**Problem Solving by Searching in Artificial Intelligence**

**Problem Solving by Searching**

Whenever the [agent](https://www.includehelp.com/ml-ai/artificial-intelligence-based-agent.aspx) is confronted by a problem, its first action is seeking a solution is its knowledge system. This is known as the search for the solution in the knowledge base. Another attempt can be to search for a solution by going into different states. The search of the agent stops in the state when the agent reaches the goal state.

There are many approaches for searching a particular goal state from all the states that the agent can be in.

## State Space Search

A state space is a mathematical representation of a problem that defines all possible states that the problem can be in. Furthermore, in search algorithms, we use a state space to represent the current state of the problem, the initial state, and the goal state. Additionally, we represent each state in the state space by a set of variables.

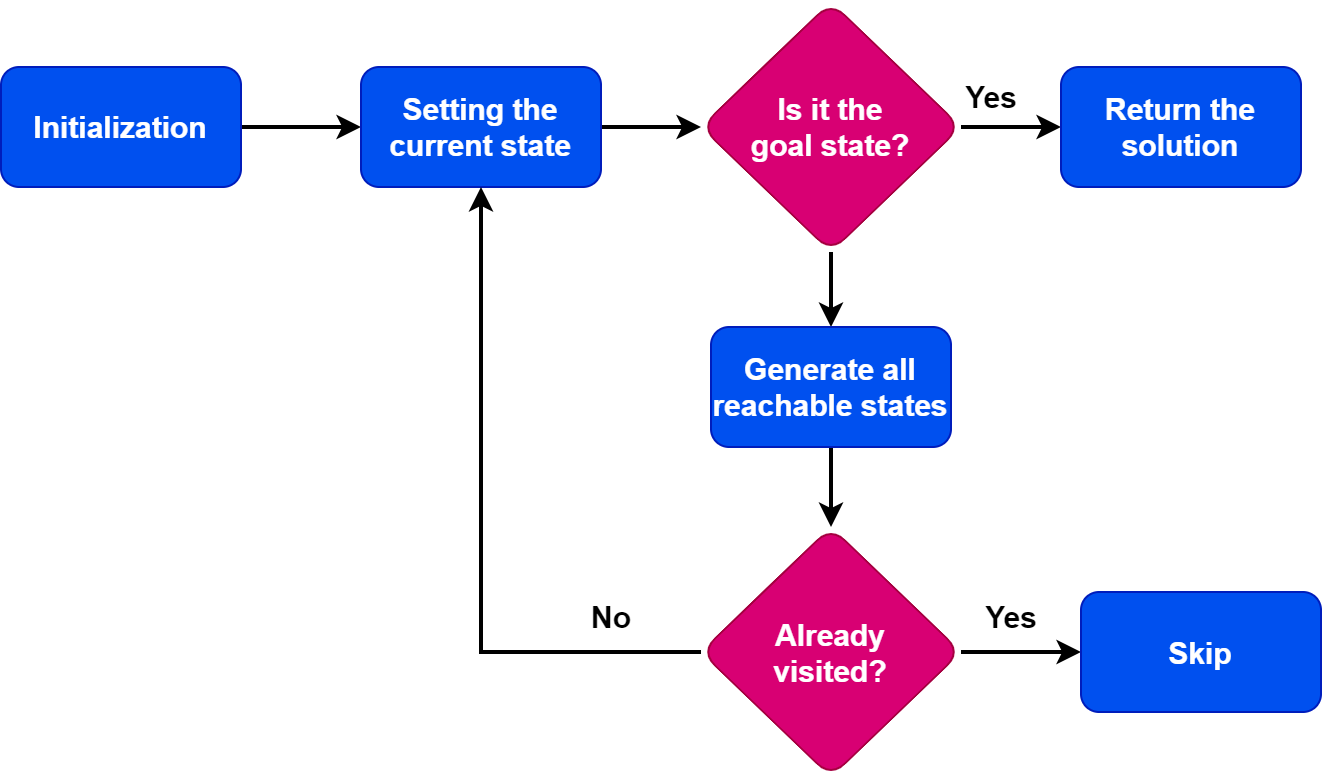
**State space search is a method used widely in**[artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence)**and computer science to find a solution to a problem by searching through the set of possible states of the problem.** Furthermore, a state space search **algorithm uses the state space to navigate from the initial state to the goal state. Additionally, it generates and explores possible** successors of the current state until we find a solution.

The state space size can greatly affect a search algorithm’s efficiency. Hence, it’s important to choose an appropriate representation and search strategy to efficiently search the state space.

