* Concept of AI
* History

# UNIT – I

**Introduction**

* Current status
* Scope
* Agents
* Environments
* Problem Formulations
* Review of Tree and Graph Structures
* State Space Representation
* Search Graph and Search Tree

**Senses of Human Being** an under-appreciated "**sixth sense**," **called** Consciousness

allows us to keep track of where our body parts are in space.

This sixth sense is a unique characteristic of a human being.

* Taste
* Smell
* Vision
* Hearing
* Touch
* However,

|  |  |
| --- | --- |
| **Human** | **Machine** |
| Human is made of Flesh &  Blood | No life for machine, Machines  are mechanical life |
| Human have feelings and  Emotions | Machines don't have feelings  and emotions |
| Human can do anything | Machines can't do |
| Human have the capability of  understanding the situations | Machines can't understand |
| Human perform the tasks as per  own intelligence | But Machines only have AI |
| Human brains are analog | Machine brains are digital |

## What is Artificial Intelligence?

* Artificial Intelligence is composed of two words Artificial and Intelligence, where Artificial defines *"man-made,"*

*"thinking* defines intelligence and *power"*, hence AI means *"a man-made thinking power.“*

* "It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions.“
* With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

## Why Artificial Intelligence?

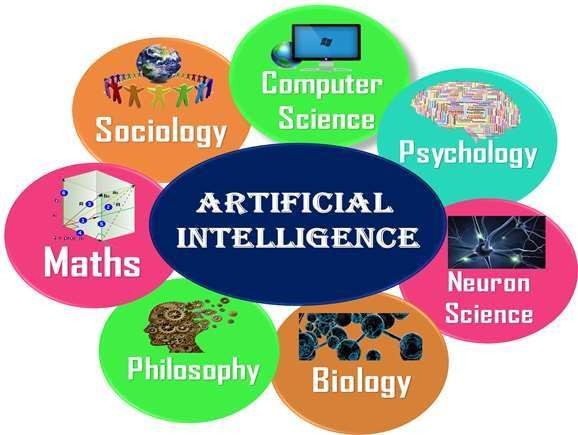
* With the help of AI, **you can create such software or devices which can solve real-world problems** very easily and with accuracy such as health issues, marketing, traffic issues, etc.
* With the help of AI, **you can create your personal virtual Assistant**, such as Cortana, Google Assistant, Siri, etc.
* With the help of AI, **you can build such Robots** which can work in an environment where survival of humans can be at risk.
* AI **opens a path for other new technologies**, new devices, and new Opportunities.

## Goals of Artificial Intelligence

1. Replicate human intelligence
2. Solve Knowledge-intensive tasks
3. An intelligent connection of perception and action
4. Building a machine which can perform tasks that requires human intelligence such as:
   * Proving a theorem
   * Playing chess
   * Plan some surgical operation
   * Driving a car in traffic

## What Comprises to Artificial Intelligence?

AI is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc**.

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

## Advantages of Artificial Intelligence

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

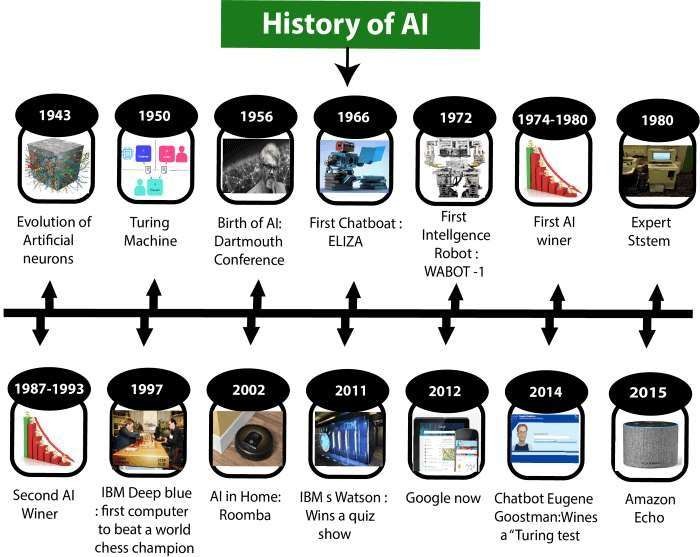
## Disadvantages of Artificial Intelligence

* **High Cost:** The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.
* **Can't think out of the box:** Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
* **No feelings and emotions:** AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.
* **Increase dependency on machines:** With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities. 14

# Foundations or Prehistory of AI

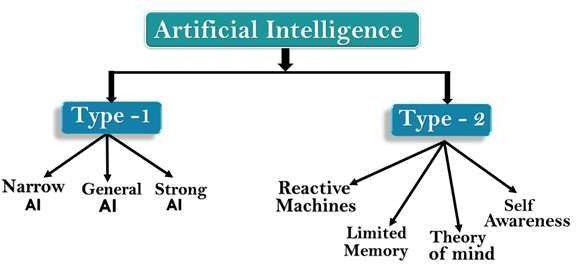
****

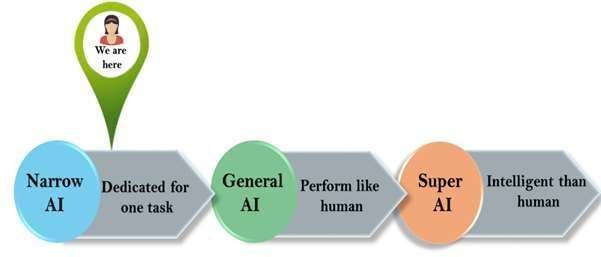
11



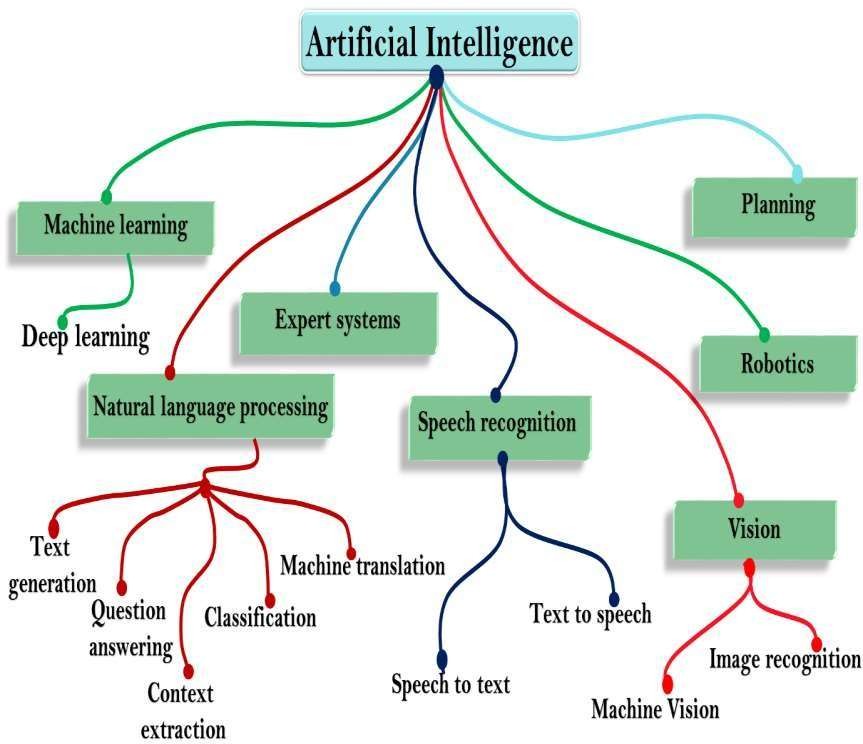
## Applications of Artificial Intelligence

**Types of Artificial Intelligence**



 **AI type-1 &2: Based on Capabilities & Functionality**

## Subsets of Artificial Intelligence

****

**State of Art (Current Status & Scope)**

The State of the Art(SOTA) in Artificial Intelligence follows the

reduction rule:

* + SOTA AI = Data Science = Machine Learning = Deep Learning =

Narrow/Weak AI

Or, DS < DL < ML = NSAI (New-Smart AI) < AI

* + The SOTA AI, as specific ML/DL models, algorithms, techniques and technologies, it is what makes today's commercially prevalent weak AI.
  + The SOTA AI is still after building machines and software agents somehow mimicking human-like cognition and intelligence (*sense (perceiving), analysis, reasoning, understanding and response)* by means of statistic learning techniques.
* Most present AI companies, from big tech to startups, are about some advanced data analytics, predictive modeling, or computational neural networks based on mathematics and algorithms, as some specific ML/DL techniques,algorithms,models or applications.

They are outperforming humans in some very narrowly defined task, focusing on imitating, simulating or faking some single cognitive ability, skill or competence.

Such a narrow weak AI can identify the patterns and correlations from big data streams while failing to discover real rules and causal patterns.

The SOTA AI is on the way to out-perform humans in detecting diseases, bank fraud, flying aircraft, translating between languages, recognizing faces, speech, emotions, trading stocks, social network filtering, aggregating news, painting, playing music, chess or Go, making purchase suggestions, sales predictions, weather forecasting, playing video games, strategic games, quiz shows, or driving cars.

The SOTA AI is widely used in e-commerce, social networks, internet search, automobile, logistic, healthcare, stock-trading, robotics, finance, transport, education, and other industries.

Still, it is hardly any real AI, aimed to integrate all the specific ML/DL/NSAI models, algorithms, techniques and technologies what make today's AI.

For, as a matter of fact, DS < DL < ML < NSAI << Real/True/Global AI

As a result, there are three different types of AI companies: the SOTA AI

companies (data

analytics companies (80%) and machine

learning

companies (19,9%)) and Real

AI companies

(0,1%).

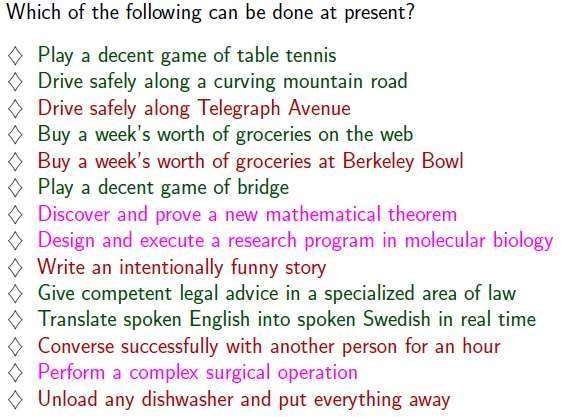
## Scope of Artificial Intelligence

1. AI in Science and Research
2. AI in Cyber Security
3. AI in Data Analysis
4. AI in Transport
5. AI in Home

(https:/[/w](http://www.upgrad.com/blog/future-scope-of-artificial-)w[w.upgrad.com/blog/future-scope-of-artificial-](http://www.upgrad.com/blog/future-scope-of-artificial-)

intelligence/)

## State of Art (Current Status & Scope)

****

20

#### INTELLIGENCE

* The capacity to learn and solve problems.
* In particular,
  + the ability to solve novel problems (i.e solve new problems)
  + the ability to act rationally (i.e act based on reason)
  + the ability to act like humans
  + the ability to deal with unexpected problems, uncertainties (reasoning and planning)

## Definitions: Artificial Intelligence

* **Artificial Intelligence** is the branch of computer science concerned with making computers behave like humans.
* 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.
* **John McCarthy**, who coined the term in 1956, defines it as "the science and engineering of making intelligent machines, especially intelligent computer programs."

**SOME OTHER DEFINITIONS OFAI**

* + **Building systems that think like humans**

―The exciting new effort to make computers think

… machines with minds, in the full and literal sense -- Haugeland, 1985

―The automation of activities that we associate with human thinking, … such as decision- making, problem solving, learning, …-- Bellman, 1978

* + **Building systems that act like humans**

―The art of creating machines that perform functions that require intelligence when performed by people -- Kurzweil, 1990

―The study of how to make computers do things at which, at the moment, people are better -- Rich and Knight, 1991 27

* + **Building systems that think rationally**

―The study of mental faculties through the use of computational models -- Charniak and McDermott, 1985

―The study of the computations that make it possible to perceive, reason, and act-Winston, 1992

* + **Building systems that act rationally**

―A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes -- Schalkoff, 1990

―The branch of computer science that is concerned with the automation of intelligent behavior -- Luger and Stubblefield, 1993

**MAJOR CATEGORIES OFAI**

* It is the study of how to make computers do things at which, at the moment, people are better. The term AI is defined by each author in own ways which falls into 4 categories

1. Systems that think like human.

2. Systems that act like human.

1. Systems that think rationally.
2. Systems that act rationally.
   1. Acting Humanly: The Turing Test Approach

* Test proposed by Alan Turing in 1950
* The computer is asked questions by a human interrogator.
* The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not. Programming a computer to pass, the computer need to possess the following capabilities:

**Natural language processing** to enable it to communicate successfully in English.

**Knowledge representation** to store what it knows or hears

**Automated reasoning** to use the stored information to answer questions and to draw new conclusions.

**Machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

* 1. Thinking humanly: The cognitive modeling approach
* We need to get inside actual working of the human mind:

–Through introspection – trying to capture our own thoughts as they go by;

–Through psychological experiments

* Alle Newell and Herbert Simon, who developed GPS, the ―General Problem Solver‖ tried to trace the reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind 31
  1. **Thinking rationally : The “laws of thought approach”**
* The Greek philosopher Aristotle was one of the first to attempt to codify ―right thinking that is irrefutable (ie. Impossible to deny) reasoning processes. His syllogism provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example, Socrates is a man; all men are mortal; therefore Socrates is mortal.‖. These laws of thought were supposed to govern the operation of the mind; their study initiated a field called logic.

#### Acting rationally : The rational agent approach

* An agent is something that acts. Computer agents are not mere programs, but they are expected to have the following attributes also:

(a)operating under autonomous control,

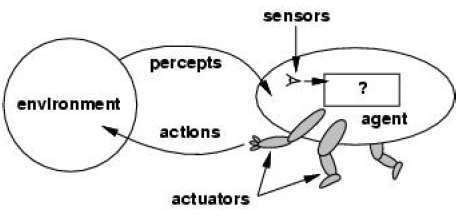
(b) perceiving their environment,

(c) persisting over a prolonged time period,

(e) adapting to change. A rational agent is one that acts so as to achieve the best outcome.

## 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.** This simple idea is illustrated in Figure.
* A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
* A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
* 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.



**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**

#### Specifying the Agent’s Task Environment

* + Performance measure
  + Environment
  + Actuators
  + Sensors

## PEAS

* All these are grouped together under the heading of the **task**

**environment.**

* We call this the **PEAS** (Performance, Environment, Actuators,

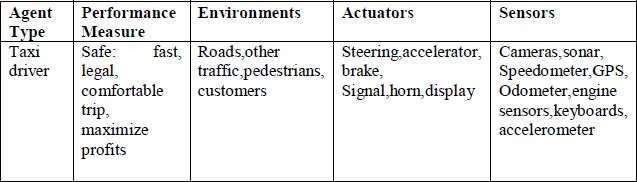
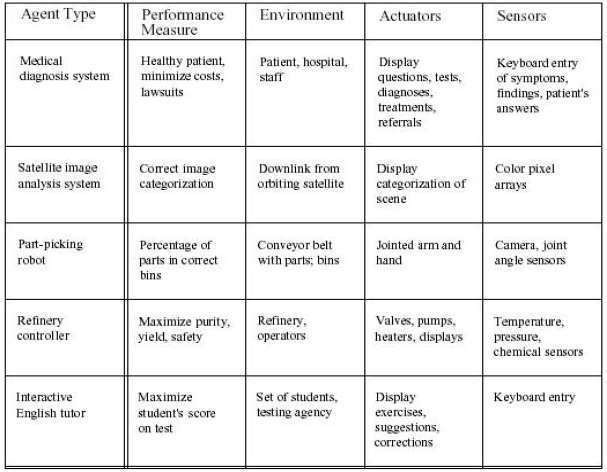
Sensors) description.

* In designing an agent, the first step must always be to specify

the task environment as fully as possible.

* The following table shows PEAS description of the task

environment for an automated taxi.



## Properties of Task Environments

1. **Fully observable vs. partially observable**
2. **Deterministic vs. stochastic**
3. **Episodic vs. sequential**
4. **Static vs. dynamic**
5. **Discrete vs. continuous**
6. **Single agent vs. multiagent**

* **Fully observable** vs. **partially observable.**
* If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable.
* A task environment is effectively fully observable if the sensors detect all

aspects that are *relevant* to the choice of action;

* An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.

**Deterministic** vs. **stochastic.**

* If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic;
* Otherwise, it is stochastic.

**Episodic** vs. **sequential**

* In an **episodic task environment**, the agent's experience is divided

into atomic episodes.

* Each episode consists of the agent perceiving and then performing a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
* For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions;
* In **sequential environments**, on the other hand, the current decision

Could affect all future decisions.

* Chess and taxi driving are sequential:

**Discrete** vs. **continuous.**

* The discrete/continuous distinction can be applied to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent.
* For example, a discrete-state environment such as a chess game has a finite

number of distinct states.

* Chess also has a discrete set of percepts and actions.
* Taxi driving is a continuous- state and continuous-time problem:
* The speed and location of the taxi and of the other vehicles sweep through a

range of continuous values and do so smoothly over time.

* Taxi-driving actions are also continuous (steering angles, etc.)

**Single agent** vs. **multiagent.**

* An agent solving a crossword puzzle by itself is clearly

single-agent

environment,

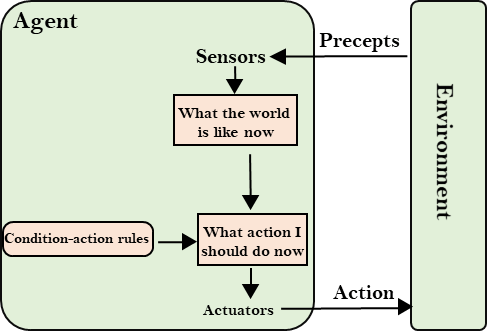
in a

* Where as an agent playing chess is in a two-agent environment.
* Multiagent is further classified in to two ways
* Competitive multiagent environment
* Cooperative multiagent environment

## Types of Agent’s Architecture

* 1. Simple Reflex Agent
  2. Model-based Reflex Agent
  3. Goal-based Agents
  4. Utility-based Agent
  5. Learning Agent
     1. **Simple Reflex Agent**
* The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
* These agents only succeed in the fully observable environment.
* The Simple reflex agent does not consider any part of percepts

history during their decision and action process.

* The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.

#### Model-based Reflex Agent

The Model-based agent can work in a partially observable environment, and

track the situation.

A model-based agent has two important factors:

**Model:** It is knowledge about "how things happen in the world," so it is

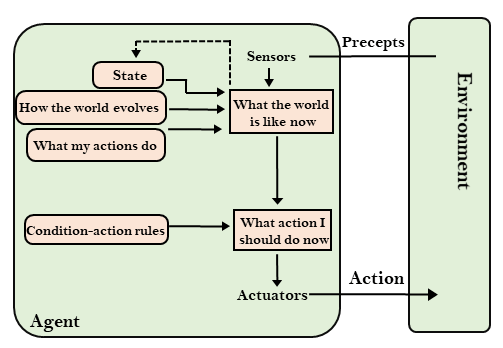
called a Model-based agent.

**Internal State:** It is a representation of the current state based on

percept history.

These agents have the model, "which is knowledge of the world" and based on the model they perform actions.

Updating the agent state requires information about:

* + How the world evolves
  + How the agent's action affects the world.

#### Goal-based Agents

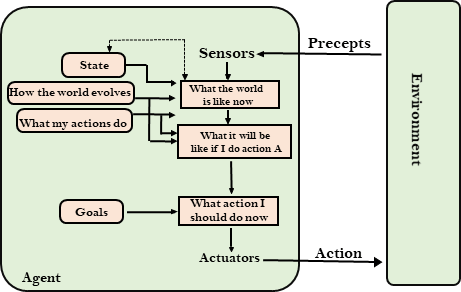
* The knowledge of the current state environment is not always sufficient

to decide for an agent to what to do.

* The agent needs to know its goal which describes desirable situations.
* Goal-based agents expand the capabilities of the model-based agent by

having the "goal" information.

* They choose an action, so that they can achieve the goal.

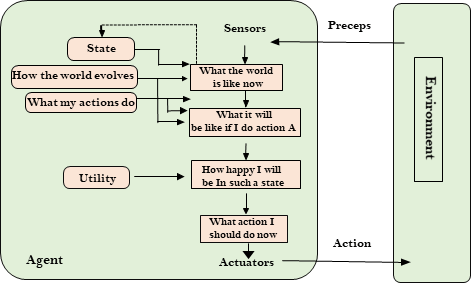


#### Utility-based Agents

* + These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
  + Utility-based agent act based not only goals but also the best

way to achieve the goal.

