MCA 305B: Artificial Intelligence

UNIT I

Introduction Concept of AI, history, current status, scope, agents, environments, Problem Formulations, Review of tree and graph structures, State space representation, Search graph and Search tree

UNIT II

Random search, Search with closed and open list, Depth first and Breadth first search, Heuristic search, Best first search, A* algorithm, Game Search.

UNIT III

Probability, conditional probability, Constraint Satisfaction, Propositional Logic & Satisfiability, Uncertainty in AI, Bayes Rule, Bayesian Networks- representation, construction and inference, temporal model, hidden Markov model.

UNIT IV

MDP formulation, utility theory, utility functions, value iteration, policy iteration and partially Observable MDPs.

UNIT V

Passive reinforcement learning, direct utility estimation, adaptive dynamic programming, temporal difference learning, active reinforcement learning- Q learning. Introduction to Machine learning ,Deep Learning.

Text Books

1. Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach" , 3rd Edition, Prentice Hall

2. Elaine Rich and Kevin Knight, "Artificial Intelligence", Tata McGraw Hill

Reference Books

1. Saroj Kaushik, "Artificial Intelligence", Cengage Learning India, 2011

2. David Poole and Alan Mackworth, "Artificial Intelligence: Foundations for Computational Agents",

Cambridge University Press 2010.

LECTURE NOTES

UNIT-1

INTRODUCTION

Artificial intelligence (**AI**) is intelligence demonstrated by machines, as opposed to the **natural intelligence** displayed by animals and humans. AI research has been defined as the field of study of intelligent agents, which refers to any system that perceives its environment and takes actions that maximize its chance of achieving its goals.[a]

The term "artificial intelligence" had previously been used to describe machines that mimic and display "human" cognitive skills that are associated with the human mind, such as "learning" and "problem-solving". This definition has since been rejected by major AI researchers who now describe AI in terms of rationality and acting rationally, which does not limit how intelligence can be articulated.[b]

AI applications include advanced web search engines (e.g., Google), recommendation systems (used by YouTube, Amazon and Netflix), understanding human speech (such as Siri and Alexa), self-driving cars (e.g., Tesla), automated decision-making and competing at the highest level in strategic game systems (such as chess and Go).[2] As machines become increasingly capable, tasks considered to require "intelligence" are often removed from the definition of AI, a phenomenon known as the AI effect.[3] For instance, optical character recognition is frequently excluded from things considered to be AI,[4] having become a routine technology.[5]

Artificial intelligence was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism,[6][7] followed by disappointment and the loss of funding (known as an "AI winter"),[8][9] followed by new approaches, success and renewed funding.[7][10] AI research has tried and discarded many different approaches since its founding, including simulating the brain, modeling human problem solving, formal logic, large databases of knowledge and imitating animal behavior. In the first decades of the 21st century, highly mathematical-statistical machine learning has dominated the field, and this technique has proved highly successful, helping to solve many challenging problems throughout industry and academia.[10][11]

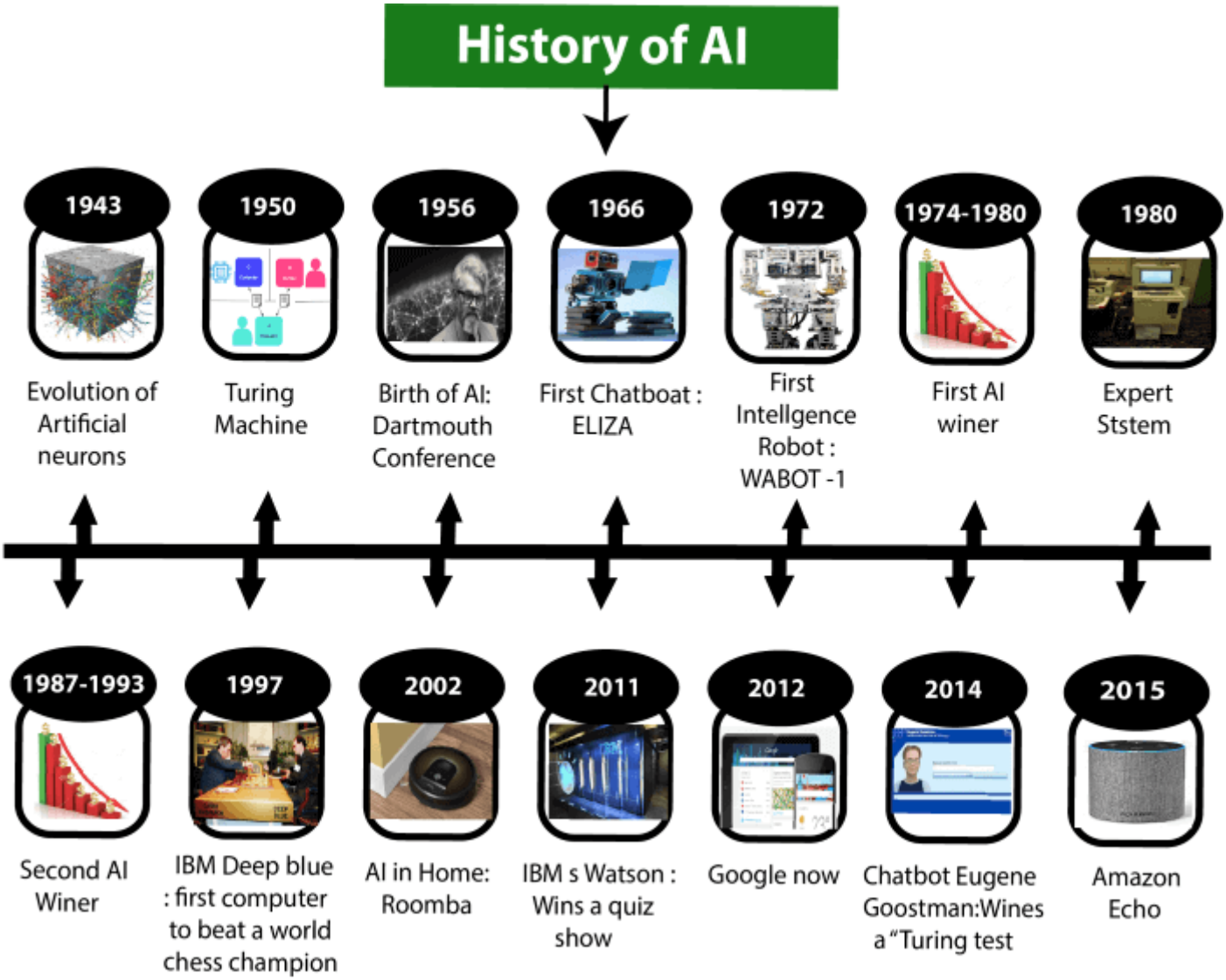
The various sub-fields of AI research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception, and the ability to move and

manipulate objects.[c] General intelligence (the ability to solve an arbitrary problem) is among the field's long-term goals.[12] To solve these problems, AI researchers have adapted and integrated a wide range of problem-solving techniques – including search and mathematical optimization, formal logic, artificial neural networks, and methods based on statistics, probability and economics. AI also draws upon computer science, psychology, linguistics, philosophy, and many other fields.

The field was founded on the assumption that human intelligence "can be so precisely described that a machine can be made to simulate it".[d] This raised philosophical arguments about the mind and the ethical consequences of creating artificial beings endowed with human-like intelligence; these issues have previously been explored by myth, fiction and philosophy since antiquity.[14] Computer scientists and philosophers have since suggested that AI may become an existential risk to humanity if its rational capacities are not steered towards beneficial goals.[e

History of Artificial Intelligence

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.



Maturation of Artificial Intelligence (1943-1952)

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

The birth of Artificial Intelligence (1952-1956)

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

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

The golden years-Early enthusiasm (1956-1974)

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

The first AI winter (1974-1980)

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

A boom of AI (1980-1987)

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

The second AI winter (1987-1993)

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

The emergence of intelligent agents (1993-2011)

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

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

- **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.

- **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
- **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
- **Year 2018:** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
- Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.

Now AI has developed to a remarkable level. The concept of Deep learning, big data, and data science are now trending like a boom. Nowadays companies like Google, Facebook, IBM, and Amazon are working with AI and creating amazing devices. The future of Artificial Intelligence is inspiring and will come with high intelligence.

SCOPE OF ARTIFICIAL INTELLIGENCE

The scope of Artificial Intelligence in India is promising. Artificial Intelligence has immense potential to change each sector of the economy for the benefit of society. There is not just one technology under AI, but there are various useful technologies

such as self-improving algorithms, machine learning, big data, pattern recognition. Soon, there would hardly be any industry or sector which would be untouched by this powerful tool in India. This is the reason why there has been an increasing demand for Artificial Intelligence online courses in India.

1. Education - Artificial Intelligence can help increase the effectiveness of our instructors via numerous AI applications such as text translation systems, real-time message to speech, automating mundane and also repeated jobs such as taking presence, automating grading, customising the discovering trip based upon ability, understanding, and also experience. The scope of Artificial Intelligence education and learning takes a look at the possibility of utilising AI ran rating machines that can evaluate unbiased solutions. This is something that is gradually being implemented in institutes. The various other applications of AI in the area of education are real-time text to speech as well as text translation systems.

Chatbots : In a country as varied as India, the combination of chatbots in the digital framework or availability via the IVRS system education domain can be transformational- they might be educated on the subject matter and a great percentage of doubts of the pupils could be responded to quickly, consequently lowering the current work of educators who could focus on even more imaginative tasks.

Automated grading : On a large scale, Machine Learning methods such as Natural Language Processing might be made use of for automated grading of assessments on systems such as E-PATHSHALA, SWAYAM (Study Webs of Active Learning for Young Aspiring Minds) and DIKSHA - not simply subjective ones however objective inquiries

too. This is due to draft of National Education Policy 2019, prioritising on the internet understanding.

Explore Artificial Intelligence Courses & Certifications by Top Providers

TalentSprint Machine Learning And Artificial Intelligence Courses & Certifications	Careerera Artificial Intelligence And Mac Learning Courses & Certifications
Edureka Artificial Intelligence And Machine Learning Courses & Certifications	Futurelearn Artificial Intelligence And M Learning Courses & Certifications
Futurelearn Artificial Intelligence And Machine Learning Courses & Certifications	Simplilearn Artificial Intelligence And M Learning Courses & Certifications
UpGrad Artificial Intelligence And Machine Learning Courses & Certifications	Great Learning Artificial Intelligence Anc Learning Courses & Certifications

2. Healthcare - The Healthcare system in India is the most dynamic and challenging sector in India. There are many challenges like affordability, accessibility but particularly the shortage of doctors and services like qualified nurses, technicians, and infrastructure. In India, good healthcare facilities are mostly near the tier1 and tier2 cities which creates non-uniform accessibility to healthcare across the country with physical access. Apart from this, with the development of Artificial Intelligence, the overall cost of healthcare would get reduced due to increased efficiency. With the ability of AI to handle large data in a speedy fashion it can help in making innovations, design, and developing medical equipment. Having an AI-enabled system helps in reducing medical errors and increases productivity. Artificial Intelligence can also overcome the barriers to access and solve the accessibility challenge by employing early detection followed by suitable diagnostic decisions.

3. Agriculture - In India, a significant population depends on agriculture for their livelihood. The problem of farmers in India is not extremely great as they depend on conventional methods for farming. If plants are obtaining an adequate quantity of water, water usage in agricultural land can be maximised by utilising thermal imaging cameras that constantly keep track of. It can help determine proper crops to expand in a desirable environment on a productive surface as well as the sowing technique to improve performance and also minimise expenses. Artificial Intelligence can also be utilised to forecast the behaviour and also study parasites which can be useful for advanced preparation of insect control. Anticipating modelling using Artificial Intelligence can be important in providing more precise demand-supply details as well as anticipating the need for farming produce to farmers.

4. Transport - There is tremendous scope of Artificial Intelligence in the transportation sector as well. Particularly, there are a few areas where AI can be used. In the aircraft, ships, spacecraft they have been using autopilot since 1922 to maintain the correct course. Another area is autonomous cars. There are many companies across the globe and even in India that are researching autonomous cars or self-driving cars. The development of such vehicles is heavily reliant on artificial intelligence and machine learning. Experts believe that self-driving cars will bring many benefits including lower emissions, error-free driving.

5. Home - We are surrounded by Artificial Intelligence. We use devices that are based on Artificial Intelligence all the time without knowing of it. For example, we use OK GOOGLE or ALEXA or CORTANA all the time to perform the various tasks with just voice command. These smart assistants use Artificial Intelligence and Machine Learning for voice recognition. They learn from the users' commands to become more efficient. You can also use this smart assistance to perform a variety of tasks like playing a song, asking a question, Buying something online.

6. Artificial Intelligence in Cyber Security - Cybersecurity is another field where Artificial Intelligence is employed. Many organisations deal with a significant amount of data. For example, in the Banking sector or government organisations which are having a huge database of people's personal data, there is always a risk of stealing, and hence a security mechanism needs to be put in place. An apt example of this field is Cognitive Artificial Intelligence. It detects as well as analyses hazards, while also giving understandings to the analysts for making better-informed decisions. By making use of Machine Learning formulas and also Deep Learning networks, the AI obtains better and much more resilience over time. IBM has IBM Resilient, which is an open as well as an agnostic platform that offers a framework and also center for managing safety and security responses.

7. Manufacturing sector - There are many Indian startups based on Artificial Intelligence that are serving the manufacturing industry. These companies develop Artificial Intelligence-based solutions to increase the growth of the manufacturing industry. Artificial Intelligence in the industry is used In controlling the various kinds of robots to do a certain kind of job. One unique technology of Artificial Intelligence is to

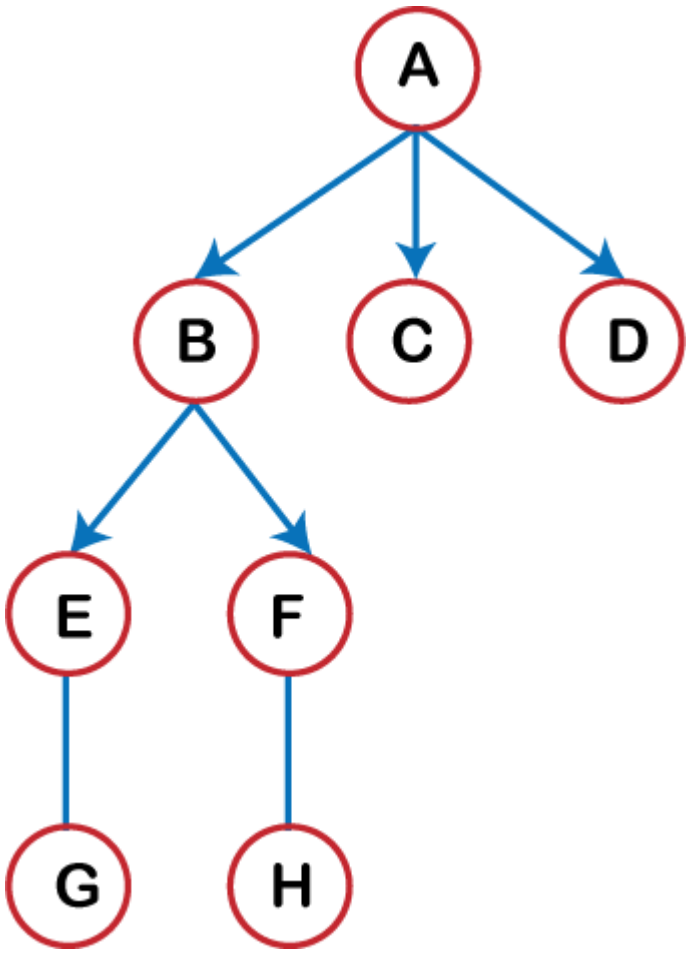
analyse the data and make future predictions. This ability of AI is used to analyse past year sales or market survey data to predict the future supply and demand and make faster decisions or amazing customer feedback of a given product to optimise the product in the coming years. AI has a wide scope in the Manufacturing industry in the coming years.

8. **Artificial Intelligence Online Course -** Online Artificial Intelligence certification course and ML can be beneficial in the future as there is a large scope of it in the coming years particularly in India where there are huge populations and several problems persist now and then. To find smart solutions to these modern-day problems AI will play an important role. Strong knowledge of data structures and data modelling. To pursue an Artificial Intelligence online course, one should know subjects such as Mathematics, engineering, physics, Programming expertise in one of the programming languages like Python, Java, C++, R, Working Knowledge of Data Science algorithms and libraries. The various job profiles in the field of AI include Machine Learning Engineer, Data scientist, Business Intelligence Developer.

3. write in detail about review of tree and graph structure?

Before knowing about the tree and graph data structure, we should know the linear and non-linear data structures. Linear data structure is a structure in which all the elements are stored sequentially and have only single level. In contrast, a non-linear data structure is a structure that follows a hierarchy, i.e., elements are arranged in multiple levels.

Let's understand the structure that forms the hierarchy.



In the above figure, we can assume the company hierarchy where A represents the CEO of the company, B, C and D represent the managers of the company, E and F represent the team leaders, and G and H represent the team members. This type of structure has more than one level, so it is known as a non-linear data structure.

What is Tree?

A tree is a non-linear data structure that represents the hierarchy. A tree is a collection of nodes that are linked together to form a hierarchy.

Let's look at some terminologies used in a ***tree*** data structure.

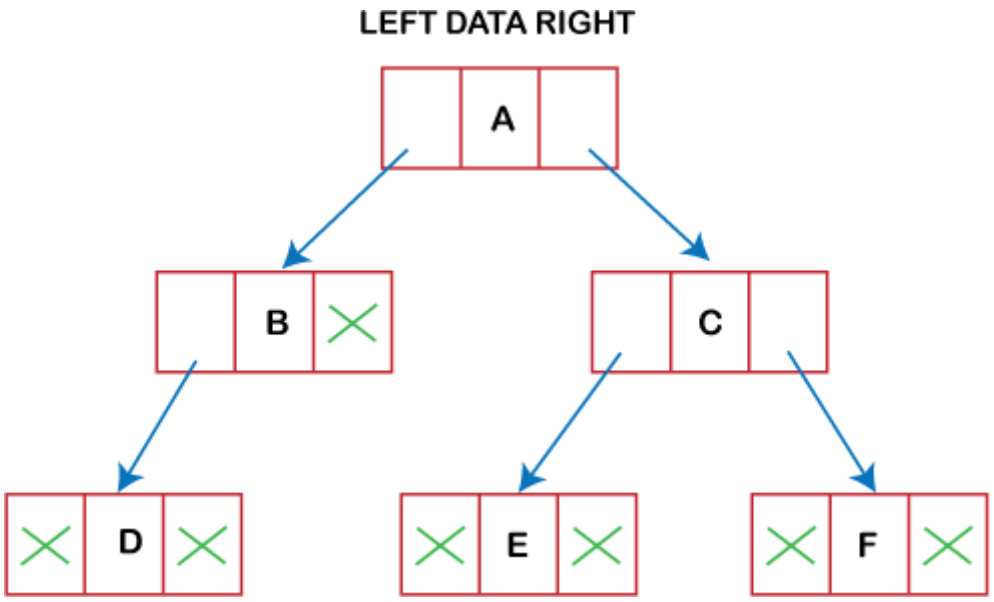
- **Root node:** The topmost node in a tree data structure is known as a root node. A root node is a node that does not have any parent.

