AI Notes

▼ **TOC**

Introduction to AI

AI can be defined considering the thought process and reasoning, and behavior in the following manner:

1. **Thinking humanly**: It uses the cognitive approach. We are suppose to say that a given thinks like a human, we must have some way of determining how humans think. There can be three ways:

   a. Through introspection i.e. observing our thoughts.

   b. Through psychological experiments i.e. observing a person in action.

   c. Through brain imaging i.e. observing the brain in action.

   Once we have a precise theory of mind it becomes possible to express that theory as a computer program.

2. **Acting humanly**: It uses Turing test approach is used.

   Turing Test:

   A computer passes the test if a human invigilator after posing some written questions cannot tell whether the written responses come from a person or a computer.

   The computer would need to posses following capabilities:

   1. Natural Language Processing — to enable it to communicate successfully in English.

   2. Knowledge representation — to store what it knows or hears.

   3. Automated reasoning — to use the store information, to answer questions, and to draw new conclusions.

    4. Machine learning — to adapt to new circumstances, and to detect and extend patterns.
    5. Computer vision — to perceive objects.
    6. Robotics — to manipulate objects, and to move about.

3. **Thinking rationally**: Based on laws approach.
   The Greek philosopher Aristotle was one of the first to codify right thinking that is impossible to deny, reasoning process or syllogism. These laws of thought was suppose to govern operations of mind, hence their study initiated the field of logic.
4. **Acting rationally**: Uses rational agent approach.
   An agent is anything that acts (to-do) i.e. all computer programs do something however computer agents are expected to do more i.e. operate autonomously, perceive their environment, persist over a prolonged time, adapt to change, create and perceive goals.
   The rational agent is one that acts so as to achieve the best outcome or when there is uncertainty, get the best expected outcome.

Terminologies

Examples of agents

Human agent has eyes, ears, etc. as sensory organs, and to take actions (actuators) hands, legs, vocal track, etc.

Robotic agent might have camera, infrared range finders as sensors, and various motors as actuators.

Software agent can receive key strokes, file contents, network packets as sensory inputs, and can act in the environment by displaying on the screen, writing files, and sending network packets.

Agent terminology

Performance measure of agent: It is the criteria which determines how successful an agent is.

Behavior of an agent: It is an action that an agent performance after any given sequence of percepts.

Percept: It refers to agents perceptual (relating to the ability to interpret or become aware of something through senses) inputs at any given instant.

Percept sequence: It is the history that all agents has perceived till date.

Agent function: Mathematically speaking, an agents behavior is described by agent function that maps any given percept sequence to an action.

Agent program: It is a concrete implementation running within some physical system.

Rational agent

A rational agent is one that does the right thing. What is rational depends on:

1. The performance measure that defines the criteria of success.
2. The agent prior knowledge of the environment.
3. The action that an agent can perform.
4. The agents percept sequence till date.

A rational agent should be autonomous.

Task environment

Following grouped together is called task environment (PEAS):

1. Performance measure
2. Environment
3. Actuator
4. Sensors

Ex. Self driving cars.

Performance measure: safety, time, legal drive, comfort.

Environment: roads, other vehicles, road sign, pedestrians.

Actuator: Steering, accelerator, brake, horn, etc.

Sensor: Camera, GPS, speedometer, accelerometer, SONAR, odometer.

Ex. Vacuum cleaner.

Performance measure: power consumption, suction efficiency, dirt holding capacity.

Environment: Interiors, objects.

Actuators: Motor, wheels, suction opening.

Sensors: Proximity sensor.

Properties of Environment

1. **Fully versus partially observable**: If an agent sensor can sense or access the complete state of an environment at each point of time than it is fully observable environment otherwise it is partially observable. A fully observable environment as there is no need to maintain the internal state to keep track history of the world. An agent with no sensors in all environments is set to be in an environment which is called as unobservable.
2. **Deterministic versus stochastic**: If an agents current state and selected action can completely determine the next state of the environment than such environment is called as deterministic environment. A stochastic is random in nature and cannot be determined completely by an agent. In a deterministic, fully observable environment agent does not need to worry about uncertainty.
3. **Episodic versus sequential**: In an episodic environment there is a series of one shot actions, and only the current percept is required of the action however, in sequential an agent requires memory of past actions to determine the next best actions.
4. **Static versus dynamic**: If the environment can change itself than it is called as dynamic otherwise it is called as static. Static environments are easy to deal because an agent does not need to continue looking at the world while decided for an action however, for dynamic environment agents need to keep looking at the world at each action.
5. **Discrete versus continuous**: If in an environment there are finite number of percepts and actions that can be performed within it then it is called as discrete otherwise it is called as continuous environment.
6. **Single agent versus multi-agent**: If only one agent is involved in an environment and operating by itself then such a environment is called as single agent environment. However, if there are multiple operating than it is called as multi-agent environment. The agent design problems in the multi-agent environment are different from single agent environment.

Types of agent

1. Simple Reflex Agents:

a. This is a simple type of agent which works on the basis of current percept and not based on the rest of the percept history.

b. The agent function is based on condition action rule where the condition or the state is mapped to the action such that action is taken only when condition is true or else it is not.

c. If the environment associated with this agent is fully observable only then is the agent function successful, if it is partially observable the agent functions enters into infinite loop that can be escaped only on randomization of its actions.

d. The problem associated with this type include very limited intelligence, no knowledge of non-perceptual parts of the state inability to adapt to changes in the environment.

2. Model Based Agents:

   a. Utilizes condition action rule where it works by finding a rule that will allow the condition which is based on the current situation to be satisfied.

   b. It can handle partially observable environment by tracking the situation and using a particular model related to the world.

   c. It consists of two important factors which are model and internal state.

   d. Model provides knowledge and understanding of the process of occurrences of different things in the surrounding such that the current situation can be studied and a condition can be created. Actions are performed by the agent based on this model.

   e. Internal state uses the perceptual history to represent a current percept. The agent keeps a track of this internal state and is adjusted by each of the percepts. The current internal state is stored by the agent inside it to maintain a kind of structure that can describe the unseen world.

   f. The state of the agent can be updated by gaining information about how the world evolves and how the agents action affects the world.

