Unit-3 - Adversarial Search Problems and Intelligent Agent

Adversarial Search Methods (Game Theory) - Mini max algorithm - Alpha beta pruning - Constraint satisfactory problems – Constraints – Crypt Arithmetic Puzzles – Constraint Domain – CSP as a search problem (Room colouring). Intelligent Agent – Rationality and Rational Agent – Performance Measures – Rationality and Performance – Flexibility and Intelligent Agents – Task environment and its properties – Types of agents.

Adversarial Search Methods (Game Theory)

Adversarial search methods, also known as game theory, involve finding optimal strategies for decision-making in competitive situations.

Key Concepts:

1. Game Tree: A tree representation of a game, where each node represents a game state and edges represent possible moves.

2. Minimax Algorithm: A recursive algorithm that finds the best move by considering the minimum possible loss (MIN) and maximum possible gain (MAX).

3. Alpha-Beta Pruning: An optimization technique that reduces the number of nodes to be evaluated in the game tree.

Adversarial Search Methods:

1. Minimax Search: A search algorithm that uses the minimax algorithm to find the best move.

2. Alpha-Beta Search: A search algorithm that uses alpha-beta pruning to optimize the minimax search.

Applications:

1. Chess: Adversarial search methods are used in chess engines to find the best moves.

2. Other Games: Adversarial search methods are used in various games, such as checkers, Go, and poker.

3. Decision-Making: Adversarial search methods can be applied to decision-making problems in fields like economics, finance, and politics.

Benefits:

1. Optimal Decision-Making: Adversarial search methods can find optimal strategies for decision-making in competitive situations.

2. Improved Performance: Adversarial search methods can improve performance in games and other competitive situations.

3. Strategic Thinking: Adversarial search methods promote strategic thinking and planning.

Challenges:

1. Computational Complexity: Adversarial search methods can be computationally expensive, especially for large game trees.

2. Game Tree Size: The size of the game tree can be enormous, making it challenging to search efficiently.

3. Heuristics: Developing effective heuristics to guide the search can be difficult.

Minimax algorithm

The Minimax algorithm is a recursive algorithm used for decision-making in games like chess, checkers, and other two-player games.

How it Works:

1. Game Tree: The algorithm constructs a game tree, where each node represents a game state and edges represent possible moves.

2. Minimax: The algorithm evaluates each node in the game tree, considering the best possible move for the maximizing player (MAX) and the worst possible move for the minimizing player (MIN).

3. Recursion: The algorithm recursively explores the game tree, evaluating each node and its children.

4. Evaluation Function: The algorithm uses an evaluation function to assign a score to each node, representing the desirability of the game state.

Minimax Algorithm Steps:

1. Initialize: Initialize the game tree and the evaluation function.

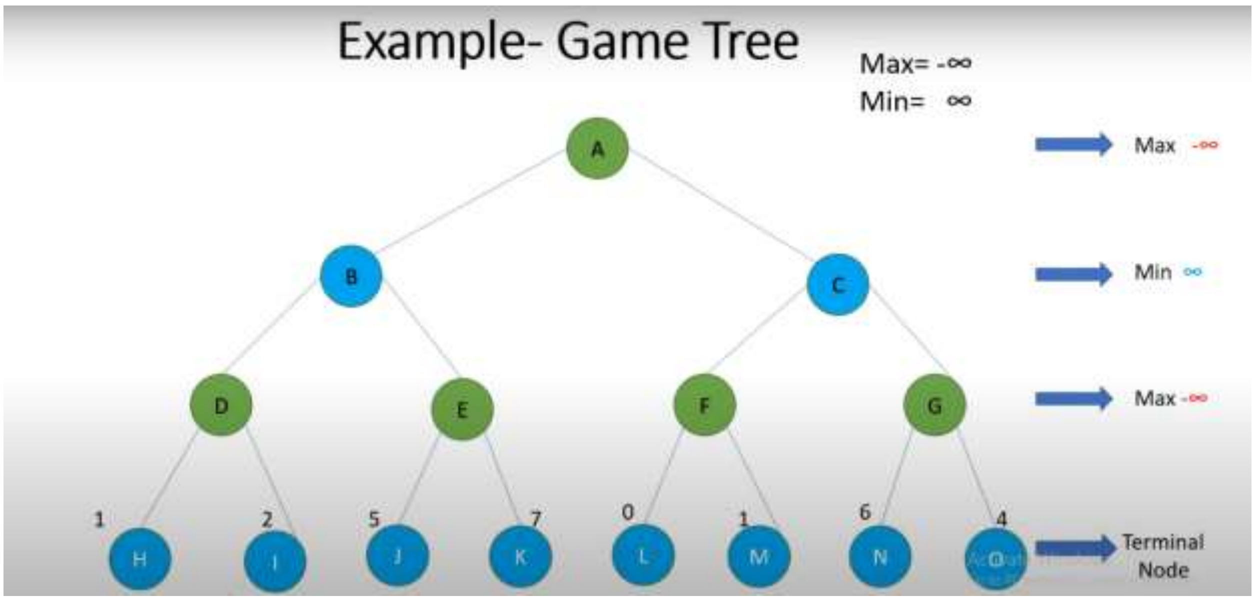
2. Explore: Recursively explore the game tree, evaluating each node and its children.