- **Parent of a node:** The immediate predecessor of a node is known as a parent of a node. Here predecessor means the previous node of that particular node.
- **Child of a node:** The immediate successor of a node is known as a ***child of a node***.
- **Leaf node:** The leaf node is a node that does not have any child node. It is also known as an external node.
- **Non-leaf node:** The non-leaf node is a node that has atleast one child node. It is also known as an ***internal node***.
- **Path:** It is a sequence of the consecutive edges from a source node to the destination node. Here edge is a link between two nodes.
- **Ancestor:** The predecessor nodes that occur in the path from the root to that node is known as an ancestor.
- **Descendant:** The successor nodes that exist in the path from that node to the leaf node.
- **Sibling:** All the children that have the same parent node are known as siblings.
- **Degree:** The number of children of a particular node is known as a degree.
- **Depth of node:** The length of the path from the root to that node is known as a depth of a node.
- **Height of a node:** The number of edges that occur in the longest path from that node to the leaf node is known as the height of a node.
- **Level of node:** The number of edges that exist from the root node to the given node is known as a level of a node.

Note: If there are n number of nodes in the tree, then there would be (n-1) number of edges.

How is a tree represented in the memory?

Each node will contain three parts, data part, address of the left subtree, and address of the right subtree. If any node does not have the child, then both link parts will have NULL values.



What is a Graph?

A graph is like a tree data structure is a collection of objects or entities known as nodes that are connected to each other through a set of edges. A tree follows some rule that determines the relationship between the nodes, whereas graph does not follow any rule that defines the relationship among the nodes. A graph contains a set of edges and nodes, and edges can connect the nodes in any possible way.

Mathematically, it can be defined as an ordered pair of a set of vertices, and a set of nodes where vertices are represented by 'V' and edges are represented by 'E'.

G= (V , E)

Here we are referring to an ordered pair because the first object must be the set of vertices, and the second object must be a set of edges.

In Graph, each node has a different name or index to uniquely identify each node in the graph. The graph shown below has eight vertices named as v1, v2, v3, v4, v5, v6, v7, and v8. There is no first node, a second node, a third node and so on. There is no ordering of the nodes. Now, we will see how can we represent the edges in a graph?. An edge can be represented by the two endpoints in the graph. We can write the name of the two endpoints as a pair, that represents the edge in a graph.

There are two types of edges:

- **Directed edge:** The directed edge represents one endpoint as an origin and another point as a destination. The directed edge is one-way. For example, there are two vertices U and V; then directed edge would represent the link or path from U to V, but

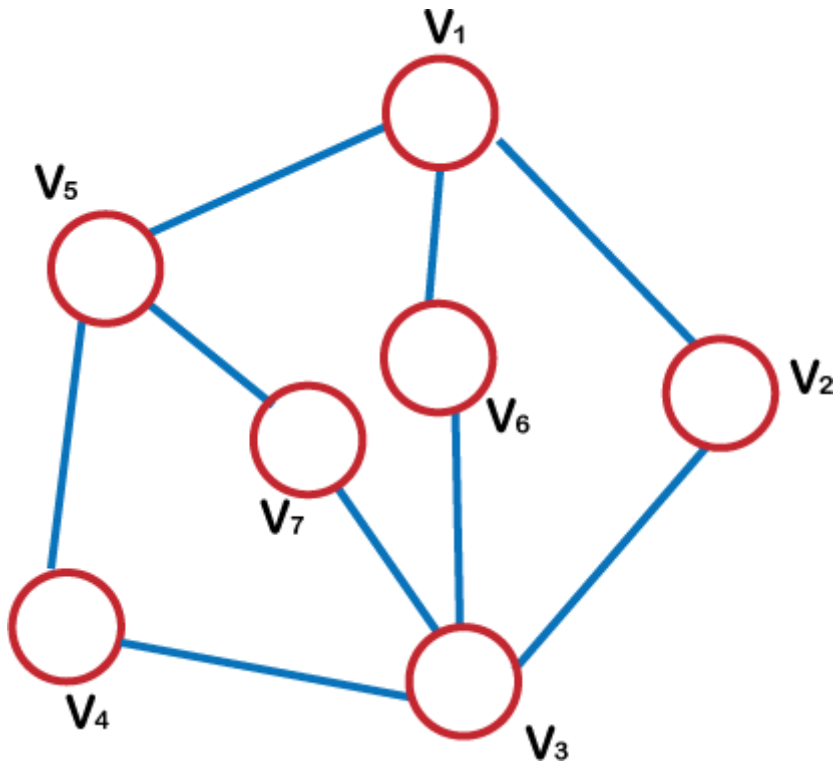
no path exists from V to U. If we want to create a path from V to U, then we need to have one more directed edge from V to U. The directed edge can be represented as an ordered pair in which the first element is the origin, whereas the second element is the destination.

- **Undirected edge:** The undirected edge is two-way means that there is no ***origin*** and ***destination***. For example, there are two vertices U and V, then undirected would represent two paths, i.e., from U to V as well as from V to U. An undirected edge can be represented as an unordered pair because the edge is ***bi-directional***.

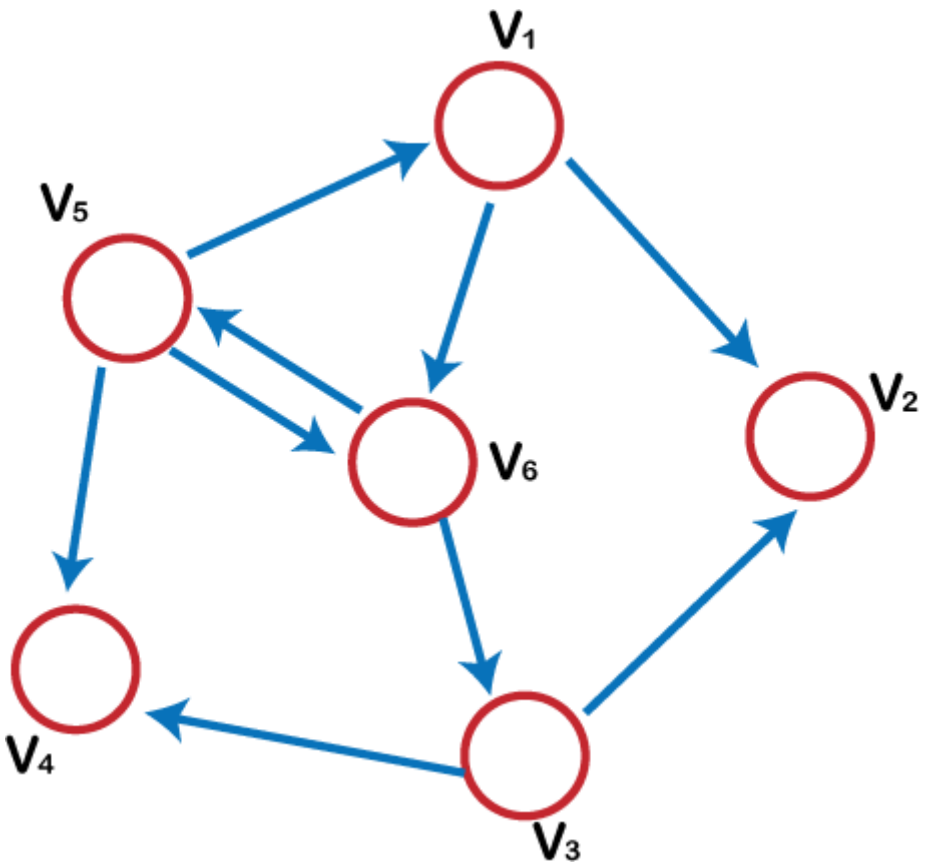
The tree data structure contains only directed edges, whereas the graph can have both types of edges, i.e., ***directed as well as undirected***. But, we consider the graph in which all the edges are either directed edges or undirected edges.

There are two types of graphs:

Directed graph: The graph with the directed edges known as a ***directed graph***.



Undirected graph: The graph with the undirected edges known as a ***undirected graph***. The directed graph is a graph in which all the edges are uni-directional, whereas the undirected graph is a graph in which all the edges are bi-directional.



Differences between tree and graph data structure.

Basis for comparison	Tree	Graph

Definition	Tree is a non-linear data structure in which elements are arranged in multiple levels.	A Graph is also a non-linear data structure.
Structure	It is a collection of edges and nodes. For example, node is represented by N and edge is represented as E, so it can be written as: T = {N,E}	It is a collection of vertices and edges. For example, vertices are represented by V, and edge is represented as 'E', so it can be written as: T = {V, E}
Root node	In tree data structure, there is a unique node known as a parent node. It represents the topmost node in the tree data structure.	In graph data structure, there is no unique node.
Loop formation	It does not create any loop or cycle.	In graph, loop or cycle can be formed.
Model type	It is a hierarchical model because nodes are arranged in multiple level, and that creates a hierarchy. For example, any organization will have a hierarchical model.	It is a network model. For example, facebook is a social network that uses the graph data structure.

Edges	If there are n nodes then there would be n-1 number of edges.	The number of edges depends on the graph.
Type of edge	Tree data structure will always have directed edges.	In graph data structure, all the edges can either be directed edges, undirected edges, or both.
Applications	It is used for inserting, deleting or searching any element in tree.	It is mainly used for finding the shortest path in the network.

SEARCH GRAPH

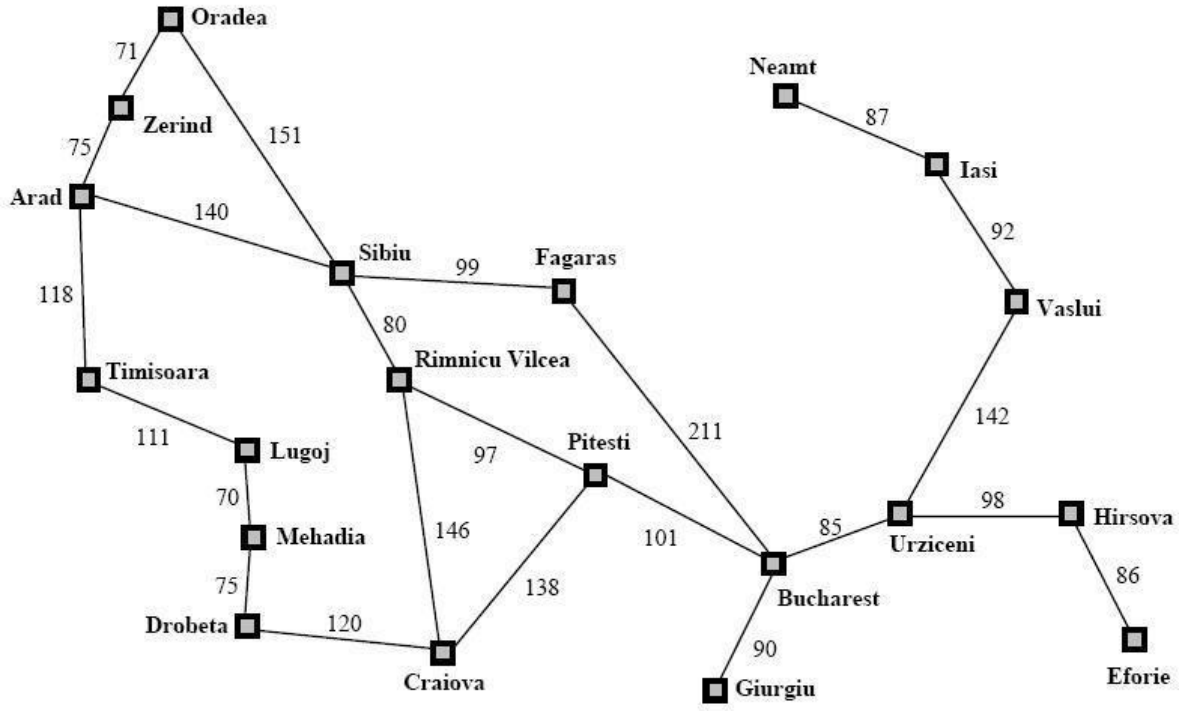
where a rational agent could be anything that makes decisions like a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts (agent's perceptual inputs at a given instance).

And what do we get with a graph search algorithm? We get a function, a program, an agent, that, given a graph, an environment, and a start node, by considering past and current percepts, calculates a series of actions for finding the goal node.

Well, we can see here that what a graph search algorithm does fits the definition of an AI agent. So, we have a basic form of AI. You see here, AI is not exactly rocket science. The definition is broad enough to include simple algorithms as the graph search. These examples give us a way to understand the basic concept of AI.

Considering the AI concepts, let's write an AI agent that analyzes the environment and calculates a series of actions that could help us to reach the goal (we will basically build a graph search algorithm but use AI terminology).

Below is the map of Romania.



Let's say that we are in Arad and we want to go, on the shortest path, to Bucharest. We can see that there are multiple routes that we can take. We can go from Arad to Timisoara, then Lugoj, and so on, or we can choose Sibiu then Fagaras and go straight to Bucharest. But we don't know which path has the lowest cost.

What's the Definition of the Problem?

A problem can be broken down in a number of components:

- We have an initial state: being in Arad
- We have a function actions (s). It returns the set of possible actions when we are in a state. E.g: being in Arad, we can go to Zerind, Sibiu, or Timisoara
- We have a function result (s,a) -> s' . Being in state s, if we apply the action a, we will go to s'
- We have a goal test function goaltest (s) -> t/f. It tells us if we reached the goal
- step_cost (s,a,s') -> cost of action. It tells us the step cost from going from s to s' using the action a
- cost (S1 -> S2 -> … -> Sn) -> n (cost of path). The total path cost

Let's now see how this maps on our goal-finding problem. The initial state is being in Arad and the goaltest function returns if we reach

Bucharest. All the possible states is called the state space. We have to explore the state space by applying actions and going from state to state. Starting with Arad and applying the function actions ("Arad") we will get three possible actions. Going to Zering, Timisoara, or Sibiu. Now, with this in mind, we can separate the state space into three parts:

- We have the explored part of the state (e.g. just Arad for now)
- We have the frontier. We call this the farthest states that have been explored (Timisoara, Zerind, and Sibiu)
- And the unexplored state: all the other cities

So, being in the explored part of the state, we need to select a new state from the frontier, apply an action, and move to that state, thus expanding the frontier.

```
TreeSearch(G, s)
  s.costSoFar = 0;
  for all nodes w in G except s
    w.costSoFar = Int.maxInt
  Let frontier be a priority queue ordered by  the cost_so_far
  frontier.enqueue(s)


  while (frontier is not empty)
    //Removing that vertex from frontier,whose neighbour will be visited no
w
    v = frontier.dequeue()
    If v is goal
      return "found the goal"
    //processing all the neighbours of v
```

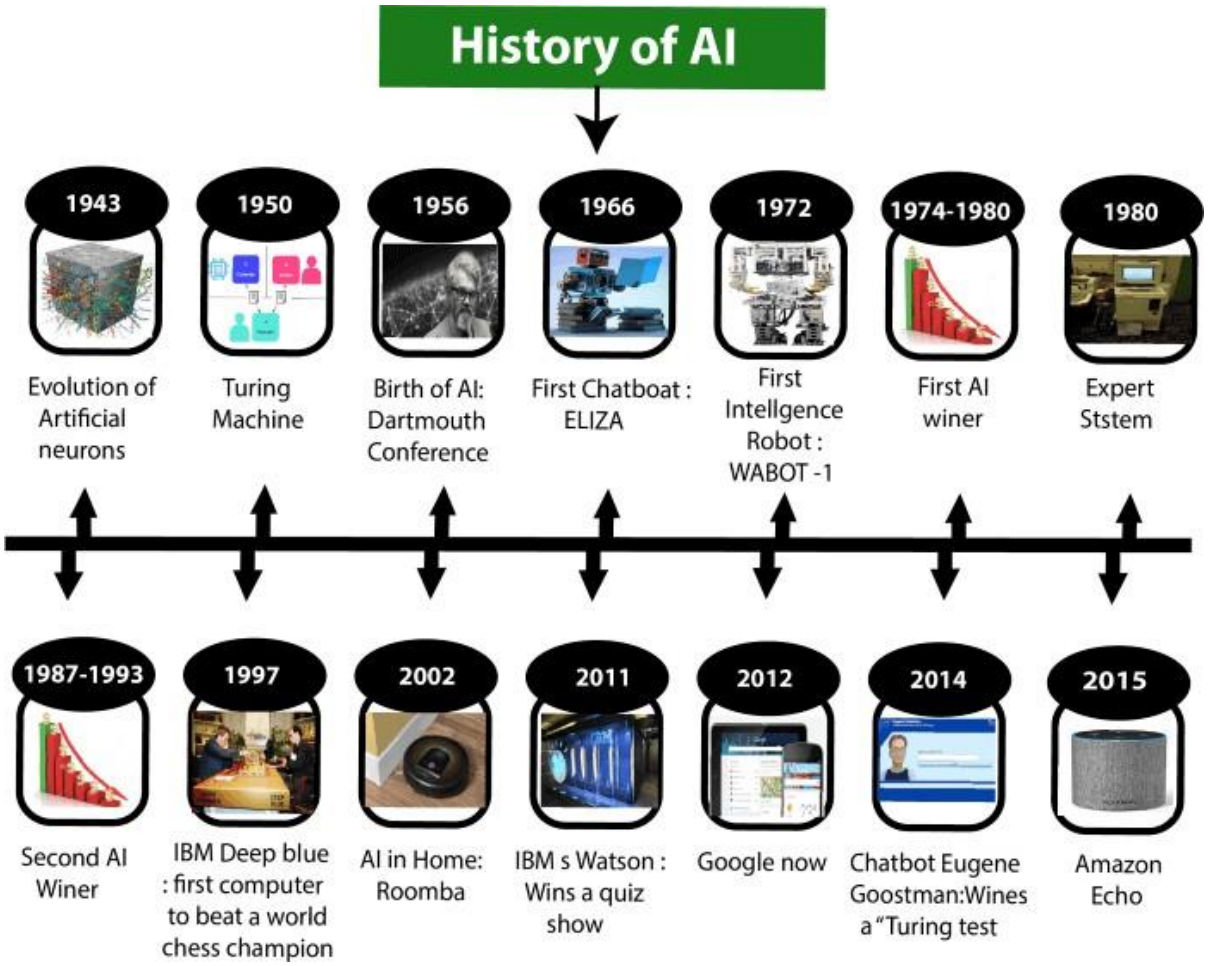
```
    for all neighbours w of v in Graph G
      currentCost = dist[v,w]
      newCost = v.costSoFar + currentCost;
      If (newCost  < w.costSoFar) {
        w.costSoFar = newCost ;
        frontier.enqueue(w)
```

So, we have a frontier that, at first, consists just in the start node. Then, at each iteration, we get the an element from the frontier, we move to the next state, and if we reached the goal, we exit with the message "found the goal". Optionally, we can also print the path. Otherwise, we add the new states to the frontier.

