NIGERIAN ARMY UNIVERSITY, BIU

FACULTY OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE

BORNO STATE, NIGERIA

CSC 422

ARTIFICIAL INTELLIGENCE

LECTURE NOTE

FIRST SEMESTER

3 UNIT COURSE

Dr. Habila Mikailu

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

What is Artificial Intelligence?

It is a branch of Computer Science that pursues creating the computers or machines as intelligent as human beings. It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable Definition: Artificial Intelligence is the study of how to make computers do things, which, at the moment, people do better. According to the father of Artificial Intelligence, John McCarthy, it is "The science and engineering of making intelligent machines, especially intelligent computer programs". Artificial Intelligence is a way of making a computer, a computer-controlled robot, or a software think intelligently, in the similar manner the intelligent humans think. AI is accomplished by studying how human brain thinks and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems. It has gained prominence recently due, in part, to big data, or the increase in speed, size and variety of data businesses are now collecting. AI can perform tasks such as identifying patterns in the data more efficiently than humans, enabling businesses to gain more insight out of their data. From a business perspective AI is a set of very powerful tools, and methodologies for using those tools to solve business problems. From a programming perspective, AI includes the study of symbolic programming, problem solving, and search.

1. UNDERSTANDING NATURAL LANGUAGES

Natural Language Processing (NLP) is one of the hottest areas of artificial intelligence (AI), thanks to applications like text generators that compose coherent essays, chatbots that fool people into thinking they're sentient, and text-to-image programs that produce photorealistic images of anything you can describe. Recent years have brought a revolution in the ability of computers to understand human languages, programming languages, and even biological and chemical sequences, such as DNA and protein structures, that resemble language. The latest AI models are unlocking these areas to analyze the meanings of input text and generate meaningful, expressive output.

What is NLP: Natural language processing (NLP) is the discipline of building machines that can manipulate human language or data that resembles human language in the way that it is written, spoken, and organized. It evolved from computational linguistics, which uses computer science to understand the principles of language, but rather than developing theoretical frameworks, NLP is an engineering discipline that seeks to build technology to accomplish useful tasks. NLP can be divided into two overlapping subfields: natural language understanding (NLU), which focuses on semantic analysis or determining the intended meaning of text, and natural language generation (NLG), which focuses on text generation by a machine. NLP is separate from but often used in conjunction with speech recognition, which seeks to parse spoken language into words, turning sound into text and vice versa.

Why Does NLP Matter? NLP is an integral part of everyday life and becoming more so as language technology is applied to diverse fields like retailing (for instance, in customer service chatbots) and medicine (interpreting or summarizing electronic health records). Conversational agents such as Amazon's Alexa and Apple's Siri utilize NLP to listen to user queries and find answers. The most sophisticated such agents such as GPT-3, which was recently opened for commercial applications can generate sophisticated prose on a wide variety of topics as well as power chatbots that are capable of holding coherent conversations. Google uses NLP to improve its search engine results, and social

networks like Facebook use it to detect and filter hate speech.

NLP is growing increasingly sophisticated, yet much work remains to be done. Current systems are prone to bias and incoherence, and occasionally behave erratically. Despite the challenges, machine learning engineers have many opportunities to apply NLP in ways that are ever more central to a functioning society.

What is NLP Used For? NLP is used for a wide variety of language-related tasks, including answering questions, classifying text in a variety of ways, and conversing with users. Here are 11 tasks that can be solved by NLP:

1. **Sentiment analysis:** is the process of classifying the emotional intent of text. Generally, the input to a sentiment classification model is a piece of text, and the output is the probability that the sentiment expressed is positive, negative, or neutral. Sentiment analysis is used to classify customer reviews on various online platforms as well as for niche applications like identifying signs of mental illness in online comments.
2. **Toxicity classification:** is a branch of sentiment analysis where the aim is not just to classify hostile intent but also to classify particular categories such as threats, insults, obscenities, and hatred towards certain identities. The input to such a model is text, and the output is generally the probability of each class of toxicity. Toxicity classification models can be used to moderate and improve online conversations by silencing offensive comments, detecting hate speech, or scanning documents for defamation.
3. **Machine translation:** automates translation between different languages. The input to such a model is text in a specified source language, and the output is the text in a specified target language. Google Translate is perhaps the most famous mainstream application. Such models are used to improve communication between people on social-media platforms such as Facebook or Skype. Effective approaches to machine translation can distinguish between words with similar meanings. Some systems also perform language identification; that is, classifying text as being in one language or another.
4. **Named entity recognition:** aims to extract entities in a piece of text into predefined categories such as personal names, organizations, locations, and quantities. The input to such a model is generally text, and the output is the various named entities along with their start and end positions. Named entity recognition is useful in applications such as summarizing news articles and combating disinformation.
5. **Spam detection:** is a prevalent binary classification problem in NLP, where the purpose is to classify emails as either spam or not. Spam detectors take as input an email text along with various other subtexts like title and sender's name. They aim to output the probability that the mail is spam. Email providers use such models to provide a better user experience by detecting unwanted emails and moving them to a designated spam folder.
6. **Grammatical error correction:** models encode grammatical rules to correct the grammar within text. This is viewed mainly as a sequence-to-sequence task, where a model is trained on an ungrammatical sentence as input and a correct sentence as output. Online grammar checkers like Grammarly and word-processing systems like Microsoft Word use such systems to provide a better writing experience to their customers. Schools also use them to grade student essays.
7. **Topic modeling:** is an unsupervised text mining task that takes a corpus of documents and discovers abstract topics within that corpus. The input to a topic model is a collection of documents, and the output is a list of topics that defines words for each topic as well as assignment proportions of each topic in a document. Latent Dirichlet Allocation (LDA), one of the most popular topic

modeling techniques, tries to view a document as a collection of topics and a topic as a collection of words. Topic modeling is being used commercially to help lawyers find evidence in legal documents.

8. **Text generation:** more formally known as natural language generation (NLG), produces text that's similar to human-written text. Such models can be fine-tuned to produce text in different genres and formats including tweets, blogs, and even computer code. Text generation has been performed using Markov processes, LSTMs, BERT, GPT-2, LaMDA, and other approaches. It's particularly useful for autocomplete and chatbots.

9. **Information retrieval:** finds the documents that are most relevant to a query. This is a problem every search and recommendation system faces. The goal is not to answer a particular query but to retrieve, from a collection of documents that may be numbered in the millions, a set that is most relevant to the query. Document retrieval systems mainly execute two processes: indexing and matching. In most modern systems, indexing is done by a vector space model through Two-Tower Networks, while matching is done using similarity or distance scores. Google recently integrated its search function with a multimodal information retrieval model that works with text, image, and video data.

10. **Summarization:** is the task of shortening text to highlight the most relevant information. Researchers at Salesforce developed a summarizer that also evaluates factual consistency to ensure that its output is accurate. Summarization is divided into two method classes:

i. **Extractive summarization** focuses on extracting the most important sentences from a long text and combining these to form a summary.
ii. **Abstractive summarization** produces a summary by paraphrasing.

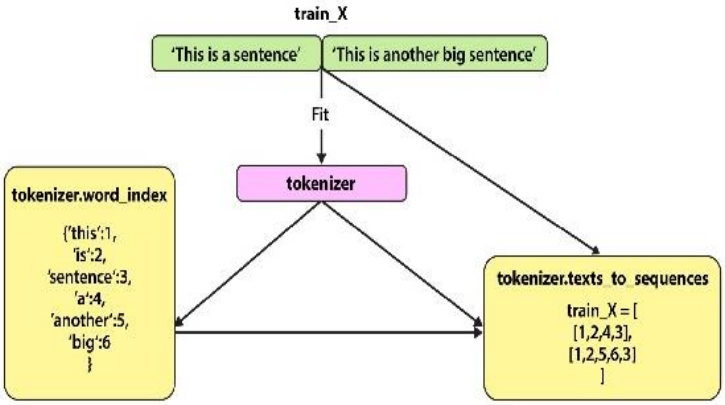
11. **Question answering:** deals with answering questions posed by humans in a natural language. One of the most notable examples of question answering was Watson, which in 2011 played the television game-show *Jeopardy* against human champions and won by substantial margins. Generally, question-answering tasks come in two flavors:

i. **Multiple choice:** The multiple-choice question problem is composed of a question and a set of possible answers. The learning task is to pick the correct answer.
ii. **Open domain**: In open-domain question answering, the model provides answers to questions in natural language without any options provided, often by querying a large number of texts.

How Does NLP Work? NLP models work by finding relationships between the constituent parts of language for example, the letters, words, and sentences found in a text dataset. NLP architectures use various methods for data preprocessing, feature extraction, and modeling. Some of these processes are:

i. **Data preprocessing:** Before a model processes text for a specific task, the text often needs to be preprocessed to improve model performance or to turn words and characters into a format the model can understand. Various techniques may be used in this data preprocessing:
    a. **Stemming and lemmatization**: Stemming is an informal process of converting words to their base forms using heuristic rules. For example, "university," "universities," and "university's" might all be mapped to the base *univers*. Lemmatization is a more formal way to find roots by analyzing a word's morphology using vocabulary from a dictionary.
    b. **Sentence segmentation:** breaks a large piece of text into linguistically meaningful sentence units.
    c. **Stop word removal:** aims to remove the most commonly occurring words that don't add much information to the text. For example, "the," "a," "an," and so on.

d. **Tokenization:** splits text into individual words and word fragments. The result generally consists of a word index and tokenized text in which words may be represented as numerical tokens for use in various deep learning methods.



ii. **Feature extraction:** Most conventional machine-learning techniques work on the features generally numbers that describe a document in relation to the corpus that contains it created by either Bag-of-Words, TF-IDF, or generic feature engineering such as document length, word polarity, and metadata (for instance, if the text has associated tags or scores). More recent techniques include Word2Vec, GLoVE, and learning the features during the training process of a neural network.

NLP Techniques

Most of the NLP tasks discussed above can be modeled by a dozen or so general techniques. It's helpful to think of these techniques in two categories: Traditional machine learning methods and deep learning methods.

Traditional Machine learning NLP techniques:

i. **Logistic regression** is a supervised classification algorithm that aims to predict the probability that an event will occur based on some input. In NLP, logistic regression models can be applied to solve problems such as sentiment analysis, spam detection, and toxicity classification.

ii. **Naive Bayes** is a supervised classification algorithm that finds the conditional probability distribution P(label | text) using the following Bayes formula:
   a. *P(label | text) = P(label) x P(text|label) / P(text)*
   b. In NLP, such statistical methods can be applied to solve problems such as spam detection or finding bugs in software code.

iii. **Decision trees** are a class of supervised classification models that split the dataset based on different features to maximize information gain in those splits.

iv. **Latent Dirichlet Allocation (LDA)** is used for topic modeling. LDA tries to view a document as a collection of topics and a topic as a collection of words. LDA is a statistical approach. The intuition behind it is that we can describe any topic using only a small set of words from the corpus.

v. **Hidden Markov models:** Markov models are probabilistic models that decide the next state of a system based on the current state. For example, in NLP, we might suggest the next word based on the previous word. We can model this as a Markov model where we might find the transition probabilities of going from word1 to word2, that is, P(word1|word2).

Deep learning NLP Techniques:

a. **Convolutional Neural Network (CNN):** The idea of using a CNN to classify text was first presented in the paper "Convolutional Neural Networks for Sentence Classification" by Yoon Kim. The central intuition is to see a document as an image. However, instead of pixels, the input is sentences or documents represented as a matrix of words.
b. **Recurrent Neural Network (RNN)**: Many techniques for text classification that use deep learning process words in close proximity using n-grams or a window (CNNs).

vi. **Autoencoders** are deep learning encoder-decoders that approximate a mapping from X to Y, i.e., input=output. They first compress the input features into a lower-dimensional representation (sometimes called a latent code, latent vector, or latent representation) and learn to reconstruct the input.

vii. **Encoder-decoder sequence-to-sequence:** The encoder-decoder seq2seq architecture is an adaptation to autoencoders specialized for translation, summarization, and similar tasks. The encoder encapsulates the information in a text into an encoded vector. Unlike an autoencoder, instead of reconstructing the input from the encoded vector, the decoder's task is to generate a different desired output, like a translation or summary.

viii. **Transformers:** The transformer,

Six Important NLP Models

Over the years, many NLP models have made waves within the AI community, and some have even made headlines in the mainstream news. The most famous of these have been chatbots and language models. Here are some of them: Eliza, Tay, BERT, Generative Pre-Trained Transformer 3 (GPT-3), Language Model for Dialogue Applications (LaMDA) and Mixture of Experts (MoE).

Here are the most useful languages that support NLP.

Python is the most-used programming language to tackle NLP tasks. Most libraries and frameworks for deep learning are written for Python. Here are a few that practitioners may find helpful:

R: Many early NLP models were written in R, and R is still widely used by data scientists and statisticians.

Applications of NLP: Chatbots, virtual assistants, sentiment analysis, translation, text summarization.