3. Goal Based Agents:

   a. This type takes decisions on the basis of its goal or desirable situations so that it can select such an action that can achieve the required goal.

   b. It is an improvement over model based agent where information about goal is also included.

c. The aim is to reduce the distance between action and the goal so that the best possible way can be selected from multiple possibilities. Once the best way is found the decision is represented explicitly which makes the agent more flexible.

d. It carries out considerations of different situations called as searching and planning by considering long sequence of possible actions for confirming its ability to achieve the goal.

e. It can easily change its behavior if required.

4. Utility Based Agents:

   a. Utility agent have their end-users as their building blocks as is used when based action and decision needs to be taken from multiple alternatives.

   b. It is an improvement over goal based agent as it not only involves goals but also the way the goal can be achieved such that the goal can be achieved in a quicker, safer, cheaper way.

   c. The extra component of utility or method to achieve the goal provides a measure of success at a particular state that makes the utility agent different.

   d. It takes the agents happiness into account and gives an idea of how happy the agent is because of the utility and hence the action with maximum utility is considered. This associated degree of happiness can be calculated by mapping a state onto a real number.

   e. Mapping of state onto a real number with a help of utility function gives the efficiency of an action to achieve the goal.

5. Learning Agents:

   a. Learning agents has the capability to learn from past experiences and take actions or decisions based on learning capabilities.

   b. It gains basic knowledge from the past and uses that learning to act and adapt automatically.

   c. It consists of four conceptual components.

      i. Learning element: It makes improvement by learning from the environment.

      ii. Critique: It provides feedback to the learning agent giving the performance measure of the agent with respect to the fixed performance standards.

iii. Performance element: It selects the external actions.
iv. Problem generator: This suggests action that lead to new and informative experiences.

Problem Solving & Problem Spaces

Water Jug Problem

The water jug problem, also known as the Die Hard problem, is a classic problem in AI that involves finding a way to measure a specific amount of water using two jugs of different sizes. In this case, we have a four-liter jug and a three-liter jug. The goal is to fill one of the jugs with exactly two liters of water.

To solve this problem, we can use a search algorithm to explore all possible states of the system until we find a solution. Here are the steps to solve this problem:

1. Define the state space: In this problem, the state space consists of all possible configurations of the two jugs. Each configuration is represented by a pair of integers (x, y), where x is the amount of water in the four-liter jug and y is the amount of water in the three-liter jug. The initial state is (0, 0), which represents both jugs being empty.
2. Define the operators: The operators in this problem are pouring water from one jug to another. We can pour water from the four-liter jug into the three-liter jug, or from the three-liter jug into the four-liter jug. We can also empty one of the jugs.
3. Define the goal state: The goal state is (x, y) where x = 2 and y can be any value.
4. Implement the search algorithm: We can use a depth-first search algorithm to explore all possible states of the system. Starting from the initial state, we apply each operator and generate new states. We then apply the operators to the new states and continue generating new states until we find the goal state. If we reach a state that we have already visited, we backtrack and try a different path.

Here is the sequence of steps to reach the goal state:

1. Start with the initial state (0, 0).
2. Pour water from the four-liter jug into the three-liter jug, resulting in the state (1, 3).

3. Empty the three-liter jug, resulting in the state (1, 0).
4. Pour water from the four-liter jug into the three-liter jug, resulting in the state (0, 1).
5. Fill the four-liter jug, resulting in the state (4, 1).
6. Pour water from the four-liter jug into the three-liter jug until the three-liter jug is full, resulting in the state (2, 3).

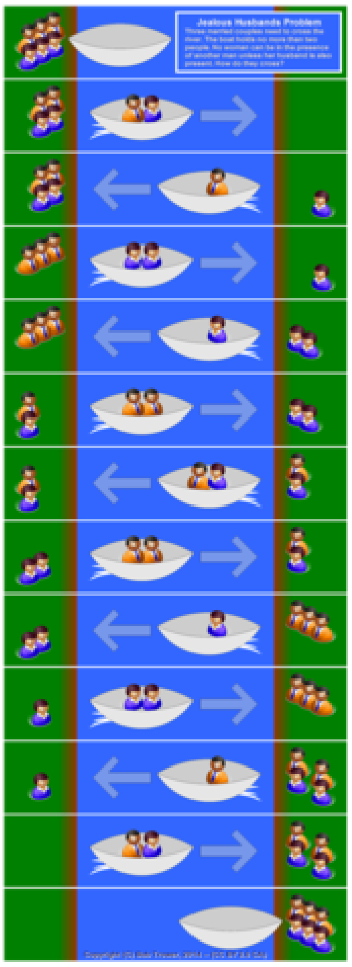
Therefore, we can measure exactly two liters of water using the four-liter and three-liter jugs by following the above steps.

Missionaries and Cannibals

The problem of crossing a river with three missionaries and three cannibals is a classic AI problem in which we need to find a way to transport all the people safely across the river. We have a boat that can hold at most two people and at least one cannibal, and we must ensure that there are never more cannibals than missionaries on either side of the river. Here are the steps to solve this problem:

1. Define the state space: In this problem, the state space consists of all possible configurations of the people and the boat on either side of the river. Each configuration is represented by a tuple (ML, CL, B, MR, CR), where ML is the number of missionaries on the left bank, CL is the number of cannibals on the left bank, B is the location of the boat (either 'left' or 'right'), MR is the number of missionaries on the right bank, and CR is the number of cannibals on the right bank. The initial state is (3, 3, 'left', 0, 0), which represents all the people and the boat being on the left bank.
2. Define the operators: The operators in this problem are moving people across the river in the boat. We can move either one or two people across the river, and at least one of them must be a cannibal. If there are more cannibals than missionaries on either side of the river, the missionaries will be eaten and the state will be invalid.
3. Define the goal state: The goal state is (0, 0, 'right', 3, 3), which represents all the people and the boat being on the right bank.
4. Implement the search algorithm: We can use a depth-first search algorithm to explore all possible states of the system. Starting from the initial state, we apply each operator and generate new states. We then apply the operators to the new states and continue generating new states until we find the goal state. If we reach a state that we have already visited, we backtrack and try a different path.

