4.2 Classical Planning

Classical Planning is a core concept in Artificial Intelligence where an agent determines a sequence of actions to transform an initial state into a goal state in a fully observable, deterministic, and static world. This process is similar to finding a path in a graph, and effective planners use heuristic search to guide this process efficiently, often using a formal language like PDDL to define the problem's components.

Core Components of a Classical Planning Problem

Initial State:

The starting condition of the environment.

Goal State:

The desired final condition of the environment.

Actions:

The available operations that an agent can perform, each with defined preconditions (what must be true to execute the action) and effects (how the state changes after execution). Key Assumptions of Classical Planning Deterministic Actions:

The outcome of an action is always predictable and known.

Fully Observable Environment:

The agent has complete knowledge of the current state of the world at all times.

Static Environment:

The environment does not change on its own; only the actions of the agent can alter it.

Discrete States and Actions:

The world and available actions are typically broken down into discrete, distinct elements.

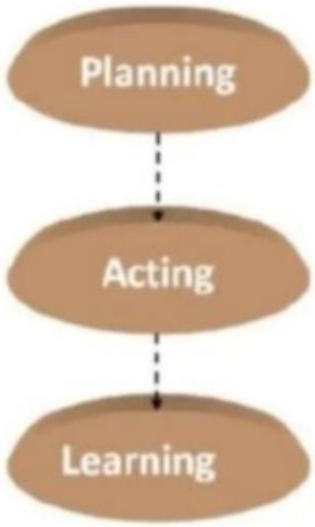
How It Works

Problem Representation: The initial state, actions, and goals are described, often using a language like PDDL (Planning Domain Definition Language), which specifies the variables, predicates, and operator schemas.

State Space Search: The planning problem is viewed as a state space search, where each node in a conceptual graph represents a possible state of the world.

Heuristic Guidance: To make the search feasible, classical planners use heuristic functions. These functions provide an estimate of the cost to reach the goal from a given state, directing the search toward more promising paths and away from irrelevant ones.

Plan Generation: The planner explores the state space, applying actions, until it finds a sequence of actions that leads from the initial state to a state satisfying the goal conditions.



Applications

Classical planning finds use in various AI domains, including:

Robotics:

Planning sequences of movements and actions for robotic systems.

Automated Problem-Solving:

Finding solutions to various types of problems that can be framed in terms of state transitions.

Game AI:

Enabling non-player characters (NPCs) to make intelligent decisions and act towards a goal.

STRIPS

STRIPS stands for Stanford Research Institute Problem Solver. It's a planning system developed in 1971 for AI, which represents actions, goals, and states in a structured way to help an AI plan a sequence of actions to reach a goal.

It's mainly used in classical planning problems.

Components of STRIPS

STRIPS represents a planning problem with three main components:

1.Initial State (S_0):

The starting condition of the world.

Represented as a set of logical statements (facts) that are true.

2.Goal State (G):

The condition(s) that need to be achieved.

Also represented as logical statements.

3.Actions (Operators): Each action has:

Preconditions: What must be true to perform the action.

Add List: Facts that become true after the action.

Delete List: Facts that are no longer true after the action.

STRIPS Planning Process

1.Start with the initial state.

2.Check which actions' preconditions are satisfied.

3.Apply an action:

Add facts from Add List to the state.

Remove facts from Delete List.

Repeat until the goal state is reached.

4.Example (Simple Blocks World)

Initial State:

On(A, Table), On(B, Table), Clear(A), Clear(B)

Goal State:

On(A, B)

Action: Stack(X, Y)

Precondition: Clear(X) Clear(Y) On(X, Table)

Add List: On(X, Y)

Delete List: Clear(Y), On(X, Table), Clear(X)

Here, STRIPS allows the AI to figure out “move A onto B” as a plan.

5.STRIPS in Short Formal method for AI planning.

Uses states, goals, and actions.

Relies on logical representation.

Forms the foundation for modern planners.

Planning Graphs

Planning graphs play a vital role in AI planning by visually representing possible states and actions that aid in decision-making. This article explores STRIP-like domains that construct and analyze the compact structure called graph planning. We will also delve into the role of mutual exclusion, providing a suitable example using a graph planning algorithm.

What is a Planning Graph?

A Planning Graph is a data structure primarily used in automated planning and artificial intelligence to find solutions to planning problems. It represents a planning problem’s progression through a series of levels that describe states of the world and the actions that can be taken. Here’s a breakdown of its main components and how it functions:

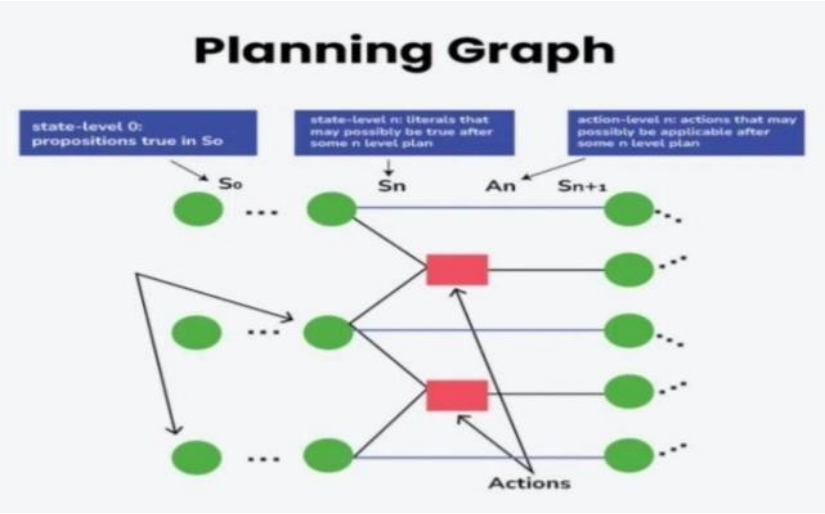
Levels: A Planning graph has two alternating types of levels: action levels and state levels. The first level is always a state level, representing the initial state of the planning problem.

State Levels: These levels consist of nodes representing logical propositions or facts about the world. Each successive state level contains all the propositions of the previous level plus any that can be derived by the actions of the intervening action levels.

Action Levels: These levels contain nodes representing actions. An action node connects to a state level if the state contains all the preconditions necessary for that action. Actions in turn can create new state conditions, influencing the subsequent state level.

Edges: The graph has two types of edges: one connecting state nodes to action nodes (indicating that the state meets the preconditions for the action), and another connecting action nodes to state nodes (indicating the effects of the action).

Mutual Exclusion (Mutex) Relationships: At each level, certain pairs of actions or states might be mutually exclusive, meaning they cannot coexist or occur together due to conflicting conditions or effects. These mutex relationships are critical for reducing the complexity of the planning problem by limiting the combinations of actions and states that need to be consider



Levels in Planning Graphs

Level S0: It is the initial state of the planning graph that consists of nodes each representing the state or conditions that can be true.

Level A0: Level A0 consists of nodes that are responsible for taking all specific actions in terms of the initial condition described in the S0.

Si: It represents the state or condition which could hold at a time I, it may be both P and ¬P.

Ai: It contains the actions that could have their preconditions satisfied at i.

Working of Planning Graph

The planning graph has a single proposition level that contains all the initial conditions. The planning graph runs in stages, each stage and its key workings are described below:

Extending the Planning Graph: At stage I (the current level), the graph plan takes the planning graph from stage i-1 (the previous stage) and extends it by one time step. This adds the next action level representing all possible actions given the propositions (states) in the previous level, followed by the proposition level representing the resulting states after actions have been performed.

Valid Plan Found: If the graph plan finds a valid plan, it halts the planning process.

Proceeding to the Next Stage: If no valid plan is found, the algorithm determines that the goals are not all achievable in time I and moves to the next stage.

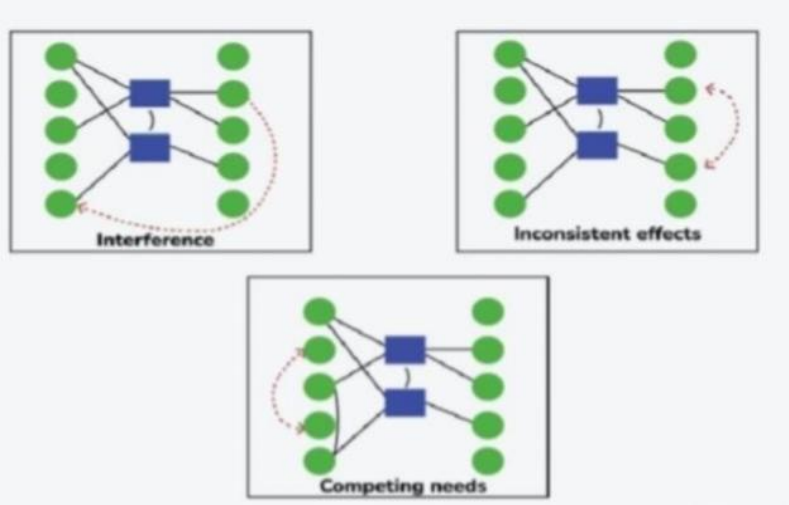
Mutual Exclusion in Planning Graph

Mutual exclusion in graph planning refers to the principle that certain actions or propositions cannot coexist or occur simultaneously due to inherent constraints or dependencies within the planning problem. Mutex relations can hold between actions and literals under various conditions.