* + The Utility-based agent is useful when there are multiple possible

alternatives, and an agent has to choose in order to perform the best action.

## Learning Agents

* + - A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
    - It starts to act with basic knowledge and then able to act

and adapt automatically through learning.

* + - A learning agent has mainly four conceptual

components, which are:

**Learning element:** It is responsible for making

improvements by learning from environment

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

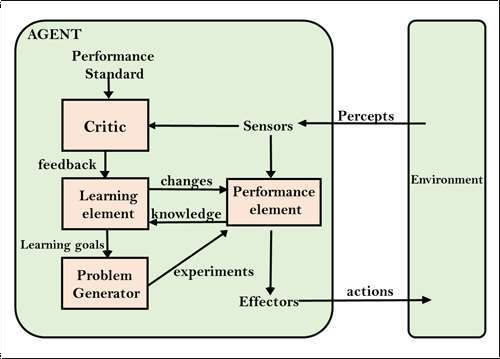
**Performance element:** It is responsible for selecting

external action

**Problem generator:** This component is responsible

for suggesting actions that will lead to new and

informative experiences. 46



## Performance Measures (Criteria for Success)

The performance measure that defines the criterion of success of an Agent’s behavior.

–The agent's prior knowledge of the environment.

–The actions that the agent can perform.

–The agent's percept sequence.

## AI Problem Solving

* Before solving the problem, we must understand the problem clearly
* To solve the problem we need the formal description of the problem
* What are aspects to understand the problem?

1. What is the goal of the problem? (Goal Formulation)
2. What is the implicit criteria for success? (how success is defined?
3. What is the initial situation?
4. Ability to perform?

(What are the procedures?)

## Well Defined Problem

* The process of deciding what actions and states to be considered is called as “Problem Formulation”.
* The solution of many problems can be described by finding a sequence of actions that lead to a desirable goal.
* Components of Well Defined Problem:

1. **Initial state** (Where the agent starts in)
2. **Successor Function** (Description of the possible actions)
3. **State Space** (Set of all possible actions)
4. **Path Cost** (Cost to reach the Goal)
5. **Goal Test** (Determines whether a given state is final state or not? Eg: Checkmate in Chess is Goal)

## Types of Problem Formulation

|  |  |
| --- | --- |
| **Incremental Formulation** | **Complete State Formulation** |
| Staring with empty set | Some basic configuration is represented in the initial state |
| Generates many sequences | It will work based on the actions |
| Memory requirements is less | Memory requirements is high |

**Phases in Problem Solving**

What are the procedures to be followed in order to find the solution for a problem

* 1. Problem Definition
  2. Problem Analysis (i/p?, environment?, expected o/p?)
  3. Knowledge Representation (Data Structures?)
  4. Technique (Selection of best technique)

## Problem-solving agents

* A Problem solving agent is a goal-based agent.
* It decides what to do by finding sequence of actions that lead to desirable states.
* Problem Solving Agent

Determines and generates

Sequence of Actions generates

Successful States

## Steps in Problem Solving

**Step 1:** Goal Setting

**Step 2:** Goal Formulation

-> to observe the current state

-> to tabulate agent performance measures

**Step 3:** Problem Formulation

-> what will be the sequence of actions?

-> what will be the sequence of states?

**Step 4:** Search in unknown environment (Learning)

**Step 5:** Execution Phase

## Types of Problems

#### Toy Problems

* + 8 -Queens problem
  + Vacuum World
  + Ball Picker Robot

#### Real World Problems

* + Route Finding Problem
  + Travelling Salesman Problem
  + Robot Navigation

## 8 Queens Problem

**Problem Statement:** 8X8 Chess Board

**Objective:** No two queens are attacking by horizontally, vertically and diagonally

**Step 1: Initial state** -> No queens on the board

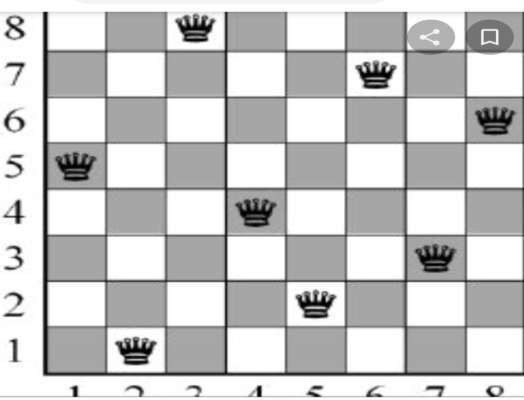
**Step 2: Successor Function** -> Insert a Queen

**Step 3: Goal Test** -> Check whether all 8 Queens are placed or not

**Step 4: Path Cost** (Total no of moves)

**Step 5: State Space** (How many solutions?)

Solution for 8 Queen’s Problem



## Problem Characteristics

1. Is the problem decomposable to smaller or easier problems?
2. Can problem solution steps be ignored or undone?
   1. Ignorable: Solution steps can be ignored
   2. Recoverable: Solution steps can be undone or

backtracking

* 1. Irrecoverable: Moves cannot be retracted

1. Is the problem universe predictable?
   1. Problem with certain outcome(8 Queens)
   2. Problem with uncertain outcome(Chess)
2. Is a good solution absolute or relative?
3. Is the solution a state or a path?
4. Is the problem using knowledge base?
5. What is the role of the knowledge?
6. Does the task require a periodic human interaction with computer?

# State Space Search

A state space represents a problem in terms of states and operators

that change states.

**A state space consists of:**

1. A representation of the **states** the system can be in. For example, in a board game, the board represents the current state of the game.
2. A set of **operators** that can change one state into another state.

In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.

1. An **initial state**.
2. A set of **final states**; some of these may be desirable, others undesirable. This set is often represented implicitly by a program that detects terminal states.

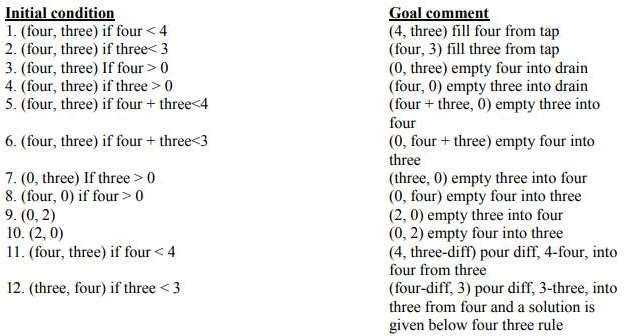
The Water Jug Problem

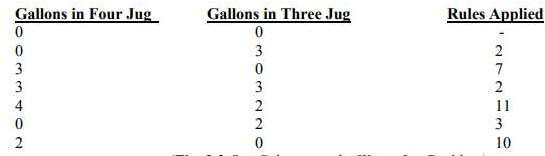
In this problem, we use two jugs called four and three; four holds a maximum of four gallons of water and three a maximum of three gallons of water. How can we get two gallons of water in the four jug?

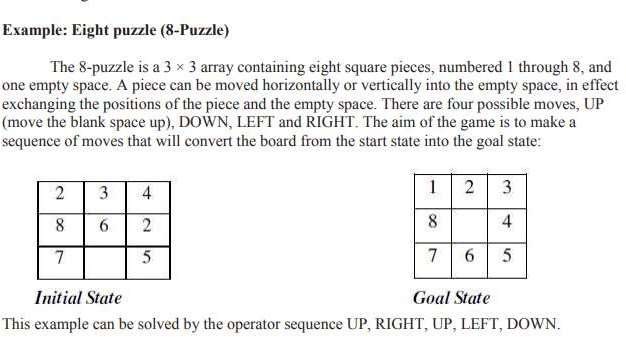
The state space is a set of prearranged pairs giving the number of gallons of water in the pair of jugs at any time, i.e., (four, three) where four = 0, 1, 2, 3 or 4 and three = 0, 1,

2 or 3.

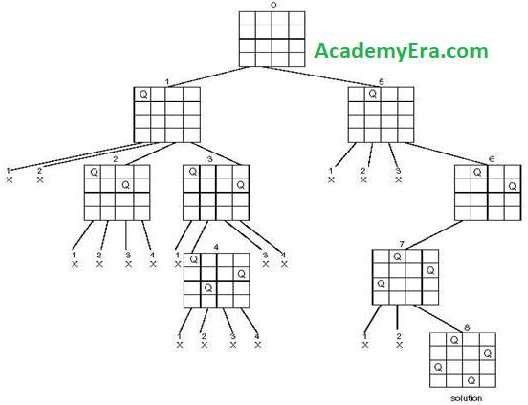
The start state is (0, 0) and the goal state is (2, n) where n may be any but it is limited to three holding from 0 to 3 gallons of water or empty. Three and four shows the name and numerical number shows the amount of water in jugs for solving the water jug problem. The major production rules for solving this problem are shown below:





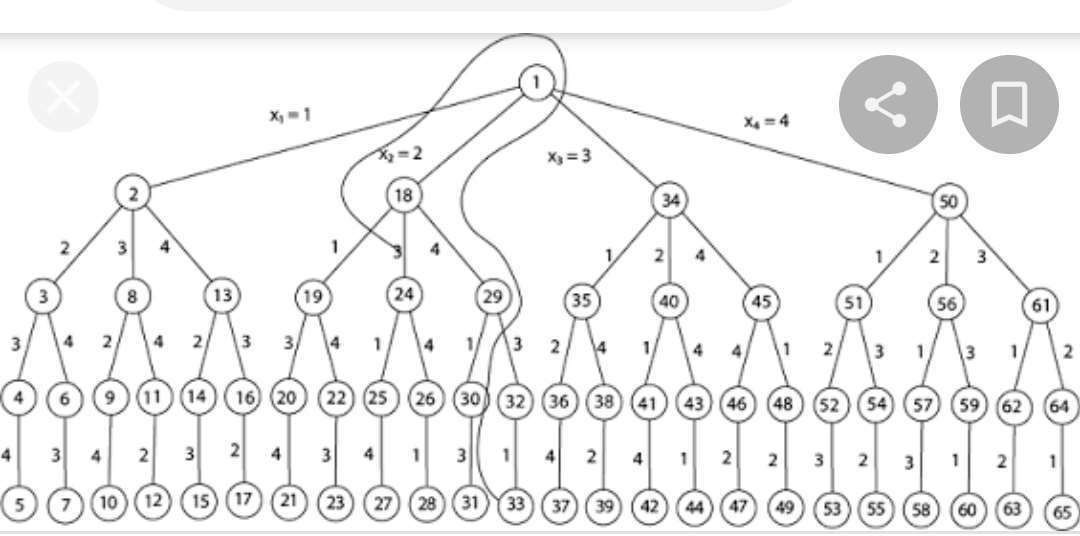


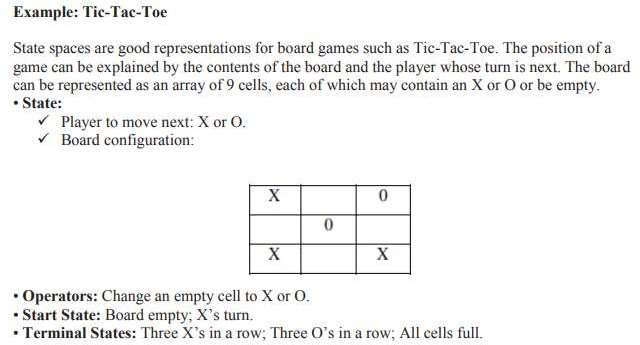
State Space Tree for 4 - Queens Problem



State Space Tree for 8

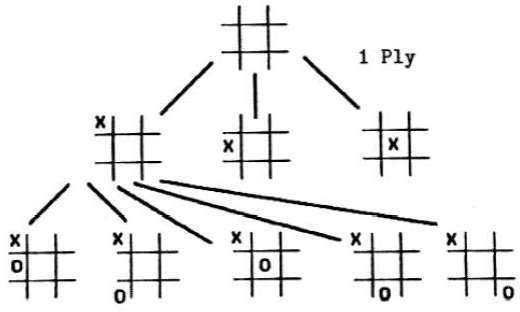
Queen’s Problem





## Search Tree for Tic-Tac-Toe Game

The sequence of states formed by possible moves is called a search tree. Each level of the tree is called a ply.



**Tree (Data Structure)**

A tree data structure can be defined [recursively](https://en.wikipedia.org/wiki/Recursion) as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value, together with a list of references to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.

**Terminology used in Trees:**

***Node***

A node is a structure which may contain a value or condition, or represent a

separate data structure.

***Root***

The top node in a tree, the prime ancestor.

***Child***

A node directly connected to another node when moving away from

the root,

an immediate descendant.

***Parent***

The converse notion of a child, an immediate ancestor.

***Siblings***

A group of nodes with the same parent. 68

***Neighbor*** Parent or child. ***Descendant***

A node reachable by repeated proceeding from parent to child. Also known

as subchild.

***Ancestor***

A node reachable by repeated proceeding from child to parent.

***Leaf / External node (not common)***

A node with no children. ***Branch node / Internal node*** A node with at least one child. ***Degree***

For a given node, its number of children. A leaf is necessarily degree

zero. The degree of a tree is the degree of its root.

***Degree of tree***

The degree of the root.

***Edge***

The connection between one node and another.

***Path***

A sequence of nodes and edges connecting a node with a descendant.

69

***Distance***

The number of edges along the shortest path between two nodes.

***Depth***

The distance between a node and the root.*Level*1 + the number of

edges between a

node and the root, i.e. (Depth + 1)

***Height***

The number of edges on the longest path between a node and a

descendant leaf.

***Width***

The number of nodes in a level.

***Breadth***

The number of leaves.

***Height of tree***

The height of the root node or the maximum level of any node in the tree.

***Forest***

Aset of *n* ≥ 0 disjoint trees.

***Sub Tree***

Atree *T* is a tree consisting of a node in *T* and all of its descendants in *T*.

***Ordered Tree***

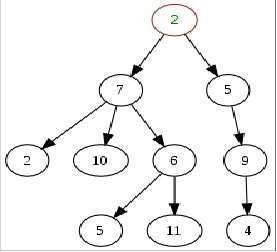
Arooted tree in which an ordering is specified for the children of each

vertex.

***Size of a tree***

Number of nodes in the tree. 70

Example: Tree



A generic, and so non-binary, unsorted, some labels duplicated, arbitrary diagram of a tree. In this diagram, the node labeled 7 has three children, labeled 2, 10 and 6, and

one parent, labeled 2. The root node, at

parent.

the top, has no

71

1. In-order Traversal

**Tree Traversals**

1. Pre-order Traversal
2. Post-order Traversal

In-order Traversal (Algorithm)

Until all nodes are traversed −

**Step 1** − Recursively traverse left subtree.

**Step 2** − Visit root node.

**Step 3** − Recursively traverse right subtree.

Pre-order Traversal (Algorithm)

Until all nodes are traversed −

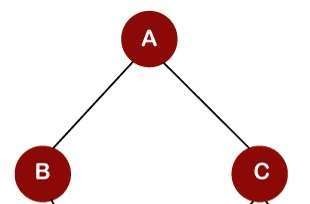
**Step 1** − Visit root node.

**Step 2** − Recursively traverse left subtree.

**Step 3** − Recursively traverse right subtree. 72

Post-order Traversal (Algorithm)

Until all nodes are traversed −

**Step 1** − Recursively traverse left subtree. **Step 2** − Recursively traverse right subtree. **Step 3** − Visit root node..

**In-order Traversal = B -> A -> C Pre-order Traversal = A -> B -> C Post-order Traversal = B -> C -> A**

#### Graphs (Data Structures)

* A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are

called **edges**.

* Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph −
* In this graph,

V = {a, b, c, d, e}

A

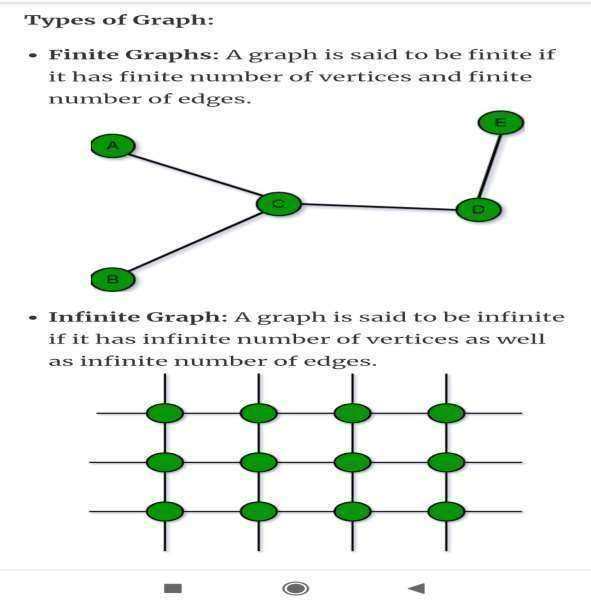
B

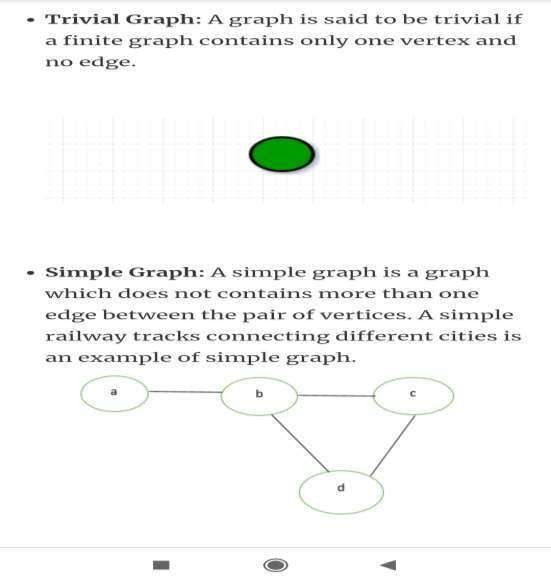
C

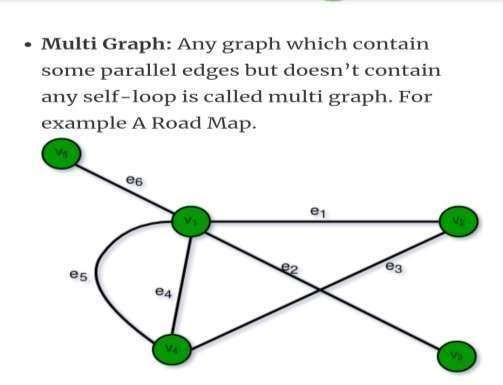
D

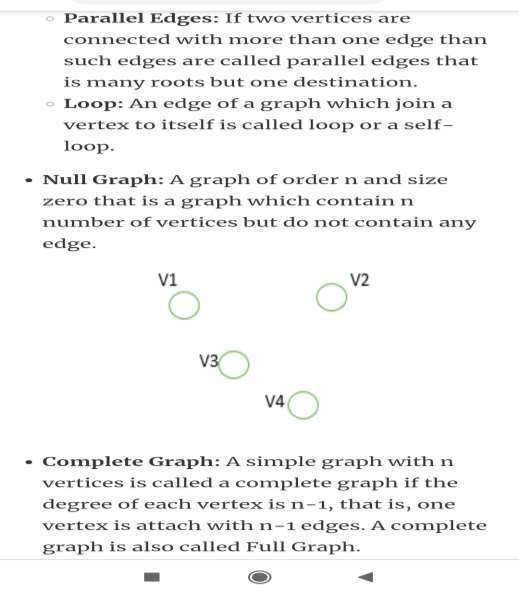
E

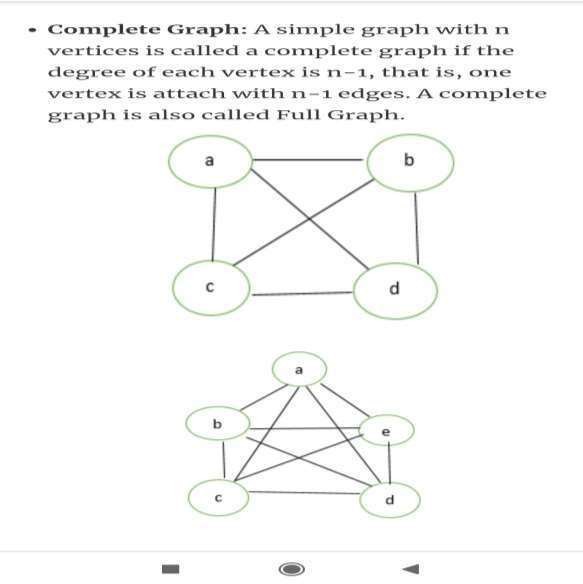
E = {ab, ac, bd, cd, de}

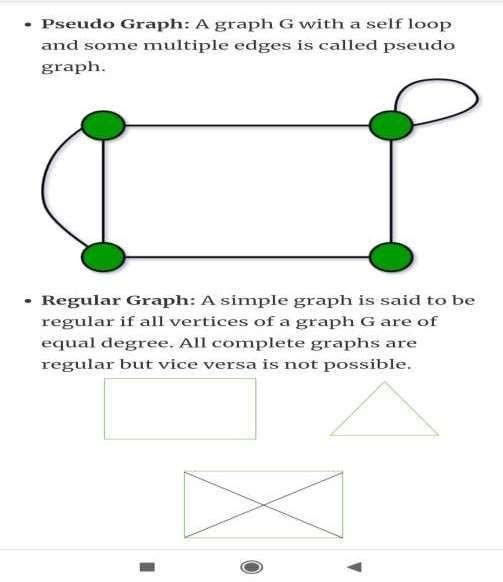












###### Search Algorithms

* The process of finding an element in the data structure is called

as “**Searching”**.