Now, depending on how you will get the new state from the frontier, you could implement a classic BFS, a uniform cost search, or an A* algorithm. Here, you could see more clearly the differences between these algorithms and the best algorithm for detecting the shortest path from Arad to Bucharest.

HISTORY OF ARTIFICIAL INTELLIGENCE

Artificial Intelligence is not a new word and not a new technology for researchers. This technology is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths. Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.



Maturation of Artificial Intelligence (1943-1952)

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

The birth of Artificial Intelligence (1952-1956)

- **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program"Which was named as **"Logic Theorist"**. This program had proved38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.

- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.

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

The golden years-Early enthusiasm (1956-1974)

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

The first AI winter (1974-1980)

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

A boom of AI (1980-1987)

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

The second AI winter (1987-1993)

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

The emergence of intelligent agents (1993-2011)

- **Year 1997:** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- **Year 2002:** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.

- **Year 2006:** AI came in the Business world till the year 2006. Companies like Facebook,Twitter, and Netflix also started using AI.

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

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

STATE SPACE REPRESENTATION

In artificial intelligence, a process known as **state space search** is used to explore all potential configurations or states of an instance until one with the necessary feature is found.

A state is a time snapshot representing some aspect of the problem. It is common to practise modelling a problem as a state space or a collection of possible states to understand it better. If an operation can change one state into another, then the two states are connected in the set of states' graph.

Because the state space is implicit—a typical state space graph would be too vast to build and maintain in-memory—state space search often deviates from conventional computer science search techniques. Instead, new nodes are created as they are visited and then frequently destroyed. In combinatorial search, a solution can be either the final state or a **sequence of states** that lead from an initial state to the final state.

Features

- A state space is a set of all possible states that it can reach from the current state.

- The nodes of a state space represent states, and the arcs connecting them represent actions.

- A path is a set of states and the actions that link them in the state space.
- A problem's solution is a node in the graph representing all possible states of the problem.
- Most AI techniques are based on state space representation.

State space representation

State Space Representation consists of identifying an INITIAL STATE (from where to begin) and a GOAL STATE (the final destination) and then following a specific sequence of actions (called States). Let's define each one individually.

- State: AI problems can be represented as a set of well-formed states. A state can be an Initial State, a Goal State, and several other possible states generated by applying rules between them.
- In an Al problem, space refers to the exhaustive collection of all conceivable states.
- Search is a technique that moves from the beginning state to the desired state by applying valid rules while traversing the space of all possible states.
- A search tree is a tree-like depiction of the search issue. The initial state corresponds to the root node of the search tree, which serves as the tree's starting point.
- It provides the agent with a description of all available actions.
- Transition model: A transition model describes what each action does.
- Path Cost is a function that assigns a cost value to each path.
- It is an activity sequence that connects the beginning node to the end node.
- The optimal option is the one with the lowest cost among all alternatives.

Conclusion

State space representation is beneficial in Al issues because it makes it easy to determine the solution path that leads from the beginning state to the objective state. Therefore, the fundamental task is to develop algorithms that can search across the issue space and discover the optimum solution path.

SHORT ANSWER QUESTIONS

1.EXPLAIN THE CONCEPT OF AI

2.HISTORY OF AI

3.SCOPE OF AI

4.ENVIRONMENT OF AI

5.PROBLEMS FORMULATION OF AI

LONG ANSWER QUSTIONS

1.STATE SPACE REPRESENTATIONS OF AI

2.REVIEW OF AI

3.EXPLAIN SEARCH GRAPH AND SEARCH TREE

4.AGENTS CONCEPT OF AI

5.EXPLAIN AI. AND BENFITS OF AI

UNIT- 2

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.

Anderson in 1953 reviewed the progress of methods in finding maximum or minimum of problems using a series of guesses distributed with a certain order or pattern in the parameter searching space, e.g. a confounded design with exponentially distributed spacings/steps.[1] This search goes on sequentially on each parameter and refines iteratively on the best guesses from the last sequence. The pattern can be a grid (factorial) search of all parameters, a sequential search on each parameter, or a combination of both. The method was developed to screen the experimental conditions in chemical reactions by a number of scientists listed in Anderson's paper. A MATLAB code reproducing the sequential procedure for the general non-linear

regression of an example mathematical model can be found here (FitNGuess @ GitHub).[2]

The name "random search" is attributed to Rastrigin[3] 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.

The algorithm described herein is a type of local random search, where every iteration is dependent on the prior iteration's candidate solution. There are alternative random search methods that sample from the entirety of the search space (for example pure random search or uniform global random search), but these are not described in this article.

Random search has been used in artificial neural network for hyper-parameter optimization.[4]

If good parts of the search space occupy 5% of the volume the chances of hitting a good configuration in search space is 5%. The probability of finding at least one good configuration is

above 95% after trying out 60 configurations (, making use of the

Contents

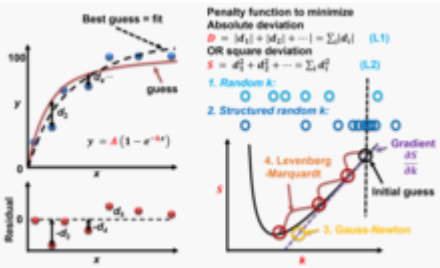
counterprobability).

Algorithm[edit]

Let f: $\mathbb{R}^n \rightarrow \mathbb{R}$ be the fitness or cost function which must be minimized. Let $\mathbf{x} \in \mathbb{R}^n$ designate a position or candidate solution in the search-space. The basic RS algorithm can then be described as:

1. Initialize **x** with a random position in the search-space.
2. Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
    1. 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.)
    2. If $f(\mathbf{y}) < f(\mathbf{x})$ then move to the new position by setting **x** = **y**

Variants[edit]



Scheme of random search using a non-linear regression problem as an example. The goal is to minimize the value of the penalty function. The right bottom shows a few example methods: 1. Non-structured random search, 2. structured random search, 3. Gauss-Newton algorithm, and 4. Levenberg-Marquardt algorithm. 1,2 do not need to know the gradient and 3,4 have to calculate the gradient and usually minimize on both A and k parameters at the same time (scheme only shows the k dimension).

Truly random search is purely by luck and varies from very costive to very lucky, but

the
structured random search is strategic. A number of RS variants have been introduced in the literature with structured sampling in the searching space:

- Friedman-Savage procedure: Sequentially search each parameter with a set of guesses that have a space pattern between the initial guess and the boundaries.[5] An example of exponentially distributed steps can be found here in a MATLAB code (FigNGuess @ GitHub).[2] This example code converges 1-2 orders

slower than the Levenberg–Marquardt algorithm, with an example also provided in the GitHub.

- Fixed Step Size Random Search (FSSRS) is Rastrigin's [3] basic algorithm which samples from a hypersphere of fixed radius.
- Optimum Step Size Random Search (OSSRS) by Schumer and Steiglitz [6] is primarily a theoretical study on how to optimally adjust the radius of the hypersphere so as to allow for speedy convergence to the optimum. The actual implementation of the OSSRS needs to approximate this optimal radius by repeated sampling and is therefore expensive to execute.
- Adaptive Step Size Random Search (ASSRS) by Schumer and Steiglitz [6] attempts to heuristically adapt the hypersphere's radius: two new candidate solutions are generated, one with the current nominal step size and one with a larger step-size. The larger step size becomes the new nominal step size if and only if it leads to a larger improvement. If for several iterations neither of the steps leads to an improvement, the nominal step size is reduced.
- Optimized Relative Step Size Random Search (ORSSRS) by Schrack and Choit [7] approximate the optimal step size by a simple exponential decrease. However, the formula for computing the decrease factor is somewhat complicated.

2. Explain depth first search?

Depth-first search (**DFS**) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. Extra memory, usually a stack, is needed to keep track of the nodes discovered so far along a specified branch which helps in backtracking of the graph.

A version of depth-first search was investigated in the

Contents

19th century by French mathematician Charles Pierre Trémaux[1] as a strategy for solving mazes.[2][3]

Properties[edit]

The time and space analysis of DFS differs according to its application area. In theoretical

computer science, DFS is typically used to traverse an entire graph, and takes time

,[4] where is the number of vertices and the number of edges. This is linear in the size

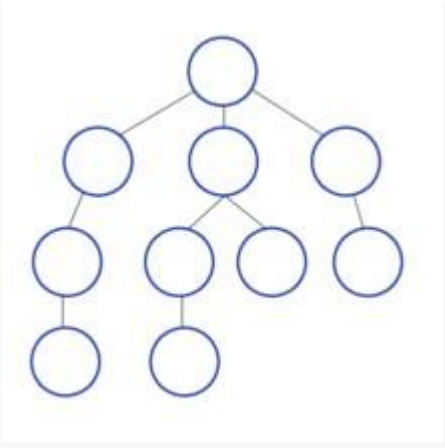
of the graph. In these applications it also uses space in the worst case to store the stack ofvertices on the current search path as well as the set of already-visited vertices. Thus, in this setting, the time and space bounds are the same as for breadth-first search and the choice of

which of these two algorithms to use depends less on their complexity and more on the different properties of the vertex orderings the two algorithms produce.

For applications of DFS in relation to specific domains, such as searching for solutions in artificial intelligence or web-crawling, the graph to be traversed is often either too large to visit in its entirety or infinite (DFS may suffer from non-termination). In such cases, search is only performed to a limited depth; due to limited resources, such as memory or disk space, one typically does not use data structures to keep track of the set of all previously visited vertices.
When search is performed to a limited depth, the time is still linear in terms of the number of expanded vertices and edges (although this number is not the same as the size of the entire graph because some vertices may be searched more than once and others not at all) but the space complexity of this variant of DFS is only proportional to the depth limit, and as a result, is much smaller than the space needed for searching to the same depth using breadth-first search. For such applications, DFS also lends itself much better to heuristic methods for choosing a likely-looking branch. When an appropriate depth limit is not known a priori, iterative deepening depth-first search applies DFS repeatedly with a sequence of increasing limits. In the artificial intelligence mode of analysis, with a branching factor greater than one, iterative deepening increases the running time by only a constant factor over the case in which the correct depth limit is known due to the geometric growth of the number of nodes per level.

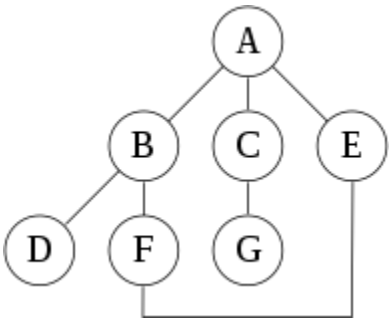
DFS may also be used to collect a sample of graph nodes. However, incomplete DFS, similarly to incomplete BFS, is biased towards nodes of high degree.

Example[edit]



Animated example of a depth-first search

For the following graph:



a depth-first search starting at the node A, assuming that the left edges in the shown graph are chosen before right edges, and assuming the search remembers previously visited nodes and will not repeat them (since this is a small graph), will visit the nodes in the following order: A, B, D, F, E, C, G. The edges traversed in this search form a Trémaux tree, a structure with important applications in graph theory. Performing the same search without remembering previously visited

game search:

- Read
- Discuss

Game Playing is an important domain of artificial intelligence. Games don’t require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve –

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

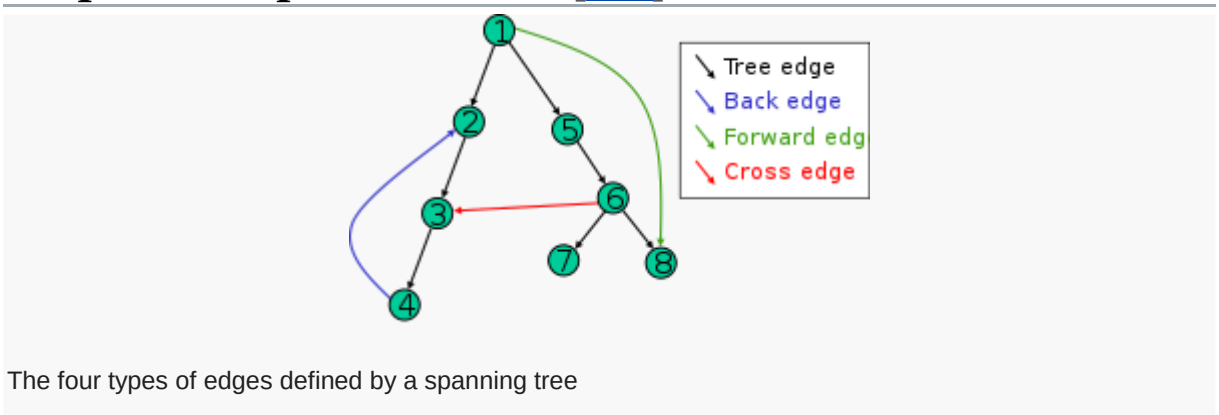
The most common search technique in game playing is Minimax search procedure. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.
Minimax algorithm uses two functions –

MOVEGEN : It generates all the possible moves that can be generated from the current position.
STATICEVALUATION : It returns a value depending upon the goodness from the viewpoint of two-player
This algorithm is a two player game, so we call the first player as PLAYER1 and second player as PLAYER2. The value of each node is backed-up from its children. For PLAYER1 the backed-up value is the maximum value of its children and for PLAYER2 the backed-up value is the minimum value of its children. It provides most promising move to PLAYER1, assuming that the PLAYER2 has make the best move. It is a recursive algorithm, as same procedure occurs at each level.

nodes results in visiting the nodes in the order A, B, D, F, E, A, B, D, F, E, etc. forever, caught in the A, B, D, F, E cycle and never reaching C or G.

Iterative deepening is one technique to avoid this infinite loop and would reach all nodes.

Output of a depth-first search[edit]



The four types of edges defined by a spanning tree

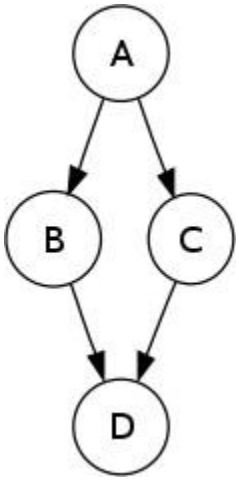
The result of a depth-first search of a graph can be conveniently described in terms of a spanning tree of the vertices reached during the search. Based on this spanning tree, the edges of the original graph can be divided into three classes: **forward edges**, which point from a node of the tree to one of its descendants, **back edges**, which point from a node to one of its ancestors, and **cross edges**, which do neither. Sometimes **tree edges**, edges which belong to the spanning tree itself, are classified separately from forward edges. If the original graph is undirected then all of its edges are tree edges or back edges.

Vertex orderings[edit]

It is also possible to use depth-first search to linearly order the vertices of a graph or tree. There are four possible ways of doing this:

- A **preordering** is a list of the vertices in the order that they were first visited by the depth-first search algorithm. This is a compact and natural way of describing the progress of the search, as was done earlier in this article. A preordering of an expression tree is the expression in Polish notation.
- A **postordering** is a list of the vertices in the order that they were *last* visited by the algorithm. A postordering of an expression tree is the expression in reverse Polish notation.
- A **reverse preordering** is the reverse of a preordering, i.e. a list of the vertices in the opposite order of their first visit. Reverse preordering is not the same as postordering.
- A **reverse postordering** is the reverse of a postordering, i.e. a list of the vertices in the opposite order of their last visit. Reverse postordering is not the same as preordering.

For binary trees there is additionally **in-ordering** and **reverse in-ordering**.



Reverse postordering produces a topological sorting of any directed acyclic graph. This ordering is also useful in control-flow analysis as it often represents a natural linearization of the control flows. The graph above might represent the flow of control in the code fragment below, and it is natural to consider this code in the order A B C D or A C B D but not natural to use the order A B D C or A C D B.

3. Explain breadth first search?

In this article, we will discuss the BFS algorithm in the data structure. Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

There are many ways to traverse the graph, but among them, BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a tree or graph data structure. BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.

Applications of BFS algorithm

The applications of breadth-first-algorithm are given as follows -

- BFS can be used to find the neighboring locations from a given source location.
- In a peer-to-peer network, BFS algorithm can be used as a traversal method to find all the neighboring nodes. Most torrent clients, such as BitTorrent, uTorrent, etc. employ this process to find "seeds" and "peers" in the network.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page and follows the links associated with the page. Here, every web page is considered as a node in the graph.

- BFS is used to determine the shortest path and minimum spanning tree.
- BFS is also used in Cheney's technique to duplicate the garbage collection.
- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

Algorithm

The steps involved in the BFS algorithm to explore a graph are given as follows -

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

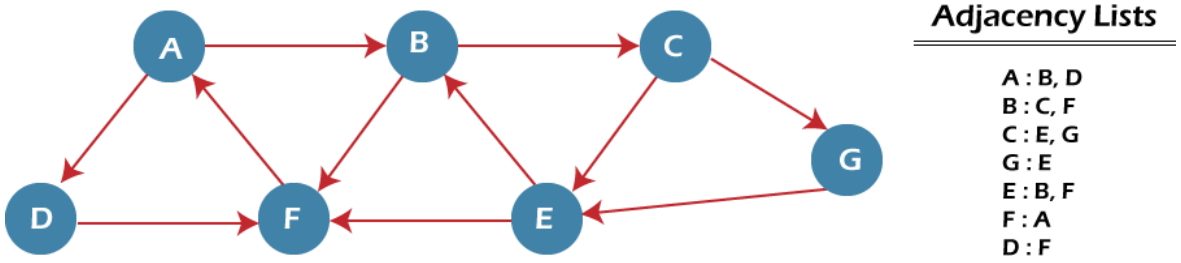
(waiting state)

[END OF LOOP]

Step 6: EXIT

Example of BFS algorithm

Now, let's understand the working of BFS algorithm by using an example. In the example given below, there is a directed graph having 7 vertices.



In the above graph, minimum path 'P' can be found by using the BFS that will start from Node A and end at Node E. The algorithm uses two queues, namely QUEUE1 and

QUEUE2. QUEUE1 holds all the nodes that are to be processed, while QUEUE2 holds all the nodes that are processed and deleted from QUEUE1.

Now, let's start examining the graph starting from Node A.

Step 1 - First, add A to queue1 and NULL to queue2.

1. QUEUE1 = {A}
2. QUEUE2 = {NULL}

Step 2 - Now, delete node A from queue1 and add it into queue2. Insert all neighbors of node A to queue1.

1. QUEUE1 = {B, D}
2. QUEUE2 = {A}

Step 3 - Now, delete node B from queue1 and add it into queue2. Insert all neighbors of node B to queue1.

1. QUEUE1 = {D, C, F}
2. QUEUE2 = {A, B}

Step 4 - Now, delete node D from queue1 and add it into queue2. Insert all neighbors of node D to queue1. The only neighbor of Node D is F since it is already inserted, so it will not be inserted again.

1. QUEUE1 = {C, F}
2. QUEUE2 = {A, B, D}

Step 5 - Delete node C from queue1 and add it into queue2. Insert all neighbors of node C to queue1.

