Name: Prathamesh Arvind Jadhav

Unit-3- Planning

Planning with Forward and Backward State Space Search in AI

State-space search is a fundamental approach to problem-solving in Artificial Intelligence (AI), where we model problems as transitions between different states using actions. **Forward and backward state-space search** are two key methods used in AI planning.

---

1. Forward State-Space Search (Progression Planning)

Concept:

- Starts from the **initial state** and **applies actions** step by step until it reaches the **goal state**.
- Expands the **successor states** by applying valid operators (actions).

Steps:

1. **Start from Initial State**: The search begins at the given initial state.
2. **Apply Operators**: Apply available actions that are applicable in the current state.
3. **Generate New States**: These new states are added to the search tree.
4. **Check Goal Condition**: If a state matches the goal state, the search stops.
5. **Repeat Until Goal is Found**.

Example:

Imagine a **robot navigating a grid** from (0,0) to (3,3):

- Start at (0,0).
- Apply available actions: Move Right, Move Up.
- Expand new states (1,0), (0,1).
- Continue expanding until reaching (3,3).

Advantages:

✔ Works well when there are few states and deterministic transitions.
✔ Simple to implement using Breadth-First Search (BFS) or Depth-First Search (DFS).

Disadvantages:

✖ Can generate a **large search space**, making it inefficient.
✖ Often explores **irrelevant paths** that don't lead to the goal.

---

2. Backward State-Space Search (Regression Planning)

Concept:

- Starts from the **goal state** and works **backward** to find a path to the **initial state**.
- Determines what actions must have preceded the goal state.

Steps:

1. **Start from the Goal State**: Identify what needs to be true to reach the goal.
2. **Find Relevant Actions**: Identify which actions can produce the goal state.
3. **Determine Preconditions**: What conditions must be true before applying those actions?
4. **Expand Backward**: Continue expanding backward until the initial state is reached.
5. **Reverse the Path**: The sequence of steps found is then followed forward to solve the problem.

Example:

In a **block-stacking problem** (goal: have A on top of B):

- Start from the goal (A on B).
- Identify what action places A on B → Stack(A, B).
- Determine preconditions: A must be **clear** and B must be **clear**.
- Find steps leading to this precondition.
- Work backward until the initial state is reached.

Advantages:

✔ Focuses only on **relevant actions**, reducing the search space.
✔ More efficient in cases where **goal conditions are more constrained** than the initial state.

Disadvantages:

✖ Can be difficult when **many sub-goals exist**.
✖ Sometimes identifying the **right preconditions** is complex.

Comparison of Forward vs Backward Search

Feature	Forward Search (Progression)	Backward Search (Regression)
Direction	Starts from Initial State	Starts from Goal State
Expansion	Expands successors	Expands predecessors
Efficiency	Explores unnecessary states	Focuses on goal-relevant actions
Best Use Case	When the goal is broad	When the goal is well-defined
Search Space	Can be large	Often smaller

Which One to Use?

- Use **Forward Search** when **few actions** are applicable and the problem space is small.
- Use **Backward Search** when **the goal is well-defined** and has **fewer relevant actions**.

Name: Prathamesh Arvind Jadhav

Partial Order Planning in AI

Introduction

Partial Order Planning (POP) is a type of AI planning that allows **flexible sequencing** of actions. Instead of fixing a strict sequence of steps, it constructs a plan where **some actions are ordered only when necessary**. This approach improves efficiency by avoiding unnecessary constraints.

What is Partial Order Planning?

Partial Order Planning is a **nonlinear** planning approach where:

- Actions are ordered **only when required** by dependencies (causality).
- It avoids **premature commitment** to the sequence of actions.
- The plan consists of **partially ordered steps**, meaning some steps are ordered while others remain flexible.

Key Features:

✔ **More flexible** than total-order planning.
✔ **Handles interdependent actions** more efficiently.
✔ **Reduces unnecessary constraints** on action execution.

Steps in Partial Order Planning

1. **Initialize the Plan**
    - Start with an **empty plan** containing only:
        - **Start state** (initial conditions).
        - **Goal state** (desired conditions).
2. **Select a Goal**
    - Choose a **subgoal** that is not yet achieved.
3. **Choose an Action**
    - Find an action that achieves the selected goal.
    - If no existing action satisfies it, **add a new action**.
4. **Handle Dependencies**

   - Ensure the action's **preconditions** are met.
   - Add necessary **ordering constraints** between actions.
   - Add causal links (ensuring preconditions stay valid).
5. **Resolve Conflicts**
   - If an action threatens a causal link (precondition gets undone), resolve it by:
     - **Reordering actions** to prevent interference.
     - **Adding new constraints** to maintain preconditions.
6. **Repeat Until All Goals Are Achieved**
   - Continue adding actions and constraints until all goals are satisfied without conflicts.

Example of Partial Order Planning

Scenario: Making a Cup of Tea

Goal: Have a cup of tea.
Actions Available:

- **Boil Water** (requires water, stove).
- **Add Tea** (requires boiled water, tea leaves).
- **Pour into Cup** (requires tea mixture, cup).

