Unit-2-Problem Solving

Solving Problems by Searching: Informed and Uninformed Search

1. Introduction to Problem-Solving by Searching

In **Artificial Intelligence (AI)**, problem-solving is often performed by searching through a **state space** to find an optimal or acceptable solution. Search algorithms explore possible solutions **step by step**, evaluating the best path to reach a goal.

There are **two main types of search algorithms**:

1. **Uninformed Search (Blind Search)** – No prior knowledge, explores all possibilities.
2. **Informed Search (Heuristic Search)** – Uses additional knowledge (heuristics) to find solutions efficiently.

---

2. Uninformed Search (Blind Search)

- **Definition**: Uninformed search algorithms explore the state space **without using any problem-specific knowledge**.
- **Characteristics**:
    - No heuristic function is used.
    - Explores all possible states systematically.
    - Less efficient than informed search.

2.1 Types of Uninformed Search

Search Type	Description	Time Complexity	Space Complexity	Completeness	Optimality
Breadth-First Search (BFS)	Explores all nodes at the current level before moving to the next level.	O(b^d)	O(b^d)	□ Yes	□ Yes (if all step costs are equal)

Search Type	Description	Time Complexity	Space Complexity	Completeness	Optimality
Depth-First Search (DFS)	Explores as deep as possible before backtracking.	O(b^d)	O(d)O(d)O(d)	□ Yes (if finite)	□ No
Depth-Limited Search (DLS)	DFS with a depth limit to prevent infinite loops.	O(b^l)	O(l)	□ No (if depth limit is too small)	□ No
Iterative Deepening DFS (IDDFS)	Combines DFS and BFS by gradually increasing depth.	O(b^d)	O(d)	□ Yes	□ Yes
Uniform Cost Search (UCS)	Explores paths based on **cumulative path cost** (like Dijkstra's Algorithm).	O(b^d)	O(b^d)	□ Yes	□ Yes

2.2 Example of Uninformed Search

□ **Breadth-First Search (BFS) Example:**
Consider a **maze-solving problem** where a robot must find the shortest path to a goal. BFS will explore all possible movements **level by level**, ensuring the shortest path is found first.

---

3. Informed Search (Heuristic Search)

- **Definition**: Informed search algorithms use **heuristic (additional) knowledge** to guide the search process toward the goal faster.
- **Characteristics**:

- Uses a heuristic function **h(n)** to estimate the cost from a node to the goal.
- More efficient than uninformed search.
- Finds solutions faster and optimally in many cases.

3.1 Types of Informed Search

Search Type	Description	Time Complexity	Space Complexity	Completeness	Optimality
Greedy Best-First Search	Chooses the path that appears closest to the goal based on heuristics.	O(b^d)	O(b^d)	□ Yes	□ No
*A Search Algorithm**	Uses both path cost and heuristic to find the optimal path.	O(b^d)	O(b^d)	□ Yes	□ Yes
Hill Climbing	Moves in the direction of increasing value, but may get stuck in local optima.	Varies	Varies	□ No (Stuck in local maxima)	□ No
Beam Search	Similar to BFS but keeps track of only the best paths.	Varies	Varies	□ Yes	□ No

3.2 Example of Informed Search

□ _A Search Example (Pathfinding in Google Maps):*_
A* Search is used in **Google Maps** to find the shortest route between two locations. It considers both:

- **g(n):** Distance traveled so far.
- **h(n):** Estimated distance remaining (heuristic, like straight-line distance).

---

4. Comparison of Uninformed & Informed Search

Feature	Uninformed Search	Informed Search
Uses Heuristic?	□ No	□ Yes
Efficiency	□ Less efficient	□ More efficient
Time Complexity	Higher (explores more states)	Lower (guides search effectively)
Space Complexity	Higher (stores all nodes)	Lower (stores only relevant paths)
Optimality	□ Sometimes (e.g., UCS)	□ Often (e.g., A*)
Example Algorithm	BFS, DFS, UCS	A*, Greedy Best-First, Hill Climbing

Constraint Satisfaction Problems (CSPs) in Detail

1. Introduction to CSPs

A **Constraint Satisfaction Problem (CSP)** is a mathematical problem defined by a set of **variables**, a **domain** of values for each variable, and a set of **constraints** that specify allowed combinations of values. The goal is to find an assignment of values to variables that satisfies all constraints.