2. KNOWLEDGE REPRESENTATION

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. But how machines do all these things comes under knowledge representation and reasoning. The Purpose is to encode real-world information so AI systems can reason and make decisions. Hence, we can describe Knowledge representation as following:

i. Knowledge representation and reasoning (KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behaviour of agents.
ii. It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real-world problems such as diagnosis a medical condition or communicating with humans in natural language.
iii. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

Following are the kind of knowledge which needs to be represented in AI systems:

i. **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
ii. **Events:** Events are the actions which occur in our world.
iii. **Performance:** It describe behavior which involves knowledge about how to do things.
iv. **Meta-knowledge:** It is knowledge about what we know.
v. **Facts:** Facts are the truths about the real world and what we represent.
vi. **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

Knowledge: Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

i. Declarative Knowledge:

Declarative knowledge is to know about something. Facts and information about the world. It includes concepts, facts, and objects. It is also called descriptive knowledge and expressed in declarative sentences. E.g. Paris is the capital of France. Water boils at 100°C. Use in AI: Stored in databases or knowledge bases (e.g., knowledge graphs like Google's).

ii. Procedural Knowledge

It is also known as imperative knowledge. Procedural knowledge is a type of knowledge which is responsible for knowing how to do something. It can be directly applied to any task. It includes rules, strategies, procedures, agenda, etc. Procedural knowledge depends on the task on which it can be applied. E.g. how to drive a car, how to solve mathematical equation. Use in AI: Encoded as algorithms or rule-based systems.

iii. Meta-knowledge:

Knowledge about the other types of knowledge is called Meta-knowledge. E.g. Knowing that a certain model works better for image classification than for text. Use in AI: Helps AI choose appropriate reasoning or learning strategies.

iv. Heuristic knowledge:

Heuristic knowledge is representing knowledge of some experts in a filed or subject. Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed. E.g. If it's cloudy, it might rain. If a person avoids eye contact, they may be nervous. Use in AI: Expert systems use heuristics to make decisions in uncertain situations.

v. Structural knowledge:

Structural knowledge is basic knowledge to problem-solving. It describes relationships between various concepts such as kind of, part of, and grouping of something. It describes the relationship that exists between concepts or objects. E.g. A dog is a type of animal. An engine is a part of a car. Use in AI: Found in ontologies and semantic networks.

The relation between knowledge and intelligence:

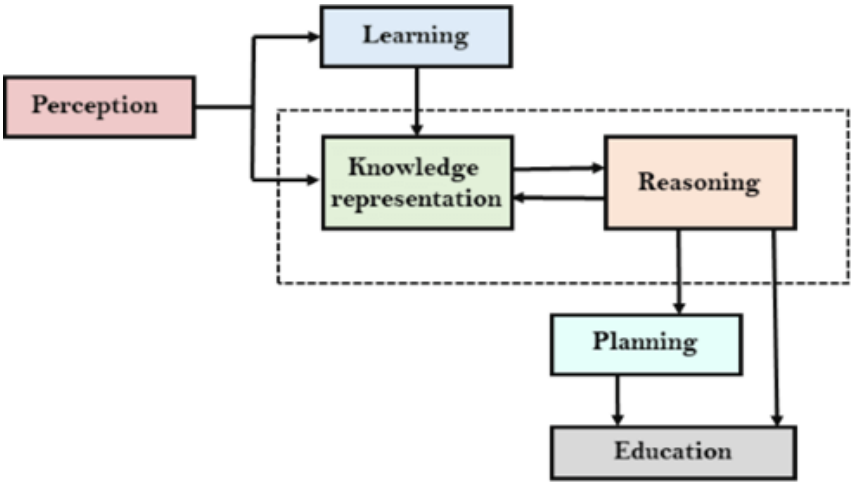
Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

AI knowledge cycle:

An AI system comprises the following key components that enable it to exhibit intelligent behaviour:

1. Perception
2. Learning
3. Knowledge Representation and Reasoning

4. Planning
5. Execution



Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

i. Simple relational knowledge
ii. Inheritable knowledge
iii. Inferential knowledge
iv. Procedural knowledge

i. Simple Relational Knowledge

This approach represents knowledge as facts stored in tables or databases, using a structure of objects and their attributes. It is the simplest form of knowledge representation and is easy to understand and query.

Example:

Student	Course	Grade
John	Math	A

This structure shows relationships between entities but does not handle complex reasoning.

ii. Inheritable Knowledge

Also known as hierarchical or frame-based representation, this approach organizes knowledge in a hierarchy where lower-level concepts inherit properties from higher-level ones.
Example:

i. Knowledge is represented as a sequence of steps or procedures to perform tasks. Unlike declarative knowledge, it focuses on "how to do" something.
   Animal → has skin
ii. Bird (inherits from Animal) → has feathers
iii. Parrot (inherits from Bird) → can talk

This allows sharing and reusing common properties across related concepts.

iii. Inferential Knowledge

This involves representing knowledge in the form of logical rules or if-then statements that allow the system to draw conclusions (make inferences). Used in expert systems and logic-based AI. E.g.

i. If it rains, then the ground becomes wet.
ii. It is raining → Therefore, the ground is wet.

iv. Procedural Knowledge

Knowledge is represented as a sequence of steps or procedures to perform tasks. Unlike declarative knowledge, it focuses on "how to do" something.

Example:
An algorithm for sorting a list or diagnosing a disease based on symptoms.
It's often encoded in the form of functions, scripts, or rules in rule-based systems.

3. SEARCH STRATEGIES

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.
In Artificial Intelligence (AI), a search strategy is the method or procedure used to decide which paths or states to explore in order to reach a solution to a problem.
Since many AI problems (like puzzle solving, pathfinding, game playing, or planning) can be represented as a search through a problem space, a strategy helps guide the search efficiently.
Key points:

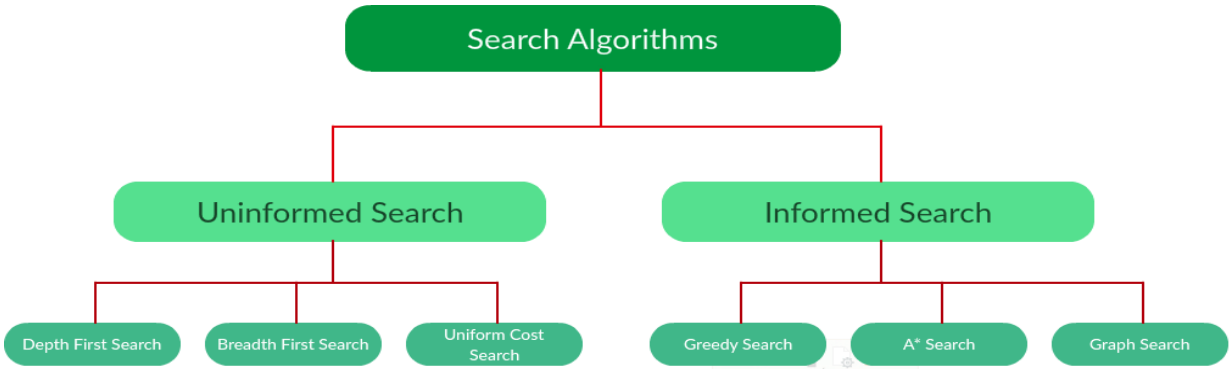
i. A search strategy defines the *order* in which nodes (possible states or solutions) are expanded.
ii. It determines which node is chosen first for exploration from the search tree or graph.
iii. The goal is usually to find a solution that is correct, efficient, and optimal.

A search problem consists of:

i. A State Space. Set of all possible states where you can be.
ii. A Start State. The state from where the search begins.
iii. A Goal State**.** is the desired final condition or outcome that the search algorithm is trying to reach. It's the state where the problem is considered *solved*.

The Solution to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state. This plan is achieved through search algorithms.

Types of search algorithms:



Uninformed Search Algorithms:

The search algorithms in this section have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and/or length of actions. Uninformed search is also called Blind search. These algorithms can only generate the successors and differentiate between the goal state and non-goal state.

i. Depth First Search ii. Breadth First Search iii. Uniform Cost Search

Each of these algorithms will have:

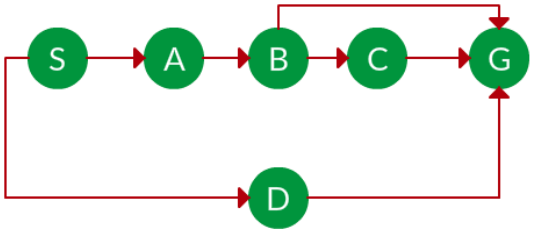
i. A problem **graph,** containing the start node S and the goal node G.
ii. A **strategy,** describing the manner in which the graph will be traversed to get to G.
iii. A **fringe,** which is a data structure used to store all the possible states (nodes) that you can go from the current states.
iv. A **tree,** that results while traversing to the goal node.
v. A solution **plan,** which the sequence of nodes from S to G.

Depth First Search:

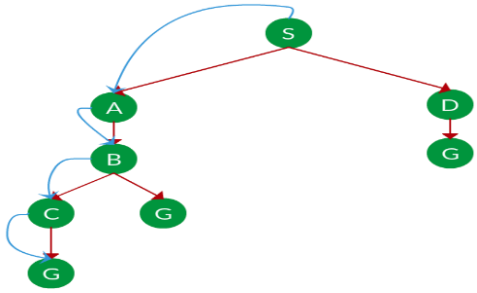
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. It uses last in- first-out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

Example:

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



Solution. The equivalent search tree for the above graph is as follows. As DFS traverses the tree "deepest node first", it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



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

Time complexity: *Equivalent to the number of nodes traversed in DFS.* $T(b) = 1 + b + b^2 + b^3 + \ldots + b^d = O(b^d)$

Space complexity: *Equivalent to how large can the fringe get.* $S(b) = O(b \times d)$

Completeness: *DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.*

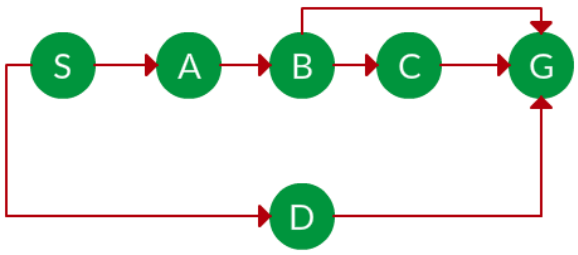
Optimality: *DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.*

Breadth First Search:

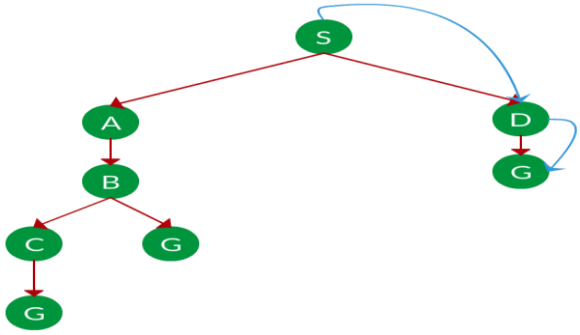
Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at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

Example:

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



Solution. The equivalent search tree for the above graph is as follows. As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Path: S -> D -> G

Time complexity: *Equivalent to the number of nodes traversed in BFS until the shallowest solution.* $T(b) = 1 + b + b^2 + b^3 + ... + b^s = O(b^s)$

Space complexity: *Equivalent to how large can the fringe get.* $S(b) = O(b^s)$

Completeness: *BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.*

Optimality: *BFS is optimal as long as the costs of all edges are equal.*

Uniform Cost Search:

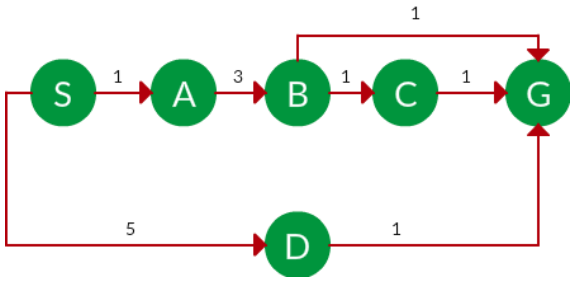
UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is the least.

Cost of a node is defined as:

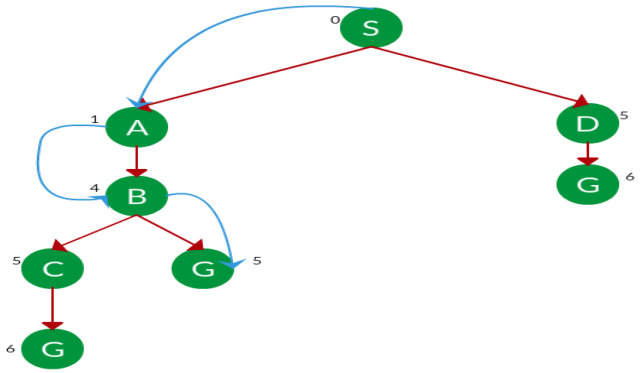
cost(node) = cumulative cost of all nodes from root cost(root) = 0

Example:

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



Solution. The equivalent search tree for the above graph is as follows. The cost of each node is the cumulative cost of reaching that node from the root. Based on the UCS strategy, the path with the least cumulative cost is chosen. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low, and discards them when a lower-cost path is found; these discarded traversals are not shown below. The actual traversal is shown in blue.