* **Search Graph** is an algorithm that visits vertices and edges in graph in an order based on the connectivity of the graph.
* **Components of Search Algorithm**
* **A State Space:** Set of all possible states where you can be.
* **A Start State:** The state from where the search begins.
* **A Goal Test:** A function that looks at the current state returns whether or not it is the goal state.
* The **Solution** to a search problem is a sequence of actions,

called the **plan** that transforms the start state to the goal state.

* This plan is achieved through search algorithms.

Search Terminology

* + **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
  + **Problem Instance** − It is Initial state + Goal state.
  + **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
  + **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
  + **Space Complexity** − The maximum number of nodes that are stored in memory.
  + **Time Complexity** − The maximum number of nodes that are created.
  + **Admissibility** − A property of an algorithm to always find an optimal solution.
  + **Branching Factor** − The average number of child nodes in the problem space graph.
  + **Depth** − Length of the shortest path from initial state to goal state. 82

**Properties of Search Algorithms:**

* + - **Completeness:** A search algorithm is said to be complete if it

guarantees to return a solution.

* + - **Optimality:** If a solution found for an algorithm is guaranteed to be

the best solution (lowest path cost) among all other solutions.

* + - **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.

**Problem-solving agents:**

**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.

**Issues in the Design of Search Programs**

* Each search process can be considered to be a tree traversal. The object of the search is to find a path from the initial state to a goal state using a tree. The number of nodes generated might be huge; and in practice many of the nodes would not be needed. The secret of a good search routine is to generate only those nodes that are likely to be useful, rather than having a precise tree.

The following issues arise when searching:

* The tree can be searched forward from the initial node to the goal

state or backwards from the goal state to the initial state.

* To select applicable rules, it is critical to have an efficient procedure for matching rules against states.
* How to represent each node of the search process? This is the knowledge representation problem or the frame problem. In games, an array suffices; in other problems, more complex data structures are needed.

## Types of Search Algorithms

1. **Brute-Force Search Strategies (Uninformed / Blind)**

* **Breadth First Search (BFS)**
* **Depth First Search (DFS)**
* Bidirectional Search
* Uniform Cost Search
* Iterative Deepening Depth-First Search

1. **Informed (Heuristic) Search Strategies**

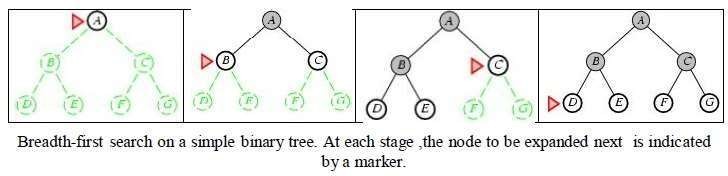
* Pure Heuristic Search (Open and Closed List)
* Greedy Best First Search
* A \* Search

1. **Local Search Algorithms**

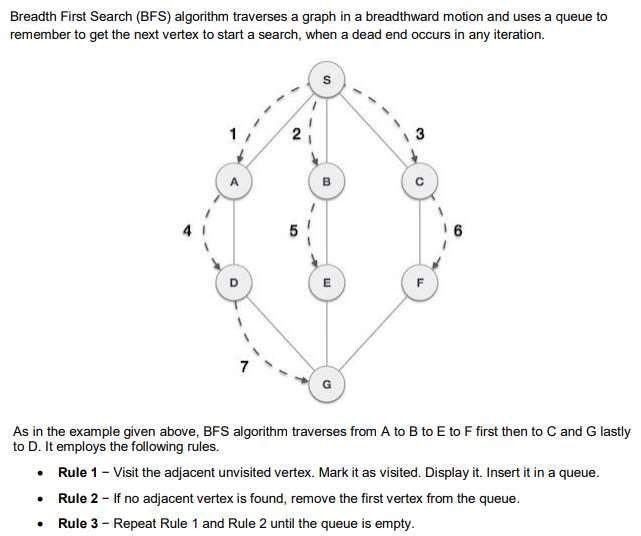
* Hill-Climbing Search
* Local Beam Search
* Simulated Annealing
* Travelling Salesman Problem

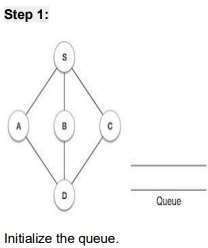
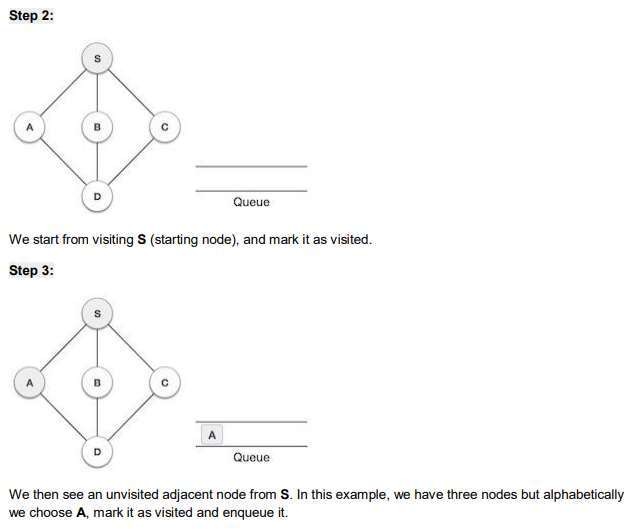
#### BREADTH FIRST SEARCH (BFS)

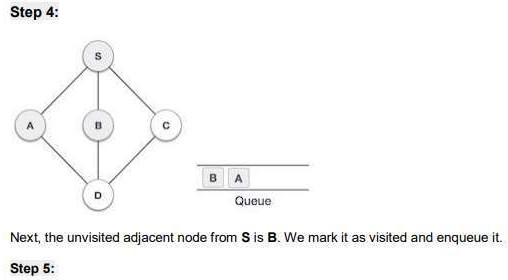
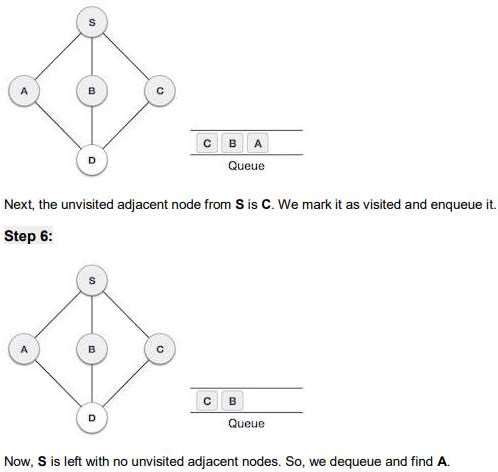
* Breadth-first search in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on.
* In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
* Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out(FIFO) queue, assuring that the nodes that are visited first will be expanded first.
* In otherwards, calling TREE-SEARCH (problem, FIFO- QUEUE()) results in breadth-first-search.
* The FIFO queue puts all newly generated successors at the end of the queue, which means that Shallow nodes are expanded before deeper nodes.

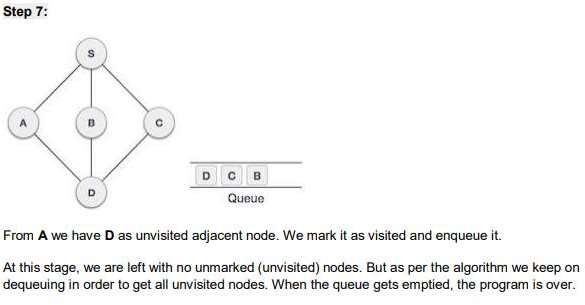


BFS - Example

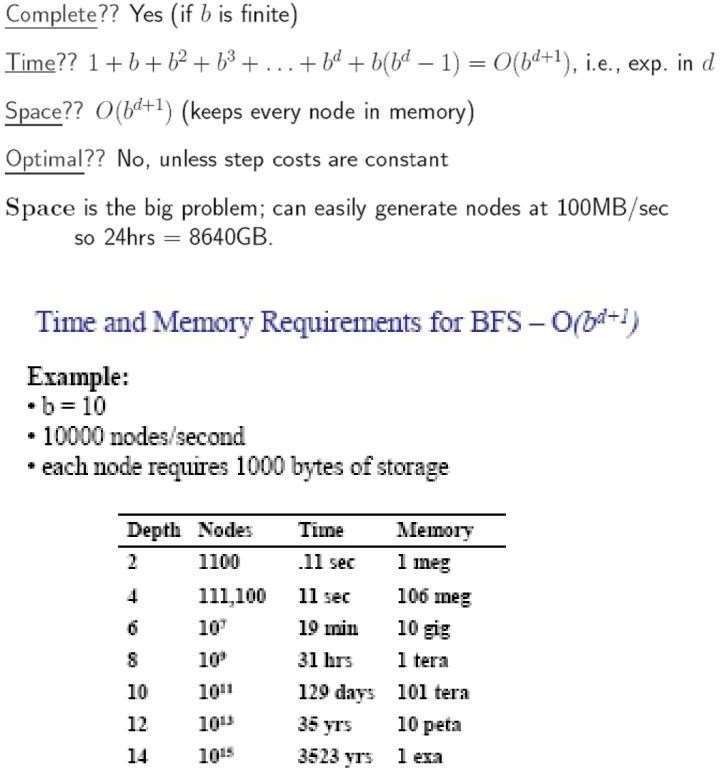








**Properties of BFS**



#### DEPTH FIRST SEARCH (DFS)

–Depth-first-search always expands the deepest node in the current fringe of the search tree.

–The progress of the search is illustrated in figure.

–The search proceeds immediately to the deepest level of the

where the nodes have no successors.

–As those nodes are expanded, they are dropped from the fringe,

search tree,

–so then the search "backs up" to the next shallowest node that still has

unexplored successors.

–This strategy can be implemented by TREE-SEARCH with a last-in-first-

out (LIFO) queue, also known as a stack.

–Depth-first-search has very modest memory requirements.

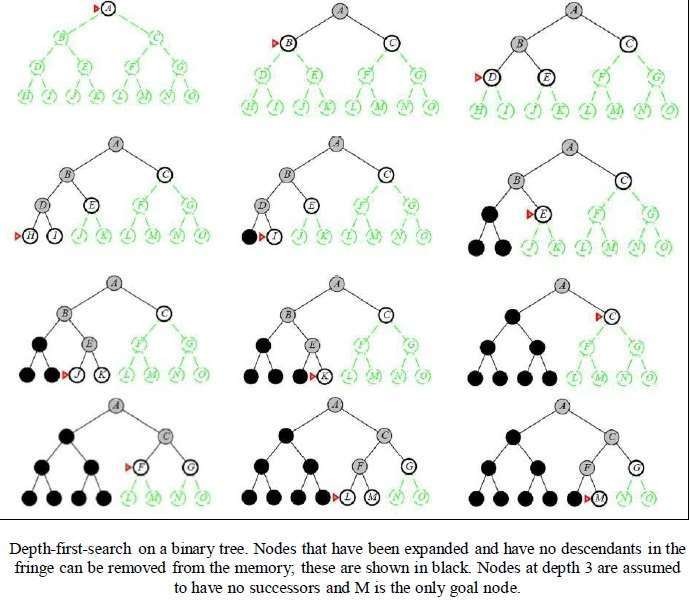
–It needs to store only a single path from the root to a leaf node along with the remaining unexpanded sibling nodes for each node on the path.

–Once the node has been expanded, it can be removed from the memory, as

soon as its descendants have been fully explored.

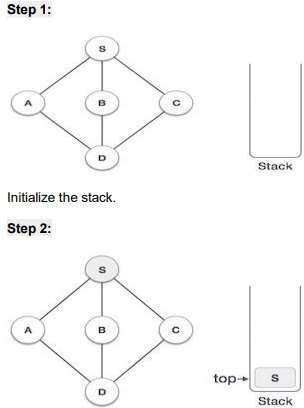
–For a state space with a branching factor b and maximum depth m, depth-

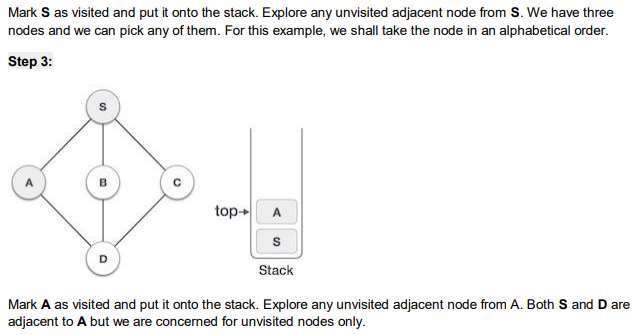
first-search requires storage of only bm + 1 nodes.

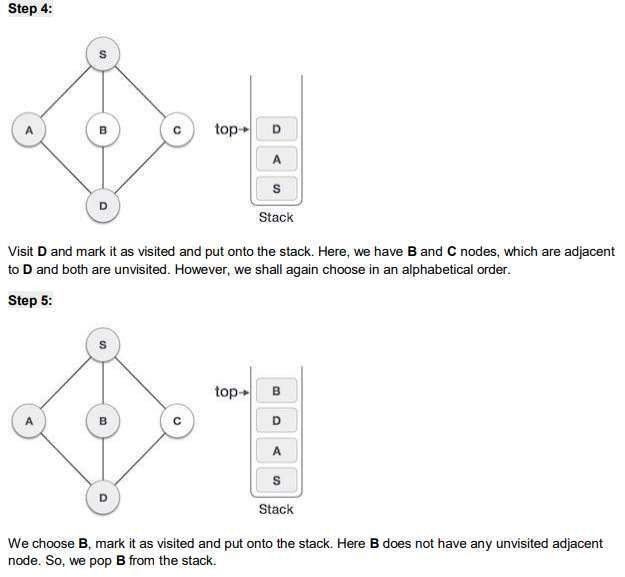


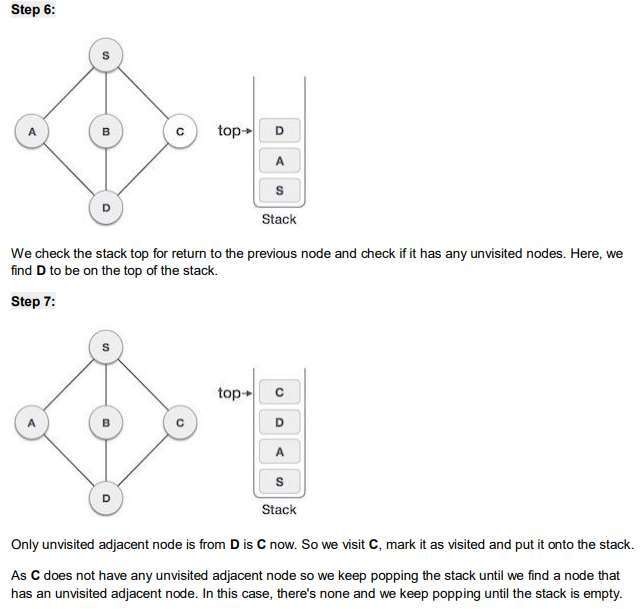
# 

# DFS - Example









|  |  |  |
| --- | --- | --- |
| **Parameters** | **BFS** | **DFS** |
| **Data structure** | Queue (FIFO) | Stack (LIFO) |
| **Source** | It is better when target is closer to source | It is better when target is far from source |
| **Speed** | Slower | Faster |
| **Time Complexity** | O(V+E) | O(V+E) |
| **Suitability for Decision Trees** | BFS considers all neighbor, so it is not suitable for decision tree | Suitable for decision tree |

**UNIT – II**

# Search Algorithms

* Random search
* Search with closed and open list
* Depth first and Breadth first search
* Heuristic search
* Best first search
* A\* Search Algorithm
* Game Search

###### Search Algorithms

* The process of finding an element in the data structure is called as “**Searching”**.
* **Search Graph** is an algorithm that visits vertices and edges in graph in an order based on the connectivity of the graph.
* **Components of Search Algorithm**
  + **A State Space:** Set of all possible states where you can be.
  + **A Start State:** The state from where the search begins.
  + **A Goal Test:** A function that looks at the current state

returns whether or not it is the goal state.

* The **Solution** to a search problem is a sequence of actions,

called the **plan** that transforms the start state to the goal state.

* This plan is achieved through search algorithms.

6

Search Terminology

* **Problem Space** − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
* **Problem Instance** − It is Initial state + Goal state.
* **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
* **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
* **Space Complexity** − The maximum number of nodes that are stored in memory.
* **Time Complexity** − The maximum number of nodes that are created.
* **Admissibility** − A property of an algorithm to always find an optimal solution.
* **Branching Factor** − The average number of child nodes in the problem space graph.
* **Depth** − Length of the shortest path from initial state to goal state. 7

**Properties of Search Algorithms:**

* + **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution.
  + **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions.
  + **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.

**Problem-solving agents:**

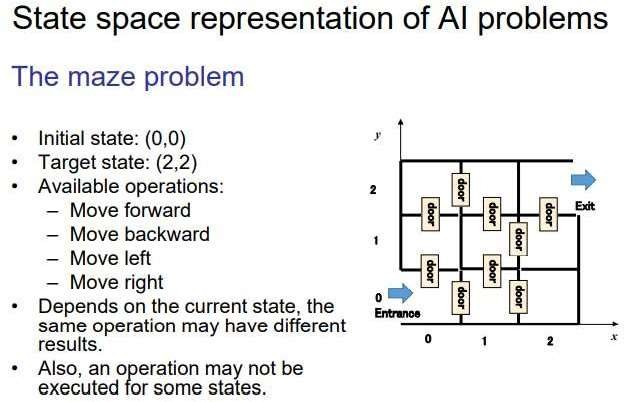
**Rational agents** or **Problem-solving agents** in AI mostly used these search strategies or algorithms to solve a specific problem and provide the best result. Problem-solving agents are the goal-based agents and use atomic representation.

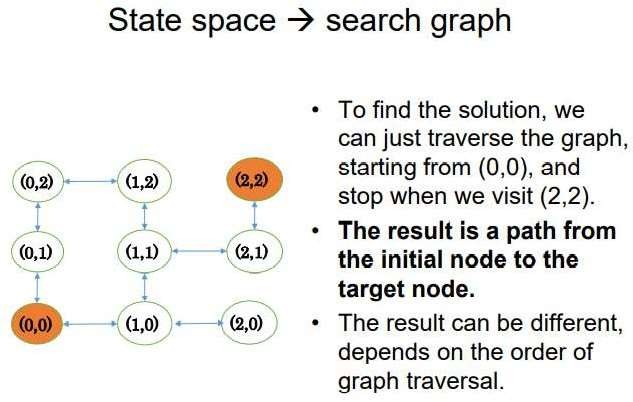
**Issues in the Design of Search Programs**

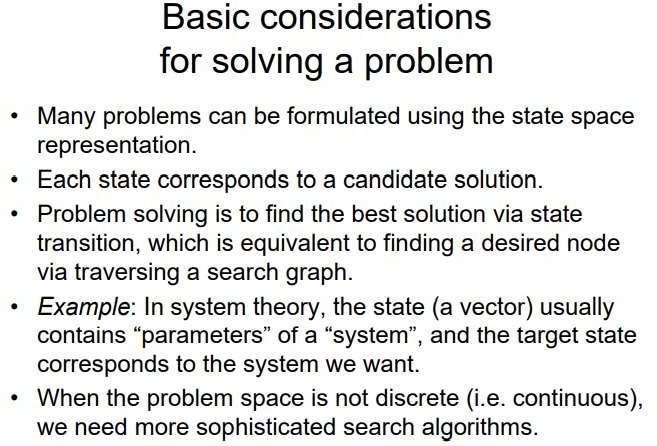
* Each search process can be considered to be a tree traversal. The object of the search is to find a path from the initial state to a goal state using a tree. The number of nodes generated might be huge; and in practice many of the nodes would not be needed. The secret of a good search routine is to generate only those nodes that are likely to be useful, rather than having a precise tree.