Mutex Conditions Between Actions

Inconsistent Effects: One action negates the effect of another

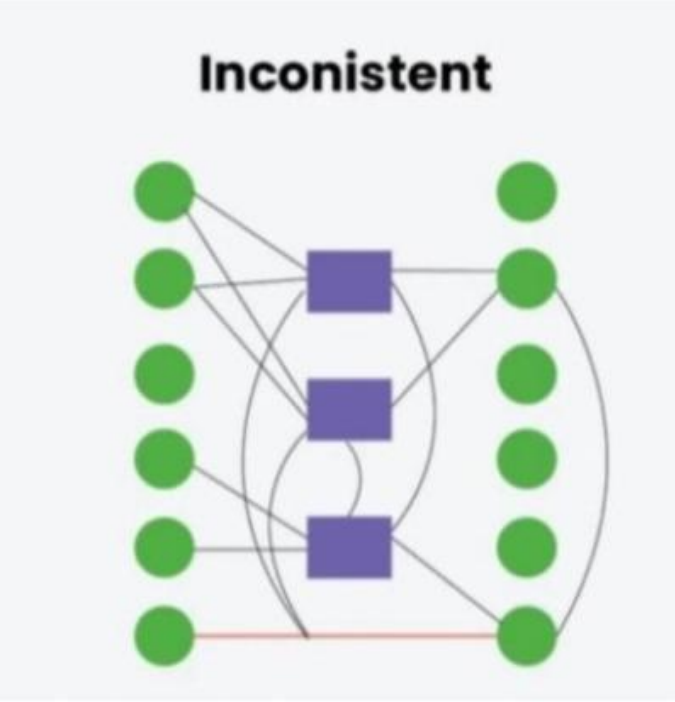
Interference: One action deletes a precondition or creates an add-effect of another.Competing Needs: Precondition of action a and precondition of action b cannot be true simultaneously



Mutex Conditions Between Literals

Negation of Each Other: Two literals are mutually exclusive if one is the negation of the other.

Achieved by Mutually Exclusive Actions: No pair of non-mutex actions can make both literals true at the same level.



Planning a Graph for a CAKE Problem

To understand the concept of planning a graph, let's use the CAKE example. This illustration demonstrates various states and actions to achieve the goal of successfully eating the cake. By examining these states, we can see how actions transition from one state to another to reach the desired outcome.

Initial State (S0)

Have(Cake): We have the cake.

¬Eaten(Cake): The cake has not been eaten yet.

Action Level (A0)

The action level represents possible actions that can transition the state from S0 to the next state. In our example, the available action is:

Eat(Cake): Represents the action to eat the cake.

Next State (S1)

The next state is determined by the action taken at the action level based on the propositional variables. In the CAKE example, we have four propositional variables in the next state:

Have(Cake): If no action is performed (no-op), we will still have the cake.

¬Have(Cake): If the action Eat(Cake) is performed, we will no longer have the cake.

Eaten(Cake): If the action Eat(Cake) is performed, the cake will have been eaten.

¬Eaten(Cake): If no action is performed (no-op), the cake will still be uneaten.

Performance of Mutual Exclusion in Planning Graph

At state level S1, all literals are obtained by considering any subset of actions at A0. In simple terms, state level S1 holds all possible outcomes after the actions in A0 are considered. In our example, since we only have the Eat(Cake) action at A0, S1 will list all possible outcomes with and without the action being taken.

Mutual Exclusion

Mutual exclusion occurs when a conflict arises between literals, indicating that the two literals cannot occur together. These conflicts are represented by mutex links, which reveal mutually exclusive propositions in S1.

For instance, if we eat the cake, we cannot have the cake at the same time. Thus, Have(Cake) and Eaten(Cake) would be mutually exclusive. The mutex links define the set of states, revealing which combinations of literals are not possible together.

For example:

If Eat(Cake) is performed, Have(Cake) and ¬Eaten(Cake) cannot be true simultaneously.

If Eat(Cake) is not performed, ¬Have(Cake) and Eaten(Cake) cannot be true together.

Second Action Level (A1) and State Level (S2)

Continuing from the above example, we introduce two more levels: Actionlevel (A1) and State-level (S2).

Actions Available at A1 from S1

Bake(Cake): The precondition for this action is ¬Have(Cake). If ¬Have(Cake) is true in S1, applying Bake(Cake) results in Have(Cake) in S2. This action does not affect the Eaten(Cake) or ¬Eaten(Cake) states.

Eat(Cake): The precondition for this action is Have(Cake). If Have(Cake) is true in S1, applying Eat(Cake) results in ¬Have(Cake) and Eaten(Cake) in S2.

Steps in the Graph Plan Algorithm

Expand the Planning Graph: The graph is expanded level by level until it reaches level n where all the goals are present and non-mutex.

Check for Plan Existence: If the planning graph levels off before all goals are present and non-mutex, the algorithm fails.

Search for Valid Plan: The algorithm performs a back search from the last level to the initial state to find a sequence of actions leading to the goals without violating mutex constraints.

Expand if No Valid Plan Found: If no valid plan is found, another level is added and the search is repeated until a plan is found or the graph levels off.

Properties of Graph Plan

The elements in the planning graph are described as increasing or decreasing monotonically:

Literals Increase Monotonically

Actions Increase Monotonically

Mutexes Decrease Monotonically

Due to these properties, the presence of a finite number of actions and literals enables the planning graph to eventually level off.

Conclusion

The graph plan algorithm leverages planning graphs to systematically explore and resolve planning problems. By incorporating mutual exclusion principles, it ensures consistency and feasibility. Through the CAKE example, we have demonstrated how planning graphs efficiently handle complex scenarios.