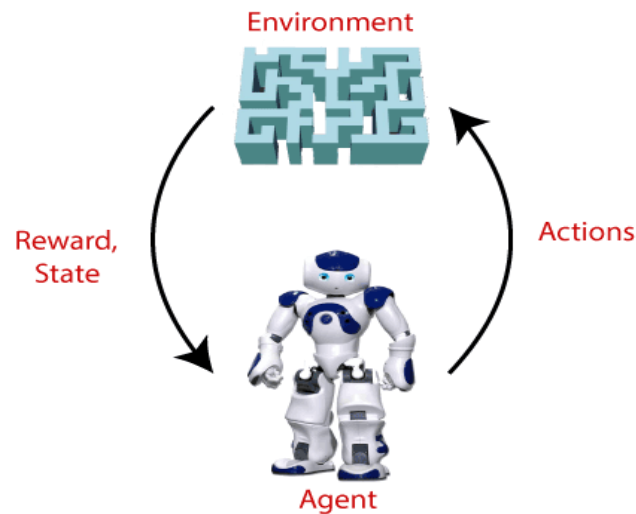
It is a core part of Artificial intelligence

- a. , and all AI agent
- b. works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.

Example: Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.

The agent continues doing these three things (take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment.

The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



Terms used in Reinforcement Learning

- Agent(): An entity that can perceive/explore the environment and act upon it.
- Environment(): A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- Action(): Actions are the moves taken by an agent within the environment.
- State(): State is a situation returned by the environment after each action taken by the agent.
- Reward(): A feedback returned to the agent from the environment to evaluate the action of the agent.
- Policy(): Policy is a strategy applied by the agent for the next action based on the current state.
- Value(): It is expected long-term return with the discount factor and opposite to the short-term reward.
- Q-value(): It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning

There are mainly three ways to implement reinforcement-learning in ML, which are:

1. Value-based:

The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π .

2. Policy-based:

Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each

step helps to maximize the future reward.

The policy-based approach has mainly two types of policy:

- Deterministic: The same action is produced by the policy (π) at any state.
- Stochastic: In this policy, probability determines the produced action.

3. Model-based:

In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function
4. Model of the environment

1) Policy:

A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

2) Reward Signal:

The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a reward signal. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the immediate signal for each good and bad action, whereas a value function specifies the good state and action for the future. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

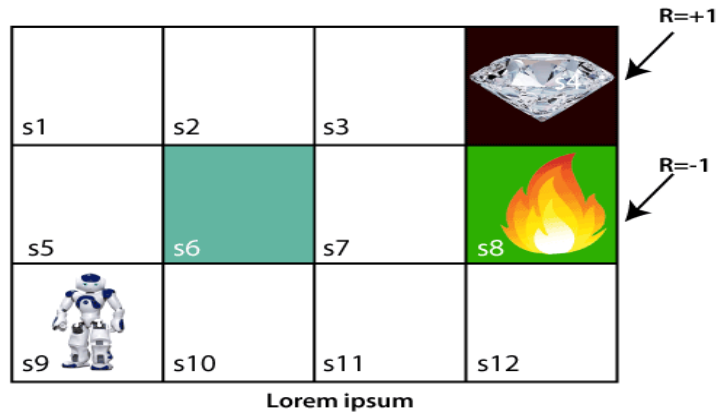
4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems with the help of the model are termed as the model-based approach. Comparatively, an approach without using a model is called a model-free approach.

WORKING OF REINFORCEMENT LEARNING:

- **Environment:** It can be anything such as a room, maze, football ground, etc.
- **Agent:** An intelligent agent such as AI robot.

Let's take an example of a maze environment that the agent needs to explore. Consider the below image:

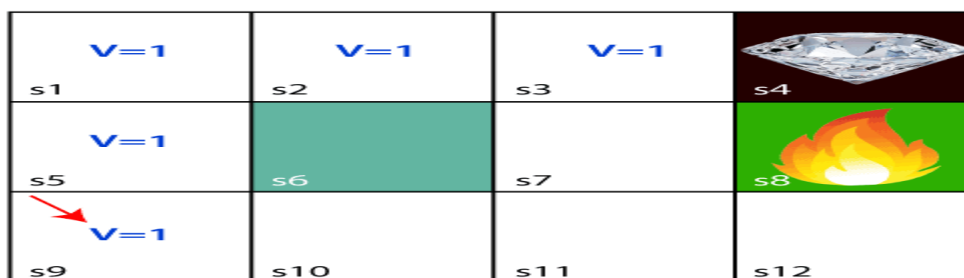


In the above image, the agent is at the very first block of the maze. The maze is consisting of an S_6 block, which is a **wall**, S_8 a **fire pit**, and S_4 a **diamond block**.







The agent cannot cross the S_6 block, as it is a solid wall. If the agent reaches the S_4 block, then get the **+1 reward**; if it reaches the fire pit, then gets **-1 reward point**. It can take four actions: **move up, move down, move left, and move right**.

The agent can take any path to reach to the final point, but he needs to make it in possible fewer steps. Suppose the agent considers the path **S9-S5-S1-S2-S3**, so he will get the +1-reward point.

The agent will try to remember the preceding steps that it has taken to reach the final step. To memorize the steps, it assigns 1 value to each previous step. Consider the below step:



Now, the agent has successfully stored the previous steps assigning the 1 value to each previous block. But what will the agent do if he starts moving from the block, which has 1 value block on both sides? Consider the below diagram:

s1 	s2  $V=1$	s3 $V=1$	s4 
s5 $V=1$ 	s6 	s7	s8 
s9 $V=1$	s10	s11	s12

It will be a difficult condition for the agent whether he should go up or down as each block has the same value. So, the above approach is not suitable for the agent to reach the destination. Hence to solve the problem, we will use the **Bellman equation**, which is the main concept behind reinforcement learning.

The Bellman Equation

The Bellman equation was introduced by the Mathematician **Richard Ernest Bellman in the year 1953**, and hence it is called as a Bellman equation. It is associated with dynamic programming and used to calculate the values of a decision problem at a certain point by including the values of previous states.

It is a way of calculating the value functions in dynamic programming or environment that leads to modern reinforcement learning.

The key-elements used in Bellman equations are:

- Action performed by the agent is referred to as "a"
- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma " γ ."

The Bellman equation can be written as:

$$V(s) = \max [R(s,a) + \gamma V(s')]$$

Where,

$V(s)$ = value calculated at a particular point.

$R(s,a)$ = Reward at a particular state s by performing an action.

γ = Discount factor

$V(s')$ = The value at the previous state.

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

For 1st block:

$V(s_3) = \max [R(s,a) + \gamma V(s')]$, here $V(s') = 0$ because there is no further state to move.

$$V(s_3) = \max [R(s,a)] \Rightarrow V(s_3) = \max [1] \Rightarrow \mathbf{V(s_3) = 1.}$$

For 2nd block:

$V(s_2) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 1$, and $R(s, a) = 0$, because there is no reward at this state.

$$V(s_2) = \max [0.9(1)] \Rightarrow V(s_2) = \max [0.9] \Rightarrow \mathbf{V(s_2) = 0.9}$$

For 3rd block:

$V(s_1) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 0.9$, and $R(s, a) = 0$, because there is no reward at this state also.

$$V(s_1) = \max [0.9(0.9)] \Rightarrow V(s_1) = \max [0.81] \Rightarrow \mathbf{V(s_1) = 0.81}$$

For 4th block:

$V(s_5) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 0.81$, and $R(s, a) = 0$, because there is no reward at this state also.





$$V(s_5) = \max [0.9(0.81)] \Rightarrow V(s_5) = \max [0.73] \Rightarrow \mathbf{V(s_5) = 0.73}$$

For 5th block:







$V(s_9) = \max [R(s,a) + \gamma V(s')]$, here $\gamma = 0.9$ (lets), $V(s') = 0.73$, and $R(s, a) = 0$, because there is no reward at this state also.

$$V(s_9) = \max [0.9(0.73)] \Rightarrow V(s_9) = \max [0.66] \Rightarrow \mathbf{V(s_9) = 0.66}$$




Consider the below image:

V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5	 s6	s7	 s8
 V=0.66 s9	s10	s11	s12

Now, we will move further to the 6th block, and here agent may change the route because it always tries to find the optimal path. So now, let's consider from the block next to the fire pit.

V=0.81 s1	V=0.9 s2	V=1 s3	
V=0.73 s5		   s7	 s8
V=0.66 s9	s10	s11	s12

Now, the agent has three options to move; if he moves to the blue box, then he will feel a bump if he moves to the fire pit, then he will get the -1 reward. But here we are taking only positive rewards, so for this, he will move to upwards only. The complete block values will be calculated using this formula. Consider the below image:

V=0.81 s1	V=0.9 s2	V=1 s3	
V=0.73 s5		V=0.9 s7	 s8
V=0.66 s9	V=0.73 s10	V=0.81 s11	V=0.73 s12

Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- **Positive Reinforcement**
- **Negative Reinforcement**

Positive Reinforcement:

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

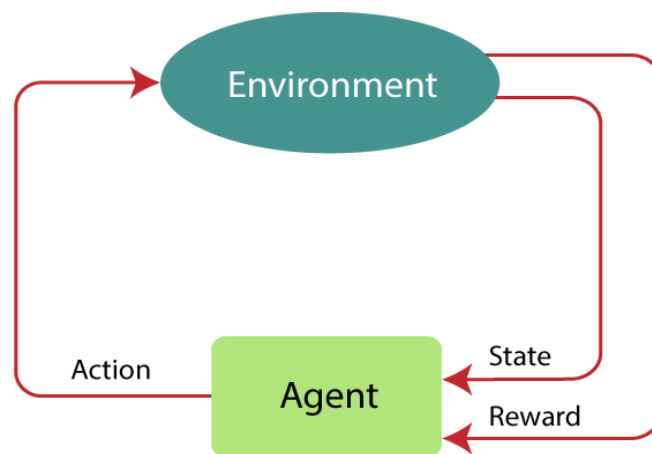
Negative Reinforcement:

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

Markov Decision Process

Markov Decision Process or MDP, is used to **formalize the reinforcement learning problems**. If the environment is completely observable, then its dynamic can be modeled as a **Markov Process**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.



MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

MDP contains a tuple of four elements (S , A , P_a , R_a):

- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S' , due to action a .
- Probability P_a .

MDP uses Markov property, and to better understand the MDP, we need to learn about it.

Markov Property:

It says that "If the agent is present in the current state S_1 , performs an action a_1 and move to the state s_2 , then the state transition from s_1 to s_2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."

Or, in other words, as per Markov Property, the current state transition does not depend on any past action or state. Hence, MDP is an RL problem that satisfies the Markov property. Such as in a Chess game, the players only focus on the current state and do not need to remember past actions or states.

Finite MDP:

A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

Markov Process:

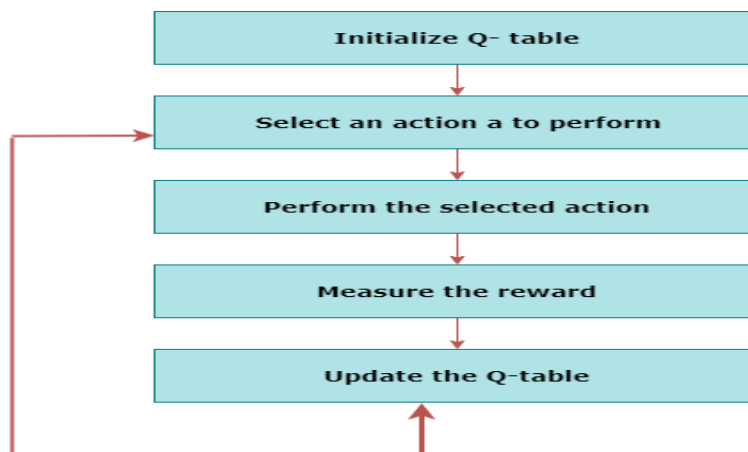
Markov Process is a memoryless process with a sequence of random states S_1, S_2, \dots, S_t that uses the Markov Property. Markov process is also known as Markov chain, which is a tuple (S, P) on state S and transition function P . These two components (S and P) can define the dynamics of the system.

REINFORCEMENT LEARNING ALGORITHMS

Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

- **Q-Learning:**

- Q-learning is an **Off policy RL algorithm**, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.
- It learns the value function $Q(S, a)$, which means how good to take action "**a**" at a particular state "**s**."
- The below flowchart explains the working of Q- learning:



- **State Action Reward State action (SARSA):**

- SARSA stands for **State Action Reward State action**, which is an **on-policy** temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.
- The goal of SARSA is to calculate the $Q \pi(s, a)$ for the selected current policy π and all pairs of $(s-a)$.
- The main difference between Q-learning and SARSA algorithms is that **unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table.**
- In SARSA, new action and reward are selected using the same policy, which has determined the original action.

- The SARSA is named because it uses the quintuple **Q(s, a, r, s', a')**. Where,
s:originastate
a:Originalaction
r:rewardobservedwhilefollowingthestates
s' and a': New state, action pair.
- **Deep Q Neural Network (DQN):**
 - As the name suggests, DQN is a **Q-learning using Neural networks**.
 - For a big state space environment, it will be a challenging and complex task to define and update a Q-table.
 - To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.

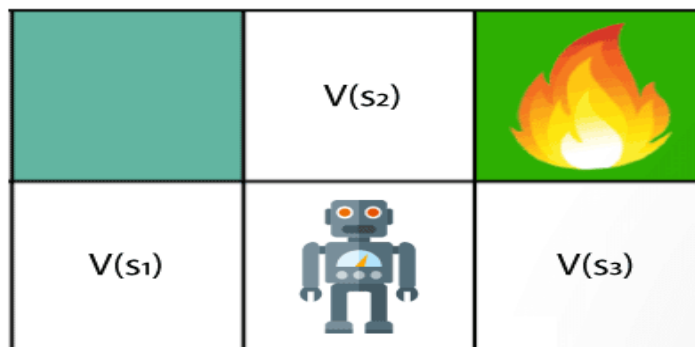
Now, we will expand the Q-learning.

Q-Learning Explanation:

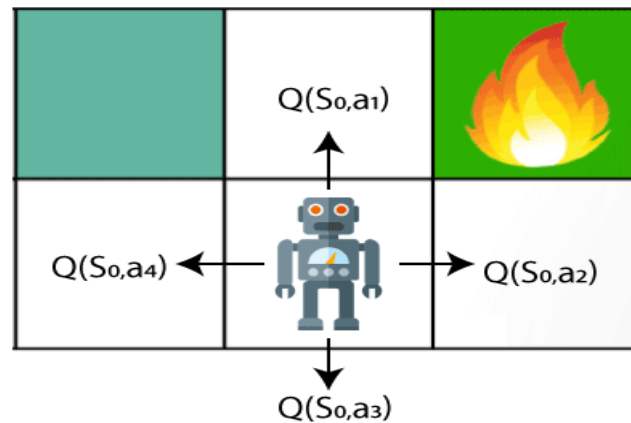
- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- **The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.**
- It is an **off-policy RL** that attempts to find the best action to take at a current state.
- The goal of the agent in Q-learning is to maximize the value of Q.
- The value of Q-learning can be derived from the Bellman equation. Consider the Bellman equation given below:

$$V(s) = \max [R(s,a) + \gamma \sum_{s'} P(s, a, s') V(s')]]$$

In the equation, we have various components, including reward, discount factor (γ), probability, and end states s' . But there is no any Q-value is given so first consider the below image:



In the above image, we can see there is an agent who has three values options, $V(s_1)$, $V(s_2)$, $V(s_3)$. As this is MDP, so agent only cares for the current state and the future state. The agent can go to any direction (Up, Left, or Right), so he needs to decide where to go for the optimal path. Here agent will take a move as per probability bases and changes the state. But if we want some exact moves, so for this, we need to make some changes in terms of Q-value. Consider the below image:



Q- represents the quality of the actions at each state. So instead of using a value at each state, we will use a pair of state and action, i.e., $Q(s, a)$. Q-value specifies that which action is more lubricative than others, and according to the best Q-value, the agent takes his next move. The Bellman equation can be used for deriving the Q-value.

To perform any action, the agent will get a reward $R(s, a)$, and also he will end up on a certain state, so the Q - value equation will be:

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s')$$

Hence, we can say that, $V(s) = \max [Q(s, a)]$

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} (P(s, a, s') \max_{a'} Q(s', a'))$$

The above formula is used to estimate the Q-values in Q-Learning.

What is 'Q' in Q-learning?

The Q stands for **quality** in **Q-learning**, which means it specifies the quality of an action taken by the agent.

Q-table:

A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e., $[s, a]$, and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table.

The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

Passive Reinforcement Learning

1. In this learning, the agent's policy is fixed and the task is to learn the utilities of states.
2. It could also involve learning a model of the environment.
3. In passive learning, the agent's policy π is fixed (i.e.) in state s , it always executes the action
 - i. $\pi(s)$.

Its goal is simply to learn the utility function $U \pi(s)$.

4. For example: - Consider the 4 x 3 world.
5. The following figure shows the policy for that world.

→	→	→	+1
↑		↑	-1
↑	←	←	←

- The following figure shows the corresponding utilities

0.812	0.868	0.918	+1
0.762		0.560	-1
0.705	0.655	0.611	0.388

- Clearly, the passive learning task is similar to the policy evaluation task.
- The main difference is that the passive learning agent does not know
 - Neither the transition model $T(s, a, s')$, which specifies the probability of reaching state s' from state s after doing action a ;
 - Nor does it know the reward function $R(s)$, which specifies the reward for each state.
- The agent executes a set of trials in the environment using its policy π .
- In each trial, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states, (4,2) or (4,3).
- Its percepts supply both the current state and the reward received in that state.
- Typical trials might look like this:

(1,1)-0.4 → (1,2)-0.4 → (1,3)-0.4 → (1,2)-0.4 → (1,3)-0.4 → (2,3)-0.4 → (3,3)-0.4 → (4,3)+1
 (1,1)-0.4 → (1,2)-0.4 → (1,3)-0.4 → (2,3)-0.4 → (3,3)-0.4 → (3,2)-0.4 → (3,3)-0.4 → (4,3)+1
 (1,1)-0.4 → (2,1)-0.4 → (3,1)-0.4 → (3,2)-0.4 → (4,2)-1 →

- Note that each state percept is subscripted with the reward received.
- The object is to use the information about rewards to learn the expected utility $U^\pi(s)$ associated with each nonterminal state s .
- The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed, the utility function is written as $U^\pi(s)$.

$$U(s) = E \left[\sum_{t=0}^{\infty} \gamma^t R(st) \mid s, s_0 \right]$$

- For the 4 x 3 world set $\gamma = 1$

Direct utility estimation:-

- A simple method for direct utility estimation is in the area of adaptive control theory by Widrow and Hoff(1960).
- The idea is that the utility of a state is the expected total reward from that state onward, and each trial provides a sample of this value for each state visited.
- Example:- The first trial in the set of three given earlier provides a sample total reward of 0.72 for state (1,1), two samples of 0.76 and 0.84 for (1,2), two samples of 0.80 and 0.88 for (1,3) and so on.
- Thus at the end of each sequence, the algorithm calculates the observed reward- to-go for each state and updates the estimated utility for that state accordingly.
- In the limit of infinitely many trails, the sample average will come together to the true expectations in the utility function.
- It is clear that direct utility estimation is just an instance of supervised learning.
- This means that reinforcement learning have been reduced to a standard inductive learning problem.