1. QUEUE1 = {F, E, G}
2. QUEUE2 = {A, B, D, C}

Step 5 - Delete node F from queue1 and add it into queue2. Insert all neighbors of node F to queue1. Since all the neighbors of node F are already present, we will not insert them again.

1. QUEUE1 = {E, G}
2. QUEUE2 = {A, B, D, C, F}

Step 6 - Delete node E from queue1. Since all of its neighbors have already been added, so we will not insert them again. Now, all the nodes are visited, and the target node E is encountered into queue2.

1. QUEUE1 = {G}
2. QUEUE2 = {A, B, D, C, F, E}

Complexity of BFS algorithm

Time complexity of BFS depends upon the data structure used to represent the graph. The time complexity of BFS algorithm is **O(V+E)**, since in the worst case, BFS algorithm explores every node and edge. In a graph, the number of vertices is O(V), whereas the number of edges is O(E).

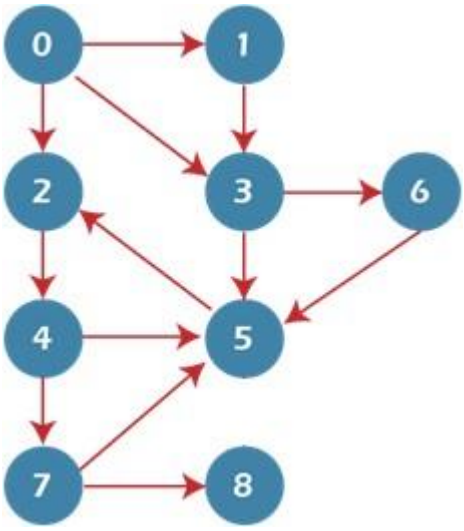
The space complexity of BFS can be expressed as **O(V)**, where V is the number of vertices.

Implementation of BFS algorithm

Now, let's see the implementation of BFS algorithm in java.

In this code, we are using the adjacency list to represent our graph. Implementing the Breadth-First Search algorithm in Java makes it much easier to deal with the adjacency list since we only have to travel through the list of nodes attached to each node once the node is dequeued from the head (or start) of the queue.

In this example, the graph that we are using to demonstrate the code is given as follows -




```
1. import java.io.*;
2. import java.util.*;
```

```
3.  public class BFSTraversal
4.  {
5.      private int vertex;       /* total number number of vertices in the graph */
6.      private LinkedList<Integer> adj[];      /* adjacency list */
7.      private Queue<Integer> que;          /* maintaining a queue */
8.      BFSTraversal(int v)
9.      {
10.         vertex = v;
11.         adj = new LinkedList[vertex];
12.         for (int i=0; i<v; i++)
13.         {
14.             adj[i] = new LinkedList<>();
15.         }
16.         que = new LinkedList<Integer>();
17.     }
18.     void insertEdge(int v,int w)
19.     {
20.         adj[v].add(w);      /* adding an edge to the adjacency list (edges are bidirectional
    in this example) */
21.     }
22.     void BFS(int n)
23.     {
24.         boolean nodes[] = new boolean[vertex];       /* initialize boolean array for holdi
    ng the data */
25.         int a = 0;
26.         nodes[n]=true;
27.         que.add(n);       /* root node is added to the top of the queue */
28.         while (que.size() != 0)
29.         {
30.             n = que.poll();       /* remove the top element of the queue */
31.             System.out.print(n+" ");    /* print the top element of the queue */
32. for (int i = 0; i < adj[n].size(); i++) /* iterate through the linked list and push all
    neighbors into queue */
33.             {
34.                 a = adj[n].get(i);
35.                 if (!nodes[a])      /* only insert nodes into queue if they have not bee
    n explored already */
```

```
36.                 {
37.                     nodes[a] = true;
38.                    que.add(a);
39.                 }
40.             }
41.         }
42.     }
43.     public static void main(String args[])
44.     {
45.         BFSTraversal graph = new BFSTraversal(10);
46.         graph.insertEdge(0, 1);
47.         graph.insertEdge(0, 2);
48.         graph.insertEdge(0, 3);
49.         graph.insertEdge(1, 3);
50.         graph.insertEdge(2, 4);
51.         graph.insertEdge(3, 5);
52.         graph.insertEdge(3, 6);
53.         graph.insertEdge(4, 7);
54.         graph.insertEdge(4, 5);
55.         graph.insertEdge(5, 2);
56.         graph.insertEdge(6, 5);
57.         graph.insertEdge(7, 5);
58.         graph.insertEdge(7, 8);
59.         System.out.println("Breadth First Traversal for the graph is:");
60.         graph.BFS(2);
61.     }
62. }
```

4. Define A * algorithm?

It is a searching algorithm that is used to find the shortest path between an initial and a final point.

It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal.

It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time.

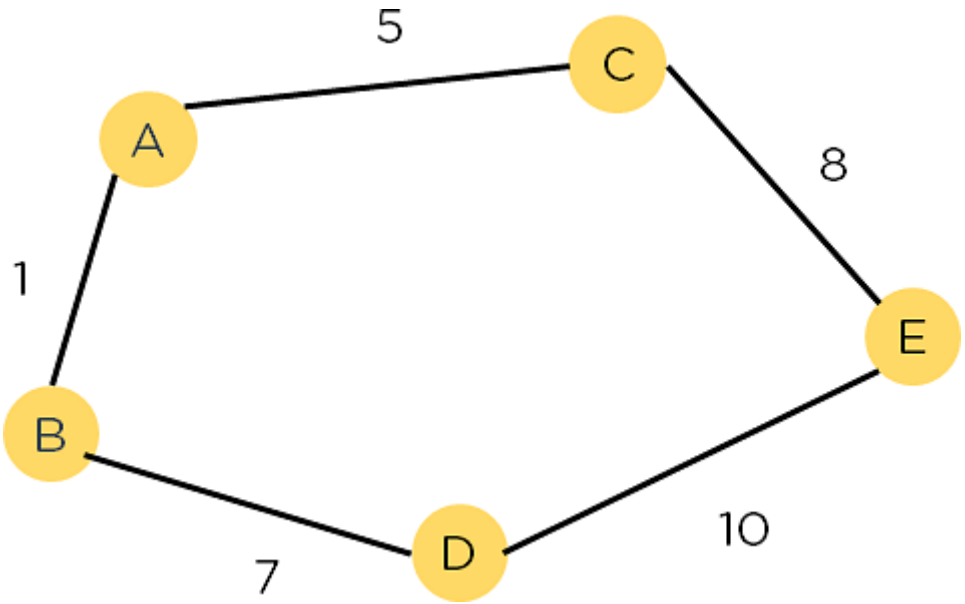


Figure 1: Weighted Graph

A major drawback of the algorithm is its space and time complexity. It takes a large amount of space to store all possible paths and a lot of time to find them.

[Related reading: Top 45 Data Structure Interview Questions and Answers for 2022]

Why A* Search Algorithm?

A* Search Algorithm is a simple and efficient search algorithm that can be used to find the optimal path between two nodes in a graph. It will be used for the shortest

path finding. It is an extension of Dijkstra's shortest path algorithm (Dijkstra's Algorithm). The extension here is that, instead of using a priority queue to store all the elements, we use heaps (binary trees) to store them. The A* Search Algorithm also uses a heuristic function that provides additional information regarding how far away from the goal node we are. This function is used in conjunction with the f-heap data structure in order to make searching more efficient.

Let us now look at a brief explanation of the A* algorithm.

Explanation

In the event that we have a grid with many obstacles and we want to get somewhere as rapidly as possible, the A* Search Algorithms are our savior. From a given starting cell, we can get to the target cell as quickly as possible. It is the sum of two variables' values that determines the node it picks at any point in time.

At each step, it picks the node with the smallest value of 'f' (the sum of 'g' and 'h') and processes that node/cell. 'g' and 'h' is defined as simply as possible below:

- 'g' is the distance it takes to get to a certain square on the grid from the starting point, following the path we generated to get there.
- 'h' is the heuristic, which is the estimation of the distance it takes to get to the finish line from that square on the grid.

Heuristics are basically educated guesses. It is crucial to understand that we do not know the distance to the finish point until we find the route since there are so many things that might get in the way (e.g., walls, water, etc.). In the coming sections, we will dive deeper into how to calculate the heuristics.

Let us now look at the detailed algorithm of A*.

5. Expalin game search algorithm?

It is a searching algorithm that is used to find the shortest path between an initial and a final point.

It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal.

It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem.

Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time.

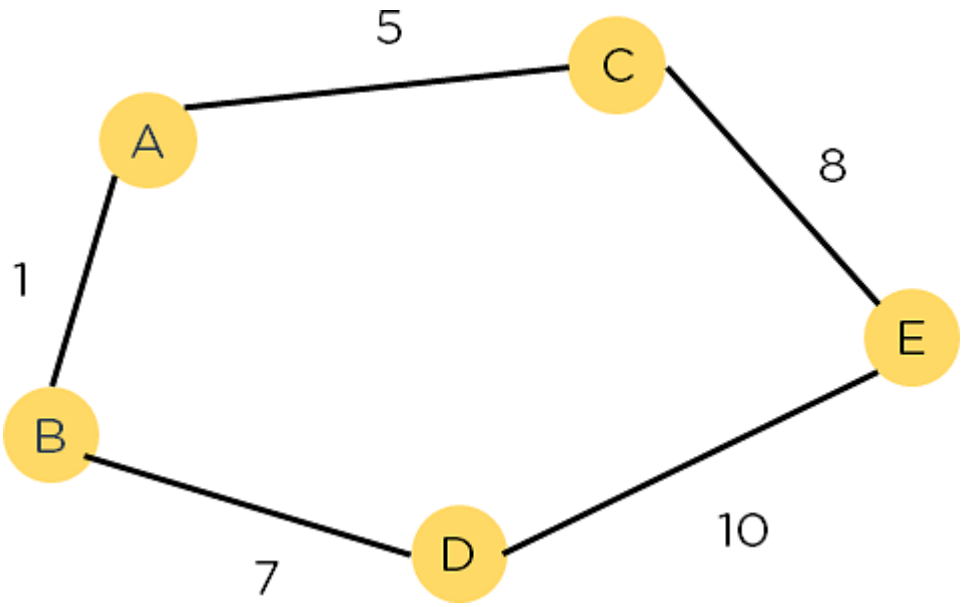


Figure 1: Weighted Graph

A major drawback of the algorithm is its space and time complexity. It takes a large amount of space to store all possible paths and a lot of time to find them.

Short answer questions

1.explain about random search

2.explain about best first search

3.explain about game search

4.explain heuristic search

5.explain about A* algorithm

Long answer questions

1.explain about search with closed and opened list

2.explain about depth first search

3.explain about breadth first search

4.explain about random serach

5.explanabout game theory

Unit-3

PROBABILITY

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 uncertainknowledge:

- **Bayes' rule**
- **Bayesian Statistics**

Note: We will learn the above two rules in later chapters.

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

Probability: Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1. $0 \leq P(A) \leq 1$, where P(A) is the probability of an event A.
1. $P(A) = 0$, indicates total uncertainty in an event A.
1. $P(A) = 1$, indicates total certainty in an event A.

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

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$ = probability of a not happening event.
- $P(\neg A) + P(A) = 1$.

Event: Each possible outcome of a variable is called an event.

Sample space: The collection of all possible events is called sample space.

Random variables: Random variables are used to represent the events and objects inthe real world.

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

Conditional probability is a probability of occurring an event when another event hasalready 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:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Where P($A \wedge B$)= Joint probability of a

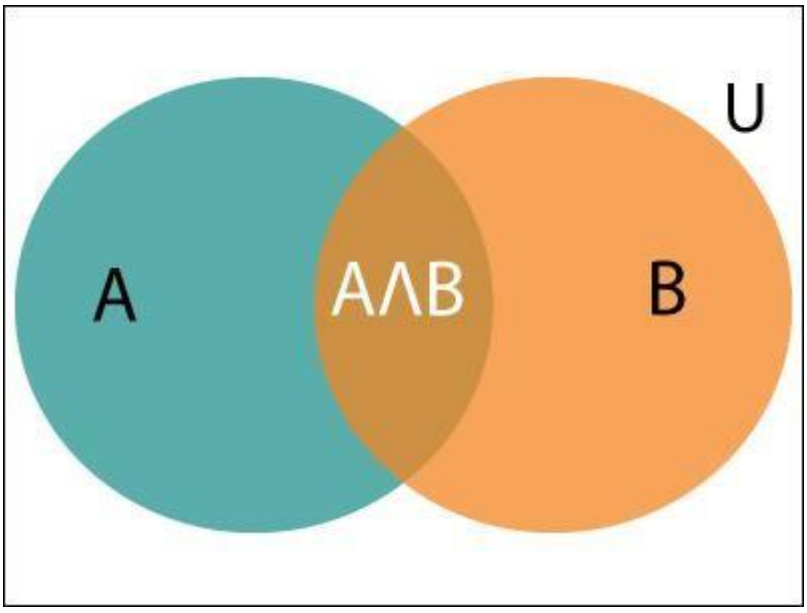
and BP(B)= Marginal probability of

B.

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

$$P(B|A) = \frac{P(A \wedge B)}{P(A)}$$

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



Example:

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

Solution:

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

2. Write a short note on uncertainty?

To act rationally under uncertainty we must be able to evaluate how likely certain things are. With FOL a fact F is only useful if it is known to be true or false. But we need to be able to evaluate how likely it is that F is true. By weighing likelihoods of events (probabilities) we can develop mechanisms for acting rationally under uncertainty.

Dental Diagnosis example.

In FOL we might formulate

P. symptom(P,toothache)→ disease(p,cavity) disease(p,gumDisease)

disease(p,foodStuck)

When do we stop?

Cannot list all possible causes.

We also want to rank the possibilities. We don't want to start drilling for a cavity before checking for more likely causes first.

Axioms Of Probability

Given a set U (universe), a probability function is a function defined over the subsets of U that maps each subset to the real numbers and that satisfies the Axioms of Probability

1.Pr(U) = 1

2.Pr(A) [0,1]

3.Pr(A ∈B) = Pr(A) + Pr(B) –Pr(A ∩B)

U

Note if A ∩B = {} then Pr(A UB) = Pr(A) + Pr(B)

CONDITIONAL PROBABILITY

Theorem: If A and B are two dependent events then the probability of occurrence of A given that B has already occurred and is denoted by P(A/B) is given by

$$P\binom{A}{B} = \frac{P(A\cap B)}{P(B)}; P(A \cap B) = P(B)xP\binom{A}{B}$$

Similarly, the probability of occurrence of B given that A has already occurred is given by

$$P\binom{B}{A} = \frac{P(A\cap B)}{P(A)};\ P(A \cap B) = P(A)xP\binom{B}{A}$$

Proof: Let S be the sample space. Then, we have

$$P(A \cap B) = \frac{P(A\cap B)}{P(S)} = \frac{P(A\cap B)}{P(A)} x \binom{P(A)}{P(S)}$$

$$= \binom{P(A)}{P(S)} x \frac{P(A\cap B)}{P(A)} \qquad \because \frac{P(A\cap B)}{P(A)} = P\binom{B}{A}$$

$$= P(A)xP\binom{B}{A} \text{.....equation (i)}$$

Interchange A and B in equation (i), we get

$$= P(B)\ x\ P\binom{A}{B}$$

Example: Find the probability of drawing a heart on each of two consecutive draws from well shuffled-packs of cards if the card is not replaced after the draw.

Solution: Let event A is a heart on the first draw, and event B is a heart on the second draw.

Then, $P(A \cap B) = P(A) \times P\binom{B}{A}$

Now, $P(A) = \frac{13}{52}$

When we get a heart on the first draw, the second draw has 51 outcomes and 12 are favorable.

$$P\binom{B}{A} = \frac{12}{51} = \frac{4}{17}$$

$$P(A \cap B) = \frac{13}{52} \times \frac{4}{17} = \frac{1}{17}$$

CONSTRAINT SATISFACTION

We have encountered a wide variety of methods, including adversarial search and instant search, to address various issues. Every method for issue has a single purpose in mind: to locate a remedy that will enable that achievement of the objective. However there were no restrictions just on bots' capability to resolve issues as well as arrive at responses in adversarial search and local search, respectively.

These section examines the constraint optimization methodology, another form or real concern method. By its name, constraints fulfilment implies that such an issue must be solved while adhering to a set of restrictions or guidelines.

Whenever a problem is actually variables comply with stringent conditions of principles, it is said to have been addressed using the solving multi - objective method. Wow what a method results in a study sought to achieve of the intricacy and organization of both the issue.

Three factors affect restriction compliance, particularly regarding:

REVIEW OF PROBABILTY

- Natural way to represent uncertainty
- People have intuitive notions about probabilities
- Many of these are wrong or inconsistent
- Most people don’t get what probabilities mean
- Understanding Probabilities
- Initially, probabilities are “relative frequencies”
- This works well for dice and coin flips
- For more complicated events, this is problematic

- What is the probability that Obama will be reelected?
- This event only happens once
- We can’t count frequencies
-
- still seems like a meaningful question

- In general, all events are unique

Probabilities and Beliefs

• Suppose I have flipped a coin and hidden the outcome

• What is P(Heads)?

• Note that this is a statement about a belief, not a statement about the world

• The world is in exactly one state (at the macro level) and it is in that state with probability 1.

• Assigning truth values to probability statements is very tricky business

• Must reference speakers state of knowledge

Frequentism and Subjectivism

• Frequentists hold that probabilities must come from relative frequencies

• This is a purist viewpoint

• This is corrupted by the fact that relative frequencies are often unobtainable

• Often requires complicated and convoluted

• assumptions to come up with probabilities

• Subjectivists: probabilities are degrees of belief

o Taints purity of probabilities

o Ofen more practical

Types are:

1 Unconditional or prior probabilities

2 Conditional or posterior probabilities

2. PROBABILISTIC REASONING

- Representing Knowledge in an Uncertain Domain

- Belief network used to encode the meaningful dependence between variables. ○ Nodes represent random variables

○ Arcs represent direct influence

- Nodes have conditional probability table that gives that var's probability given the different states of its parents
- Is a Directed Acyclic Graph (or DAG)

The Semantics of Belief Networks

- To construct net, think of as representing the joint probability distribution.
- To infer from net, think of as representing conditional independence statements.
- Calculate a member of the joint probability by multiplying individual conditional probabilities:
- P(X1=x1, . . . Xn=xn) =
- = P(X1=x1|parents(X1)) * . . . * P(Xn=xn|parents(Xn))
- Note: Only have to be given the immediate parents of Xi, not all other nodes:
- P(Xi|X(i-1),...X1) = P(Xi|parents(Xi))

- To incrementally construct a network:

1. Decide on the variables
2. Decide on an ordering of them
3. Do until no variables are left:

a. Pick a variable and make a node for it

b. Set its parents to the minimal set of pre-existing nodes

c. Define its conditional probability

- Often, the resulting conditional probability tables are much smaller than the exponential size of the full joint

- If don't order nodes by "root causes" first, get larger conditional probability tables
- Different tables may encode the same probabilities.
- Some canonical distributions that appear in conditional probability tables:

○ deterministic logical relationship (e.g. AND, OR) ○ deterministic numeric relationship (e.g. MIN)

○ parameteric relationship (e.g. weighted sum in neural net) ○ noisy logical relationship (e.g. noisy-OR, noisy-MAX)

Direction-dependent separation or D-separation:

- If all undirected paths between 2 nodes are d-separated given evidence node(s) E, then the 2 nodes are independent given E.

