DEPARTMENT OF COMPUTER SCIENCE AND ENGG.
INTELLIGENT SYSTEM
SEM-6TH
SECTION-A (NOTES)

HISTORY OF AI:

The history of artificial intelligence (AI) began in antiquity, with myths, stories and rumors of artificial beings endowed with intelligence or consciousness by master craftsmen; as Pamela McCorduck writes, AI began with "an ancient wish to forge the gods."

The seeds of modern AI were planted by classical philosophers who attempted to describe the process of human thinking as the mechanical manipulation of symbols. This work culminated in the invention of the programmable digital computer in the 1940s, a machine based on the abstract essence of mathematical reasoning. This device and the ideas behind it inspired a handful of scientists to begin seriously discussing the possibility of building an electronic brain

What is Artificial Intelligence?

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.

Goals of AI

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

What Contributes to AI?

Artificial intelligence is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering. A major thrust of AI

is in the development of computer functions associated with human intelligence, such as reasoning, learning, and problem solving.

What is AI Technique?

In the real world, the knowledge has some unwelcomed properties −

- Its volume is huge, next to unimaginable.
- It is not well-organized or well-formatted.
- It keeps changing constantly.

AI Technique is a manner to organize and use the knowledge efficiently in such a way that −

- It should be perceivable by the people who provide it.
- It should be easily modifiable to correct errors.
- It should be useful in many situations though it is incomplete or inaccurate.

AI techniques elevate the speed of execution of the complex program it is equipped with.

Applications of AI

AI has been dominant in various fields such as −

- **Gaming** − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing** − It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** − These systems understand, interpret, and comprehend visual input on the computer. For example,
    - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
    - Doctors use clinical expert system to diagnose the patient.
    - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
- **Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
- **Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

- **Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment

What is Intelligence?

The ability of a system to calculate, reason, perceive relationships and analogies, learn from experience, store and retrieve information from memory, solve problems, comprehend complex ideas, use natural language fluently, classify, generalize, and adapt new situations.

Types of Intelligence

As described by Howard Gardner, an American developmental psychologist, the Intelligence comes in multifold −

Intelligence	Description	Example
Linguistic intelligence	The ability to speak, recognize, and use mechanisms of phonology (speech sounds), syntax (grammar), and semantics (meaning).	Narrators, Orators
Musical intelligence	The ability to create, communicate with, and understand meanings made of sound, understanding of pitch, rhythm.	Musicians, Singers, Composers
Logical-mathematical intelligence	The ability of use and understand relationships in the absence of action or objects. Understanding complex and abstract ideas.	Mathematicians, Scientists
Spatial intelligence	The ability to perceive visual or spatial information, change it, and re-create visual images without reference to the objects, construct 3D images, and to move and rotate them.	Map readers, Astronauts, Physicists
Bodily-Kinesthetic intelligence	The ability to use complete or part of the body to solve problems or fashion products, control over fine and coarse motor skills, and manipulate the objects.	Players, Dancers
Intra-personal intelligence	The ability to distinguish among one's own feelings, intentions, and motivations.	Gautam Buddhha
Interpersonal intelligence	The ability to recognize and make distinctions among other people's	Mass Communicators,

feelings, beliefs, and intentions. Interviewers

You can say a machine or a system is artificially intelligent when it is equipped with at least one and at most all intelligences in it.

What is Intelligence Composed of?

The intelligence is intangible. It is composed of −

- Reasoning
- Learning
- Problem Solving
- Perception
- Linguistic Intelligence

- Learning − It is the activity of gaining knowledge or skill by studying, practising, being taught, or experiencing something. Learning enhances the awareness of the subjects of the study.