Here is the sequence of steps to transport all the people safely to the opposite bank if we start at the initial state (3, 3, 'left', 0, 0):



1. Move one cannibal and one missionary from the left bank to the right bank, resulting in the state (2, 2, 'right', 1, 1).
2. Move one missionary back to the left bank, resulting in the state (3, 2, 'left', 0, 1).
3. Move two cannibals to the right side of the bank, resulting in the state (3, 0, 'right', 0, 3).
4. Move one cannibal from the right bank to the left bank, resulting in the state (3, 1, 'left', 0, 2).
5. Move two missionaries from the left bank to the right bank, resulting in the state (1, 1, 'right', 2, 2).
7. Move one missionary and one cannibal from the right bank to the left bank, resulting in the state (2, 2, 'left', 1, 1).
8. Move two missionaries from the left bank to the right bank, resulting in the state (0, 2, 'right', 3, 1).
9. Move one cannibal back to the left bank, resulting in the state (0, 3, ‘left’, 3, 0).
10. Move two cannibals to the right bank, resulting in the state (0, 1, ‘right’, 3, 2).
11. Move one missionary from the right bank to the left bank, resulting in the state (1, 1, 'left', 2, 2).
12. Move one missionary and one cannibal from the left bank to the right bank, resulting in the state (0, 0, 'right', 3, 3).

In this sequence of steps, all six people have been safely transported across the river to the opposite bank, and at no point have there been more cannibals than missionaries on either side of the river. This solution satisfies all the constraints of the problem and is a valid solution.

8 Puzzles Problem

Consider 8 puzzle problem it consists of 3*3 board with 8 numbered tiles and 1 blank space. A tile adjustment to the blank space can slide into the space, the objective is to reach the specified goal state.

1. **States**: A state in the 8 puzzle problem consists of a 3x3 grid of tiles with numbers 1-8 and a blank space. There are 9! (362880) possible states, but not all of them are reachable from any given state.
2. **Initial state**: Let the initial state be as follows, where the blank space is represented by the symbol '_':

   1 2 3
   4 5 _
   7 8 6

3. **Actions**: In the 8 puzzle problem, a valid action is to slide a tile adjacent to the blank space into the blank space. Specifically, if the blank space is at position (i, j), then a tile can be moved into the blank space from any of the adjacent positions (i-1, j), (i+1, j), (i, j-1), or (i, j+1), provided that the new position is within the bounds of the grid.
4. **Transition model**: Given a state s and an action a, the transition model T(s, a) returns the resulting state after applying the action to the current state. For example, if the current state is:

   1 2 3
   4 5 _
   7 8 6

and the action is to slide the tile '5' into the blank space, then the resulting state would be:

```
1 2 3
4 _ 5
7 8 6
```

Note that the action does not change the state of any other tile, but simply moves the tile into the blank space.

5. **Goal state**: The goal state in the 8 puzzle problem is typically specified as follows:

   1 2 3
   4 5 6

7 8 _

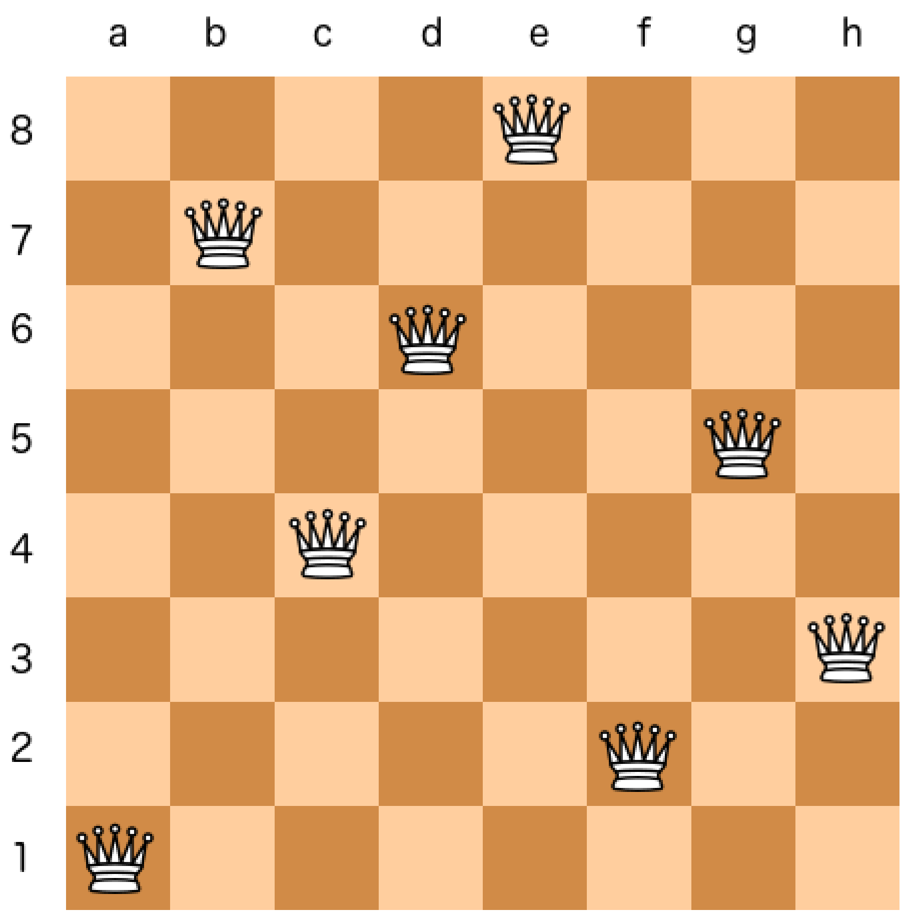
That is, all the tiles are in their proper positions, except for the blank space which is in the bottom right corner of the grid.