3. Evaluate: Evaluate each node using the evaluation function.

4. Back-propagate: Back-propagate the scores from the leaf nodes to the root node.

5. Choose: Choose the move that maximizes the score.

Example: Solve the Given Game Tree using Minimax algorithm



Algorithm for Solution

Node D: max(1, - ∞) => max(1,2) = 2

Update node value of D =2

Node E: max(5, - ∞) => max (5,7) = 7

Update node value of E =7

Node F: max(0, - ∞) => max (0,1) = 1

Update node value of F =1

Node G: max(6, - ∞) => max (6,4) = 6

Update node value of G =6

Node B: min(2, ∞) => min(2,7) = 2

Update node value of B =2

Node C: min(1, ∞) => min(1,6) = 1

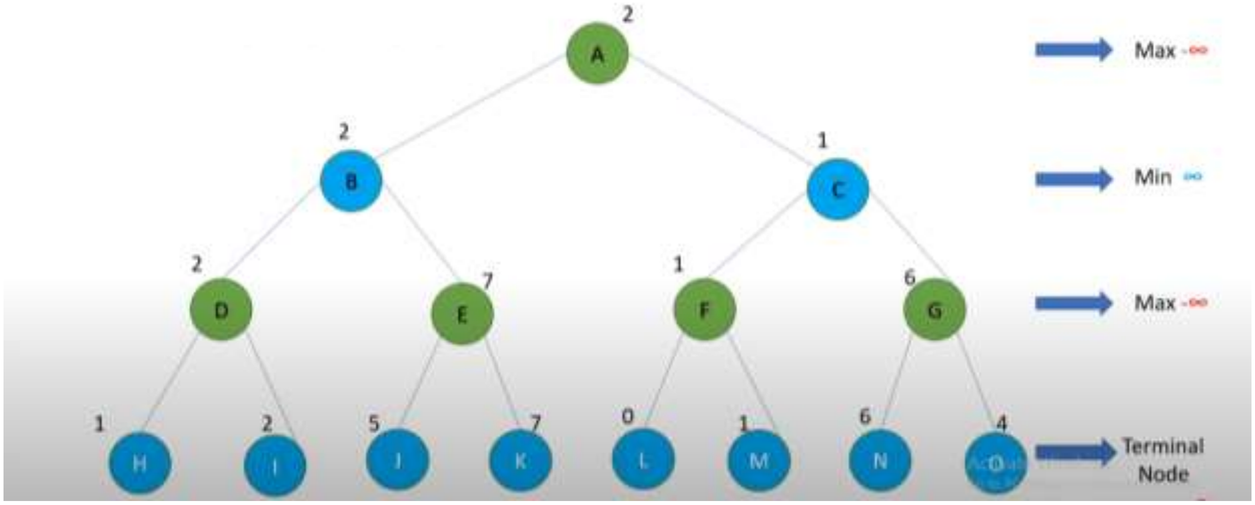
Update node value of C =1

Node A: max(2, - ∞) => max (2,1) = 2

Update node value of A =2

Solution:

Optimal Path: A-B-D-I



Benefits:

1. Optimal Decision-Making: The Minimax algorithm can make optimal decisions in games with perfect information.

2. Strategic Thinking: The algorithm promotes strategic thinking and planning.

Limitations:

1. Computational Complexity: The Minimax algorithm can be computationally expensive, especially for large game trees.

2. Game Tree Size: The size of the game tree can be enormous, making it challenging to search efficiently.

Optimizations:

1. Alpha-Beta Pruning: A technique that reduces the number of nodes to be evaluated in the game tree.

2. Heuristics: Using heuristics to guide the search and reduce the number of nodes to be evaluated.

Applications:

1. Chess Engines: The Minimax algorithm is used in chess engines to make decisions.

2. Other Games: The algorithm is used in various games, such as checkers, Go, and poker.

3. Decision-Making: The Minimax algorithm can be applied to decision-making problems in fields like economics, finance, and politics.

Alpha Beta Pruning

Alpha-beta pruning is an optimization technique used in the Minimax algorithm to reduce the number of nodes to be evaluated in the game tree.

How it Works:

1. Alpha (α): Alpha represents the best possible score for the maximizing player (MAX).

2. Beta (β): Beta represents the best possible score for the minimizing player (MIN).

3. Pruning: The algorithm prunes branches of the game tree that will not affect the final decision.

Alpha-Beta Pruning Steps:

1. Initialize: Initialize alpha and beta values.

2. Explore: Recursively explore the game tree, evaluating each node and its children.

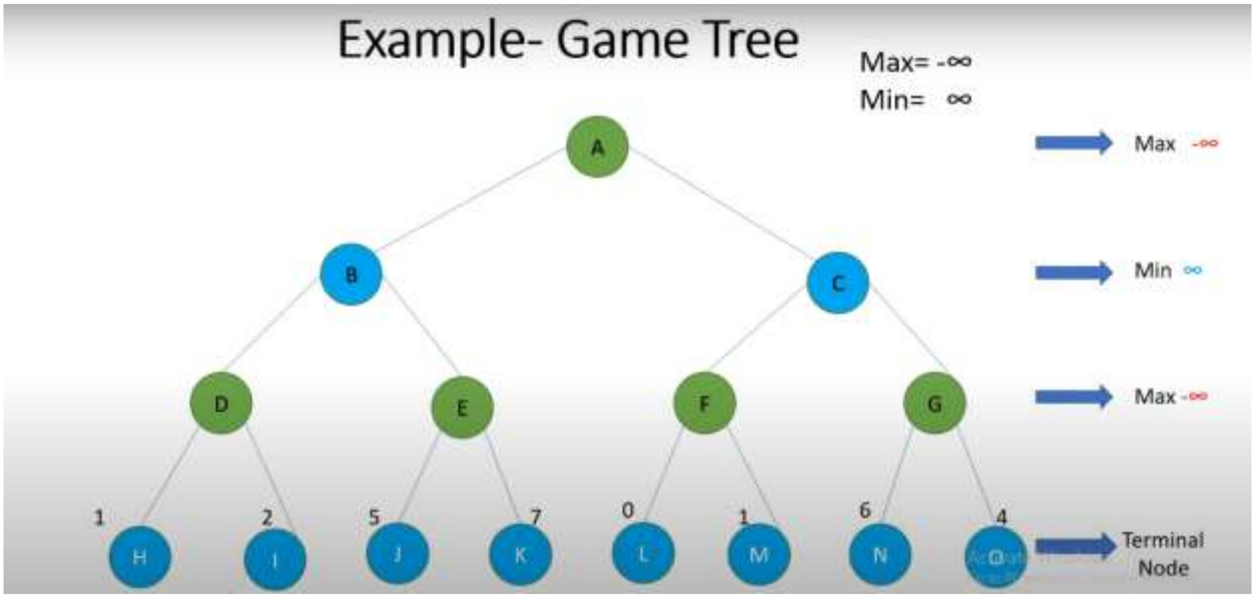
3. Prune: Prune branches that will not affect the final decision.

4. Update: Update alpha and beta values based on the evaluation.

When to Prune:

1. **$\alpha >= \beta$:** Prune the current branch if alpha is greater than or equal to beta.

Example: Solve the Given Game Tree using Alpha-beta pruning



<u>Algorithm for Solution</u>

Forward track from A to B to D,

Node D: max(1, - ∞) => max(1,2) = 2 = α

Update node value of D = 2

Backtrack to B,

Node B: min(2, ∞) = 2 = β

Update node value of B = 2

Forward track to E,

Node E: max(5, - ∞) = 5 = α

We got α = 5, β = 2

Check Condition: 5>2. So cut down the E to K branch.