  The ability of learning is possessed by humans, some animals, and AI-enabled systems. Learning is categorized as −

  - Auditory Learning − It is learning by listening and hearing. For example, students listening to recorded audio lectures.
  - Episodic Learning − To learn by remembering sequences of events that one has witnessed or experienced. This is linear and orderly.
  - Motor Learning − It is learning by precise movement of muscles. For example, picking objects, Writing, etc.
  - Observational Learning − To learn by watching and imitating others. For example, child tries to learn by mimicking her parent.
  - Perceptual Learning − It is learning to recognize stimuli that one has seen before. For example, identifying and classifying objects and situations.
  - Relational Learning − It involves learning to differentiate among various stimuli on the basis of relational properties, rather than absolute properties. For Example, Adding 'little less' salt at the time of cooking potatoes that came up salty last time, when cooked with adding say a tablespoon of salt.
  - Spatial Learning − It is learning through visual stimuli such as images, colors, maps, etc. For Example, A person can create roadmap in mind before actually following the road.
  - Stimulus-Response Learning − It is learning to perform a particular behavior when a certain stimulus is present. For example, a dog raises its ear on hearing doorbell.
- **Problem Solving** − It is the process in which one perceives and tries to arrive at a desired solution from a present situation by taking some path, which is blocked by known or unknown hurdles.

Problem solving also includes decision making, which is the process of selecting the best suitable alternative out of multiple alternatives to reach the desired goal are available.

- **Perception** − It is the process of acquiring, interpreting, selecting, and organizing sensory information.

  Perception presumes sensing. In humans, perception is aided by sensory organs. In the domain of AI, perception mechanism puts the data acquired by the sensors together in a meaningful manner.

Properties of Environment

The environment has multifold properties −

- **Discrete / Continuous** − If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- **Observable / Partially Observable** − If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- **Static / Dynamic** − If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.
- **Single agent / Multiple agents** − The environment may contain other agents which may be of the same or different kind as that of the agent.
- **Accessible / Inaccessible** − If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.
- **Deterministic / Non-deterministic** − If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.
- **Episodic / Non-episodic** − In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead

Search Terminology

- Problem Space − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- Problem Instance − It is Initial state + Goal state.
- Problem Space Graph − It represents problem state. States are shown by nodes and operators are shown by edges.
- Depth of a problem − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
- Space Complexity − The maximum number of nodes that are stored in memory.

- Time Complexity − The maximum number of nodes that are created.
- Admissibility − A property of an algorithm to always find an optimal solution.
- Branching Factor − The average number of child nodes in the problem space graph.
- Depth − Length of the shortest path from initial state to goal state.

Brute-Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements −

- State description
- A set of valid operators
- Initial state
- Goal state description

Breadth-First Search

It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = b^d.

The total no of nodes created in worst case is $b + b^2 + b^3 + \ldots + b^d$.

Disadvantage − Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.

Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor *b* and depth as *m*, the storage space is *bm*.

Disadvantage − This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is *d*, and if chosen cut-off is lesser

than *d*, then this algorithm may fail. If chosen cut-off is more than *d*, then execution time increases. Its complexity depends on the number of paths. It cannot check duplicate nodes.

Heuristic Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

PROBLEM REDUCTION (AND - OR graphs - AO * Algorithm)

When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, AND-OR graphs or AND - OR trees are used for representing the solution. The decomposition of the problem or problem reduction generates AND arcs. One AND are may point to any number of successor nodes. All these must be solved so that the arc will rise to many arcs, indicating several possible solutions. Hence the graph is known as AND - OR instead of AND. Figure shows an AND - OR graph.

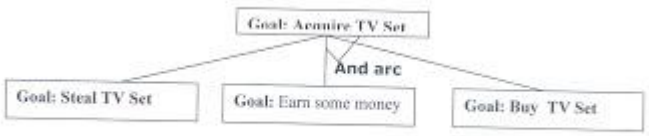

Figure shows AND - Or graph - an example.

An algorithm to find a solution in an AND - OR graph must handle AND area appropriately. A* algorithm can not search AND - OR graphs efficiently. This can be understand from the give figure.

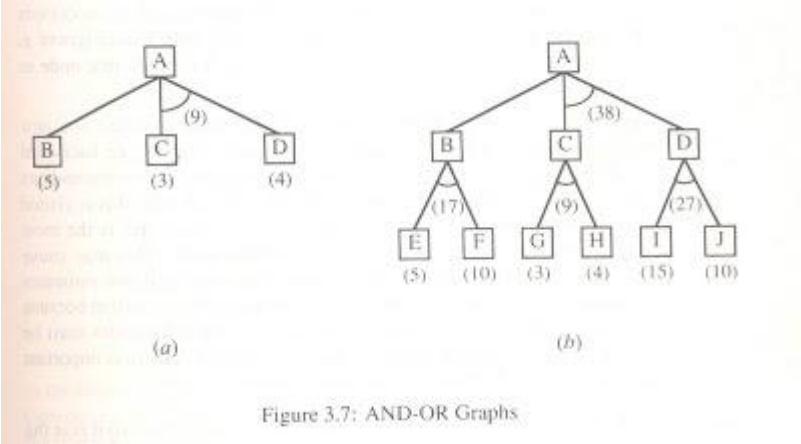

Figure 3.7: AND-OR Graphs

FIGURE : AND - OR graph

1. traverse the graph starting at the initial node and following the current best path, and accumulate the set of nodes that are on the path and have not yet been expanded.

