Planning in AI

Contents

What is Planning in AI?

- Planning in AI is the process of devising a plan of action to achieve a specific goal or set of goals.
  - *Involves reasoning about future actions and their consequences*
  - *Aims to find an optimal sequence of actions*
  - *Critical for autonomous systems and decision-making processes*
- Example: A robot navigating a warehouse to pick up and deliver packages.

Classical Planning

- Classical planning refers to a simplified form of planning with the following assumptions:
  - *Fully observable environment*
  - *Deterministic actions*
  - *Static world (only changes through agent's actions)*
  - *Finite set of actions and states*
  - *Single agent acting in the world*
- Example: Blocks World problem - arranging blocks on a table to achieve a specific configuration.

Components of a Classical Planning Problem

- Initial State: The starting condition of the world
- Goal State: The desired outcome or condition
- Actions: Set of possible actions the agent can take
- Transition Model: Describes how actions change the state
- Action Costs: The cost associated with each action
- Example (Blocks World):
  - *Initial State: Block A on table, B on A, C on table*
  - *Goal State: C on B, B on A, A on table*
  - *Actions: Move(x, y, z) - Move block x from y to z*

Representation Languages for Planning

- Common languages used to represent planning problems:
  - *STRIPS (Stanford Research Institute Problem Solver)*
  - *ADL (Action Description Language)*
  - *PDDL (Planning Domain Definition Language)*
- These languages help formalize the planning problem and its components.
- Example: PDDL is used in the International Planning Competition to define standardized planning problems.

STRIPS Representation: Example

- STRIPS uses:
  - *Preconditions: Conditions that must be true before an action*
  - *Effects: Changes that occur after an action is performed*
  - *Add list: Facts that become true after the action*
  - *Delete list: Facts that become false after the action*
- Example: Move(A, B, C) - Move block A from B to C
  - *Preconditions: On(A, B), Clear(A), Clear(C)*
  - *Add list: On(A, C), Clear(B)*
  - *Delete list: On(A, B), Clear(C)*

Planning Domain Definition Language (PDDL): Example

- PDDL separates domain definition from problem definition:
- Example (Blocks World Domain):

```
(define (domain blocks)
  (:predicates (on ?x ?y) (ontable ?x) (clear ?x) (handempty) (holding ?x))
  (:action pick-up
    :parameters (?x)
    :precondition (and (clear ?x) (ontable ?x) (handempty))
    :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty))
                 (holding ?x)))
  (:action put-down
    :parameters (?x)
    :precondition (holding ?x)
    :effect (and (not (holding ?x)) (clear ?x) (handempty) (ontable ?x))))
```

Algorithms for Classical Planning

- Goal: Find a sequence of actions to transform the initial state into the goal state
- Key components of a planning problem:
  - *Initial state*
  - *Goal state*
  - *Set of actions (with preconditions and effects)*
- Main categories of planning algorithms:
  - *State-space search*
  - *Plan-space search*
  - *Graph-based planning*

State-Space Search

- Searches through possible world states
- Two main approaches:
  - *Forward search (progression)*
  - *Backward search (regression)*
- Can use uninformed (e.g., BFS, DFS) or informed (e.g., A*) search strategies
- State representation: Complete description of the world at a given point
- Example: Blocks World
  - *State: Position of each block (on table or on another block)*
  - *Actions: Move a block from one position to another*

Forward State-Space Search

- Algorithm steps:
  - *Start with the initial state*
  - *Generate successor states by applying all applicable actions*
  - *Check if the current state matches the goal state*
  - *If not, repeat steps 2-3 for each new state*
  - *Stop when the goal state is reached or no more states to explore*
- Example (Blocks World):
  - *Initial State: A on table, B on table, C on table*
  - *Goal State: C on B, B on A, A on table*
- Search progression:
  - *Initial: (At, Bt, Ct)*
  - *Move B to A: (At, BA, Ct)*
  - *Move C to B: (At, BA, CB)*
  - *Goal reached!*