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

Cost: *5*

Advantages:

i. UCS is complete only if states are finite and there should be no loop with zero weight.
ii. UCS is optimal only if there is no negative cost.

Disadvantages:

i. Explores options in every "direction".
ii. No information on goal location.

Informed Search Algorithms:

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

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

Search Heuristics: In an informed search, a heuristic is a *function* that estimates how close a state is to the goal state. For example - Manhattan distance, Euclidean distance, etc. (Lesser the distance, closer the goal.) Different heuristics are used in different informed algorithms discussed below.

Greedy Search:

In greedy search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).

Heuristic: A heuristic h is defined as-

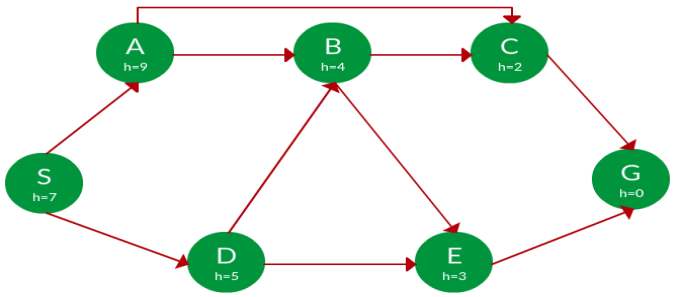
h(x) = Estimate of distance of node x from the goal node.

Lower the value of h(x), closer is the node from the goal.

Strategy: Expand the node closest to the goal state, *i.e.* expand the node with a lower h value.

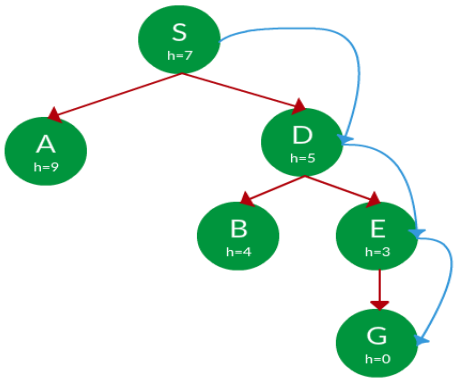
Example:

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



Solution. Starting from S, we can traverse to A(h=9) or D(h=5). We choose D, as it has the lower heuristic cost. Now from D, we can move to B(h=4) or E(h=3). We choose E with a lower heuristic

cost. Finally, from E, we go to G(h=0). This entire traversal is shown in the search tree below, in blue.



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

Advantage: Works well with informed search problems, with fewer steps to reach a goal.

Disadvantage: Can turn into unguided DFS in the worst case.

A* Tree Search

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

Heuristic: The following points should be noted with heuristics in A* search. f(x) = g(x) + h(x)

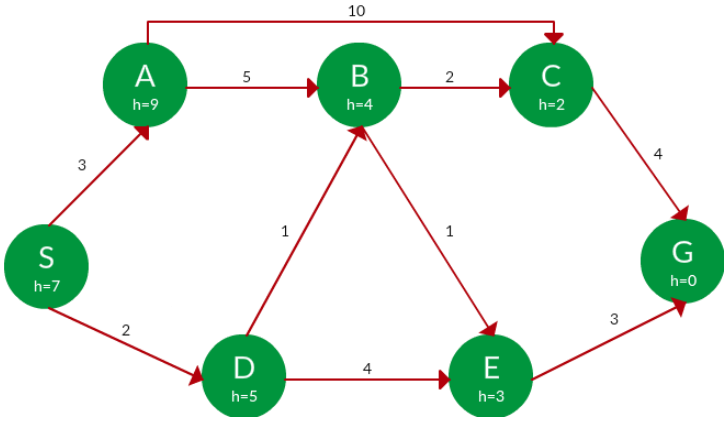
- Here, h(x) is called the **forward cost** and is an estimate of the distance of the current node from the goal node.
- And, g(x) is called the **backward cost** and is the cumulative cost of a node from the root node.
- A* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h*(x) to reach the goal. This property of *A** heuristic is called **admissibility**.

Admissibility: $0 \leq h(x) \leq h^*(x)$

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

Example:

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



Solution. Starting from S, the algorithm computes g(x) + h(x) for all nodes in the fringe at each step, choosing the node with the lowest sum. The entire work is shown in the table below. Note that in the fourth set of iterations, we get two paths with equal summed cost f(x), so we expand them both in the next set. The path with a lower cost on further expansion is the chosen path.

Path	h(x)	g(x)	f(x)

S	**7**	**0**	**7**
S -> A	**9**	**3**	**12**
S -> D	**5**	**2**	**7**
S -> D -> B	**4**	**2 + 1 = 3**	**7**
S -> D -> E	**3**	**2 + 4 = 6**	**9**
S -> D -> B -> C	**2**	**3 + 2 = 5**	**7**
S -> D -> B -> E	**3**	**3 + 1 = 4**	**7**
S -> D -> B -> C -> G	**0**	**5 + 4 = 9**	**9**
S -> D -> B -> E -> G	**0**	**4 + 3 = 7**	**7**

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

Cost: 7

A* Graph Search

A* tree search works well, except that it takes time re-exploring the branches it has already explored. In other words, if the same node has expanded twice in different branches of the search tree, A* search might explore both of those branches, thus wasting time

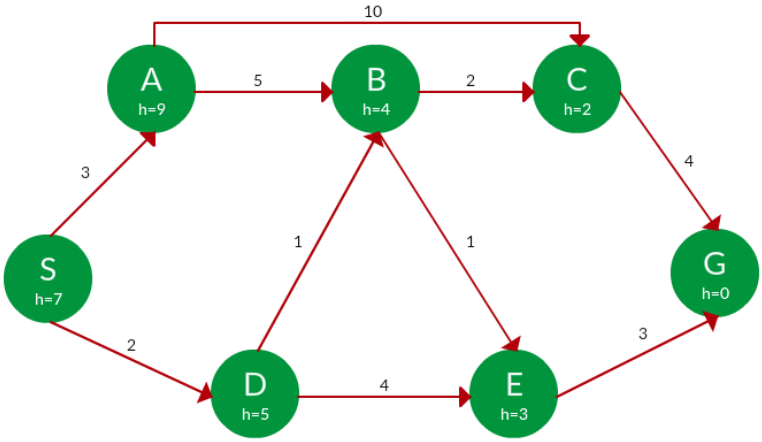
Graph Search removes this limitation by adding this rule: do not expand the same node more than once.

Heuristic. Graph search is optimal only when the forward cost between two successive nodes A and B, given by h(A) - h (B), is less than or equal to the backward cost between those two nodes g(A –> B). This property of the graph search heuristic is called **consistency**.

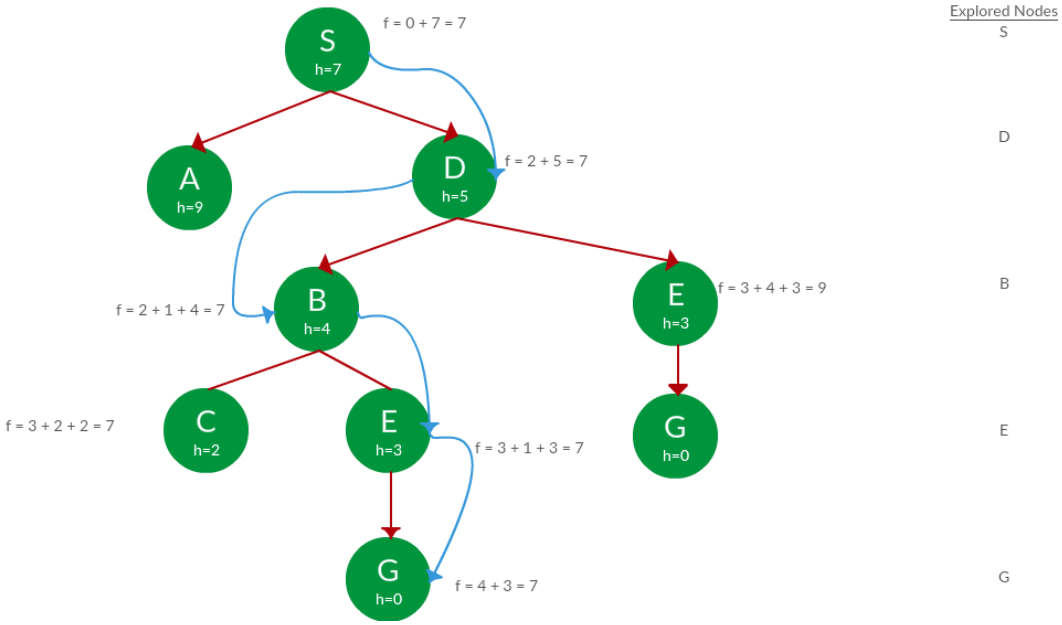
Consistency: $h(A) - h(B) \leq g(A \rightarrow B)$

Example:

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



We solve this question pretty much the same way we solved last question, but in this case, we keep a track of nodes explored so that we don't re-explore them.



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

4. SYMBOLIC LOGIC EXPERT SYSTEMS AND APPLICATIONS

Symbolic Logic

Symbolic logic, also known as formal logic or mathematical logic, is a discipline that represents logical expressions through the use of symbols and formal systems. It offers a powerful framework for analyzing the structure of arguments, distinguishing valid reasoning from invalid ones, and ensuring clarity and precision in the expression of ideas. Symbolic logic is foundational not only to philosophy and mathematics but also to computer science, artificial intelligence, linguistics, and cognitive science. Unlike natural language, which is often ambiguous and context-dependent, symbolic logic employs a formal language that uses well-defined symbols and rules. This helps in avoiding misinterpretation and provides a universal means for analyzing logical relationships. The study of symbolic logic strengthens critical thinking, analytical reasoning, and problem-solving skills, which are essential in both academic and real-world decision-making.

Propositions and Logical Statements

A proposition is a declarative sentence that can be clearly identified as either true or false, but not both. In symbolic logic, these statements are the fundamental units.

Examples of Propositions:

i. "The sun rises in the east." → True
ii. "2 + 2 = 5" → False
iii. "Nigeria is a country in Africa." → True

Non-examples (not propositions):

i. "What time is it?" → A question, not a proposition.
ii. "Read your book." → A command, not a proposition.

We often label propositions with letters such as:

i. p = "It is raining."
ii. q = "The ground is wet."

Logical Connectives

Logical connectives are operators that combine one or more propositions to form compound

statements. Each has specific truth rules.

a. Negation (¬p):

Means "not p." It reverses the truth value of a statement.

p	¬p
T	F
F	T

Example:

If p = "It is hot."

Then ¬p = "It is not hot."

b. Conjunction (p ∧ q):

Means "p and q." True only if both p and q are true.

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

Example:

p = "I have a pen", q = "I have paper"

p ∧ q = "I have a pen and I have paper"

c. Disjunction (p ∨ q):

Means "p or q." True if at least one of p or q is true.

p	q	p ∨ q
T	T	T
T	F	T
F	T	T
F	F	F

Example:

p = "Today is Monday", q = "Today is Friday"

p ∨ q = "Today is Monday or Friday"

d. Conditional (p → q):

Means "if p, then q." False only if p is true and q is false.

p	q	p → q
T	T	T
T	F	F
F	T	T
F	F	T

Example:

p = "It rains", q = "The ground gets wet"

p → q = "If it rains, then the ground gets wet"

e. Biconditional (p ↔ q):

Means "p if and only if q." True when both p and q have the same truth value.

p	q	p ↔ q
T	T	T
T	F	F

F	T	F
F	F	T

Example:

p = "The switch is on", q = "The light is on"

p ↔ q = "The switch is on if and only if the light is on"

Translation from Natural Language to Symbolic Form

The skill of symbolic logic lies in converting natural language statements into symbolic form for clarity and analysis.

Examples:

Natural Language	Symbolic Logic
"It is raining."	p
"It is not raining."	¬p
"It is raining and I have an umbrella."	p ∧ q
"If it rains, then I will stay home."	p → q
"I will pass the course if and only if I study."	p ↔ q

More Complex Example:

Statement: "If today is Monday and it is raining, then I will stay indoors."

Let:

i. p = "Today is Monday"
ii. q = "It is raining"
iii. r = "I will stay indoors"

Symbolic form: (p ∧ q) → r

Logical Equivalence

A truth table lists all possible truth values of propositions and shows the result of logical operations.

Checking Logical Equivalence

Are ¬(p ∧ q) and (¬p ∨ ¬q) logically equivalent?

Construct truth tables:

p	q	p ∧ q	¬(p ∧ q)	¬p	¬q	¬p ∨ ¬q
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Since the last two columns are equal, the statements are logically equivalent (De Morgan's Law).

Expert Systems and Applications

An expert system is a computer program designed to simulate the decision-making ability of a human expert. It uses a knowledge base of facts and rules and an inference engine to apply logical rules to the known facts and deduce new facts. The goal is to provide expert-level solutions, explanations, and recommendations in a specific domain.

