[Unit 4 Artificial Intelligence](https://www.studocu.com/in/document/vignana-bharathi-institute-of-technology/artificial-intelligence/unit-4-artificial-intelligence/100442281?utm_campaign=shared-document&utm_source=studocu-document&utm_medium=social_sharing&utm_content=unit-4-artificial-intelligence)

What is the Role of Planning in Artificial Intelligence?

Artificial intelligence is an important technology in the future. Whether it is intelligent robots, self-driving cars, or smart cities, they will all use different aspects of artificial intelligence!!! But Planning is very important to make any such AI project.

Even Planning is an important part of Artificial Intelligence which deals with the tasks and domains of a particular problem. Planning is considered the logical side of acting.

Everything we humans do is with a definite goal in mind, and all our actions are oriented towards achieving our goal. Similarly, Planning is also done for Artificial Intelligence.

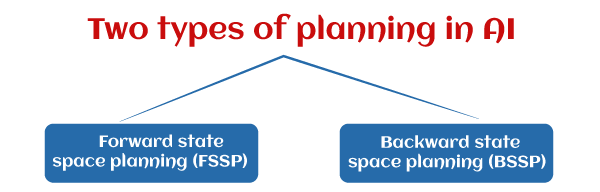
**For** **example**, Planning is required to reach a particular destination. It is necessary to find the best route in Planning, but the tasks to be done at a particular time and why they are done are also very important.

That is why Planning is considered the logical side of acting. In other words, Planning is about deciding the tasks to be performed by the artificial intelligence system and the system's functioning under domain-independent conditions.

# What is a Plan?

We require domain description, task specification, and goal description for any planning system. A plan is considered a sequence of actions, and each action has its preconditions that must be satisfied before it can act and some effects that can be positive or negative.

So, we have **Forward** **State** **Space** **Planning** **(FSSP)** and **Backward** **State** **Space** **Planning** **(BSSP)** at the basic level.



#### Forward State Space Planning (FSSP)

FSSP behaves in the same way as forwarding state-space search. It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' (which also contains some new terms), called a progression. It continues until we reach the target position. Action should be taken in this matter.

* + **Disadvantage**: Large branching factor
  + **Advantage**: The algorithm is Sound

#### Backward State Space Planning (BSSP)

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g, tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for consistency. The action should be relevant in this case.

* + **Disadvantages**: not sound algorithm (sometimes inconsistency can be found)
  + **Advantage**: Small branching factor (much smaller than FSSP)

So for an efficient planning system, we need to combine the features of FSSP and BSSP, which gives rise to target stack planning which will be discussed in the next article.

# What is planning in AI?

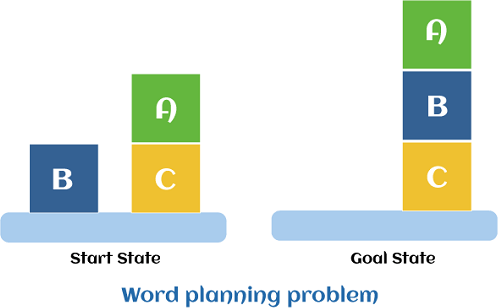
Planning in artificial intelligence is about decision-making actions performed by robots or computer programs to achieve a specific goal.

Execution of the plan is about choosing a sequence of tasks with a high probability of accomplishing a specific task.

#### Block-world planning problem

* + The block-world problem is known as the Sussmann anomaly.
  + The non-interlaced planners of the early 1970s were unable to solve this problem. Therefore it is considered odd.
  + When two sub-goals, G1 and G2, are given, a non-interleaved planner either produces a plan for G1 that is combined with a plan for **G2** or vice versa.
  + In the block-world problem, three blocks labeled 'A', 'B', and 'C' are allowed to rest on a flat surface. The given condition is that only one block can be moved at a time to achieve the target.

###### The start position and target position are shown in the following diagram.



**Components** **of** **the** **planning** **system**

The plan includes the following important steps:

* + Choose the best rule to apply the next rule based on the best available guess.
  + Apply the chosen rule to calculate the new problem condition.
  + Find out when a solution has been found.
  + Detect dead ends so they can be discarded and direct system effort in more useful directions.
  + Find out when a near-perfect solution is found.