Advantage:-

- Direct utility estimation succeeds in reducing the reinforcement learning problem to an inductive learning problem.

Disadvantage:-

- It misses a very important source of information, namely, the fact that the utilities of states are not independent
 - Reason:-** The utility of each state equals its own reward plus the expected utility of its successor states. That-is, the utility values obey the Bellman equations for a fixed policy
 - $U(s) = R(s) + \gamma \sum_{s'} T(s, a(s), s') U(s')$
- It misses opportunities for learning
 - Reason:-** It ignores the connections between states
- The algorithm often converges very slowly.
 - Reason:-** More broadly, direct utility estimation can be viewed as searching in a hypothesis space for U that is much larger than it needs to be, in that it includes many functions that violate the Bellman equations.

ADAPTIVE DYNAMIC PROGRAMMING:-

- Agent must learn how states are connected.
- Adaptive Dynamic Programming agent works by learning the transition model of the environment as it goes along and solving the corresponding Markov Decision process using a dynamic programming method.
- For passive learning agent, the transition model $T(s, a(s), s')$ and the observed rewards $R(S)$ into Bellman equation to calculate the utilities of the states.
- The process of learning the model itself is easy, because the environment is fully observable i.e. we have a supervised learning task where the input is a state-action pair and the output is

the resulting state.

- We can also represent the transition model as a table of probabilities.
- The following algorithm shows the passive ADP agent,

Function PASSIVE-ADP-AGENT(percept) **returns** an action

Inputs: percept, a percept indicating the current state s' and reward signal r'

Static: π , a fixed policy

Mdb, an MDP with model T , rewards R , discount γ

U , a table of utilities, initially empty

N_{sa} , a table of frequencies for state-action pairs, initially zero

N , a table of frequencies for state-action-state triples, initially zero

sas

S, a , the previous state and action, initially null

If s' is new **then do** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$

If s is not null **then do**

Increment $N_{sa}[s, a]$ and $N_{sas}'[s, a, s']$

For each t such that $N_{sas}'[s, a, t]$ is nonzero **do**

$T[s, a, t] \leftarrow N_{sas}'[s, a, t] / N_{sa}[s, a]$

$U \leftarrow \text{VALUE-DETERMINATION}(\pi, U, mdb)$

If TERMINALS? $[s']$ **then** $s, a \leftarrow \text{null}$ else $s, a \leftarrow s', \pi[s']$

return a

- Its performance on the 4×3 world is shown in the following figure.
- The following figure shows the root-mean square error in the estimate for $U(1,1)$, averaged over 20 runs of 100 trials each.

Advantages:-

It can converge quite quickly

Reason:- The model usually changes only slightly with each observation, the value iteration process can use the previous utility estimates as initial values.

The process of learning the model itself is easy

Reason:- The environment is fully observable. This means that a supervised learning task exists where the input is a state-action pair and the output is the resulting state.

It provides a standard against which other reinforcement learning algorithms can be measured.

Disadvantage:-

It is intractable for large state spaces

Temporal Difference Learning:-

- In order to approximate the constraint equation $U^\pi(S)$, use the observed transitions to adjust the values of the observed states, so that they agree with the constraint equation.
- When the transition occurs from S to S^1 , we apply the following update to $U^\pi(S)$

$$U^\pi(S) \leftarrow U^\pi(S) + \alpha (R(S) + U^\pi(S^1) - U^\pi(S))$$

- Where α = learning rate parameter.
- The above equation is called Temporal difference or TD equation.
- The following algorithm shows the passive reinforcement learning agent using temporal differences,

Function PASSIVE-TD-AGENT(precept)**returns** an action

Inputs:percept,a percept indicating the current state s' and reward signal r'

Static: π ,a fixed policy

U ,a table of utilities,initially empty

N_s ,a table of frequencies for states,initially zero

S,a,r ,the previous state,action,and reward,initially null

If s' is new **then** $U[s'] \leftarrow r'$

If s is not null **then do**

Increment $N_s[s]$

$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$

If TERMINAL? $[s']$ **then** $s,a,r \leftarrow \text{null}$ **else** $s,a,r \leftarrow s',\pi[s'],r'$

return a

- **Advantages:-**
 - It is much simpler
 - It requires much less computation per observation
- **Disadvantages:-**
 - It does not learn quite as fast as the ADP agent
 - It shows much higher variability
- The following table shows the difference between ADP and TD approach,

ADP Approach	TD Approach
ADP adjusts the state to agree with all of the successors that might occur, weighted by their probabilities	TD adjusts a state to agree with its observed successor
ADP makes as many adjustments as it needs to restore consistency between the utility estimates U and the environment model T	TD makes a single adjustment per observed transition

- The following points shows the relationship between ADP and TD approach,
 - Both try to make local adjustments to the utility estimates in order to make each state “agree” with its successors.
 - Each adjustment made by ADP could be seen, from the TD point of view, as a result of a “pseudo-experience” generated by simulating the current environment model.
 - It is possible to extend the TD approach to use an environment model to generate several “pseudo-experiences-transitions that the TD agent can imagine might happen, given its current model.

- For each observed transition, the TD agent can generate a large number of imaginary transitions. In this way the resulting utility estimates will approximate more and more closely those of ADP- of course, at the expense of increased computation time.

Active Reinforcement learning:-

- A passive learning agent has a fixed policy that determines its behavior.
- **“An active agent must decide what actions to do”**
- An ADP agent can be taken and considered how it must be modified to handle this new freedom.
- The following are the **required modifications**:-
 - First the agent will need to learn a complete model with outcome probabilities for all actions. The simple learning mechanism used by PASSIVE-ADP-AGENT will do just fine for this.
 - Next, take into account the fact that the agent has a choice of actions. The utilities it needs to learn are those defined by the optimal policy.

$$U(s) \approx R(s) + \sum_a T(s, a, s') U(s')$$
 - These equations can be solved to obtain the utility function U using the value iteration or policy iteration algorithms.
 - Having obtained a utility function U that is optimal for the learned model, the agent can extract an optimal action by one-step look ahead to maximize the expected utility;
 - Alternatively, if it uses policy iteration, the optimal policy is already available, so it should simply execute the action the optimal policy recommends.

Exploration:-

- Greedy agent is an agent that executes an action recommended by the optimal policy for the learned model.
- The following figure shows the suboptimal policy to which this agent converges in this particular sequence of trials.

→	→	→	+1
↓		↑	-1
→	→	↑	↓

- The agent does not learn the true utilities or the true optimal policy! what happens is that, in the 39th trial, it finds a policy that reaches +1 reward along the lower route via (2,1), (3,1),(3,2), and (3,3).
- After experimenting with minor variations from the 276th trial onward it sticks to that policy, never learning the utilities of the other states and never finding the optimal route via (1,2),(1,3) and (2,3).
- Choosing the optimal action cannot lead to suboptimal results.
- The fact is that the learned model is not the same as the true environment; what is optimal in

the learned model can therefore be suboptimal in the true environment.

- Unfortunately, the agent does not know what the true environment is, so it cannot compute the optimal action for the true environment.
- Hence this can be done by the means of **Exploitation**.
- The greedy agent can overlook that actions do more than provide rewards according to the current learned model; they also contribute to learning the true model by affecting the percepts that are received.
- An agent therefore must make a trade-off between **exploitation** to maximize its reward and **exploration** to maximize its long-term well being.
- Pure exploitation risks getting stuck in a rut.
- Pure exploitation to improve ones knowledge id of no use if one never puts that knowledge into practice.

GLIE Scheme:-

- To come up with a reasonable scheme that will eventually lead to optimal behavior by the agent a GLIE Scheme can be used.
- A GLIE Scheme must try each action in each state an unbounded number of times to avoid having a finite probability that an optimal action is missed because of an unusually bad series of outcomes.
- An ADP agent using such a scheme will eventually learn the true environment model.
- A GLIE Scheme must also eventually become greedy, so that the agents actions become optimal with respect to the learned (and hence the true) model.
- There are several GLIE Scheme as follows,
 - The agent can choose a random action a fraction $1/t$ of the time and to follow the greedy policy otherwise.

Advantage:- This method eventually converges to an optimal policy

Disadvantage:- It can be extremely slow

- Another approach is to give some weight to actions that the agent has not tried very often, while tending to avoid actions that are believed to be of low utility. This can be implemented by altering the constraint equation, so that it assigns a higher utility estimate to relatively UP explored state-action pairs.
- Essentially, this amounts to an optimistic prior over the possible environments and causes the agent to behave initially as if there were wonderful rewards scattered all over the place.

Exploration function:-

- Let U^+ denotes the optimistic estimate of the utility of the state s , and let $N(a,s)$ be the number of times action a has been tried in state s .
- Suppose that value iteration is used in an ADP learning agent; then rewrite the update equation to incorporate the optimistic estimate.
- The following equation does this,

$$U^+(s) \leftarrow R(s) + \max_a \left[\frac{1}{N(a,s)} T(s, a, s') U^+(s'), N(a, s) \right]$$

$s \rightarrow s'$

- Here $f(u, n)$ is called the **exploration** function.
- It determines how greed is trade off against curiosity.
- The function $f(u, n)$ should be increasing in u and decreasing in n .
- The simple definition is

$$f(u, n) = \begin{cases} R^+ & \text{in } n < N_c \\ u & \text{otherwise} \end{cases}$$
 where R^+ = optimistic estimate of the best possible reward obtainable in any state and N_c is a fixed parameter.
- The fact that U^+ rather than U appears on the right hand side of the above equation is very important.
- If U is used, the more pessimistic utility estimate, then the agent would soon become unwilling to explore further a field.
- The use of U^+ means that benefits of exploration are propagated back from the edges of unexplored regions, so that actions that lead toward unexplored regions are weighted more highly, rather than just actions that are themselves unfamiliar.

Learning an action value function:-

- To construct an active temporal difference learning agent, it needs a change in the passive TD approach.
- The most obvious change that can be made in the passive case is that the agent is no longer equipped with a fixed policy, so if it learns a utility function U , it will need to learn a model in order to be able to choose an action based on U via one step look ahead.
- The update rule of passive TD remains unchanged. This might seem old.
- **Reason:-**
 - Suppose the agent takes a step that normally leads to a good destination, but because of non determinism in the environment the agent ends up in a disastrous state.
 - The TD update rule will take this as seriously as if the outcome had been the normal result of the action, where the agent should not worry about it too much since the outcome was a fluke.
 - It can be shown that the TD algorithm will converge to the same values as ADP as the number of training sequences tends to infinity.

Generalization in Reinforcement Learning:-

- The utility function and Q-functions learned by the agents are represented in tabular form with one output value for each input tuple.
- This approach works well for small sets spaces.
- **Example:-** The game of chess where the state spaces are of the order 10^{50} states. Visiting all the states to learn the game is tedious.
- One way to handle such problems is to use **FUNCTION APPROXIMATION**.
- **Function approximation** is nothing but using any sort of representation for the function other than the table.
- **For Example:-** The evaluation function for chess is represented as a weighted linear function of set of **features or basic functions** f_1, \dots, f_n

$$U(S) = w_0 + w_1 f_1(S) + w_2 f_2(S) + \dots + w_n f_n(S)$$

- The reinforcement learning can learn value for the parameters $\theta_1, \dots, \theta_n$.
- Such that the evaluation function U_θ approximates the true utility function.
- As in all inductive learning, there is a tradeoff between the size of the hypothesis space and the time it takes to learn the function.
- For reinforcement learning, it makes more sense to use an online learning algorithm that updates the parameter after each trial.
- Suppose we run a trial and the total reward obtained starting at $(1, 1)$ is 0.4.
- This suggests that $U_\theta(1,1)$, currently 0.8 is too large and must be reduced.
- The parameter should be adjusted to achieve this. This is done similar to neural network learning where we have an error function which computes the gradient with respect to the parameters.
- If $U_j(S)$ is the observed total reward for state S onward in the j th trial then the error is defined as half the squared difference of the predicted total and the actual total.

$$E^j(S) = (U(S) - U_j(S))^2 / 2$$
- The rate of change of error with respect to each parameter $\theta_i = E_j / \theta_i$, is to move the parameter in the direction of the decreasing error.

$$\theta_i = \theta_i - (E_j(S) / C_j) = \theta_i - (U_j(S) - U_\theta(S))(\partial U_\theta(S) / \partial \theta_i)$$
- This is called **Widrow-Hoff Rule or Delta Rule**.

Advantages:-

- It requires less space.
- Function approximation can also be very helpful for learning a model of the environment.
- It allows for inductive generalization over input states.

Disadvantages:-

- The convergence is likely to be displayed.
- It could fail to be any function in the chosen hypothesis space that approximates the true utility function sufficiently well.
- Consider the simplest case, which is direct utility estimation. With function approximation, this is an instance of supervised learning.

APPLICATIONS OF REINFORCEMENT LEARNING

1. RL in Marketing

Marketing is all about promoting and then, selling the products or services either of your brand or someone else's. In the process of marketing, finding the right audience which yields larger returns on investment you or your company is making is a challenge in itself.

And, it is one of the reasons companies are investing dollars in managing digitally various marketing campaigns. Through **real-time bidding** supporting well the fundamental capabilities of RL, your and other companies, smaller or larger, can expect: –

- more display ad impressions in real-time.
- increased ROI, profit margins.
- predicting the choices, reactions, and behavior of customers towards your products/services.

2. RL in Broadcast Journalism

Through **different types of Reinforcement Learning**, attracting likes and views along with tracking the reader's behavior is much simpler. Besides, recommending news that suits the frequently-changing preferences of readers and other online users can possibly be achieved since journalists can now be equipped with an RL-based system that keeps an eye on intuitive news content as well as the headlines. Take a look at other advantages too which Reinforcement Learning is offering to readers all around the world.

- News producers are now able to receive the feedback of their users instantaneously.
- Increased communication, as users are more expressive now.
- No space for disinformation, hatred.

3. RL in Healthcare

Healthcare is an important part of our lives and through DTRs (a sequence-based use-case of RL), doctors can discover the treatment type, appropriate doses of drugs, and timings for taking such doses. Curious to know how is this possible!! See, DTRs are equipped with: –

- a sequence of rules which confirm the current health status of a patient.
- Then, they optimally propose treatments that can diagnose diseases like diabetes, HIV, Cancer, and mental illness too.

If required, these DTRs (i.e. Dynamic Treatment Regimes) can reduce or remove the delayed impact of treatments through their multi-objective healthcare optimization solutions.

4. RL in Robotics

Robotics without any doubt facilitates **training a robot** in such a way that a robot can perform tasks – just like a human being can. But still, there is a bigger challenge the robotics industry is facing today – Robots aren't able to use common sense while making various moral, social decisions. Here, a combination of Deep Learning and Reinforcement Learning i.e. **Deep Reinforcement Learning** comes to the rescue to enable the robots with, "Learn How To Learn" model. With this, the robots can now: –

- manipulate their decisions by grasping well various objects visible to them.
- solve complicated tasks which even humans fail to do as robots now know what and how to learn from different levels of abstractions of the types of datasets available to them.

5. RL in Gaming

Gaming is something nowadays without which you, me, or a huge chunk of people can't live. With **games optimization through Reinforcement Learning** algorithms, we may expect better performances of our favorite games related to adventure, action, or mystery.

To prove it right, the Alpha Go example can be considered. This is a computer program that defeated the strongest Go (a challenging classical game) Player in October 2015 and itself became the strongest Go player. The trick of Alpha Go to defeat the player was Reinforcement Learning which kept on developing stronger as the game is constantly exposed to unexpected gaming challenges. Like Alpha Go, there are many other games available. Even you can also optimize your favorite games by applying appropriately prediction models which learn how to win in even complex situations through RL-enabled strategies.

6. RL in Image Processing

Image Processing is another important method of enhancing the current version of an image to extract some useful information from it. And there are some steps associated like:

- Capturing the image with machines like scanners.
- Analyzing and manipulating it.
- Using the output image obtained after analysis for representation, description-purposes.

Here, ML models like **Deep Neural Networks** (whose framework is Reinforcement Learning) can be leveraged for simplifying this trending image processing method. With Deep Neural Networks, you can either enhance the quality of a specific image or hide the info. of that image. Later, use it for any of your computer vision tasks.

7. RL in Manufacturing

Manufacturing is all about producing goods that can satisfy our basic needs and essential wants. **Cobot Manufacturers** (or Manufacturers of **Collaborative Robots** that can perform various manufacturing tasks with a workforce of more than 100 people) are helping a lot of businesses with their own RL solutions for packaging and quality testing. Undoubtedly, their use is making the process of manufacturing quality products faster that can say a big no to negative customer feedback. And the lesser negative feedbacks are, the better is the product's performance and also, sales margin too.

POLICY SEARCH

Policy search is a subfield in reinforcement learning which focuses on finding good parameters for a given policy parametrization. It is well suited for robotics as it can cope with high-dimensional state and action spaces, one of the main challenges in robot learning.

1. Policy search methods are a family of systematic approaches for continuous (or large) actions and state space.
2. With policy search, expert knowledge is easily embedded in initial policies (by demonstration, imitation).
3. Policy search is more preferred than other RL methods in practical applications (e.g. robotics).