The most famous state space search algorithm is the [A\* algorithm](https://www.baeldung.com/cs/a-star-algorithm). Other popular state space search algorithms are [breadth-first search (BFS)](https://www.baeldung.com/cs/dfs-vs-bfs), [depth-first search (DFS)](https://www.baeldung.com/cs/dfs-vs-bfs), [hill climbing](https://www.baeldung.com/cs/hill-climbing-algorithm), [simulated annealing](https://en.wikipedia.org/wiki/Simulated_annealing), and [genetic algorithms](https://www.baeldung.com/cs/genetic-algorithms-applications).

## Steps

Now let’s discuss the steps of a typical state space search algorithm:



The first step is to initialize the search by setting the initial state as the current state. After the initialization step, we check if the current state is a goal state. Finally, if the current state is a goal state, we terminate the algorithm and return the result.

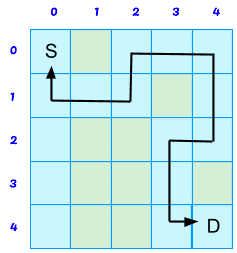
However, if the current state is not the goal state, we generate the set of possible states that can be reached from the current state. Additionally, these states are known as successor states. Furthermore, for each successor state, we check if it has been previously visited. If the state is already explored, we skip the state. If it has not been visited, we add it to the queue of states to be visited.

Moving forward, we set the next state in the queue as the current state and check if it’s a goal state. If we find the goal state, we return the result. Otherwise, we repeat the previous step until we find the goal state or finish exploring all the states. Furthermore, if all possible states have been explored and we can’t reach the target state, we return with no solution.

The specific implementation details of the algorithm depend on the problem. Additionally, **the algorithm’s** performance depends on the data structures we use to represent the states and keep track of the search.

**State Space Search Examples:**

Example 1. Maze

[](https://www.vtupulse.com/wp-content/uploads/2022/01/maze.png)

A maze problem can be represented as a state-space

* Each state represents “where you are” that is the current position in the maze
* The start state or initial state represents your starting position
* The goal state represents the exit from the maze

Rules (for a rectangular maze) are: move north, move south, move east, and move west

* Each rule takes you to a new state (maze location)
* Rules may not always apply, because of walls in the maze

**Production system**

A production system is based on a set of rules about behavior. These rules are a basic representation found helpful in expert systems, automated planning, and action selection. It also provides some form of [artificial intelligence](https://www.edureka.co/ai-deep-learning-with-tensorflow). In this article, we will talk about the production system in artificial intelligence in the following sequence:

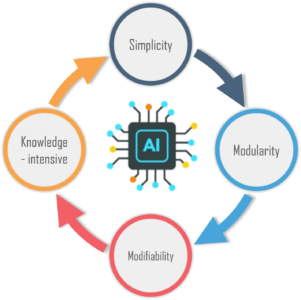
### ****Components of Production System****

The major components of Production System in [Artificial Intelligence](https://www.edureka.co/blog/artificial-intelligence-tutorial/) are:

* **Global Database:** The global database is the central data structure used by the production system in Artificial Intelligence.
* **Set of Production Rules:** The production rules operate on the global database. Each rule usually has a precondition that is either satisfied or not by the global database. If the precondition is satisfied, the rule is usually be applied. The application of the rule changes the database.
* **A Control System:** The control system then chooses which applicable rule should be applied and ceases computation when a termination condition on the database is satisfied. If multiple rules are to fire at the same time, the control system resolves the conflicts.

**Features of Production System in Artificial Intelligence**

The main features of the production system include:



**1. Simplicity:** The structure of each sentence in a production system is unique and uniform as they use the “IF-THEN” structure. This structure provides simplicity in [knowledge representation](https://www.edureka.co/blog/knowledge-representation-in-ai/). This feature of the production system improves the readability of production rules.

**2. Modularity:** This means the production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deleterious side effects.

**3. Modifiability:** This means the facility for modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

**4. Knowledge-intensive:** The knowledge base of the production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation.

**Control/Search Strategies**

How would you decide which rule to apply while searching for a solution for any problem? There are certain requirements for a good control strategy that you need to keep in mind, such as:

* The first requirement for a good control strategy is that it should **cause motion**.
* The second requirement for a good control strategy is that it should be **systematic**.
* Finally, it must be **efficient** in order to find a good answer.

**Production System Rules**

Production System rules can be classified as:

* **Deductive Inference Rules**
* **Abductive Inference Rules**

You can represent the knowledge in a production system as a set of rules along with a control system and database. It can be written as:

***If(Condition) Then (Condition)***

The production rules are also known as condition-action, antecedent-consequent, pattern-action, situation-response, feedback-result pairs.