Example:

- **Variables:** $X=\{X1,X2,X3\}$
- **Domains:** $X1\in\{1,2,3\},X2\in\{1,2,3\},X3\in\{1,2,3\}$
- **Constraints:** x1 != x2 , x2 != x3

In this case, a valid solution is (X1=1,X2=2,X3=3)

---

2. Components of CSPs

A CSP consists of the following elements:

2.1 Variables (X)

A finite set of variables X={X1,X2,...,Xn}
Example: In a Sudoku puzzle, each cell in the grid is a variable.

2.2 Domains (D)

Each variable Xi has a set of possible values, called its domain Di
Example: In Sudoku, the domain of each variable is {1,2,3,4,5,6,7,8,9}

2.3 Constraints (C)

A set of rules that restrict possible value assignments to variables.
Example: In Sudoku, each row must contain unique numbers from 1 to 9.

---

3. Types of CSPs

3.1 Binary CSPs

- Constraints involve at most **two variables**.
- **Example:** Graph coloring, where no two adjacent nodes can have the same color.

3.2 Unary CSPs

- Constraints involve only **one variable**.
- **Example:** A cell in a Sudoku grid cannot have a number outside {1-9}.

3.3 Higher-order CSPs

- Constraints involve **three or more variables**.

- **Example:** In a scheduling problem, three employees cannot be assigned to the same shift.

---

4. Solving CSPs

CSPs can be solved using various methods:

4.1 Backtracking Algorithm

- A **recursive** method that assigns values to variables one by one.
- If a **conflict** occurs, it **backtracks** to the previous step.
- **Example:** Solving Sudoku by filling one cell at a time and checking constraints.

4.2 Forward Checking

- After assigning a value to a variable, it **eliminates inconsistent values** from the domains of remaining variables.
- Helps reduce the search space and speed up backtracking.

4.3 Arc Consistency (AC-3 Algorithm)

- Ensures that for every variable, there exists at least one valid value for the connected variables.
- Used to **prune** invalid values early.
- **Example:** Checking that every cell in Sudoku has at least one valid number left.

4.4 Constraint Propagation

- **Reduces the search space** by **propagating constraints** across variables.
- **Example:** In a crossword puzzle, filling one letter restricts choices for adjacent words.

---

5. Example CSP Applications

CSPs are widely used in AI, optimization, and planning problems.

Application	Example
Sudoku Solving	Each cell is a variable, numbers 1-9 are domains, row/column constraints apply.
Graph Coloring	Assigning colors to a map so no adjacent regions share the same color.
Timetable Scheduling	Assigning time slots to classes while avoiding conflicts.
Cryptarithmetic Puzzles	Solving puzzles like SEND + MORE = MONEY where each letter represents a unique digit.
Job Scheduling	Assigning tasks to workers based on constraints like time availability.

Adversarial Search in Detail

1. Introduction to Adversarial Search

In many real-world problems, an **AI agent** competes against an **opponent**, meaning the environment is dynamic and unpredictable. **Adversarial Search** is used in such competitive settings where multiple agents (players) have **conflicting goals**.

□ **Example:** Chess, Tic-Tac-Toe, and other two-player games where each player tries to defeat the opponent.

Unlike normal search problems (like finding the shortest path), adversarial search involves:

- **Maximizing the agent's gain** while
- **Minimizing the opponent's gain** (since the opponent is trying to defeat the agent).

---

2. Characteristics of Adversarial Search

- **Multi-agent environment** (multiple players involved).
- **Competitive nature** (players have conflicting goals).
- **Turn-based decision-making** (each player takes turns).

- **Game-tree representation** (possible moves are structured in a tree).
- **Zero-sum assumption** (one player’s gain is the other player’s loss).