Policy gradient

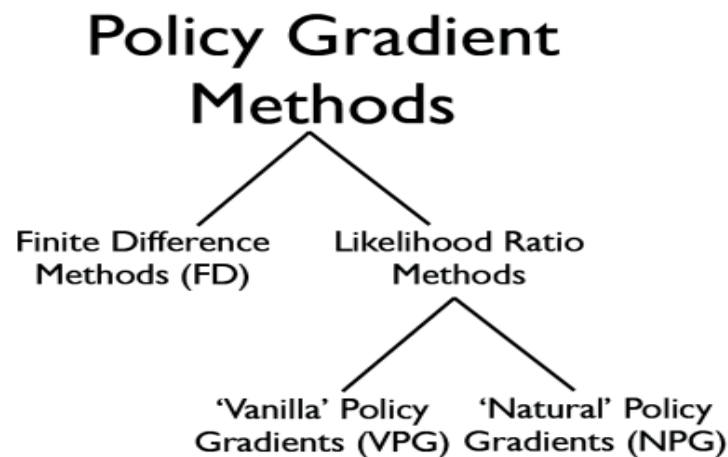
- Family of randomized policy $\mu(s, a) = \Pr(a|s)$ (deterministic policy is a special case).
- the performance measure is $J(\mu) = E_{\mu} r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$
- Policy $\mu_{\theta}(s, a)$ is parameterized by a parameter space $\theta \in$

The parametric performance measure becomes $J(\theta) = E_{\theta} r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$

- Solution: gradient-descent algorithms. $\theta_{k+1} = \theta_k + \alpha \nabla_{\theta} J(\theta_k)$

Policy gradient updates $\theta_{\mu 0} = \theta_{\mu} + \alpha \nabla_{\theta} J(\theta_{\mu})$

- Guarantee the performance improvement: $J(\theta_{\mu 0}) \geq J(\theta_{\mu}) \Rightarrow \theta_{\mu 0}$ at least better than or equal to θ_{μ}



Policy gradient: Black-box approaches

- Approximate the gradient using supervised learning (regression).
- Collect data $D = \{\delta \theta_i, \delta J_i\}$ (the sampled gradients). By – perturbing the parameters: $\theta + \delta \theta$ – applying the new policy $\mu(\theta + \delta \theta)$ to get $\delta J_i = J(\theta + \delta \theta) - J(\theta)$
- the finite different (FD) gradient estimation by regression $g_{FD}(\theta) = (\Delta \theta > \Delta \theta)^{-1} \Delta \theta > \Delta J$
- gradient update $\theta \leftarrow \theta + \alpha g_{FD}(\theta)$

Policy gradient: Likelihood Ratio Gradient

- rewrite the performance measure

$$J(\theta) = E_{\theta} [r_0 + \gamma r_1 + \gamma^2 r_2 + \dots] = \int p(\xi|\mu_{\theta}) R(\xi) d\xi$$

where $\xi = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots\}$ is a trajectory.

$$R(\xi) = r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$

$$p(\xi|\mu_{\theta}) = p(s_0) \prod_{t=0}^{\infty} p(s_{t+1}|s_t, a_t) \mu_{\theta}(a_t|s_t)$$

- Gradient derivation

$$\nabla_{\theta} J(\theta) = \int \nabla_{\theta} p(\xi|\mu_{\theta}) R(\xi) d\xi$$

$$\xi = \int p(\xi|\mu_{\theta}) \nabla_{\theta} \log p(\xi|\mu_{\theta}) R(\xi) d\xi$$

(the trick is $\nabla f = f \nabla \log f$)

$$= E_{\xi} [\nabla_{\theta} \log p(\xi|\mu_{\theta}) R(\xi)]$$

- Using Monte-Carlo simulations: sampling M trajectories ξ_i from policy μ_{θ} .

$$\nabla_{\theta} J(\theta) \approx \frac{1}{M} \sum_{i=1}^M \nabla_{\theta} \log p(\xi_i|\mu_{\theta}) R(\xi_i)$$

$$= \frac{1}{M} \sum_{i=1}^M \sum_{t=0}^{\infty} \nabla_{\theta} \log \mu_{\theta}(a_t|s_t) R(\xi_i)$$

because $p(s_{t+1}|s_t, a_t)$ not depends on θ , so its gradient w.r.t θ is 0

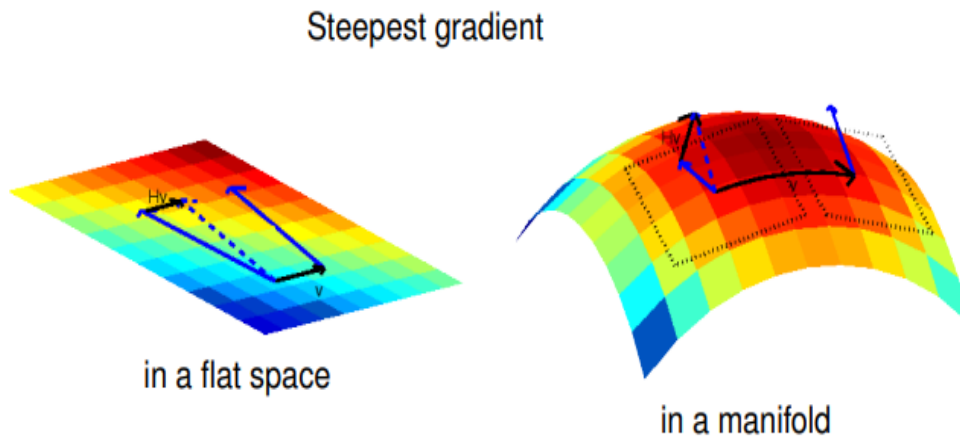
A vanilla policy gradient algorithm

- initialize θ_0
- for $k = 0 : \infty$ (until convergence)
 - =generate M trajectories ξ_i from policy μ_{θ_k}
 - =compute $\nabla_{\theta} J(\theta_k)$ from $\{\xi_i\}_{i=1}^M$
 - update θ : $\theta_{k+1} = \theta_k + \alpha \nabla_{\theta} J(\theta_k)$
- end for

Natural policy gradient

$$\tilde{\nabla} J(\theta) = G^{-1}(\theta) \nabla J(\theta)$$

where $G(\theta)$ is the Fisher Information Matrix (Amari, 1998)



Natural Policy Gradient: Steepest descent

- The optimization (maximize $J(\theta)$) is over the space of trajectories, when considering $p(\xi|\mu\theta)$ is a function of parameter ξ (instead of θ) of dimension $|\theta|$.

- We try to define a Riemannian structure on the manifold of trajectories ξ

The optimization (maximize $J(\theta)$) is over the space of trajectories, when considering $p(\xi|\mu\theta)$ is a function of parameter ξ (instead of θ) of dimension $|\theta|$.

- We try to define a Riemannian structure on the manifold of trajectories ξ

- The steepest descent should minimize the $J(\theta + \delta\theta)$ after update. This is formulated as an optimization problem $\min J(\theta + \delta\theta) = J(\theta) + \delta\theta \nabla J(\theta)$ subject to $\|\delta\theta\|, \delta\theta \in p(\xi|\mu\theta) =$

Using Lagrange multiplier

$$L(\theta, \lambda) = J(\theta) + \delta\theta \nabla J(\theta) + \lambda (X_{i,j} G_{i,j} \delta\theta_i, \delta\theta_j -)$$

- Taking derivative

$$\theta_i, \nabla J(\theta) + \lambda G_{i,j} \delta\theta_j = 0$$

- Therefore, $\delta = G^{-1} \nabla J(\theta)$

The metric is the distances on probability spaces.

The KL-divergence between two distributions is a natural divergence on changes in a distribution.

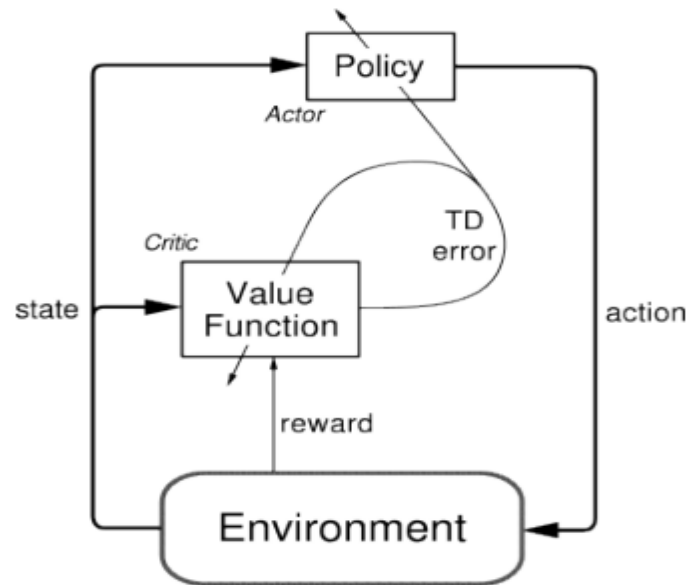
$$G(\theta) = \mathbb{E} \left[\log p(\xi|\mu\theta), \delta \log p(\xi|\mu\theta) \right]$$

- Estimate the Fisher information matrix using sampled trajectories.

$$G(\theta) \approx \frac{1}{M} \sum_{i=1}^M [\nabla_{\theta} \log p(\xi_i | \mu_{\theta}) R(\xi_i)] \times [\nabla_{\theta} \log p(\xi_i | \mu_{\theta}) R(\xi_i)]^T$$

$$= \frac{1}{M} \sum_{i=1}^M \left[\sum_{t=0}^T \nabla_{\theta} \log \mu_{\theta}(a_t | s_t) R(\xi_i) \right] \times \left[\sum_{t=0}^T \nabla_{\theta} \log \mu_{\theta}(a_t | s_t) R(\xi_i) \right]^T$$

Actor-Critic methods



1. The policy structure is known as the actor, (\rightarrow tuned by gradient updates.)
2. The estimated value function is known as the critic (\rightarrow tuned by TD error).

Policy Gradient Theorem Algorithm

- $R(\xi)$ can be estimated by $Q \pi_t(s_t, a_t)$
- $$\nabla_{\theta} J(\theta) = E \left[\sum_{t=0}^T \nabla_{\theta} \log \mu_{\theta}(a_t | s_t) \sum_{j=t}^T r_j \right]$$
- $$= E \left[\sum_{t=0}^T \nabla_{\theta} \log \mu_{\theta}(a_t | s_t) Q \pi_t(s_t, a_t) \right]$$

Policy gradient with function approximation

$$Q \pi_t(s_t, a_t) = \phi(s_t, a_t) \cdot w$$

- Compatible function approximation: how to choose $\phi(s_t, a_t)$? such that
 - does not introduce bias
 - reduce the variance

UNIT-1V

NATURAL LANGUAGE FOR COMMUNICATION

Concept of Grammar

Grammar is very essential and important to describe the syntactic structure of well-formed programs. In the literary sense, they denote syntactical rules for conversation in natural languages. Linguistics have attempted to define grammars since the inception of natural languages like English, Hindi, etc.

The theory of formal languages is also applicable in the fields of Computer Science mainly in programming languages and data structure. For example, in 'C' language, the precise grammar rules state how functions are made from lists and statements.

A mathematical model of grammar was given by **Noam Chomsky** in 1956, which is effective for writing computer languages.

Mathematically, a grammar G can be formally written as a 4-tuple (N, T, S, P) where –

- N or V_N = set of non-terminal symbols, i.e., variables.
- T or Σ = set of terminal symbols.
- S = Start symbol where $S \in N$
- P denotes the Production rules for Terminals as well as Non-terminals. It has the form $\alpha \rightarrow \beta$, where α and β are strings on $V_N \cup \Sigma$ and least one symbol of α belongs to V_N

Phrase Structure or Constituency Grammar

Phrase structure grammar, introduced by Noam Chomsky, is based on the constituency relation. That is why it is also called constituency grammar. It is opposite to dependency grammar.

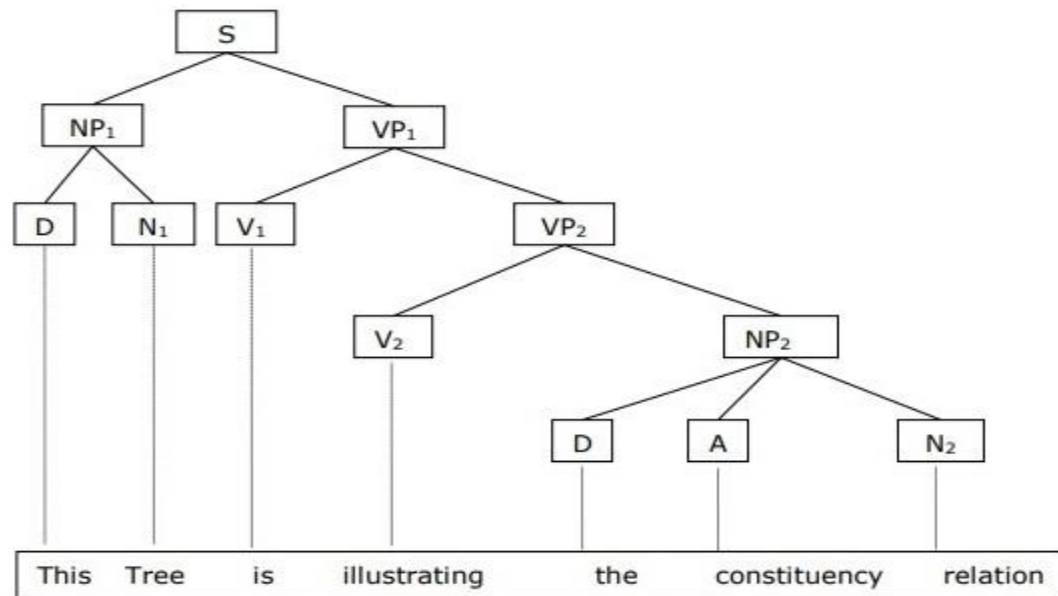
Example

Before giving an example of constituency grammar, we need to know the fundamental points about constituency grammar and constituency relation.

- All the related frameworks view the sentence structure in terms of constituency relation.
- The constituency relation is derived from the subject-predicate division of Latin as well as Greek grammar.

- The basic clause structure is understood in terms of **noun phrase NP** and **verb phrase VP**.

We can write the sentence “**This tree is illustrating the constituency relation**” as follows –



Dependency Grammar

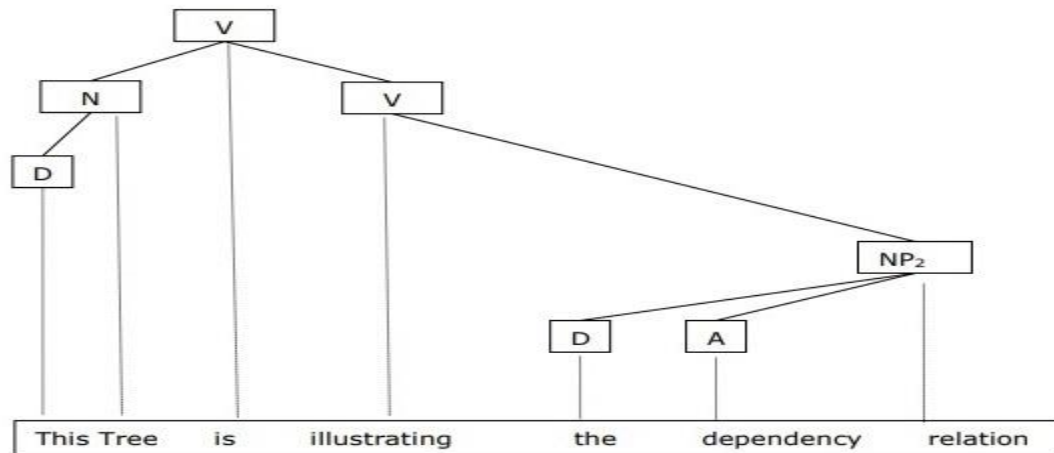
It is opposite to the constituency grammar and based on dependency relation. It was introduced by Lucien Tesniere. Dependency grammar (DG) is opposite to the constituency grammar because it lacks phrasal nodes.

Example

Before giving an example of Dependency grammar, we need to know the fundamental points about Dependency grammar and Dependency relation.

- In DG, the linguistic units, i.e., words are connected to each other by directed links.
- The verb becomes the center of the clause structure.
- Every other syntactic units are connected to the verb in terms of directed link. These syntactic units are called **dependencies**.

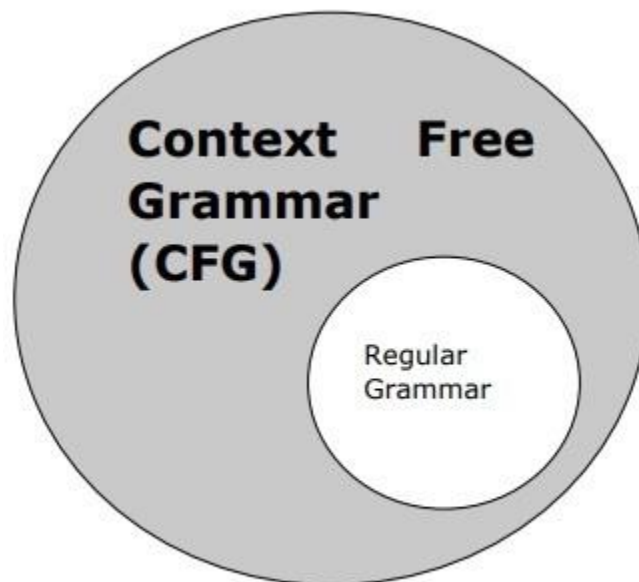
We can write the sentence “**This tree is illustrating the dependency relation**” as follows;



Parse tree that uses Constituency grammar is called constituency-based parse tree; and the parse trees that uses dependency grammar is called dependency-based parse tree.

Context Free Grammar

Context free grammar, also called CFG, is a notation for describing languages and a superset of Regular grammar. It can be seen in the following diagram –



Definition of CFG

CFG consists of finite set of grammar rules with the following four components –

Set of Non-terminals

It is denoted by V . The non-terminals are syntactic variables that denote the sets of strings, which further help defining the language, generated by the grammar.

Set of Terminals

It is also called tokens and defined by Σ . Strings are formed with the basic symbols of terminals.

Set of Productions

It is denoted by P . The set defines how the terminals and non-terminals can be combined. Every production(P) consists of non-terminals, an arrow, and terminals (the sequence of terminals). Non-terminals are called the left side of the production and terminals are called the right side of the production.