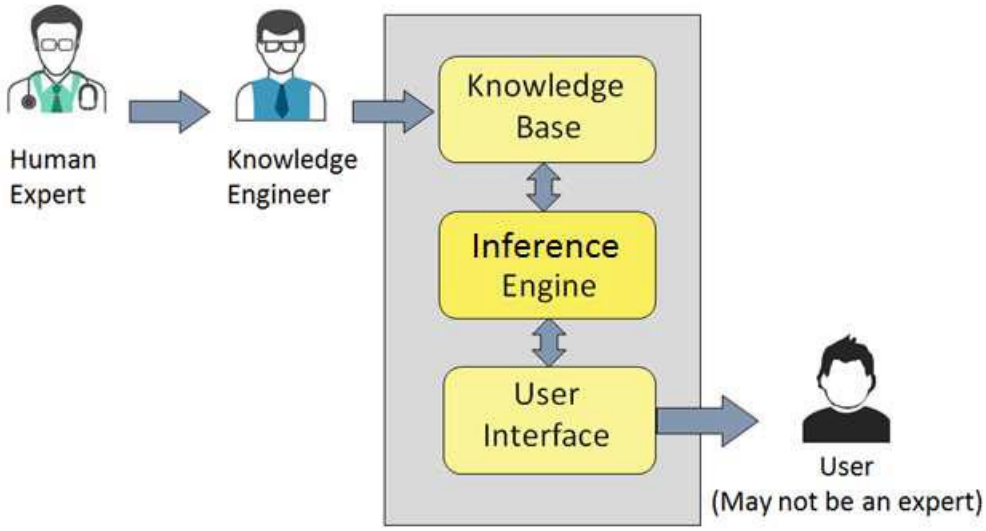
Features of Expert Systems

i. High performance in a narrow domain.
ii. Explanation ability (can justify conclusions).
iii. Reasoning under uncertainty (using probabilistic logic or fuzzy logic).
iv. Rule-based or case-based reasoning.
v. User-friendly interfaces.

Basic Components of an Expert System

Component	Description

Knowledge Base	Contains domain-specific facts and rules.
Inference Engine	Applies logical rules to the knowledge base to deduce conclusions.
User Interface	Allows users to interact with the system.



Knowledge Base

It contains domain-specific and high-quality knowledge. Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

i. Factual Knowledge – It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
ii. Heuristic Knowledge – It is about practice, accurate judgment, one's ability of evaluation, and guessing.

Knowledge Acquisition

The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base. The knowledge base is formed by readings from various experts, scholars, and the **Knowledge Engineers**. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills. He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Inference Engine

Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct, flawless solution. In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.

In case of rule-based ES, it:

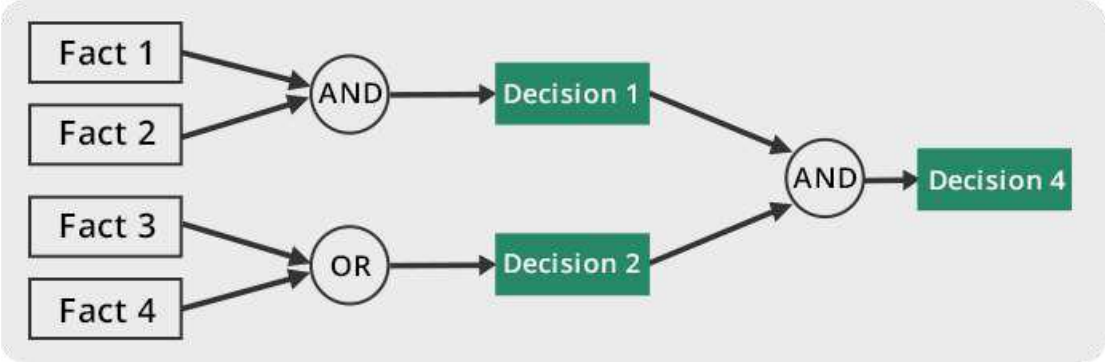
i. Applies rules repeatedly to the facts, which are obtained from earlier rule application.
ii. Adds new knowledge into the knowledge base if required.
iii. Resolves rules conflict when multiple rules are applicable to a particular case

To recommend a solution, the inference engine uses the following strategies:

i. Forward Chaining
ii. Backward Chaining

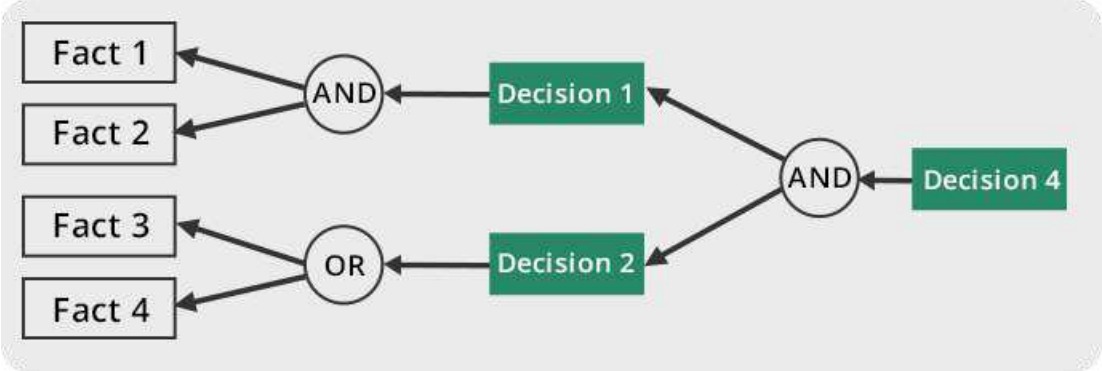
Forward Chaining

It is a strategy of an expert system to answer the question, **"What can happen next?"** Here, the inference engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution. This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



Backward Chaining

With this strategy, an expert system finds out the answer to the question, **"Why this happened?"** On the basis of what has already happened, the inference engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.



User Interface

User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence. It explains how the ES has arrived at a particular recommendation. The explanation may be in the following forms:

i. Natural language displayed on screen
ii. Verbal narrations in natural language
iii. Listing of rule numbers displayed on the screen.

The user interface makes it easy to trace the credibility of the deductions.

Requirements of Efficient ES User Interface

i. It should help users to accomplish their goals in shortest possible way.
ii. It should be designed to work for user's existing or desired work practices.
iii. Its technology should be adaptable to user's requirements; not the other way round.
iv. It should make efficient use of user input.

Expert Systems Limitations

No technology can offer easy and complete solution. Large systems are costly, require significant development time, and computer resources. ESs have their limitations which include:

i. Limitations of the technology

ii. Difficult knowledge acquisition
iii. ES are Difficult to maintain
iv. High Development costs

Applications of Expert System

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.
Agriculture	Pest diagnosis and crop management and Soil and fertilizer advisory systems.
Education	Intelligent tutoring systems and Personalized learning paths.

Expert System Technology

There are several levels of ES technologies available. Expert systems technologies include:

a. Expert System Development Environment: The ES development environment includes hardware and tools. They are:

i. Workstations, minicomputers, mainframes
ii. High level Symbolic Programming Languages such as **LIS P**rogramming (LISP) and **PRO**grammation en **LOG**ique (PROLOG).
iii. Large databases

b. Tools: They reduce the effort and cost involved in developing an expert system to large extent.

i. Powerful editors and debugging tools with multi-windows.
ii. They provide rapid prototyping
iii. Have Inbuilt definitions of model, knowledge representation, and inference design.

c. Shells: A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility. For example, few shells are given below:

i. Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.
ii. *Vidwan*, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

Development of Expert Systems: General Steps

The process of ES development is iterative. Steps in developing the ES include:

a. Identify Problem Domain

i. The problem must be suitable for an expert system to solve it.
ii. Find the experts in task domain for the ES project.
iii. Establish cost-effectiveness of the system.

b. Design the System

i. Identify the ES Technology.

ii. Know and establish the degree of integration with the other systems and databases.
iii. Realize how the concepts can represent the domain knowledge best.

c. Develop the Prototype

Form Knowledge Base: The knowledge engineer works to:

i. Acquire domain knowledge from the expert.
ii. Represent it in the form of If-THEN-ELSE rules.

d. Test and Refine the Prototype

i. The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
ii. End users test the prototypes of the ES.

e. Develop and Complete the ES

i. Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
ii. Document the ES project well.
iii. Train the user to use ES.

f. Maintain the System

i. Keep the knowledge base up-to-date by regular review and update.
ii. Cater for new interfaces with other information systems, as those systems evolve.

Benefits of Expert Systems

i. **Availability:** They are easily available due to mass production of software.
ii. **Less Production Cost:** Production cost is reasonable. This makes them affordable.
iii. **Speed:** They offer great speed. They reduce the amount of work an individual puts in.
iv. **Less Error Rate:** Error rate is low as compared to human errors.
v. **Reducing Risk:** They can work in the environment dangerous to humans.
vi. **Steady response:** They work steadily without getting motional, tensed or fatigued.

Limitations

i. Limited to specific domains
ii. Expensive and time-consuming to develop
iii. Cannot learn or adapt unless updated manually
iv. Lacks common sense and emotional intelligence

Future Trends

i. Integration with machine learning and big data
ii. Use in real-time decision-making systems
iii. Mobile and cloud-based expert systems
iv. Hybrid systems combining expert knowledge with learning algorithms

5. AI GAMES AND THEOREMS PROVING

AI Games

AI games refer to the use of artificial intelligence techniques in computer games to make them more interactive, challenging, and realistic. It involves designing algorithms that allow non-player characters (NPCs) or opponents to simulate human-like decision making, adapt to player actions, and create engaging experiences. AI techniques allow virtual players to adapt to different situations, predict strategies, and provide dynamic gameplay.

i. AI in games helps control NPC behavior (e.g., enemies, allies).
ii. It uses methods like search strategies, rule-based systems, pathfinding (A*), finite state machines, machine learning, and reinforcement learning.
iii. The goal is to make gameplay dynamic, fun, and less predictable, while balancing difficulty.

Examples**:**

i. In *Chess*, AI decides optimal moves.
ii. In *FIFA*, AI controls teammates and opponents.
iii. In *Call of Duty*, AI governs enemy behavior and tactics.

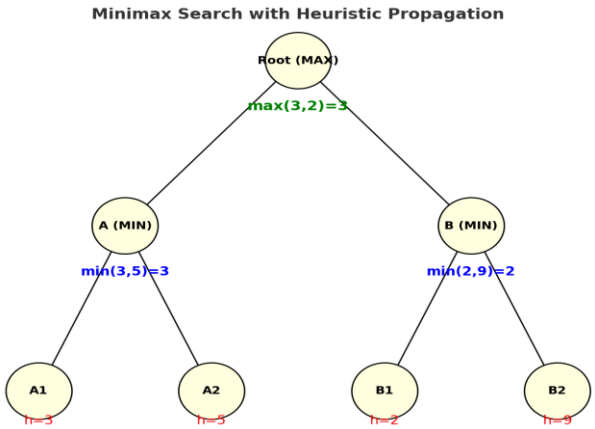
Minmax Search

Minmax is a decision-making algorithm used in two-player games (like Chess, Checkers, and Tic-Tac-Toe). It assumes that both players play optimally:

i. MAX player: tries to maximize the score.
ii. MIN player: tries to minimize the score.

The algorithm explores a game tree of possible moves:

i. MAX chooses the highest value.
ii. MIN chooses the lowest value.



Leaves: Heuristic values are shown in red.

MIN nodes: Choose the smaller value from their children

A(MIN) = min(3,5) = 3

B(MIN) = min(2,9) = 2

Root (MAX): Chooses the larger between A and B

Root = max(3,2) = 3

Final decision: The best move for MAX is via A, leading to a heuristic value of 3.

Heuristic Evaluation in Minimax

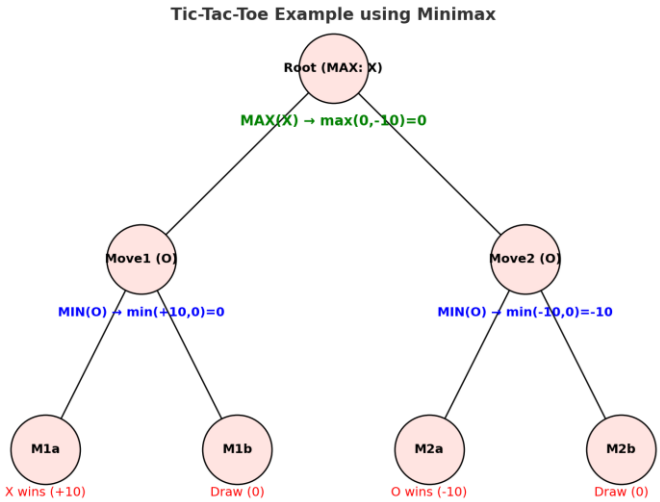
In large games like Chess, exploring the full game tree is impractical. Instead, the search is limited to a certain depth, and a heuristic evaluation function is used.

Heuristics = rules to estimate how good a board position is.

Example (Chess heuristic):

i. Queen = 9 points
ii. Rook = 5 points
iii. Knight/Bishop = 3 points
iv. Pawn = 1 point

The algorithm sums up these values (including strategic factors like king safety, control of the board) to evaluate positions.