Update node value of E = 5

Back track to B, update node value of B =2

Back track to A,

Node A: max(2, - ∞) = 2 = α

Update node value of A =2

Forward track from A to C to F,

Node F: max(2, 0) => max(2,1) = 2 = α

Update node value of F =1

Back track to C,

Node C: min(1, ∞) = 1 = β

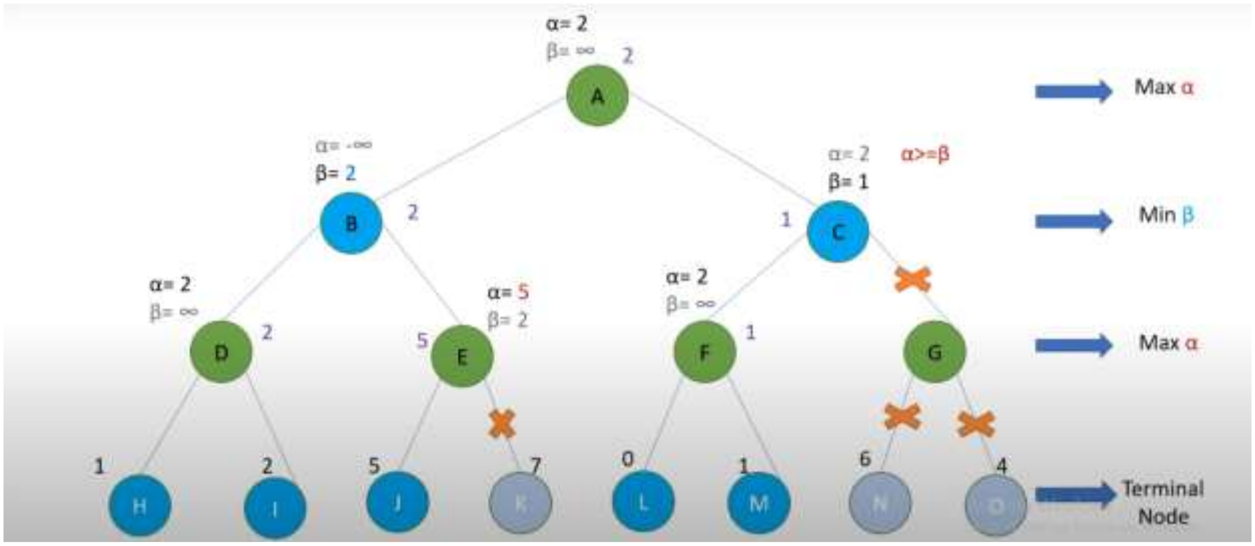
We got α = 2, β = 1

Check Condition: 2>1. So cut down the C to G branch.

Update node value of C =1

Solution:

Optimal Path: A-B-D-I



Benefits:

1. Reduced Computational Complexity: Alpha-beta pruning reduces the number of nodes to be evaluated.

2. Improved Efficiency: The algorithm is more efficient and can handle larger game trees.

Applications:

1. Chess Engines: Alpha-beta pruning is used in chess engines to optimize the Minimax algorithm.

2. Other Games: The technique is used in various games, such as checkers, Go, and poker.

3. Decision-Making: Alpha-beta pruning can be applied to decision-making problems in fields like economics, finance, and politics.

Constraint satisfactory problems

Constraint satisfaction problems (CSPs) are mathematical problems where the goal is to find a solution that satisfies a set of constraints.

Key Components:

1. Variables: A set of variables that need to be assigned values.

2. Domains: A set of possible values for each variable.

3. Constraints: A set of constraints that restrict the values of the variables.

Types of Constraints:

1. Unary Constraints: Constraints that involve a single variable.

2. Binary Constraints: Constraints that involve two variables.

3. Higher-Order Constraints: Constraints that involve more than two variables.

Examples:

1. Scheduling: Scheduling problems, such as scheduling classes or meetings.

2. Resource Allocation: Resource allocation problems, such as allocating resources to tasks.

3. Configuration: Configuration problems, such as configuring a product.

Solving CSPs:

1. Backtracking: A search algorithm that explores the solution space by assigning values to variables.

2. Constraint Propagation: A technique that reduces the search space by propagating the constraints.

3. Local Search: A search algorithm that starts with an initial solution and iteratively improves it.

Applications:

1. Artificial Intelligence: CSPs are used in artificial intelligence to model and solve complex problems.

2. Operations Research: CSPs are used in operations research to optimize resource allocation and scheduling.

3. Computer Science: CSPs are used in computer science to solve problems in areas like database systems and software engineering.

