Discuss various problem characteristics with examples.

1. **Is the problem decomposable?**

Decomposable problems can be solved by solving each subproblem individually, and their solutions can then be combined to solve the overall problem. Some problems, like complex integrals, can be decomposed into simpler subproblems.

2. **Can solution steps be ignored or undone?**

It means whether the solution steps can be ignored or undone if they lead to a dead end. For instance, in the 8-puzzle, moves can be undone to try different arrangements of tiles.

On the other hand, in chess, once a move is made, it cannot be undone.

3. **Is the problem's universe predictable?**

It means whether the outcomes of the problem can be determined with certainty or if they involve uncertainty. For example, some optimization problems may have multiple potential solutions with different probabilities of being optimal.

4. **Is a good solution absolute or relative?**

An absolute solution is one where finding a single correct path is sufficient to achieve the desired goal e.g. water jug puzzle. A relative solution is one that requires evaluating multiple possible paths to find the best or optimal solution e.g. traveling salesman problem

5. **Is the solution a state or a path?**

In some problems, the desired outcome is a specific state that satisfies the problem's requirements e.g. 8-puzzle. In other problems, the solution involves finding a path to reach the desired goal state e.g. maze-solving.

6. **What is the role of knowledge?**

In some problems, extensive domain-specific knowledge is required to find possible solutions e.g. chess requires. In contrast, other problems may rely more on general problem-solving algorithms and heuristics, requiring less domain-specific knowledge.

7. **Can a computer give the solution, or humans interaction is required?**

In some cases, computers can find solutions to problems without any interaction with humans. For example, computers can efficiently solve mathematical equations. However, in more complex and uncertain problems, human interaction may be necessary to provide guidance.

What do you mean by Intelligent Agents? Give types and examples.

Intelligent agents are entities that perceive their environment, make decisions, and take actions to achieve specific goals.

Types of Intelligent Agents:

1. Simple Reflex Agents: These agents make decisions based solely on the current percept without considering the history of percepts. Actions are determined by predefined rules.

Example: A thermostat that turns on the heating system when the temperature falls below a certain threshold.

2. Model-Based Reflex Agents: These agents maintain an internal model of the world and use it to make decisions. They consider the history of percepts to enhance decision-making.

Example: A robot vacuum cleaner that creates a map of the room as it cleans and adjusts its cleaning strategy based on obstacles encountered.

3. Goal-Based Agents: Goal-based agents have explicit goals and make decisions that lead them toward achieving these goals. They consider their current state, goals, and possible actions.

Example: An autonomous delivery drone that navigates to a specified destination by considering its current location, the destination, and the obstacles in its path.

4. Utility-Based Agents: These agents make decisions by evaluating the utility of different outcomes. They aim to maximize overall utility, considering the preferences or priorities of the agent.

Example: An investment portfolio management system that selects investments based on their expected returns and risk, aiming to maximize the portfolio's overall utility.

5. Learning Agents: Learning agents have the ability to improve their performance over time through learning from experience. They adapt to their environment and make better decisions based on feedback.

Example: An email spam filter that learns to classify emails as spam or not spam based on user feedback and evolving email patterns.

What do you mean by Overestimation and underestimation? How can this be solved?

Overestimation: When the heuristic overestimates the actual remaining cost, it might cause the A* algorithm to miss the optimal path. This happens because the algorithm prioritizes nodes with lower estimated costs, but an overestimated cost makes a potentially good path appear less promising.

Underestimation: When the heuristic underestimates the remaining cost, it might unnecessarily expand nodes that won't lead to the optimal path, wasting time and resources. While underestimation doesn't directly lead to a suboptimal path, it can make the A* algorithm less efficient.

Solution:

- **Admissibility:** Overestimation and underestimation can be avoided by using an admissible heuristic.
- **Heuristic accuracy:** A more accurate heuristic leads to better performance, regardless of admissibility.

What do you mean by computational based leaning? Explain.

Computational learning theory deals with the design and analysis of machine learning algorithms. Its goal is to understand the computational properties of these algorithms, including their ability to learn from data and generalize to new data.

The main methods of computational learning theory are:

1. Inductive learning: In inductive learning, a computer program is given a set of training data, and it is then tasked with generating a general rule that can be used to predict the output for new inputs.

2. Deductive learning: In deductive learning, a computer program is given a set of rules, and it is then tasked with generating correct outputs for new inputs using these rules.