6. **Path cost**: The cost of a path from the initial state to a goal state in the 8 puzzle problem is typically measured by the number of moves required to reach the goal state. In other words, the path cost is equal to the number of actions (tile slides) required to transform the initial state into the goal state.

8 Queens Problem

The goal of 8 queens problem is to place on the chess board such that no queen attacks any other (A queen attacks any other queen placed in the same row, column or diagonal).

1. **States**: A state in the 8 Queens problem consists of a 8x8 chessboard with 8 queens placed on it. There are 64 choose 8 (4,426,165,368) possible states, but not all of them are valid solutions.
2. **Initial state**: Let the initial state be an empty chessboard with no queens placed on it.
3. **Actions**: In the 8 Queens problem, a valid action is to place a queen in an empty square on the chessboard, provided that the placement does not violate any of the rules of the problem. Specifically, a queen can be placed in a square if and only if no other queen is attacking it horizontally, vertically, or diagonally.
4. **Transition model**: Given a state s and an action a, the transition model T(s, a) returns the resulting state after applying the action to the current state. For example, if the current state is an empty chessboard and the action is to place a queen in the top-left corner, then the resulting state would be a chessboard with one queen placed in the top-left corner.
5. **Goal state**: The goal state in the 8 Queens problem is a state in which 8 queens are placed on the chessboard such that no two queens are attacking each other horizontally, vertically, or diagonally.
6. **Path cost**: The cost of a path from the initial state to a goal state in the 8 Queens problem is typically measured by the number of actions (queen placements) required to reach the goal state. In other words, the path cost is equal to the number of queens that need to be placed on the chessboard to form a valid solution. Note that there can be multiple valid solutions to the problem, and the path cost can vary depending on the specific solution found.



Search Strategies

1. Uninformed search:
    a. It means that the machine follows the algorithm regardless of whether it is right or wrong, efficient or inefficient.
    b. These algorithms are brute force algorithms and they don't have extra information about the search space.
    c. The only information they have is on how to traverse or visit the nodes in the tree.
    d. Uninformed search algorithms are also called blind search algorithms.
    e. The search algorithm produces the search tree without using any domain knowledge.
    f. They don't have any background information on how to approach the goal.
    g. Types of uninformed search algorithms:
        i. Breadth-First Search:

1. It is the most common search strategy for traversing trees or graphs.
2. This algorithm searches breadth-wise; it starts searching from the root node and expands all the successor nodes at the current level before moving to the nodes of the next level.
3. It uses FIFO strategy.

ii. Depth-First Search:

1. It explores the search space by always extending the current path with a new node at the deepest level of the search tree.
2. It uses LIFO strategy.
3. It is often implemented recursively.
4. It can get stuck in an infinite loop if the search space has cycles.

2. Informed search:

a. In contrast to uninformed search, informed search uses additional knowledge to guide the search towards the goal state.

b. This additional knowledge is also known as a heuristic function, which is an estimate of the distance to the goal.

c. The heuristic function can be used to evaluate the quality of each node, and then the search algorithm can select the node that appears to be closest to the goal state.

d. Informed search algorithms are generally more efficient than uninformed search algorithms, but they require domain-specific knowledge to define the heuristic function.

e. Types of informed search algorithms:

i. Greedy Best-First Search:

1. It evaluates the nodes using only the heuristic function, which is the estimated distance to the goal state.
2. It always selects the node that appears to be closest to the goal state, without considering the cost of the path to reach that node.
3. It can be efficient, but it may not always find the optimal solution.

ii. A* Search:

            1. It evaluates the nodes using a combination of the heuristic function and the cost of the path to reach that node.
            2. It selects the node that has the lowest sum of the heuristic function and the cost of the path to reach that node.
            3. It is guaranteed to find the optimal solution if the heuristic function is admissible and consistent.

Constraint Satisfaction

1. Definition: A search procedure that operates in a space of constraint sets.
2. Goal is to discover a problem state that satisfies a given set of constraints.
    a. Process:
        i. Constraints are discovered and propagated throughout the system.
        ii. If there is still no solution, then the search begins. A guess is made about something and added as a new constraint.
3. Components of CSP:
    a. Variable (V) Set: {V1, V2,..., Vn}.
    b. Domain (D): Every variable has a domain {D1, D2,..., Dn}.
    c. Constraints (C):
        i. Specify allowable combinations of values.
        ii. Constraint is represented as Ci = (scope, relation).
            - Scope is a tuple of variables that participate in the constraint.
            - Relation defines values that a variable can take on.
        iii. A relation can be represented as:
            - An explicit list of all tuples of values that satisfy the constraint.
            - An abstract relation that supports two operations:
                - Testing if a tuple is a member of the relation.
                - Enumerating the members of the relation.

Examples:

1. Sudoku

2. Crypt arithmetic problems
3. Graph coloring problems

Min-Max Algorithm

1. The Min-Max Algorithm is a recursive algorithm used to select an optimal move for a player assuming that the other player is also playing optimally.
2. Examples of games that use the Min-Max Algorithm include Tic-Tac-Toe, Chess, and many others.
3. These types of games are referred to as games of perfect information because it is possible to see all expected moves of a particular game.
4. However, there can be two-player games that are not of perfect information, such as Scrabble, because the opponent's move cannot be predicted.
5. The algorithm involves constructing a game tree, which is a structure in the form of a tree consisting of all possible moves that allow you to move from a state of the game to the next state.
6. A game can be defined as a search problem with the following components:
    a. Initial state: It comprises the position of the board and shows different actions or moves.
    b. Successor function: It defines the legal moves a player can take.
7. Steps to execute the Min-Max Algorithm:
    a. Construct the complete game tree.
    b. Evaluate scores for leaves using evaluating functions.
    c. Backup score from leaves to root considering the player time, i.e. for the max player, select the child with the maximum score. For the min player, select the child with the minimum score.
    d. At the root node, select the node with the maximum value and perform the corresponding move.