The following issues arise when searching:

* The tree can be searched forward from the initial node to the goal state or backwards from the goal state to the initial state.
* To select applicable rules, it is critical to have an efficient procedure for matching rules against states.
* How to represent each node of the search process? This is the knowledge representation problem or the frame problem. In games, an array suffices; in other problems, more complex data structures are needed.

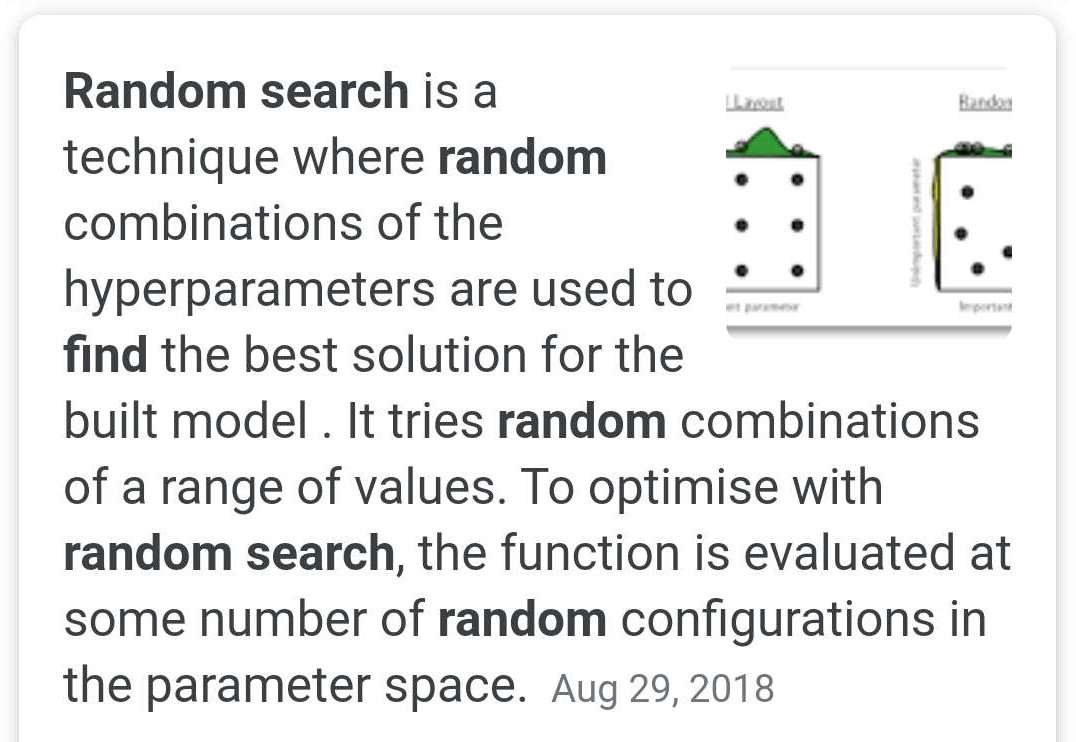


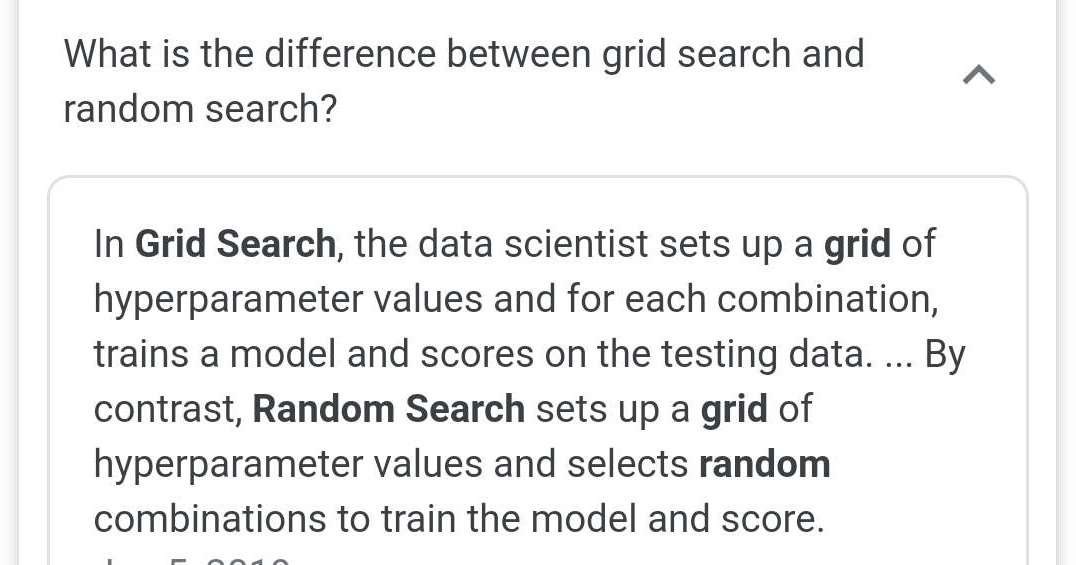


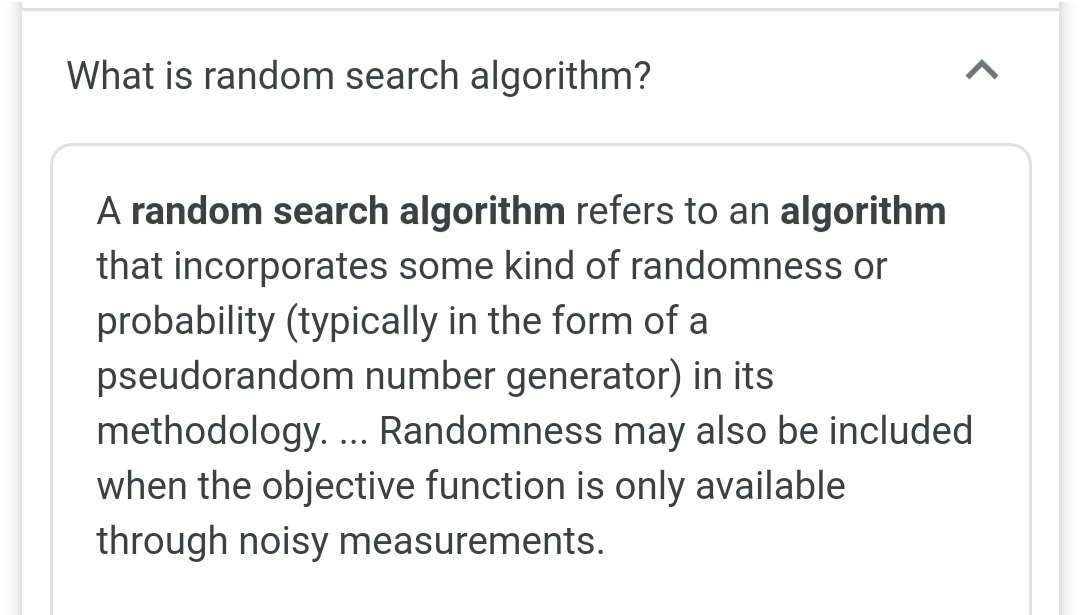


#### Random Search

* **Random search (RS)** is a family of numerical optimization methods that do not require the gradient of the problem to be optimized, and RS can hence be used on functions that are not continuous or differentiable. Such optimization methods are also known as direct-search, derivative-free, or black-box methods.
* The name "random search" is attributed to Rastrigin who made an early presentation of RS along with basic mathematical analysis. RS works by iteratively moving to better positions in the search-space, which are sampled from a hypersphere surrounding the current position.





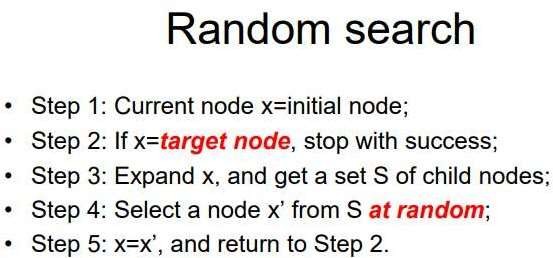


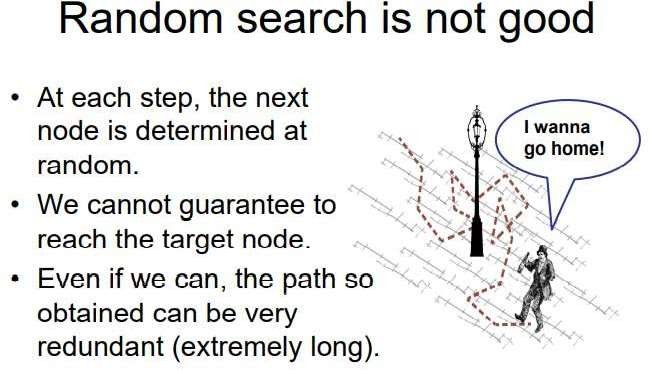
**Random Search Algorithm**

* Let *f*: ℝ*n* → ℝ be the fitness or cost function which must be minimized. Let **x** ∈ ℝ*n* designate a position or candidate solution in the search-space. The basic RS algorithm can then be described as:
* Initialize **x** with a random position in the search-space.
* Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
  + Sample a new position **y** from the hypersphere of a given radius surrounding the current position **x** (see

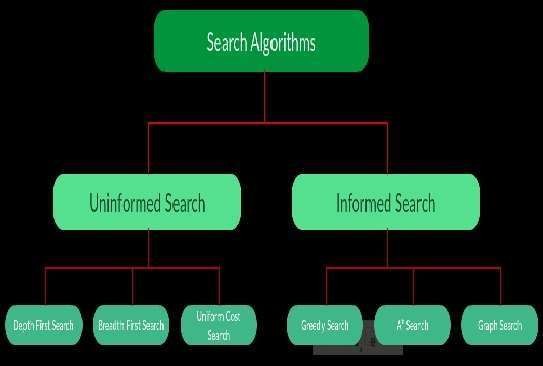
e.g. Marsaglia's technique for sampling a hypersphere.)

* + If *f*(**y**) < *f*(**x**) then move to the new position by setting **x** = **y**





#### TYPES OF SEARCH ALGORITHMS

****

* 1. **Brute-Force Search Strategies (Uninformed / Blind)**
* Breadth-First Search
* Depth-First Search
* Bidirectional Search
* Uniform Cost Search
* Iterative Deepening Depth-First Search
  1. **Informed (Heuristic) Search Strategies**
* Pure Heuristic Search (Open and Closed List)
* Greedy Best First Search
* A \* Search
  1. **Local Search Algorithms**
* Hill-Climbing Search
* Local Beam Search
* Simulated Annealing
* Travelling Salesman Problem

UNINFORMED SEARCH ALGORITHMS

* The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called **Blind search**.
* The following uninformed search algorithms are discussed in this section.

1. Depth First Search
2. Breath First Search
3. Uniform Cost Search

* Each of these algorithms will have:
* A problem **graph,** containing the start node S and the goal node G.
* A **strategy,** describing the manner in which the graph will be

traversed to get to G .

* A **fringe,** which is a data structure used to store all the possible states

(nodes) that you can go from the current states.

* A **tree,** that results while traversing to the goal node.
* A solution **plan,** which the sequence of nodes from S to G.

**UNINFORMED SEARCH STRATEGIES**

* **Uninformed Search Strategies** have no additional information

about states beyond that provided in the **problem definition**.

* **Strategies** that know whether one non goal state is "more promising" than another are called I**nformed search or heuristic search** strategies.
* There are six uninformed search strategies as given below.
  1. Breadth-first search
  2. Uniform-cost search
  3. Depth-first search
  4. Depth-limited search
  5. Iterative deepening search
  6. Bidirectional Search

**BREADTH FIRST SEARCH (BFS)**

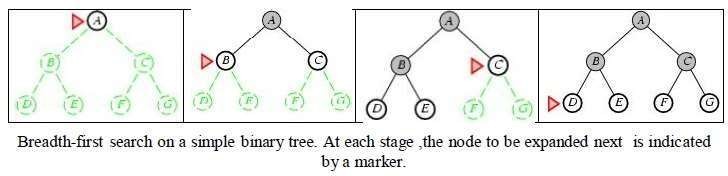
* Breadth-first search in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on.
* In general, all the nodes are expanded at a given depth in the search

tree before any nodes at the next level are expanded.

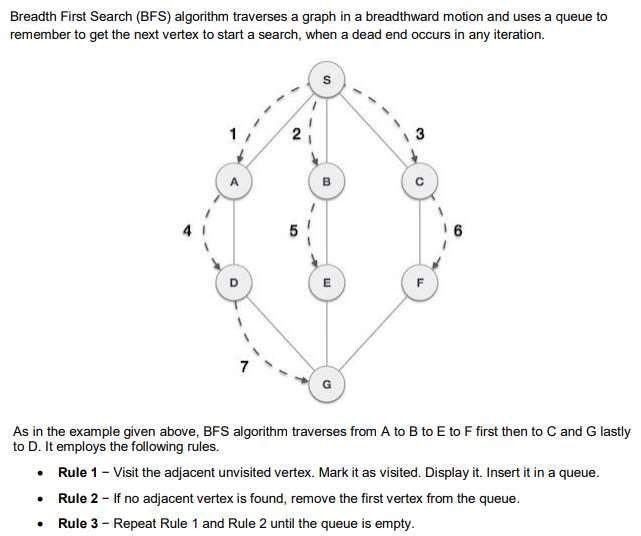
* Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out(FIFO) queue, assuring that the nodes that are visited first will be expanded first.
* In otherwards, calling TREE-SEARCH (problem, FIFO-QUEUE())

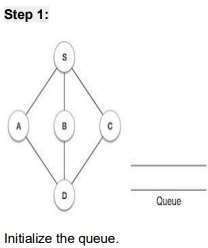
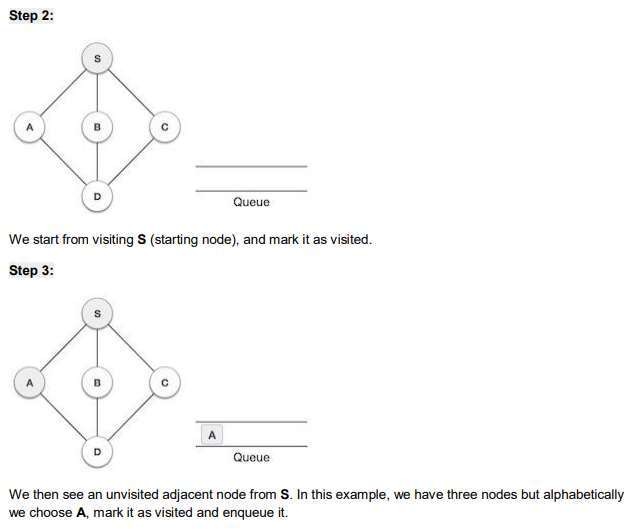
results in breadth-first-search.

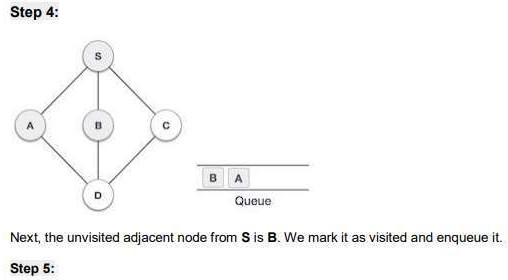
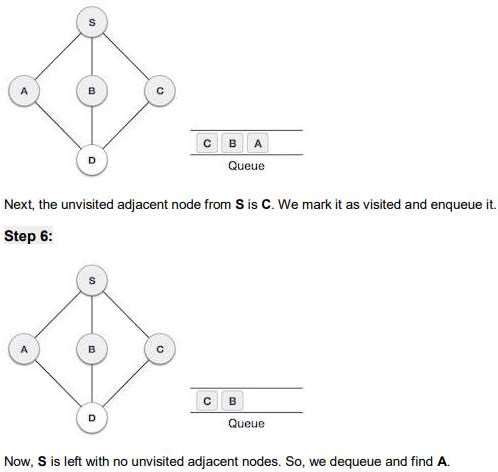
* The FIFO queue puts all newly generated successors at the end of the queue, which means that Shallow nodes are expanded before deeper nodes.

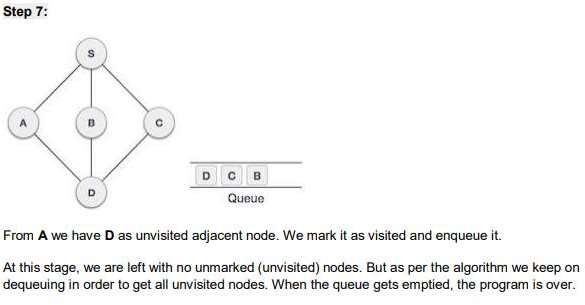


**BFS - Example**

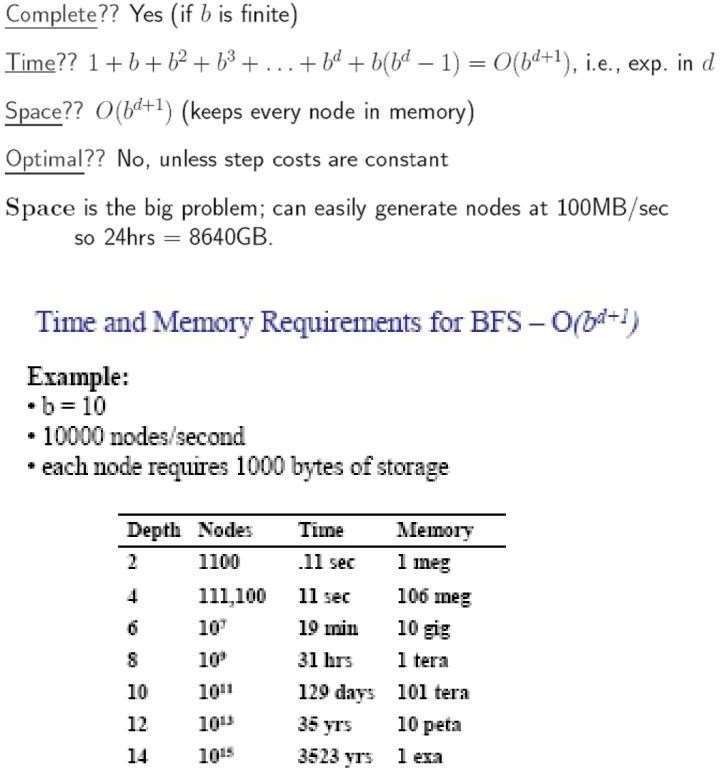








Properties of BFS



Time complexity for BFS

* Assume every state has b successors.
* The root of the search tree generates b nodes at the first level,each of

which generates b more nodes,for a total of b2 at the second level.

* Each of these generates b more nodes,yielding b3 nodes at the third

level,and so on.

* Now suppose,that the solution is at depth d.
* In the worst case,we would expand all but the last node at level d,generating bd+1 - b nodes at level d+1.
* Then the total number of nodes generated is

b + b2 + b3 + …+ bd + (bd+1 + b) = O(bd+1).

* Every node that is generated must remain in memory,because it is

either part of the fringe or is an ancestor of a fringe node.

* The space compleity is,therefore ,the same as the time complexity

#### DEPTH FIRST SEARCH (DFS)

* + Depth-first-search always expands the deepest node in the current

fringe of the search tree.

* + The progress of the search is illustrated in figure.
  + The search proceeds immediately to the deepest level of the search

tree, where the nodes have no successors.