Step 1: O (MIN) chooses minimum values

i. Move1 → min(+10, 0) = 0
ii. Move2 → min(-10, 0) = -10

Step 2: X (MAX) chooses maximum

i. Root → max(0, -10) = 0

The best X can do is force a draw (0) instead of losing.

Applications of Minimax in AI Games

i. Tic-Tac-Toe: AI can play perfectly and never lose.
ii. Chess: AI like Deep Blue uses minimax with heuristics and pruning.
iii. Checkers: Perfect play can force draws.

Minimax is a core algorithm for AI in games. It models competitive situations where two players oppose each other.

Heuristics allow decision-making in large games. Practical implementations use optimizations like Alpha-Beta Pruning to reduce computation.

What is Tic-Tac-Toe?

i. A two-player game played on a 3×3 grid.
ii. Players take turns placing their symbols:
    X (usually goes first)
    O (goes second)
iii. The goal is to be the first to align three of your symbols either:
    Horizontally (row),
    Vertically (column), or
    Diagonally.

Game Rules

1. The game starts with an empty 3×3 grid.
2. Players alternate turns, placing their mark (X or O) in an empty square.
3. The first player to form a straight line of three identical marks wins.
4. If all 9 squares are filled and no player has three in a row, the game ends in a draw.

Background and Evolution

The concept of AI in games dates back to the 1950s, when early experiments like *Tic-Tac-Toe* and *Chess* programs were developed on mainframe computers. These weren't "video games" as we know them today, but they showed how algorithms could simulate human-like opponents. From there, Game AI evolved through several key stages:

1. Early Arcade AI (1970s–1980s)

i. Games like Pong (1972) and Space Invaders (1978) introduced simple scripted behaviors.
ii. Pac-Man (1980) became famous for its four ghosts, each following a unique movement algorithm an early example of differentiated NPC behavior.

2. Rule-Based and Scripted AI (1980s–1990s)

i. AI followed if–then rules: if the player is near, attack; if health is low, retreat.
ii. Games like Doom (1993) used fixed patterns with some randomness for enemy movement.

3. Search and Pathfinding (1990s)

- Introduction of algorithms like A* for pathfinding in large environments.
- Real-Time Strategy (RTS) games like Warcraft II (1995) used AI for resource management and tactical combat.

4. Adaptive and Learning AI (2000s)

i. AI started to adapt to player behavior.
ii. Sports games (like FIFA) and racing games (like Gran Turismo) used machine learning and recorded player data to simulate realistic opponents.

5. Modern AI in Games (2010s–present)

i. Integration of machine learning, neural networks, and procedural content generation.
ii. Games like Middle-earth: Shadow of Mordor introduced the Nemesis System, where enemies remembered the player's actions and changed behavior accordingly.
iii. Deep reinforcement learning is now used for complex decision-making, as seen in research versions of StarCraft II bots.

What AI Game Is All About

At its core, Game AI is about three intertwined goals:

i. **Challenge**: Creating intelligent opponents that push the player's skills without making them feel unfairly punished.
ii. **Immersion**: Making the game world feel alive, believable, and responsive.
iii. **Replayability**: Using unpredictability and adaptive systems so the game feels fresh every time.

Core Components of Game AI

i. **Decision-Making Systems**: Finite State Machines, Decision Trees, Behavior Trees, Utility Systems.
ii. **Navigation and Pathfinding**: Algorithms like A*, Dijkstra's, Navigation Meshes.
iii. **Scripting and Event Triggers**: For controlling NPC actions during key events.
iv. **Procedural Content Generation (PCG)**: Automatically generating worlds, levels, or quests.
v. **Learning Systems**: Machine learning for adaptive difficulty, player modeling, and self-learning agents.
vi. **Opponent Modeling**: Predicting player moves to create smarter enemies.

Key Difference from Real-World AI

i. Game AI focuses on player experience, fun, and balance, even if it means "cheating" to create a good challenge.
ii. Real-world AI focuses on correctness, efficiency, and solving practical problems without biasing results for entertainment.

Conceptual Map of Game AI

[Game World State]
↓

i. Perception
↓
ii. Decision-Making
↓
iii. Action Execution
↓
iv. 4. Feedback & Adaptation
↺ (loop repeats)

i. Perception (Sensing the World)

Just like humans use their senses, Game AI needs to perceive its surroundings.

a. Input Sources:
   - Player location, health, and actions
   - Environment data (obstacles, cover, weather)
   - Other NPC positions and states

b. **Techniques**:
   - **Raycasting**: To check line of sight (e.g., enemy spotting you around a corner).
   - **Trigger Zones**: Define regions that detect when a player enters.
   - **Field of View (FOV)**: Restricting NPC awareness to a vision cone.

Example: In *Metal Gear Solid*, guards patrol until your character enters their vision cone, then they react.

ii. Decision-Making (Choosing What to Do)

Once the AI knows what's happening, it needs to decide how to respond.

Common approaches:

a. **Finite State Machines (FSM):** NPCs switch between states like "Patrol → Chase → Attack → Flee."
b. **Decision Trees**: Hierarchical yes/no checks leading to an action.
c. **Behavior Trees**: Modular, reusable decision structures (common in AAA games).
d. **Utility AI**: Assign scores to possible actions and pick the best.
e. **Rule-Based Systems**: If-then logic for simple NPCs.

Example: In *Halo*, enemies retreat when low on health, flank if possible, and attack in groups.

iii. Action Execution (Doing It)

After deciding, the AI **performs the action**:

a. **Animation Control**: Play the right movement, attack, or interaction.
b. **Pathfinding**: Move to target location using algorithms like **A*** or navigation meshes.
c. **Tactical Positioning**: Taking cover, avoiding traps, staying in advantageous spots.

Example: In *The Last of Us*, enemies coordinate movements to flush you out of cover.

iv. Feedback & Adaptation (Learning and Adjusting)

Modern AI often monitors its success or failure and adjusts future behavior.

a. **Dynamic Difficulty Adjustment (DDA)**: Makes the game harder or easier depending on player skill.
b. **Reinforcement Learning**: AI improves by trial and error.
c. **Player Profiling**: Predicts your style and counters it.

Example: In *Left 4 Dead*, the "AI Director" spawns zombies and resources differently each playthrough based on player performance.

Quick Visual Flow

Sense → Think → Act → Learn → (Repeat)

This loop runs every frame or game tick, making NPCs seem alive.

Example 1: Enemy Guard Patrol and Chase

Scenario: A guard patrols between waypoints. If it spots the player, it chases them. If it loses sight, it returns to patrol.

Behavior Model: Finite State Machine (FSM)

States: Patrol, Chase, Search

Pseudo-Code:

```
state = "Patrol"
while game_running:
    if state == "Patrol":
        move_to_next_waypoint()
        if player_in_fov():
            state = "Chase"
        elif state == "Chase":
        move_towards(player.position)
        if not player_in_fov():
            last_known_position = player.position
            state = "Search"
      elif state == "Search":
        move_to(last_known_position)
        if player_in_fov():
            state = "Chase"
        elif search_time_expired():
            state = "Patrol"
```

Real Game Example: Guards in *Metal Gear Solid* and *Assassin's Creed*.

Example 2: Zombie Horde AI

Scenario: Zombies wander until they detect a sound or see a player, then they swarm.

Behavior Model: **Utility AI**

Actions: Wander, Chase, Attack

Scoring System:

i. Wander = 1 point
ii. Chase = 5 points if player visible, +2 points if sound detected
iii. Attack = 10 points if within range

Pseudo-Code:

```
actions = {
    "wander": lambda: wander_around(),
    "chase": lambda: move_towards(player.position),
    "attack": lambda: bite_player()
}
def choose_action():
    scores = {}
    scores["wander"] = 1
    scores["chase"] = (5 if player_in_fov() else 0) + (2 if sound_detected() else 0)
    scores["attack"] = 10 if in_attack_range(player) else 0
    return max(scores, key=scores.get)
while game_running:
    action = choose_action()
    actions[action]()
```

Real Game Example: Zombie behavior in *Left 4 Dead*.

Example 3: Racing Game Opponent AI

Scenario: Opponent racers follow a track, avoid collisions, and adapt speed based on curves and the player's position.

Behavior Model: Waypoint Navigation + Dynamic Difficulty Adjustment

Pseudo-Code:

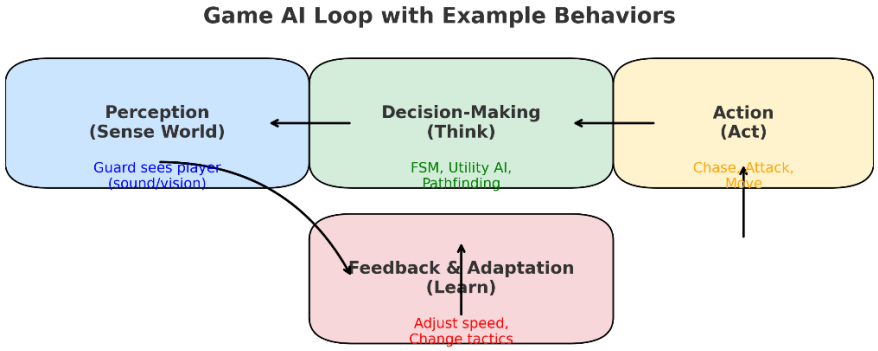
```
speed = base_speed
while race_in_progress:
    next_point = get_next_waypoint()
    steer_towards(next_point)
    if curve_ahead():
```

```
    speed = base_speed * 0.8
  else:
    speed = base_speed
  if player_ahead():
    speed += 2  # catch up boost
  if close_to_player():
    try_to_overtake()
  move_forward(speed)
```

Real Game Example: *Gran Turismo*, *Need for Speed* opponent AI.

How These Examples Fit the AI Loop

i. **Perception**: Detecting player, sound, obstacles.
ii. **Decision-Making**: FSM, Utility Scores, or Adaptive Adjustments.
iii. **Action Execution**: Movement, attack, pathfinding.
iv. **Feedback**: Adjusting speed or tactics over time.



This diagram shows how perception, decision-making, action, and feedback fit together with examples like guard patrol, zombie chase, and racing AI.

Theorem Proving in Artificial Intelligence

In Artificial Intelligence (AI), *theorem proving* refers to the use of computational techniques to automatically or semi-automatically establish the truth or falsehood of logical statements based on a set of axioms and inference rules.
It plays a central role in knowledge representation, automated reasoning, and expert systems, enabling AI systems to deduce new facts from existing knowledge.
Key idea:
If knowledge can be represented formally, AI can reason about it using logical inference to prove or disprove propositions.
AI Perspective: Theorem proving is not about solving textbook math proofs but about enabling a computer to reason like a human, proving facts, verifying conditions, and finding contradictions.
Relation to Knowledge Representation: Often integrated with predicate logic, semantic networks, or ontologies.
AI Applications:

i. Verifying system safety in autonomous vehicles
ii. Proving logical rules in expert medical systems
iii. Validating game-winning strategies in Game AI
iv. Security protocol verification

Types of Theorems Proving in AI

i. **Automated Theorem Proving (ATP):** Fully automated systems where a computer finds proofs without human intervention. Examples: Prover9, Vampire, E, Coq.
ii. **Interactive Theorem Proving (ITP):** Human guides the proof process, and the system checks correctness. Examples: Isabelle, HOL Light, Lean.

Logical Foundations in AI Theorem Proving

Propositional Logic: Deals with statements that are true or false. Example:

If *It is raining* → *The ground is wet*

and *It is raining*

⇒ *The ground is wet*.

Predicate Logic (First-Order Logic, FOL): More expressive; deals with objects, properties, and relations. Example:

∀x (Human(x) → Mortal(x))

Human(Socrates)

⇒ Mortal(Socrates)

Common Theorem Proving Methods in AI

Method	Description	AI Use Case
Resolution	Converts statements to conjunctive normal form (CNF) and applies resolution rule until goal is found or refuted	Expert systems, knowledge bases
Natural Deduction	Uses inference rules like Modus Ponens and Modus Tollens to build proofs	Planning systems
Forward Chaining	Data-driven reasoning from known facts to conclusions	Rule-based AI
Backward Chaining	Goal-driven reasoning from conclusions back to facts	Prolog-based AI
Model Checking	Verifies if a statement holds in all possible states of a model	AI safety verification

Theorem Proving in AI Systems

i. **Knowledge Representation**: Define axioms in logic form.
ii. **Inference Engine**: Apply logical rules to derive conclusions.
iii. **Proof Search**: Explore search space of possible proof steps.
iv. **Output**: Either a proof, a counterexample, or failure.