Hill Climbing Algorithm

1. Hill Climbing Algorithm is a local search algorithm that tries to find the optimal solution to a problem by continuously moving in the direction of increasing

elevation or value, towards the peak of the mountain or best solution.

2. The algorithm terminates when it reaches a peak value where no neighbor has a higher value.
3. It generates possible solutions until it finds the expected solution.

Simple Hill Climbing Algorithm:

1. The Simple Hill Climbing Algorithm considers one neighboring node at a time and selects the first one that optimizes the current cost to be the next node.
2. The algorithm can be summarized into the following steps:
    a. Evaluate the initial state. If it is a goal state, return success and stop.
    b. Loop until a solution is found or there are no new operators left to apply.
    c. Select and apply an operator to the current state.
    d. Check the new state:
        i. If it is a goal state, return success and quit.
        ii. Otherwise, if it is better than the current state, assign the new state as the current state.
        iii. If it is not better than the current state, return to Step 2.
    e. Terminate the algorithm.

Hill Climbing Algorithm is commonly used in optimization problems, such as in finding the best solution to a scheduling problem or in optimizing the weights of a neural network.

Knowledge Representation and Reasoning

Knowledge Representation and Reasoning is a sub-area of AI concerned with understanding, designing, and implementing ways of representing information in computers so that programs (Agents) can use this information.

This area of AI is used for deriving information, deciding what to do next, planning future activities, and solving problems in areas that normally require human expertise.

Reasoning: Reasoning is the use of symbolic representation of statements in order to derive new statements.

Parts of Knowledge Representation

1. Object:
    a. Object knowledge is the information regarding an object and can be considered a type of knowledge.
    b. Examples of object knowledge include "cars have wheels" and "trains are locomotives".
2. Events:
    a. Our perception of the world is based on knowledge regarding different events that have taken place.
    b. Examples of event knowledge include "achievements" and "advancements of societies".
3. Performance:
    a. Performance knowledge deals with how humans and other beings perform certain actions in different situations.
    b. It helps in understanding the behavioral side of knowledge.

Types of Knowledge

Meta Knowledge: Meta knowledge is the knowledge of what you know.

Facts: Facts are a type of knowledge and refer to the factual description of the world.

Knowledge Based: Knowledge based refers to a group of information regarding any discipline.

Types of Knowledge Representation

To represent knowledge to machines, we have to identify and classify different types of knowledge.

1. Declarative Knowledge: Declarative knowledge is the knowledge that represents the facts, objects, and concepts that help us to describe the world around us.
2. Procedural Knowledge:
    a. Procedural knowledge is more complex than declarative knowledge as it refers to how things behave and work.

   b. This knowledge is used to accomplish any task using certain procedures, rules, and strategies, so that the system can work efficiently.

3. Meta Knowledge: Meta knowledge is the knowledge regarding other types of knowledge.
4. Heuristic Knowledge:
   a. Heuristic knowledge is the knowledge provided by experts of certain domains or disciplines of fields and has been obtained after years of experience.
   b. This type of knowledge helps in taking the best approach to solve a particular problem and deciding upon something.
5. Structural Knowledge:
   a. Structural knowledge helps in establishing relationships between concepts or objects and their description.
   b. It acts as the basic form of knowledge to solve real-world problems.

Properties of knowledge representation

1. **Representational Adequacy**: The main property of KRS is that it should be adequate and can make the AI system understand i.e. represent all the knowledge required to deal with a particular field or domain
2. **Inferential adequacy**: The KRS must be flexible enough to deal with the present knowledge to make a way for newly possessed knowledge
3. **Inferential efficiency**: The representation system cannot accommodate new knowledge in the presence of old knowledge however it can add this knowledge efficiently
4. **Acquisitional efficiency**: It is the ability to gain new knowledge automatically, helping the AI to add its current knowledge to become smarter and productive

Knowledge representation techniques

1. Logical representation:
   a. It is a basic form of representing knowledge to machines using a well-defined syntax with proper rules.

    b. It is best used for representing facts to machines and acts as communication rules.
    c. There are two types: Propositional Logic & First Order Logic

Propositional Logic

1. Propositions are sentences or statements that are either true or false.
2. It is derived from human logic.
3. Examples of propositions: "All humans are mortal" or "I am attending a lecture."
4. Examples of things that are not propositions: question marks, commands.
5. Propositional logic is a mathematical system for reasoning about propositions and how they relate to one another.
6. **Propositional variables:**
    a. They represent arbitrary propositions using upper-case letters.
    b. Example: P: "It is humid and it is hot."
    c. Each variable can take either true or false values. If the proposition is true, then we say its truth value is true.
7. **Propositional connectives:**
    a. **Logical not (¬P) read as not P:** Not P is true if and only if P is false. It is also called logical negation.
    b. **And (P ∧ Q) read as P and Q:** P and Q are true if both P and Q are true. It is called logical conjunction.
    c. **Or (P ∨ Q) read as P or Q:** P or Q is true if at least one of P or Q is true. It is also called logical disjunction.
    d. **Implication (P → Q) read as P implies Q or If P then Q:** It is false when P is true and Q is false and is true otherwise. It is also called conditional operator.
    e. **Biconditional (P ↔ Q) read as P if and only if Q:** It is true if P and Q have the same truth values. It is true if P and Q have the same values; otherwise, it is false. It is also called biconditional operators.
8. When we use implementation, **P is called premise or hypothesis, and Q is called conclusion.** Typically, there is a causal relationship between P and Q.