Start Symbol

The production begins from the start symbol. It is denoted by symbol S . Non-terminal symbol is always designated as start symbol.

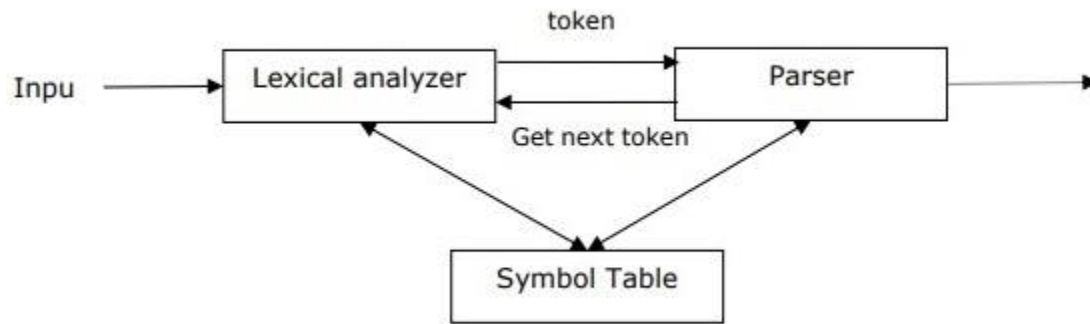
Syntactic Analysis

Syntactic analysis or parsing or syntax analysis is the third phase of NLP. The purpose of this phase is to draw exact meaning, or you can say dictionary meaning from the text. Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar. For example, the sentence like “hot ice-cream” would be rejected by semantic analyzer.

In this sense, syntactic analysis or parsing may be defined as the process of analyzing the strings of symbols in natural language conforming to the rules of formal grammar. The origin of the word ‘**parsing**’ is from Latin word ‘**pars**’ which means ‘**part**’.

Concept of Parser

It is used to implement the task of parsing. It may be defined as the software component designed for taking input data (text) and giving structural representation of the input after checking for correct syntax as per formal grammar. It also builds a data structure generally in the form of parse tree or abstract syntax tree or other hierarchical structure.



The main roles of the parse include –

- To report any syntax error.
- To recover from commonly occurring error so that the processing of the remainder of program can be continued.
- To create parse tree.
- To create symbol table.
- To produce intermediate representations (IR).

Types of Parsing

Derivation divides parsing into the followings two types –

- Top-down Parsing
- Bottom-up Parsing

Top-down Parsing

In this kind of parsing, the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input. The most common form of topdown parsing uses recursive procedure to process the input. The main disadvantage of recursive descent parsing is backtracking.

Bottom-up Parsing

In this kind of parsing, the parser starts with the input symbol and tries to construct the parser tree up to the start symbol.

Concept of Derivation

In order to get the input string, we need a sequence of production rules. Derivation is a set of production rules. During parsing, we need to decide the non-terminal, which is to be replaced along with deciding the production rule with the help of which the non-terminal will be replaced.

Types of Derivation

In this section, we will learn about the two types of derivations, which can be used to decide which non-terminal to be replaced with production rule –

Left-most Derivation

In the left-most derivation, the sentential form of an input is scanned and replaced from the left to the right. The sentential form in this case is called the left-sentential form.

Right-most Derivation

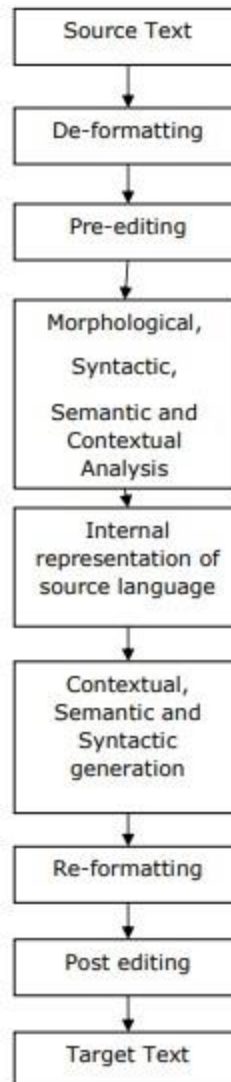
In the left-most derivation, the sentential form of an input is scanned and replaced from right to left. The sentential form in this case is called the right-sentential form.

Concept of Parse Tree

It may be defined as the graphical depiction of a derivation. The start symbol of derivation serves as the root of the parse tree. In every parse tree, the leaf nodes are terminals and interior nodes are non-terminals. A property of parse tree is that in-order traversal will produce the original input string.

MACHINE TRANSLATION

Machine translation (MT), process of translating one source language or text into another language, is one of the most important applications of NLP. We can understand the process of machine translation with the help of the following flowchart –



Types of Machine Translation Systems

There are different types of machine translation systems. Let us see what the different types are.

Bilingual MT System

Bilingual MT systems produce translations between two particular languages.

Multilingual MT System

Multilingual MT systems produce translations between any pair of languages. They may be either uni-directional or bi-directional in nature.

Approaches to Machine Translation (MT)

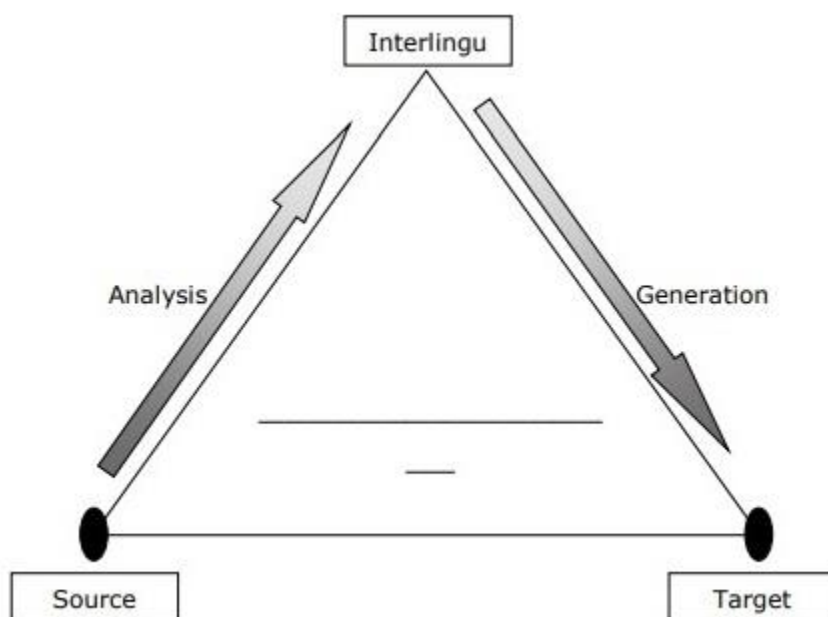
Let us now learn about the important approaches to Machine Translation. The approaches to MT are as follows –

Direct MT Approach

It is less popular but the oldest approach of MT. The systems that use this approach are capable of translating SL (source language) directly to TL (target language). Such systems are bi-lingual and uni-directional in nature.

Interlingua Approach

The systems that use Interlingua approach translate SL to an intermediate language called Interlingua (IL) and then translate IL to TL. The Interlingua approach can be understood with the help of the following MT pyramid –



Transfer Approach

Three stages are involved with this approach.

- In the first stage, source language (SL) texts are converted to abstract SL-oriented representations.

- In the second stage, SL-oriented representations are converted into equivalent target language (TL)-oriented representations.
- In the third stage, the final text is generated.

Empirical MT Approach

This is an emerging approach for MT. Basically, it uses large amount of raw data in the form of parallel corpora. The raw data consists of the text and their translations. Analogybased, example-based, memory-based machine translation techniques use empirical MT approach.

Speech recognition

Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition, or speech-to-text, is a capability which enables a program to process human speech into a written format. While it's commonly confused with voice recognition, speech recognition focuses on the translation of speech from a verbal format to a text one whereas voice recognition just seeks to identify an individual user's voice.

Key features of effective speech recognition

Many speech recognition applications and devices are available, but the more advanced solutions use AI and machine learning. They integrate grammar, syntax, structure, and composition of audio and voice signals to understand and process human speech. Ideally, they learn as they go — evolving responses with each interaction.

The best kind of systems also allow organizations to customize and adapt the technology to their specific requirements — everything from language and nuances of speech to brand recognition. For example:

- **Language weighting:** Improve precision by weighting specific words that are spoken frequently (such as product names or industry jargon), beyond terms already in the base vocabulary.
- **Speaker labeling:** Output a transcription that cites or tags each speaker's contributions to a multi-participant conversation.
- **Acoustics training:** Attend to the acoustical side of the business. Train the system to adapt to an acoustic environment (like the ambient noise in a call center) and speaker styles (like voice pitch, volume and pace).
- **Profanity filtering:** Use filters to identify certain words or phrases and sanitize speech output.

Meanwhile, speech recognition continues to advance. Companies, like IBM, are making inroads in several areas, the better to improve human and machine interaction.

Speech recognition algorithms

The vagaries of human speech have made development challenging. It's considered to be one of the most complex areas of computer science – involving linguistics, mathematics and statistics. Speech recognizers are made up of a few components, such as the speech input, feature extraction, feature vectors, a decoder, and a word output. The decoder leverages acoustic models, a pronunciation dictionary, and language models to determine the appropriate output.

Speech recognition technology is evaluated on its accuracy rate, i.e. word error rate (WER), and speed. A number of factors can impact word error rate, such as pronunciation, accent, pitch, volume, and background noise. Reaching human parity – meaning an error rate on par with that of two humans speaking – has long been the goal of speech recognition systems. Research from Lippmann (link resides outside IBM) (PDF, 344 KB) estimates the word error rate to be around 4 percent, but it's been difficult to replicate the results from this paper.

Various algorithms and computation techniques are used to recognize speech into text and improve the accuracy of transcription. Below are brief explanations of some of the most commonly used methods:

- **Natural language processing (NLP):** While NLP isn't necessarily a specific algorithm used in speech recognition, it is the area of artificial intelligence which focuses on the interaction between humans and machines through language through speech and text. Many mobile devices incorporate speech recognition into their systems to conduct voice search—e.g. Siri—or provide more accessibility around texting.
- **Hidden markov models (HMM):** Hidden Markov Models build on the Markov chain model, which stipulates that the probability of a given state hinges on the current state, not its prior states. While a Markov chain model is useful for observable events, such as text inputs, hidden markov models allow us to incorporate hidden events, such as part-of-speech tags, into a probabilistic model. They are utilized as sequence models within speech recognition, assigning labels to each unit—i.e. words, syllables, sentences, etc.—in the sequence. These labels create a mapping with the provided input, allowing it to determine the most appropriate label sequence.
- **N-grams:** This is the simplest type of language model (LM), which assigns probabilities to sentences or phrases. An N-gram is sequence of N-words. For example, “order the pizza” is a trigram or 3-gram and “please order the pizza” is a 4-gram. Grammar and the probability of certain word sequences are used to improve recognition and accuracy.

- **Neural networks:** Primarily leveraged for deep learning algorithms, neural networks process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold) and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. While neural networks tend to be more accurate and can accept more data, this comes at a performance efficiency cost as they tend to be slower to train compared to traditional language models.
- **Speaker Diarization (SD):** Speaker diarization algorithms identify and segment speech by speaker identity. This helps programs better distinguish individuals in a conversation and is frequently applied at call centers distinguishing customers and sales agents.

SPEECH RECOGNITION USE CASES

- A wide number of industries are utilizing different applications of speech technology today, helping businesses and consumers save time and even lives. Some examples include:
- **Automotive:** Speech recognizers improves driver safety by enabling voice-activated navigation systems and search capabilities in car radios.
- **Technology:** Virtual agents are increasingly becoming integrated within our daily lives, particularly on our mobile devices. We use voice commands to access them through our smartphones, such as through Google Assistant or Apple’s Siri, for tasks, such as voice search, or through our speakers, via Amazon’s Alexa or Microsoft’s Cortana, to play music. They’ll only continue to integrate into the everyday products that we use, fueling the “Internet of Things” movement.
- **Healthcare:** Doctors and nurses leverage dictation applications to capture and log patient diagnoses and treatment notes.
- **Sales:** Speech recognition technology has a couple of applications in sales. It can help a call center transcribe thousands of phone calls between customers and agents to identify common call patterns and issues. AI chatbots can also talk to people via a webpage, answering common queries and solving basic requests without needing to wait for a contact center agent to be available. It both instances speech recognition systems help reduce time to resolution for consumer issues.
- **Security:** As technology integrates into our daily lives, security protocols are an increasing priority. Voice-based authentication adds a viable level of security.

PERCEPTION

- Perception is a process to interpret, acquire, select and then organize the sensory information that is captured from the real world.

For example: Human beings have sensory receptors such as touch, taste, smell, sight and hearing. So, the information received from these receptors is transmitted to human brain to organize the received information.

- According to the received information, action is taken by interacting with the environment to manipulate and navigate the objects.
- Perception and action are very important concepts in the field of Robotics. The following figures show the complete **autonomous robot**.

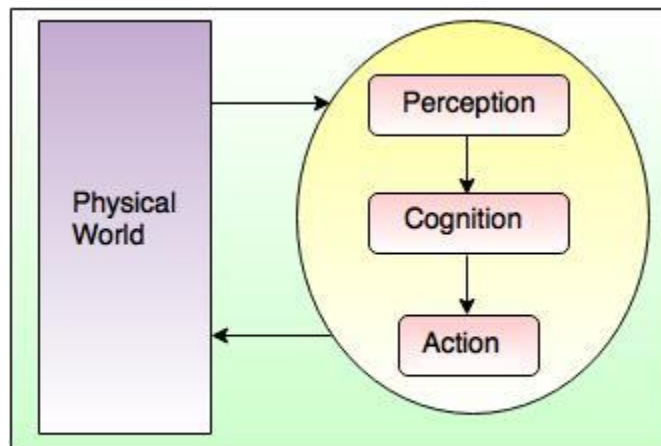


Fig: Autonomous Robot

- There is one important difference between the artificial intelligence program and robot. The AI program performs in a computer stimulated environment, while the robot performs in the physical world.

Example:

In chess, an AI program can be able to make a move by searching different nodes and has no facility to touch or sense the physical world.

However, the chess playing robot can make a move and grasp the pieces by interacting with the physical world.

Image formation

Image formation is a physical process that captures object in the scene through lens and creates a 2-D image.

Image formation in digital camera

Let's understand the geometry of a pinhole camera shown in the following diagram.

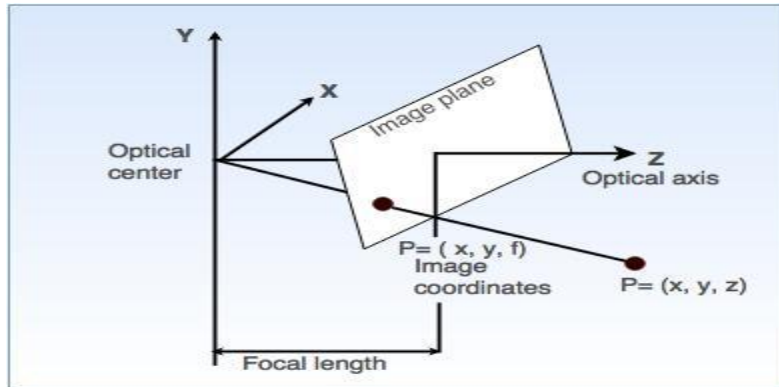


Fig: Geometry of Image Formation in the pinhole camera

In the above figure, an optical axis is perpendicular to the image plane and image plane is generally placed in front of the optical center.

So, let **P** be the point in the scene with coordinates (X,Y,Z) and **P'** be its image plane with coordinates (x, y, z).

If the focal length from the optical center is **f**, then by using properties of similar triangles, equation is derived as,

$$-x/f = X/Z \text{ so } x = -fX/Z \text{equation (i)}$$

$$-y/f = -Y/Z \text{ so } y = -fY/Z \text{equation (ii)}$$

These equations define an image formation process called as **perspective projection**.

What is the purpose of edge detection?

- Edge detection operation is used in an image processing.
- The main goal of edge detection is to construct the ideal outline of an image.
- Discontinuity in brightness of image is affected due to:
 - i) Depth discontinuities
 - ii) Surface orientation discontinuities

- iii) Reflectance discontinuities
- iv) Illumination.

3D-Information extraction using vision

The 3-D information extraction process plays an important role to perform the tasks like manipulation, navigation and recognition. It deals with the following aspects:

1. Segmentation of the scene

The segmentation is used to arrange the array of image pixels into regions. This helps to match semantically meaningful entities in the scene.

- The goal of segmentation is to divide an image into regions which are homogeneous.
- The union of the neighboring regions should not be homogeneous.
- **Thresholding** is the simplest technique of **segmentation**. It is simply performed on the object, which has an homogeneous intensity and a background with a different intensity level and the pixels are partitioned depending on their intensity values.
-

2. To determine the position and orientation of each object

Determination of the position and orientation of each object relative to the observer is important for manipulation and navigation tasks.

For example: Suppose a person goes to a store to buy something. While moving around he must know the locations and obstacles, so that he can make the plan and path to avoid them.

- The whole orientation of image should be specified in terms of a three dimensional rotation.
-

3. To determine the shape of each and every object

When the camera moves around an object, the distance and orientation of that object will change but it is important to preserve the shape of that object.

For example: If an object is cube, that fact does not change, but it is difficult to represent the global shape to deal with wide variety of objects present in the real world.

- If the shape of an object is same for some manipulating tasks, it becomes easy to decide how to grasp that object from a particular place.
- The **object recognition** plays most significant role to identify and classify the objects as an example only when the geometric shapes are provided with color and texture

There are number of techniques available in the visual stimulus for 3D-image extraction such as **motion, binocular stereopsis, texture, shading, and contour**. Each of these techniques operates on the background assumptions about physical scene to provide interpretation.

Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it. The image processing system usually treats all images as 2D signals when applying certain predetermined signal processing methods.

There are five main types of image processing:

- Visualization - Find objects that are not visible in the image
- Recognition - Distinguish or detect objects in the image
- Sharpening and restoration - Create an enhanced image from the original image
- Pattern recognition - Measure the various patterns around the objects in the image
- Retrieval - Browse and search images from a large database of digital images that are similar to the original image

Fundamental Image Processing Steps

Image Acquisition

Image acquisition is the first step in image processing. This step is also known as preprocessing in image processing. It involves retrieving the image from a source, usually a hardware-based source.