Example in AI

Scenario: In a medical expert system

Given:

∀x (Flu(x) → Fever(x))

Flu(John)

Prove: Fever(John)

Process (Resolution):

i. Step 1: From (1), rewrite in CNF: ¬Flu(x) ∨ Fever(x)
ii. Step 2: From (2): Flu(John)
iii. Step 3: Resolve: ¬Flu(John) ∨ Fever(John) with Flu(John) ⇒ Fever(John)

Challenges in AI Theorem Proving

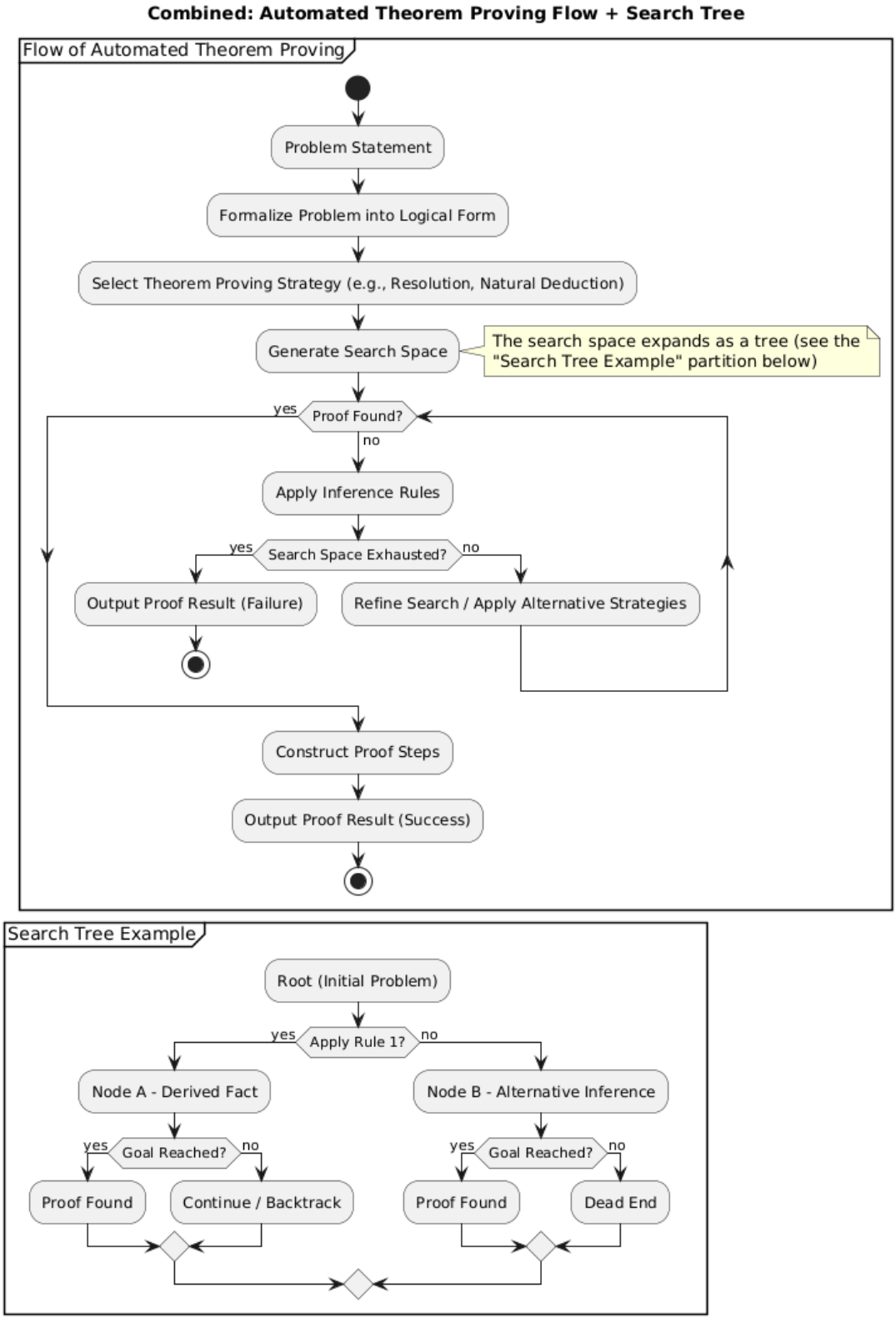
i. **Computational Complexity**: Proof search may be exponential.
ii. **Incomplete Knowledge**: Missing axioms can block proofs.
iii. **Ambiguity**: Natural language translation to logic can be difficult.
iv. **Scalability**: Handling large, real-world datasets is challenging.

Modern Trends

i. **Neuro-Symbolic AI**: Combining neural networks with symbolic theorem proving for robust reasoning.
ii. **Machine Learning-assisted Proof Search**: Using AI to guide which inference steps to try first.
iii. **Game AI**: Using theorem proving to reason about win/lose conditions and generate optimal strategies.

Theorem proving in AI is a computational reasoning process that enables intelligent systems to deduce, verify, and discover new knowledge based on logic. It is essential for building reliable, explainable, and trustworthy AI systems in domains ranging from healthcare to autonomous robotics.

Flow of Automated Theorem Proving in AI: Shows the pipeline from problem statement to proof result. And **Search Tree Example for Theorem Proving**: Illustrates branching of proof attempts leading to goal or dead ends.



6. PATTERN RECOGNITION AND MACHINE LEARNING

In our digital age, pattern is all that is around you. A pattern can be noticed either physically or mathematically through the use of various algorithms. Both methods are viable options. The classification of data based on previously acquired knowledge or on statistical information extracted from patterns and/or their representation is an example of pattern recognition.

Pattern recognition is the process of identifying and classifying input data, such as text, audio, and photos, by applying machine learning algorithms to outline patterns in the given data. This process can be applied to data such as text, speech, and photographs. Also, is the science for observing, distinguishing the patterns of interest, and making correct decisions about the patterns or pattern classes. Thus, a biometric system applies pattern recognition to identify and classify the individuals, by comparing it with the stored templates. Examples of Pattern Recognition:

i. Speech recognition,
ii. Speaker identification,
iii. Multimedia document recognition (MDR),
iv. Automatic medical diagnosis.

During the course of a typical pattern recognition application, the raw data is subjected to processing and then transformed into a format that can be utilized by a computer. The process of pattern recognition includes categorizing and clustering different patterns.

In general, a pattern can be a fingerprint image, a handwritten cursive word, a human face, a speech signal, a bar code, or a web page on the Internet. The individual patterns are often grouped into various categories based on their properties. When the patterns of same properties are grouped together, the resultant group is also a pattern, which is often called a pattern class.

The pattern recognition technique conducts the following tasks:

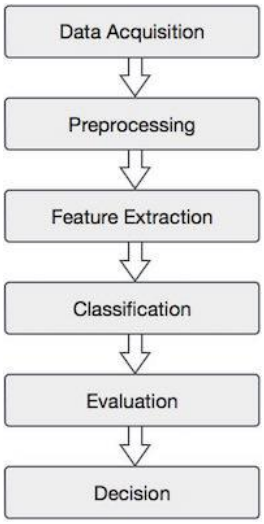
i. **Classification:** Identifying handwritten characters, CAPTCHAs, distinguishing humans from computers.
ii. **Segmentation**: Detecting text regions or face regions in images.
iii. **Syntactic Pattern Recognition**: Determining how a group of math symbols or operators are related, and how they form a meaningful expression.

The following table highlights the role of pattern recognition in biometrics:

Pattern Recognition Task	**Input**	**Output**
Character Recognition (Signature Recognition)	Optical signals or Strokes	Name of the character
Speaker Recognition	Voice	Identity of the speaker
Fingerprint, Facial image, hand geometry image	Image	Identity of the user

Components of Pattern Recognition

Pattern recognition technique extracts a random pattern of human trait into a compact digital signature, which can serve as a biological identifier. The biometric systems use pattern recognition techniques to classify the users and identify them separately.



Benefits of Pattern Recognition

i. Problems with classification can be solved using pattern recognition.
ii. The issue of false positives in biometric identification can be resolved by pattern recognition.
iii. People who are visually handicapped or blind can benefit from using it to recognize patterns on cloth.
iv. It is useful for diarizing the speaker's thoughts.
v. Certain things are recognizable to us no matter what angle we view them from.

Drawbacks of Pattern Recognition

i. The method of syntactic pattern recognition is difficult to put into practice because it is such a tedious procedure.
ii. Sometimes having a larger dataset is necessary in order to attain better accuracy.
iii. It is unable to provide an explanation as to why a specific object is recognized.
iv. Consider the difference between my face and the face of my friend.

Applications of Image Processing

i. **Image processing, segmentation, and analysis** Image processing necessitates the employment of machines that are capable of human-like recognition, and this can be accomplished through the application of pattern recognition.
ii. **Computer vision** In the field of computer vision, pattern recognition is utilised for a variety of applications, including imaging in the fields of biology and medicine, in order to extract meaningful characteristics from given image or video samples.
iii. **Seismic analysis** In seismic array recordings, the pattern recognition approach is employed for the purpose of discovering, visualizing, and providing an interpretation of temporal patterns. Several distinct kinds of seismic analysis models incorporate and make use of statistical pattern recognition.
iv. **Radar signal classification/analysis** Methods of pattern recognition and signal processing are utilized in a wide variety of applications of radar signal classifications, including the detection and identification of AP mines.
v. **Speech recognition** Pattern recognition models have shown to be the most successful in voice recognition efforts. [Citation needed] [Citation needed] It is utilized in numerous speech recognition algorithms, all of which treat larger units, such as words, as patterns in an effort to circumvent the issues that arise from attempting to describe speech at the phoneme level.
vi. **Fingerprint identification** The market for biometric technology is dominated by the technology that can recognize a person's fingerprint. The process of matching fingerprints has been carried out with the assistance of a variety of recognition strategies, the most common of which being pattern recognition.

The most popular pattern generation algorithms are:

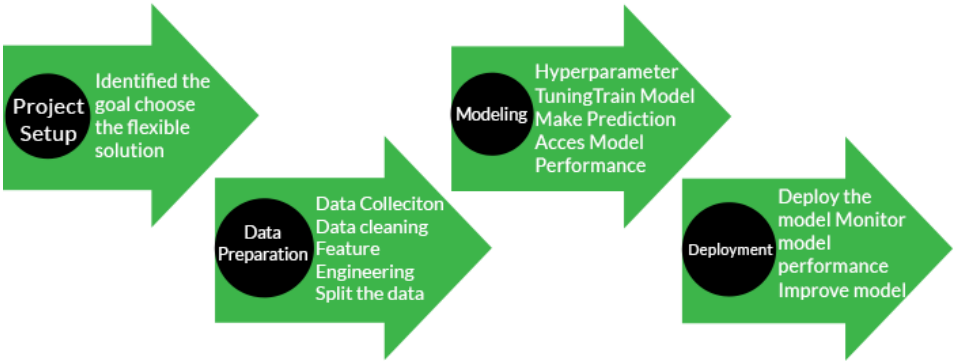
i. **Nearest Neighbor Algorithm**: You need to take the unknown individuals vector and compute its distance from all the patterns in the database. The smallest distance gives the best match.
ii. **Back-Propagation (Backprop) Algorithm:** It is a bit complex but very useful algorithm that involves a lot of mathematical computations.

Machine Learning

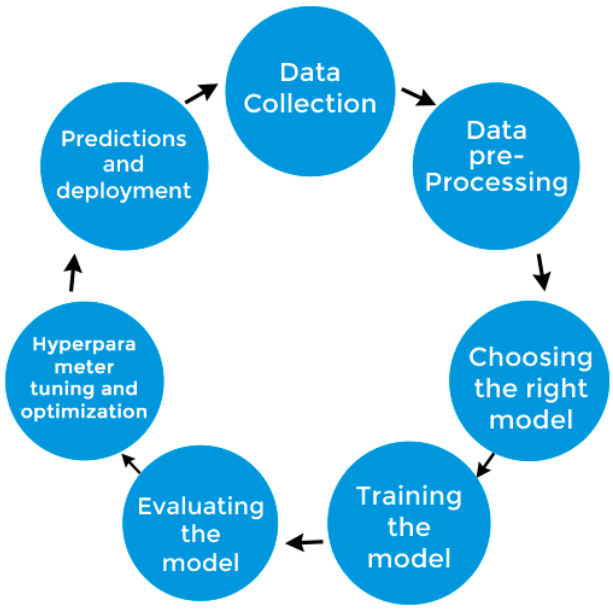
Machine Learning (ML) is a branch of Artificial Intelligence (AI) that works on algorithm developments and statistical models that allow computers to learn from data and make predictions or decisions without being explicitly programmed.

How does Machine Learning Work?

Machine Learning process includes Project Setup, Data Preparation, Modeling and Deployment. The following figure demonstrates the common working process of Machine Learning. It follows some set of steps to do the task; a sequential process of its workflow is as follows:



The following are the stages (detailed sequential process) of Machine Learning:



Data Collection: Data collection is an initial step in the process of machine learning. In this stage, it collects data from the different sources such as databases, text files, pictures, sound files, or web scraping. This process organizes the data in an appropriate format, such as a CSV file or database, and makes sure that they are useful for solving your problem.

Data Pre-processing: It is a key step in the process of machine learning, which involves deleting duplicate data, fixing errors, managing missing data either by eliminating or filling it in, and adjusting and formatting the data.

Choosing the Right Model: The next step is to select a machine learning model; once data is prepared, then we apply it to ML models like linear regression, decision trees, and neural networks

that may be selected to implement. This selection depends on many factors, such as the kind of data and your problem, the size and type of data, the complexity, and the computational resources.