**Classes of Production System in Artificial Intelligence**

There are four major classes of Production System in Artificial Intelligence:

* **Monotonic Production System**: It’s a production system in which the application of a rule never prevents the later application of another rule, that could have also been applied at the time the first rule was selected.
* **Partially Commutative Production System**: It’s a type of production system in which the application of a sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state x into state Y. Theorem proving falls under the monotonic partially communicative system.
* **Non-Monotonic Production Systems**: These are useful for solving ignorable problems. These systems are important from an implementation standpoint because they can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path was followed. This production system increases efficiency since it is not necessary to keep track of the changes made in the search process.
* **Commutative Systems**: These are usually useful for problems in which changes occur but can be reversed and in which the order of operation is not critical. Production systems that are not usually not partially commutative are useful for many problems in which irreversible changes occur, such as chemical analysis. When dealing with such systems, the order in which operations are performed is very important and hence correct decisions must be made at the first attempt itself.

**Advantages & Disadvantages**

Some of the **advantages** of Production system in artificial intelligence are:

* Provides **excellent tools** for structuring AI programs
* The system is highly **modular** because individual rules can be added, removed or modified independently
* Separation of **knowledge** and **Control-Recognises Act Cycle**
* A natural **mapping** onto state-space research data or goal-driven
* The system uses pattern directed control which is more **flexible** than algorithmic control
* Provides opportunities for **heuristic control** of the search
* A good way to model the **state-driven nature** of intelligent machines
* Quite helpful in**a real-time** environment and applications.

 Now, let’s have a look at some of the**disadvantages**:

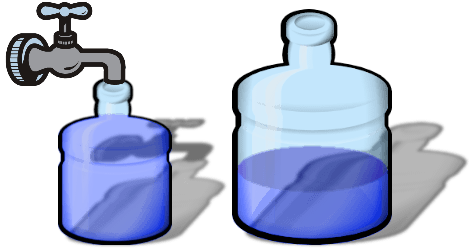
 It is very **difficult** to analyze the flow of control within a production system

* It describes the operations that can be performed in a search for a solution to the problem.
* There is an **absence of learning** due to a rule-based production system that does not store the result of the problem for future use.
* The rules in the production system should not have any type of **conflict resolution** as when a new rule is added to the database it should ensure that it does not have any conflict with any existing rule.

**Production System in Artificial Intelligence: Example**

**Problem Statement:**

We have two jugs of capacity 5l and 3l (liter), and a tap with an endless supply of water. The objective is to obtain 4 liters exactly in the 5-liter jug with the minimum steps possible.



**Production System:**

* Fill the 5 liter jug from tap
* Empty the 5 liter jug
* Fill the 3 liter jug from tap
* Empty the 3 liter jug
* Then, empty the 3 liter jug to 5 liter
* Empty the 5 liter jug to 3 liter
* Pour water from 3 liters to 5 liter
* Pour water from 5 liters to 3 liters but do not empty

**Solution:**

**1,8,4,6,1,8**or**3,5,3,7,2,5,3,5;**

**Problem Characteristics, its classification and Issues in the design of search programs**

In order to choose the most appropriate problem solving method, it is necessary to analyze the problem along various key dimensions.

These dimensions are referred to as problem characteristics discussed below.

**Is the problem decomposable into a set of independent smaller or easier sub-problems?**

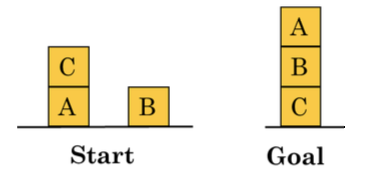
A very large and composite problem can be easily solved if it can be broken into smaller problems and recursion could be used.

For example, we want to solve :- ∫ 𝑥2 + 3𝑥 + 𝑠𝑖𝑛2𝑥 𝑐𝑜𝑠2𝑥 𝑑𝑥

This can be done by breaking it into three smaller problems and solving each by applying specific rules. Adding the results we can find the complete solution.

But there are certain problems which cannot be decomposed into sub-problems.

For example Blocks world problem in which, start and goal state are given as,



Here, solution can be achieved by moving blocks in a sequence such that goal state can be derived.

Solution steps are interdependent and cannot be decomposed in sub problems.

These two examples, symbolic integration and the blocks would illustrate the difference between decomposable and non-decomposable problems.

## ****Can solution steps be ignored or at least undone if they prove unwise?****

Problem fall under three classes, (i) ignorable, (ii) recoverable and (iii) irrecoverable.

This classification is with reference to the steps of the solution to a problem.

Consider theorem proving. We may later find that it is of no use. We can still proceed further, since nothing is lost by this redundant step. This is an example of ignorable solutions steps.