Partial Order Plan:

1. Start with an empty plan.
2. Identify **subgoals**:
   - Water should be boiled before adding tea.
   - Tea must be added before pouring into the cup.
3. Select actions:
   - **Boil Water** → produces boiled water.
   - **Add Tea** → needs boiled water.
   - **Pour into Cup** → needs tea mixture.
4. Order actions **only where necessary**:
   - Boil Water → Add Tea → Pour into Cup
   - No need to specify an exact order for unrelated steps (e.g., taking a cup beforehand).

Final Plan (Partial Order):

```
Boil Water → Add Tea → Pour into Cup
(Other actions unordered if they don't affect each other)
```

Advantages of Partial Order Planning

✔ **More flexibility**: Actions are ordered **only when necessary**, making the plan adaptable.
✔ **More efficient**: Avoids unnecessary constraints, reducing the search space.
✔ **Handles parallelism**: Allows independent actions to occur **simultaneously**.

Comparison: Partial Order vs. Total Order Planning

Feature	Partial Order Planning	Total Order Planning
Ordering	Only necessary actions are ordered	All actions are strictly ordered
Flexibility	High, allows parallel execution	Low, fixed sequence of actions
Efficiency	Avoids unnecessary constraints	Can explore unnecessary sequences
Best Used For	Large, complex domains	Simple, sequential tasks

Planning Graphs in AI

Introduction

A **Planning Graph** is a data structure used in AI planning to efficiently find solutions to planning problems. It is primarily used in **GraphPlan**, an algorithm

that generates **optimal plans** by analyzing the relationships between actions and states.

What is a Planning Graph?

A **Planning Graph** is a **bipartite graph** with alternating layers of **states (facts)** and **actions**. It helps in:

- **Representing possible actions** at each step.
- **Checking feasibility** of achieving goals.
- **Pruning unnecessary actions**, reducing search space.

Structure of a Planning Graph:

A Planning Graph consists of **two alternating levels**:

1. **State Levels (S_0, S_1, S_2, …)** → Represent facts (conditions) that are true at a given step.
2. **Action Levels (A_0, A_1, A_2, …)** → Represent actions that can be executed based on the previous state.

Layers in a Planning Graph:

1. **Initial State (S_0):** The first level contains the initial conditions.
2. **Action Level (A_0, A_1, ...):** Contains all possible actions that can be executed in the previous state.
3. **Resulting State Level (S_1, S_2, ...):** Contains the effects of the actions from the previous level.
4. **Mutex (Mutual Exclusion Links):** Represent conflicts where two actions **cannot** occur together.

Steps to Construct a Planning Graph

Step 1: Initialize with the Initial State

- The first state level (S_0) contains all facts known to be true initially.

Name: Prathamesh Arvind Jadhav

Step 2: Expand Action Level

- Add all possible actions (A_0) whose preconditions are met in S_0.
- Actions include **persistence** (no-op) to carry unchanged facts forward.

Step 3: Expand Resulting State Level

- Apply the actions in A_0 to generate the new set of facts (S_1).
- If an action's effect contradicts another action's effect, they are **mutually exclusive (mutex)**.

Step 4: Repeat Expansion

- Continue expanding ($A_1 \rightarrow S_2 \rightarrow A_2 \rightarrow S_3 \ldots$) until either:
  - The **goal state is reached**.
  - The graph **stops growing**, meaning the problem is unsolvable.

Example of a Planning Graph

Scenario: Making a Sandwich

Initial State (S_0):

- Has Bread
- Has Butter
- Has Jam

Available Actions (A_0):

- Spread Butter (needs Has Bread, Has Butter)
- Spread Jam (needs Has Bread, Has Jam)

New State Level (S_1):

- Bread with Butter
- Bread with Jam

Goal: Bread with Butter and Jam

The **graph expands levels** until the goal appears at some state level.

Name: Prathamesh Arvind Jadhav

How Planning Graphs Help in AI Planning

✔ **Detects Irrelevant Actions**: If an action cannot contribute to the goal, it is ignored.
✔ **Detects Infeasibility**: If the graph stops expanding without reaching the goal, the problem is **unsolvable**.
✔ **Efficient Heuristic Estimation**: Planning graphs provide **relaxed heuristics** (ignoring mutex constraints) for **faster planning**.
✔ **Handles Parallel Actions**: Identifies **independent actions** that can execute simultaneously, speeding up planning.

Comparison: Planning Graph vs. State-Space Search

Feature	Planning Graph	State-Space Search
Efficiency	More structured and efficient	Can be slow due to large branching
Parallelism	Supports parallel actions	Actions are sequential
Use Case	Large, complex planning problems	Simpler problems

Planning with Propositional Logic in AI

Introduction

Planning with **Propositional Logic** in AI involves formulating planning problems as **logical statements** and using **automated reasoning** to find a sequence of actions that achieve a goal. This approach ensures a **systematic, logical** way of solving planning problems.