2. Pick one of these unexpanded nodes and expand it. Add its successors to the graph and computer f ' (cost of the remaining distance) for each of them.

3. Change the f ' estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which of the current best path.

The propagation of revised cost estimation backward is in the tree is not necessary in A* algorithm. This is because in AO* algorithm expanded nodes are re-examined so that the current best path can be selected. The working of AO* algorithm is illustrated in figure as follows:

The algorithm for performing a heuristic search of an AND - OR graph is given below. Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G. G represents the part of the search graph generated so far. Each node in G points down to its immediate successors and up to its immediate predecessors, and also has with it the value of h' cost of a path from itself to a set of solution nodes. The cost of getting from the start nodes to the current node "g" is not stored as in the A* algorithm. This is because it is not possible to compute a single such value since there may be many paths to the same state. In AO* algorithm serves as the estimate of goodness of a node. Also a there should value called FUTILITY is used. The estimated cost of a solution is greater than FUTILITY then the search is abandoned as too expansive to be practical.

AO* ALGORITHM:

AO* Search Procedure.

1. Place the start node on open.

2. Using the search tree, compute the most promising solution tree TP .

3. Select node n that is both on open and a part of tp, remove n from open and place it no closed.

4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.

5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open ,with unsolvable ancestors.

6. Otherwise, expand node n generating all of its successor compute the cost of for each newly generated node and place all such nodes on open.

7. Go back to step(2)

Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

function Hill-Climbing (problem), returns a state that is a local maximum.

```
inputs: problem, a problem
local variables: current, a node
                 neighbor, a node
current <-Make_Node(Initial-State[problem])
loop
  do neighbor <- a highest_valued successor of current
    if Value[neighbor] ≤ Value[current] then
    return State[current]
    current <- neighbor

end
```

Disadvantage − This algorithm is neither complete, nor optimal.

Game tree

In game theory, a game tree is a directed graph whose nodes are positions in a game and whose edges are moves. The complete game tree for a game is the game tree starting at the initial position and containing all possible moves from each position; the complete tree is the same tree as that obtained from the extensive-form game representation.

The first two plies of the game tree for tic-tac-toe.

The diagram shows the first two levels, or *plies*, in the game tree for tic-tac-toe. The rotations and reflections of positions are equivalent, so the first player has three choices of move: in the center, at the edge, or in the corner. The second player has two choices for the reply if the first player played in the center, otherwise five choices. And so on.

The number of leaf nodes in the complete game tree is the number of possible different ways the game can be played. For example, the game tree for tic-tac-toe has 255,168 leaf nodes.

Game trees are important in artificial intelligence because one way to pick the best move in a game is to search the game tree using the minimax algorithm or its variants. The game tree for tic-tac-toe is easily searchable, but the complete game trees for larger games like chess are much too large to search. Instead, a chess-playing program searches a partial game tree: typically as many plies from the current position as it can search in the time available. Except for the case of "pathological" game trees [1] (which seem to be quite rare in practice), increasing the search depth (i.e., the number of plies searched) generally improves the chance of picking the best move.

Two-person games can also be represented as and-or trees. For the first player to win a game, there must exist a winning move for all moves of the second player. This is represented in the and-or tree by using disjunction to represent the first player's alternative moves and using conjunction to represent all of the second player's moves.

Alpha–beta

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games (Tic-tac-toe, Chess, Go, etc.). It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decisionThe benefit of alpha–beta pruning lies in the fact that branches of the search tree can be eliminated. This way, the search time can be limited to the 'more promising' subtree, and a deeper search can be performed in the same time. Like its predecessor, it belongs to the branch and bound class of algorithms. The optimization reduces the effective depth to slightly more than half that of simple minimax if the nodes are evaluated in an optimal or near optimal order (best choice for side on move ordered first at each node).