Backward State-Space Search

- Algorithm steps:
  - *Start with the goal state*
  - *Generate predecessor states by finding actions that could lead to the current state*
  - *Check if the current state matches the initial state*
  - *If not, repeat steps 2-3 for each new state*
  - *Stop when the initial state is reached or no more states to explore*
- Example (Blocks World):
  - *Goal State: C on B, B on A, A on table*
  - *Initial State: A on table, B on table, C on table*
- Search regression:
  - *Goal: (At, BA, CB)*
  - *Possible predecessor: (At, BA, C?) - C could be anywhere*
  - *Possible predecessor: (At, B?, C?) - B and C could be anywhere*
  - *Initial state reached: (At, Bt, Ct)*

Plan-Space Search

- Searches through the space of partial plans
- A partial plan is a set of actions with ordering constraints
- Starts with an empty plan and refines it by adding actions and constraints
- Focuses on resolving flaws in the plan
- Main algorithm: Partial-Order Planning (POP)
- Key concepts:
  - *Causal link: Shows how one action fulfills the precondition of another*
  - *Threat: An action that could interfere with a causal link*

Partial-Order Planning (POP) Algorithm

- Steps:
  - *Start with initial state action (Start) and goal state action (Finish)*
  - *Choose an unsatisfied precondition (flaw) in the plan*
  - *Resolve the flaw by: a) Adding an existing action that achieves the precondition, or b) Adding a new action that achieves the precondition*
  - *Add ordering constraints and causal links*
  - *Resolve any threats to causal links*
  - *Repeat steps 2-5 until no flaws remain*
- Example (Simplified Blocks World):
  - *Initial: On(A, table), On(B, table)*
  - *Goal: On(B, A)*
- Partial plan development:
  - *Start → Finish*
  - *Add Move(B, table, A) to achieve On(B, A)*
  - *Add causal link: Move(B, table, A) → On(B, A) → Finish*
  - *Resolve threat: Ensure Clear(A) before Move(B, table, A)*

Graph-Based Planning

- Represents the planning problem as a layered graph
- Allows for efficient analysis of action dependencies and parallel actions
- Main algorithm: GraphPlan
- Graph structure:
  - *Alternating levels of proposition nodes and action nodes*
  - *Mutex (mutual exclusion) relations between nodes*
- Benefits:
  - *Can quickly determine if a plan exists*
  - *Efficiently handles parallel actions*
  - *Provides a compact representation of the planning problem*

GraphPlan Algorithm

- Steps:
  - *Construct the planning graph: a) Start with initial state propositions b) Add applicable actions c) Add resulting propositions d) Repeat b-c until goal propositions appear or graph levels off*
  - *Extract a solution by searching backward from the goals*
- Example (Simplified Blocks World):
  - *Initial: On(A, table), On(B, table)*
  - *Goal: On(B, A)*
- Graph construction:
  - *P0: On(A, table), On(B, table), Clear(A), Clear(B)*
  - *A0: Move(B, table, A), NoOp(A), NoOp(B)*
  - *P1: On(A, table), On(B, A), Clear(B)*
- Solution extraction:
  - *Goal On(B, A) appears in P1*
  - *Achieved by Move(B, table, A) in A0*
  - *Preconditions satisfied in P0*

Heuristics in AI Planning

- What is a Heuristic?
  - *A heuristic is a function that estimates the cost of reaching the goal from a given state.*
- Why Heuristics?
  - *They reduce search space, leading to faster solutions.*
  - *Help to guide search algorithms like A*, ensuring optimality in many cases.*
- Example: In navigation, using straight-line distance (Euclidean distance) as a heuristic.

Evaluating Heuristics

- Admissible Heuristics: A heuristic is admissible if it never overestimates the cost of reaching the goal.
  - *Example: Manhattan distance in a grid world is admissible since it never overestimates the path cost.*
- Consistent Heuristics: A heuristic is consistent if for every node, the estimated cost is always less than or equal to the cost of reaching any neighboring node plus the cost from that neighbor to the goal.
  - *Example: If moving between two adjacent points costs 1 unit, a consistent heuristic never suggests a lower cost for that move.*

Common Heuristics in Planning

- Zero Heuristic (h = 0):
  - *Use: Used in uninformed search (like breadth-first search). No guidance, always expands nodes equally.*
  - *Result: Slow, as it explores all possible paths equally.*
- Goal Distance Heuristic (h = distance to goal):
  - *Use: Often used in A* search.*
  - *Example: In a city map, using straight-line distance to the goal.*
- Domain-Specific Heuristics:
  - Example: Heuristic specifically used for the water-jug problem.