3. Abductive learning: In abductive learning, a computer program is given a set of training data, and it is then tasked with generating a hypothesis that is consistent with the data.

4. Reinforcement learning: In reinforcement learning, a computer program is given a set of rewards and punishments, and it is then tasked with learning a policy that will maximize the rewards and minimize the punishments.

Explain A* algorithm in brief.

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Explain Unification Algorithm with a suitable example.

Step 1: If A_1 or A_2 is a variable or constant, then:

a) If A_1 or A_2 are identical, then return NIL.

b) Else if $A_1$is a variable,

a. then if A_1 occurs in A_2, then return FAILURE

b. Else return {(A_2/ A_1)}.

c) Else if A_2 is a variable,

a. If A_2 occurs in A_1 then return FAILURE,

b. Else return {(A_1/ A_2)}.

d) Else return FAILURE.

Step 2: If the initial Predicate symbol in A_1 and A_2 are not same, then return FAILURE.

Step 3: IF A_1 and A_2 have a different number of arguments, then return FAILURE.

Step 4: Set Substitution set(SUBST) to NIL.

Step 5: For i=1 to the number of elements in A_1.

a) Call Unify function with the ith element of A_1 and ith element of A_2, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

Step 6: Return SUBST.

Conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Example: Find most general unifier MGU (if exists) in each case

1. Find the MGU of {p(f(a), g(Y)) and p(X, X)}

S0 => Here, A1 = p(f(a), g(Y)), and A2 = p(X, X)

SUBST θ= {f(a) / X}

S1 => A1 = p(f(a), g(Y)), and A2 = p(f(a), f(a))

SUBST θ= {f(a) / g(y)}, **Unification failed.**

Unification is not possible for these expressions.

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

Here, A1 = p(b, X, f(g(Z))) , and A2 = p(Z, f(Y), f(Y))

S0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}

SUBST θ={b/Z}

S1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}

SUBST θ={f(Y) /X}

S2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}

SUBST θ= {g(b) /Y}

S2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} **Unified Successfully.**

And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

Differentiate between Propositional & Predicate Logic.

Propositional Logic	Predicate Logic
Propositional logic is the logic that deals with a collection of declarative statements which have a truth value, true or false.	Predicate logic is an expression consisting of variables with a specified domain. It consists of objects, relations and functions between the objects.
It is the basic and most widely used logic. Also known as Boolean logic.	It is an extension of propositional logic covering predicates and quantification.
A proposition has a specific truth value, either true or false.	A predicate's truth value depends on the variables' value.
Scope analysis is not done in propositional logic.	Predicate logic helps analyse the scope of the subject over the predicate.
It is a more generalized representation.	It is a more specialized representation.
It cannot deal with sets of entities.	It can deal with set of entities with the help of quantifiers.

Define State Space of a problem. Define "Water Jug "problem and write the production rules for the same.

A **state space** is a way to mathematically represent a problem by defining all the possible states in which the problem can be.

The **Water Jug Problem** is a classic puzzle in artificial intelligence involving two jugs, one with a capacity of 'x' litres and the other 'y' litres, and a water source. The goal is to measure a specific 'z' litres of water using these jugs, with no volume markings. It's a test of problem-solving and state space search, where the initial state is both jugs empty and the goal is to reach a state where one jug holds 'z' litres.

Production Rules for Water Jug Problem

- **Fill Jug A:** Fill jug A to its full capacity.
- **Fill Jug B:** Fill jug B to its full capacity.
- **Empty Jug A:** Empty the jug A.
- **Empty Jug B:** Empty the Jug B.
- **Pour from A to B:** Pour water from jug A to jug B unless you get an empty jug A or full jug B.
- **Pour from B to A:** Pour water from jug B to jug A until either jug B is empty or jug A is full.

Explain Simulated Annealing with an example?

Simulated annealing is a technique used in AI to find an approximate solution to a problem by slowly changing a set of values in order to find a minimum or maximum value.

Example: The traveling salesman problem

In this problem, a salesman must visit some large number of cities while minimizing the total mileage travelled. If the salesman starts with a random itinerary, he can then pairwise trade the order of visits to cities, hoping to reduce the mileage with each exchange. The difficulty with this approach is that while it rapidly finds a local minimum, it cannot get from there to the global minimum.

Simulated annealing improves this strategy. After making many trades and observing that the cost function declines only slowly, one lowers the temperature, and thus limits the size of allowed "bad" trades. After lowering the temperature several times to a low value, one may then "quench" the process by accepting only "good" trades in order to find the local minimum of the cost function.