Now consider the [8 puzzle problem](https://medium.com/@dpthegrey/8-puzzle-problem-2ec7d832b6db) tray and arranged in specific order.

While moving from the start state towards goal state, we may make some stupid move but we can backtrack and undo the unwanted move. This only involves additional steps and the solution steps are recoverable.

Lastly consider the [game of chess](https://medium.com/@dpthegrey/program-to-play-chess-e992610cddc9). If a wrong move is made, it can neither be ignored nor be recovered. The thing to do is to make the best use of current situation and proceed. This is an example of an irrecoverable solution steps.

Knowledge of these will help in determining the control structure.

* Ignorable problems can be solved using a simple control structure that never backtracks.
* Recoverable problems can be solved by a slightly more complicated control strategy that allows backtracking.
* Irrecoverable problems will need to be solved by a system that expands a great deal of effort making each decision since decision must be final.

## ****Is the problem’s universe predictable?****

Problems can be classified into those with certain outcome ([eight puzzle](https://medium.com/@dpthegrey/8-puzzle-problem-2ec7d832b6db) and [water jug](https://medium.com/@dpthegrey/water-jug-problem-fa14e039e730) problems) and those with uncertain outcome (playing cards).

In certain — outcome problems, planning can be done to generate a sequence of operators that guarantees to lead to solution.

Planning helps to avoid unwanted solution steps.

For uncertain outcome problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution.

The uncertain outcome problems do not guarantee a solution and it is often very expensive since the number of solution paths to be explored increases exponentially with the number of points at which the outcome cannot be predicted.

Thus one of the hardest types of problems to solve is the irrecoverable, uncertain — outcome problems (Ex:- Playing cards.)

## Is a good solution to the problem obvious without comparison to all other possible solutions?

There are two categories of problems — Any path problem and Best path problem.

In any path problem, like the [water jug](https://medium.com/@dpthegrey/water-jug-problem-fa14e039e730) and [8 puzzle](https://medium.com/@dpthegrey/8-puzzle-problem-2ec7d832b6db) problems, we are satisfied with the solution, irrespective of the solution path taken.

Whereas in the other category not just any solution is acceptable but we want the best path solution.

Like that of travelling sales man problem, which is the shortest path problem.

In any — path problems, by heuristic methods we obtain a solution and we do not explore alternatives.

Any path problems can often be solved in a reasonable amount of time by using heuristics that suggest good paths to explore.

For the best-path problems all possible paths are explored using an exhaustive search until the best path is obtained.

Best path problems are computationally harder.

## Is the desired solution a state of the world or a path to a state?

Consider the problem of natural language processing.

Finding a consistent interpretation for the sentence “The bank president ate a dish of pasta salad with the fork”.

We need to find the interpretation but not the record of the processing by which the interpretation is found.

Contrast this with the [water jug problem](https://medium.com/@dpthegrey/water-jug-problem-fa14e039e730).

In [water jug problem](https://medium.com/@dpthegrey/water-jug-problem-fa14e039e730), it is not sufficient to report that we have solved, but the path that we found to the state (2, 0). Thus the statement of a solution to this problem must be a sequence of operations that produces the final state.

## What is the role of knowledge?

Though one could have unlimited computing power, the size of the knowledge base available for solving the problem does matter in arriving at a good solution.

Take for example the [game of playing chess](https://medium.com/@dpthegrey/program-to-play-chess-e992610cddc9), just the rules for determining legal moves and some simple control mechanism is sufficient to arrive at a solution.

But additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program. The solution would then be realistic.

Consider the case of predicting the political trend. This would require an enormous amount of knowledge even to be able to recognize a solution, leave alone the best.

## Does the task require interaction with a person?

The problems can again be categorized under two heads.

1. Solitary in which the computer will be given a problem description and will produce an answer, with no intermediate communication and with the demand for an explanation of the reasoning process. Simple theorem proving falls under this category. Given the basic rules and laws, the theorem could be proved, if one exists.
2. Conversational, in which there will be intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user, or both, such as medical diagnosis fall under this category, where people will be unwilling to accept the verdict of the program, if they cannot follow its reasoning.

## Problem Classification

Actual problems are examined from the point of view of all these questions; it becomes apparent that there are several broad classes into which the problems fall.

## Issues in the design of search programs

1. The direction in which to conduct search (forward versus backward reasoning). If the search proceeds from start state towards a goal state, it is a forward search or we can also search from the goal.
2. How to select applicable rules (Matching). Production systems typically spend most of their time looking for rules to apply. So, it is critical to have efficient procedures for matching rules against states.
3. How to represent each node of the search process (knowledge representation problem).