Benefits:

1. Flexibility: CSPs can model complex problems with multiple constraints.

2. Efficiency: CSPs can be solved efficiently using specialized algorithms.

3. Scalability: CSPs can be applied to large-scale problems.

Challenges:

1. Complexity: CSPs can be computationally expensive to solve.

2. Constraint Satisfaction: Finding a solution that satisfies all constraints can be challenging.

3. Optimization: Optimizing the solution to minimize or maximize a objective function can be difficult.

Crypt Arithmetic Puzzles

Cryptarithmetic problems are a type of mathematical puzzle where numbers are represented by letters or symbols, and the goal is to decipher the code and find the numerical values of the letters.

Constraints:

- Every character must have a unique value
- Digits should be from 0 -9 only
- Starting character of number cannot be zero
- In case of addition of two numbers, if there is carry to next step then, the carry can be 1.

Example:

Solve the following cryptarithmetic problem:

```
  SEND
+ MORE
  ------
 MONEY
```

Each letter represents a digit from 0 to 9, and each letter has a unique digit. The goal is to find the numerical values of the letters that make the equation true.

Solution:

Variable: S, E, N, D, M, O, R, Y

Domain: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Constraints:

- Every character must have a unique value
- Digits should be from 0 -9 only
- Starting character of number cannot be zero
- In case of addition of two numbers, if there is carry to next step then, the carry can be 1.

One possible solution is:

	S	**E**	**N**	**D**
(+)	**M**	**O**	**R**	**E**
M	**O**	**N**	**E**	**Y**

S	**9**
E	**5**
N	**6**
D	**7**
M	**1**
O	**0**
R	**8**
Y	**2**

Substituting these values into the equation, we get:

	9	**5**	**6**	**7**
(+)	**1**	**0**	**8**	**5**
1	**0**	**6**	**5**	**2**

Characteristics:

1. Unique Digit Assignment: Each letter has a unique digit.
2. Carry Propagation: The solution must account for carry propagation in the addition.

Solving Techniques:

1. Constraint Propagation: Using constraints to reduce the search space.
2. Backtracking: Exploring the solution space by assigning values to letters.

Applications:

1. Mathematical Puzzles: Crypt arithmetic problems are used as mathematical puzzles.

2. Code breaking: Crypt arithmetic problems can be used to introduce concepts related to code breaking.

Benefits:

1. Improved Problem-Solving Skills: Solving crypt arithmetic problems can improve problem-solving skills.

2. Logical Reasoning: Crypt arithmetic problems require logical reasoning and deduction.

Constraint satisfaction coloring map problem

The Map Coloring Problem is a classic example of a Constraint Satisfaction Problem (CSP). The goal is to color a map such that no two adjacent regions have the same color.

Problem Definition:

Given a map with regions (e.g., countries, states), assign a color to each region such that:

1. No adjacent regions have the same color.
2. Each region is assigned a color from a predefined set of colors.

Constraints:

1. Adjacency constraints: No two adjacent regions can have the same color.

2. Color constraints: Each region must be assigned a color from the predefined set.

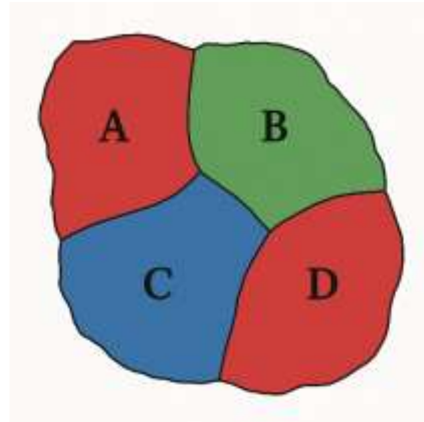
Example:

Suppose we have a map with four regions: A, B, C, and D. The adjacency relationships are:

- A is adjacent to B and C
- B is adjacent to A and D
- C is adjacent to A and D
- D is adjacent to B and C

We want to color the map using three colors: red, green, and blue.