Name: Prathamesh Arvind Jadhav

How Propositional Logic is Used in Planning

1. Representing States and Actions

- **States** are described using **propositional variables** (true/false statements).
- **Actions** are defined with **preconditions** (must be true before execution) and **effects** (changes caused by the action).

2. Encoding the Planning Problem

A planning problem in propositional logic consists of:

- **Initial State**: A set of propositions describing the starting condition.
- **Goal State**: A set of propositions that must be true at the end.
- **Action Set**: Actions represented as logical rules with:
  - **Preconditions**: What must be true before execution.
  - **Effects**: How the state changes after execution.

3. Finding a Plan

- Convert the problem into a **logical formula**.
- Use a **SAT solver** (Boolean satisfiability solver) to determine if a sequence of actions leads to the goal.

Example: Making Tea

Step 1: Define Propositional Variables

Let's define **true/false** variables for different conditions:

- BoiledWater: Water is boiled.
- HasTea: Tea leaves are available.
- HasCup: A cup is available.
- TeaReady: The tea is ready.

Step 2: Define Actions

Name: Prathamesh Arvind Jadhav

1. Boil Water

- **Precondition**: WaterAvailable
- **Effect**: BoiledWater

2. Add Tea

- **Precondition**: BoiledWater ∧ HasTea
- **Effect**: TeaReady

Step 3: Encode as Logical Formula

1. **Initial State**: WaterAvailable ∧ HasTea ∧ HasCup
2. **Goal**: TeaReady
3. **Action Encoding**:
    - BoiledWater ← WaterAvailable ∧ Boil(Water)
    - TeaReady ← BoiledWater ∧ HasTea ∧ AddTea

Step 4: Solve Using a SAT Solver

- A **SAT solver** checks whether a sequence of actions can satisfy the goal formula.
- If a **valid sequence exists**, the problem is solvable.

Advantages of Propositional Logic in AI Planning

✔ **Guarantees correctness**: Logical reasoning ensures valid plans.
✔ **Optimized by SAT solvers**: Efficient techniques exist for solving logical formulas.
✔ **Can handle complex constraints**: Good for structured environments with well-defined rules.

Name: Prathamesh Arvind Jadhav

Comparison: Propositional Logic vs. State-Space Search

Feature	Propositional Logic Planning	State-Space Search
Solution Method	Uses logic formulas & SAT solvers	Uses search algorithms (BFS, DFS)
Efficiency	Efficient for large, structured problems	Can become slow with large search space
Flexibility	Handles constraints and dependencies well	Needs careful state-space design

Planning and Acting in the Real World in AI

Introduction

In AI, **planning and acting in the real world** refers to how intelligent systems **generate plans** and then **execute them** in dynamic, unpredictable environments. Unlike theoretical AI planning, real-world applications involve **uncertainty, real-time constraints, and continuous interactions** with the environment.

Key Challenges of Real-World Planning and Acting

1. **Uncertainty**: The environment may change unpredictably (e.g., a robot navigating a crowded area).
2. **Partial Observability**: The system may not have complete information about the world (e.g., self-driving cars detecting obstacles).
3. **Dynamic Environment**: Conditions can change while the AI is still executing a plan.
4. **Time Constraints**: Decisions need to be made in real-time (e.g., stock market trading algorithms).
5. **Resource Limitations**: Limited computation, memory, and energy may affect decision-making.

Name: Prathamesh Arvind Jadhav

Components of Real-World Planning and Acting

1. Perception

- The AI **senses** the environment using cameras, sensors, or user input.
- Example: A **robotic vacuum** detects obstacles before moving.

2. Planning

- The AI generates an **optimal sequence of actions** to achieve a goal.
- Planning techniques include:
  - **Classical Planning**: Assumes a predictable environment.
  - **Probabilistic Planning**: Deals with uncertainty.
  - **Hierarchical Task Planning (HTN)**: Breaks complex tasks into subtasks.

3. Execution

- The AI executes actions while continuously monitoring the environment.
- Example: A **self-driving car** follows a planned route but adjusts for pedestrians.

4. Replanning & Adaptation

- If the environment changes, the AI **modifies its plan** on the fly.
- Example: If a **drone** encounters strong wind, it adjusts its flight path.

5. Learning & Feedback

- AI systems **learn from past experiences** to improve future planning.
- Example: A **chess AI** refines its strategy based on previous matches.

Example: Self-Driving Cars

A self-driving car must:

1. **Perceive** the road, traffic signals, and obstacles.

2. **Plan** a safe route to its destination.
3. **Act** by accelerating, braking, and steering.
4. **Replan** if a new obstacle appears.
5. **Learn** from traffic patterns to improve navigation.

Techniques Used in Real-World Planning & Acting

Technique	Description	Example
Reactive Planning	Responds instantly to environment changes	Autonomous robots avoiding obstacles
Hierarchical Task Planning (HTN)	Breaks tasks into subtasks for better execution	AI assistant scheduling tasks
Probabilistic Planning (MDPs, POMDPs)	Handles uncertainty with probabilities	Medical diagnosis AI
Real-Time Planning	Makes decisions under time constraints	Stock trading AI