# Target stack plan

* + It is one of the most important planning algorithms used by STRIPS.
  + Stacks are used in algorithms to capture the action and complete the target. A knowledge base is used to hold the current situation and actions.
  + A target stack is similar to a node in a search tree, where branches are created with a choice of action.

###### The important steps of the algorithm are mentioned below:

1. Start by pushing the original target onto the stack. Repeat this until the pile is empty. If the stack top is a mixed target, push its unsatisfied sub-targets onto the stack.
2. If the stack top is a single unsatisfied target, replace it with action and push the action precondition to the stack to satisfy the condition.
3. If the stack top is an action, pop it off the stack, execute it and replace the knowledge base with the action's effect.

# Non-linear Planning

This Planning is used to set a goal stack and is included in the search space of all possible sub-goal orderings. It handles the goal interactions by the interleaving method.

###### Advantages of non-Linear Planning

Non-linear Planning may be an optimal solution concerning planning length (depending on the search strategy used).

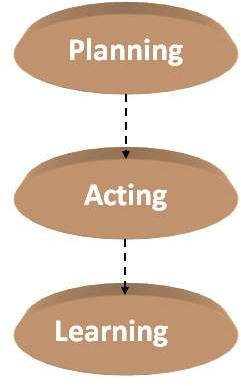
###### Disadvantages of Nonlinear Planning

It takes a larger search space since all possible goal orderings are considered.

**Classical** **Planning**

Classical Planning is the planning where an agent takes advantage of the problem structure to construct complex plans of an action. The agent performs three tasks in classical planning:

* **Planning:** The agent plans after knowing what is the problem.
* **Acting:** It decides what action it has to take.
* **Learning:** The actions taken by the agent make him learn new things.



A language known as **PDDL(Planning** **Domain** **Definition** **Language)** which is used to represent all actions into one action schema.

**PDLL** **describes** **the** **four** **basic** **things** **needed** **in** **a** **search** **problem:**

* **Initial** **state:** It is the representation of each state as the conjunction of the ground and functionless atoms.
* **Actions:** It is defined by a set of action schemas which implicitly define the **ACTION()** and **RESULT()** functions.
* **Result:** It is obtained by the set of actions used by the agent.
* **Goal:** It is same as a precondition, which is a conjunction of literals (whose value is either positive or negative).

**There** **are** **various** **examples** **which** **will** **make** **PDLL** **understandable:**

* Air cargo transport
* The spare tire problem
* The blocks world and many more.

**Let’s** **discuss** **one** **of** **them**

* **Air** **cargo** **transport**

**This** **problem** **can** **be** **illustrated** **with** **the** **help** **of** **the** **following** **actions:**

* **Load:** This action is taken to load cargo.
* **Unload:** This action is taken to unload the cargo when it reaches its destination.
* **Fly:** This action is taken to fly from one place to another.

Therefore, the Air cargo transport problem is based on loading and unloading the cargo and flying it from one place to another.

**Below** **is** **the** **PDLL** **description** **for** **Air** **cargo** **transport:**

Init (On(C1, SFO) ? On(C2, JFK) ? On(P1, SFO) ? On(P2, JFK)?

Cargo(C1) ? Cargo(C2) ? Plane(P1) ? Plane(P2)

? Airport (JFK) ? Airport (SFO)) Goal (On(C1, JFK) ? On(C2, SFO))

Action(Load (c, p, a),

PRECOND: On(c, a) ? On(p, a) ? Cargo(c) ? Plane(p) ? Airport (a)

EFFECT: ? On(c, a) ? In(c, p))

Action(Unload(c, p, a),

PRECOND: In(c, p) ? On(p, a) ? Cargo(c) ? Plane(p) ? Airport (a)

EFFECT: On(c, a) ? ?In(c, p))

Action(Fly(p, from, to),

PRECOND: On(p, from) ? Plane(p) ? Airport (from) ? Airport (to)

EFFECT: ? On(p, from) ? On(p, to))

**The** **above** **described** **actions,** **(i.e.,** **load,** **unload,** **and** **fly)** **affects** **the** **following** **two** **predicates:**

* **(c,p):** In this, the cargo is inside the plane **p**.
* **(x,a):** In this, the object **x** is at the airport **a**. Here, object can be the **cargo** or **plane**.