---

3. Game Representation in Adversarial Search

A game can be defined by the following components:

1. **Initial State** – Describes the starting position of the game.
2. **Players** – A set of players (e.g., MAX and MIN in a two-player game).
3. **Actions** – A set of possible moves each player can take.
4. **Transition Function** – Defines how the game moves from one state to another.
5. **Terminal Test** – Checks if the game has ended (win, loss, or draw).
6. **Utility Function** – Assigns a numerical value to terminal states (e.g., +1 for a win, -1 for a loss, 0 for a draw).

 Example: In Chess, the **initial state** is the starting board position, **players** are White and Black, **actions** are legal moves, and the **utility function** can be based on piece values.

---

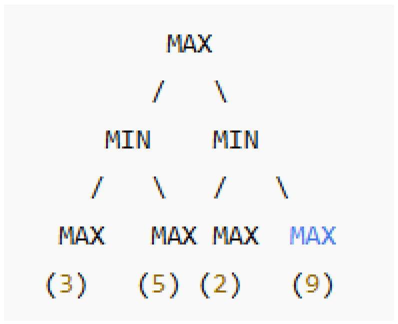
4. Minimax Algorithm (Basic Adversarial Search)

The **Minimax algorithm** is a fundamental technique used for decision-making in two-player zero-sum games.

4.1 How Minimax Works

- **MAX Player** (tries to **maximize** the score).
- **MIN Player** (tries to **minimize** the score).
- The game is represented as a **tree**, where:
  - **MAX nodes** choose the highest value (best for MAX).
  - **MIN nodes** choose the lowest value (worst for MAX, best for MIN).
- The game proceeds until a **terminal state** is reached.

 Example of Minimax Game Tree (Tic-Tac-Toe)



```
        MAX
       /   \
     MIN    MIN
    /   \  /   \
  MAX   MAX MAX  MAX
  (3)   (5) (2)   (9)
```

- **MAX chooses the highest value** among child nodes.
- **MIN chooses the lowest value** among child nodes.

4.2 Minimax Algorithm Steps

1. Generate the **game tree** up to a depth limit or until a terminal state.
2. Evaluate the **utility function** for leaf nodes.
3. Apply the **Minimax decision** by propagating values from leaf nodes to the root.
4. Choose the move with the highest value for MAX.

Time Complexity: O(b^d), where:

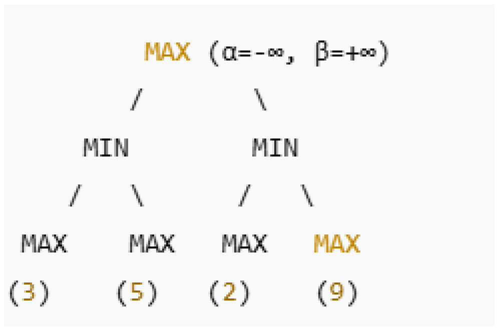
- b = branching factor (number of moves per state).
- d = depth of the game tree.

5. Alpha-Beta Pruning (Optimized Minimax)

Minimax explores the entire tree, making it **slow** for large games. **Alpha-Beta Pruning** improves performance by eliminating unnecessary nodes.

5.1 How Alpha-Beta Pruning Works

- Uses two values:
  - **Alpha (α)** → Best choice so far for **MAX**.
  - **Beta (β)** → Best choice so far for **MIN**.
- If a branch **cannot** produce a better result than an already explored path, it is **pruned** (ignored).



```
        MAX (α=-∞, β=+∞)
       /       \
     MIN        MIN
    /   \      /   \
 MAX    MAX   MAX   MAX
(3)    (5)   (2)    (9)
```

 Suppose **MIN (left subtree) picks 3**.
  If **MAX has an option better than 3 elsewhere**, the remaining moves **don't need to be explored**.

Time Complexity (Best Case): O(b^{d/2}) (almost **half** of minimax).

6. Other Adversarial Search Techniques

6.1 Expectimax Algorithm

- Used in **games with uncertainty** (e.g., dice-based games).
- Instead of MIN, uses **expected values** for chance nodes.
- Example: Pac-Man AI predicting **ghost movements**.

6.2 Monte Carlo Tree Search (MCTS)

- Used in **large games like Go**.
- Uses **random simulations** to evaluate moves.
- Used in **AlphaGo** (Google's AI for playing Go).