Solution:



One possible solution is:

- A: red
- B: green
- C: blue
- D: red

This solution satisfies the constraints:

- No adjacent regions have the same color.
- Each region is assigned a color from the predefined set.

Solving Techniques:

1. Backtracking: Exploring the solution space by assigning colors to regions.

2. Constraint Propagation: Reducing the search space by propagating the constraints.

Applications:

1. Map coloring: The Map Coloring Problem has applications in cartography and geography.

2. Scheduling: Similar problems arise in scheduling, where resources need to be allocated without conflicts.

3. Resource allocation: The problem is relevant to resource allocation in various domains.

Benefits:

1. Improved problem-solving skills: Solving the Map Coloring Problem can improve problem-solving skills.

2. Logical reasoning: The problem requires logical reasoning and deduction.

The Map Coloring Problem is a fundamental problem in computer science and operations research, and its solution has far-reaching implications in various fields.

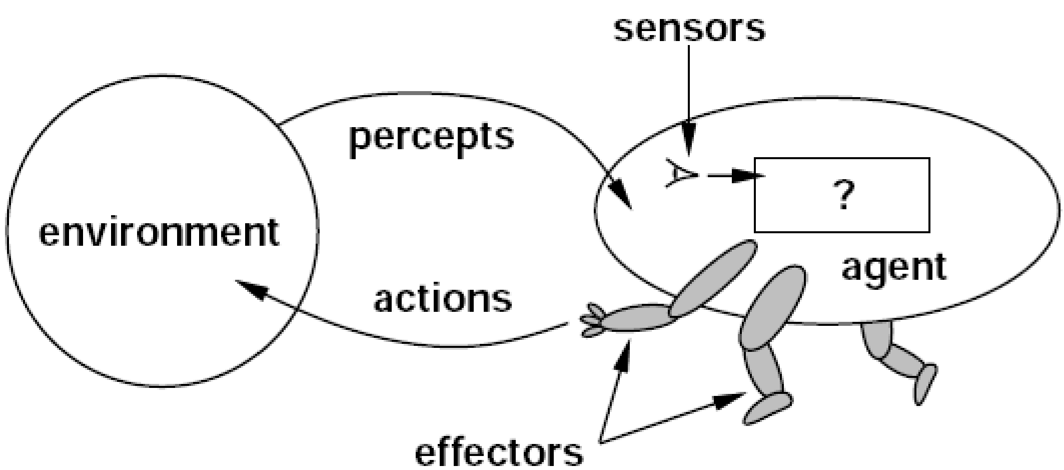
Intelligent Agent

An intelligent agent is a computer system that can perceive its environment, make decisions, and take actions to achieve its goals.

Example:

A robot vacuum cleaner is an agent.

It perceives its surroundings (the room), decides where to go, and acts by moving to cleaning the floor.



Architecture of an Agent

The architecture of an agent typically consist of three main components:

1. **Perception**: The agent receives information from the environment (using sensors).
2. **Decision Making:** The agent decides what action to be take based on the perception (processing and reasoning)
3. **Action:** The agent performs an action in the environment (using actuators).
4. **Feedback:** The environment may change based on the agents action, and the agent perceives this feedback, which can influence future actions.

Example:

In the robot vacuum cleaner,

- The sensors perceives the obstacles
- The decision-making component decides where to move
- The actuator make it move or change direction

Characteristics:

1. Autonomy: Intelligent agents can operate independently without human intervention.

2. Reactivity: Intelligent agents can perceive their environment and respond to changes.

3. Proactivity: Intelligent agents can take initiative and act to achieve their goals.

4. Goal-oriented: Intelligent agents have goals and can plan to achieve them.

Applications:

1. Robotics: Intelligent agents are used in robotics to control robots that can perform tasks autonomously.

2. Virtual Assistants: Intelligent agents are used in virtual assistants like Siri, Alexa, and Google Assistant.

3. Game Playing: Intelligent agents are used in game playing to create opponents that can play games like chess or Go.

4. Recommendation Systems: Intelligent agents are used in recommendation systems to suggest products or services based on user preferences.

