Planning in Artificial Intelligence

In Artificial Intelligence, there are agents which perceive the environment via sensors and act upon the environment through actuators or effectors. Just like humans have sensors through which we sense our surroundings (eyes, ears, nose, tongue, and skin) and actuators (limbs) to perform actions on these surroundings. The agent starts from the initial state and performs a series of actions in order to reach the goal state. For instance, a vacuum cleaner agent will perform actions of moving right and left, and sucking in dirt to reach the goal of successfully cleaning the environment.

Planning and its types

This activity of coming up with a sequence of actions in order to accomplish the target or goal is called as Planning. Planning can be Classical or Non-classical. In case of Classical Planning, the environment is fully observable, deterministic, static and discrete, whereas in case of Non-classical Planning, the environment is partially observable (i.e. the entire state of the environment is not visible at a given instant) or non- deterministic (or stochastic, i.e. the current state and chosen action cannot completely determine the next state of the environment).

Planning languages

Representation of Planning problems is often done using STRIPS (Stanford Research Institute Problem Solver). It consists of:

A set of states- It is a conjunction of positive ground literals.

A set of goals- partially specified state represented as a conjunction of positive literals, and

A set of actions- For each action, there is a precondition that must be satisfied and an effect which reflects the impact the action has on the environment after it has been performed.

Planning problems can also be represented using PDDL (Planning Domain Definition Language).

Planning using State Space Search

State space consists of the initial state, set of goal states, set of actions or operations, set of states and the path cost. This state space needs to be searched to find a sequence of actions leading to the goal state. This can be done in the forward or backward direction.

Forward State Space Search

It is also called Progression. It starts from the initial state and searches in the forward direction till we reach the goal. It uses STRIPS representation. This is how the problem formulation looks like.

Initial state: start state

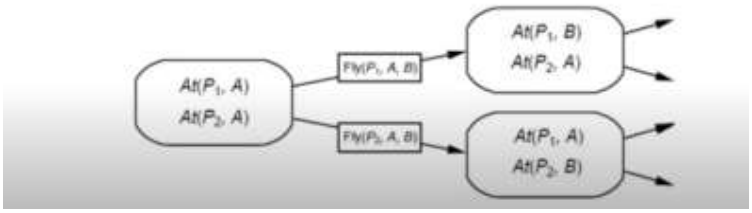
Actions: Each action has a particular precondition to be satisfied before the action can be performed and an effect that the action will have on the environment.

Goal test: To check if the current state is the goal state or not.

Step cost: Cost of each step which is assumed to be 1.

FSSS starts from the initial state and applies actions to reach the next state. It then checks whether this state is the goal state or not. If not, it continues to apply other actions till the goal is reached.

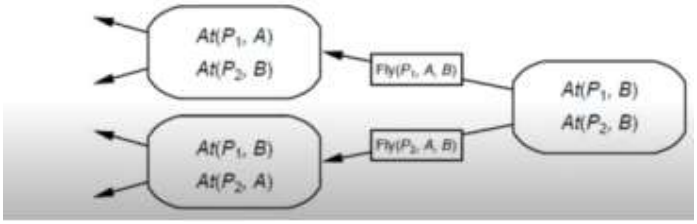
$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$



Let's take an example of the Air cargo problem. Here P1 and P2 are the two planes and A and B are the two airports. We start from the initial state and use the problem's actions to search forward for the goal state.

Backward State Space Search

It is also called as Regression. It uses STRIPS representation. The problem formulation is similar to that of FSSS and consists of the initial state, actions, goal test and step cost. In BSSS, the searching starts from the goal state, and moves in the backward direction until the initial state is reached. It starts at the goal, checks if it is the initial state. If not, it applies the inverse of the actions to produce sub goals until start state is reached. For instance,



Total Order planning (TOP)

FSSS and BSSS are examples of TOP. They only explore linear sequences of actions from start to goal state, They cannot take advantage of problem decomposition, i.e. splitting the problem into smaller sub-problems and solving them individually.

Partial Order Planning (POP)

It works on problem decomposition. It will divide the problem into parts and achieve these sub goals independently. It solves the sub problems with sub plans and then combines these sub plans and reorders them based on requirements. In POP, ordering of the actions is partial. It does not specify which action will come first out of the two actions which are placed in the plan. Let's look at this with the help of an example. The problem of wearing shoes can be performed through total order or partial order planning.

