CHAPTER

1

Artificial Intelligence : Approaches to AI

Intelligence: It is an ability to learn OR understand from the experience. It is the ability to learn and retain the knowledge, the ability to respond quickly to a new situation, ability of reason (apply the logic), etc.

Artificial Intelligence (AI) is the study of how to make computers do things which, at the moment, people do better.
In other words, Artificial Intelligence is the branch of computer science in which intelligence is incorporated in an artificial device. Purpose of AI is to create an Human clone.

The central problem or goal of AI research include reasoning knowledge, planning, learning, natural language processing, perception and ability to manipulate objects. The AI field is interdisciplinary in which a number of sciences and professional coverage including computer science, mathematics psychology, linguistics, philosopy as well as specialized fields such as artificial psychology.

At the heart of research in artificial intelligence lies what Newell and Simon [1976] call the physical symbol system hypothesis. They define a physical symbol system as follows:
A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus, a symbol structure is composed of a number of instances (tokens) of symbols related in some physical way (such as one token being next to another). At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on expressions to produce other expressions: processes through time an envolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves.

Physical symbol system hypothesis : A physical symbol system has the neccessary and sufficient means for general intelligent action.

- Symbol is meaningful pattern that can be manipulated. Example : word, sentence etc.
- A symbol system creates, copies, modifies and destroys symbols.
- Term physical is used, because symbols in physical symbol system are physical objects that are part of the real world, even though they may be internal to computers and brains.

Knowledge level : It is a level of abstraction that considers what an agent knows and believes and what it goals are :
The knowledge level considers what an agent knows, but not know it reasons.

AI Technique :

One of the result about AI research states that "intelligence requires knowledge".
The knowledge possesses some less desirable properties as given below:

- ♦ It is voluminous
- ♦ It is hard to characterize accurately
- ♦ It is constantly changing
- ♦ It differs from the data by being organized in a way that corresponds to the ways it will be used.

Thus, we conclude that on AI technique is a method that exploits knowledge that should be represented in such a way that

- The knowledge captures generalization.
- It can be understood by people who must provide it.
- It can easily be modified to correct errors and to reflect changes in the world and its view.
- It can be used in a great many situations even if it is not totally accurate or complete.
- It can be used to overcome its own sheer bulk by helping to narrow the range of possibilities that must usually be considered.

It is possible to solve AI problems without using AI techniques and it is also possible to apply AI techniques to the solution of non AI problems.

Types of Artificial intelligence

1. **Strong AI :** It is also known as artificial general intelligence (A.G.I.)
   It is A.I. system with generalized cognitive abilities which find solutions to the unfamilier task it comes across.
   It aims to build a machine which can truely reason and solve the problem.
   - It is self aware
   - Its overall intelligence ability is indistinguishable from the human being
2. **Weak AI :** Also known as narrow A.I. It is an A.I. system that is developed and trained for a particular task. It is very good at routine physical and cognitive jobs. To create a machine that can not truely reason and solve the problem but its behaviour in such a way that it is very intelligent.
   Example : Narrow A.I. can identify pattern and correlations from data more efficiently than humans.
   Sales predictions, Purchase suggestions, Google's translation engine are the implementation of narrow A.I.
   In the automotive industry, self driving cars are the result of coordination of several narrow A.I.
   It can not expand and take tasks beyond its field.
3. **Applied AI :** To build commercially smart system.
4. **Cognitive AI :** It perform tests on the way the human mind work simulation.

Uncertainty : A.I. agents have to act in non-deterministic environments a self driving car cannot be 100% confident what other cars pedestrians are going to do.
In medicine, the measurable data are often insufficient to predict the outcome of a treatment with full certainty. A.I. agents have to represent these uncertainties somehow to act optimally.

Environment uncertainty : It is the property of the observable environment. Perhaps the simplest example of environment uncertainty is a coin flip.

When we flip a coin, the outcome will be heads or tail with 50 : 50 chance.

Model uncertainty : Our agent's own lack of knowledge about the environment. For example, the agent is in castle and has to choose between two doors : one of them leading to the princess will be there or not.