Image Enhancement

Image enhancement is the process of bringing out and highlighting certain features of interest in an image that has been obscured. This can involve changing the brightness, contrast, etc.

Image Restoration

Image restoration is the process of improving the appearance of an image. However, unlike image enhancement, image restoration is done using certain mathematical or probabilistic models.

Color Image Processing

Color image processing includes a number of color modeling techniques in a digital domain. This step has gained prominence due to the significant use of digital images over the internet.

Wavelets and Multiresolution Processing

Wavelets are used to represent images in various degrees of resolution. The images are subdivided into wavelets or smaller regions for data compression and for pyramidal representation.

Compression

Compression is a process used to reduce the storage required to save an image or the bandwidth required to transmit it. This is done particularly when the image is for use on the Internet.

Morphological Processing

Morphological processing is a set of processing operations for morphing images based on their shapes.

Segmentation

Segmentation is one of the most difficult steps of image processing. It involves partitioning an image into its constituent parts or objects.

Representation and Description

After an image is segmented into regions in the segmentation process, each region is represented and described in a form suitable for further computer processing. Representation deals with the image's characteristics and regional properties. Description deals with extracting quantitative information that helps differentiate one class of objects from the other.

RECOGNITION

Recognition assigns a label to an object based on its description.

Applications of Image Processing

Medical Image Retrieval

Image processing has been extensively used in medical research and has enabled more efficient and accurate treatment plans. For example, it can be used for the early detection of breast cancer using a sophisticated nodule detection algorithm in breast scans. Since medical usage calls for highly trained image processors, these applications require significant implementation and evaluation before they can be accepted for use.

Traffic Sensing Technologies

In the case of traffic sensors, we use a video image processing system or VIPS. This consists of a) an image capturing system b) a telecommunication system and c) an image processing system. When capturing video, a VIPS has several detection zones which output an “on” signal whenever a vehicle enters the zone, and then output an “off” signal whenever the vehicle exits the detection zone. These detection zones can be set up for multiple lanes and can be used to sense the traffic in a particular station.



Left - normal traffic image | Right - a VIPS image with detection zones (source)

Besides this, it can auto record the license plate of the vehicle, distinguish the type of vehicle, monitor the speed of the driver on the highway and lots more.

Image Reconstruction

Image processing can be used to recover and fill in the missing or corrupt parts of an image. This involves using image processing systems that have been trained extensively with existing photo datasets to create newer versions of old and damaged photos.

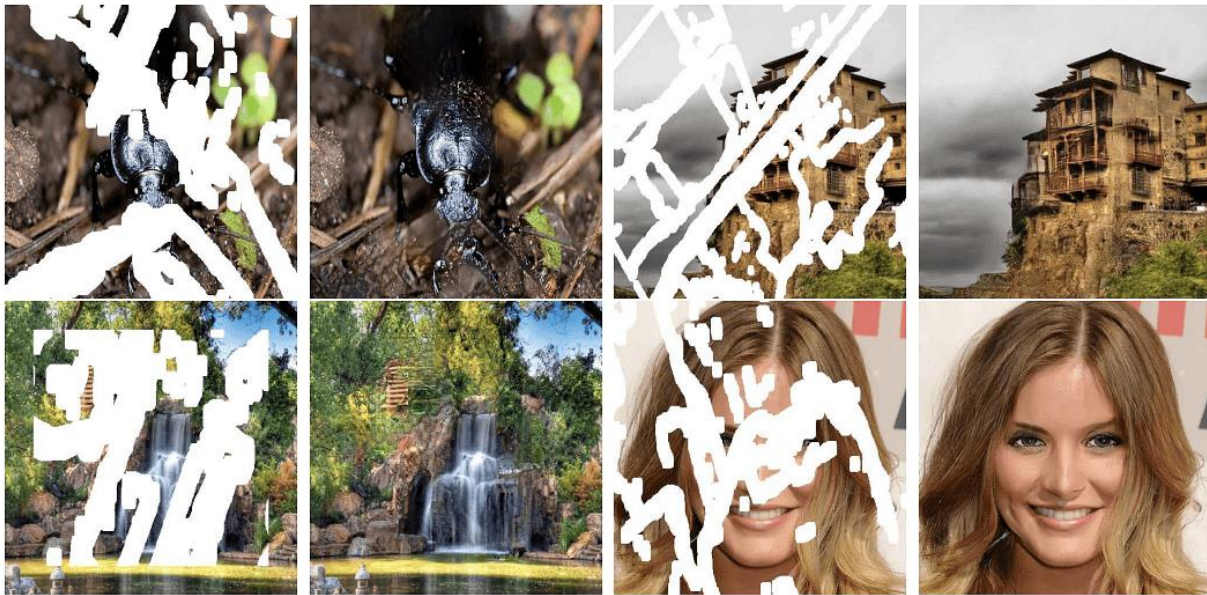


Fig: Reconstructing damaged images using image processing

Face Detection

One of the most common applications of image processing that we use today is face detection. It follows deep learning algorithms where the machine is first trained with the specific features of human faces, such as the shape of the face, the distance between the eyes, etc. After teaching the machine these human face features, it will start to accept all objects in an image that resemble a human face. Face detection is a vital tool used in security, biometrics and even filters available on most social media apps these days.

Benefits of Image Processing

The implementation of image processing techniques has had a massive impact on many tech organizations. Here are some of the most useful benefits of image processing, regardless of the field of operation:

- The digital image can be made available in any desired format (improved image, X-Ray, photo negative, etc)
- It helps to improve images for human interpretation
- Information can be processed and extracted from images for machine interpretation
- The pixels in the image can be manipulated to any desired density and contrast
- Images can be stored and retrieved easily
- It allows for easy electronic transmission of images to third-party providers

There are two methods of image processing:

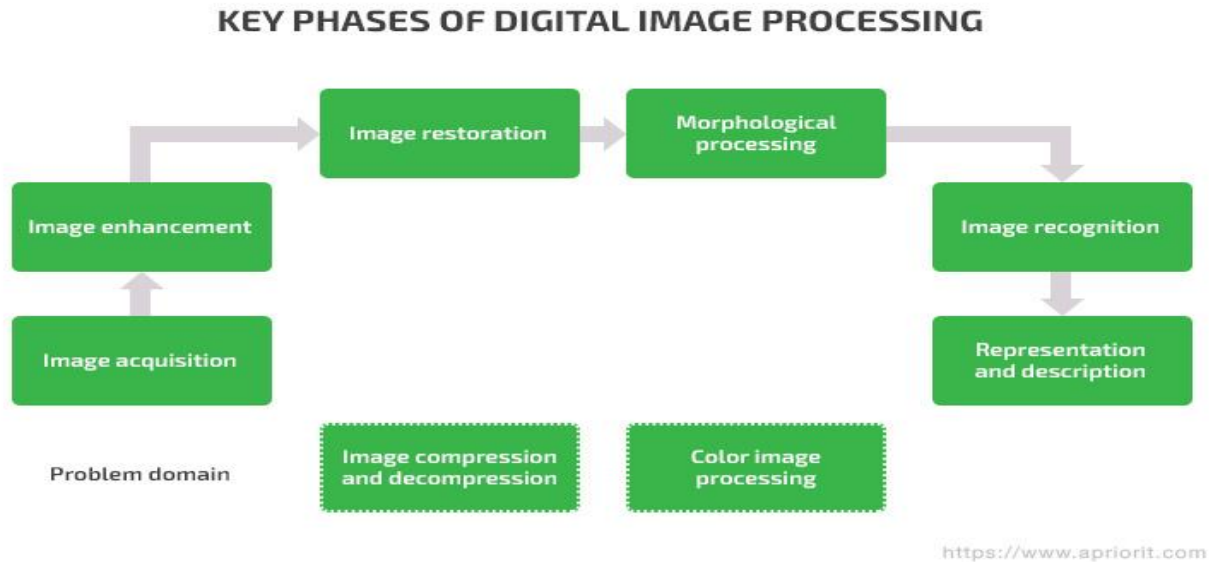
Analog Image processing: It is used for processing physical photographs, printouts and other hard copies of images

For analog image processing, the output is always an image.

Digital Image Processing: It is used for manipulating digital images with the help of computer algorithms

For digital image processing, however, the output may be an image or information associated with that image, such as data on features, characteristics, bounding boxes, or masks

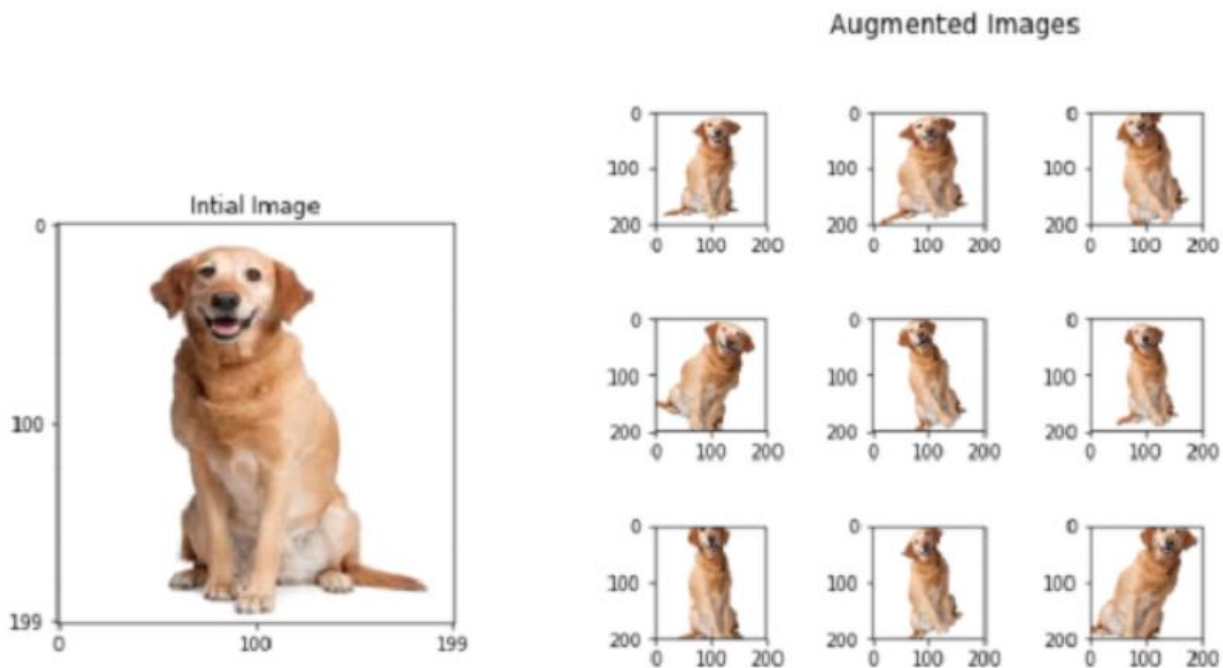
Digital image processing includes eight key phases:



1. **Image acquisition** is the process of capturing an image with a sensor (such as a camera) and converting it into a manageable entity (for example, a digital image file). One popular image acquisition method is scraping.
2. **Image enhancement** improves the quality of an image in order to extract hidden information from it for further processing.
3. **Image restoration** also improves the quality of an image, mostly by removing possible corruptions in order to get a cleaner version. This process is based mostly on probabilistic and mathematical models and can be used to get rid of blur, noise, missing pixels, camera misfocus, watermarks, and other corruptions that may negatively affect the training of a neural network.
4. **Color image processing** includes the processing of colored images and different color spaces. Depending on the image type, we can talk about pseudocolor processing (when colors are assigned grayscale values) or RGB processing (for images acquired with a full-color sensor).

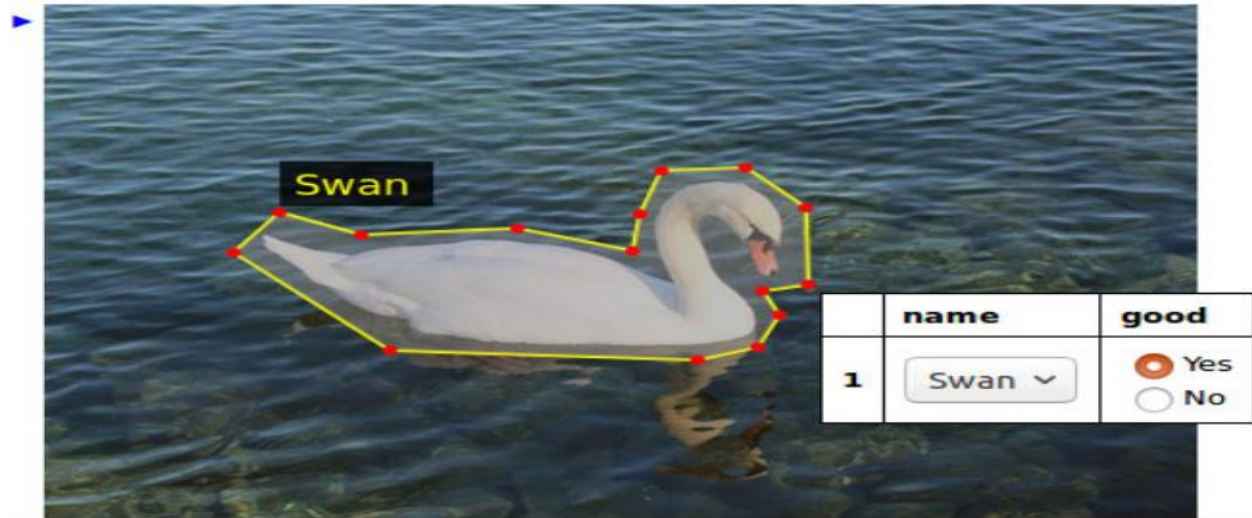
5. **Image compression and decompression** allow for changing the size and resolution of an image. Compression is responsible for reducing the size and resolution, while decompression is used for restoring an image to its original size and resolution.

These techniques are often used during the image augmentation process. When you lack data, you can extend your dataset with slightly augmented images. In this way, you can improve the way your neural network model generalizes data and make sure it provides high-quality results.



Example: Image Augmentation

6. **Morphological processing** describes the shapes and structures of the objects in an image. Morphological processing techniques can be used when creating datasets for training AI models. In particular, morphological analysis and processing can be applied at the annotation stage, when you describe what you want your AI model to detect or recognize.



An example of the annotation process of morphological analysis

7. **Image recognition** is the process of identifying specific features of particular objects in an image. AI-based image recognition often uses such techniques as object detection, object recognition, and segmentation.

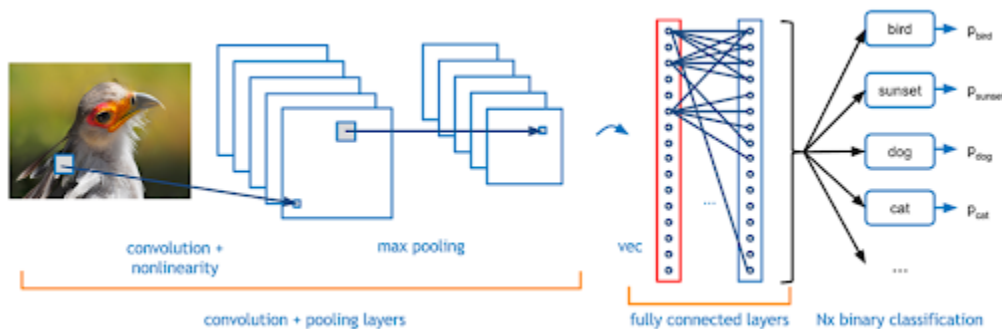


Image recognition with a CNN

8. **Representation and description** is the process of visualizing and describing processed data. AI systems are designed to work as efficiently as possible. The raw output of an AI system looks like an array of numbers and values that represent the information the AI model was trained to produce. Yet for the sake of system performance, a deep neural network usually doesn't include any output data representations. Using special visualization tools, you can turn these arrays of numbers into readable images suitable for further analysis.

OBJECT RECOGNITION

Object recognition is a computer vision technique for identifying objects in images or videos. Object recognition is a key output of deep learning and machine learning algorithms. When humans look at a photograph or watch a video, we can readily spot people, objects, scenes, and visual details. The goal is to teach a computer to do what comes naturally to humans: to gain a level of understanding of what an image contains.

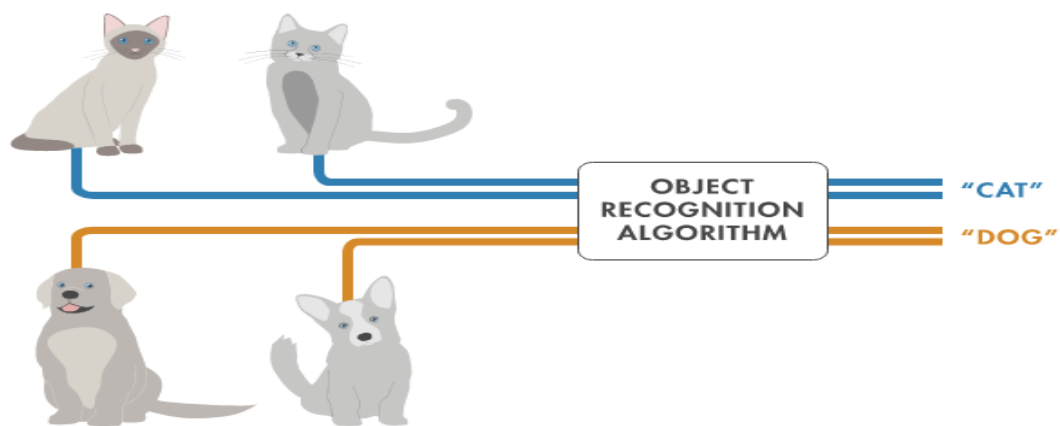


Figure 1. Using object recognition to identify different categories of objects.

Object recognition is a key technology behind driverless cars, enabling them to recognize a stop sign or to distinguish a pedestrian from a lamppost. It is also useful in a variety of applications such as disease identification in bioimaging, industrial inspection, and robotic vision.

Object Recognition vs. Object Detection

Object detection and object recognition are similar techniques for identifying objects, but they vary in their execution. Object detection is the process of finding instances of objects in images. In the case of deep learning, object detection is a subset of object recognition, where the object is not only identified but also located in an image. This allows for multiple objects to be identified and located within the same image.