Init: Barefoot

Goal: RightShoeOn ^ LeftShoeOn

Action: 1. RightShoeOn

Precondition: RightSockOn

Effect: RightShoeOn

2. LeftShoeOn

Precondition: LeftSockOn

Effect: LeftShoeOn

3. LeftSockOn

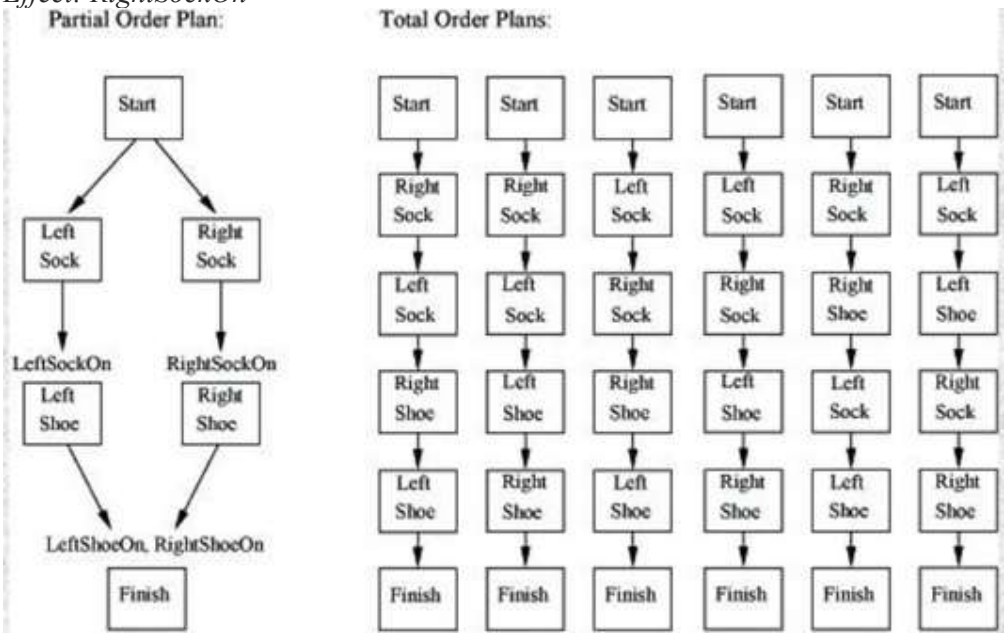
Precondition: Barefoot

Effect: LeftSockOn

4. RightSockOn

Precondition: Barefoot

Effect: RightSockOn



The TOP consists of six sequences, one of which can be taken in order to reach the finish state. However, the POP is less complex. It combines two action sequences. The first branch covers the left sock and left shoe. To wear left shoe, wearing the left sock is a precondition. Similarly the second branch covers the right sock and right show. Once these actions are taken, we achieve our goal and reach the finish state.

How Machine Learning and Artificial Intelligence Changing the Face of eCommerce? | Data Driven…

The eCommerce development company, nowadays, integrating advancement to take customer experience to the next level…

www.datadriveninvestor.com

Defining a Partial Order Plan

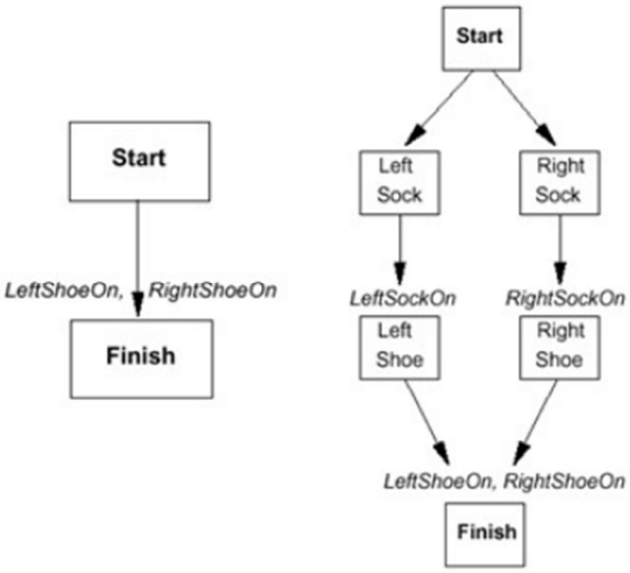
- A set of actions, that make up the steps of the plan. For instance, {RightShoe, RightSock, LeftShoe, LeftSock, Start, Finish}.
- A set of ordering constraints, A before B. For instance, {RightSock < RightShoe, LeftSock < LeftShoe}.