Planning and Acting in Non-deterministic Domains

- Unlike classical planning, real-world scenarios often involve uncertainty
- Non-deterministic domains: Actions may have multiple possible outcomes
- Challenges:
  - *Uncertain action effects*
  - *Partial observability*
  - *Dynamic environments*

Characteristics of Non-deterministic Planning

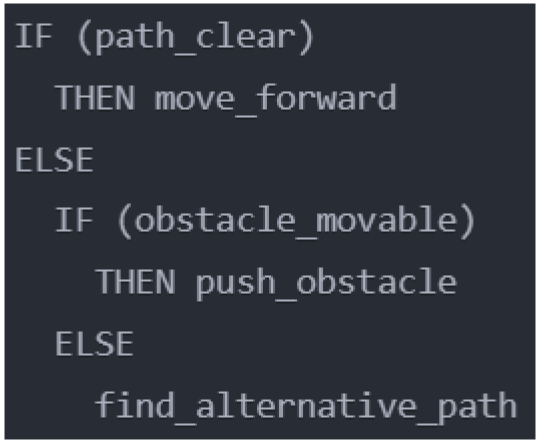
- Goal: Create robust plans that work despite uncertainty
- Key differences from classical planning:
  - *Actions can have multiple possible outcomes*
  - *The exact state may not be fully known*
  - *The environment may change independently of the agent's actions*

Approaches to Non-deterministic Planning

- Contingency Planning
- Replanning
- Reactive Planning
- Probabilistic Planning

Contingency Planning

- Creates a branching plan that accounts for different possible outcomes
- Example: Robot Navigation



```
IF (path_clear)
  THEN move_forward
ELSE
  IF (obstacle_movable)
    THEN push_obstacle
  ELSE
    find_alternative_path
```

Replanning

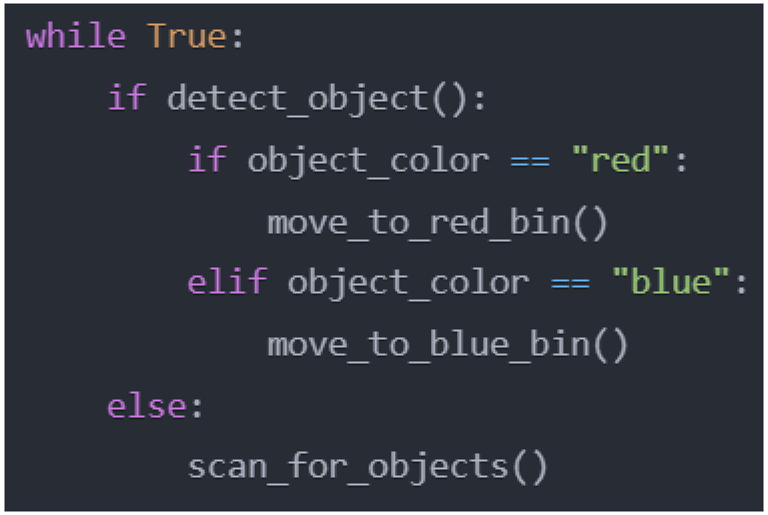
- Create an initial plan based on current knowledge
- If the plan fails during execution, replan from the current state
- Advantages: Simple, adaptable to changing environments
- Disadvantages: May be computationally expensive for frequent replanning
- Example: Package Delivery Robot
  - *Initial plan: Route $A \rightarrow B \rightarrow C$*
  - *Road block encountered between A and B*
  - *Replan: New route $A \rightarrow D \rightarrow B \rightarrow C$*

Reactive Planning

- Instead of creating a full plan in advance, make decisions based on current percepts
- Often implemented as a set of condition-action rules
- Advantages: Fast response, low memory requirements
- Disadvantages: May not find optimal solutions for complex problems

Contd…

- Example: Reactive Robotic Arm



```
while True:
    if detect_object():
        if object_color == "red":
            move_to_red_bin()
        elif object_color == "blue":
            move_to_blue_bin()
    else:
        scan_for_objects()
```