With an (average or constant) branching factor of *b*, and a search depth of *d* plies, the maximum number of leaf node positions evaluated (when the move ordering is pessimal) is $O(b*b*...*b) = O(b^d)$ – the same as a simple minimax search. If the move ordering for the search is optimal (meaning the best moves are always searched first), the number of leaf node positions evaluated is about $O(b*1*b*1*...*b)$ for odd depth and $O(b*1*b*1*...*1)$ for even

Normally during alpha–beta, the subtrees are temporarily dominated by either a first player advantage (when many first player moves are good, and at each search depth the first move checked by the first player is adequate, but all second player responses are required to try to find a refutation), or vice versa. This advantage can switch sides many times during the search if the move ordering is incorrect, each time leading to inefficiency. As the number of positions searched decreases exponentially each move nearer the current position, it is worth spending considerable effort on sorting early moves. An improved sort at any depth will exponentially reduce the total number of positions searched, but sorting all positions at depths near the root node is relatively cheap as there are so few of them. In practice, the move ordering is often determined by the results of earlier, smaller searches, such as through iterative deepening.

The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta <= alpha). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.

Additionally, this algorithm can be trivially modified to return an entire principal variation in addition to the score. Some more aggressive algorithms such as MTD(f) do not easily permit such a modification.

Minimax Search Algorithm

The standard algorithm for two-player perfect-information games such as chess, checkers or othello is minimax search with heuristic static evaluation. The minimax search algorithm searches forward to a fixed depth in the game tree, limited by the amount of time available per move. At this search horizon, a heuristic function is applied to the frontier nodes. In this case, a heuristic evaluation is a function that takes a board position and returns a number that indicates how favourable that position is for one player relative to the other. For example, a very simple heuristic evaluator for chess would count the total number of pieces on the board for one player, appropriately weighted by their relative strength, and subtract the weighted sum of the opponent's places. Thus, large positive values would correspond to strange positions for one player called *MAX*, whereas large negative values would represent advantageous situation for the opponent called *MIN*.

Given the heuristic evaluations of the frontier nodes, minimax search algorithm recursively computes the values for the interior nodes in the tree according to the *maximum rule*. The value

of a node where it is MAX's turn to move is the maximum of the values of its children, while the value of the node where MIN is to move is the minimum of the values of its children. Thus at alternative levels of the tree, the maximum values of the children are backed up. This continues until the values of the immediate children of the current position are computed at which point one move to the child with the maximum or minimum value is made depending on whose turn it is to move.

A minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of his legal moves.

A possible allocation method consists in assigning a certain win for A as +1 and for B as −1. This leads to combinatorial game theory as developed by John Horton Conway. An alternative is using a rule that if the result of a move is an immediate win for A it is assigned positive infinity and, if it is an immediate win for B, negative infinity. The value to A of any other move is the minimum of the values resulting from each of B's possible replies. For this reason, A is called the *maximizing player* and B is called the *minimizing player*, hence the name *minimax algorithm*. The above algorithm will assign a value of positive or negative infinity to any position since the value of every position will be the value of some final winning or losing position. Often this is generally only possible at the very end of complicated games such as chess or go, since it is not computationally feasible to look ahead as far as the completion of the game, except towards the end, and instead positions are given finite values as estimates of the degree of belief that they will lead to a win for one player or another.

The algorithm can be thought of as exploring the nodes of a *game tree*. The *effective branching factor* of the tree is the average number of children of each node (i.e., the average number of legal moves in a position). The number of nodes to be explored usually increases exponentially with the number of plies (it is less than exponential if evaluating forced moves or repeated positions). The number of nodes to be explored for the analysis of a game is therefore approximately the branching factor raised to the power of the number of plies. It is therefore impractical to completely analyze games such as chess using the minimax algorithm.

The performance of the naïve minimax algorithm may be improved dramatically, without affecting the result, by the use of alpha-beta pruning. Other heuristic pruning methods can also be used, but not all of them are guaranteed to give the same result as the un-pruned search.

A minimax algorithm may be trivially modified to additionally return an entire Principal Variation along with a minimax score