---

7. Real-World Applications of Adversarial Search

Application	Description
Chess Engines	AI-powered chess bots like Stockfish use Minimax with Alpha-Beta Pruning.
Self-Driving Cars	Decision-making against obstacles (other "players" on the road).

Application	Description
Robotic Soccer	AI-controlled robots competing in RoboCup.
AI in Video Games	NPC enemies using adversarial search for strategy.
Cybersecurity	AI defending against cyber threats using adversarial strategies.

Knowledge and Reasoning: Knowledge Representation & First-Order Logic (FOL)

---

1. Introduction to Knowledge and Reasoning

Knowledge Representation and Reasoning (KRR) is a fundamental area in Artificial Intelligence (AI) that deals with how information is stored, processed, and used for decision-making.

□ **Knowledge Representation (KR)** – Defines how AI systems store and structure knowledge.
□ **Reasoning** – The process of deriving new facts from known knowledge.
□ **First-Order Logic (FOL)** – A powerful formal system for expressing knowledge in AI.

□ **Example:** A self-driving car uses KR to store traffic rules and reasoning to decide when to stop at a red light.

---

2. Knowledge Representation (KR)

2.1 What is Knowledge Representation?

Knowledge Representation is a method of encoding knowledge about the world into a form that AI systems can understand and process.

□ It allows **AI systems to make decisions**, answer questions, and solve problems.
□ Knowledge is stored in **symbols, rules, graphs, and logic-based systems**.

2.2 Types of Knowledge in AI

1. **Declarative Knowledge** – "Knowing that" (facts, statements).
    Example: "The sun rises in the east."
2. **Procedural Knowledge** – "Knowing how" (steps, procedures).
    Example: "To drive a car, press the accelerator and steer."
3. **Heuristic Knowledge** – Knowledge gained from experience (rules of thumb).
    Example: "If the road is wet, drive slowly."
4. **Meta-Knowledge** – Knowledge about knowledge.
    Example: "Mathematics is based on axioms and theorems."
5. **Common-Sense Knowledge** – General knowledge humans have.
    Example: "Water is wet, and fire is hot."

2.3 Approaches to Knowledge Representation

1. **Logical Representation (FOL, Propositional Logic)** – Uses rules and logical reasoning.
2. **Semantic Networks** – Uses graphs where nodes represent concepts and edges represent relationships.
3. **Frames** – Data structures for representing knowledge in objects.
4. **Production Rules** – "If-Then" rules for decision-making.
5. **Ontologies** – A structured way to represent domain knowledge.

 Example: In a medical expert system, knowledge about diseases, symptoms, and treatments is stored using KR techniques.

3. First-Order Logic (FOL)

3.1 What is First-Order Logic?

First-Order Logic (FOL) is an extension of **Propositional Logic** that allows for **quantifiers, predicates, and objects.**

🔹 Unlike **Propositional Logic**, which deals with simple true/false statements, FOL allows **relationships between objects.**

📌 *Example:*

- **Propositional Logic:** "John is a doctor." → P (Simple statement)
- **First-Order Logic:** "All doctors treat patients." → $\forall x.Doctor(x) \Rightarrow TreatsPatients(x)$

3.2 Syntax of FOL

FOL has **symbols, functions, predicates, and quantifiers** to express complex knowledge.

1️⃣ Constants

- Represent **specific objects** in the domain.
- **Example:** `"John"`, `"Apple"`, `"Paris"`

2️⃣ Variables

- Represent **general objects** (used with quantifiers).
- **Example:** `x`, `y`, `z`

3️⃣ Predicates

- Represent **relationships** between objects.
- **Example:** `Doctor(John)`, `Loves(Alice, Bob)`, `Greater(5, 3)`

4️⃣ Functions

- Represent **properties** of objects.
- **Example:** `Father(John) → David` (David is John's father)

5️⃣ Quantifiers

- **Universal Quantifier (∀)** → "For all"
  📌 *Example:* `∀x. Human(x) → Mortal(x)` → "All humans are mortal."
- **Existential Quantifier (∃)** → "There exists at least one"
  📌 *Example:* `∃x. Animal(x) ∧ HasWings(x)` → "There exists an animal that has wings."