- Evidence node(s) E d-separate X and Y if for every path between them E contains a node Z that:

○ has an arrow in on the path leading from X and an arrow out on the path leading to Y (or vice versa)

○ has arrows out leading to both X and Y

○ does NOT have arrows in from both X and Y (nor Z's children too)

Inference in Belief Networks

- Want to compute posterior probabilities of query variables given evidence variables.

- Types of inference for belief networks:

○ Diagnostic inference: symptoms to causes ○ Causal inference: causes to symptoms

○ Intercausal inference:

○ Mixed inference: mixes those above

Inference in Multiply Connected Belief Networks

- Multiply connected graphs have 2 nodes connected by more than one path
- Techniques for handling:

○ Clustering: Group some of the intermediate nodes into one meganode. Pro: Perhaps best way to get exact evaluation.

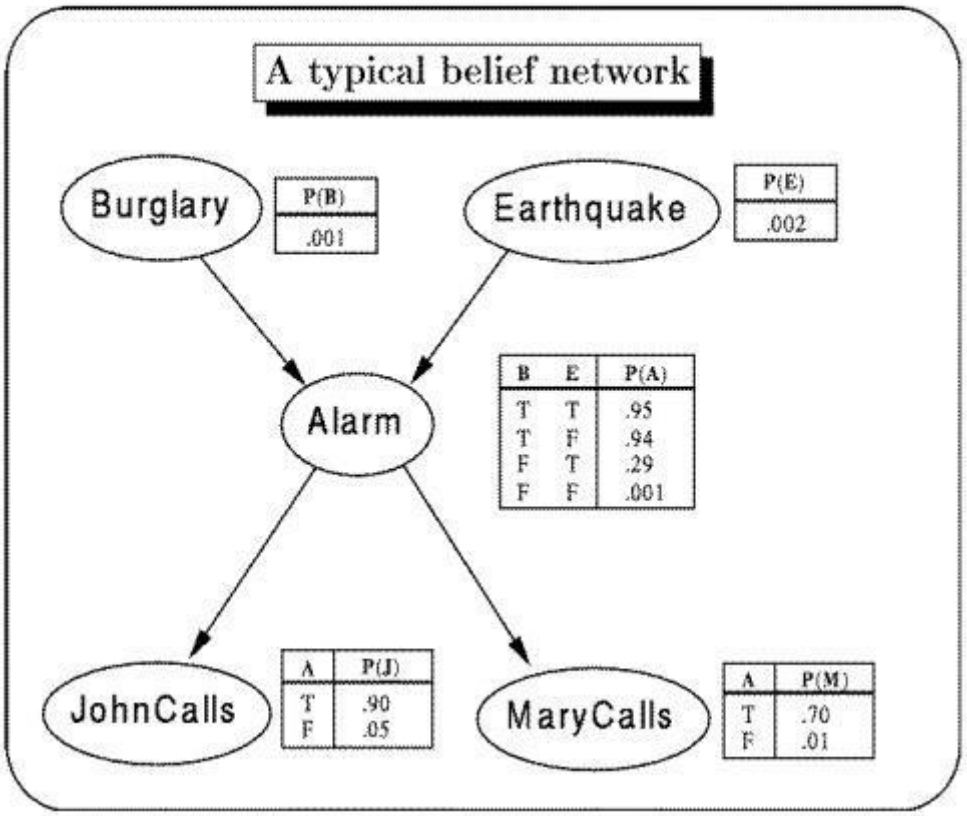
Con: Conditional probability tables may exponentially increase in size.

○ Cutset conditioning: Obtain simplier polytrees by instantiating variables as constants.

Con: May obtain exponential number of simplier polytrees.

Pro: It may be safe to ignore trees with lo probability (bounded cutset conditioning).

Stochastic simulation: run thru the net with randomly choosen values for each node (weighed by prior probabilities).



4. BAYESIAN NETWORK

Bayes' nets:

A technique for describing complex joint distributions (models) using simple, local distributions

(conditional probabilities)

More properly called graphical models

Local interactions chain together to give global indirect interactions

A Bayesian network is a graphical structure that allows us to represent and reason about an uncertain domain. The nodes in a Bayesian network represent a set of random variables,

X=X1;::Xi;:::Xn, from the domain. A set of directed arcs(or links) connects pairs of nodes, Xi!Xj, representing the direct dependencies between variables.

Assuming discrete variables, the strength of the relationship between variables is quantified by conditional probability distributions associated with each node. The only constraint on the arcs allowed in a BN is that there must not be any directed cycles: you cannot return to a node simply by following directed arcs.

Such networks are called directed acyclic graphs, or simply dags. There are a number of steps that a knowledge engineer must undertake when building a Bayesian network. At this stage we will present these steps as a sequence; however it is important to note that in the real-world the process is not so simple.

Nodes and values

First, the knowledge engineer must identify the variables of interest. This involves answering the question: what are the nodes to represent and what values can they take, or what state can they be in? For now we will consider only nodes that take discrete values. The values should be both mutually exclusive and exhaustive , which means that the variable must take on exactly one of these values at a time. Common types of discrete nodes include:

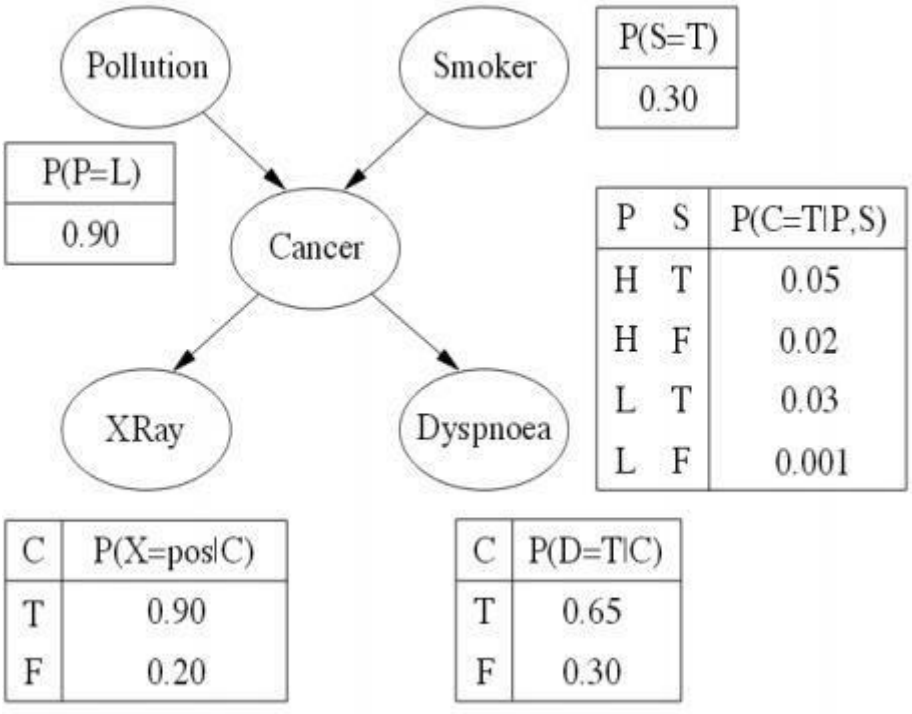
Boolean nodes, which represent propositions, taking the binary values true (T) and false (F). In a medical diagnosis domain, the node Cancer would represent the proposition that a patient has cancer.

Ordered values. For example, a node Pollution might represent a patient's pollution exposure and take the values low, medium, high

Integral values. For example, a node called Age might represent a patient's age and have possible values from 1 to 120.

Even at this early stage, modeling choices are being made. For example, an alternative to representing a patient's exact age might be to clump patients into

different age groups, such as baby, child, adolescent, young, middleaged, old. The trick is to choose values that represent the domain efficiently.



1 Representation of joint probability distribution

2 Conditional independence relation in Bayesian network

3. Expalin 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:

1. P(A ∧ B)= P(A|B) P(B) or

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

1. P(A ∧ B)= P(B|A) P(A)

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

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.

P(A) is called the **prior probability**, probability of hypothesis before considering the evidence

P(B) is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write P (B) = P(A)*P(B|Ai), hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A_1, A_2, A_3,. , A_n is a set of mutually exclusive and exhaustive events.

Applying Bayes' rule:

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

$$\mathbf{P(cause|effect)} = \frac{\mathrm{P(effect|cause)\,P(cause)}}{\mathrm{P(effect)}}$$

Example-1:

Question: what is the probability that a patient has diseases meningitis with a stiff neck?

Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let a be the proposition that patient has stiff neck and b be the proposition that patient has meningitis. , so we can calculate the following as:

P(a|b) = 0.8

P(b) = 1/30000

P(a)= .02

$$\mathbf{P(b|a)} = \frac{\mathrm{P(a|b)P(b)}}{\mathrm{P(a)}} = \frac{0.8*(\frac{1}{30000})}{0.02} = 0.001333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

Example-2:

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

Solution:

$$P(king|face) = \frac{P(Face|king)*P(King)}{P(Face)} \quad .....(i)$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

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

$$P(king|face) = \frac{1*(\frac{1}{13})}{(\frac{3}{13})} = 1/3,$$ **it is a probability that a face card is a king card.**

Application of Bayes' theorem in Artificial intelligence:

Following are some applications of Bayes' theorem:

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

G	It will always be the case that...
F	It will sometimes be the case that...
H	It has always been the case that...
P	It has at some time operators the case that...

Bayes' theorem in Artificial intelligence

Bayes' theorem:

Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

Example: If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

$P(A \wedge B) = P(A|B)\, P(B)$ or

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

$P(A \wedge B) = P(B|A)\, P(A)$

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

$$P(A|B) = \frac{P(B|A)\, P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

Bayesian Belief Network in artificial intelligence

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

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

It is also called a **Bayes network, belief network, decision**

network, or **Bayesian model**.

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

What is a Hidden Markov Model?

Hidden Markov Model (HMM) is a statistical Markov model in which the model states are hidden. It is important to understand that the state of the model, and not the parameters of the model, are hidden. A Markov model with fully known parameters is still called a HMM. While the model state may be hidden, the state-dependent output of the model is visible. Information about the state of the model can be gleaned from the probability distribution over possible output tokens because each model state creates a different distribution. A sequence of output tokens will provide insight into the sequence of states in a process known as pattern theory.

Why is this Useful?

The main usefulness of HMM is the recovery of a data sequence that is hidden by observing the output which is dependent on that hidden data sequence.

Example

Two people, let's call them Isla and Donnie, talk about food they like to eat. Donnie likes to eat pizza, pasta and pie. He tends to choose which to eat depending on his emotions. Isla has a rough understanding of the likelihood that Donnie is happy or upset and his tendency to pick food based on those emotions. Donnie's food choice is the Markov process and Isla knows the parameters but she does not know the state of Donnie's emotions; this is a hidden Markov model. When they talk, Isla can determine the probability of Donnie being either happy or upset based on which of the three foods he chose to eat at a given moment.

Applications of Hidden Markov Model

Computational finance
Cryptanalysis
Speech recognition – Notably Apple's Siri
Handwriting recognition
Time series analysis

Short answer questions

1.what is probability

2.explain about the conditional probability

3.Explain about the constraint satisfaction

4.explain about the construction and interface in

AI

5.EXPLAIN ABOUT THE NETWORK S REPRESENTATION

LONG ANSWER QUESTIONS

1.PLAIN ABOUT THE HIDDEN MARKOV MODEL

2.EXPLAIN ABOUT THE BAYES RULES IN AI

3.EXLAIN ABOUT THE BAYESIAN NETWORKS-REPRESENTATION

4.WHAT IS PROOSITIONAL LOGICS

UNIT-4

1.Explain MDP Formulation?

In mathematics, a **Markov decision process** (**MDP**) is a discrete-time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partlyunder the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming. MDPs were known at least as early as the 1950s;[1] a core body of research on Markov decision processes resulted from Ronald Howard's 1960 book, *Dynamic Programming and Markov Processes*.[2] They are used in many disciplines, including robotics, automatic control, economics and manufacturing. The name of MDPs comes from the Russianmathematician Andrey Markov as they are an extension of Markov chains.

At each time step, the process is in some state , and the decision maker may choose any action that is available in state . The process responds at thenext time step by randomly moving into a new state, and giving the decision maker a corresponding reward .

The probability that the process moves into its new state is influenced by the chosen action. Specifically, it is given by the state transition function. Thus, thenext state depends on the current stateand the decision maker's

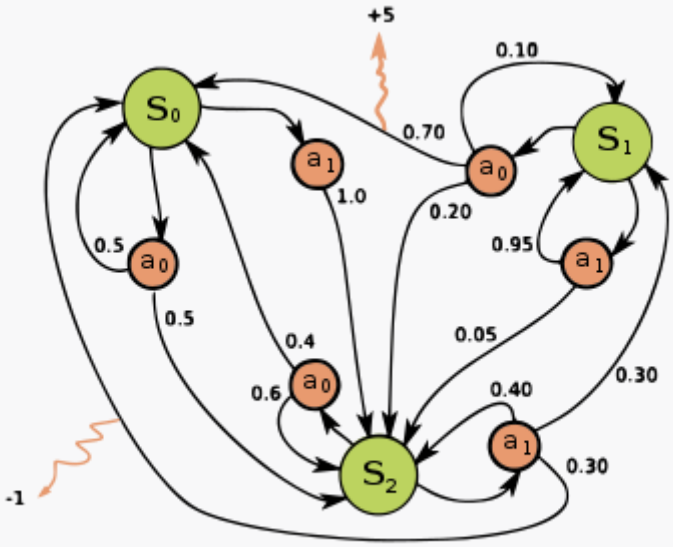
action But given and , it is conditionally independent of all previous states and actions; in other words, the state transitions of an MDP satisfy the Markovproperty.

Markov decision processes are an extension of Markov chains; the difference is theaddition of actions (allowing choice) and rewards (giving motivation). Conversely, if only one action exists for each state

(e.g. "wait") and all rewards are the same (e.g."zero"), a Markov decision process reduces to a Markov chain.

Contents

Definition[edit]



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

A Markov decision process is a 4-[tuple](), where:

- is a [set]() of states called the *state space*,
- is a set of actions called the *action space* (alternatively, is the set of actions available from state),
- is the probability that action in state at time will lead to state at time ,
- is the immediate reward (or expected immediate reward) received after transitioning from state to state , due to action

The state and action spaces may be finite or infinite, for example the [set of real numbers](). Some processes with countably infinite state and action spaces can be reduced to ones with finite state and action spaces.[3]

A policy function is a (potentially probabilistic) mapping from state space ( ) to action space ( ).

Optimization objective[edit]

The goal in a Markov decision process is to find a good "policy" for the decision maker: a function that specifies the action that the decision maker will choose when in state . 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 (since the action chosen in state is completely determined by and reduces to , a Markov transition matrix).

The objective is to choose a policy that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

(where we choose , i.e. actions given by the policy). And the expectation is taken over

where is the discount factor satisfying , which is usually close to 1 (for example, for some discount rate r). A lower discount factor motivates the decision maker to favor taking actions early, rather than postpone them indefinitely.

A policy that maximizes the function above is called an *optimal policy* and is usually denoted . A particular MDP may have multiple distinct optimal policies. Because of the Markov property, it can be shown that the optimal policy is a function of the current state, as assumed above.

UTIITY TEORY

Defines axioms on preferences that involve uncertainty and ways to manipulate them. • Uncertainty is modeled through lotteries – Lottery: • Outcome A with probability p • Outcome C with probability (1-p) • The following six constraints are known as the axioms of utility theory. The axioms are the most obvious semantic constraints on preferences with lotteries. • Notation: - preferable - indifferent (equally preferable)

Axioms of the utility theory • Orderability: Given any two states, the a rational agent prefers one of them, else the two as equally preferable. • Transitivity: Given any three states, if an agent prefers A to B and prefers B to C, agent must prefer A to C. • Continuity: If some state B is between A and C in preference, then there is a p for which the rational agent will be indifferent between state B and the lottery in which A comes with probability p, C with probability (1-p). (A f B) ∨ (B f A) ∨ (A ~ B) (A f B) ∧ (B f C) ⇒ (A f C) (A f B f C) ⇒ ∃p [p : A;(1 − p) : C] ~ B

Utility theory If the agent obeys the axioms of the utility theory, then 1. there exists a real valued function U such that: 2. The utility of the lottery is the expected utility, that is the sum of utilities of outcomes weighted by their probability 3. Rational agent makes the decisions in the presence of uncertainty by maximizing its expected utility

Explain Utility Functions?

In economics, utility represents the satisfaction or pleasure that consumers receive for consuming a good or service. Utility function measures consumers' preferences for a set of goods and services.

Utility is measured in units called utils—the Spanish word for useful— but calculating the benefit or satisfaction that consumers receive is abstract and difficult to pinpoint. As a result, economists measure utility in terms of revealed preferences by observing consumers' choices. From there, economists create an ordering of consumption baskets from least desired to the most preferred.

KEY TAKEAWAYS

- In economics, utility function is an important concept that measures preferences over a set

Utility represents the satisfaction that consumers receive for choosing and consuming aproduct or service.

- Economists track consumer choices to ascertain one product's utility versus another and assign a numerical value to that utility.
- Company executives research consumers' utility to guide the company's sales and marketing plans, advertising, and new product offerings.
- Ordinal utility ranks choices by preference, whereas cardinal utility measures the utility received from a choice.

Understanding Utility Function

In economics, the utility function measures the welfare or satisfaction of a consumer as a function of the consumption of real goods, such as food or clothing. Utility function is widely used in rational choice theory to analyze human behavior.

When economists measure or rank the preferences of consumers, it is referred to as ordinal utility. In other words, the order in which consumers choose one product over another can establish that consumers assign a higher value to the chosen product. Ordinal utility measures how consumers rank products, but it does not measure how much more one ranks above the other.

To better understand ordinal utility, consider the following example. Three contestants vie for first place in a dance competition. Contestant A is declared the winner. Contestant B is the runner-up, and contestant B ranked third. Ordinal utility reveals that the judges preferred contestant A over contestants B and C and contestant B over C. What ordinal function does not tell us is to what degree one was preferred over the other.

Mainly used in microeconomics, cardinal utility assigns a numeric value to the consumer's preference, indicating the *degree* to which one choice ranks above another. Cardinal utility will define how much more contestant A was preferred over contestants B and C, and so on.

When considering utility, it is important to understand the concepts of total utility and marginal utility. Marginal utility measures the satisfaction or benefits a person gets from consuming an additional unit of a product or service. Total utility measures the satisfaction or benefits a person gets from the total consumption—including marginal utility—of a product or service.

If consuming 10 units of a product yields 20 utils, and consuming one additional unit yields 1 util, the total utility is 21 utils. If consuming another unit yields .5 utils, the total utility would then become 21.5 utils.