9. **Operator Precedence:**
    a. NOT
    b. AND
    c. OR
    d. Implication
    e. Biconditional
10. Well formed Formula: (Construct them using propositional logic). A sentence is called a well-formed formula if:
    a. A symbol S is a sentence.
    b. If S is a sentence, then the negation of S (]S) is a sentence.
    c. If S is a sentence, then (S) is a sentence.
    d. If S and T are sentences, then S^T, SvT, S->T, and S<->T are sentences.
11. Arguments in Propositional Logic:
    - An argument in propositional logic is a sequence of propositions. All but the final proposition are called premises. The last statement is the conclusion.
    - The argument is valid if the premises imply the conclusion. An argument that is valid no matter what propositions are substituted into its propositional variables is called an argument form. If the premises are P1, P2, … Pn and the conclusion is Q, then (P1^P2^P3…Pn)-> Q is a tautology.
    - A tautology is a formula that is always true, i.e., it is true for every assignment of truth values to its simple components. You can think of a tautology as a rule of logic.
    - The opposite of a tautology is a contradiction. A contradiction is always false, i.e., for every assignment of truth values to its simple components.
12. Disadvantages of propositional logic:
    a. It is hard to identify individuals.
    b. Properties of individuals or relations between individuals cannot be directly expressed, e.g., "ABJ is intelligent".
    c. Generalization, patterns, etc. cannot be easily represented. For example, "All triangles have three sides".

First Order Logic

FOL is a mode of representation in AI. It is an extension of propositional logic. It represents natural language statements in a concise manner. It is also called Predicate Logic. It is a powerful language used to develop information about an object and express the relationships between objects.

Three things of a language are a concern:

1. Syntax: It specifies which group of symbols are arranged in what way and how they are formed properly.
2. Symmetrix: Specify what the well-formed expressions are supposed to mean.
3. Pragmatics: Specifies how the meaningful expression can be used.

FOL consists of the following:

1. Objects – for e.g., people, color, names, etc.
2. Relations – which are defined between the objects. For e.g.: Brother of, etc.
3. Functions – It is a subset of relations that maps object to one another.

In FOL, each variable refers to some object in a set called Domain or Universe or Universe of discourse.

It uses predicates which is a property that a variable or finite collection of variables can have. It can take any number of arguments P(x1,x2,…..xn). But each predicate has a fixed number of arguments called arity -> n.

A predicate becomes a proposition when specific values are assigned to the variables which is either true or false.

Domain or Universe or Universe of discourse.
It is a set of values that can be assigned to a predicate. For e.g. if
U= {1,2…..10} (universe or domain)
P(x) : x is even
Output = {2,4,6,8,10}
The truth set of p(x) the set of all elements “t” of ‘U’ such that P(t) is true. i.e., t belongs to U such that P(t) is true
{t E U}

Functions
Returns objects associated with other objects.

Syntax of FOL
There are 2 types of symbols:

1. Variables - Any sequence of lowercase alphabets and numeric characters in which the first character is lowercase alphabet.
2. Constants – Object constant. It is used to name a specific element of a universe of discourse.
3. Function Constants – It is used to designate a function on members of universe of discourse.
4. Relation constants – It is used to name a relation on a universe of discourse.

   What is provided by FOL:

   FOL provides:
5. Variable Symbols (x,y,z……)
6. Connectives (v, ^, ->, <->)
7. Quantifiers
   a. Universal Quantifier
   b. Existential Quantifiers.

Terminology used in FOL

1. Term (Denotes a real-world individual) it is a constant symbol, a variable, function of n terms, etc. A term with no variable is called – Ground term.
2. Atomic sentence (Has value true of false) is an n-place predicate of n terms
3. Complex sentence- it is formed from atomic sentences connected by the logical connectives. For e.g., ⏋P, P^Q
4. Quantified sentence – Here Quantifiers like for all and there exist are added. Quantified sentences provide a more flexible way of talking about objects in the universe of discourse.
5. Well-formed Formula (WFF) – It is a sentence consisting of no free variables i.e., All variables are bound by universal and existential Quantifiers.
6. Equality – It is represented by x = y. It says whether 2 objects are equal or not. It is a logical constant.

Quantifiers

Quantifiers are words that express the quantity or extent of something. In English, there are two main quantifiers: "some" and "all".

- "Some" indicates that at least one item satisfies a condition. For example: "Some students passed the exam." This means that there are students who passed the exam, but not necessarily all of them.
- "All" indicates that every item satisfies a condition. For example: "All students passed the exam." This means that every student passed the exam, and there were no exceptions.

Bound and free variables

1. A variable can occur as a term in a sentence without an enclosing quantifier. When used in this way, a variable is said to be free.
2. All variables in a predicate must be bound to turn a predicate into a proposition.
3. We bind a variable by assigning it a value or quantifying it.
4. If a sentence has no free variable, then it is called a closed sentence.
5. If a variable is not free nor bound, it is called a ground sentence.

De Morgan's Law for Quantifiers

1. $\exists x P(x) \equiv \neg \forall x \neg P(x)$
2. $\forall x P(x) \equiv \neg \exists x \neg P(x)$

Properties of Quantifiers

1. $\exists x \exists y$ is the same as $\forall y \forall x$.
2. ∋x∋y is not the same as ∋y∋x.

Note: Order of quantifier is important when including both existential and universal quantifier in FOL.

FOL inference rules for quantifier

1. As propositional logic, we also have inference rules in FOL:
    a. Universal generalization:
        - It states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as $\forall x P(x)$, it is represented as:
          P(c) $\therefore$ $\forall x P(x)$
    b. Existential instantiation:

- It states that if there exists at least one object c in the universe of discourse that satisfies the predicate P(x), then we can replace the existential quantifier with a specific constant c. It is represented as:
  ∃xP(x) $\therefore$ P(c) [c is a new constant]

c. Universal instantiation:

- It states that if a formula P(x) is true for any arbitrary element c in the universe of discourse, then we can replace the universal quantifier with a specific constant c. It is represented as:
  ∀xP(x) $\therefore$ P(c) [c is a constant in the universe of discourse]

d. Existential generalization:

- It states that if a formula P(x) is true for some element in the universe of discourse, then we can introduce an existential quantifier. It is represented as:
  P(c) $\therefore$ ∃xP(x) [c is a new constant]

Planning

Planning as Search

In planning as search, the planner constructs a search tree, where each node represents a state of the world and each edge represents an action that can be taken to transform one state into another. The search algorithm then explores this tree, looking for a path from the initial state to the goal state. The efficiency of the search algorithm depends on the complexity of the problem and the search strategy used.