How Object Recognition Works

You can use a variety of approaches for object recognition. Recently, techniques in machine learning and deep learning have become popular approaches to object recognition problems. Both techniques learn to identify objects in images, but they differ in their execution.

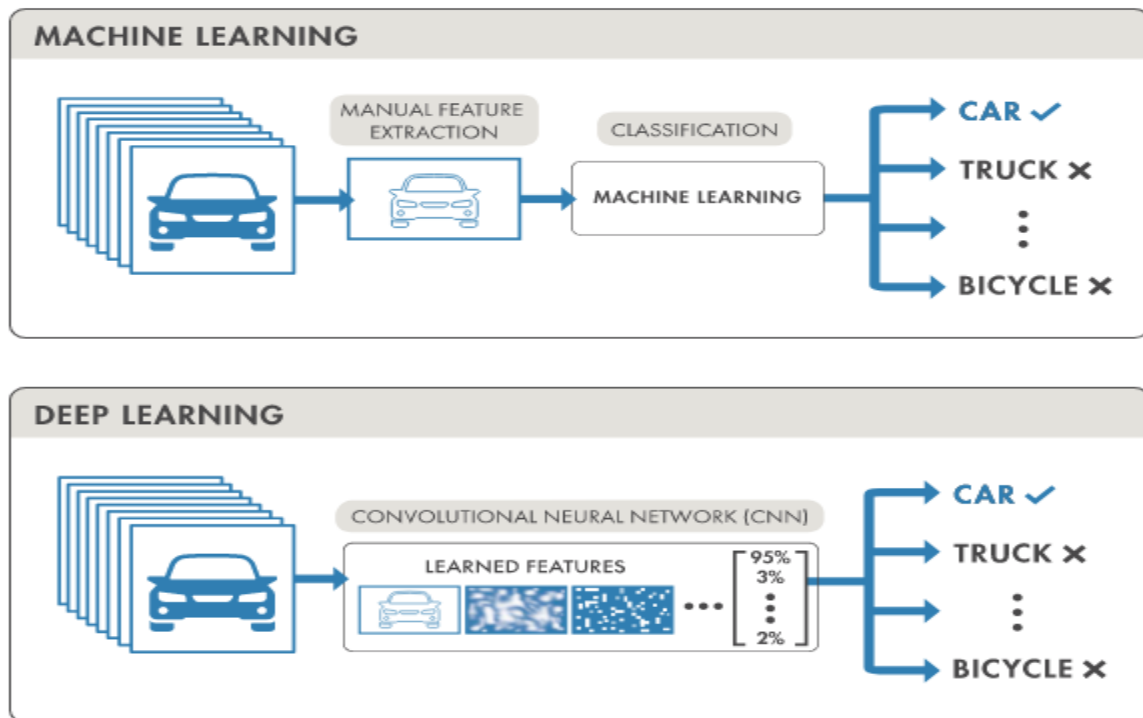


Figure 3: Machine learning and deep learning techniques for object recognition.

OBJECT RECOGNITION TECHNIQUES

1.Template matching

Template matching is a technique for finding small parts of an image which match a template image. It is a straightforward process. In this technique template images for different objects are stored. When an image is given as input to the system, it is matched with the stored template images to determine the object in the input image. Templates are frequently used for recognition of characters, numbers, objects, etc. It can be performed on either color or gray level images. Template matching can either be pixel to pixel matching or feature based. In feature based the features of template image is compared to features of sub-images of the given input image; to determine if the template object is present in the input image.

B. Color based

Color provides potent information for object recognition. A simple and efficient object detection scheme is to represent and match images on the basis of color histograms.

The color information is extended in two existing methods for object detection, the part-based detection framework and the Efficient Subwindow Search approach . The three main criteria which should be taken into account when choosing an approach to integrating color into object detection are feature Combination, photometric invariance and compactness.

variety of color models used for recognition of multicolored objects according to the following criteria: 1. Robustness to a change in viewing direction

2. Robustness to a change in object geometry

3. Robustness to a change in the direction of the illumination

4. Robustness to a change in the intensity of the illumination

5. Robustness to a change in the spectral power distribution (SPD) of the illumination.

The color models have High discriminative power; robustness to object occlusion and cluttering; robustness to noise in the images.

3.Active and Passive

Object detection in passive manner does not involve local image samples extracted during scanning. Two main object-detection approaches that employ passive scanning:

1. The window-sliding approach:

It uses passive scanning to check if the object is present or not at all locations of an evenly spaced grid. This approach extracts a local sample at each grid point and classifies it either as an object or as a part of the background

2.The part-based approach:

It uses passive scanning to determine interest points in an image. This approach calculates an interest value for local samples at all points of an evenly spaced grid. At the interest points, the approach extracts new local samples that are evaluated as belonging to the object or the background

Some methods try to bound the region of the image in which passive scanning is applied. It is a computationally expensive and inefficient scanning method. In this method at each sampling point costly feature extraction is performed, while the probability of detecting an object or suitable interest point can be squat In active scanning local samples are used to guide the scanning process. At the current scanning position a local image sample is extracted and mapped to a shifting vector indicating the next scanning position. The method takes successive samples towards the expected object location, while skipping regions unlikely to contain the object. The goal of active scanning is to save computational effort, while retaining a good detection performance.

The active object-detection method (AOD-method) scans the image for multiple discrete time steps in order to find an object. In the AOD-method this process consists of three phases:

- 1.Scanning for likely object locations on a coarse scale
2. Refining the scanning position on a fine scale
- 3.Verifying object presence at the last scanning position with a standard object detector.

4.Shape based

Recently, shape features have been extensively explored to detect objects in real-world images. The shape features are more striking as compared to local features like SIFT because most object categories are better described by their shape then texture, such as cows, horses and cups and also for wiry objects like bikes, chair or ladders, local features unavoidably contain large amount of background mess. Thus shape features are often used as a replacement or complement to local features

Berg, et.al. have proposed a new algorithm to find correspondences between feature points for object recognition in the framework of deformable shape matching. The basic subroutine in deformable shape matching takes as input an image with an unknown object (shape) and compares it to a model by solving the correspondence problem between the model and the object. Then it performs aligning transformation and computes a similarity based on both the aligning transform and the residual after applying the aligning transformation

5. Local and global features

The most common approach to generic object detection is to slide a window across the image and to classify each such local window as containing the target or background. This approach has been successfully used to detect rigid objects such as faces and cars in and In, a method of object recognition and segmentation using Scale-Invariant Feature Transform (SIFT) and Graph Cuts is presented. SIFT feature is invariant for rotations, scale changes, and illumination changes. By combining SIFT and Graph Cuts, the existence of objects is recognized first by vote processing of SIFT keypoints. Then the object region is cut out by Graph Cuts using SIFT keypoints as seeds. Both recognition and segmentation are performed automatically under cluttered backgrounds including occlusion.

Object Recognition Using Deep Learning

Deep learning techniques have become a popular method for doing object recognition. Deep learning models such as convolutional neural networks, or CNNs, are used to automatically learn an object's inherent features in order to identify that object. For example, a CNN can learn to identify differences between cats and dogs by analyzing thousands of training images and learning the features that make cats and dogs different. There are two approaches to performing object recognition using deep learning:

- **Training a model from scratch:** To train a deep network from scratch, you gather a very large labeled dataset and design a network architecture that will learn the features and build the model. The results can be impressive, but this approach requires a large amount of training data, and you need to set up the layers and weights in the CNN.
- **Using a pretrained deep learning model:** Most deep learning applications use the transfer learning approach, a process that involves fine-tuning a pretrained model. You start with an existing network, such as AlexNet or GoogLeNet, and feed in new data containing previously unknown classes. This method is less time-consuming and can provide a faster outcome because the model has already been trained on thousands or millions of images.

Deep learning offers a high level of accuracy but requires a large amount of data to make accurate predictions.

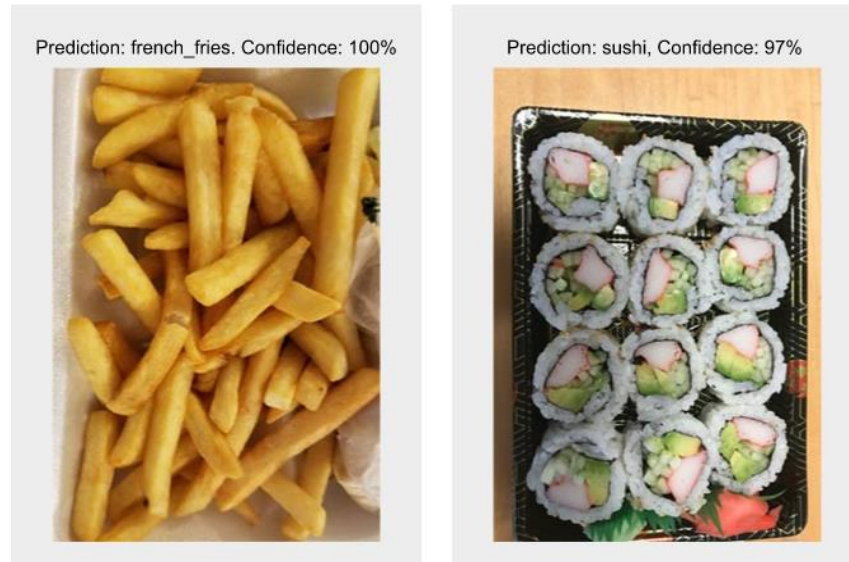


Figure 4: Deep learning application showing object recognition of restaurant food.

Machine Learning Workflow

To perform object recognition using a standard machine learning approach, you start with a collection of images (or video), and select the relevant features in each image. For example, a feature extraction algorithm might extract edge or corner features that can be used to differentiate between classes in your data.

These features are added to a machine learning model, which will separate these features into their distinct categories, and then use this information when analyzing and classifying new objects.

You can use a variety of machine learning algorithms and feature extraction methods, which offer many combinations to create an accurate object recognition model.

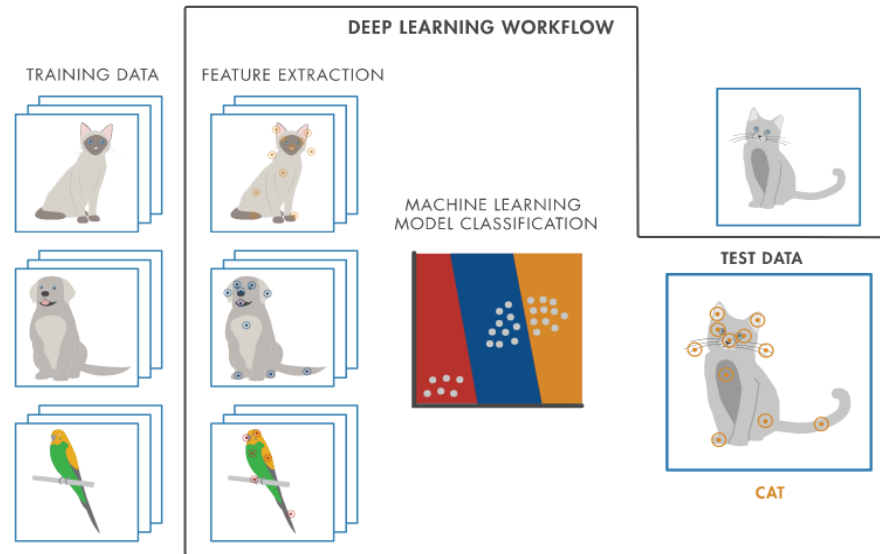


Figure 5: Machine learning workflow for object recognition.

OBJECT RECOGNITION BY APPEARANCE

Appearance means what an object tends to look like. For example, football is rather round in shape. It is important to know every class of images with a classifier. Taking the example of faces, looking at the camera-every face looks similar under good light and perfect resolution. A strategy called sliding window includes computing features for an object and present it to a classifier. One strategy is to estimate and correct the illumination in each image window and another is to build features out of gradient orientations. To find faces of different sizes, repeat the sweep over larger or smaller versions of the image. Then, we post process the responses across scales and location to produce the final set of detections. Postprocessing have several overlapping windows that each report a match for a face. To yield a single high quality match, we can combine these partial overlapping matches at nearby locations. Therefore, it gives a face detector that can search over locations and scales.

1. Complex appearance and pattern elements

Since, several effects can move features around in an image of the object, many objects produce much more complex patterns than faces do. Effects include: **Foreshortening**: which causes a pattern viewed at a slant to be significantly distorted **Aspect**: which causes objects to look different when seen from different directions. **Occlusion**: when some parts are hidden from some viewing directions. **Self-occlusion** can be defined as objects can occlude one another or parts of an object can occlude other parts. **Deformation**: where internal degrees of freedom of the object change its appearance. An object recognizer is then a collection of features that can tell whether the pattern elements are present, and whether they are in about the right place. With a histogram

of the pattern elements that appear can be considered as the most obvious approach to represent the image window. This approach does not work particularly well, because too many patterns get confused with one another.

2. Pedestrian detection with HOG features

Each year car accidents kill about 1.2 million people, and to avoid this problem cars should be provided with sensors which detect pedestrians and result in saving many lives. The most usual cases are lateral or frontal views of a walk. In these cases, we see either a “lollipop” shape-the torso is wider than the legs, which are together in the stance phase of the walk-or a “scissor” shape-where the legs are swinging in the walk. Therefore, we need to build a useful movingwindow pedestrian detector. To represent the image window, it is better to use orientations than edges because there isn’t always a strong contrast between a pedestrian and the background. Pedestrians can move their arms and legs around, so we should use a histogram to suppress some spatial detail in the feature. When breaking up the window into cells, overlapping occurs and hence, build an orientation histogram in each cell. Through this feature, we can determine whether head-and-shoulders curve is at the top of the window or at the bottom, but will not change, if the head is moved bit slightly.

RECONSTRUCTING THE 3D WORLD

This section shows how to go from 2D image to a 3D representation of the scene. A fundamental question arises which is how do we recover 3D information when given all the points in the scene that fall along a ray to pinhole are projected to the same point in the image? The two solutions are, If we have two or more images from different camera positions, then we can triangulate to find the position of a point in the scene. We can exploit background knowledge about the physical scene that gave rise to the image. Given an object model $P(\text{scene})$ and a rendering model $P(\text{image} | \text{scene})$. We can compute a posterior distribution $P(\text{scene} | \text{image})$. For a scene reconstruction, we survey 8 commonly used visual cues because there is no single unified theory for it. Motion Parallax: We state an equation understanding the concept of relation among the optical flow velocity and depth in the scene.

$$v_x(x, y) = \frac{-T_x + xT_z}{Z(x, y)}, \quad v_y(x, y) = \frac{-T_y + yT_z}{Z(x, y)},$$

The point in the scene corresponding to the point in the image at (x,y) . Both components of the optical flow, $v_x(x,y)$ and $v_y(x,y)$, are zero at the point $x=T_x/T_z$, $y=T_y/T_z$. This point is called focus of expansion of the flow field. If we change the origin in the x - y plane to lie at the focus of expansion, then the expressions for optical flow take on a particularly simple form. Let (x',y') be the new coordinates defined by $x'=x-T_x/T_z$, $y'=y-T_y/T_z$. Then,

$$v_x(x',y') = \frac{x'T_z}{Z(x',y')}, \quad v_y(x',y') = \frac{y'T_z}{Z(x',y')}.$$

Binocular Stereopsis

This idea is similar to motion parallax, but we use two or more images separated in space. Binocular stereopsis enables a wider field of vision for the predators who have eyes in the front. If we super pose the two images there will be a disparity in the location of the image feature in the two images as a given feature in the scene will be in a different place relative to the Z -axis of each image plane. Using optical flow equations vector T acting for time δt , with $T_x=b/\delta t$ and $T_y=T_z=0$. Horizontal disparity is equal to the ratio of base line to the depth, and vertical disparity is zero i.e., $H=b/Z$, $V=0$. Humans fixate at a point in the scene at which the optical axes of the two eyes intersect under normal viewing conditions. The actual disparity is $\delta\theta$, we have disparity $=b\delta Z/Z^2$. In humans, b (the base line distance between the eyes) is about 6cm. So for $Z=30$ cm we get small value $\delta Z=0.036$ mm which means at a distance of 30cm humans can differentiate the depths which differ by a little length as 0.036mm.

Multiple views

Most of the techniques that have been developed had make the use of the information available in multiple views, even from hundreds or thousands of cameras. There are few problems in multiple views which can be solved algorithmically: Correspondence problem: In the 3D world, identifying features in the different images that are projections of the same feature. Relative orientation problem: Finding out the transformation between the coordinate systems fixed to the different cameras. Depth estimation problem: Finding out the depths of various points in the world for which image plane projections were available in at least two views.

Texture

Texture is used to estimate distances and for segmenting objects. The texture elements are also known as texels.

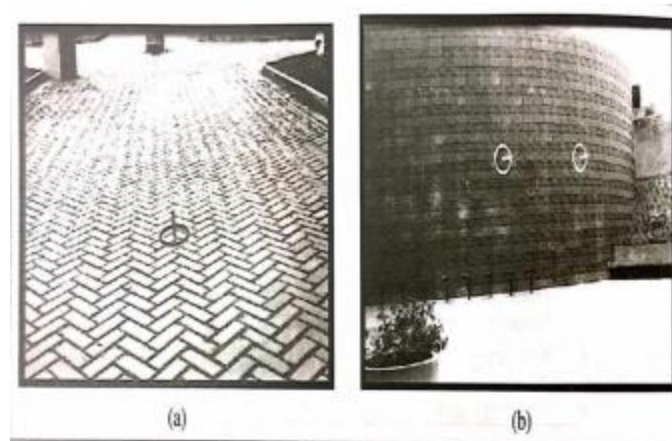


Fig.2. Texture

As shown in the above figure, the paving tiles are identical in the scene but appear different in the image for 2 reasons. Differences in the distances of the texels from the camera. Differences in the foreshortening of the texels. Various algorithms have been developed to make use of variation in the appearance of the projected texels as a basis for finding out the surface normals but they were not accurate as much as the algorithms which were used for multiple views.