Economists believe that the amount of satisfaction one receives from each additional unit of consumption diminishes with each unit consumed. This concept is called the law of diminishing marginal utility. Diminishing marginal utility doesn't state that consuming additional units will fail to satisfy the consumer; it states that the satisfaction from consuming more and more units is less than the first additional units consumed.

How to Calculate a Utility Function

Utility functions are expressed as a function of the quantities of a bundle of goods or services. It is often denoted as $U(X_1, X_2, X_3, X_n)$.

A utility function that describes a preference for one bundle of goods (X_a) vs another bundle of goods (X_b) is expressed as $U(X_a, X_b)$.

Where there are perfect complements, the utility function is written as $U(X_a, X_b) = MIN[X_a, X_b]$, where the smaller of the two is assigned the function's value.

In certain situations, the goods may be considered perfect substitutes for each other, and the appropriate utility function must reflect such preferences with a utility form of $U(X_a, X_{b)} = X_a + X_b$.1

Example of Utility Function

Let's say a consumer is shopping for a new car and has narrowed the choice down to two cars. The cars are nearly identical, except the second car has enhanced safety features. As a result, the second car costs $2,000 more than the first car.

The incremental or marginal utility or satisfaction derived from car two could be represented numerically as the $2,000 price difference between the two cars. In other words, the consumer is receiving $2,000 in incremental or marginal utility from car two.

Furthermore, let's say that 100,000 consumers throughout the economy preferred car two to car one. Economists might infer that consumers, overall, received $200 million (100,000 x $2,000) worth of incremental utility from the safety features of car two. Utility is derived from the consumer's belief that they are likely to have fewer accidents due to the added safety features of car two.

Advantages and Disadvantages of Utility Function

Economists can't assign a true numerical value to a consumer's level of satisfaction from a preference or choice. Also, pinpointing the reason for purchase can be difficult; there are usually many variables to consider.

In the previous example, the two cars were nearly identical. In reality, there might be several features or differences between the two cars. As a result, assigning a value to a consumer's preference can be challenging since one consumer might prefer the safety features while another might prefer something else.

Tracking and assigning values to utility can still be useful to economists. Over time, choices and preferences may indicate changes in spending patterns and in utility.

Understanding the logic behind consumer choices and their level of satisfaction is not only important to economists but to companies, as well. Company executives can use utility to track how consumers view their products.

Utility function is essentially a "model" used to represent consumer preferences, so companies often implement them to gain an edge on the competition. For example, studying consumers' utility can help guide management on anything from marketing and sales to product upgrades and new offerings.

Utility Function FAQs

What Is Utility Function?

Utility describes the benefits gained or satisfaction experienced with the consumption of goods or services. Utility function measures the preferences consumers apply to their consumption of goods and services. For instance, if a customer prefers apples to oranges no matter the amount consumed, the utility function could be expressed as U(apples) > U(oranges).

What Is the Difference Between Utility Function and Marginal Utility?

Utility function ranks consumers' consumption of goods or services by preference. Marginal utility measures the change in utility when the rate of consumption changes (i.e., how much more satisfaction is gained by consuming another unit of a good or service).

Why Is Utility Function Important?

Economists use utility function to better understand consumer behaviors, as well as determine how well goods and services provide satisfaction to consumers.

Utility function can also help analysts determine how to distribute goods and services to consumers in a way that total utility is realized.

Companies can use utility function to determine which product(s) within their product line (or that of a competitor) consumers prefer. Knowing these preferences can help management

2.Explain Policy Iteration?

Policy iteration

Pulling together policy evaluation and policy improvement, we can define an *policy Iteration*, which computes an optimal π by performing a sequence of interleaved policy evaluations and improvements:

Algorithm – Policy Iteration

Input: MDP $M=\langle S,s0,A,Pa(s'|s),r(s,a,s')\rangle$
Output: Policy π

Set $V\pi$ to arbitrary value function; e.g., $V\pi(s)=0$ for all s.

Set π to arbitrary policy; e.g. $\pi(s)=a$ for all s, where $a\in A$ is an arbitrary action.

Repeat
 Compute $V\pi(s)$ for all s using policy evaluation
 For each $s\in S$
 $\pi(s)\leftarrow argmax_{a\in A(s)}Q\pi(s,a)$
Until π does not change
The policy iteration algorithm finishes with an optimal π after a finite number of iterations, because the number of policies is finite, bounded by $O(|A||S|)$, unlike value iteration, which can theoretically require infinite iterations.

However, each iteration costs O(|S|2|A|+|S|3). Empirical evidence suggests that the most efficient is dependent on the particular MDP model being solved, but that surprisingly few iterations are often required for policy iteration.

Implementation

Below is a Python implementation for policy iteration. In this implementation, the parameter `max_iterations` is the maximum number of iterations of the policy iteration, and the parameter `theta` the largest amount the value function corresponding to the current policy can change before the policy evaluation look terminates.

```
from tabular_policy import TabularPolicy
from tabular_value_function import TabularValueFunction
from qtable import QTable

class PolicyIteration:
    def __init__(self, mdp, policy):
        self.mdp = mdp
        self.policy = policy

    def policy_evaluation(self, policy, values, theta=0.001):

        while True:
            delta = 0.0
            new_values = TabularValueFunction()
            for state in self.mdp.get_states():
                # Calculate the value of V(s)
                old_value = values.get_value(state)
                new_value = values.get_q_value(self.mdp, state,
policy.select_action(state))
                values.update(state, new_value)
                delta = max(delta, abs(old_value - new_value))

            # terminate if the value function has converged
            if delta < theta:
                break

        return values

    """ Implmentation of policy iteration iteration. Returns the number
of iterations exected """

    def policy_iteration(self, max_iterations=100, theta=0.001):

        # create a value function to hold details
        values = TabularValueFunction()

        for i in range(1, max_iterations + 1):
            policy_changed = False
            values = self.policy_evaluation(self.policy, values, theta)
            for state in self.mdp.get_states():
                old_action = self.policy.select_action(state)
```

```
                q_values = QTable()
                for action in self.mdp.get_actions(state):
                    # Calculate the value of Q(s,a)
                    new_value = values.get_q_value(self.mdp, state,
action)
                    q_values.update(state, action, new_value)

                # V(s) = argmax_a Q(s,a)
                (new_action, _) = q_values.get_max_q(state,
self.mdp.get_actions(state))
                self.policy.update(state, new_action)

                policy_changed = (
                    True if new_action is not old_action else
policy_changed
                )

            if not policy_changed:
                return i

        return max_iterations
```

From this, we can see that policy evaluation looks very similar to value iteration. The main differences is in the inner loop: instead of finding the action with the maximum Q-value, we simply find the value of the action that is given the policy: `policy.select_action(state)`.

We can execute this to get the policy:

```
from gridworld import GridWorld
from policy_iteration import PolicyIteration
from tabular_policy import TabularPolicy


gridworld = GridWorld()
policy = TabularPolicy(default_action=gridworld.LEFT)
PolicyIteration(gridworld, policy).policy_iteration(max_iterations=100)
gridworld.visualise_policy(policy)
```

We can see that this matches the optimal policy according to value iteration.

Let's look at the policies that are generated after each iteration, noting that the initial policy is defined by taking a random action (left) and using that for every state:

3. Explain Utility Theory?

Definition: Utility theory is an economic hypothesis that postulates the fact that consumers make purchase decisions based in the degree of utility or satisfaction they obtain from a given item. This means that the higher the utility level the higher the item will be prioritized in the consumer's budget.

What Does Utility Theory Mean?

This theory states that consumers rank products in their minds whenever they are facing a purchase decision. These ranking function drives their budget allocation, which means that resources are poured into the purchases that will bring the highest degree of satisfaction. It is assumed that individual budgets are limited and therefore there is a limited amount of goods or services that can be purchased, taking this into account, an individual will weigh which of the options currently available within the open market is the best suit to fulfill his current set of needs or desires.

In these cases, preferences also play a key role and these can be defined as a set of predispositions that each individual possesses towards certain brands or products by elements such as colors, shapes, tastes or smells. Finally, there are four essential types of utility and these are form utility, time utility, place utility and possession utility.

Example

Harold is a 45 year old computer engineer that was recently hired by a company called Tech Mogul Co. which is a firm that provide security solutions for information systems, mostly to the banking industry. Harold is considered to be a very sophisticated person who enjoys luxurious accessories and gadgets. His salary is big enough to allow him to purchase such items and he is normally up to date with new technological devices. Recently, Harold was presented with the new version of the smartphone he currently owns.

This new device costs $1,100 and it was offered to a few VIP clients of the firm. Even though there are other amazing smartphones available in the market, Harold prefers this new version because he is loyal to the brand. The utility he gets from this phone comes in the form of possession, as owning this new device makes him feel important and appreciated.

4.Write about Value Iteration?

Value Iteration

Learning outcomes

The learning outcomes of this chapter are:

1. Apply value iteration to solve small-scale MDP problems manually and program value iteration algorithms to solve medium-scale MDP problems automatically
2. Construct a policy from a value function
3. Discuss the strengths and weaknesses of value iteration

Overview

Value Iteration is a method for finding the optimal value function $V*$ by solving the Bellman equations iteratively. It uses the concept of dynamic programming to maintain a value function V that approximates the optimal value function $V*$, iteratively improving V until it converges to $V*$ (or close to it).

Algorithm

Once we understand the Bellman equation, the value iteration algorithm is straightforward: we just repeatedly calculate V using the Bellman equation until we converge to the solution or we execute a pre-determined number of iterations.

Algorithm – Value Iteration

Input: MDP $M=\langle S,s0,A,Pa(s'|s),r(s,a,s')\rangle$
Output: Value function V

Set V to arbitrary value function; e.g., V(s)=0 for all s

Repeat
$\Delta \leftarrow 0$
For each $s \in S$
$V'(s) \leftarrow \max a \in A(s) \sum s' \in S Pa(s'|s)\ [r(s,a,s')+\gamma\ V(s')]$ ‿Bellman equation
$\Delta \leftarrow \max(\Delta, |V'(s)-V(s)|)$
$V \leftarrow V'$
Until $\Delta \leq \theta$
As we can see, this is just applying the Bellman equation iteratively until either the value function V doesn't change anymore, or until it changes in by a very small amount (θ).

We could also write the algorithm using the idea of Q-values, which is closer to a code-based implementation. For this, the loop is:

$\Delta \leftarrow 0$
For each $s \in S$
For each $a \in A(s)$
$Q(s,a) \leftarrow \sum_{s' \in S} P_a(s' | s) \, [r(s,a,s') + \gamma \, V(s')]$
$\Delta \leftarrow \max(\Delta, |\max_{a \in A(s)} Q(s,a) - V(s)|)$
$V(s) \leftarrow \max_{a \in A(s)} Q(s,a)$

Value iteration converges to the optimal policy as iterations continue: $V \mapsto V_*$ as $i \mapsto \infty$, where i is the number of iterations. So, given an infinite amount of iterations, it will be optimal.

Value iteration converges to the optimal value function V_* asymptotically, but in practice, the algorithm is stopped when the *residual* Δ reaches some pre-determined threshold θ – that is, when the largest change in the values between iterations is "small enough".

A policy can now be easily defined: in a state s, given V, choose the action with the highest expected reward using policy extraction. The loss of the result greedy policy is bound by $\frac{2\gamma\Delta}{1-\gamma}$.

Note that we do not need an optimal value function V to obtain an optimal policy. A value function that is "close enough" can still give an optimal policy because the small values do not change the resulting policy. Of course, we would not *know* whether a policy is optimal unless we know the value function is optimal.

Complexity

The complexity of each iteration is $O(|S|^2|A|)$. On each iteration, we iterate in an outer loop over all states in S, and in each outer loop iteration, we need to iterate over all states ($\sum_{s' \in S}$), meaning $|S|^2$ iterations. But also within each outer loop iteration, we need to calculate the value for every action to find the maximum.

It is clear to see that the value iteration can be easily parallelised by updating the value of many states at once: the values of states at step t+1 are dependent only on the value of other states at step t.

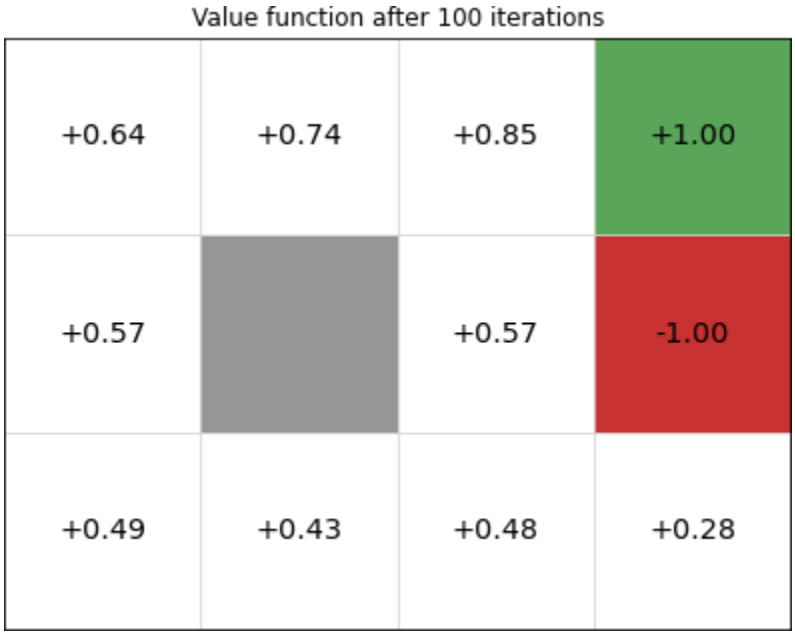
Implementation

Below is a Python implementation for value iteration. In this implementation, the parameter `iterations` is the number of iterations around the loop, which will terminate before convergence is the maximum number of iterations is reach. The parameter `theta` is θ in the value iteration algorithm above. Once the difference (Δ) is less than `theta` , the loop will terminate.

```
from gridworld import *
from tabular_value_function import *
from qtable import *


class ValueIteration:
    def __init__(self, mdp, values):
        self.mdp = mdp
        self.values = values

    def value_iteration(self, max_iterations=100, theta=0.001):

        for i in range(max_iterations):
            delta = 0.0
            new_values = TabularValueFunction()
            for state in self.mdp.get_states():
                qtable = QTable()
                for action in self.mdp.get_actions(state):
                    # Calculate the value of Q(s,a)
                    new_value = 0.0
                    for (new_state, probability) in
self.mdp.get_transitions(
                        state, action
                    ):
                        reward = self.mdp.get_reward(state, action,
new_state)
                        new_value += probability * (
                            reward
                            + (
                                self.mdp.get_discount_factor()
                                * self.values.get_value(new_state)
                            )
                        )

                    qtable.update(state, action, new_value)

                # V(s) = max_a Q(sa)
                (_, max_q) = qtable.get_max_q(state,
self.mdp.get_actions(state))
                delta = max(delta, abs(self.values.get_value(state) -
max_q))
                new_values.update(state, max_q)

            self.values.merge(new_values)

            # Terminate if the value function has converged
            if delta < theta:
                return i
```

Given this, we can create a GridWorld MDP, and solve using value iteration. The code below computes a value function using value iteration for 100 iterations:

```
from gridworld import GridWorld
from value_iteration import ValueIteration
from tabular_value_function import TabularValueFunction

gridworld = GridWorld()
values = TabularValueFunction()
ValueIteration(gridworld, values).value_iteration(max_iterations=100)
gridworld.visualise_value_function(values, "Value function after 100
iterations")
```



From the value function, we extract a policy:

```
policy = values.extract_policy(gridworld)
gridworld.visualise_policy(policy, "Policy after 100 iterations")
```

Using the visualisation belong, stepping through the 100 iterations, we can see that using value iteration, the values converge within about 10 iterations (to two decimal places), with each iteration giving us diminishing returns:

Partially observable Markov decision process

A **partially observable Markov decision process** (**POMDP**) is a generalization of a Markov decision process (MDP). A POMDP models an agent decision process in which it is assumed that the system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state. Instead, it must maintain a sensor model (the probability distribution of different observations given the underlying state) and the underlying MDP. Unlike the policy function in MDP which maps the underlying states to the actions, POMDP's policy is a mapping from the history of observations (or belief states) to the actions.

The POMDP framework is general enough to model a variety of real-world sequential decision processes. Applications include robot navigation problems, machine maintenance, and planning under uncertainty in general. The general framework of Markov decision processes with imperfect information was described by Karl Johan Åström in 1965[1] in the case of a discrete state space, and it was further studied in the operations research community where the acronym POMDP was coined. It was later adapted for problems in artificial intelligence and automated planning by Leslie P. Kaelbling and Michael L. Littman.[2]

An exact solution to a POMDP yields the optimal action for each possible belief over the world states. The optimal action maximizes the expected reward (or minimizes the cost) of the agent over a possibly infinite horizon. The sequence of optimal actions is known as the optimal policy of the agent for interacting with its environment.

Definition[edit]

Formal definition[edit]

A discrete-time POMDP models the relationship between an agent and its environment. Formally, a

POMDP is a 7-tuple , where

- is a set of states,
- is a set of actions,
- is a set of conditional transition probabilities between states,
- is the reward function.
- is a set of observations,
- is a set of conditional observation probabilities, and
- is the discount factor.

At each time period, the environment is in some state . The agent takes an action , which causes the environment to transition to state with probability . At the same time, the agent receives an observation which depends on the new state of the environment, , and on the just taken action, , with probability (or sometimes depending on the sensor model). Finally, the agent receives a reward equal to . Then the process repeats. The goal is for the agent to choose actions at each time step that maximize its expected future discounted

reward: , where is the reward earned at time . The discount factor determines how much immediate rewards are favored over more distant rewards. When the agent only cares about which action will yield the largest expected immediate reward; when the agent cares about maximizing the expected sum of future rewards.

Discussion[edit]

Because the agent does not directly observe the environment's state, the agent must make decisions under uncertainty of the true environment state. However, by interacting with the environment and receiving observations, the agent may update its belief in the true state by updating the probability distribution of the current state. A consequence of this property is that the optimal behavior may often include (information gathering) actions that are taken purely because they improve the agent's estimate of the current state, thereby allowing it to make better decisions in the future.

It is instructive to compare the above definition with the definition of a Markov decision process. An MDP does not include the observation set, because the agent always knows with certainty the environment's current state. Alternatively, an MDP can be reformulated as a POMDP by setting the observation set to be equal to the set of states and defining the observation conditional probabilities to deterministically select the observation that corresponds to the true state.

Belief update[edit]

After having taken the action and observing , an agent needs to update its belief in the state the environment may (or not) be in. Since the state is Markovian (by assumption), maintaining a belief over the states solely requires knowledge of the previous belief state, the action taken, and the current observation. The operation is denoted . Below we describe how this belief update is computed.

After reaching , the agent observes with probability . Let be a probability distribution over the state space . denotes the probability that the environment is in state . Given , then after taking action and observing ,

where is a normalizing constant with .