There are several search strategies that can be used for planning, including depth-first search, breadth-first search, and A* search. Depth-first search explores the search tree by going as deep as possible before backtracking. Breadth-first search explores the search tree level-by-level, starting from the initial state. A* search is an informed search algorithm that uses a heuristic function to guide the search towards the goal state.

Partial Order Planning

Partial order planning is a more flexible approach to planning compared to total order planning, where actions are executed in a fixed order. In partial order planning, actions can be executed in any order as long as they do not conflict with each other. This makes it possible to generate plans that are more efficient than those produced by total order planning.

The planner constructs a partial order plan by first creating a set of actions that achieve the goal conditions. Then, it identifies the causal relationships between the actions and orders them in a partial order that satisfies the goal conditions. The planner can also refine the partial order plan by adding new actions or changing the order of existing actions to improve the plan's quality.

Situation Calculus

The situation calculus is a logical formalism used for reasoning about actions and change. It provides a way to represent the effects of actions on the world and to reason about the consequences of performing a sequence of actions. The situation calculus is based on first-order logic and uses the concept of a situation to represent a state of the world.

In the situation calculus, a situation is a complete description of the world at a particular time. Actions are represented as formulas that describe their preconditions and effects on the world. For example, an action that moves a block from one location to another would have a precondition that the block is at the starting location and an effect that the block is at the destination location. The situation calculus also includes axioms that describe how situations and actions relate to each other.

Sussman Anomaly

The Sussman anomaly is a problem in AI planning that was first described by Gerald Sussman in 1973. It arises when a planner attempts to solve a problem that involves a set of blocks that need to be moved from one location to another, subject to certain constraints. The Sussman anomaly is a difficult problem for traditional planning approaches because it involves a large number of possible actions and the planner needs to reason about the order in which the actions are executed.

The Sussman anomaly problem involves three pegs and a set of disks of different sizes. The disks are initially placed on one of the pegs in a particular order, with the largest disk at the bottom and the smallest at the top. The goal is to move the disks to another peg, following certain rules, such as only moving one disk at a time and never placing a larger disk on top of a smaller one.

Representing and Reasoning with Uncertain Knowledge

Bayes rule

Bayes rule, also known as Bayes' theorem or Bayes' law, is a fundamental concept in probability theory that describes how to update our beliefs about an event or situation based on new evidence. It provides a way to calculate the probability of a hypothesis (or event) given some observed data or evidence.

The formula for Bayes' rule is: **P(A|B) = P(B|A)P(A) / P(B)**

where P(A|B) is the probability of A given B, P(B|A) is the probability of B given A, P(A) is the prior probability of A, and P(B) is the prior probability of B.

Bayes' rule is widely used in a variety of fields, including medicine, finance, engineering, and machine learning. For example, in medical diagnosis, Bayes' rule can be used to calculate the probability that a patient has a particular disease given their symptoms and medical history.

Bayesian networks

A Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional dependencies using a directed acyclic graph. It provides a way to model uncertainty and make predictions based on probabilistic reasoning.

In a Bayesian network, nodes represent random variables, and edges represent conditional dependencies between them. Each node has a conditional probability distribution that specifies the probability of that node given its parents in the graph. The joint probability distribution over all the nodes in the graph is the product of the conditional probability distributions of each node given its parents.

Bayesian networks have a wide range of applications, including medical diagnosis, financial forecasting, speech recognition, and natural language processing. For example, in medical diagnosis, a Bayesian network can be used to model the relationships between symptoms and diseases, and to predict the probability of a patient having a particular disease given their symptoms.

Decision tree networks

A decision tree network is a type of decision support tool that uses a tree-like graph or model of decisions and their possible consequences. It provides a way to make decisions based on uncertain or incomplete information.

In a decision tree network, each node represents a decision point, and each edge represents a possible outcome of that decision. The leaves of the tree represent the final outcomes or decisions. The decision tree network can be constructed using various algorithms, including the ID3 algorithm and the C4.5 algorithm.

Decision tree networks have a wide range of applications, including data mining, finance, and healthcare. For example, in finance, a decision tree network can be used to predict the likelihood of a borrower defaulting on a loan based on their credit history and other financial information.

Applications

Bayesian methods, including Bayes rule, Bayesian networks, and decision tree networks, have a wide range of applications in various fields, including finance, healthcare, marketing, and natural language processing.

In finance, Bayesian methods can be used to model and predict stock prices or market trends. For example, a Bayesian network can be used to predict the likelihood of a stock market crash given historical market data.

In healthcare, Bayesian methods can be used to predict disease outcomes or develop personalized treatment plans. For example, a Bayesian network can be used to predict the likelihood of a patient having a heart attack based on their medical history and other risk factors.

In marketing, Bayesian methods can be used to analyze consumer behavior and optimize advertising campaigns. For example, a Bayesian network can be used to predict the likelihood that a customer will purchase a particular product based on their demographic and behavioral data.

In natural language processing, Bayesian methods can be used for tasks such as sentiment analysis and language modeling. For example, a Bayesian network can be used to predict the sentiment of a text message

Decision-Making

Utility theory

Utility theory is a mathematical framework that helps us to understand how individuals make decisions when faced with multiple options that have uncertain outcomes. The basic idea behind utility theory is that people choose the option that maximizes their expected utility, which is a measure of the subjective value or usefulness that they attach to each option.