Mathematical description of reconstruction

Given a group of 3D points viewed by N cameras with matrices $\{P^i\}_{i=1, \dots, N}$, define $m_j^i \simeq P^i w_j$, to be the homogeneous coordinates of the projection of the j th point onto the i th camera. The reconstruction problem can be changed to: given the group of pixel coordinates $\{m_j^i\}$, find the corresponding set of camera matrices $\{P^i\}$ and the scene structure $\{w_j\}$ such that

$$m_j^i \simeq P^i w_j \quad (1)$$

Generally, without further restrictions, we will obtain a projective

reconstruction.^{[4][5]} If $\{P^i\}$ and $\{w_j\}$ satisfy (1), $\{P^i T\}$ and $\{T^{-1} w_j\}$ will satisfy (1) with any 4×4 nonsingular matrix T .

A projective reconstruction can be calculated by correspondence of points only without any a priori information

OBJECT RECOGNITION FROM STRUCTURAL INFORMATION

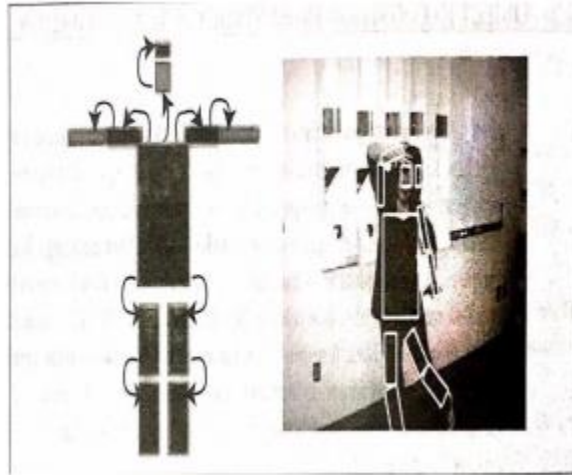
Human body parts are very small in images and vary in color, texture among individuals and hence it is difficult to detect them using moving window method. Some parts are very small to a size of two to three pixels wide. As the layout of body describes the movement, it is important to conclude layout of body in the images. whether the configuration are acceptable or not it can be described by a model called deformable template. The simplest deformable template model of a person connects lower arms to upper arms, upper arms to torso, and so on.

1. The geometry of bodies: finding arms and legs

A tree of eleven segments is used to model the geometry of body. Segments are rectangular in shape.” Cardboard people” are few models which are used to assume the position and pose(orientation) of human body parts and segments in the images. Evaluation of configuration is done based on 2 criteria: First, an image rectangle should look like its segment. A function ϕ_i describes how well an image rectangle matches a body segment. Function ψ defines how the relations of a pair of rectangle segments meet the expected body segments. Each segment has only one parent because the dependencies between segments form a tree. The parent can be described by a function $\psi_{i,p,a(i)}$.

$$\sum_{i \in \text{segments}} \phi_i(m_i) + \sum_{i \in \text{segments}} \psi_{i,pa(i)}(m_i, m_{pa(i)}).$$

There is an angle between segments mainly for the ankles and knees to be differentiated. If there are M image rectangles having the right torso as O(M6) for a model, then the best allocation of rectangles to segments will get slow. However this can be solved by using various speed-ups which are available for an appropriate choice of ψ . This model is usually known as pictorial structure model.



We generally have to build a model of segment appearances when we don't have an idea of what a person looks like. Appearance model is the description of what a person looks like.

2 .Coherent appearance: tracking people in video

By improving the concept of tracking people in a video we can get further in game interfaces and surveillance systems, but many methods haven't succeeded with the problem because people in the videos tend to move fast and produce large accelerations. The effective methods state the fact that, from frame to frame the appearances change. We assume the video to be a huge collection of pictures we wish to track. The people in the video can be tracked by detecting their body segments. In some cases, the segment detector may generate lots of false positives because the people are appearing against a near fixed background. This problem can be solved using an alternative which is quite in practice, by applying a detector to a fixed body configuration for all the frames. We can know when we found a real person in a video by tuning the detector to low false positive rate.

USING VISION

If vision systems could analyze video and understood what people are doing, we would be able to: design buildings and public places better by collecting and using data about what people do in public; build more accurate, more secure, and less intrusive surveillance systems; build computer sports commentators; and build human-computer interfaces that watch people and react to their behavior

Using vision for controlling movement

One of the principal uses of vision is to provide information both for manipulating objects— picking them up, grasping them, twirling them, and so on—and for navigating while avoiding obstacles. The ability to use vision for these purposes is present in the most primitive of animal visual systems. In many

cases, the visual system is minimal, in the sense that it extracts from the available light field just the information the animal needs to inform its behavior. Quite probably, modern vision systems evolved from early, primitive organisms that used a photosensitive spot at one end to orient themselves toward (or away from) the light.

Let us consider a vision system for an automated vehicle driving on a freeway. The tasks faced by the driver include the following:

1. **Lateral control**—ensure that the vehicle remains securely within its lane or changes lanes smoothly when required.
2. **Longitudinal control**—ensure that there is a safe distance to the vehicle in front.
3. **Obstacle avoidance**—monitor vehicles in neighboring lanes and be prepared for evasive maneuvers if one of them decides to change lanes. The problem for the driver is to generate appropriate steering, acceleration, and braking actions to best accomplish these tasks.

For lateral control, one needs to maintain a representation of the position and orientation of the car relative to the lane. We can use edge-detection algorithms to find edges corresponding to the lane-marker segments. We can then fit smooth curves to these edge elements. The parameters of these curves carry information about the lateral position of the car, the direction it is pointing relative to the lane, and the curvature of the lane. This information, along with information about the dynamics of the car, is all that is needed by the steering-control system. If we have good detailed maps of the road, then the vision system serves to confirm our position (and to watch for obstacles that are not on the map).

For longitudinal control, one needs to know distances to the vehicles in front. This can be accomplished with binocular stereopsis or optical flow. Using these techniques, visioncontrolled cars can now drive reliably at highway speeds.

UNIT-V

ROBOTICS

Robotics

Robotics is the term used in artificial intelligence that deals with a study of creating intelligent and efficient robots.

What are Robots

Robots are multifunctional, re-programmable, automatic industrial machine designed for replacing human in hazardous work.

Robots can be work as:-

- An automatic machine sweeper
- In space
- A machine removing mines in a war field
- An automatic car for a child to play with
- In military, etc.

Objective

The aim of the robot is to manipulate the objects by perceiving, moving, picking, modifying the physical properties of object.

Robotics Definition

Robotics is a branch of Artificial Intelligence (AI), it is mainly composed of electrical engineering, mechanical engineering and computer science engineering for construction, designing and

application of robots.

Robotics is science of building or designing an application of robots. The aim of robotics is to design an efficient robot.

Aspects of Robotics

- The robots have **electrical components** for providing power and control the machinery.
- They have **mechanical construction**, shape, or form designed to accomplish a particular task.
- It contains some type of **computer program** that determines what, when and how a robot does something.

Robotics History

The word robot was firstly introduced to public by Czech writer Karel Capek in his play Rossum's Universal Robots (R.U.R), published in 1920. The play begins with a factory that makes artificial people known as robots.

The word "Robotics", was coined accidentally by the Russian-born, American scientist, Issac Asimov in 1940s .

The three laws of Robotics:

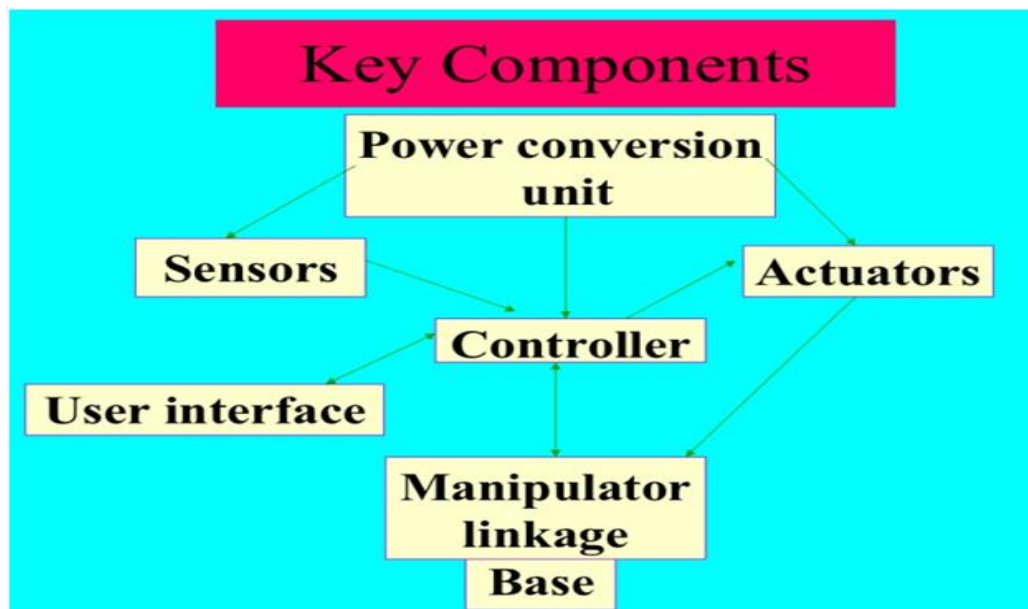
Issac Asimov also proposed his three "Laws of Robotics", and he later added a "zeroth law"

- **Zeroth Law** - A robot is not allowed to injured humanity, or, through inaction it allows humanity to come to harm.
- **First Law** - A robot can not injure a human being, or, through inaction it allows a human being to come to harm, unless it would violate the higher order law.
- **Second Law** - A robot should follow the orders given it by human beings, except when such orders give by humans would conflict with a higher order law.
- **Third Law** - A robot is allowed to protect its own existence as long as such protection would not conflict with a higher order law.

The first industrial robot: UNIMATE

In 1954 first programmable robot is designed by George Devol, who coins the term Universal Automation. He later shortens this term to Unimation, which become the name of the first robot company in 1962.

ROBOT HARDWARE



- **Sensors** - Sensors provide real time information on the task environment. Robots are equipped with tactile sensor it imitates the mechanical properties of touch receptors of human fingerprints and a vision sensor is used for computing the depth in the environment.

Passive sensors, such as cameras, are true observers of the environment: they capture signals that are generated by other sources in the environment.

Active sensors, such as sonar, send energy into the environment

Range finders are sensors that measure the distance to nearby objects.

- A second important class of sensors is location sensors. Most location sensors use range sensing as a primary component to determine location. Outdoors, the Global Positioning System (GPS) is the most common solution to the localization problem. GPS measures GLOBAL POSITIONING SYSTEM the distance to satellites that emit pulsed signals
- The third important class is proprioceptive sensors, which inform the robot of its own motion. To measure the exact configuration of a robotic joint, motors are often equipped with shaft decoders that count the revolution of motors in small increments. On robot arms, shaft decoders can provide accurate information over any period of time.
- **Actuators** - Actuators are the energy conversion device used inside a robot. The major function of actuators is to convert energy into movement.

ROBOTIC PERCEPTION

Robotic perception is crucial for a robot to **make decisions, plan**, and operate in real-world environments, by means of numerous functionalities and operations from occupancy grid mapping to object detection. Sensor-based environment representation/mapping is a very important part of a robotic perception system.

Robot-Planning to Move

Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.

For example, consider a mobile robot navigating inside a building to a distant waypoint. It should execute this task while avoiding walls and not falling down stairs. A motion planning algorithm would take a description of these tasks as input, and produce the speed and turning commands sent to the robot's wheels. Motion planning algorithms might address robots with a larger number of joints (e.g., industrial manipulators), more complex tasks (e.g. manipulation of objects),

different constraints (e.g., a car that can only drive forward), and uncertainty (e.g. imperfect models of the environment or robot).

As another example, consider navigating one or more unmanned aircraft or drones in a cluttered indoor or outdoor environment. The same motion planning strategies can still be applied in a 3D space with environmental disturbances, such as wind and bad weather.

Motion planning has several robotics applications, such as autonomy, automation, and robot design in CAD software, as well as applications in other fields, such as animating digital characters, video game, artificial intelligence, architectural design, robotic surgery, and the study of biological molecules.

PLANNING UNCERTAIN MOVEMENTS

In robotics uncertainty exists at both planning and execution time. Effective planning must make sure that enough information becomes available to the sensors during execution, to allow the robot to correctly identify the states it traverses. It requires selecting a set of states, associating a motion command with every state, and synthesizing functions to recognize state achievement

Motion planning with uncertainty is a critical problem in robotics. Indeed, even the most complex models of the physical world cannot be perfectly accurate. All details omitted from these models unite to form uncertainty. A robot planner must produce plans that reliably achieve their goals despite errors, i.e., differences between the models and the real world

APPLICATION DOMAINS

INSPECTION

In the Inspection program the use of robotics technologies is investigated for tasks in which inspection of not easily reachable structures is necessary. This can be the case for underground gas mains like in the project Pirate, high structures or plains like in the flying robot project Airobots, continuous monitoring of the dutch dikes like in the project Rose or even modeling for inspection of space rovers in collaboration with ESA for the Exomars. For the latter the modeling of altitude

maps and their use in the prediction of the navigation of the ExoMars rover is considered. Problems, which arise in this study relate a combination of sparse altitude data with precise terramechanics simulations. This is tackled using a hybrid use of differential geometric techniques and computational geometric methods.

MEDICAL

In the Medical program developments are taking place which have directly or indirectly to do with the human body. The program is composed of two parts: 1. Robotic Surgery in which new robotic instruments and methodologies are studied with projects like Teleflex and Miriam. 2. Prosthetics is related to the development or artificial limbs like transradial hand prosthesis (MyoPro) and transfemoral leg prosthesis.

HOME & CARE

The Home & Care program deals with development of methods and techniques to design service robots. These robots have to act in an unstructured environment, and have regularly and deliberately physical contact with its environment. Contexts in which these robots have to act are normal (household) situation and care taking situations. Topics of study are robot architectures that facilitate resilient, robust and fault-tolerant behaviour of the autonomous service robot.

ROBOTIC SOFTWARE ARCHITECTURES

A methodology for structuring algorithms is called a software architecture. An architecture software architecture includes languages and tools for writing programs, as well as an overall philosophy for how programs can be brought together. Architectures that combine reactive and deliberate techniques are called hybrid architectures.

Subsumption architecture

Rodney Brook's subsumption architecture addresses this:

- Different commands have different levels

- Higher level commands subsume lower level ones
- They may even inhibit lower level ones
- If a high level command is missing, the robot can still operate, though at a reduced capacity.

Here is an example:

Suppose a robot is to explore its environment by:

- following the lights it sees,
- avoiding loud noises and
- avoiding other obstacles.

It will have sensors for lights, noises and obstacles It will have behaviour for each of these sensors It will have an overarching ‘explore’ behaviour All of these will combine to drive the motors.

THREE-LAYER ARCHITECTURE

Hybrid architectures combine reaction with deliberation. The most popular hybrid architecture is the three-layer architecture, which consists of a reactive layer, an executive layer, THREE-LAYER ARCHITECTURE and a deliberative layer.

REACTIVE LAYER

The reactive layer provides low-level control to the robot. It is characterized by a tight sensor–action loop. Its decision cycle is often on the order of milliseconds.

EXECUTIVE LAYER

The executive layer (or sequencing layer) serves as the glue between the reactive layer and the deliberative layer. It accepts directives by the deliberative layer, and sequences them for the reactive layer. For example, the executive layer might handle a set of via-points generated by a deliberative path planner, and make decisions as to which reactive behavior to invoke. Decision cycles at the executive layer are usually in the order of a second. The executive layer is also responsible for integrating sensor information into an internal state representation. For example, it may host the robot’s localization and online mapping routines.

DELIBERATIVE LAYER

The deliberative layer generates global solutions to complex tasks using planning. Because of the computational complexity involved in generating such solutions, its decision cycle is often in the order of minutes. The deliberative layer (or planning layer) uses models for decision making. Those models might be either learned from data or supplied and may utilize state information gathered at the executive layer

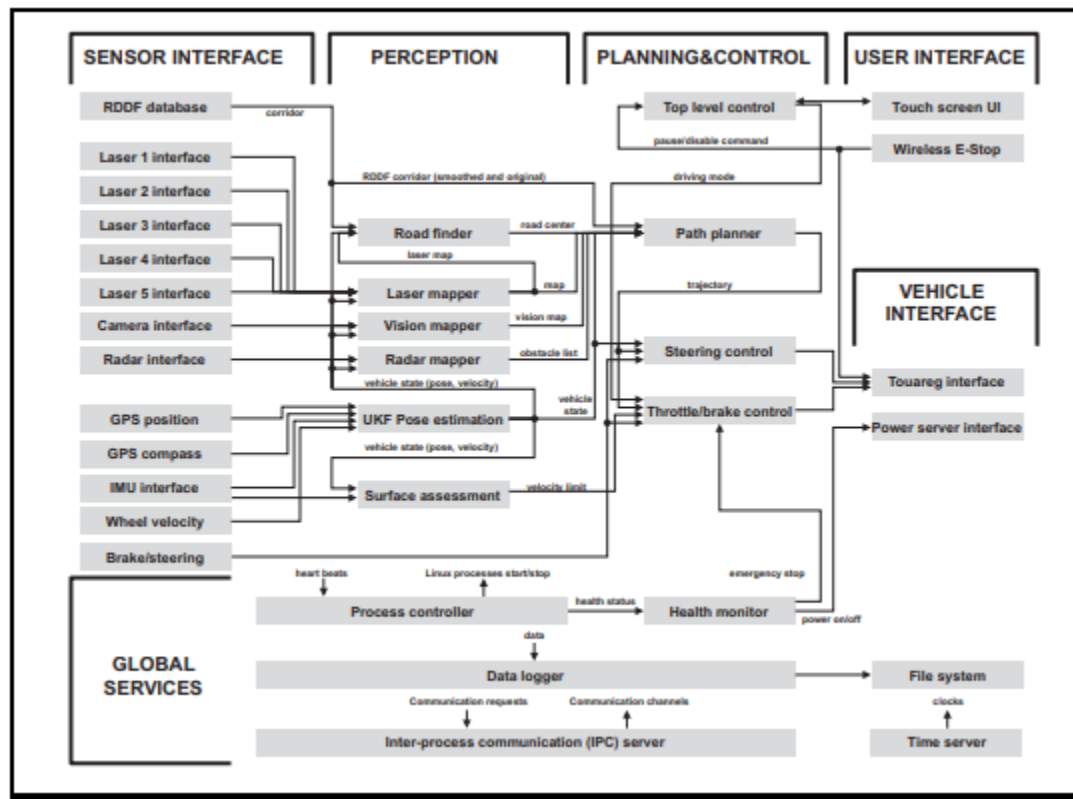


FIG:SOFTWARE ARCHITECTURE IN ROBOTIC CAR

PIPELINE ARCHITECTURE

Just like the subsump- PIPELINE ARCHITECTURE tion architecture, the pipeline architecture executes multiple process in parallel. However, the specific modules in this architecture resemble those in the three-layer architecture.