Training the Model: This step includes training the model from the data so it can make better predictions.

Evaluating the model: When module is trained, the model has to be tested on new data that they haven't been able to see during training.

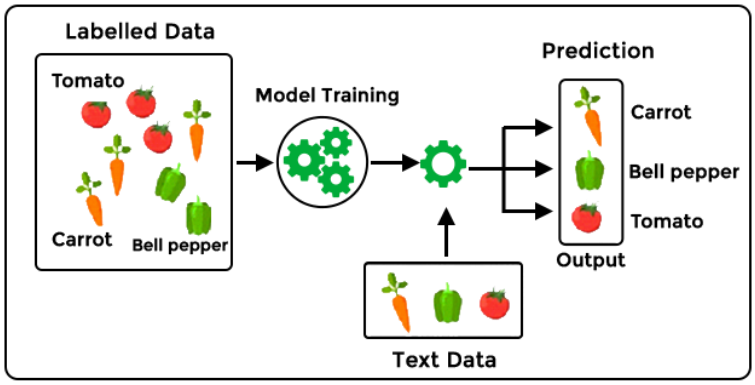
Hyperparameter Tuning and Optimization: After evaluating the model, you may need to adjust its hyperparameters to make it more efficient. You should try different combinations of parameters and cross-validation to ensure that the model performs well on different data sets.

Predictions and Deployment: When the model has been programmed and optimized, it will be ready to estimate new data. This is done by adding new data to the model and using its output for decision-making or other analysis. The deployment includes its integration into a production environment to make it capable of processing real-world data.

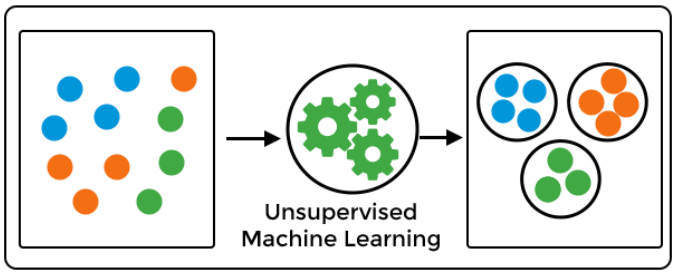
Types of Machine Learning

Machine learning models fall into the following categories:

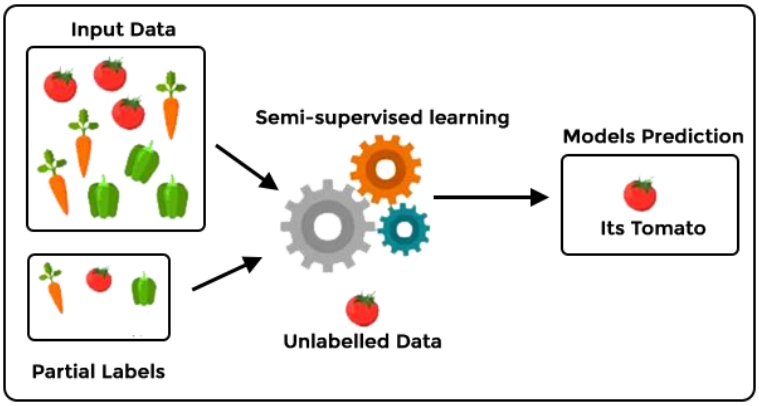
1. Supervised Machine Learning: It is a type of machine learning that trains the model using labeled datasets to predict outcomes.



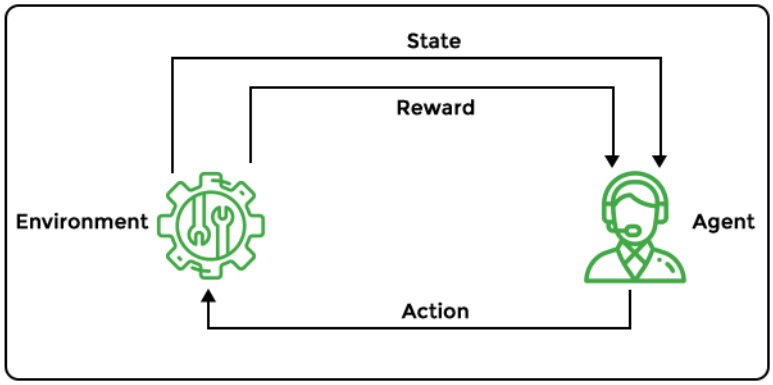
2. Unsupervised Machine Learning: It is a type of machine learning that learns patterns and structures within the data without human supervision.



3. Semi-supervised Learning: It is a type of machine learning that is neither fully supervised nor fully unsupervised. The semi-supervised learning algorithms basically fall between supervised and unsupervised learning methods.



4. Reinforcement Machine Learning: It is a type of machine learning model that is similar to supervised learning but does not use sample data to train the algorithm. This model learns by trial and error.



Common Machine Learning Algorithms

i. **Neural Networks**: It works like the human brain with many connected nodes. They help to find patterns and are used in language processing, image and speech recognition, and creating images.
ii. **Linear Regression**: It predicts numbers based on past data. For example, it helps estimate house prices in an area.
iii. **Logistic Regression**: It predicts like "yes/no" answers and it is useful for spam detection and quality control.
iv. **Clustering**: It is used to group similar data without instructions and it helps to find patterns that humans might miss.
v. **Decision Trees**: They help to classify data and predict numbers using a tree-like structure. They are easy to check and understand.
vi. **Random forests**: They combine multiple decision trees to improve predictions.

Importance of Machine Learning

Machine Learning is important in automation, extracting insights from data, and decision-making processes. It has its significance due to the following reasons:

i. **Data Processing**: Machine learning is useful to analyze large data from social media, sensors, and other sources and help to reveal patterns and insights to improve decision-making.
ii. **Data-Driven Insights**: Machine learning algorithms find trends and connections in big data that humans might miss, which helps to take better decisions and predictions.
iii. **Automation**: Machine learning automates the repetitive tasks, reducing errors and saving time.

iv. **Personalization**: Machine learning is useful to analyze the user preferences to provide personalized recommendations in e-commerce, social media, and streaming services. It helps in many manners, such as to improve user engagement, etc.
v. **Predictive Analytics**: Machine learning models use past data to predict future outcomes, which may help for sales forecasts, risk management, and demand planning.
vi. **Pattern Recognition**: Machine learning is useful in pattern recognition during image processing, speech recognition, and natural language processing.
vii. **Finance**: Machine learning is used in credit scoring, fraud detection, and algorithmic trading.
viii. **Retail**: Machine learning helps to enhance the recommendation systems, supply chain management, and customer service.
ix. **Fraud Detection & Cybersecurity**: Machine learning detects the fraudulent transactions and security threats in real time.
x. **Continuous Improvement**: Machine learning models update regularly with new data, which allows them to adapt and improve over time.

Applications of Machine Learning

Machine learning is used in various fields. Some of the most common applications include:

i. **Speech Recognition**: Machine learning is used to convert spoken language into text using natural language processing (NLP). It is used in voice assistants like Siri, voice search, and text accessibility features on mobile devices.
ii. **Customer Service**: There are several chatbots that are useful for reducing human interaction and providing better support on websites and social media, handling FAQs, giving recommendations, and assisting in e-commerce. For example, virtual agents, Facebook Messenger bots, and voice assistants.
iii. **Computer Vision**: It helps computers in analyzing the images and videos to take action. It is used in social media for photo tagging, in healthcare for medical imaging, and in self-driving cars for navigation.
iv. **Recommendation Engines**: ML recommendation engines suggest products, movies, or content based on user behavior. Online retailers use them to improve shopping experiences.
v. **Robotic Process Automation (RPA)**: RPA uses AI to automate repetitive tasks and reduce manual work.
vi. **Automated Stock Trading**: AI-driven trading platforms make rapid trades to optimize stock portfolios without human intervention.
vii. **Fraud Detection**: Machine learning identifies suspicious financial transactions, which help banks to detect fraud and prevent unauthorized activities.

Prerequisites to Learn Machine Learning

You should have a basic understanding of the technical aspects of Machine Learning. Learners should be familiar with data, information, and its basics. Knowledge of Data, information, structured data, unstructured data, semi-structured data, data processing, and Artificial Intelligence basics; Proficiency in labeled/unlabeled data, feature extraction from data, and their application in ML to solve common problems is a must.

7. ROBOTICS

Robots are nothing but machines that mostly help us with work, not only can we dedicate the role of robots to work but also to education and entertainment. Apart from a fun build the process of building this would give you hands-on experience in many areas of physics and electronics along with some

mechanical concepts.

What is Robotics?

Robotics is a branch of AI, which is composed of Electrical Engineering, Mechanical Engineering, and Computer Science for designing, construction, and application of robots.

Aspects of Robotics

i. The robots have mechanical construction, form, or shape designed to accomplish a particular task.
ii. They have electrical components which power and control the machinery.
iii. They contain some level of computer program that determines what, when and how a robot does something.

Difference in Robot System and Other AI Program

Here is the difference between the two:

AI Programs	Robots
They usually operate in computer-stimulated Worlds.	They operate in real physical world.
The input to an AI program is in symbols and rules.	Inputs to robots is analog signal in the form of speech waveform or images.
They need general purpose computers to operate on.	They need special hardware with sensors and effectors.

Robot Locomotion

Locomotion is the mechanism that makes a robot capable of moving in its environment. There are various types of locomotion:

i. Legged
ii. Wheeled
iii. Combination of Legged and Wheeled Locomotion
iv. Tracked slip/skid

Legged Locomotion

i. This type of locomotion consumes more power while demonstrating walk, jump, trot, hop, climb up or down, etc.
ii. It requires a greater number of motors to accomplish a movement. It is suited for rough as well as smooth terrain where irregular or too smooth surface makes it consume more power for a wheeled locomotion. It is little difficult to implement because of stability issues.
iii. It comes with the variety of one, two, four, and six legs. If a robot has multiple legs, then leg coordination is necessary for locomotion.

The total number of possible **gaits** (a periodic sequence of lift and release events for each of the total legs) a robot can travel depends upon the number of its legs. If a robot has *k* legs, then the number of possible events N = (2k-1)!.

In case of a two-legged robot (k=2), the number of possible events is N = (2k-1)! = (2*2-1)! = 3! = 6. Hence there are six possible different events:

i. Lifting the Left leg
ii. Releasing the Left leg
iii. Lifting the Right leg
iv. Releasing the Right leg
v. Lifting both the legs together
vi. Releasing both the legs together.

In case of k=6 legs, there are 39916800 possible events. Hence the complexity of robots is

directly proportional to the number of legs.

Wheeled Locomotion

It requires fewer number of motors to accomplish a movement. It is little easy to implement as there are less stability issues in case of more number of wheels. It is power efficient as compared to legged locomotion.

i. **Standard wheel:** Rotates around the wheel axle and around the contact
ii. **Castor wheel:** Rotates around the wheel axle and the offset steering joint
iii. **Swedish 45° and Swedish 90° wheels:** Omni-wheel, rotates around the contact point, around the wheel axle, and around the rollers.
iv. **Ball or spherical wheel:** Omnidirectional wheel, technically difficult to implement.

Slip/Skid Locomotion

In this type, the vehicles use tracks as in a tank. The robot is steered by moving the tracks with different speeds in the same or opposite direction. It offers stability because of large contact area of track and ground.

Components of a Robot

Robots are constructed with the following:

i. **Power Supply:** The robots are powered by batteries, solar power, hydraulic, or pneumatic power sources.
ii. **Actuators:** They convert energy into movement.
iii. **Electric motors (AC/DC):** They are required for rotational movement.
iv. **Pneumatic Air Muscles:** They contract almost 40% when air is sucked in them.
v. **Muscle Wires:** They contract by 5% when electric current is passed through them.
vi. **Piezo Motors and Ultrasonic Motors:** Best for industrial robots.
vii. **Sensors:** They provide knowledge of real time information on the task environment. Robots are equipped with vision sensors to be to compute the depth in the environment. A tactile sensor imitates the mechanical properties of touch receptors of human fingertips.

Applications of Robotics

The robotics has been instrumental in the various domains such as:

i. **Industries:** Robots are used for handling material, cutting, welding, color coating, drilling, polishing, etc.
ii. **Military:** Autonomous robots can reach inaccessible and hazardous zones during war. A robot named *Daksh*, developed by Defense Research and Development Organization (DRDO), is in function to destroy life-threatening objects safely.
iii. **Medicine:** The robots are capable of carrying out hundreds of clinical tests simultaneously, rehabilitating permanently disabled people, and performing complex surgeries such as brain tumors.
iv. **Exploration:** The robot rock climbers used for space exploration, underwater drones used for ocean exploration are to name a few.
v. **Entertainment:** Disney's engineers have created hundreds of robots for movie making.

Building Your First Robot: Step-by-Step

Building a robot can be a rewarding and educational experience, but it can also be quite complex depending on the type of robot you want to create. Below, is a general step-by-step tutorial for building a simple wheeled robot using off-the-shelf components. Keep in mind that this is a basic guide, and the specifics might vary based on your chosen components and goals.