$A \prec B$

- A set of causal links, A achieves P for B.

$A \xrightarrow{p} B$

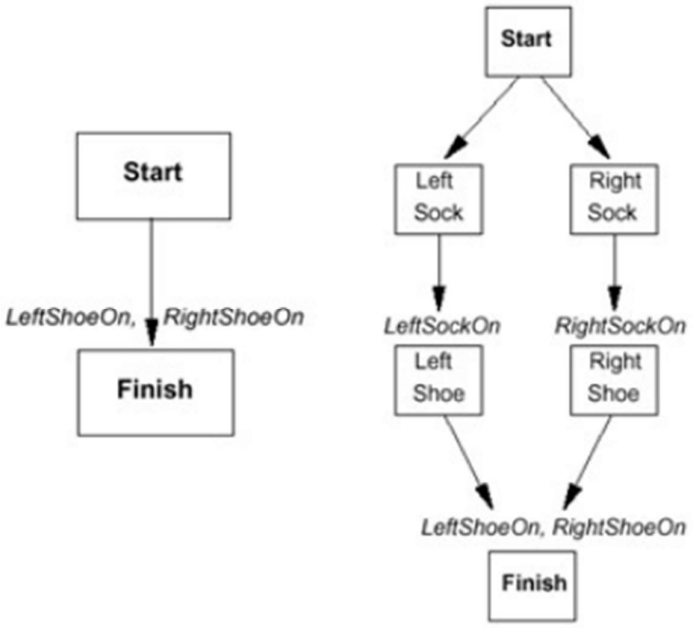
- A set of open preconditions. A precondition is open if it is not achieved by some action in the plan.



Partial Order Planning in AI

- Any algorithm that can place two actions into a plan without specifying which comes first is called as Partial Order Planning.
- Partial Order Planning works on several sub-goals independently & solves them with several sub-plans & thereafter combines them.
- With Partial Order Planning, problems can be decomposed so it can work well with non-cooperative environments. Thus, it is a plan which specifies all actions that need to be taken, but only specifies the order between actions *when necessary*.
- It uses a least commitment strategy.
- It consists of two states: “start” & “finish” where,

1. **Start:** No preconditions & Effects are initial states.
2. **Finish:** No effects & Preconditions are goals.



Partial Order Planning for Wearing a Shoe

Here, the partial order planner for wearing a shoe combines two actions sequences:

1. LeftSock & LeftShoe: Wearing the LeftSock is a precondition for wearing the LeftShoe.
2. RightSock & RightShoe: Wearing the RightSock is a precondition for wearing the RightShoe.

Partial Order Planning as a search problem:

A partial-order plan consists of four components:

1. Set of actions:

- They form the steps of the plans.
- Actions which can be performed, to achieve a goal are stored in set of actions component.
- Eg: Set of actions = { Start, RightSock, RightShoe, LeftSock, LeftShoe, Finish }

2. Set of ordering preconditions:

- They are ordering constraints i.e. without performing action “x” we cannot perform “y”.
- Eg: Set of ordering = { RightSock < RightShoe; LeftSock < LeftShoe } i.e to wear a shoe, first a sock has to be worn.

3. Set of casual links:

- It specifies which actions meet which preconditions of other actions.
- It provides a link from outcome of one step to precondition of another.

- Eg: Set of casual links = { RightSock -> RightSockOn -> RightShoe, LeftSock -> LeftSockOn -> LeftShoe, RightShoe -> RightShoeOn -> Finish, LeftShoe -> LeftShoeOn -> Finish }

4. Set of open preconditions:

- It specifies which preconditions are not fulfilled by any action in the partial order plan.
- A plan is a solution if the set of open preconditions is empty.

Advantages of Partial Order Planning:

1. Solves a huge state space problem in only a few steps.
2. Least commitment strategy means that search only occurs in places where sub-plans interact & delays the choice during the search.
3. Causal links used in this planner allows to recognize when to abandon a unworkable plan without wasting time exploring irrelevant parts.

The solution for partial order planning is a complete, consistent plan.

1. A complete plan: Every precondition of every step is achieved by some other step.
2. A consistent plan: There are no contradictions in the ordering or causal constraints.