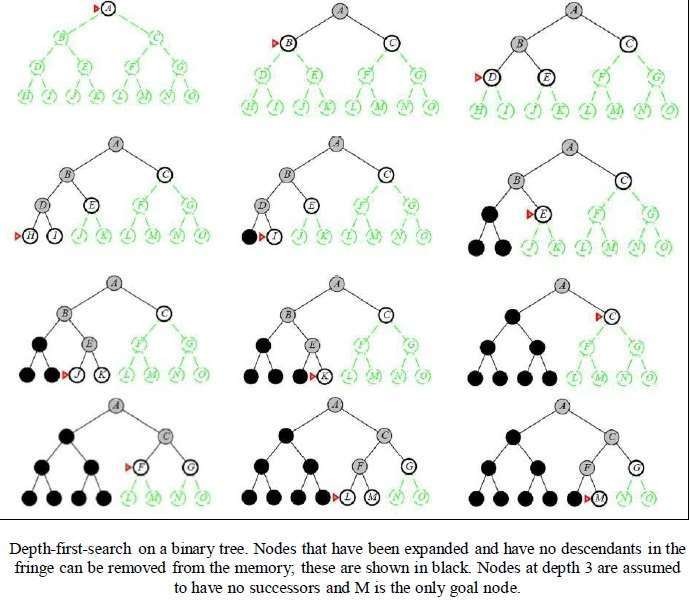
* + As those nodes are expanded, they are dropped from the fringe,
  + so then the search "backs up" to the next shallowest node that still has unexplored successors.
  + This strategy can be implemented by TREE-SEARCH with a last-in-

first- out (LIFO) queue, also known as a stack.

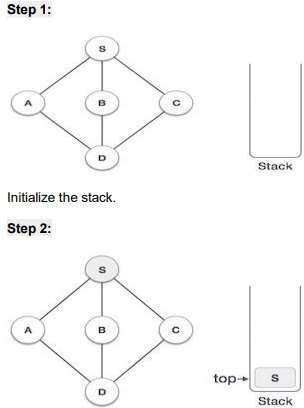
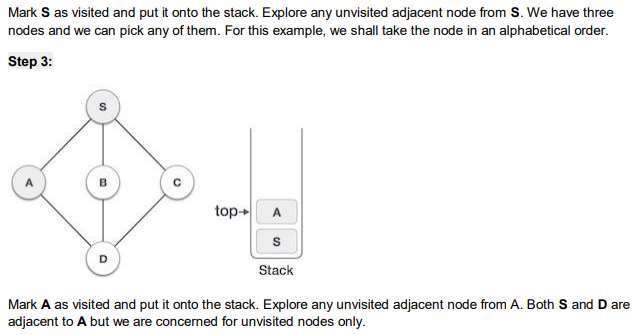
* + Depth-first-search has very modest memory requirements.
  + It needs to store only a single path from the root to a leaf node along with the remaining unexpanded sibling nodes for each node on the path.
  + Once the node has been expanded, it can be removed from the

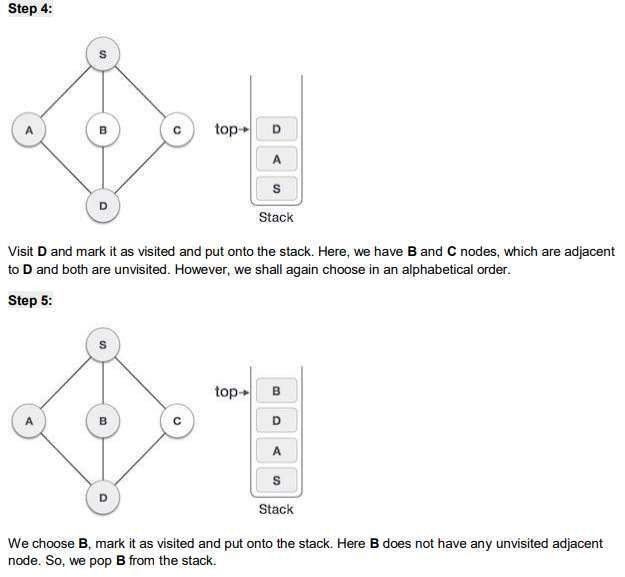
memory, as soon as its descendants have been fully explored.

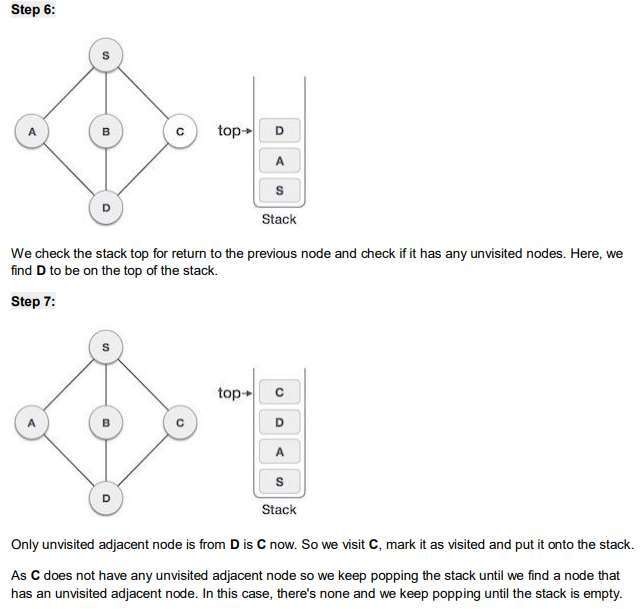
* + For a state space with a branching factor b and maximum depth m, depth- first-search requires storage of only bm + 1 nodes.



# DFS - Example

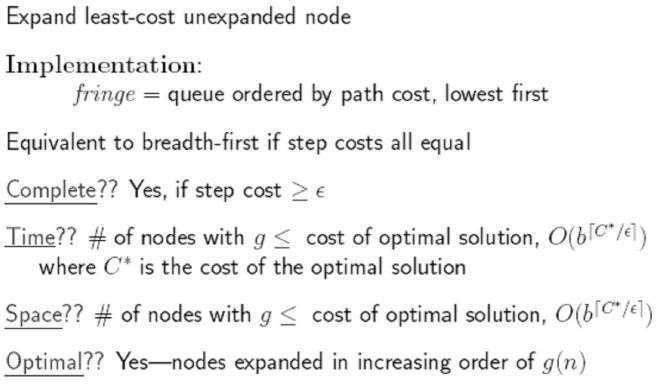






|  |  |  |
| --- | --- | --- |
| **Parameters** | **BFS** | **DFS** |
| **Data structure** | Queue (FIFO) | Stack (LIFO) |
| **Source** | It is better when target is closer to source | It is better when target is far from source |
| **Speed** | Slower | Faster |
| **Time Complexity** | O(V+E) | O(V+E) |
| **Suitability for Decision Trees** | BFS considers all neighbor, so it is not suitable for decision tree | Suitable for decision tree |

UNIFORM-COST SEARCH

* Instead of expanding the shallowest node, **uniform-cost search** expands the node n with the lowest path cost.
* uniform-cost search does not care about the number of steps a path has, but only about their total cost.
* **Properties of Uniform-cost-search:**

**INFORMED SEARCH ALGORITHMS**

* Here, the algorithms have information on the goal state, which helps in more efficient searching. This information is obtained by something called a *heuristic.*
* In this section, we will discuss the following search algorithms.

1. Greedy Search
2. A\* Tree Search
3. A\* Graph Search

* **Search Heuristics:** In an informed search, a heuristic is a *function* that estimates how close a state is to the goal state. For examples – Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.)

**INFORMED SEARCH AND EXPLORATION**

**Informed (Heuristic) Search Strategies**

* **Informed search strategy** is one that uses problem-specific knowledge beyond the definition of the problem itself.
* It can find solutions more efficiently than uninformed strategy.

**Best-first search**

* **Best-first search** is an instance of general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function** f(n).
* The node with lowest evaluation is selected for expansion, because the evaluation measures the distance to the goal.
* This can be implemented using a priority-queue, a data structure that will maintain the fringe in ascending order of f- values.

**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.
* The key component of Best-first search algorithm is a

**heuristic function**, denoted by h(n).

h(n) = estimated cost of the **cheapest path** from node n to a **goal node**.

* Heuristic function are the most common form in which additional knowledge is imparted to the search algorithm.

INFORMED SEARCH ALGORITHMS

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

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

**Admissibility of the heuristic function is given as:**

h(n) <= h\*(n)

Here h(n) is heuristic cost, and h\*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

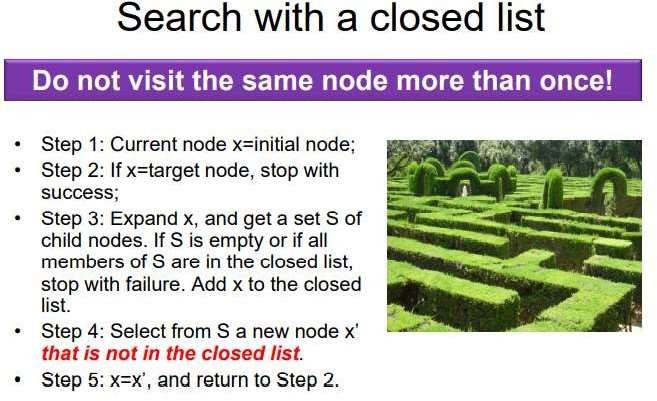
* 1. **Search with closed and open list**

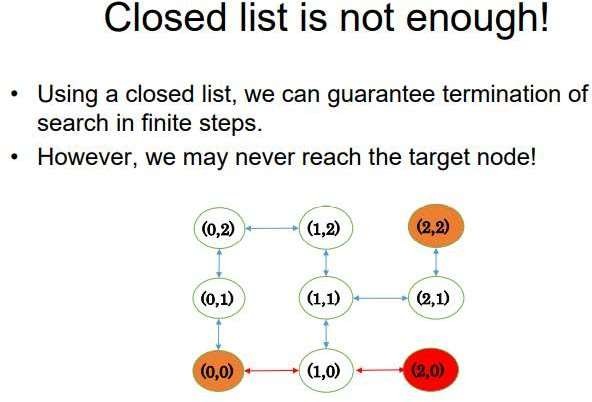
Pure Heuristic Search

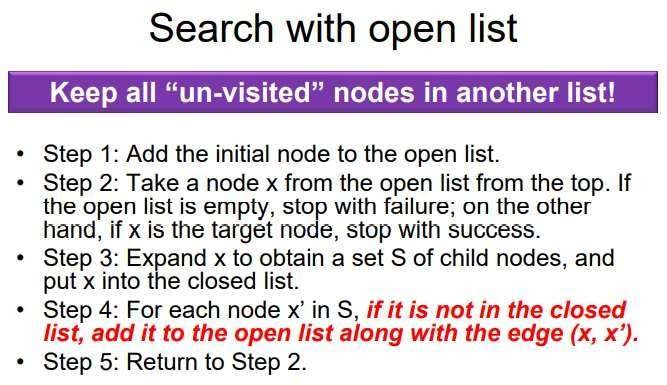
* It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.
* In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

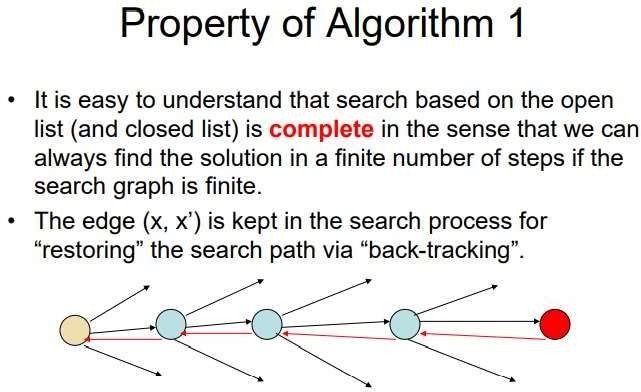
**Pure Heuristic Search:**

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









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

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

f(n)= g(n) + h(n) 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:**

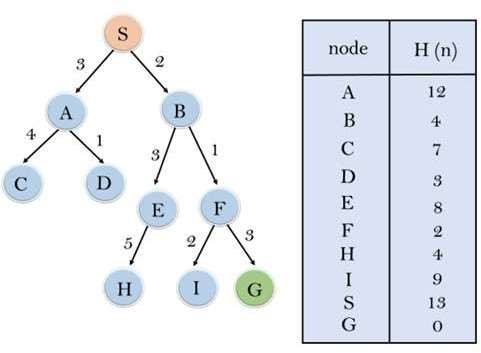
* It can behave as an unguided depth-first search in the worst case

scenario.

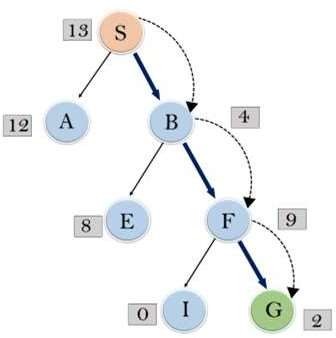
* It can get stuck in a loop as DFS.
* This algorithm is not optimal.

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]

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

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

* Hence the final solution path will be: **S----> B----->F----> G**
* **Time Complexity:** The worst case time complexity of Greedy best

first search is O(bm).

* **Space Complexity:** The worst case space complexity of Greedy best first search is O(bm). Where, m is the maximum depth of the search space.
* **Complete:** Greedy best-first search is also incomplete, even if the

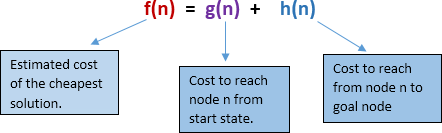
given state space is finite.

* **Optimal:** Greedy best first search algorithm is not optimal.

3.) 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**.



**A\* Tree Search**

* A\* Tree Search, or simply known as A\* Search, combines the strengths of uniform-cost search and greedy search. In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in greedy search, denoted by h(x). The summed cost is denoted by f(x).
* **Heuristic:** The following points should be noted wrt heuristics in A\* search. **f(x)=g(x)+h(x)**
* Here, h(x) is called the **forward cost**, and is an estimate of the distance of the current node from the goal node. And, g(x) is called the **backward cost**, and is the cumulative cost of a node from the root node. A\* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h\*(x) to reach the goal. This property of A\* heuristic is called **admissibility**.

Admissibility: 0<=h(x)<=h\*(x)

* **Strategy:** Choose the node with lowest f(x) value.

**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:**

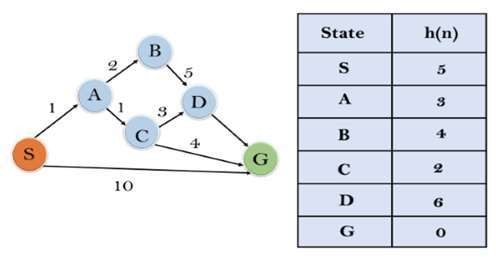
* A\* search algorithm is the best algorithm than other search algorithms.
* A\* search algorithm is optimal and complete.
* This algorithm can solve very complex problems.

**Disadvantages:**

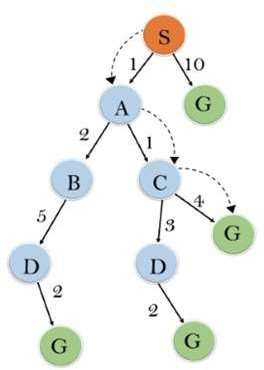
* It does not always produce the shortest path as it mostly based on heuristics and approximation.
* A\* search algorithm has some complexity issues.
* The main drawback of A\* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

**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--> A, 4), (S-->G, 10)}

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

**Iteration3:** {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S-

-> A-->B, 7), (S-->G, 10)}

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

**Points to remember:**

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

**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)**

**GAME PLAYING**

Game Playing is one of the oldest sub-fields in AI. Game playing involves abstract and pure form of competition that seems to require intelligence. It is easy to represent the states and actions. To implement the game playing very little world knowledge is required.

* The most common used AI technique in game is search. Game playing research has contributed ideas on how to make the best use of time to reach good decisions.
* Game playing is a search problem defined by:
* Initial state of the game
* Operators defining legal moves
* Successor function
* Terminal test defining end of game states
* Goal test
* Path cost/utility/payoff function

Single-player games

* Sudoku
* Crossword

Single-agent-path-finding Problems

* 3X3 eight-tile
* 4X4 fifteen-tile
* 5X5 twenty four tile puzzles
* Travelling Salesman Problem
* Rubik’s Cube
* Theorem Proving
* More popular games are too complex to solve, requiring the program to take its best guess. “ for example in chess, the search tree has 1040 nodes (with branching factor of 35). It is the opponent because of whom uncertainty arises.
* Characteristics of game playing
  + 1. There are always an “unpredictable” opponent:
  + The opponent introduces uncertainty
  + The opponent also wants to win

The solution for this problem is a strategy, which specifies

a move for every possible opponent reply.

* + 1. Time limits:

Game are often played under strict time constraints

(eg:chess) and therefore must be very effectively handled.

**Types of games**

* There are basically two types of games

Deterministic games

Chance games

* Game like chess and checker are perfect information deterministic games whereas games like scrabble and bridge are imperfect information. We will consider only two player discrete, perfect information games, such as tic-tac-toe, chess, checkers etc... . Two- player games are easier to imagine and think and more common to play.
* **Minimize search procedure**
* Typical characteristic of the games is to look ahead at future position in order to succeed. There is a natural correspondence between such games and state space problems.
* In a game like tic-tac-toe

States-legal board positions Operators-legal moves Goal-winning position

* The game starts from a specified initial state and ends in position that can be declared win for one player and loss for other or possibly a draw. Game tree is an explicit representation of all possible plays of the game. We start with a 3 by 3 grid..

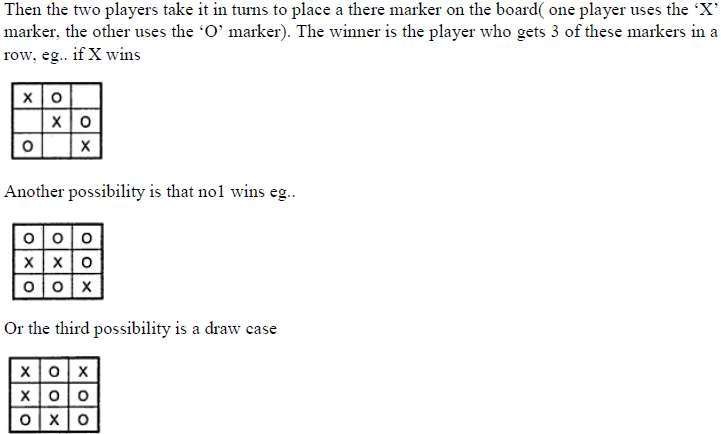
**Search tree for tic-tac-toe**

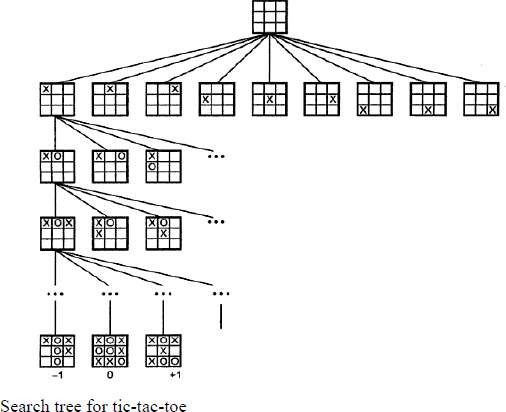
* The root node is an initial position of the game. Its successors are the positions that the first player can reach in one move; their successors are the positions resulting from the second player's replies and so on. Terminal or leaf nodes are presented by WIN, LOSS or DRAW. Each path from the root ro a terminal node represents a different complete play of the game. The moves available to one player from a given position can be represented by OR links whereas the moves available to his opponent are AND links.
* The trees representing games contain two types of nodes:

1. MAX- nodes (assume at even level from root)
2. MIN - nodes [assume at odd level from root)

* The leaves nodes are labeled WIN, LOSS or DRAW depending on whether they represent a win, loss or draw position from Max‟s viewpoint. Once the leaf nodes are assigned their WIN-LOSS or DRAW status, each nodes in the game tree can be labeled WIN, LOSS or DRAW by a bottom up process.
* Game playing is a special type of search, where the intention of all

players must be taken into account.





**Part-A:**

# UNIT – III

**Probabilistic Reasoning**

* Probability
* Conditional probability
* Bayes Rule
* Bayesian Networks- representation
* Construction and inference

**Part-B:**

* Temporal Model
* Hidden Markov Model

**Probabilistic reasoning in Artificial Intelligence**

**Uncertainty:**

* Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called **uncertainty**.
* So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

**Causes of Uncertainty:**

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources
2. Experimental Errors
3. Equipment fault
4. Temperature variation
5. Climate change

### Probabilistic Reasoning

* Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.
* We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.
* In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

#### Need of probabilistic reasoning in AI:

* When there are unpredictable outcomes.
* When specifications or possibilities of predicates becomes too large to handle.
* When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

#### Bayes' rule

* **Bayesian Statistics**

**Probability**

* Probability can be defined as a chance that an uncertain event will occur.
* It is the numerical measure of the likelihood that an event will occur.
* The value of probability always remains between 0 and 1 that represent ideal uncertainties.
* Probability implies 'likelihood' or 'chance'.
* When an event is certain to happen then the probability of occurrence of that event is 1 and when it is certain that the event cannot happen then the probability of that event is 0.
* Hence the value of probability ranges from 0 to 1. Probability has been defined in a varied manner by various schools of thought.