Step 1: Define Your Robot's Purpose and Design

1. Decide the Purpose: Determine the purpose of your robot. Is it going to be a line-following robot, a remote-controlled vehicle, or something else entirely?
2. Design and Plan: Sketch out a rough design of your robot. Consider the size, shape, number of wheels, sensors, and any other features you want to include.

Step 2: Gather Components and Materials

1. Microcontroller: Choose a microcontroller or microprocessor board to serve as the brain of your robot. Popular choices include Arduino, Raspberry Pi, or microcontrollers specifically designed for robotics.
2. Chassis and Wheels: Select a chassis for your robot, which is the structural frame. Choose wheels that fit your design and intended movement.
3. Motor Drivers: Depending on the motors you choose, you might need motor driver circuits to control their speed and direction.
4. Power Source: Decide on a power source, such as batteries, and make sure it provides the appropriate voltage for your components.
5. Sensors: If your robot requires sensors (like distance sensors, line-following sensors, etc.), choose and gather them.
6. Connectors and Wires: Collect the necessary wires, connectors, and soldering tools.

Step 3: Assemble the Hardware

1. Assemble Chassis: Put together the chassis following the manufacturer's instructions.
2. Mount Motors: Attach the motors to the chassis and connect them to the motor drivers if needed.

3. Attach Wheels: Fix the wheels on the motors.
4. Mount Electronics: Place the microcontroller, motor drivers, sensors, and other components onto the chassis. Secure them properly.
5. Wire Connections: Carefully connect all the components using appropriate wires and connectors. Double-check your connections.

Step 4: Program the Robot

1. Choose Programming Language: Decide on a programming language. For microcontrollers like Arduino, C/C++ is common. For Raspberry Pi, Python is popular.
2. Write or Upload Code: Write the code to control your robot's movements and interactions with sensors. If using Arduino, you'll upload the code to the microcontroller.

Step 5: Test and Debug

1. Power Up: Power on your robot and make sure everything is working as expected.
2. Test Movements: Test the robot's movement using your programmed commands. Make sure it can move forward, backward, turn, and stop correctly.
3. Sensor Testing: If your robot has sensors, test their readings and make necessary adjustments in your code.

Step 6: Iterate and Improve

1. Refine Design: Based on your testing, make any necessary adjustments to the design, hardware, or software.
2. Enhance Functionality: If your robot performs basic tasks well, consider adding more features or improving its capabilities.

Step 7: Document and Showcase

1. Documentation: Keep notes, schematics, and code snippets organized. This will help you troubleshoot issues and replicate your work in the future.
2. Showcase: Share your creation with friends, online communities, or in educational settings.

This provides a basic outline for building a simple robot. Depending on your expertise and the complexity of your project, the steps and details might vary. Remember, always prioritize safety while working with electronic components and tools.

Personal Study: Computer Vision

8. REPRESENTATION AND PROBLEM SOLVING IN LISP/CLIPS.

LISP stands for LISt Programming. John McCarthy invented LISP in 1958, shortly after the development of FORTRAN. It was first implemented by Steve Russell on an IBM 704 computer. It is particularly suitable for Artificial Intelligence programs, as it processes symbolic information efficiently. Common LISP originated during the decade of 1980 to 1990, in an attempt to unify the work of several implementation groups, as a successor of Maclisp like ZetaLisp and New Implementation of LISP (NIL) etc. It serves as a common language, which can be easily extended for specific implementation. Programs written in Common LISP do not depend on machinespecific characteristics, such as word length etc.

LISP expressions are called symbolic expressions or s-expressions. The s-expressions are composed of three valid objects, atoms, lists and strings. Any s-expression is a valid program. LISP programs run either on an **interpreter** or as **compiled code.** The interpreter checks the source code in a repeated loop, which is also called the read-evaluate-print loop (REPL). It reads the program code, evaluates it, and prints the values returned by the program.

Features of Common LISP

i. It is machine-independent

ii. It uses iterative design methodology
iii. It has easy extensibility
iv. It allows to update the programs dynamically
v. It provides high level debugging.
vi. It provides advanced object-oriented programming.
vii. It provides convenient macro system.
viii. It provides wide-ranging data types like, objects, structures, lists, vectors, adjustable arrays, hash-tables, and symbols.
ix. It is expression-based.
x. It provides an object-oriented condition system.
xi. It provides complete I/O library.
xii. It provides extensive control structures.

A simple Program
Write an s-expression to find the sum of three numbers 7, 9 and 11. To do this we can type at the interpreter prompt.

```
; execuate sum of three numbers
(+ 7 9 11)
```

LISP return the result: 27
If you would like to run the same program as a compiled code, then create a LISP source code file named myprog.lisp and type the following code in it.
main.lisp
; print sum of three numbers
(write (+ 7 9 11))
Output
When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is: 27

LISP Uses Prefix Notation
You might have noted that LISP uses **prefix notation.** In the above program the + symbol works as the function name for the process of summation of the numbers. In prefix notation, operators are written before their operands. Example 1:
(a * (b + c)) / d
will be written as:
; prefix mode operation
(/ (* a (+ b c)) d)
Example 2: write code for converting Fahrenheit temp of 60° F to the centigrade scale.
The mathematical expression for this conversion will be:
(60 * 9 / 5) + 32
Create a source code file named main.lisp and type the following code in it.
main.lisp
; evaluate and print arithmetic expression
(write(+ (* (/ 9 5) 60) 32))
Output
When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is: 140
The 'Hello World' Program

Learning a new programming language doesn't really take off until you learn how to greet the entire world in that language, right!
So, please create new source code file named main.lisp and type the following code in it.
main.lisp

```
; print Hello World
(write-line "Hello World")
; print the statement
(write-line "I am at 'AI Class'! Learning LISP")
```

Output
When you click the Execute button, or type Ctrl+E, LISP executes it immediately and the result returned is:

```
Hello World
I am at 'AI Class'! Learning LISP
```

LISP - Basic Syntax
Basic Building Blocks in LISP
LISP programs are made up of three basic building blocks −

i. atom
ii. list
iii. string

An atom is a number or string of contiguous characters. It includes numbers and special characters.
Following are examples of some valid atoms:

```
hello-from-AI Class
name
123008907
*hello*
Block#221
abc123
```

A list is a sequence of atoms and/or other lists enclosed in parentheses.
Following are examples of some valid lists −

```
( i am a list)
(a ( a b c) d e fgh)
(father tom (Susan Bill Joe))
(sun mon tue wed thur fri sat)
( )
```

A string is a group of characters enclosed in double quotation marks.
Following are examples of some valid strings:

```
" I am a string"
"a ba c d efg #$%^&!"
"Please enter the following details :"
"Hello from 'Tutorials Point'! "
```

Adding Comments
The semicolon symbol (;) is used for indicating a comment line.
For Example,
Open Compiler

```
(write-line "Hello World"); greet the world
;tell them your whereabouts
```

The Lisp Executer
The source code written in source file is the human readable source for your program. It needs to be "executed", to turn into machine language so that your CPU can actually execute the program as per instructions given.

This Lisp programming language will be used to execute your source code into final executable program. I assume you have basic knowledge about a programming language.
CLISP is the GNU Common LISP multi-architechtural compiler used for setting up LISP in Windows. The windows version emulates a unix environment using MingW under windows. The installer takes care of this and automatically adds clisp to the windows PATH variable.
You can get the latest CLISP for Windows from its home page CLISP - an ANSI Common Lisp from SourceForge - https://sourceforge.net/projects/clisp/files/clisp/2.49/clisp-2.49-win32-mingw-big.zip/download
Extract the files to any folder of your choice and add path to clisp.exe in PATH variable.
How to use CLISP
Once, clisp is in PATH, you can simply open a new Command Prompt window and type clisp to bring up the compiler.
To run a *.lisp or *.lsp file, simply use −
clisp hello.lisp
Considering following content in hello.lisp
hello.lisp
```
;print the Hello World
(write-line "Hello World")
;print the statment
(write-line "I am at 'Tutorials Point'! Learning LISP")
```
Output
You will see the following output on running the LISP compiler
```
Hello World
I am at 'Tutorials Point'! Learning LISP
```
CLISP is a specific **implementation of Common Lisp**.
Here's a breakdown:
What is CLISP?

i. **CLISP** = *GNU CLISP*
ii. It is an **implementation of Common Lisp** that is:
    a. **Portable**: runs on many operating systems (Windows, Linux, macOS, BSD, etc.).
    b. **Open-source**: licensed under GNU GPL.
    c. **Efficient**: comes with an interpreter and compiler for Common Lisp.

Features of CLISP

1. Interactive Environment (REPL)

CLISP provides a Read-Eval-Print Loop (REPL) for experimenting with code.

2. clisp
3. > (+ 2 3)
4. 5
5. Compiler & Interpreter

You can either interpret code interactively or compile it to bytecode.

6. Built-in Libraries

Provides modules for numerical computing, string handling, object-oriented programming (CLOS – Common Lisp Object System), and file I/O.

7. Foreign Language Interface
    - Can interface with C, making it possible to call external libraries.
8. **Unicode Support**

Handles multiple character sets, useful for international applications.

Why is CLISP Important?

- It's **widely used in teaching** symbolic programming and AI, since it's beginner-friendly and well-documented.
- Provides **cross-platform compatibility**.
- Helps researchers and students **experiment with symbolic AI systems** without worrying about low-level implementation.

Example in CLISP

```
(defun factorial (n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
(print (factorial 5)); Output: 120
```

CLISP is not a language itself, but one of the most popular *implementations* of the Common Lisp programming language.

COURSE WORK

Student Project Plan

Step 1: Grouping

- Divide class into groups of **5–8 students**.
- Each group selects 1 project from the list of 15 simple AI projects.

Step 2: Implementation Expectations

Each group should:

1. **Describe the problem** (short background, why it matters).
2. **Explain the method** (logic rules, ML algorithm, or search strategy).
3. **Show the implementation** (Python/CLIPS/LISP code).
4. **Demonstrate the system** (live demo or screenshots).
5. **Discuss limitations & future improvements** (to show critical thinking).

Step 3: Seminar Presentation

- **Time per group:** 15–20 minutes.
- **Format:** Slides + demo.
- **Assessment:**
    - Clarity of explanation (30%)
    - Working system/demo (40%)
    - Teamwork & presentation (20%)
    - Originality/extra effort (10%)

Pool of 15 Projects: Each group has freedom to choose based on their interest and coding comfort level, based on FIFO.

Project Options

1. Truth-Table Generator
    - Objective: Generate truth tables for logical expressions.
    - Tools: Python.
    - Output: Table showing truth values for given propositions.
2. Family Tree in LISP/CLIPS
    - Objective: Represent and query family relationships.
    - Tools: LISP / CLIPS.
    - Output: Query answers (e.g., “Who is the ancestor of X?”).
3. Rule-Based Chatbot
    - Objective: Answer user queries with keyword-based responses.
    - Tools: Python.
    - Output: Interactive chatbot (console or GUI).
4. Maze Solver (BFS/DFS)
    - Objective: Solve a maze using search algorithms.
    - Tools: Python.
    - Output: Path found by BFS/DFS (text or visual).
5. Tic-Tac-Toe AI (Minimax)
    - Objective: Build a Tic-Tac-Toe game where computer never loses.
    - Tools: Python.

- Output: Playable Tic-Tac-Toe with AI opponent.

6. Calculator Chatbot
   - Objective: Perform basic math operations via chatbot.
   - Tools: Python.
   - Output: Bot that answers math queries.
7. Animal Guessing Expert System
   - Objective: Guess an animal by asking yes/no questions.
   - Tools: Python / CLIPS.
   - Output: Animal identified through rules.
8. Traffic Light Simulation
   - Objective: Simulate traffic lights and car movement with rules.
   - Tools: Python (console or Pygame).
   - Output: Light changes and simple car simulation.
9. Spam Message Classifier
   - Objective: Detect spam vs. non-spam messages.
   - Tools: Python, Scikit-learn.
   - Output: Classifier with accuracy score.
10. Student Performance Prediction
    - Objective: Predict if a student passes/fails based on features.
    - Tools: Scikit-learn.
    - Output: Prediction results + accuracy.
11. Weather Predictor
    - Objective: Predict sunny/rainy/cloudy from dataset.
    - Tools: Scikit-learn.
    - Output: Predicted weather condition.
12. Sentiment Analysis (Movie Reviews)
    - Objective: Classify reviews as positive or negative.
    - Tools: Python, Scikit-learn, NLTK.
    - Output: Sentiment label for input text.
13. House Price Prediction
    - Objective: Predict house prices using regression.
    - Tools: Scikit-learn.
    - Output: Predicted house price from features.
14. Number Recognition (k-NN)
    - Objective: Recognize handwritten digits (0–9).
    - Tools: Scikit-learn.
    - Output: Digit classification results.
15. Simple Plagiarism Checker
    - Objective: Compare two texts and measure similarity.
    - Tools: Scikit-learn (TF-IDF + cosine similarity).
    - Output: Similarity percentage.

**All can be done in Python notebooks, except the family tree (LISP/CLIPS) which gives them symbolic AI exposure.*