*It* *is* *to* *be* *noted* *that* *when* *the* *plan* *flies* *from* *one* *place* *to* *another,* *it* *should* *carry* *all* *cargo* *inside* *it.* *It* *becomes* *difficult* *with* *the* *PDLL* *to* *give* *solution* *for* *such* *a* *problem.* *Because* *PDLL* *do* *not* *have* *the* *universal* *quantifier.* **Thus,** **the** **following** **approach** **is** **used:**

* piece of cargo ceases to be**On** anywhere when it is In a plane.
* the cargo only becomes**On** the new airport when it is unloaded.

**Therefore,** **the** **planning** **for** **the** **solution** **is:**

Load (C1, P1, SFO), Fly(P1, SFO, JFK),Unload(C1, P1, JFK),

Load (C2, P2, JFK), Fly(P2, JFK, SFO),Unload(C2, P2, SFO)] .

**Note:** Some problems can be ignored because they does not cause any problem in planning.

* **The** **spare** **tire** **problem**

The problem is that the agent needs to change the flat tire. The aim is to place a good spare tire over the car’s axle. **There** **are** **four** **actions** **used** **to** **define** **the** **spare** **tire** **problem:**

* 1. Remove the spare from the trunk.
  2. Remove the flat spare from the axle.
  3. Putting the spare on the axle.
  4. Leave the car unattended overnight. Assuming that the car is parked at an unsafe neighborhood.

**The** **PDLL** **description** **for** **the** **spare** **tire** **problem** **is:**

Init(Tire1(Flat ) ? Tire1(Spare) ? At(Flat , Axle) ? At(Spare, Trunk ))

Goal (At(Spare, Axle)) Action(Remove(obj , loc), PRECOND: At(obj , loc)

EFFECT: ? At(obj , loc) ? At(obj , Ground)) Action(PutOn(t , Axle),

PRECOND: Tire1(t) ? At(t , Ground) ?¬At(Flat , Axle) EFFECT: ? At(t , Ground) ? At(t , Axle)) Action(LeaveOvernight ,

PRECOND:

EFFECT: ? At(Spare, Ground) ?¬At(Spare, Axle) ?¬At(Spare, Trunk)

?¬At(Flat, Ground) ?¬At(Flat , Axle) ?¬At(Flat, Trunk))

**The** **solution** **to** **the** **problem** **is:**

[Remove(Flat,Axle),Remove(Spare,Trunk), PutOn(Spare, Axle)].

*Similarly,* *we* *can* *design* *PDLL* *for* *various* *problems.*

**Complexity** **of** **the** **classical** **planning**

**In** **classical** **planning,** **there** **occur** **following** **two** **decision** **problems:**

1. **PlanSAT:** It is the question asking if there exists any plan that solves a planning problem.
2. **Bounded** **PlanSAT:** It is the question asking if there is a solution of length k or less than it.

**We** **found** **that:**

* + **PlanSAT** and **Bounded** **PlanSAT** are decidable for classical planning.
  + Both decision problems lie in the complexity class PSPACE, which is larger than NP.

**Note:** **PSPACE** is the class which refers to those problems that can be solved via deterministic Turing machine under a polynomial time space.

**From** **the** **above,** **it** **can** **be** **concluded** **that:**

1. PlanSAT is P whereas Bounded PlanSAT is NP-complete.
2. Optimal planning is hard with respect to sub-optimal planning.

**Advantages** **of** **Classical** **Planning**

**There** **are** **following** **advantages** **of** **Classical** **planning:**

* + It has provided the facility to develop accurate domain-independent heuristics.
  + The systems are easy to understand and work efficiently.

**State** **Space** **Search** **in** **Artificial** **Intelligence**

A **state** **space** is a way to mathematically represent a problem by defining all the possible states in which the problem can be. This is used in search algorithms to represent the initial state, goal state, and current state of the problem. Each state in the state space is represented using a set of variables.

## Features of State Space Search

**State** **space** **search** has several features that make it an effective problem-solving technique in Artificial Intelligence. These features include:

###### Exhaustiveness:

State space search explores all possible states of a problem to find a solution.