**Classical Definition of Probability**

* As the name suggests the classical approach to defining probability is the oldest approach. It states that if there are n exhaustive, mutually exclusive and equally likely cases out of which m cases are favorable to the happening of event A.

**Example**

* **Problem Statement:**

A coin is tossed. What is the probability of getting a head?

* **Solution:**

Total number of equally likely outcomes (n) = 2 (i.e. head or tail)

Number of outcomes favorable to head (m) = 1

* 0 ≤ P(A) ≤ 1, nt A.

where P(A) is the probability of an eve

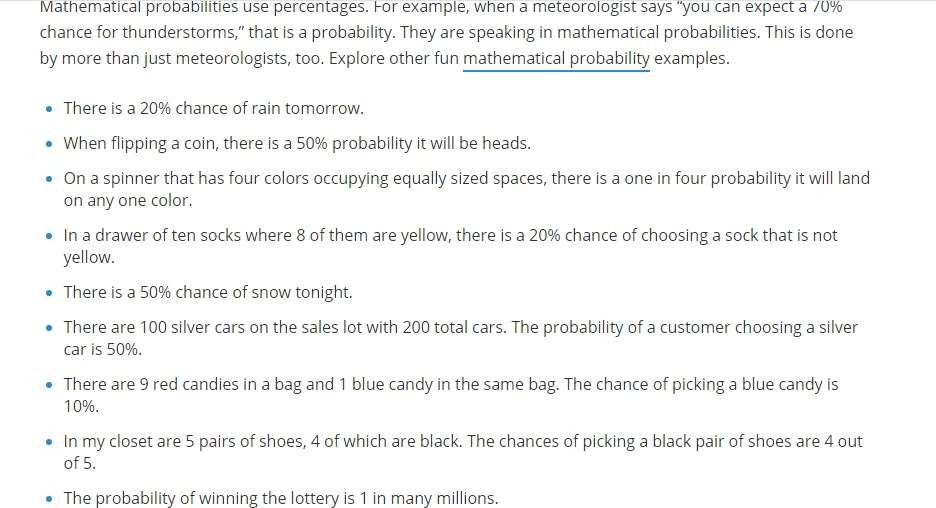
* P(A) = 0, indicates total uncertainty in an event A.
* P(A) =1, indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

* Probability of Occurrence = (Number of desired Outcomes / Total No of Outcomes)
* P(¬A) = probability of a not happening event.
* P(¬A) + P(A) = 1.

#### Terminologies in Probability Theory

* **Event:** Each possible outcome of a variable is called an event.
* **Sample space:** The collection of all possible events is called sample space.
* **Random variables:** Random variables are used to represent the events and objects in the real world.
* **Prior probability:** The prior probability of an event is probability computed before observing new information.
* **Posterior Probability:** The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.



#### Basic Probability Rules

* Probability Rule One (For any event A, 0 ≤ P(A) ≤ 1)
* Probability Rule Two (The sum of the probabilities of all possible outcomes is 1)
* Probability Rule Three (The Complement Rule)
* Probabilities Involving Multiple Events.
* Probability Rule Four (**Addition** Disjoint Events)
* Finding P(A and B) using Logic. Rule for

#### How is probability used in real life?

* You **use probability** in **daily life** to make decisions when you don't know for sure what the outcome will be. Most of the time, you won't perform actual **probability** problems, but you'll **use** subjective **probability** to make judgment calls and determine the best course of action.

#### What is a certain event?

* A **certain event** is an **event** that is sure to happen. E is a **certain event** if and only if P(E)= 1. Example. In flipping a coin once, a **certain event** would be getting a head or a tail.
* The probability formula provides the ratio of the number of favorable outcomes to the total number of possible outcomes. The

Probability of an Event = favorable outcomes) (Total(Number of number of

possible outcomes) P(A) = n(E) / n(S)

#### What is impossible event?

* An **impossible event** is an **event** that cannot happen. E is an **impossible event** if and only if P(E) = 0. Example. In flipping a coin once, an **impossible event** would be getting BOTH a head AND a tail.
* The **probability line** is shows probabilities a line that and how

these probabilities relate to each other. Since the probability of an event is a number from 0 to 1, we can use the probability line above to show the possible ranges of probability values.

#### Four perspectives on probability are commonly used:

* Classical (sometimes called "A priori" or "Theoretical") ...
* Empirical (sometimes called "A posteriori" or "Frequentist") ...
* Subjective. ...
* Axiomatic.

#### What are the different types of probability distributions?

* There are many different classifications of probability distributions. Some of them include the normal distribution, chi **square** distribution, binomial distribution, and Poisson distribution.

#### What are probability models?

* A **probability model** is a mathematical representation of a random phenomenon. It is defined by its sample space, events within the sample space, and **probabilities** associated with each event. The sample space S for a **probability model** is the set of all possible outcomes.

#### Who is known as father of probability?

* A gambler's dispute in 1654 led to the creation of a mathematical theory of probability by two famous French mathematicians, Blaise Pascal and Pierre de Fermat.

#### What are the 3 axioms of probability?

* For any event A, P(A) ≥ 0. In English, that's “For any event A, the **probability** of A is greater or equal to 0”.
* When S is the sample space of an experiment; i.e., the set of all possible outcomes, P(S) = 1.
* If A and B are mutually exclusive outcomes, P(A 𝖴 B ) = P(A) + P(B).

#### Conditional Probability

* Conditional probability is a probability of occurring an event when another event has already happened.
* Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:



Where P(*A*⋀*B*)= Joint probability of A and B P(B)= Marginal probability of B.

If the probability of A is given and we need to find the probability of B, then it will be given as:



It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of P(A⋀B) by P( B ).



* **Conditiona probability** is the **probability** of one event occurring with some relationship to one or more other events. **For example:** Event A is that it is raining outside, and it has a 0.3 (30%) chance of raining today. Event B is that you will need to go outside, and that has a probability of 0.5 (50%).

#### What is the difference between probability and conditional probability?

* Their only difference is that the conditional probability assumes that we already know something -- that B is true.

#### How do you proportions?

* + The analog

#### calculate conditional

of **conditional**

**proportion** is **conditional** probability: P(A|B) means “probability that A happens, if we know that B happens”. The **formula** is P(A|B) = P(A and B)/P(B).

#### Independent Events

* + Events can be "[**Independent**](https://www.mathsisfun.com/data/probability-events-independent.html)", meaning each event is **not affected** by any other events.

#### Example: Tossing a coin.

* + Each toss of a coin is a perfect isolated thing.
  + What it did in the past will not affect the current toss.
  + The chance is simply 1-in-2, or 50%, just like ANY toss of the coin.
  + So each toss is an **Independent Event**.

Dependent Events

* + But events can also be "dependent" ... which means they **can be affected by previous events.**

Example: Marbles in a Bag

* + 2 blue and 3 red marbles are in a bag.
  + What are the chances of getting a blue marble?
  + The chance is **2 in 5**
  + **But after taking one out** the chances change!
  + So the next time:
  + if we got a **red** marble before, then the chance of a blue marble next is **2 in 4**
  + if we got a **blue** marble before, then the chance of a blue marble next is **1 in 4**

#### Bayes Rule

* + Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.
  + In probability theory, it relates the conditional probability and marginal probabilities of two random events.
  + Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.
  + It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).
  + Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.
  + **Example**: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.
  + Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:
  + As from product rule we can write:

P(A ⋀ B)= P(A|B) P(B) or

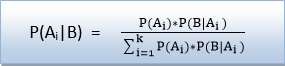
Similarly, the probability of event B with known event A:

P(A ⋀ B)= P(B|A) P(A)

Equating right hand side of both the equations, we will get:



* + - The above equation (a) is called as Bayes' rule or Bayes' theorem. This equation is basic of most modern AI systems for probabilistic inference.
    - It shows the simple relationship between joint and conditional probabilities. Here,
    - P(A|B) is known as posterior, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.
    - P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.
  + P(A) is called the prior probability, probability of hypothesis before considering the evidence
  + P(B) is called marginal probability, pure probability of an evidence.
  + In the equation (a), in general, we can writeP (B) = P(A)\*P(B|Ai), hence the Bayes' rule can be written as:



Where A1, A2, A3,........, An is a set of mutually exclusive and exhaustive events.

#### Applying Bayes' Rule:

Bayes' rule allows us to compute the single term P(B|A) in terms of P(A|B), P(*B*), and P(A). This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:



* + **Example-1:**

**Question: what is the probability that a patient has diseases meningitis**

**with a stiff neck?**

**Given Data:**

* + A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:
  + The Known probability that a patient has meningitis disease is 1/30,000.
  + The Known probability that a patient has a stiff neck is 2%.
  + Let a be the proposition that patient has stiff neck and b be the proposition

that patient has meningitis. , so we can calculate the following as:

* + P(a|b) = 0.8
  + P(b) = 1/30000
  + P(a)= 0.02



Hence, we can assume that 1 patient out of 750 patients has

meningitis disease with a stiff neck.

**Example-2:**

**Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.**

**Solution:**



P(king): probability that the card is King= 4/52= 1/13 P(face): probability that a card is a face card= 3/13 P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

#### Application of Bayes' theorem in Artificial Intelligence:

* + It is used to calculate the next step of the robot when the already executed step is given.
  + Bayes' theorem is helpful in forecasting.
  + It can solve the Monty Hall problem.weather

#### Bayesian Belief Network

* + Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

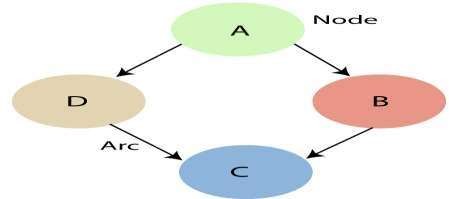
* + It is also called a **Bayes network, belief network, decision network**, or **Bayesian model**.
  + Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection.
  + Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction**, and **decision making under uncertainty**.
  + Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

**Directed Acyclic Graph**

**Table of conditional probabilities.**

* + The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram**.

**A Bayesian network graph is made up of nodes and Arcs (directed links), where:**



* + Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
  + **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph. These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
  + In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.
  + If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
  + Node C is independent of node A.
  + The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a **directed acyclic graph or DAG.**
  + The Bayesian network has mainly two components:

1. Causal Component
2. Actual numbers

Each node in the Bayesian network has condition probability distribution P(Xi |Parent(Xi) ), which determines the effect of the parent on that node.

Bayesian network is based on Joint probability

Distribution and conditional probability. understand the joint probability distribution:

**Joint probability distribution:**So let's first

* + If we have variables x1, x2, x3,....., xn, then the probabilities of a different combination of x1, x2, x3.. xn, are known as Joint probability distribution.

P[x1, x2, x3,....., xn], it can be written as the following way in terms of the joint probability distribution.

= P[x1| x2, x3,....., xn]P[x2, x3,....., xn]

= P[x1| x2, x3,....., xn]P[x2|x3,....., xn]. P[xn-

1|xn]P[xn].

Explanation of Bayesian Network:

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

**Example:** Harry installed a new burglar alarm at his home

To detect burglary. The detecting a burglary but

Alarmalso

reliably responds at responds for for minor

earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

* + **Problem:**

**Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake**

**occurred, Harry.**

* + **Solution:**

**And David and Sophia both called the**

* + The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
  + The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
  + The conditional distributions for each node are given as conditional probabilities table or CPT.
  + Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
  + In CPT, a boolean variable with k boolean parents contains 2K probabilities. Hence, if there are two parents, then CPT will contain 4 probability values.
  + **List of all events occurring in this network:**
    - **Burglary (B)**
    - **Earthquake(E)**
    - **Alarm(A)**
    - **David Calls(D)**
    - **Sophia calls(S)**
  + We can write the events of problem statement in the form of probability: **P[D, S, A, B, E]**, can rewrite the above probability statement using joint probability distribution:

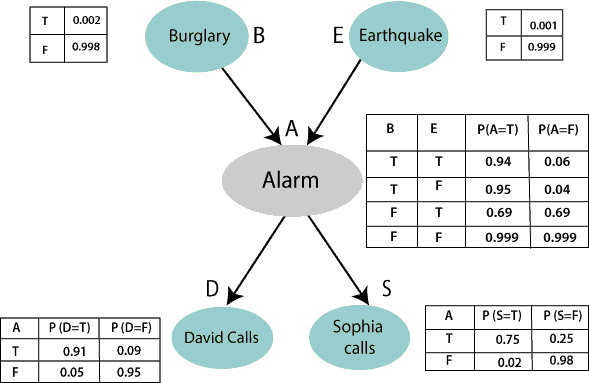
P[D, S, A, B, E]= P[D | S, A, B, E]. P[S, A, B, E]

**=P[D | S, A, B, E]. P[S | A, B, E]. P[A, B, E]**

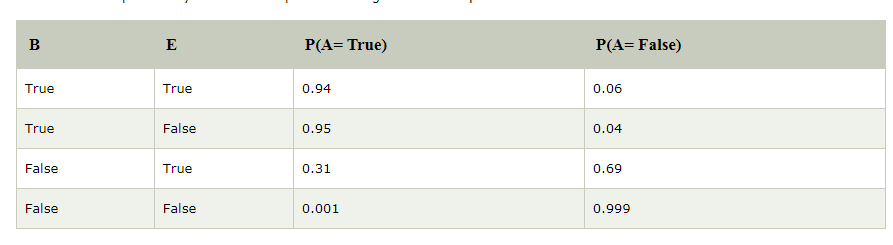
= P [D| A]. P [ S| A, B, E]. P[ A, B, E]

**= P[D | A]. P[ S | A]. P[A| B, E]. P[B, E]**

**= P[D | A ]. P[S | A]. P[A| B, E]. P[B |E]. P[E]**



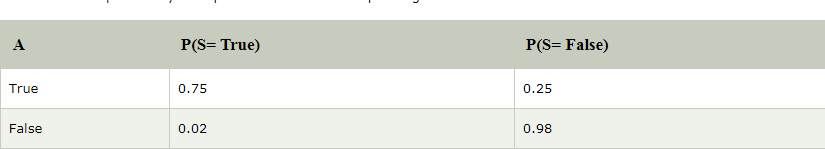
* + Let's take the observed probability for the Burglary and earthquake component:
  + P(B= True) = 0.002, which is the probability of burglary.
  + P(B= False)= 0.998, which is the probability of no burglary.
  + P(E= True)= 0.001, which is the probability of a minor earthquake
  + P(E= False)= 0.999, Which is the probability that an earthquake not occurred.
  + We can provide the conditional probabilities as per the below tables:
  + **Conditional probability table for Alarm A:**

The Conditional probability of Alarm A depends on Burglar and earthquake:

#### Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

#### Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm”.

From the formula of joint distribution, we can write the problem

statement in the form of probability distribution:

**P(S, D, A, ¬B, ¬E) = P (S|A) \*P (D|A)\*P (A|¬B ^ ¬E) \*P (¬B) \*P**

**(¬E).**

= 0.75\* 0.91\* 0.001\* 0.998\*0.999

**= 0.00068045.**

**Hence, a Bayesian network can answer any query about the domain by using Joint distribution.**

The semantics of Bayesian Network

* + There are two ways to understand the semantics of the Bayesian network, which is given below:

|  |  |  |
| --- | --- | --- |
| 1. **To understand** | **the** | **network as the** |
| **representation of** | **the** | **Joint probability** |
| **distribution.** |  |  |

It is helpful to understand how to construct the network.

2. To understand the network as an encoding of a collection of conditional independence statements.

It is helpful in designing inference procedure.

**Types Probability Models**

* + Bayes’ Net
  + Temporal Probability Model
  + Dynamic Bayes’ Net (DBN)

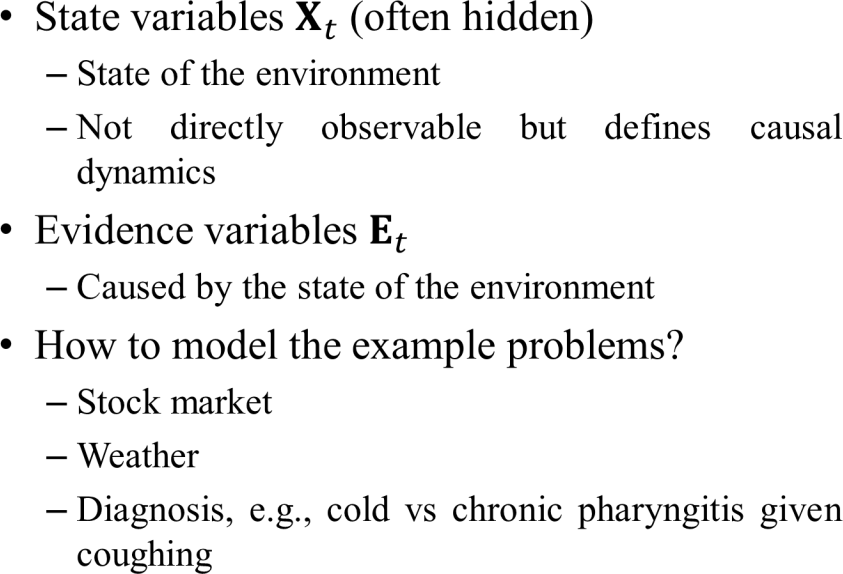
Special classes of DBN

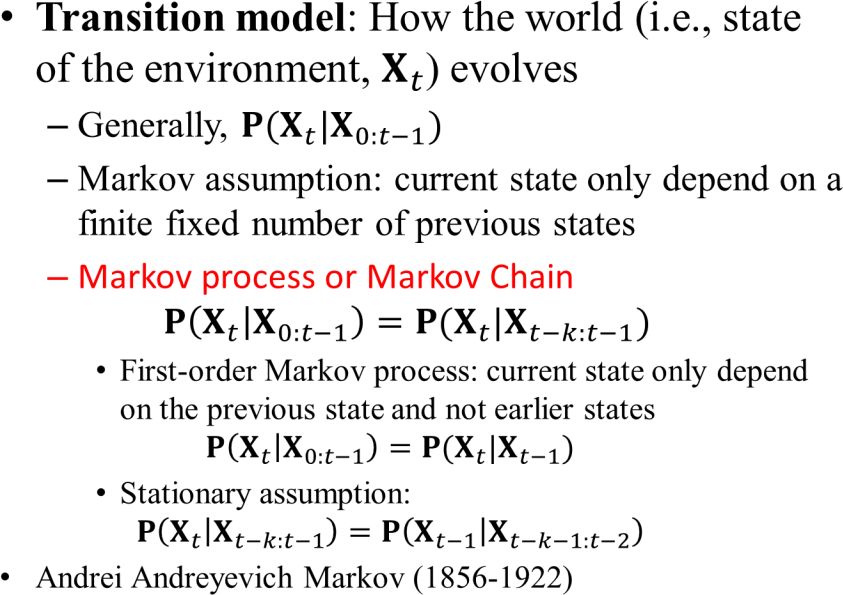
* Hidden Markov Model (HMM)
* Kalman Filter

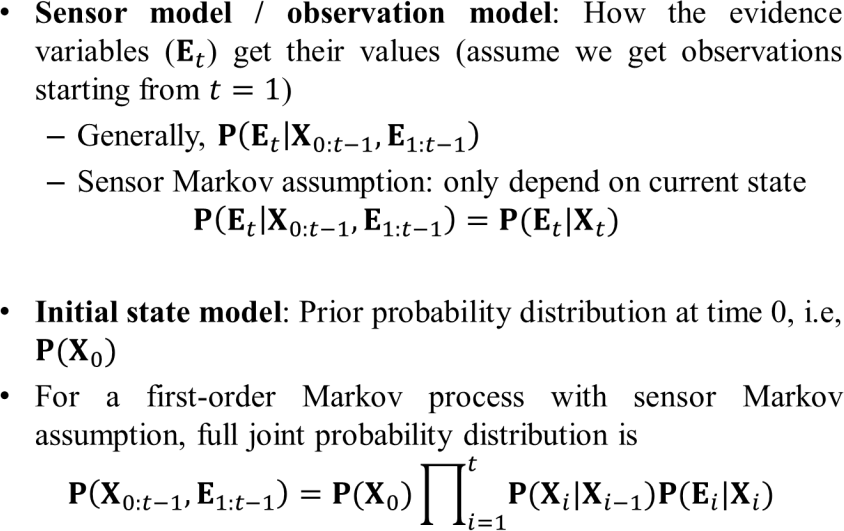
**Temporal Probabilistic Model**

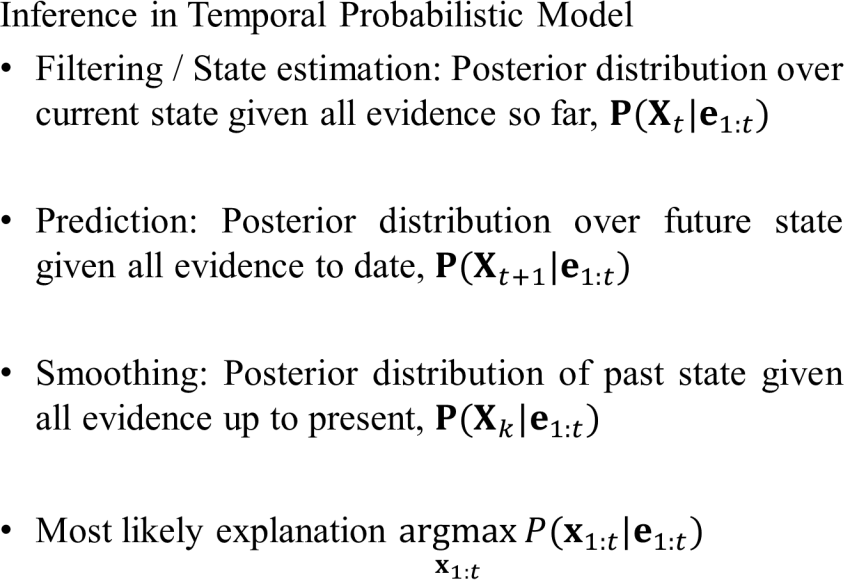
* + Why do we need temporal probabilistic model?
* The world changes over time, and what happens now impacts what will happen in the future
  + Stock market
  + Weather
* Sometimes world state become clearer as more evidence is collected over time
  + Diagnosis, e.g., cold vs chronic pharyngitis given coughing
  + How to model the time?