Benefits:

1. Autonomy: Intelligent agents can operate independently, freeing humans from routine tasks.

2. Efficiency: Intelligent agents can optimize processes and improve efficiency.

3. Personalization: Intelligent agents can personalize experiences based on user preferences.

Challenges:

1. Complexity: Building intelligent agents that can handle complex environments and tasks.

2. Uncertainty: Dealing with uncertainty and ambiguity in the environment.

3. Ethics: Ensuring that intelligent agents are designed and used in ways that are ethical and responsible.

Rationality and Rational Agent

Rationality and rational agents are fundamental concepts in artificial intelligence, economics, and decision theory.

Rationality:

Rationality refers to the ability of an agent to make decisions that maximize its expected utility or achieve its goals. A rational agent is one that:

1. Has well-defined goals: The agent has clear and consistent goals.

2. Makes decisions based on evidence: The agent makes decisions based on available evidence and reasoning.

3. Maximizes expected utility: The agent chooses actions that maximize its expected utility or achieve its goals.

Rational Agent:

A rational agent is an agent that acts rationally, making decisions that maximize its expected utility or achieve its goals. Rational agents:

1. Perceive their environment: The agent perceives its environment and gathers information.

2. Make decisions: The agent makes decisions based on its goals, preferences, and available information.

3. Act: The agent takes actions to achieve its goals.

Types of Rationality:

1. Perfect rationality: The agent has complete knowledge and makes optimal decisions.

2. Bounded rationality: The agent has limited knowledge and computational resources, and makes decisions based on heuristics and approximations.

Applications:

1. Decision-making: Rational agents are used in decision-making systems to make optimal decisions.

2. Game playing: Rational agents are used in game playing to make strategic decisions.

3. Economics: Rational agents are used in economics to model economic behavior and decision-making.

Benefits:

1. Optimal decision-making: Rational agents can make optimal decisions that maximize expected utility.

2. Efficient resource allocation: Rational agents can allocate resources efficiently to achieve their goals.

3. Improved performance: Rational agents can improve performance in complex environments.

Challenges:

1. Complexity: Building rational agents that can handle complex environments and tasks.

2. Uncertainty: Dealing with uncertainty and ambiguity in the environment.

3. Computational limitations: Overcoming computational limitations to make optimal decisions.

PEAS Representation

PEAS (Performance measure, Environment, Actuators, and Sensors) is a representation used to describe the components of an intelligent agent. Here's a PEAS representation for a vacuum cleaning robot:

Example: PEAS Representation for vacuum cleaning Robot:

Performance Measure:

- Cleanliness: The percentage of the floor area that is clean.

- Efficiency: The amount of time taken to clean the floor.

- Safety: The robot's ability to avoid obstacles and prevent accidents.

Environment:

- Indoor environment: The robot operates in a house or building with various rooms and surfaces.

- Floor types: The robot encounters different types of floors, such as hardwood, carpet, or tile.

- Obstacles: The robot may encounter obstacles like furniture, stairs, or walls.

Actuators:

- Vacuum motor: The robot's vacuum motor sucks up dirt and debris.

- Wheels or tracks: The robot's wheels or tracks allow it to move around the environment.

- Sensors and navigation system: The robot's sensors and navigation system enable it to detect and avoid obstacles.

Sensors:

- Infrared sensors: The robot uses infrared sensors to detect obstacles and stairs.

- Ultrasonic sensors: The robot uses ultrasonic sensors to detect objects and navigate.

- Camera: The robot may use a camera to detect dirt, debris, or obstacles.

- Bump sensors: The robot may use bump sensors to detect collisions.

Benefits:

The PEAS representation helps to:

1. Identify agent components: PEAS helps identify the key components of the vacuum cleaning robot.

2. Design and development: PEAS informs the design and development of the robot's hardware and software.

3. Testing and evaluation: PEAS provides a framework for testing and evaluating the robot's performance.

By using the PEAS representation, we can better understand the requirements and challenges of building an effective vacuum cleaning robot.

Performance Measure:

A performance measure is a way to evaluate the success of an intelligent agent in achieving its goals.

Types of Performance Measures:

1. **Objective performance measures:** Quantifiable measures that can be directly observed, such as cleanliness or time.

2. **Subjective performance measures:** Measures that are based on user feedback or satisfaction, such as user ratings or satisfaction surveys.