Explain mini-max search procedure and alpha- beta cutoffs.

The **MINIMAX GAME STRATEGY** is to select the move that leads to the successor node with the highest (lowest) score. The scores are computed starting from the leaves of the tree and backing up their scores to their predecessor in accordance with the Minimax strategy. The problem with this strategy is that it explores each node in the tree.

```
function MINIMAX(N) is
    begin
       if N is a leaf then
            return the estimated score of this leaf
       else
            Let N1, N2, .., Nm be the successors of N;
            if N is a Min node then
            return min{MINIMAX(N1), .., MINIMAX(Nm)}
            else
            return max{MINIMAX(N1), .., MINIMAX(Nm)}
    end MINIMAX;
```

ALPHA-BETA cut-off is a method for reducing the number of nodes explored in the Minimax strategy. For the nodes it explores it computes, an *alpha value* and a *beta value* in addition to the score

ALPHA value of a node is never greater than the true score of this node. Initially it is the score of that node, if the node is a leaf, otherwise it is -infinity. Then at a MAX node it is set to the largest of the scores of its successors explored up to now, and at a MIN node to the alpha value of its predecessor.

BETA value of a node is never smaller than the true score of this node. Initially it is the score of that node, if the node is a leaf, otherwise it is +infinity. Then at a MIN node it is set to the smallest of the scores of its successors explored up to now, and at a MAX node to the beta value of its predecessor.

```
function MINIMAX-AB(N, A, B) is ;; Here A is always less than B
    begin
       if N is a leaf then
            return the estimated score of this leaf
       else
       Set Alpha value of N to -infinity and Beta value of N to +infinity;
            if N is a Min node then
                For each successor Ni of N loop
                  Let Val be MINIMAX-AB(Ni, A, Min{B,Beta of N});
                  Set Beta value of N to Min{Beta value of N, Val};
                  When A >= Beta value of N then
                  Return Beta value of N
                endloop;
                Return Beta value of N;
            else
                For each successor Ni of N loop
                  Let Val be MINIMAX-AB(Ni, Max{A,Alpha value of N}, B);
                  Set Alpha value of N to Max{Alpha value of N, Val};
                  When Alpha value of N >= B then
                  Return Alpha value of N
                endloop;
                Return Alpha value of N;
    end MINIMAX-AB;
```

Explain constraint satisfaction problem.

Constraint Satisfaction Problem is defined as a triple (X, D, C), where:

- X is a set of variables $\{x_1, x_2, ..., x_n\}$.

• D is a set of domains {D_1, D_2, ..., D_n}, where each D_i is the set of possible values for x_i.

• C is a set of constraints {C_1, C_2, ..., C_m}, where each C_i is a constraint that restricts the values that can be assigned to a subset of the variables.

For example, in Sudoku, a variable that represents a puzzle cell can have as its domain a range of values from 1 to 9. The constraint might be that only one of each number from 1 to 9 can appear in each row, column, and 3*3 box.

What are Bayesian Networks?

A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph. It consists of two parts:

- Directed Acyclic Graph
- Table of conditional probabilities.

Bayesian network is based on Joint probability distribution and conditional probability. The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as influence diagram.

Differentiate between Partial Order and Total Order Planners

Partial Order Planners	Total Order Planners
Follows non-deterministic approach.	Follows deterministic approach.
Only actions with causal dependencies need to be ordered.	All actions need to be linearly ordered.
Use dependency graphs	Use linear sequences
Backtracking is frequently used	Backtracking is rarely used
Allows concurrent execution of actions.	Doesn't allow concurrent execution of actions.
Explores smaller search space	Explores larger search space
Suitable for domains with parallelism	Suitable for domains with strict sequencing

How do we find a decision tree that agrees with the training?

Steps to make Decision Tree using ID3 algorithm

a) Take the Entire dataset as an input.

b) Calculate the Entropy of the target variable, As well as the predictor attributes

c) Calculate the information gain of all attributes.

d) Choose the attribute with the highest information gain as the Root Node

e) Repeat the same procedure on every branch until the decision node of each branch is finalized.

What if a decision tree is too large? Can we approximate with smaller trees?

Overfitting occurs when a model fits too closely to the training data and may become less accurate when encountering new data or predicting future outcomes**.**

Pruning is a technique that removes parts of the decision tree and prevents it from growing to its full depth. Pruning removes those parts of the decision tree that do not have the power to classify instances. Pruning is of two types — Pre-Pruning and Post-Pruning.