###### Completeness:

If a solution exists, state space search will find it.

###### Optimality:

Searching through a state space results in an optimal solution.

###### Uninformed and Informed Search:

State space search in artificial intelligence can be classified as uninformed if it

provides additional information about the problem.

In contrast, informed search uses additional information, such as heuristics, to guide the search process.

## Steps in State Space Search

The steps involved in state space search are as follows:

* + To begin the search process, we set the current state to the initial state.
  + We then check if the current state is the goal state. If it is, we terminate the algorithm and return the result.
  + If the current state is not the goal state, we generate the set of possible successor states that can be reached from the current state.
  + For each successor state, we check if it has already been visited. If it has, we skip it, else we add it to the queue of states to be visited.
  + Next, we set the next state in the queue as the current state and check if it's the goal state. If it is, we return the result. If not, we repeat the previous step until we find the goal state or explore all the states.
  + If all possible states have been explored and the goal state still needs to be found, we return with no solution.

## State Space Representation

**State** **space** **Representation** involves defining an INITIAL STATE and a GOAL STATE and then determining a sequence of actions, called states, to follow.

###### State:

A state can be an Initial State, a Goal State, or any other possible state that can be

generated by applying rules between them.

###### Space:

In an AI problem, space refers to the exhaustive collection of all conceivable states.

###### Search:

This technique moves from the beginning state to the desired state by applying good

rules while traversing the space of all possible states.

###### Search Tree:

To visualize the search issue, a search tree is used, which is a tree-like structure that

represents the problem. The initial state is represented by the root node of the search tree, which is the starting point of the tree.

###### Transition Model:

This describes what each action does, while Path Cost assigns a cost value to each

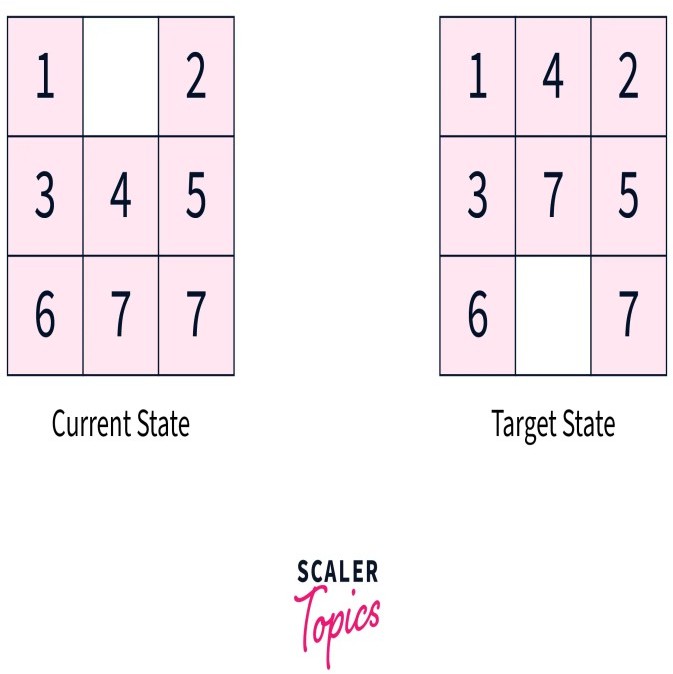
path, an activity sequence that connects the beginning node to the end node. The optimal option has the lowest cost among all alternatives.

Example of State Space Search

The 8-puzzle problem is a commonly used example of a state space search. It is a sliding puzzle game consisting of 8 numbered tiles arranged in a 3x3 grid and one blank space. The game aims to rearrange the tiles from their initial state to a final goal state by sliding them into the blank space.