– View the world as time slices: discrete time steps









#### Introduction to Hidden Markov Models

* + **Hidden Markov models** are generative **models**, in which the joint distribution of observations and **hidden** states, or equivalently both the prior distribution of **hidden** states (the transition probabilities) and conditional distribution of observations given states (the emission probabilities), is modeled.
  + What is hidden Markov model used for?

A **hidden Markov model** (**HMM**) is a statistical **model** that can be **used** to describe the evolution of observable events that depend on internal factors, which are not directly observable. We call the observed event a `symbol' and the invisible factor underlying the observation a `state'.

* + What is hidden Markov model in artificial intelligence?

A **hidden Markov model** (**HMM**) is an

Augmentation observations.ofthe**Markov chain**to include Just like the state transition of

the **Markov chain**, an **HMM** also includes observations of the state. ... The observations are modeled using the variable Ot for each time t whose domain is the set of possible observations.

* + How do hidden Markov models work?

**Hidden Markov Models** (HMMs) **are** a class of probabilistic graphical **model** that allow us **to** predict a sequence of unknown (**hidden**) variables from a set of observed variables. A simple example of an **HMM** is predicting the weather (**hidden** variable) based on the type of clothes that someone wears (observed).

* + Are hidden Markov models still used?

In the last five decades, various researchers explored the HMM and its variant in various application domains. In 1970s, HMM has been applied in speech recognition. Since 1980, HMM has been extensively used in the domain of bioinformatics

* + What is the difference between Markov model and hidden Markov model?

**Markov model** is a state machine with the state changes being probabilities. **In a hidden Markov model**, you don't know the probabilities, but you know the outcomes.

* + What is hidden in hidden Markov model?

**Hidden Markov Model** (**HMM**) is a statistical **Markov model** in which the system being modeled is assumed to be a **Markov process** – call it – with unobservable ("**hidden**") states. **HMM** assumes that there is another **process** whose behavior "depends" on . The goal is to learn about by observing .

* + What are the main issues of hidden Markov model?

It is mostly used in speech recognition, to some extent it is also applied for classification task. **HMM** provides solution of three **problems** : evaluation, decoding and learning to find most likelihood classification.

* + How does Markov model work?

In probability theory, a **Markov model** is a stochastic **model** used to **model** randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it (that is, it assumes the **Markov** property).

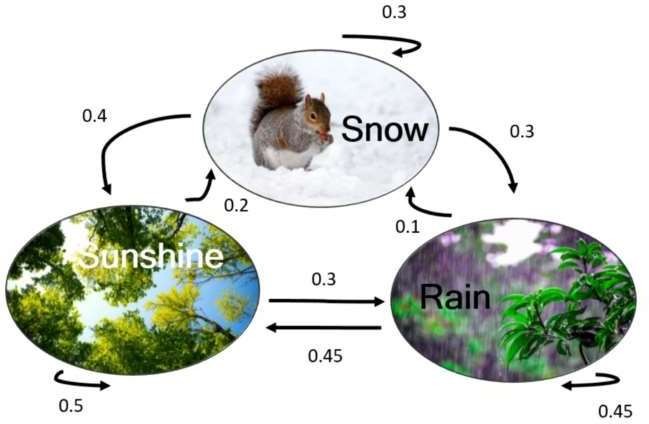
Markov Chains

* + Let us first give a brief introduction to Markov Chains, a type of a random process. We begin with a few “states” for the chain, {*S*₁,…,*S*ₖ}; For instance, if our chain represents the daily weather, we can have

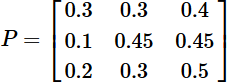
{Snow, Rain, Sunshine}. The property a process (*X*ₜ)ₜ should have to be a Markov Chain is:



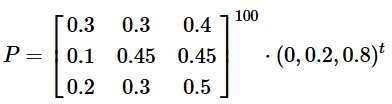
* + In words, the probability of being in a state *j* depends only on the previous state, and not on what happened before.
  + Markov Chains are often described by a graph with transition probabilities, i.e, the probability of moving to state *j* from state *i*, which are denoted by *p*ᵢ,ⱼ. Let’s look at the following example:



* + The chain has three states; For instance, the transition probability between Snow and Rain is 0.3, that is — if it was snowing yesterday, there is a 30% chance it will rain today. The transition probabilities can be summarized in a matrix:



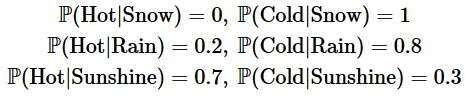
* + Notice that the sum of each row equals 1 (think why). Such a matrix is called a Stochastic Matrix. The (i,j) is defined as pᵢ,ⱼ -the transition probability between i and j.
  + Fact: if we take a power of the matrix, Pᵏ, the (i,j) entry represents the probability to arrive from state i to state j at k steps.
  + In many cases we are given a vector of initial probabilities *q*=(*q*₁,…,*q*ₖ) to be at each state at time *t*=0. Thus, the probability to be at state *i* at time *t* will be equal to the *i-*th entry of the vector *P*ᵏ*q*.
  + For instance, if today the probabilities of snow, rain and sunshine are 0,0.2,0.8, then the probability it will rain in 100 days is calculated as follows:



* + The 2nd entry equals ≈ 0.44.

Hidden Markov Model

* + In a Hidden Markov Model (HMM), we have an invisible Markov chain (which we cannot observe), and each state generates in random one out of *k* observations, which are visible to us.
  + Let’s look at an example. Suppose we have the Markov Chain from above, with three states (snow, rain and sunshine), *P* - the transition probability matrix and *q* — the initial probabilities. This is the invisible Markov Chain — suppose we are home and cannot see the weather. We can, however, feel the temperature inside our room, and suppose there are two possible observations: hot and cold, where:



**Basic Example**

As a first example, we apply the HMM to calculate the probability that we feel cold for two consecutive days. In these two days, there are 3\*3=9 options for the underlying Markov states. Let us give an example for the probability computation of one of these 9 options:



Summing up all options gives the desired probability.

Finding Hidden States — Viterbi Algorithm

* + In some cases, we are given a series of observations, and want to find the most probable corresponding hidden states.
  + A brute force solution would take exponential time (like the calculations above); A more efficient approach is called the **Viterbi Algorithm**; its main idea is as follows: we are given a sequence of observations *o*₁,…,*o*ₜ . For each state *i* and *t*=1,…,*T*, we define



* That is, the maximum probability of a path which ends at time *t* at the state *i*, given our observations. The main observation here is that by the Markov property, if the most likely path that ends with *i* at time *t* equals to some *i*\* at time *t*−1, then *i*\* is the value of the last state of the most likely path which ends at time *t*−1. This gives us the following forward recursion:



* here, αⱼ(oₜ) denotes the probability to have oₜ when the hidden Markov state is j .

Learning and the Baum-Welch Algorithm

* + A similar approach to the one above can be used for parameter learning of the HMM model. We have some dataset, and we want to find the parameters which fit the HMM model best. The **Baum-Welch** Algorithm is an iterative process which finds a (local) maximum of the probability of the observations P(*O*|M), where M denotes the model (with the parameters we want to fit). Since we know P(M|*O*) by the model, we can use a Bayesian approach to find P(M|*O*) and converge to an optimum.
  + HMM have various applications, from character recognition to financial forecasts (detecting regimes in markets).

# UNIT – IV

**Markov Decision Process**

* MDP Formulation
* Utility Theory
* Utility Functions
* Value Iteration
* Policy Iteration
* Partially observable MDPs

Markov Decision Process

**Reinforcement Learning :**

* + Reinforcement Learning is a type of Machine Learning. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.
  + There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed to decide the best action to select based on his current state. When this step is repeated, the problem is known as a **Markov Decision Process**.
  + A **Markov Decision Process (MDP)** model contains:
    - A set of possible world states S.
    - A set of Models.
    - A set of possible actionsA.
    - A real valued reward function R(s,a).
    - A policy the solution of **Markov Decision Process**.
  + What is a State?

A **State** is a set of tokens that represent every state that the agent can be in.

* + **What is a Model?**

A **Model** (sometimes called Transition Model) gives an action’s effect in a state. In particular, T(S, a, S’) defines a transition T where being in state S and taking an action ‘a’ takes us to state S’ (S and S’ may be same). For stochastic actions (noisy, non-deterministic) we also define a probability P(S’|S,a) which represents the probability of reaching a state S’ if action ‘a’ is taken in state S. Note Markov property states that the effects of an action taken in a state depend only on that state and not on the prior history.

* + **What is Actions?**

An **Action** A is set of all possible actions. A(s) defines the set of actions that can be taken being in state S.

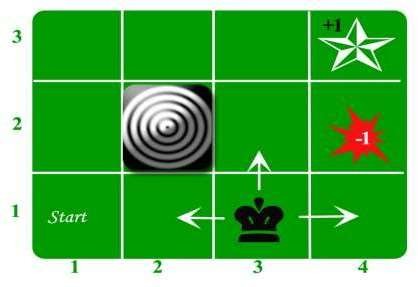
* + What is a Reward?

A **Reward** is a real-valued reward function. R(s) indicates the reward for simply being in the state S. R(S,a) indicates the reward for being in a state S and taking an action ‘a’. R(S,a,S’) indicates the reward for being in a state S, taking an action ‘a’ and ending up in a state S’.

* + What is a Policy?

A **Policy** is a solution to the Markov Decision Process. A policy is a mapping from S to a. It indicates the action ‘a’ to be taken while in state S.

Example:

Let us take the example of a grid world:

An agent lives in the grid. The above example is a 3\*4 grid. The grid has a START state(grid no 1,1). The purpose of the agent is to wander around the grid to finally reach the Blue Diamond (grid no 4,3). Under all circumstances, the agent should avoid the Fire grid (orange color, grid no 4,2). Also, the grid no 2,2 is a blocked grid, it acts like a wall hence the agent cannot enter it.

* + The agent can take any one of these actions: **UP, DOWN, LEFT, RIGHT**
  + Walls block the agent path, i.e., if there is a wall in the direction the agent would have taken, the agent stays in the same place. So for example, if the agent says LEFT in the START grid he would stay put in the START grid.
  + **First Aim:** To find the shortest sequence getting from START to the Diamond. Two such sequences can be found:
* RIGHT RIGHT UP UP RIGHT
* **UP UP RIGHT RIGHT RIGHT**
  + Let us take the second one (UP UP RIGHT RIGHT RIGHT) for the subsequent discussion. The move is now noisy. 80% of the time the intended action works correctly. 20% of the time the action agent takes causes it to move at right angles. For example, if the agent says UP the probability of going UP is 0.8 whereas the probability of going LEFT is 0.1 and probability of going RIGHT is 0.1 (since LEFT and RIGHT is right angles to UP).
  + The agent receives rewards each time step:-
* Small reward each step (can be negative when can also be term as punishment, in the above example entering the Fire can have a reward of -1).
* Big rewards come at the end (good or bad).
* The goal is to Maximize sum of rewards.

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.

**Markov Property:**

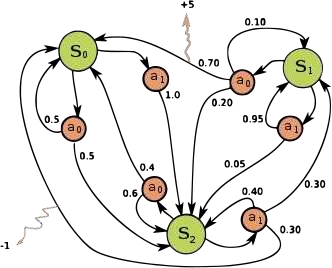
* + It says that ***"If the agent is present in the current state S1, performs an action a1 and move to the state s2, then the state transition from s1 to s2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."***
  + Or, in other words, as 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 S1, S2, ....., St 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.

The goal of reinforcement learning, contrary to the previously seen methods of machine learning (supervised/unsupervised learning), is to tweak the system in order to perform right or wrong in certain ways.



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

* + A mathematical representation of a complex decision making process is “**Markov Decision Processes**” (MDP).
  + We do that by attaching rewards and punishments to different outcomes, which ultimately drive the machine to find the “right” priorities.
  + An example of it is self-driving cars. There the system “prefers” one optimality versus another, taking into consideration rewards and punishments along the way.
  + Supervised learning is characterized by x,y pairs and the goal is to find function approximation that defines them and that predicts future data.
  + In Unsupervised learning, instead, we are given a bunch of x’s and the goal is to find a function that describes the unrolling clustering description.
  + Instead, in reinforcement learning we are given a pair of data (x,z) and we need to learn the function that generates y.
  + The final goal of the MDP is to find a policy that can tell us, for any state, which action to take.
  + The optimal policy is the one that maximizes the long-term expected reward.
  + There are two properties on which the Markovian process is based on:
  + Only the present matters; which means that the transition function only depends on the current state S and not any of the previous states.
  + Things are stationarity; therefore rules do no change over time.
  + ***Stationarity of sequences***

This claims that if we prefer one sequence of states today over another sequence of states, then we prefer that sequence of states over the same sequence of states even tomorrow.

**Utility Function:**

* A sequence of states is the “**utility**” function.
* Basically, a reward gives us immediate feedback, present gratification, while utility gives a long-term feedback, since it accounts for all the delayed rewards.

#### Optimal Policy:

The **optimal policy** is the one that maximizes the expected rewards. The utility does indeed depend upon the policy. It is the expected set of states that we see from that point on, given that we follow a policy.

*Bellman Equation:*

* + The utility of a certain state is equal to the reward in the current state, plus the discount from all the rewards I get from that point on, transitioning from the current state to a future state.
  + The “discount” mentioned above is a value between zero and one, and it allows us to treat an infinite sequence with finite value. Otherwise, the utility in an infinite time horizon will always be the same, of an infinite magnitude.
  + To solve the Bellman equation we normally start with arbitrary utilities and then update them based on the neighbors (which are all the states that it can reach from the current state I am in). Finally, we repeat that until convergence.
  + This process is called “value iteration”.
  + As we can infer, the utility is similar to a regression, mapping through continuous values, while the policy is more like a classifier.
  + Besides the above-mentioned **value-based** function (from which we observe the values as outputs, find the utility and finally infer the policy), there are two other approaches.
  + One is the “**policy-search**” algorithm, with which we observe the actions from different states and deduct the policy, which we can directly use.
  + The other one is a “**model-based**” algorithm, from which we get state/action pairs as input and we receive rewards/states as output.
  + It’s a direct learning approach, similar to a supervised learning one, but with an indirect use. In fact, from this one we will need to solve the Bellman equation in order to deduct the utility and then finally derive the policy.

#### Optimization objective of MDP:

* + The goal in a Markov decision process is to find a good "policy" for the decision maker. Once a Markov decision process is combined with a policy in this way, this fixes the action for each state and the resulting combination behaves like a [Markov chain](https://en.wikipedia.org/wiki/Markov_chain).

#### Utility Theory

* + Utility Theory as one of the main vehicles AI agents use to make decisions under uncertainty.
  + The main elements of Utility Theory known as the principle of maximum utility or Maximum Expected Utility (MEU) which is an indispensable element of AI algorithms.

#### The Axioms of Utility Theory

Consider a hypothetical scenario that have you at a nice restaurant struggling to make a decision between a salmon or a chicken dish. As you can imagine, there are many unknown factors(uncertainly) that can contribute to that decision. To help you through your struggles, Utility Theory introduces six fundamental axioms.

#### Orderability

* + The principle of Orderability states that given two sets of options, a rational AI agents should either prefer one to the other or rate the two as equally preferable.
  + Going back to our nice dinner scenario, you

Must decide between chicken dish or rate the salmon and the two as equally

delicious. Orderability is another way to say that the AI agent can’t avoid making a decision at any given state.

#### Transitivity

* + The Transitivity axiom states that given any three paths Op1, Op2 and Op3. If an AI agent prefers Op1 to Op2 and Op2 to Op3 then it must prefer Op1 to Op3.
  + In our scenario, the principle of Transitivity means that if you prefer salmon to duck and duck to chicken then you must prefer salmon to chicken.

#### Substitutability

* + The principle of Substitutability states that if an AI agent is indifferent between two options Op1 and Op2, then the agent is also indifferent between two more complex options that are identical except that Op1 is substituted for Op2 in one of them.
  + Using Substitutability, if you don’t have any preference between salmon and chicken then you should also like salmon fettuccine and chicken fettuccine the same.

#### Monotonicity

* + The Monotonicity axiom states that if two options have the same two possible outcomes Oc1 and Oc2, if an agent prefers Oc1 to Oc2, it should also prefer the option with the highest probability of Oc1 occurring.
  + Suppose that you are going to pair the salmon or the chicken with either a red Burgundy (Chambertin) or a white one (Montrachet ). If you prefer the Montrachet to the Chambertin, then you should also prefer the salmon to the chicken as the former goes better with the Montrachet.

#### Decomposability

* + The principle of decomposability states that complex options can be reduced to simpler ones using the laws of probability.
  + In our dinner scenario, Decomposability tells us that if you prefer salmon fettuccine to chicken fettuccine there is also a high probability that you would prefer salmon to chicken.

#### Continuity

* + The Continuity law states that if there is an Op3 between Op1 and Op2 in preference, then there is a probability P for which an AI agent will be indifferent between Op3 and an option that yields Op1 with probability P and Op2 with probability 1-P
  + I know the principle of Continuity might sound confusing but, if we take our scenario, it simply “sorts of” means that if you prefer chicken to rabbit and rabbit to salmon, there is some dish based on both chicken and salmon that you would like as much as rabbit( that’s just a theory though ;) ).
  + In some complete

#### Utility Functions

context, we could generalize the spectrum of AI applications as

scenarios that involve a utility function that needs to be maximized by a rational agent.

* + Utility functions are a product of Utility Theory which is one of the disciplines that helps to address the challenges of building knowledge under uncertainty.
* In **AI**, a **utility function** assigns values to certain actions that the **AI** system can take. An **AI** agent's preferences over possible outcomes can be captured by a **function** that maps these outcomes to a **utility** value; the higher the number the more that agent likes that outcome.
* How do you find the utility function?

If you are given a **utility function** U(x,y), it is easy to **derive** a given indifference curve from it: simply plot all points (x,y) such that U(x,y) equals a constant. This is a **utility function** in which the consumer values x as much as a/b units of y.

* Utility Theory is often combined with probabilistic theory to create what we know as decision-theoretic agents. Conceptually, a decision-theoretic agent is an AI program that can make rational decisions based on what it believes and what it wants.
* Utility Theory is the discipline that lays out the

foundation to evaluate Utility Functions.

create Typically,

and Utility

Theory uses the notion of Expected Utility (EU) as a value that represents the average utility of all possible outcomes of a state, weighted by the probability that the outcome occurs.

#### What is utility in decision making?

In **decision theory**, **utility** is a measure of the desirability of consequences of courses of action that applies to **decision making** under risk--that is, under uncertainty with known probabilities.

What is utility example?

**Utility** is defined as want-satisfying capacity of the commodity. For **example**, when a person is hungry, bread has **utility** for him. It is a relative concept. **eg**. plough is useful for a farmer but has no **utility** for a fisherman.