3.3 Semantics of FOL

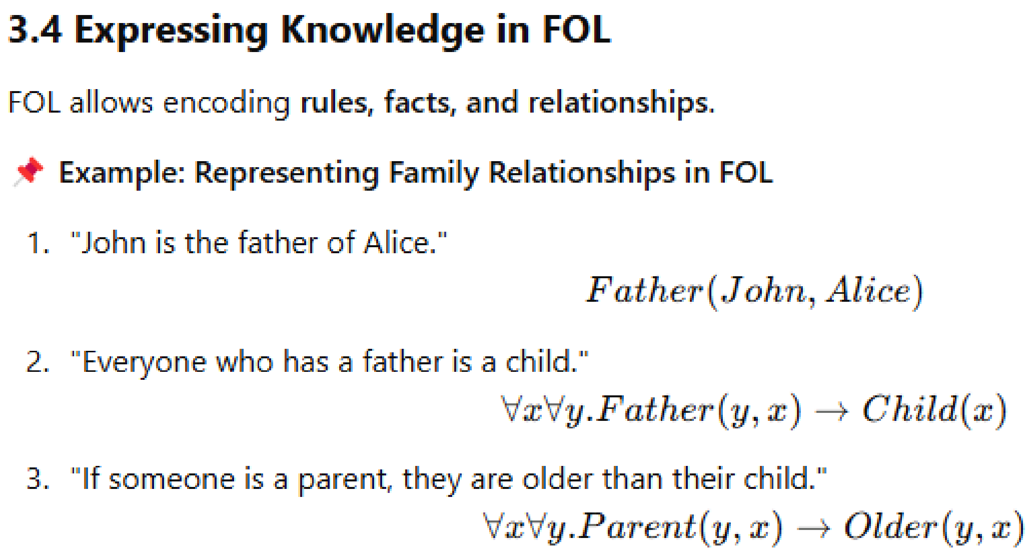
Semantics define **how FOL statements are interpreted.**

📌 Example Interpretation:

- `Loves(Alice, Bob) → True` (Alice loves Bob)
- `∀x. Mammal(x) → WarmBlooded(x)` (All mammals are warm-blooded)

📌 Negation Example:

- `¬Rich(John)` → "John is not rich."



3.4 Expressing Knowledge in FOL

FOL allows encoding **rules, facts, and relationships.**

📌 **Example: Representing Family Relationships in FOL**

1. "John is the father of Alice."
$$Father(John, Alice)$$
2. "Everyone who has a father is a child."
$$\forall x \forall y . Father(y, x) \rightarrow Child(x)$$
3. "If someone is a parent, they are older than their child."
$$\forall x \forall y . Parent(y, x) \rightarrow Older(y, x)$$

4. Reasoning in AI

Once knowledge is represented, **reasoning** is used to infer new facts.

4.1 Types of Reasoning

1. **Deductive Reasoning**
   - **General → Specific**
   - □ *Example:* "All humans are mortal. Socrates is a human. → Socrates is mortal."
2. **Inductive Reasoning**
   - **Specific → General**
   - □ *Example:* "Every swan I have seen is white. → All swans are white."
3. **Adductive Reasoning**
   - **Inference based on best explanation**
   - □ *Example:* "The grass is wet. It probably rained."
4. **Monotonic vs. Non-Monotonic Reasoning**
   - **Monotonic**: New knowledge **does not** change previous conclusions.
   - **Non-Monotonic**: New knowledge **can** change previous conclusions.

---

5. Applications of Knowledge Representation & FOL

Application	Description
Expert Systems	AI-powered medical diagnosis, legal advisory systems.
Chatbots & Virtual Assistants	AI understands and responds based on stored knowledge.
Autonomous Robots	AI robots use KR to make decisions (e.g., warehouse robots).
Search Engines	Google uses FOL-based ontologies to understand queries.
AI in Healthcare	AI diagnoses diseases based on symptoms using FOL.