Probabilistic Planning

- Assigns probabilities to action outcomes and states
- Often uses Markov Decision Processes (MDPs) or Partially Observable MDPs (POMDPs)
- Goal: Maximize expected utility or minimize expected cost
- Example: MDP for Robot Navigation
  - *States: Grid locations*
  - *Actions: Move North, South, East, West*
  - *Transition Model: 80% chance of moving in intended direction, 10% chance of moving left or right*
  - *Reward: -1 for each move, +10 for reaching goal, -10 for hitting obstacle*

Comparison of Non-deterministic Planning Approaches

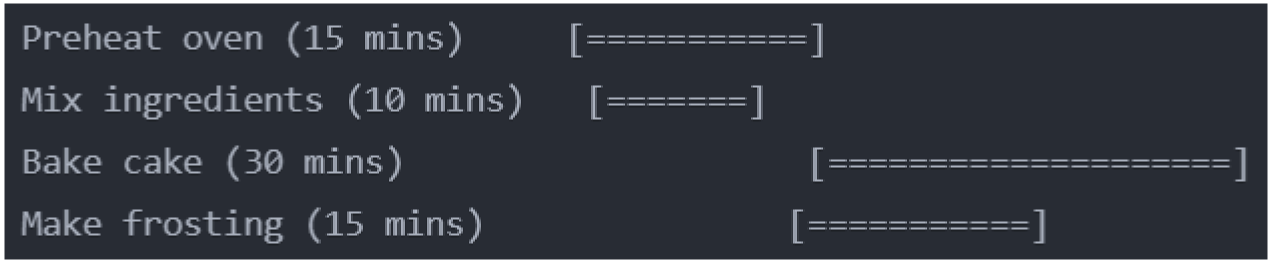
Approach	Pros	Cons
Contingency Planning	Handles all possible outcomes	Can be complex for large problems
Replanning	Adaptable to changes	May require frequent replanning
Reactive Planning	Fast response time	May not find optimal solutions
Probabilistic Planning	Optimal for stochastic environments	Can be computationally intensive

Time, Schedules, and Resources in AI Planning (Temporal Planning)

- Classical planning assumes instantaneous actions
- Real-world actions:
  - *Take time to complete*
  - *Can overlap*
  - *May require specific resources*
  - *Have temporal constraints*
- Example: Making a meal (multiple dishes cooking simultaneously)

Time in Planning

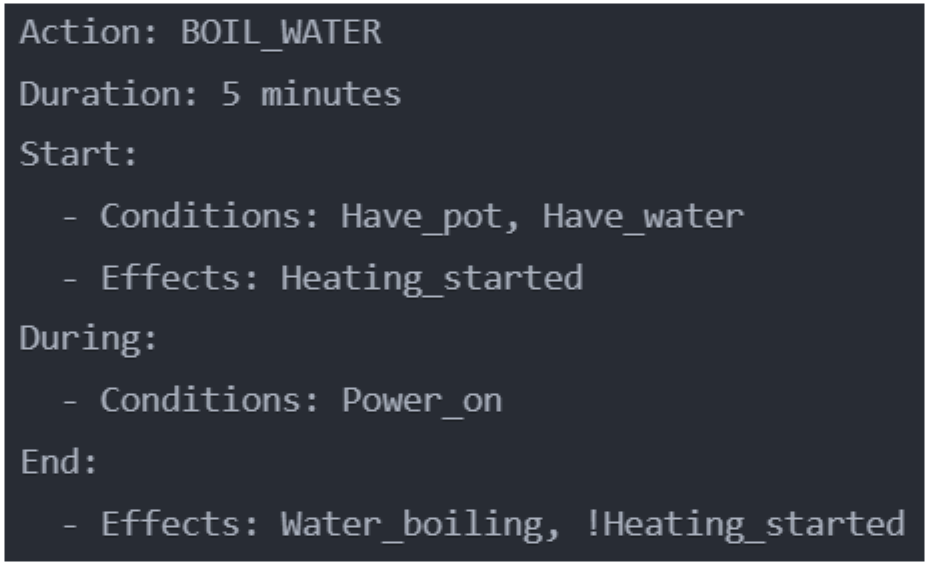
- Types of Temporal Relations:
  - *Before: Action A must complete before B starts*
  - *After: Action A must start after B completes*
  - *During: Action A must occur during B*
  - *Overlaps: Part of Action A overlaps with B*
- Example: Baking a Cake



```
Preheat oven (15 mins)     [===========]
Mix ingredients (10 mins)   [=======]
Bake cake (30 mins)                    [====================]
Make frosting (15 mins)               [===========]
```