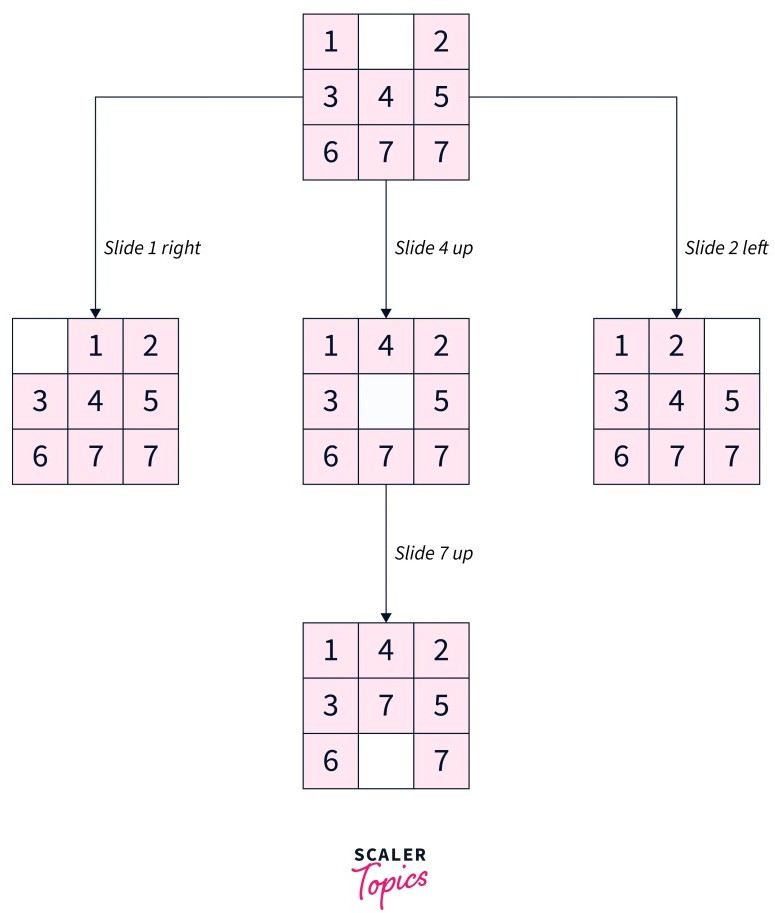
To represent the state space in this problem, we use the nine tiles in the puzzle and their respective positions in the grid. Each state in the state space is represented by a 3x3 array with values ranging from 1 to 8, and the blank space is represented as an empty tile.

The initial state of the puzzle represents the starting configuration of the tiles, while the goal state represents the desired configuration. Search algorithms utilize the state space to find a sequence of moves that will transform the initial state into the goal state.



This algorithm guarantees a solution but can become very slow for larger state spaces. Alternatively, other algorithms, such as **A** **search**, use heuristics to guide the search more efficiently.

Our objective is to move from the current state to the target state by sliding the numbered tiles through the blank space. Let's look closer at reaching the target state from the current state.



To summarize, our approach involved exhaustively exploring all reachable states from the current state and checking if any of these states matched the target state.

## Applications of State Space Search

* + State space search algorithms are used in various fields, such as robotics, game playing, computer networks, operations research, bioinformatics, cryptography, and supply chain management. In artificial intelligence, state space search algorithms can solve problems like **pathfinding**, **planning**, and **scheduling**.
  + They are also useful in planning robot motion and finding the best sequence of actions to achieve a goal. In games, state space search algorithms can help determine the best move for a player given a particular game state.
  + **State** **space** **search** **algorithms** can optimize routing and resource allocation in computer networks and operations research.
  + In **Bioinformatics**, state space search algorithms can help find patterns in biological data and predict protein structures.
  + In **Cryptography**, state space search algorithms are used to break codes and find cryptographic keys.

##### Planning Graphs in AI

A Planning Graph is similar to a valid plan, but without the requirement that the actions at a given time step not interfere. It is, in essence, a type of constraint graph that encodes the planning problem. More precisely, a Planning Graph is a directed, leveled graph2 with two kinds of nodes and three kinds of edges.

Planning Graphs

* Planning graphs are an efÏcient way to create a representation of a planning problem that can be used to /Achieve better heuristic estimates

/Directly construct plans

* Planning graphs only work for propositional problems

Planning graphs consists of a sequence of levels that correspond to time steps in the plan. / Level 0 is the initial state. / Each level consists of a set of literals and a set of actions that represent what might be possible at that step in the plan / Might be is the key to efÏciency /Records only a restricted subset of possible negative interactions among actions.