Belief MDP[edit]

A Markovian belief state allows a POMDP to be formulated as a Markov decision process where every belief is a state. The resulting *belief MDP* will thus be defined on a continuous state space (even if the "originating" POMDP has a finite number of states: there are infinite belief states (in) because there are an infinite number of probability distributions over the states (of)).[2]

Formally, the belief MDP is defined as a tuple where

- is the set of belief states over the POMDP states,
- is the same finite set of action as for the original POMDP,
- is the belief state transition function,
- is the reward function on belief states,
- is the discount factor equal to the in the original POMDP.

Of these, and need to be derived from the original POMDP. is

where is the value derived in the previous section and

The belief MDP reward function ( ) is the expected reward from the POMDP reward function over the belief state distribution:

.

The belief MDP is not partially observable anymore, since at any given time the agent knows its belief, and by extension the state of the belief MDP.

SHORT ANWER QUETUIONS

1.EXPLAIN MDP FORMULATION
2.EXPLAIN UTILITY TEORY
3.EXPLAIN UTILITYFUCTONS
4.EXPLAIN VALUE ITERATON
5.EXPLAIN POLICY ITERATION

LONG ANSWER QUETIOS

1EXPLAIN ABOUT THE POLICY ITERATION AND PARTIALLY OBERVATIONS
2.EXPLAIN ABOUT THE UTILITY THEROY AND UTILITY FUNCTIONS

3.EXPLAIN ABOUT THE PARTIALLY OBRSERVABLE MDPS

Unit-5

1.Explain Passive Reinforcement 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 $U\pi(s)$ for each state s. This can be done in three ways.

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 that* ***state utilities are independent*** *while in reality they are* ***Markovian***. 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

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$$

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

2. 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.

$$U^{\pi}(s) = U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

Where $R(s)$ = reward for being in state s, $P(s'|s, \pi(s))$ = transition model, γ = discount factor and $U\pi(s)$ = 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.

Fig 2: AI playing Super Mario using Deep RL

3. 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.

$$U^{\pi}(s) = U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

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.***

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} f\left(\sum_{s'} P(s'|s,a) U_i(s'), N(s,a)\right)$$

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

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

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.

2. Q-Learning

Q-learning is a TD learning method which does not require the agent to learn the transitional model, instead learns Q-value functions *Q(s, a)* .

$$U(s) = max_a\, Q(s,a)$$

Q-values can be updated using the following equation,

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Next action can be selected using the following policy,

$$a_{next} = arg\ max_{a'}\, f(Q(s',a'),\ N(s',a'))$$

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

	Fixed Policy(Active)	**Policy not fixed(Passive)**
Model-free (real world)	Temporal Difference Learning (TD)	Q-learning
Model-based (simulation)	Adaptive Dynamic Programming(ADP)	ADP with proper exploration function

Table 1: Comparison of active and passive RL methods

2.Explain Adaptive Dynamic Programming?

Artificial intelligence can be various things: doing intelligent things with computers, or doing smart things with computers the manner in which individuals do them. The distinction is significant. Computers work uniquely in contrast to our brains: our minds are serial consciously, however, parallel underneath. Computers are serial underneath, however, we can have different processors, and there are now parallel hardware architectures too. All things considered, it's difficult to do parallel in parallel, though we're normally that way. Copying human methodologies has been a long-standing exertion in AI, as a mechanism to affirm our comprehension. If we can get similar outcomes from a computer simulation, we can propose that we have a strong model of what's going on. Obviously, the connections work, inspired by frustration with some artifacts of cognition, shows that some of the previous emblematic models were approximations rather than exact portrayals. Presently, issues in information security, communication bandwidth, and processing latency are driving AI from the cloud to the edge. Nonetheless, a similar AI innovation that acquired significant headway in cloud computing, fundamentally through the availability of GPUs for training and running large neural networks, are not appropriate for edge AI. Edge AI gadgets work with tight resource budgets, for example, memory, power and computing horsepower. Training complex deep neural networks (DNN) is already a complex process, and preparing for edge targets can be limitlessly more troublesome. Conventional methodologies in training AI for the edge are restricted in light of the fact that they depend on the idea that the processing for the inference is statically characterized during training. These static methodologies incorporate post-training quantization and pruning, and they don't consider how deep networks may need to work diversely at runtime. Compared with the static methodologies above, Adaptive AI is an essential move in the manner AI is trained and how current and future computing needs are resolved. The reason behind why it could outpace traditional machine learning (ML) models soon is for its capability to encourage organizations in accomplishing better results while contributing less time, effort and assets.

Robust, Efficient and Agile

The three primary precepts of Adaptive AI are robustness, efficiency, and agility. Robustness is the capacity to accomplish high algorithmic precision. Efficiency is the capability to accomplish low resource usage (for example computer, memory, and power). Agility manages the capacity to adjust operational conditions dependent on current needs. Together, these three precepts of Adaptive AI plan the key measurements toward super proficient AI inference for edge devices.

Data-informed Predictions

The Adaptive Learning technique utilizes a single pipeline. With this strategy, you can utilize a constantly advanced learning approach that keeps the framework updated and encourages it to accomplish high-performance levels. The Adaptive Learning process screens and learns the new changes made to the info and yield values and their related qualities. Furthermore, it gains from the occasions that may change the market behavior in real-time and, henceforth, keeps up its

precision consistently. Adaptive AI acknowledges the input got from the operating environment and follows up on it to make data-informed predictions.

Sustainable System

Adaptive Learning tackles the issues while building ML models at scale. Since the model is prepared through a streaming methodology, its proficiency for spaces with profoundly meager datasets where noise handling is significant. The pipeline is intended to deal with billions of features across tremendous datasets while each record can have many features, leading to sparse data records. This system takes a shot at a single pipeline instead of the conventional ML pipelines that are isolated into two sections. This gives quick solutions for verification of ideas and simple deployment in production. The underlying exhibition of the Adaptive Learning framework is comparable to batch-model systems yet proceeds to outperform them by acting and gaining from the feedback obtained by the system, making it unquestionably more robust and sustainable in the long term.

Future Prospects

Adaptive AI will be widely used to deal with changing AI computing needs. Operational effectiveness is resolved during runtime with regards to what algorithmic performance is required and what computing resources are available. Edge AI frameworks that can powerfully alter their computing needs are the best way to bring down compute and memory resources needs. The qualities of Adaptive AI make it profoundly solid in the dynamic software environments of CSPs where inputs/outputs change with each framework overhaul. It can play a key part in their digital transformation across network operations, marketing, customer care, IoT, security, and help evolve customer experience.

3.Explain About Reinforcement Learning?

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Example: The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.

The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Difference between Reinforcement learning and Supervised learning:

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game	Example: Object recognition

Types of Reinforcement: There are two types of Reinforcement:

1. **Positive –**
   Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.
   Advantages of reinforcement learning are:
   - Maximizes Performance
   - Sustain Change for a long period of time
   - Too much Reinforcement can lead to an overload of states which can diminish the results
2. **Negative –**
   Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.
   Advantages of reinforcement learning:
   - Increases Behavior
   - Provide defiance to a minimum standard of performance
   - It Only provides enough to meet up the minimum behavior

Various Practical applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

RL can be used in large environments in the following situations:

1. A model of the environment is known, but an analytic solution is not available;
2. Only a simulation model of the environment is given (the subject of simulation-based optimization)
3. The only way to collect information about the environment is to interact with it.

4. Give a Brief Information about Machine Learning?

Do you get automatic recommendations on Netflix and Amazon Prime about the movies you should watch next? Or maybe you get options for People You may

know on Facebook or LinkedIn? You might also use Siri, Alexa, etc. on your phones. That's all **Machine Learning**! This is a technology that is becoming more and more popular. Chances are that Machine Learning is used in almost every technology around you!

And it is hardly a new concept. Researchers have always been fascinated by the capacity of machines to learn on their own without being programmed in detail by humans. However, this has become much easier to do with the emergence of big data in modern times. Large amounts of data can be used to create much more accurate Machine Learning algorithms that are actually viable in the technical industry. And so, Machine Learning is now a buzz word in the industry despite having existed for a long time.

But are you wondering what is Machine Learning after all? What are its various types and what are the different Machine Learning algorithms? Read on to find the answers to all your questions!

What is Machine Learning?

Machine Learning, as the name says, is all about machines learning automatically without being explicitly programmed or learning without any direct human intervention. This machine learning process starts with feeding them good quality data and then training the machines by building various machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data we have and what kind of task we are trying to automate.

As for the formal definition of Machine Learning, we can say that a Machine Learning algorithm learns from **experience E** with respect to some type of **task T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.
For example, If a Machine Learning algorithm is used to play chess. Then the experience E is playing many games of chess, the task T is playing chess with many players, and the performance measure P is the probability that the algorithm will win in the game of chess.

What is the difference between Artificial Intelligence and Machine Learning?

Artificial Intelligence and Machine Learning are correlated with each other, and yet they have some differences. Artificial Intelligence is an overarching concept that aims to create intelligence that mimics human-level intelligence. Artificial Intelligence is a general concept that deals with creating human-like critical thinking capability and reasoning skills for machines. On the other hand, Machine Learning is a subset or specific application of Artificial intelligence that aims to create machines that can learn autonomously from data. Machine Learning is specific, not general, which means it allows a machine to make predictions or take some decisions on a specific problem using data.

What are the types of Machine Learning?

Let's see the different types of Machine Learning now:

1. Supervised Machine Learning

Imagine a teacher supervising a class. The teacher already knows the correct answers but the learning process doesn't stop until the students learn the answers as well. This is the essence of Supervised Machine Learning Algorithms. Here, the algorithm learns from a training dataset and makes predictions that are compared with the actual output values. If the predictions are not correct, then the algorithm is modified until it is satisfactory. This learning process continues until the algorithm achieves the required level of performance. Then it can provide the desired output values for any new inputs.

2. Unsupervised Machine Learning

In this case, there is no teacher for the class and the students are left to learn for themselves! So for Unsupervised Machine Learning Algorithms, there is no specific answer to be learned and there is no teacher. In this way, the algorithm doesn't figure out any output for input but it explores the data. The algorithm is left unsupervised to find the underlying structure in the data in order to learn more and more about the data itself.

3. Semi-Supervised Machine Learning

The students learn both from their teacher and by themselves in Semi-Supervised Machine Learning. And you can guess that from the name itself! This is a combination of Supervised and Unsupervised Machine Learning that uses a little amount of labeled data like Supervised Machine Learning and a larger amount of unlabeled data like Unsupervised Machine Learning to train the algorithms. First, the labeled data is used to partially train the Machine Learning Algorithm, and then this partially trained model is used to pseudo-label the rest of the unlabeled data. Finally, the Machine Learning Algorithm is fully trained using a combination of labeled and pseudo-labeled data.

4. Reinforcement Machine Learning
Well, here are the hypothetical students who learn from their own mistakes over time (that's like life!). So the Reinforcement Machine Learning Algorithms learn optimal actions through trial and error. This means that the algorithm decides the next action by learning behaviors that are based on its current state and that will maximize the reward in the future. This is done using reward feedback that allows the Reinforcement Algorithm to learn which are the best behaviors that lead to maximum reward. This reward feedback is known as a reinforcement signal.

What are some popular Machine Learning algorithms?

Let's look at some of the popular Machine Learning algorithms that are based on specific types of Machine Learning.

Supervised Machine Learning

Supervised Machine Learning includes Regression and Classification algorithms. Some of the more popular algorithms in these categories are:

1. Linear Regression Algorithm
The Linear Regression Algorithm provides the relation between an independent and a dependent variable. It demonstrates the impact on the dependent variable when the independent variable is changed in any way. So the independent variable is called the explanatory variable and the dependent variable is called the factor of interest. An example of the Linear Regression Algorithm usage is to analyze the property prices in the area according to the size of the property, number of rooms, etc.

2. Logistic Regression Algorithm
The Logistic Regression Algorithm deals in discrete values whereas the Linear Regression Algorithm handles predictions in continuous values. This means that Logistic Regression is a better option for binary classification. An event in Logistic Regression is classified as 1 if it occurs and it is classified as 0 otherwise. Hence, the probability of a particular event occurrence is predicted based on the given predictor variables. An example of the Logistic Regression Algorithm usage is in medicine to predict if a person has malignant breast cancer tumors or not based on the size of the tumors.

3. Naive Bayes Classifier Algorithm
Naive Bayes Classifier Algorithm is used to classify data texts such as a web page, a document, an email, among other things. This algorithm is based on the Bayes Theorem of Probability and it allocates the element value to a population from one of the categories that are available. An example of the Naive Bayes Classifier Algorithm usage is for Email Spam Filtering. Gmail uses this algorithm to classify an email as Spam or Not Spam.

Unsupervised Machine Learning

Unsupervised Machine Learning mainly includes Clustering algorithms. Some of the more popular algorithms in this category are:

1. K Means Clustering Algorithm

Let's imagine that you want to search the name "Harry" on Wikipedia. Now, "Harry" can refer to Harry Potter, Prince Harry of England, or any other popular Harry on Wikipedia! So Wikipedia groups the web pages that talk about the same ideas using the K Means Clustering Algorithm (since it is a popular algorithm for cluster analysis). K Means Clustering Algorithm in general uses K number of clusters to operate on a given data set. In this manner, the output contains K clusters with the input data partitioned among the clusters.

2. Apriori Algorithm

The Apriori Algorithm uses the if-then format to create association rules. This means that if a certain event 1 occurs, then there is a high probability that a certain event 2 also occurs. For example: IF someone buys a car, THEN there is a high chance they buy car insurance as well. The Apriori Algorithm generates this association rule by observing the number of people who bought car insurance after buying a car. For example, Google auto-complete uses the Apriori Algorithm. When a word is typed in Google, the Apriori Algorithm looks for the associated words that are usually typed after that word and displays the possibilities.

What is Deep Learning?

Deep Learning is a subset of Machine Learning. It is based on learning by example, just like humans do, using Artificial Neural Networks. These Artificial Neural Networks are created to mimic the neurons in the human brain so that Deep Learning algorithms can learn much more efficiently. Deep Learning is so popular now because of its wide range of applications in modern technology. From self-driving cars to image, speech recognition, and natural language processing, Deep Learning is used to achieve results that were not possible before.

What are Artificial Neural Networks?

Artificial Neural Networks are modeled after the neurons in the human brain. They contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Networks in a system. A layer can have only a dozen units or millions of units as this depends on the complexity of the system. Commonly, Artificial Neural Networks have an input layer, output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze

or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.
In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to another, the neural network learns more and more about the data which eventually results in an output from the output layer.

What is Machine Learning used for?

Machine Learning is used in almost all modern technologies and this is only going to increase in the future. In fact, there are applications of Machine Learning in various fields ranging from smartphone technology to healthcare to social media, and so on.

Smartphones use **personal voice assistants** like Siri, Alexa, Cortana, etc. These personal assistants are an example of ML-based speech recognition that uses Natural Language Processing to interact with the users and formulate a response accordingly. Machine Learning is also used in social media. Let's take Facebook's **'People you may know'** as an example. It is mind-boggling how social media platforms can guess the people you might be familiar with in real life. And they are right most of the time!!! This is done by using Machine Learning algorithms that analyze your profile, your interests, your current friends, and also their friends and various other factors to calculate the people you might potentially know.
Machine Learning is also very important in **healthcare diagnosis** as it can be used to diagnose a variety of problems in the medical field. For example, Machine Learning is used in oncology to train algorithms that can identify cancerous tissue at the microscopic level at the same accuracy as trained physicians.
Another famous application of Machine Learning is **Google Maps**. The Google Maps algorithm automatically picks the best route from one point to another by relying on the projections of different timeframes and keeping in mind various factors like traffic jams, roadblocks, etc. In this way, you can see that the applications of Machine Learning are limitless. If anything, they are only increasing and Machine Learning may one day be used in almost all fields of study!

Are Machine Learning Algorithms totally objective?

Machine Learning Algorithms are trained using data sets. And unfortunately, sometimes the data may be **biased** and so the ML algorithms are not totally objective. This is because the data may include human biases, historical inequalities, or different metrics of judgement based on gender, race, nationality,

sexual orientation, etc. For example, Amazon found out that their Machine Learning based recruiting algorithm was biased against women. This may have occurred as the recruiting algorithm was trained to analyze the candidates' resumes by studying Amazon's response to the resumes that were submitted in the past 10 years. However, the human recruiters who analyzed these resumes in the past were mostly men with an inherent bias against women candidates that were passed on to the AI algorithm.
This means that some Machine Learning Algorithms used in the real world may not be objective due to biased data. However, companies are working on making sure that only objective algorithms are used. One way to do this is to preprocess the data so that the bias is eliminated before the ML algorithm is trained on the data. Another way is to post-process the ML algorithm after it is trained on the data so that it satisfies an arbitrary fairness constant that can be decided beforehand.

Which Cloud Computing Platforms offer Machine Learning?

There are many Cloud Computing Platforms that offer Machine Learning services to other companies. The most popular among them are:

1. Amazon Web Services (AWS)

Some of the products that Amazon Web Services provides include Amazon SageMaker for creating and training machine learning models, Amazon Forecast to increase the forecast accuracy, Amazon Translate for language translation using natural language processing, Amazon Polly for converting text into life-like speech, etc.

2. Microsoft Azure

Some of the products that Microsoft Azure provides include Microsoft Azure Machine Learning for creating and deploying machine learning models, Cognitive Service for providing smart cognitive services, Databricks for Apache Spark-based analytics, Bot Service for smart and intelligent bot services, etc.

3. Google Cloud

Some of the products that Google Cloud provides include Google Cloud AutoML for training an AutoML machine learning model, Vision AI for cloud vision, Speech-to-Text for transmitting from speech to text, Text-to-Speech for transmitting from text to speech, Natural Language for natural language processing, etc.

4. IBM Watson Cloud

Some of the products that IBM Watson Cloud provides include IBM Watson Studio for building machine learning and artificial intelligence models, Speech-to-Text for transmitting from speech to text, Text-to-Speech for transmitting from text to speech, Assistant for creating and managing virtual assistants, Natural Language Understanding for natural language processing, etc.

What are the courses to learn Machine Learning?

There are many online courses on platforms such as GeeksforGeeks, etc. that can help you to learn Machine Learning effectively. These courses can easily teach the basics of Machine Learning and will also let you know the practical implementation of these concepts through various projects. Some of the popular courses are – **Machine Learning Foundation With Python – Live** and **Machine Learning – Basic Level Course**.

From translation apps to autonomous vehicles, all powers with Machine Learning. It offers a way to solve problems and answer complex questions. It is basically a process of training a piece of software called an algorithm or model, to make useful predictions from data. This article discusses the categories of machine learning problems, and terminologies used in the field of machine learning.