There is no environment uncertainty in this case, the princess is in one of the rooms, and if you try the same door twice, you will get the same answer deterministically. The agent's uncertainty now stems from the fact that it has incomplete knowledge about the environment.

Note : Two kinds of uncertainty can be present at the same time. The agent can be uncertain about how uncertain the outcome of an action is.

Uncertain knowledge representation : Logic and Uncertainty : One problem with logical approaches :

Agents almost never have access to the whole truth about their environments.

- Very often, even in simple worlds, there are important questions for which there is no boolean answer.
- In that case, an agent must reason under uncertainty.
- Uncertainty also arieses because of an agent's incomplete or incorrect understanding of its environment.

Uncertainty : Let action L_t = "leave for airport t minutes before flight".

Will L_t get me there on time ?

Problems :

→ Partial observability (road state, other driver's plan).

→ Noisy sensors (unreliable traffic reports).

→ Uncertainty in action outcomes (flat tire, etc.).

→ Immense complexity of modeling and predicting traffic.

Hence a purely logical approach either :

1. Risk falsehood ("A_{25} will get me there on time"), or
2. Leads to conclusion that are too weak for decision making ("A_{25} will get me there on there if there is no accident on the way, it doesn't rain, my tires remain intact, ...").

Handling uncertain knowledge :

FOL-based approaches fail to cope with domains like medical diagnosis.

→ Laziness : Too much work to write complete oxions or too hard to work with the enormous sentences that result.

→ Theoretical Ignorance : The available knowledge of the domain is complete.

→ Practical Ignorance : The theoretical knowledge of the domain is complete but some evidential facts are missing.

Six Types of A.I. environment : There are several categories we use to group A.I. problems based on the nature of the problem.

1. **Complete vs. Incomplete :** A.I. environment are those on which, at any given time, have enough information to comeplete a branch of the problem.
   For example : Chess
   A.I. strategies can't anticipate many moves in advance and instead, they focus on finding a good "equilibrium" at any given time, poker is an example of incomplete environments.
2. **Fully observable vs. Partially observable :** It has access to all required information to complete target task. Image recognition operates in fully observable domains. Self driving vehicle scenarios deal with partial info in order to solve A.I. problems.
3. **Competitive :** It face A.I. agents against each other in order to optimize a specific outcome. Example : Game as go or chess.
   Collaboration : It rely on the cooperation between multiple A.I. agents. Self driving vehicles or cooperating to avoid collisions or smart home sensors interactions are examples of collaborations A.I. environments.
4. **Static :** Data knowledge sources that don't change frequently over-time.
   Dynamic : Change frequently. Example : vision A.I. system in drones.
5. **Discrete :** Finite set of possibilities can drive the final outcome of the task. For example : Chess.
   Continuous : It rely on unknown and rapidly changing data sources. For example : Self driving cars.
6. **Deterministic :** Outcome can be determined based on specific state.

Stochastic : Most real state world A.I. environments are not deterministic. Instead, they can be classified as stochastic. For example : Self driving vehicles.

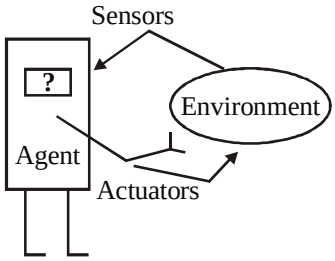
Agents in artificial intelligence : A.I. is defined as study of rational agents.

A rational agent could be anything which makes decisions, like a person, firm, machine or software, it carries out an action with the best outcome after considering past and current percepts (agents perceptual inputs at a given instance).

Artificial Intelligence = Agent + Agent's Environment

Agent : Perceiving its environment through sensors.

Acting upon that environment through actuations.



To understand the structure of intelligent agents, we should be familier with architecture and agent program.

Architecture : is the machinery that the agent executes on. It is a device with sensors and actuators. Example: Robotic car, Camera, P.C.

Agent program : Implementation of an agent function.

Agent function : is a map from the percept sequence (history of all that an agent has perceived till date) to an action.

Agent = Architecture + Agent Program

Types of Agents :

- Simple reflex agents
- Model-based reflex agents
- Goal-based agent
- Utility-based agent
- Learning Agents

Simple reflex agents : Ignore the rest of the percept history and act only on basis current percept. Perceived history is the history of all that an agent has perceived till date.