#### Utility scales and Utility assessments

The agents use the utility theory for making decisions. To help an agent in making decisions and behave accordingly, we need to build a decision-theoretic system. For this, we need to understand the utility function. This process is known as **preference elicitation** In this, the agents are provided with some choices and using the observed preferences, the respected utility function is chosen. Generally, there is no scale for the utility function. But, a scale can be established by fixing the boiling and freezing point of water.

#### Multi-attribute utility functions

Multi-attribute utility functions include those problems whose outcomes are categorized by two or more attributes. Such problems are handled by multi-attribute utility theory.

#### Terminology used in Utility functions

* + **Dominance:** If there are two choices say A and B, where A is more effective than B. It means that A will be chosen. Thus, A will dominate B. Therefore, multi-attribute utility function offers two types of dominance:
* **Strict Dominance:** If there are two websites T and D, where the cost of T is less and provides better service than D. Obviously, the customer will prefer T rather than D. Therefore, T strictly

dominates known.

1. Here, the attribute values are
   * **Stochastic Dominance:** It is a generalized approach where the attribute value is unknown. It frequently occurs in real problems. Here, a uniform distribution is given, where that choice is picked, which stochastically dominates the other choices. The exact relationship can be viewed by examing the cumulative distribution of the attributes.
     + [Types of Notable Variant](https://en.wikipedia.org/wiki/Markov_decision_process)s (Algorithms)
2. [Value iteration](https://en.wikipedia.org/wiki/Markov_decision_process)
3. [Policy iteration](https://en.wikipedia.org/wiki/Markov_decision_process)
4. [Modified policy iteration](https://en.wikipedia.org/wiki/Markov_decision_process)
5. [Prioritized sweeping](https://en.wikipedia.org/wiki/Markov_decision_process)

#### Value Iteration

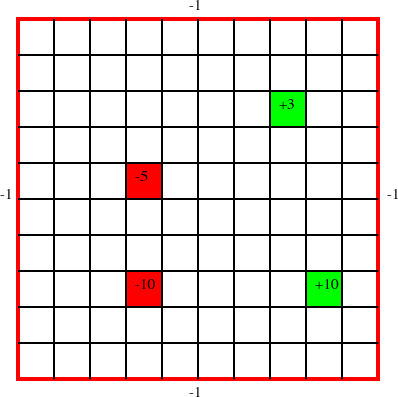
Value iteration is a method of computing an optimal MDP policy and its value.

Value iteration starts at the "end" and then works backward, refining an estimate of either *Q\** or *V\**. There is really no end, so it uses an arbitrary end point. Let *Vk* be the value function assuming there are *k* stages to go, and let *Qk* be the *Q*-function assuming there are *k* stages to go. These can be defined recursively. Value iteration starts with an arbitrary function *V0* and uses the following equations to get the functions for *k+1* stages to go from the functions for *k* stages to go:

Qk+1(s,a)= ∑s' P(s'|s,a) (R(s,a,s')+ γVk(s')) for k ≥ 0Vk(s)= maxa Qk(s,a) for k>0

It can either save the *V[S]* array or the *Q[S,A]* array. Saving the *V* array results in less storage, but it is more difficult to determine an optimal action, and one more iteration is needed to determine which action results in the greatest value.

**Example:** Consider the 9 squares around the *+10* reward of following figure. The discount is *γ=0.9*. Suppose the algorithm starts with *V0[s]=0* for all states *s*.



The above figure shows a *10×10* grid world, where the robot can choose one of four actions: up, down, left, or right. If the agent carries out one of these actions, it has a *0.7* chance of going one step in the desired direction and a *0.1* chance of going one step in any of the other three directions. If it bumps into the outside wall (i.e., the square computed as above is outside the grid), there is a penalty of 1 (i.e., a reward of *-1*) and the agent does not actually move. There are four rewarding states (apart from the walls), one worth *+10* (at position *(9,8)*; 9 across and 8 down), one worth *+3* (at position *(8,3)*), one worth *-5* (at position *(4,5)*), and one worth *-10* (at position *(4,8)*). In each of these states, the agent gets the reward after it carries out an action in that state, not when it enters the state. When the agent reaches the state *(9,8)*, no matter what it does at the next step, it is flung, at random, to one of the four corners of the grid world.

The values of *V1*, *V2*, and *V3* (to one decimal point) for these nine cells is

0 0 *-0.1*

0 10 *-0.1*

0 0 *-0.1*

0 *6.3 -0.1*

*6.3 9.8 6.2*

0 *6.3 -0.1*

|  |  |  |
| --- | --- | --- |
| *4.5* | *6.2* | *4.4* |
| *6.2* | *9.7* | *6.6* |
| *4.5* | *6.1* | *4.4* |

After the first step of value iteration, the nodes get their immediate expected reward. The center node in this figure is the *+10* reward state. The right nodes have a value of *-0.1*, with the optimal actions being up, left, and down; each of these has a *0.1* chance of crashing into the wall for a reward of *-1*.

The middle grid shows *V2*, the values after the second step of value iteration. Consider the node that is immediately to the left of the *+10* rewarding state. Its optimal value is to go to the right; it has a 0.7 chance of getting a reward of 10 in the following state, so that is worth 9 (10 times the discount of *0.9*) to it now. The expected reward for the other possible resulting states is *0*. Thus, the value of this state is *0.7×9=6.3*.

Consider the node immediately to the right of the *+10* rewarding state after the second step of value iteration. The agent's optimal action in this state is to go left. The value of this state is

Prob

0.7×(

Reward

0 +

Future Value 0.9×10)

+0.1×(

+0.1×(

*Agent goes left*

0 + 0.9×-0.1)

*Agent goes up*

-1 + 0.9×-0.1)

*Agent goes right*

+0.1×( 0 + 0.9×-0.1)

*Agent goes down*

which evaluates to 6.173.

* + - Notice also how the *+10* reward state now has a value less than 10. This is because the agent gets flung to one of the corners and these corners look bad at this stage.
    - After the next step of value iteration, shown on the right-hand side of the figure, the effect of the +10 reward has progressed one more step. In particular, the corners shown get values that indicate a reward in 3 steps.

#### Policy Iteration

**Policy iteration** starts with a policy and iteratively improves it. It starts with an arbitrary policy *π0* (an approximation to the optimal policy works best) and carries out the following steps starting from *i=0*.

* + - Policy evaluation: determine *Vπi(S)*. The definition of *Vπ* is a set of *|S|* linear equations in *|S|* unknowns. The unknowns are the values of *Vπi(S)*. There is an equation for each state. These equations can be solved by a linear equation solution method (such as Gaussian elimination) or they can be solved iteratively.
    - Policy improvement: choose *πi+1(s)= argmaxa Qπi(s,a)*, where the *Q*-value can be obtained from *V.* To detect when the algorithm has converged,

it should only change the policy if the new action for some state improves the expected value; that is, it should set *πi+1(s)* to be *πi(s)* if *πi(s)* is one of the actions that maximizes *Qπ i(s,a)*.

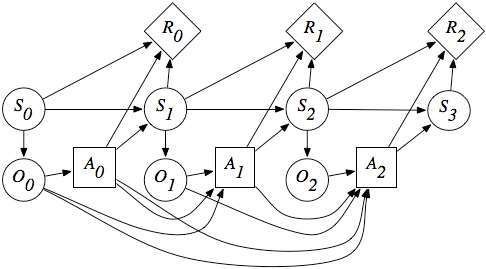
* + - Stop if there is no change in the policy - that is, if *πi+1=πi* - otherwise increment *i* and repeat.
    - A variant of policy iteration, called **modified policy iteration**, is obtained by noticing that the agent is not required to evaluate the policy to improve it; it can just carry out a number of backup steps and then do an improvement.
    - The idea behind policy iteration is also useful for systems that are too big to be represented directly as MDPs. Suppose a controller has some parameters that can be varied. An estimate of the derivative of the cumulative discounted reward of a parameter *a* in some context *s*, which corresponds to the derivative of *Q(a,s)*, can be used to improve the parameter.
    - Such an iteratively improving controller can get into a local maximum that is not a global maximum. Policy iteration for MDPs does not result in non- optimal local maxima, because it is possible to improve an action for a state without affecting other states, whereas updating parameters can affect many states at once.

#### Partially Observable Decision Processes

* + - A **partially observable Markov decision process** (**POMDP**) is a combination of an [MDP](https://artint.info/html/ArtInt_224.html) and a [hidden Markov model](https://artint.info/html/ArtInt_161.html). Instead of assuming that the state is observable, we assume that there are some partial and/or noisy observations of the state that the agent gets to observe before it has to act.

A POMDP consists of the following:

* + - *S*, a set of states of the world;
    - *A*, a set of actions;
    - *O*, a set of possible observations;
    - *P(S0)*, which gives the probability distribution of the starting state;
    - *P(S'|S,A)*, which specifies the dynamics - the probability of getting to state *S'* by doing action *A* from state *S*;
    - *R(S,A,S')*, which gives the expected reward of starting in state *S*, doing action *A*, and transitioning to state *S'*; and
    - *P(O|S)*, which gives the probability of observing *O* given the state is *S*.



A POMDP as a dynamic decision network

* + - There are three main ways to approach the problem of computing the optimal policy for a POMDP:
    - Solve the associated dynamic decision network using variable elimination for decision networks, extended to include discounted rewards. The policy created is a function of the [history of the agent](https://artint.info/html/ArtInt_36.html). The problem with this approach is that the history is unbounded, and the number of possible histories is exponential in the planning horizon.
    - Make the policy a function of the belief state - a probability distribution over the states. Maintaining the belief state is the problem of [filtering](https://artint.info/html/ArtInt_161.html). The problem with this approach is that, with *n* states, the set of belief states is an *(n-1)*-dimensional real space. However, because the value of a sequence of actions only depends on the states, the expected value is a linear function of the values of the states. Because plans can be conditional on observations, and we only consider optimal actions for any belief state, the optimal policy for any finite look-ahead, is piecewise linear and convex.
    - Search over the space of controllers for the best [controller](https://artint.info/html/ArtInt_36.html). Thus, the agent searches over what to remember and what to do based on its belief state and observations. Note that the first two proposals are instances of this approach: the agent remembers all of its history or the agent has a belief state that is a probability distribution over possible states. In general, the agent may want to remember some parts of its history but have probabilities over some other features. Because it is unconstrained over what to remember, the search space is enormous.

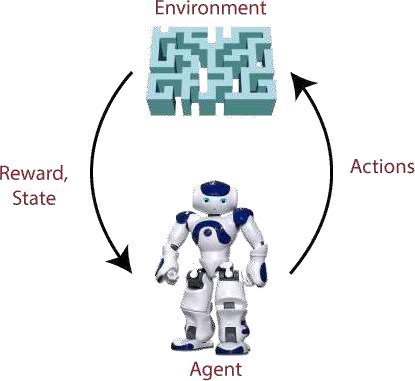
# UNIT – V

**Reinforcement Learning**

* Passive Reinforcement Learning
* Direct Utility Estimation
* Adaptive Dynamic Programming
* Temporal Difference Learning
* Active Reinforcement Learning
* Q Learning

What is 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.](https://www.javatpoint.com/supervised-machine-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, and all AI agent works on the concept of reinforcement learning.](https://www.javatpoint.com/agents-in-ai) 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 retuned 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 ReinforcementLearning

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

#### 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.

* + - 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.

* + - **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

* + - Policy
    - Reward Signal
    - Value Function
    - 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:

For deterministic policy: a = π(s) For stochastic policy: π(a | s) = P[At =a | St = s]

* + - **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.

Types of Reinforcement learning

* + - There are mainly two types of reinforcement learning, which are:
    - Positive Reinforcement
    - **Negative Reinforcement Positive Reinforcement:**
    - The positive reinforcement something to increase the

learning tendency

means adding thatexpected

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.

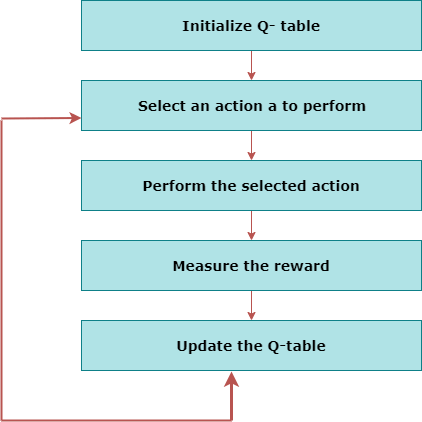
#### Negative Reinforcement:

* + - 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.

## Reinforcement Learning Algorithms

**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 π (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:**

**original state a: Original action**

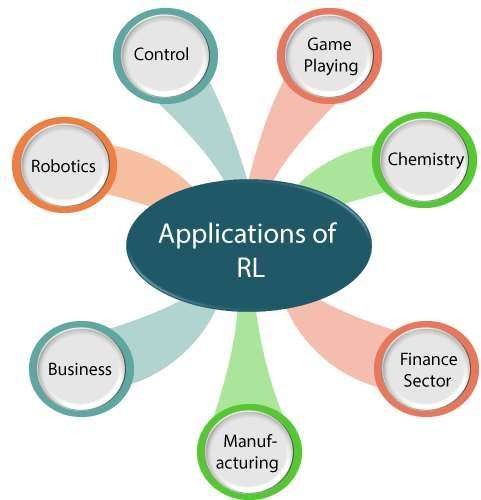
**r: reward observed while following the states s' and a': New state, action pair.**

**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.

#### Reinforcement Learning Applications

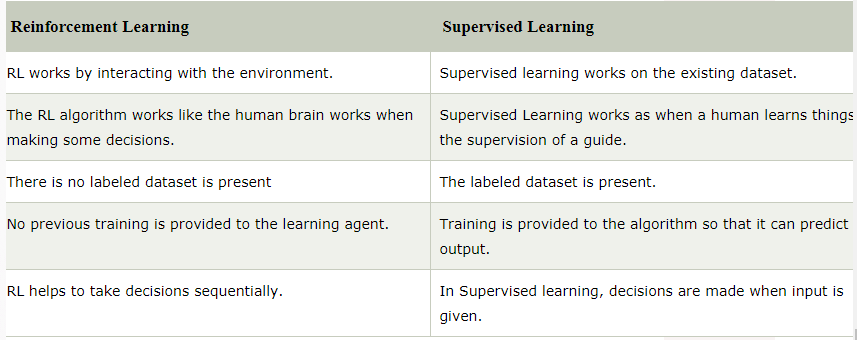


* + - **Robotics:**
      * RL is used in **Robot navigation, Robo-soccer, walking, juggling**, etc.
    - **Control:**
      * RL can be used for **adaptive control** such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.
    - **Game Playing:**
      * RL can be used in **Game playing** such as tic-tac-toe, chess, etc.
    - **Chemistry:**
      * RL can be used for optimizing the chemical reactions.
    - **Business:**
      * RL is now used for business strategy planning.
    - **Manufacturing:**
      * In various automobile manufacturing companies, the robots use deep

reinforcement learning to pick goods and put them in some containers.

* + - **Finance Sector:**
      * The RL is currently used in the finance sector for evaluating trading

strategies.



#### What are the required elements to solve an RL problem?

Let’s consider a problem where an agent can be in various states and can choose an action from a set of actions. Such type of problems

|  |  |  |  |
| --- | --- | --- | --- |
| are | called | ***Sequential Decision*** | ***Problems***. |
| An | [**MDP**](https://en.wikipedia.org/wiki/Markov_decision_process) | is the mathematical | framework |

which captures such a **fully observable, non- deterministic environment** with **Markovian Transition Model and additive rewards** in which the agent acts.

The solution to an MDP is an ***optimal policy*** which refers to the choice of action for

Every state that **Maximizes** overall

**cumulative**

**reward**. Thus

the ***transition***

***model*** that represents an agent’s

environment(when the environment is known) and the ***optimal policy*** which decides what action the agent needs to perform in each state are required elements for training the agent learn a specific behavior.

#### What is meant by passive and active

**Reinforcement learning and compare the two?** how do we

* + - Both active and passive reinforcement learning are types of RL. In case of passive RL, the agent’s policy is fixed which means that it is ***told what to do***. In contrast to this, in active RL, an agent ***needs to decide what to do*** as there’s no fixed policy that it can act on. Therefore, the goal of a passive RL agent is to execute a fixed policy (sequence of actions) and evaluate it while that of an active RL agent is to act and learn an optimal policy.

#### What are some common active and passive RL techniques?

1. **Passive Learning**
   1. *Direct Utility Estimation*
   2. *Adaptive Dynamic Programming(ADP)*
   3. *Temporal Difference Learning (TD)*

#### Active Learning

* 1. *ADP with exploration function*
  2. *Q-Learning*

#### Passive Learning

As the goal of the agent is to evaluate how good an optimal policy is, the agent needs to learn the expected utility *Uπ(s)* for each state *s*. This can be done in three ways.

1. ***Direct Utility Estimation***

In this method, the agent executes a sequence of trials or runs (sequences of states-actions transitions that continue until the agent reaches the terminal state).

Each trial gives a sample value and the agent estimates the utility based on the samples values. Can be calculated as **running averages of sample values**. *The main drawback is that this method makes a wrong*

*assumption*

##### independent

*that*

*while*

##### state utilities are

*in reality they*

*are* [***Markovian***](https://en.wikipedia.org/wiki/Markov_property)*.* Also, it is slow to converge.

Suppose we have a 4x3 grid as the environment in which the agent can move either Left, Right, Up or Down(set of available actions). An example of a run



Total reward starting at *(1,1)* = 0.72

##### Adaptive Dynamic Programming(ADP)

ADP is a smarter method than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.



Where *R(s)* = reward for being in state *s*, *P(s’|s, π(s))* = transition model, *γ* = discount factor and *Uπ(s)* = utility of being in state *s’*.

* + - It can be solved using **value-iteration algorithm**. The algorithm converges fast but can become quite costly to compute for large state spaces. ADP is a model based approach

and requires

the transition model of the

environment.

A model-free

approach is

Temporal Difference Learning.

##### Temporal Difference Learning (TD)

TD learning does not require the agent to learn the transition model. The update occurs between successive states and agent only updates states that are directly affected.



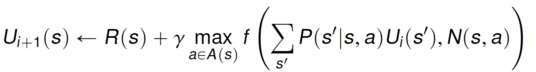
Where *α* = learning rate which determines the convergence to true utilities.

***While ADP adjusts the utility of s with all its successor states, TD learning adjusts it with that of a single successor state s’*.** TD is slower in convergence but much simpler in terms of computation.

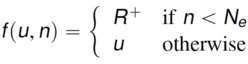
#### Active Learning

##### ADP with exploration function

As the goal of an active agent is to learn an optimal policy, the agent needs to learn the expected utility of each state and update its policy. Can be done using a passive ADP agent and then using value or policy iteration it can learn optimal actions. But this approach results into a greedy agent. ***Hence, we use an approach that gives higher weights to unexplored actions and lower weights to actions with lower utilities.***



Where *f(u, n)* is the exploration function that increases with expected value *u* and decreases with number of tries *n*



*R+* is an optimistic reward and *Ne* is the number of times we want an agent to be forced to pick an action in every state. ***The exploration function converts a passive agent into an active one.***

##### Q-Learning

Q-learning is a TD learning method which

Does not transitional

Require the agent to learn the model, instead learns Q-value

functions *Q(s, a)*.

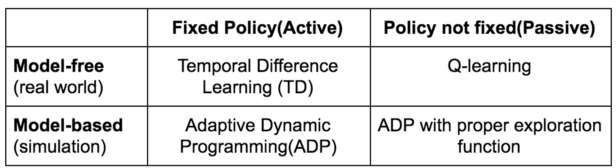


Q-values can be updated using the following equation,



Next action can be selected using the following policy,

Again, this is simpler to compute but slower than ADP.

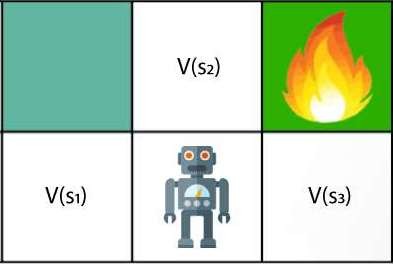


Comparison of active and passive RL methods

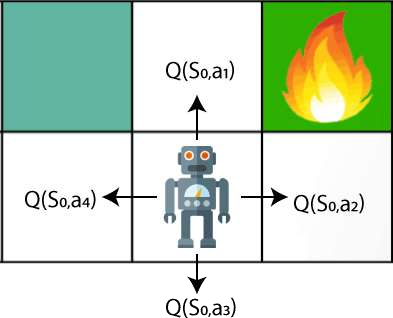
#### Q-Learning Explanation

* 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:

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(s1), V(s2), V(s3). 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:



Hence, we can say that, ***V(s) = max [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.