Types of machine learning problems

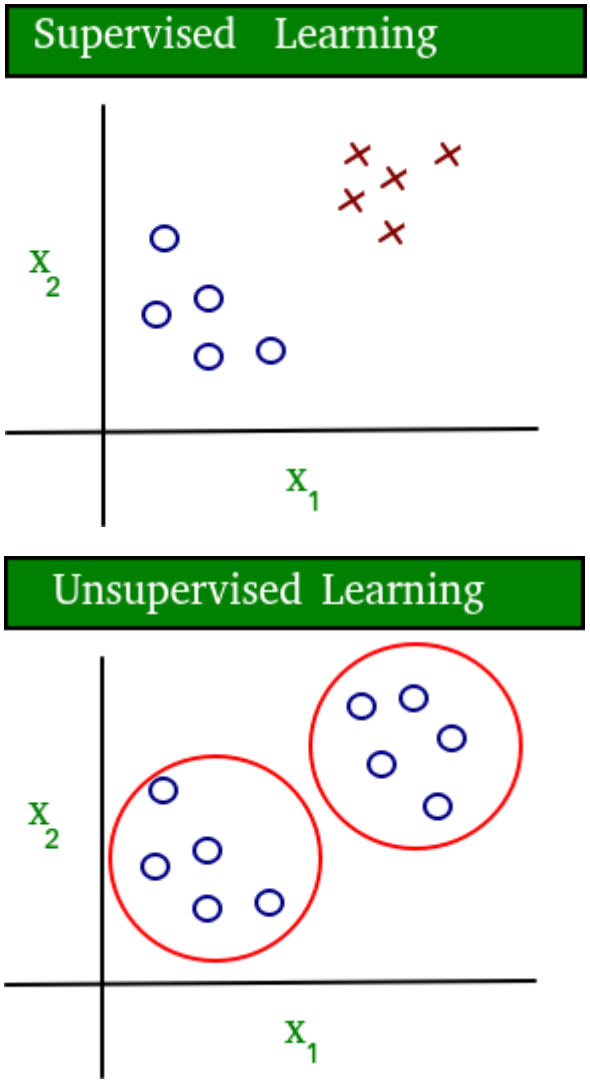
There are various ways to classify machine learning problems. Here, we discuss the most obvious ones.

1. On basis of the nature of the learning "signal" or "feedback" available to a learning system

- **Supervised learning**: The model or algorithm is presented with example inputs and their desired outputs and then finding patterns and connections between the input and the output. The goal is to learn a general rule that maps inputs to outputs. The training process continues until the model achieves the desired level of accuracy on the training data. Some real-life examples are:
    - **Image Classification:** You train with images/labels. Then in the future you give a new image expecting that the computer will recognize the new object.
    - **Market Prediction/Regression:** You train the computer with historical market data and ask the computer to predict the new price in the future.
- **Unsupervised learning**: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. It is used for clustering population in different groups. Unsupervised learning can be a goal in itself (discovering hidden patterns in data).
    - **Clustering:** You ask the computer to separate similar data into clusters, this is essential in research and science.

- **High Dimension Visualization:** Use the computer to help us visualize high dimension data.
- **Generative Models:** After a model captures the probability distribution of your input data, it will be able to generate more data. This can be very useful to make your classifier more robust.

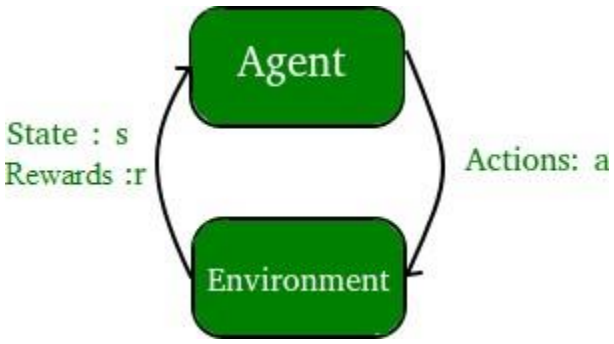
A simple diagram which clears the concept of supervised and unsupervised learning is shown below:



As you can see clearly, the data in supervised learning is labelled, where as data in unsupervised learning is unlabelled.

- **Semi-supervised learning**: Problems where you have a large amount of input data and only some of the data is labeled, are called semi-supervised learning problems. These problems sit in between both supervised and unsupervised learning. For example, a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

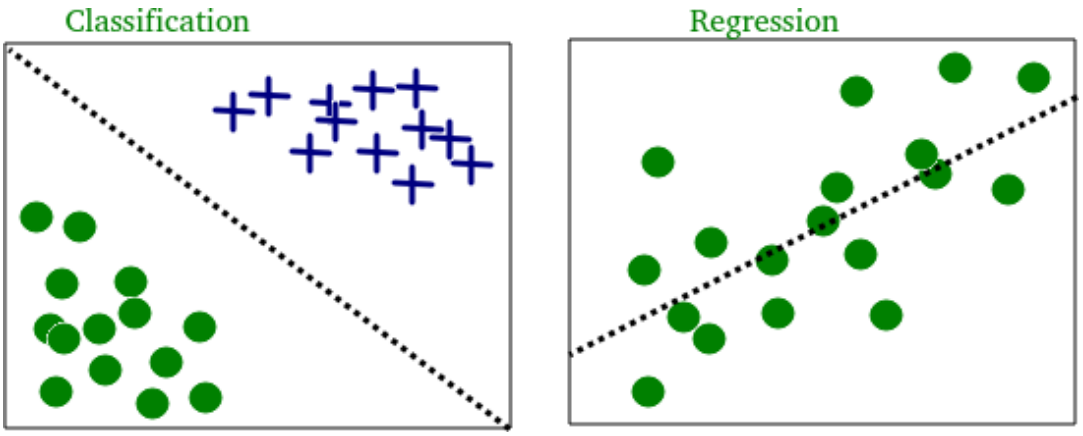
- **Reinforcement learning**: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.



2. Two most common use cases of Supervised learning are:

- **Classification**: Inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one or more (multi-label classification) of these classes and predicting whether or not something belongs to a particular class. This is typically tackled in a supervised way. Classification models can be categorized in two groups: Binary classification and Multiclass Classification. Spam filtering is an example of binary classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".
- **Regression**: It is also a supervised learning problem, that predicts a numeric value and outputs are continuous rather than discrete. For example, predicting the stock prices using historical data.

An example of classification and regression on two different datasets is shown below:



3. Most common Unsupervised learning are:

- **Clustering**: Here, a set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task. As you can see in the example below, the given dataset points have been divided into groups identifiable by the colors red, green and blue.
- **Density estimation**: The task is to find the distribution of inputs in some space.
- **Dimensionality reduction**: It simplifies inputs by mapping them into a lower-dimensional space. Topic modeling is a related problem, where a program is given a list of human language documents and is tasked to find out which documents cover similar topics.

On the basis of these machine learning tasks/problems, we have a number of algorithms which are used to accomplish these tasks. Some commonly used machine learning algorithms are Linear Regression, Logistic Regression, Decision Tree, SVM(Support vector machines), Naive Bayes, KNN(K nearest neighbors), K-Means, Random Forest, etc. **Note:** All these algorithms will be covered in upcoming articles.

Terminologies of Machine Learning

- **Model** A model is a **specific representation** learned from data by applying some machine learning algorithm. A model is also called **hypothesis**.
- **Feature** A feature is an individual measurable property of our data. A set of numeric features can be conveniently described by a **feature vector**. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, **etc. Note:** Choosing informative, discriminating and independent features is a crucial step for effective algorithms. We generally employ a **feature extractor** to extract the relevant features from the raw data.
- **Target (Label)** A target variable or label is the value to be predicted by our model. For the fruit example discussed in the features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.
- **Training** The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.
- **Prediction** Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label). But make sure if the machine performs well on unseen data, then only we can say the machine performs well.

5.Write about temporal difference Learning?

Temporal difference (TD) learning is an approach to learning how to predict a quantity that depends on future values of a given signal. The name TD derives from its use of changes, or differences, in predictions over successive time steps to drive the learning process. The prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next time step. It is a supervised learning process in which the training signal for a prediction is a future prediction. TD algorithms are often used in reinforcement learning to predict a measure of the total amount of reward expected over the future, but they can be used to predict other quantities as well. Continuous-time TD algorithms havealso been developed.

The Problem

Suppose a system receives as input a time sequence of vectors (x_t,y_t) , $t=0,1,2,\ldots$, where each x_t is an arbitrary signal and y_t is a real number. TD learning applies to the problem of producing at each discrete time step t , an estimate, or prediction, p_t , of the following quantity:

$$Y_t=y_{t+1}+\gamma y_{t+2}+\gamma 2y_{t+3}+\cdots=\sum_{i=1}^{\infty}\gamma_{i-1}y_{t+i},$$

where γ is a *discount factor*, with $0\leq\gamma<1$. Each estimate is a prediction because it involves future values of y . The signal x_t is the information available to the system at time t to enable it to make the prediction p_t . In other words, p_t is a function of x_t , and we can write $p_t=P(x_t)$, where P is a *prediction function*. The discount factor determines how strongly future values of y influence current predictions. When $\gamma=0$, p_t predicts just the next value of y , that is, y_{t+1} . As γ increases toward one, values of y in the more distant future become more significant. The problem, then, is to select a function P so that $p_t=P(x_t)\approx Y_t$ as closely as possible for $t=0,1,2,\ldots$. This is the *infinite-horizon discounted* prediction problem. TD algorithms apply to other prediction problems as described below.

Of course, it is not always possible to find a prediction function that does a good job of solving this problem. If there are no regularities in the relationship between signals x_t and future values of y , then the best predictions will be no better than random guessing. However, if there are regularities in this relationship, then predictions may be possible that are more accurate than chance. The usual way to model possible regularities is to assume that the signals x_t are derived from the states of a Markov chain, on which the numbers y also functionally depend. Depending on how the x_t represent the Markov chain's states (e.g., do these signals unambiguously identify states, or are some states *unobservable*), a prediction function may exist that accurately gives the expected value of the quantity Y_t for each t .

The Simplest TD Algorithm

A good way to explain TD learning is to start with a simple case and then extend it to the full algorithm. Consider first the problem of making a one-step-ahead prediction, i.e., the above problem with $\gamma=0$, meaning that one wants $p_t=y_{t+1}$ for each t . Incremental error-correction supervised learning can be used to update the prediction function as inputs arrive.

Letting P_t denote the prediction function at step t, the algorithm updates P_t to a new prediction function P_{t+1} at each step. To do this, it uses the error between the current prediction, p_t, and the prediction target (the quantity being predicted), y_{t+1}. This error can be obtained by computing p_t by applying the current prediction function, P_t, to x_t, *waiting one time step* while remembering p_t, then observing y_{t+1} to obtain the information required to compute the error $y_{t+1}-p_t=y_{t+1}-P_t(x_t)$.

The simplest example of a prediction function is one implemented as a lookup table. Suppose x_t can take only a finite number of values and that there is an entry in a lookup table to store a prediction for each of these values. At step t, the table entry for input x_t is $p_t=P_t(x_t)$. When y_{t+1} is observed, the table entry for x_t is changed from its current value of p_t to $p_t+\alpha(y_{t+1}-p_t)=(1-\alpha)p_t+\alpha y_{t+1}$, where α is a positive fraction that controls how quickly new data is incorporated into the table and old data forgotten. Only the table entry corresponding to x_t is changed from step t to step $t+1$, to produce the a table, P_{t+1}. The fraction α is called the *learning rate* or *step-size* parameter. Note that if $\alpha=1$, the table entry is simply set to y_{t+1}, which is appropriate if the predictive relationship is deterministic. Using $\alpha<1$, on the other hand, causes the table entries to approach the *expected* prediction targets when the predictive relationship is not deterministic.

To extend this approach to the full prediction problem with $\gamma\neq 0$, the prediction target would be $Y_t=y_{t+1}+\gamma y_{t+2}+\gamma 2y_{t+3}+\cdots$ instead of just y_{t+1}. The algorithm above would have to update the table entry for x_t by changing its value from p_t to $p_t+\alpha(Y_t-p_t)=(1-\alpha)p_t+\alpha Y_t$. But to do this would require *waiting for the arrival of all the future values of y* instead of waiting for just the next value. This would prevent the approach from forming a useful learning rule. TD algorithms use the following observation to get around this problem. Notice that

$$Y_t=y_{t+1}+\gamma y_{t+2}+\gamma 2y_{t+3}+\cdots :$$

$$=y_{t+1}+\gamma[y_{t+2}+\gamma y_{t+3}+\gamma 2y_{t+4}+\cdots] :$$

$$=y_{t+1}+\gamma Y_{t+1},(1)$$

for $t=0,1,2,\ldots$. Therefore, since $P_t(x_{t+1})$ is the estimate of Y_{t+1} available at step t (i.e., using the step-t table), one can estimate Y_t by the quantity $y_{t+1}+\gamma P_t(x_{t+1})$. That is, the current prediction function, P_t, *applied to the next input*, x_{t+1}, multiplied by γ, gives an estimate of the part of the prediction target that would otherwise require waiting forever (in this case) to obtain. The result in this lookup-table case is an algorithm that, upon receiving input (x_{t+1},y_{t+1}), updates the table entry for x_t by changing its value from p_t to $p_t+\alpha[y_{t+1}+\gamma P_t(x_{t+1})-p_t]=(1-\alpha)p_t+\alpha[y_{t+1}+\gamma P_t(x_{t+1})]$.

More concisely, define the following *temporal difference error* (or TD error):

$$\delta_{t+1}=y_{t+1}+\gamma P_t(x_{t+1})-P_t(x_t).$$

Then the simplest TD algorithm updates a lookup-table as follows: for each $t=0,1,2,\ldots$, upon observing (x_{t+1},y_{t+1}) :

$$P_{t+1}(x)=\{P_t(x)+\alpha\delta_{t+1}P_t(x)\text{if}x=x_t\text{otherwise},(2)$$

where x denotes any possible input signal. [Note that some authors would label the TD error as defined here δ_t instead of δ_{t+1} , giving the impression that TD learning is somehow not *causal* in that its updates depend on unavailable future values of certain variables. If it is understood that the update defined here occurs on step $t+1$, it is completely causal. The time subscript for δ is unimportant.]

Although the detailed behavior of this algorithm is complicated, an intuitive understanding is possible. The algorithm uses a prediction of a later quantity, $P_t(x_{t+1})$, to update a prediction of an earlier quantity, $P_t(x_t)$, where each prediction is computed via the same prediction function, P_t . This may seem like a futile process since neither prediction need be accurate, but in the course of the algorithm's operation, later predictions tend to become accurate sooner than earlier ones, so there tends to be an overall error reduction as learning proceeds. This depends on the learning system receiving an input sequence with sufficient regularity to make predicting possible. In formal treatments, the inputs x_t represent states of a Markov chain and the y values are given by a function of these states. This makes it possible to form accurate predictions of the expected values of the discounted sums Y_t .

Another view of the TD algorithm is that it operates to maintain a consistency condition that must be satisfied by correct predictions. Since $Y_t=y_{t+1}+\gamma Y_{t+1}$, for $t=0,1,\ldots$ (Equation (1), the following relationship should hold between successive predictions:

$$P(x_t)=y_{t+1}+\gamma P(x_{t+1})$$

for $t=0,1,2,\ldots$. One can show, in fact, that within an appropriate mathematical framework any function that satisfies this condition for all t must actually give the correct predictions. TD algorithms adjust the prediction function with the goal of making its values always satisfy this condition. The TD error indicates how far the current prediction function deviates from this condition for the current input, and the algorithm acts to reduce this error. This view of TD learning is connected to the theory of Markov decision processes, where the prediction function corresponds to a value function and the update process is closely related to dynamic programming methods. Most of the theoretical results about TD learning derive from these connections.

SHORT ANSWER QUETIONS

1.WHATIS POSSIVE REINFORCEMENT LEARNING DIREC UTILITY ESTIMATION

2.EXPLAIN ABOUT THE ADAPTIVE DYNAMIC PROGRAMMING

3.WHAT IS THE TEMPORAL DIFFERENCE LEARNING

4.WHAT IS REAINFORECEMENT LEARNING
5. WHAT IS MACHINE LEARNIG

LONG ANSWER QUESTIONS

1.EXPLAIN ABOUT THE DEEP LEARNING AND MACHINE LEARNING
2.EXPLAIN ABOUT THE ACTIVE REINFORCEMENT LEARNIG-Q LEARNING
3.EXPLAIN ABOUT PASSIVE REAINFORCEMENT LERNING
4.EXPLAIN ABOUT DIRECT UTILITY ESTIMATUION

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF COMPUTER APPLICATIONS

ARTIFICIAL INTELLIGENCE

MARKS=30 INTERNAL EXAMINATION-I (TIME=2HR)

SECTION-A

Answer any five of the following(5*2=10m)

1. Define AI?
2. Explain Scope of AI?

3. Write about review of tree and graph structures?
4. Explain about Search Graph?
5. Explain about Random Search?
6. Explain about Depth First Search?
7. Explain about A* Algorithm?
8. Explain Game Search Algorithm?

SECTION-B

Answer one question from each unit(2*10=20m)

UNIT-1

9. Write about History of AI?

(or)

10. Explain about Bayes Rule?

Unit-2

11. Explain about temporal model?

(or)

12. Explain about Hidden marker Model?

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF COMPUTER APPLICATIONS

ARTIFICIAL INTELLIGENCE

MARKS=30 INTERNAL EXAMINATION-I I (TIME=2HR)

SECTION-A

Answer any five of the following(5*2=10m)

1. Define MDP Formulation?
2. Explain about Utility Functions?

3.What is Policy iteration?

4.Write about value iteration?

5.Write about reinforcement learning?

6.Difference between Active and Passive reinforcement learning?

7.Write any two Advantages of Utility Functions?

8.write the types of difference learning?

SECTION-B

Answer one question from each unit(2*10=20m)

UNIT-1

9.What are the Different Markov models?

(or)

10.write about the Iteration Models?

Unit-2

11.What are the types of Reinforcement Learning?

(or)

12.Explain about the adaptive dynamic programming?

MASTER OF COMPUTER APPLICATIONS DEGREE EXAMINATIONS,

AUGUST-2022

THIRD SEMESTER

PAPER-MCA:305C-MOBILE APPLICATION DEVELOPMENT

(Under c.b.c.s.revised regulations w.e.f. 2021-2022)

(common paper to university and all affiliated colleges)

Time:3hrs max.marks:70

Part-A

(COMPULSORY)

Answer any five of the following questions. Each questions carries two marks(5*2=10)

1.a) Define AI?

b) What is AI techniques?

c)What are the Characteristics of AI?

d)Define Resolution Principle/

e) Difference Between Forward and Backward reasoning symbolic?

f) Define Waltzs Algorithm/

g) Explain about Transitional Network?

h) What are the characteristics of expert systems?

i) Define PROLOG?

j)Define about Expert System Shells?

PART-B

Answer any ONE question from each unit (5*12=60m)

Unit-1

2. Explain about in detailed History of AI?

(or)

3. Write a short note on Search Graph?

Unit-2

4. Explain about DFSL? In Detail.

(Or)

5. Explain BFS? and what are the different types of algorithms?

Unit-3

6. Explain Hidden Markov Model and its Advantages?

(or)

7. Explain about bayes rule? And its history And the importance of bayes rule?

Unit-4

8. Define MDP and Explain about formulation of MDP?

(or)

9. Explain Utility Functions? How does it helpful in AI?

Unit-5

10. Write about temporal Difference Learning?

(or)

11. Explain about

a) Passive Reinforcement Learning?

b) Active Reinforcement Learning?