Data enters this pipeline at the sensor interface layer. The perception layer then updates the robot's internal models of the environment based on this data. Next,

these models are handed to the planning and control layer, which adjusts the robot's internal.

PLANNING AND CONTROL LAYER

Plans turns them into actual controls for the robot. Those are then communicated back to the vehicle through the vehicle interface layer.

VEHICLE INTERFACE LAYER

The key to the pipeline architecture is that this all happens in parallel. While the perception layer processes the most recent sensor data, the control layer bases its choices on slightly older data. In this way, the pipeline architecture is similar to the human brain. We don't switch off our motion controllers when we digest new sensor data. Instead, we perceive, plan, and act all at the same time. Processes in the pipeline architecture run asynchronously, and all computation is data-driven. The resulting system is robust, and it is fast.

PHILOSOPHICAL FOUNDATIONS

Strong AI	Weak AI
A computer machine gets capable of thinking atleast equal to human beings.	A computer machine gets a 'thinking' like feature to make it more powerful.
A machine can perform tasks on its own, just like human beings.	A machine can perform tasks but need human intervention.
A computer program adds an algorithm itself to act in different situations.	Here, tasks to be performed are added manually.
This type of AI allows machines to solve problems in an unlimited domain.	This type of AI allows machines to solve problems in a limited domain.

It is an independent AI which can take the decision on its own.	It is dependent on humans and can simulate like human beings.
Currently, strong AI does not exist in the real world.	Weak AI is in its advance phase of working.
There is no proper example of it.	Example: Automatic car, APPLE Siri, etc.

Strong artificial intelligence (AI), also known as artificial general intelligence (AGI) or general AI, is a theoretical form of AI used to describe a certain mindset of AI development. If researchers are able to develop Strong AI, the machine would require an intelligence equal to humans; it would have a self-aware consciousness that has the ability to solve problems, learn, and plan for the future.

Strong AI aims to create intelligent machines that are indistinguishable from the human mind. But just like a child, the AI machine would have to learn through input and experiences, constantly progressing and advancing its abilities over time.

Ethics of AI: Benefits and risks of artificial intelligence

Ethics in AI is essentially questioning, constantly investigating, and never taking for granted the technologies that are being rapidly imposed upon human life.

That questioning is made all the more urgent because of *scale*. AI systems are reaching tremendous size in terms of the compute power they require, and the data they consume. And their prevalence in society, both in the scale of their deployment and the level of responsibility they assume, dwarfs the presence of computing in the PC and Internet eras. At the same time, increasing scale means many aspects of the technology, especially in its deep learning form, escape the comprehension of even the most experienced practitioners.

Ethical concerns range from the esoteric, such as who is the author of an AI-created work of art; to the very real and very disturbing matter of surveillance in the hands of military authorities who can use the tools with impunity to capture and kill their fellow citizens.

Why are AI ethics important?

AI is a technology designed by humans to replicate, augment or replace human intelligence. These tools typically rely on large volumes of various types of data to develop insights. Poorly designed projects built on data that is faulty, inadequate or biased can have unintended, potentially harmful, consequences. Moreover, the rapid advancement in algorithmic systems means that in some cases it is not clear to us how the AI reached its conclusions, so we are essentially relying on systems we can't explain to make decisions that could affect society.

An AI ethics framework is important because it shines a light on the risks and benefits of AI tools and establishes guidelines for its responsible use. Coming up with a system of moral tenets and techniques for using AI responsibly requires the industry and interested parties to examine major social issues and ultimately the question of what makes us human.

What are the ethical challenges of AI?

- **Explainability.** When AI systems go awry, teams need to be able to trace through a complex chain of algorithmic systems and data processes to find out why. Organizations using AI should be able to explain the source data, resulting data, what their algorithms do and why they are doing that. "AI needs to have a strong degree of traceability to ensure that if harms arise, they can be traced back to the cause," said Adam Wisniewski, CTO and co-founder of AI Clearing.
- **Responsibility.** Society is still sorting out responsibility when decisions made by AI systems have catastrophic consequences, including loss of capital, health or life. Responsibility for the consequences of AI-based decisions needs to be

sorted out in a process that includes lawyers, regulators and citizens. One challenge is finding the appropriate balance in cases where an AI system may be safer than the human activity it is duplicating but still causes problems, such as weighing the merits of autonomous driving systems that cause fatalities but far fewer than people do.

- **Fairness.** In data sets involving personally identifiable information, it is extremely important to ensure that there are no biases in terms of race, gender or ethnicity.
- **Misuse.** AI algorithms may be used for purposes other than those for which they were created. Wisniewski said these scenarios should be analyzed at the design stage to minimize the risks and introduce safety measures to reduce the adverse effects in such cases.

What is an AI code of ethics?

- **Policy.** This includes developing the appropriate framework for driving standardization and establishing regulations. Efforts like the Asilomar AI Principles are essential to start the conversation, and there are several efforts spinning up around policy in Europe, the U.S. and elsewhere. Ethical AI policies also need to address how to deal with legal issues when something goes wrong. Companies may incorporate AI policies into their own code of conduct. But effectiveness will depend on employees following the rules, which may not always be realistic when money or prestige are on the line.
- **Education.** Executives, data scientists, front-line employees and consumers all need to understand policies, key considerations and potential negative impacts of unethical AI and fake data. One big concern is the tradeoff between ease of use around data sharing and AI automation and the potential negative repercussions of oversharing or adverse automations. "Ultimately, consumers' willingness to proactively take control of their data and pay attention to potential threats enabled by AI is a complex equation based on a combination of instant gratification, value, perception and risk," Shepherd said.

- **Technology.** Executives also need to architect AI systems to automatically detect fake data and unethical behavior. This requires not just looking at a company's own AI but vetting suppliers and partners for the malicious use of AI. Examples include the deployment of deep fake videos and text to undermine a competitor, or the use of AI to launch sophisticated cyberattacks. This will become more of an issue as AI tools become commoditized. To combat this potential snowball effect, organizations need to invest in defensive measures rooted in open, transparent and trusted AI infrastructure. Shepherd believes this will give rise to the adoption of trust fabrics that provide a system-level approach to automating privacy assurance, ensuring data confidence and detecting unethical use of AI.

RISKS OF AI

AI systems: Bias and discrimination

AI systems designers choose the features, metrics, and analytics structures of the models that enable data mining. Thus, data-driven technologies, such as Artificial Intelligence, can potentially replicate the preconceptions and biases of their designer.

Data samples train and test algorithmic systems. Yet, they can often be insufficiently representative of the populations from which they are drawing inferences; thus, creating possibilities of biased and discriminatory outcomes due to a flaw from the start when the designer feeds the data into the systems.

AI systems: Denial of individual autonomy, recourse, and rights

In the past, AI systems that automate cognitive functions were attributable exclusively to accountable human agents. Today, AI systems make decisions, predictions, and classifications that affect citizens.

Certain situations may arise where such individuals are unable to hold accountable the parties responsible for the outcomes. One of the most common responses from

humans to justify negative results is to blame the AI system, adding that there is nothing they can do to change the outcome. Something which is not real.

Such a response is utterly ridiculous since AI systems are designed and programmed by a human designer. Therefore, a human is who can correct and change an outcome that is not satisfactory. Take as an example a case of injuries, or a negative consequence such an accountability gap, which may harm the autonomy and violate the rights of the affected individuals.

AI systems: Non-transparent, unexplainable, or unjustifiable outcomes

In some cases, machine learning models may generate their results by operating on high-dimensional correlations that are beyond the interpretive capabilities of human reasoning.

These are cases in which the rationale of algorithmically produced outcomes that directly affect decision subjects may remain opaque to those subjects. In some use cases, this lack of explainability may not be a cause of too much trouble.

However, in applications where the processed data could harbor traces of discrimination, bias, inequity, or unfairness, the lack of clarity of the model may be deeply problematic.

AI systems: Invasions of privacy

AI systems pose threats to privacy in two ways:

- As a result of their design and development processes
- As a result of their deployment

AI projects lay on the grounds of the structuring and processing of big data. Massive amounts of personal data are collected, processed, and utilized to develop AI technologies. More often than not, big data is captured and extracted without gaining the proper consent of the data owner subject. Quite often, some use of big data reveals—or places under risk—personal information, compromising the privacy of the individual.

The deployment of AI systems can target, profile, or nudge data owner subjects without their knowledge or consent. It means that such AI systems are infringing

upon the ability of the individuals to lead a private life. Privacy invasion can consequently harm the right to pursue goals or life plans free from unchosen influence.

AI systems: Isolation and disintegration of social connection

The capacity of AI systems to curate individual experiences and to personalize digital services has the potential of improving consumer life and service delivery. This, which is a benefit if done right, yet it comes with potential risks.

Such risks may not be visible or show as risks at the start. However, excessive automation may potentially lead to the reduction of human-to-human interaction, and with it, solving problematic situations at an individual level could not be possible any longer.

Algorithmically enabled hyper-personalization might improve customer satisfaction, but limits our exposure to worldviews different from ours, and this might polarize social relationships.

Since the times of Greek philosopher Plato, well-ordered and cohesive societies have built on relations of human trust, empathy, and mutual understanding. As Artificial Intelligence technologies become more prevalent, it is paramount that these relations of human trust, or empathy, or mutual understanding remain intact.

AI systems: Unreliable, unsafe, or poor-quality outcomes

The implementation and distribution of AI systems that produce unreliable, unsafe, or poor-quality outcomes may be the result of irresponsible data management, negligent design production processes, or questionable deployment practices. Consequently, this can directly lead to damaging the wellbeing of individuals as well as damaging the public welfare.

Such outcomes can also undermine public trust in the responsible use of societally beneficial AI technologies. Furthermore, they can create harmful inefficiencies by the dedication of limited resources to inefficient or even detrimental AI technologies.

How can Artificial Intelligence be risky?

Most of the researchers agree that super AI cannot show human emotions such as **Love, hate or kindness**. Moreover, we should not expect an AI to become intentionally generous or spiteful. Further, if we talk about AI to be risky, there can be mainly two scenarios, which are:

1. AI is programmed to do something destructive:

Autonomous weapons are artificial intelligence systems that are programmed to kill. In the hands of the wrong person, these weapons could easily cause mass casualties. Moreover, an AI arms race could inadvertently lead to an AI war resulting in mass casualties. To avoid being dissatisfied with the enemy, these weapons would be designed to be extremely difficult to "turn off," so humans could plausibly lose control of such a situation. This risk is present even with narrow AI but grows as levels of AI intelligence and autonomy increase.

2. Misalignment between our goals and machines:

The second possibility of AI as a risky technology is that if intelligent AI is designed to do something beneficial, it develops destructive results. For example, Suppose we ask the self-driving car to "take us at our destination as fast as possible." The machine will immediately follow our instructions. It may be dangerous for human lives until we specify that traffic rules should also be followed and we value human life. It may break traffic rules or meet with an accident, which was not really what we wanted, but it did what we have asked to it. So, super-intelligent machines can be destructive if they ask to accomplish a goal that doesn't meet our requirements.

Agent Components

A learning agent in AI is the type of agent that can learn from its past experiences or it has learning capabilities. It starts to act with basic knowledge and then is able to act and adapt automatically through learning.

A learning agent has mainly four conceptual components

1. **Learning element:** It is responsible for making improvements by learning from the environment
2. **Critic:** The learning element takes feedback from critics which describes 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.

AGENT ARCHITECTURE

Agent architecture in computer science is a blueprint for software agents and intelligent control systems, depicting the arrangement of components. The architectures implemented by intelligent agents are referred to as cognitive architectures.

Architecture: This refers to machinery or devices that consists of actuators and sensors. The intelligent agent executes on this machinery. Examples include a personal computer, a car, or a camera.

- **Sensors:** These are devices that detect any changes in the environment. This information is sent to other devices. In artificial intelligence, the environment of the system is observed by intelligent agents through sensors.

Sensors are used in everyday life like touch sensors in your phone to smoke detectors in our homes, we are completely surrounded by sensors.

Accelerometer

Whenever you use a compass application on your phone, your phone somehow happens to know the direction your phone is pointing. Also, for a VR Headset, you move your head and you happen to be moving in the application too, ever wondered how is your phone able to get this kind of information. Accelerometers are the key to getting this information.

Ultrasonic Sensor (SONAR)

You all have an idea about the working of a SONAR, an ultrasonic sensor works on a similar principle. On receiving a signal from a microcontroller, your sensor emits an ultrasonic wave ($f > 20\text{KHz}$) and detects an echo after some time. Based on this time interval, it sends a voltage signal to the microcontroller encoding the information of the distance of the object from the sensor.

- **Actuators:** These are components through which energy is converted into motion. They perform the role of controlling and moving a system. Examples include rails, motors, and gears.

Actuators range from hydraulic, pneumatic to even thermal/ magnetic actuators electrical actuators i.e. motors.

Motors

Electric motors operate with the interaction of winding currents with magnetic fields to generate a rotating torque (or motion) thus converting electrical energy into mechanical energy. There are several types of motors some of which are listed below.

DC Motors

Let's take an example of an RC toy car, we use DC motors to actuate the motion of the wheels of the car, increasing the angular velocity of the left wheel to turn right and vice versa. All you need to do is apply power, and weeeee it spins, to rotate it in reverse direction just rotate the polarity of the input, while to vary it's speed simply change the input voltage levels.

BLDC Motors

Brush-Less DC motors are more power efficient, with significantly lower noise than DC motors. They are often used in pumps, fans and electric vehicles. Moreover,

owing to their lightweight high RPM values, most of the drones these days are predominantly based on BLDC motors.

Servo Motors

Servo motors are not used for continuous rotation (like wheels) instead, they are used for position control systems like robotic arms. Their main purpose is to rotate/ move something by a fixed angle/ distance. Unlike DC motors they have a feedback control signal in addition to the power supply.

Stepper Motors

Their working principle is similar to DC motors except for the fact that it has multiple coils. In order to generate motor rotation, these coils need to be activated in a particular pattern. They can be used for precise step movements such as in the case of a clock.

Are we going in the right direction?

Most important current challenges AI creates for humanity, and some of the likely future directions the technology might take.

Specifications to keep in mind for AI

Four Possibilities

PERFECT RATIONALITY

. A perfectly rational agent acts at every instant in such a way as to maximize its expected utility, given the information it has acquired from the environment. We have seen that the calculations necessary to achieve perfect rationality in most environments are too time consuming, so perfect rationality is not a realistic goal.

CALCULATIVE RATIONALITY

. This is the notion of rationality that we have used implicitly in designing logical and decision-theoretic agents, and most of theoretical AI research has focused on this property. A calculatively rational agent eventually returns what would have been the rational choice at the beginning of its deliberation. This is an interesting property for a system to exhibit, but in most environments, the right answer at the wrong time is of no value. In practice, AI system designers are forced to compromise on decision quality to obtain reasonable overall performance; unfortunately, the theoretical basis of calculative rationality does not provide a well-founded way to make such compromises.

BOUNDED RATIONALITY

. Herbert Simon (1957) rejected the notion of perfect (or even approximately perfect) rationality and replaced it with bounded rationality, a descriptive theory of decision making by real agents. He wrote, The capacity of the human mind for formulating and solving complex problems is very small compared with the size of the problems whose solution is required for objectively rational behavior in the real world—or even for a reasonable approximation to such objective rationality. He suggested that bounded rationality works primarily by satisficing—that is, deliberating only long enough to come up with an answer that is “good enough.” Simon won the Nobel Prize in economics for this work and has written about it in depth (Simon, 1982). It appears to be a useful model of human behaviors in many cases. It is not a formal specification for intelligent agents, however, because the definition of “good enough” is not given by the theory. Furthermore, satisficing seems to be just one of a large range of methods used to cope with bounded resources.

BOUNDED OPTIMALITY

. A bounded optimal agent behaves as well as possible, given its computational resources. That is, the expected utility of the agent program for a bounded optimal agent is at least as high as the expected utility of any other agent program running on the same machine.

Of these four possibilities, bounded optimality seems to offer the best hope for a strong theoretical foundation for AI. It has the advantage of being possible to achieve: there is always at least one best program—something that perfect rationality lacks.

Future of Artificial Intelligence(If AI Succeed)

Healthcare:

AI will play a vital role in the healthcare sector for diagnosing diseases quickly and more accurately. New drug discovery will be faster and cost-effective with the help of AI. It will also enhance the patient engagement in their care and also make ease appointment scheduling, bill paying, with fewer errors. However, apart from these beneficial uses, one great challenge of AI in healthcare is to ensure its adoption in daily clinical practices.

Cyber security:

Undoubtedly, cyber security is a priority of each organization to ensure data security. There are some predictions that cyber security with AI will have below changes:

- With AI tools, security incidents will be monitored.
- Identification of the origin of cyber-attacks with NLP.
- Automation of rule-based tasks and processes with the help of RPA bots.

Transportation:

The fully autonomous vehicle is not yet developed in the transportation sector, but researchers are reaching in this field. AI and machine learning are being applied in the cockpit to help reduce workload, handle pilot stress and fatigue, and improve on-time performance. There are several challenges to the adoption of AI in transportation, especially in areas of public transportation. There's a great risk of over-dependence on automatic and autonomous systems.

E-commerce:

Artificial Intelligence will play a vital role in the e-commerce sector shortly. It will positively impact each aspect of the e-commerce sector, ranging from user experience to marketing and distribution of products. We can expect e-commerce with automated warehouse and inventory, shopper personalization, and the use of chatbots in future.

Employment:

Nowadays, employment has become easy for job seekers and simple for employers due to the use of Artificial Intelligence. AI has already been used in the job search market with strict rules and algorithms that automatically reject an employee's resume if it does not fulfil the requirement of the company. It is hoping that the employment process will be driven by most AI-enabled applications ranging from marking the written interviews to telephonic rounds in the future.