Pre-Pruning involves the tuning of the hyperparameters prior to training the model.

Post-Pruning involves removing non-significant branches of the model using the Cost Complexity Pruning (CCP) technique.

What is Conditional Entropy?

The conditional entropy defines the amount of information needed to describe the outcome of a random variable given that the value of another random variable is known.

The conditional entropy of Y given X is defined as

$$\mathrm{H}(Y|X) = -\sum_{x\in\mathcal{X}, y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)}$$

where x and y denote the support sets of X and Y

Heuristic Search and Heuristic Function

Heuristic search is used to search a solution space for an optimal solution for a problem. We can perform the Heuristic techniques into two categories:

- **Direct Heuristic Search:** They utilize an arbitrary sequencing of operations and look for a solution throughout the entire state space. e.g. Breadth-First Search (BFS) and Depth First Search (DFS).
- **Weak Heuristic Search:** They require domain-specific information. e.g. Best First Search (BFS) and A* search, Hill climbing and constraint satisfaction problem

Heuristic function (also called heuristic) determines how near a state is to the desired state.

Properties of good heuristic:

- **Admissibility:** The heuristic should never overestimate the true cost of reaching the goal.
- **Consistency:** The heuristic values along any path should be non-decreasing.

Knowledge representation techniques

There are mainly four ways of knowledge representation which are given as follows:

- **Logical Representation:** It consists of precisely defined syntax and semantics. There are two types of logical representation: Propositional Logic and First-order Logic.
    - Propositional logic is based on the Boolean system, which means that propositions are evaluated as either true or false.
    - In First-order logic, propositions are constructed using predicates. Predicates are statements that describe relation between objects.

  Advantage: Facilitates logical reasoning

  Disadvantage: Not intuitive, inefficient

- **Semantic Network Representation:** A semantic network is a graphical representation of knowledge, where nodes represent objects, and links represent relationships between them.

  Advantage: More intuitive than logical reasoning

  Disadvantage: Computationally expensive

- **Frame Representation:** Frame representation is a technique for organizing knowledge in a hierarchical structure.

  Advantage: Highly flexible, groups related data

  Disadvantage: Inefficient, not specific

- **Production Rules:** Production rules are a set of IF-THEN statements that represent knowledge. The IF part of a rule is a condition, and the THEN part is an action to be taken if the condition is met.

  Advantage: Modularity

  Disadvantage: Lacks learning capabilities

Iterative Deepening Search(IDS) or Iterative Deepening Depth First Search(IDDFS)

IDDFS combines depth-first search's space-efficiency and breadth-first search's fast search.

Algorithm:

```
bool IDDFS(src, target, max_depth)
    for limit from 0 to max_depth
       if DLS(src, target, limit) == true
           return true
    return false
```

```
bool DLS(src, target, limit)
    if (src == target)
        return true;

    if (limit <= 0)
        return false;

    for each adjacent i of src
        if DLS(i, target, limit?1)
            return true

    return false
```

Write the POP (Partial Order Planning) Algorithm.

```
function POP($initial, goal, actions$) returns $plan$
    $plan \leftarrow$ MAKE-MINIMAL-PLAN($initial, goal$)
    loop do
        if SOLUTION?($plan$) then return $plan$
        $S_{need}, c \leftarrow$ SELECT-SUBGOAL($plan$)
        CHOOSE-ACTION($plan, actions, S_{need}, c$)
        RESOLVE-THREATS($plan$)
    end
```

```
function SELECT-SUBGOAL($plan$) returns $S_{need}, c$
    pick a plan step $S_{need}$ from STEPS($plan$)
        with a precondition $c$ that has not been achieved
    return $S_{need}, c$
```

```
procedure CHOOSE-ACTION($plan, actions, S_{need}, c$)
    choose a step $S_{add}$ from $actions$ or STEPS($plan$) that has $c$ as an effect
    if there is no such step then fail
    add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS($plan$)
    add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS($plan$)
    if $S_{add}$ is a newly added step from $actions$ then
        add $S_{add}$ to STEPS($plan$)
        add $Start \prec S_{add} \prec Finish$ to ORDERINGS($plan$)
```

```
procedure RESOLVE-THREATS($plan$)
    for each $S_{threat}$ that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS($plan$) do
        choose either
            Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS($plan$)
            Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS($plan$)
        if not CONSISTENT($plan$) then fail
    end
```