The concept of utility is often used in economics to explain how individuals make choices based on their preferences and beliefs about the world. For example, when deciding whether to invest in a particular stock, an investor might weigh the potential return against the risk of losing money. By assigning a utility value to each potential outcome, the investor can determine which option is most likely to yield the highest overall utility.

Decision theory

Decision theory is a systematic approach to making optimal decisions in situations of uncertainty, where the outcomes of different choices are not known with certainty. Decision theory provides a framework for evaluating the costs and benefits of different options based on the probabilities of different outcomes, helping decision-makers to choose the option that maximizes their expected utility.

One of the key concepts in decision theory is the idea of expected value, which is the weighted average of the possible outcomes of a decision, taking into account their probabilities. By calculating the expected value of each option, decision-makers can choose the option that has the highest expected value and is therefore most likely to yield the best outcome.

Game theory

Game theory is a mathematical framework that helps us to understand how individuals or groups of individuals make decisions in situations where their outcomes are interdependent, and their actions affect each other. Game theory provides a way to model and analyze strategic interactions between different players, helping decision-makers to anticipate the actions of others and choose the best course of action given the actions of others.

Game theory is often used in economics and political science to model behavior in situations such as bargaining, pricing, and voting. One of the key concepts in game theory is the Nash equilibrium, which is a set of strategies where no player can gain by unilaterally changing their strategy, assuming that all other players remain constant.

Applications

The field of artificial intelligence (AI) has developed many techniques for making decisions in complex and uncertain environments. These techniques are often used in applications such as finance, healthcare, transportation, and more. Some of the most commonly used decision-making algorithms in AI include:

- Reinforcement learning: Reinforcement learning is a type of machine learning that involves training an agent to make decisions by interacting with an environment and receiving feedback in the form of rewards or punishments. Reinforcement learning is commonly used in robotics, gaming, and other applications where the agent must learn to make decisions based on sensory input.
- Decision trees: Decision trees are a type of machine learning algorithm that involves constructing a tree-like model of decisions and their possible consequences. Decision trees are often used in applications such as credit scoring and fraud detection, where decisions must be made based on a large number of variables.
- Bayesian networks: Bayesian networks are a type of probabilistic graphical model that allows decision-makers to model the relationships between different variables and their likelihoods. Bayesian networks are commonly used in medical diagnosis, environmental modeling, and other applications where multiple variables must be taken into account.
- Fuzzy logic systems: Fuzzy logic systems are a type of rule-based system that allows decision-makers to reason about imprecise or uncertain information. Fuzzy logic systems are often used in control systems, where precise measurements may not be possible.

Overall, the study of decision-making is an important field that has applications in many areas of life, from personal decision-making to business strategy and public policy. By understanding the principles and techniques of decision-making, individuals and organizations can make more informed and effective decisions

Machine Learning and Knowledge Acquisition

Machine learning (ML) is a subfield of artificial intelligence (AI) that deals with the development of algorithms and models that enable machines to learn from data and make predictions or decisions without being explicitly programmed. The knowledge acquisition process involves obtaining information from various sources and incorporating it into the system's knowledge base to enhance its performance.

Various Components of Learning System

A machine learning system consists of several components, each of which performs a specific function in the overall learning process. The input data component provides the data to the system in a format that can be used for machine learning. The feature extraction component identifies the relevant features in the input data that are most useful for predicting the outcome. The model training component uses the labeled training data to train the machine learning model. The model evaluation component tests the trained model on a separate dataset to evaluate its performance. Finally, the prediction component uses the trained model to make predictions on new, unseen data.

Knowledge Acquisition

In order to improve the performance of a machine learning system, it may be necessary to incorporate additional knowledge beyond what is contained in the input data. Knowledge acquisition is the process of obtaining this additional knowledge and integrating it into the system. This can involve identifying relevant knowledge sources, extracting relevant information from those sources, and organizing that information in a format that can be used by the machine learning system. Knowledge acquisition is an important part of many machine learning systems, especially those that rely on expert knowledge to supplement the input data.

Naïve Bayes

Naïve Bayes is a probabilistic machine learning algorithm that is commonly used for classification problems. It is based on Bayes' theorem, which is a mathematical formula that describes the probability of an event based on prior knowledge of conditions that might be related to the event. Naïve Bayes assumes that the features in the input data are independent of each other, meaning that the presence or absence of one feature does not affect the probability of any other feature being present. This assumption simplifies the calculation of the probabilities, making Naive Bayes a fast and efficient algorithm for classification problems.

Decision Tree Classifiers

A decision tree is a tree-like structure that can be used for both classification and regression problems. The tree consists of nodes that represent different decision points and branches that represent the possible outcomes of those decisions. Each internal node in the tree represents a decision based on a feature in the input data, and each leaf node represents a class or regression value. Decision tree classifiers work by recursively partitioning the input data based on the feature values until a leaf node is reached, which corresponds to a predicted class or regression value. Decision trees are a popular and powerful algorithm for machine learning because they are easy to interpret and can handle both categorical and continuous data.

Q-Learning

Q-learning is a reinforcement learning algorithm that is used to solve problems where an agent must take a sequence of actions to maximize its reward. The algorithm works by maintaining a table of Q-values that represent the expected reward for each action in each state. The Q-values are updated based on the rewards that the agent receives for each action it takes. The agent then selects the action with the highest Q-value at each step in order to maximize its reward. Q-learning is a powerful algorithm that has been used to solve a variety of reinforcement learning problems, including game-playing and robotics applications.

Reinforcement Learning

Reinforcement learning is a type of machine learning that is used to train an agent to make decisions in an environment to maximize a reward signal. The agent learns by interacting with the environment and receiving feedback in the form of rewards or punishments. The goal of reinforcement learning is to develop an optimal policy that maximizes the expected cumulative reward over time. Reinforcement learning is a powerful approach that has been used to solve a variety of problems, including game-playing, robotics, and control systems. It is particularly useful in situations where it is difficult to specify an explicit goal or objective function.