Durative Actions

- Actions with explicit duration
- Three parts:
  - Start conditions and effects
  - Invariant conditions (must hold during execution)
  - End conditions and effects
- Example: Boiling Water



```
Action: BOIL_WATER
Duration: 5 minutes
Start:
  - Conditions: Have_pot, Have_water
  - Effects: Heating_started
During:
  - Conditions: Power_on
End:
  - Effects: Water_boiling, !Heating_started
```

Schedule Management

- Key Components:
  - *Tasks: Activities to be scheduled*
  - *Resources: Required for task execution*
  - *Constraints: Temporal and resource limitations*
  - *Optimization Goals: Minimize time, cost, etc.*
- Common Scheduling Problems:
  - *Job Shop Scheduling*
  - *Project Scheduling*
  - *Employee Shift Planning*
  - *Transportation Scheduling*

Job Shop Scheduling Example

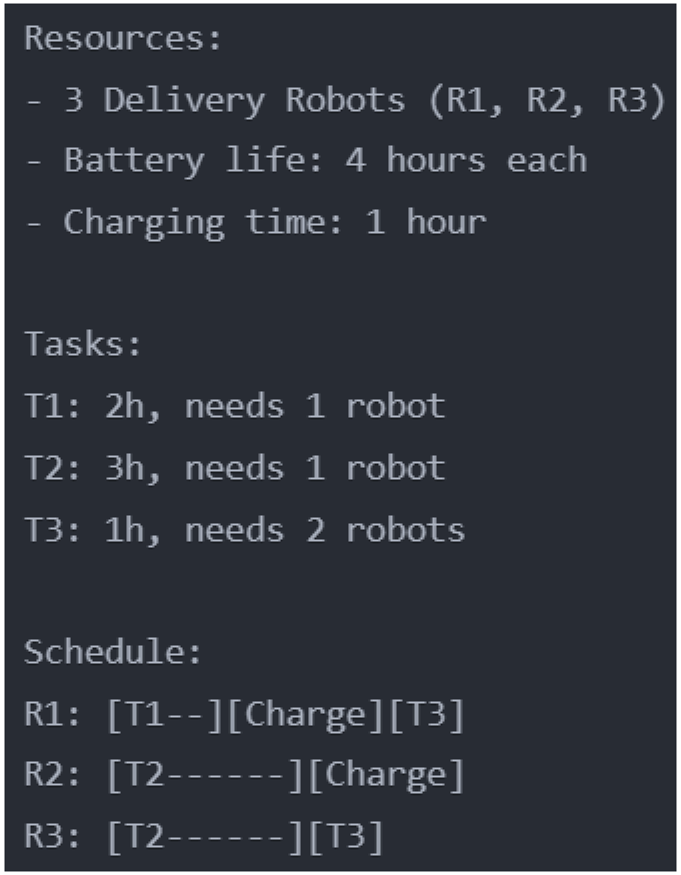
- https://www.youtube.com/watch?v=eiy-OTcNSR0&t=50s

Resource Management

- Types of Resources:
  - *Consumable: Used up when consumed (e.g., fuel)*
  - *Reusable: Available after use (e.g., machines)*
  - *Renewable: Replenished over time (e.g., battery charge)*
- Resource Constraints:
  - *Capacity limitations*
  - *Availability windows*
  - *Setup/cleanup requirements*
  - *Maintenance schedules*

Resource Allocation Example

- Robot Delivery System:



```
Resources:
- 3 Delivery Robots (R1, R2, R3)
- Battery life: 4 hours each
- Charging time: 1 hour

Tasks:
T1: 2h, needs 1 robot
T2: 3h, needs 1 robot
T3: 1h, needs 2 robots

Schedule:
R1: [T1--][Charge][T3]
R2: [T2------][Charge]
R3: [T2------][T3]
```

Best Practices for Temporal Planning

- Resource Utilization
  - *Balance resource allocation*
  - *Account for setup/cleanup time*
  - *Plan for maintenance*
- Schedule Optimization
  - *Minimize makespan*
  - *Maximize resource efficiency*
  - *Handle uncertainty*
- Constraint Management
  - *Identify critical paths*
  - *Maintain flexibility*
  - *Plan for contingencies*