Example:

Let's consider a vacuum cleaning robot that operates in a house with multiple rooms. The performance measure could be:

- **Cleanliness:** The percentage of the floor area that is clean.

- **Time:** The time taken to clean the entire house.

- **Battery life:** The amount of time the robot can operate on a single charge.

Suppose we have a vacuum cleaning robot that can:

- Clean 90% of the floor area in 2 hours.

- Operate for 4 hours on a single charge.

The performance measure could be:

- Cleanliness: 90%

- Time: 2 hours

- Battery life: 4 hours

Evaluation:

The performance measure can be used to evaluate the robot's performance and identify areas for improvement. For example:

- If the robot is not cleaning 100% of the floor area, we might need to adjust its navigation algorithm or increase its suction power.

- If the robot is taking too long to clean the house, we might need to optimize its route planning or increase its speed.

Performance measures are essential for:

1. **Evaluating agent performance:** Performance measures help evaluate the success of an intelligent agent in achieving its goals.

2. **Improving agent performance:** Performance measures can identify areas for improvement and guide the development of more effective agents.

3. **Comparing agents:** Performance measures can be used to compare the performance of different agents or algorithms.

Task Environment:

A task environment refers to the external environment in which an intelligent agent operates and performs its tasks.

Properties of Task Environment:

1. **Fully observable vs. partially observable:** Is the agent able to observe the entire state of the environment, or only a part of it?

2. **Deterministic vs. stochastic:** Is the outcome of the agent's actions deterministic, or is there uncertainty and randomness?

3. **Episodic vs. sequential:** Does the agent's experience consist of a series of independent episodes, or is there a sequence of actions and observations?

4. **Static vs. dynamic:** Does the environment change over time, or is it static?

5. **Discrete vs. continuous:** Is the environment discrete (e.g., a grid world) or continuous (e.g., a real-world environment)?

Task Environment Examples:

1. **Vacuum cleaning robot:** A robot that navigates and cleans a house.

   - **Fully observable:** No (the robot may not be able to see the entire house).

   - **Deterministic:** No (the robot's actions may be affected by uncertainty).

   - **Sequential:** Yes (the robot's actions are sequential).

   - **Dynamic:** Yes (the environment changes as the robot cleans).

   - **Continuous:** Yes (the robot operates in a continuous environment).

2. **Chess game:** A computer program that plays chess.

   - **Fully observable:** Yes (the program can see the entire board).

   - **Deterministic:** Yes (the outcome of moves is deterministic).

   - **Sequential:** Yes (the game is a sequence of moves).

   - **Static:** No (the board changes after each move).

   - **Discrete:** Yes (the board is a discrete grid).

Summary

Understanding the properties of a task environment is crucial for:

1. **Designing intelligent agents:** Agents need to be designed to operate effectively in their task environment.

2. **Choosing algorithms:** The properties of the task environment influence the choice of algorithms and techniques.

3. **Evaluating performance:** The properties of the task environment affect the evaluation of an agent's performance.

By analyzing the properties of a task environment, we can better design and develop intelligent agents that can operate effectively in complex environments.

Types of Agents

There are several types of agents, each with its own characteristics and applications:

1. Simple Reflex Agents:

- React to the current state of the environment without considering future consequences.

- Make decisions based on condition-action rules.

- Example: A thermostat that turns on/off based on temperature.

2. Model-Based Reflex Agents:

- Use a model of the environment to make decisions.

- Consider the current state and predicted future states.

- It uses internal memory to remember past

- Example: A self-driving car that uses sensors and mapping data to navigate.

3. Goal-Based Agents:

- Have specific goals and make decisions to achieve them.

- Use planning and problem-solving to achieve goals.

- Example: A robot that navigates to a specific location.

4. Utility-Based Agents:

- Make decisions based on a utility function that estimates the desirability of outcomes.

- Choose actions that maximize expected utility.

- Example: A recommendation system that suggests products based on user preferences.

5. Learning Agents:

- Can learn from experience and adapt to new situations.

- Use machine learning algorithms to improve performance.

- Example: A spam filter that learns to recognize spam emails.

6. Hybrid Agents:

- Combine different types of agents (e.g., reflex and goal-based).

- Use multiple approaches to make decisions.

- Example: A robot that uses reflex actions for obstacle avoidance and goal-based planning for navigation.