Each level consists of

* Literals = all those that could be true at that time step, depending upon the actions executed at preceding time steps.
* Actions = all those actions that could have their preconditions satisfied at that time step, depending on which of the literals actually hold.

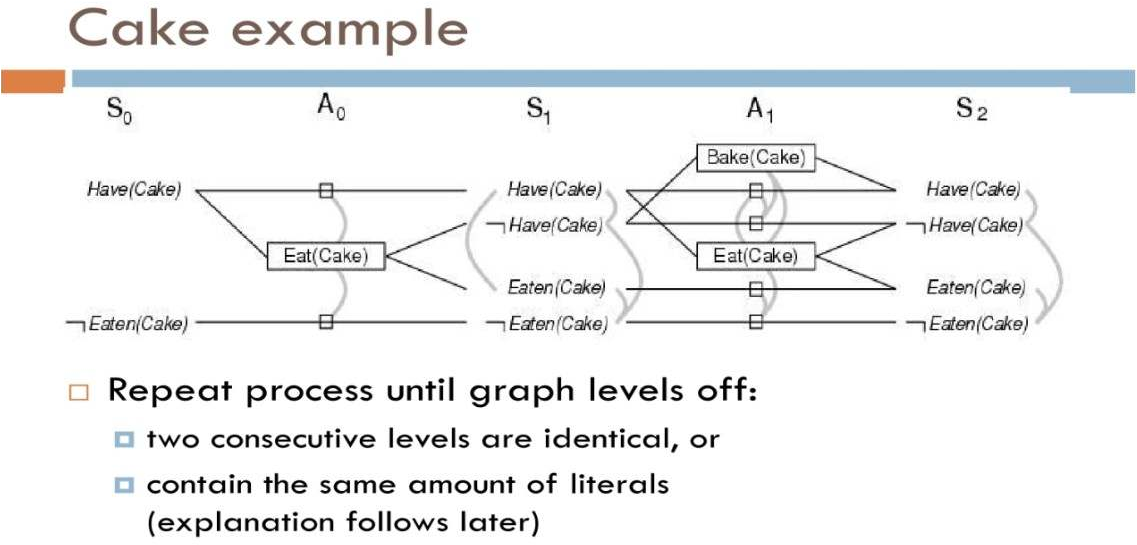
##### EXAMPLE of planning a Graph

Init(Have(Cake)) Goal(Have(Cake)  Eaten(Cake)) Action(Eat(Cake),

PRECOND: Have(Cake)

EFFECT: ¬Have(Cake)  Eaten(Cake)) Action(Bake(Cake),

PRECOND: ¬ Have(Cake) EFFECT: Have(Cake))



### What is hierarchical planning with example?

A planning hierarchy is a combination of characteristic values based on the characteristics of one information structure . Planning hierarchies provide a framework for your planning activities in consistent planning and level-by-level planning.

Hierarchical planning is also called as plan decomposition. Generally plans are organized in Hierarchical format.

Pop one level planner:

If you are planning to take a trip, then first you have to decide the location. To decide location we can search for various good locations from internet based on, whether conditions, travelling expenses, etc. This is level one planning.

Hierarchy of actions:

In terms of major and minor or actions, hierarchy of actions can be decided. Minor activities would cover more precise activities to accomplish the major activities.

Major steps are given more importance. Once major steps are decided we attempt to solve the minor detailed actions.

Planner:

1. First identify a hierarchy of major conditions.
2. Construct a plan in levels.
3. Patch major level details.
4. Finally demonstrate

##### Multi Agent planning:

"Multiagent planning is **concerned** **with** **planning** **by** **(and** **for)** **multiple** **agents**. It can involve agents planning for a common goal, an agent coordinating the plans (plan merging) or planning of others, or agents refining their own plans while negotiating over tasks or resources.

Multiagent planning problems

In general, a multiagent planning problem can be defined as the problem of planning by and for a group of agents. Except for more centralized (multiagent) planning problems, each agent in such a problem has in fact a private, individual planning problem.

### What is an example of a multi-agent?

###### Applications

* Robotic Logistics and Planning. Coordinating a large swarm of robots requires advanced planning algorithms, and this a fundamental cornerstone of MAS research.

...

* Autonomous Vehicles. ...
* Agent Based Modelling. ...
* Video Games.