The agent function is based on the condition-action rule.

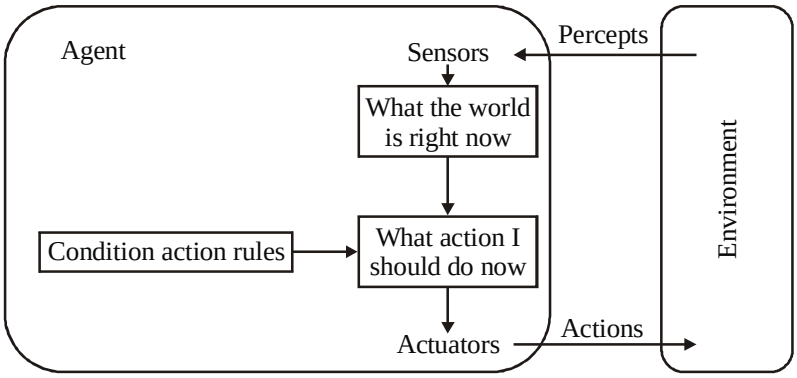
A condition-action rule is a rule that maps a state i.e., condition to an action. If condition is true → action taken else not.

This agent function only succeeds when the environment is fully observable.

- In partial observable. Ex. infinite loops are often unavoidable.
- By randomize its actions could lead to avoidance of infinite loop.

Problems with simple reflex agents are :

- Very limited intelligence.
- No knowledge of non-perceptual parts of state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.



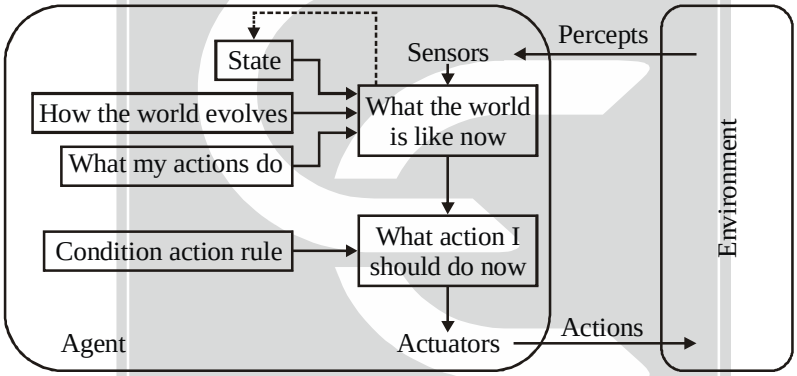
Model-based reflex agents : It works by finding a rule whose condition matches the current situation.

A model-based agent can handle partially observable environments by use of model about the world. The agent has to keep track of internal state which is adjusted by each percept and that depends on the percept history.

The current state is stored inside the agent which maintains some kind of structure describing the part of world which can not be seen.

Updating the states requires the information about.

- How the world evolves in-dependently from the agent.
- How the agent actions affects the world.



Goal based agent : These kind of agents take decision based on how far they are currently from their goal (description of desirable situations).

Their every action is intended to reduce its distance from goal. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning the goal based agent's behaviour can easily be changed.

Utility Based Agents : The agents which are developed having their end uses as building blocks are called utility based agents.

- When there are multiple possible alternatives, then to decide which are is best, utility based agents are used.
- They choose actions based on a preference (utility) for each state.
- We may look for quicker, safer, cheaper trip to reach a destination.
- Here agents happiness should be taken into consideration.
  Utility describes how "happy" the agent is because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility.
- A utility function makes a state on to a real number which describes the associated degree of happiness.

Learning Agents : The essential component of autonomy, this agent is capable of learning from experience, it has the capability of automatic information acquisition and integration into the system. Any agent designed and expected to be successful in an uncertain enviornment is considered to be learning agent.

To build a system to solve a particular problem, we need to do four things:

1. **Define the problem precisely**

   The definition must include precise specifications of what the initial situation(s) will be as well as what final situations constitute acceptable solutions to the problem.

2. **Analyze the problem**

   A few very important features can have an immense impact on the appropriateness of various possible techniques for solving the problem.

3. Isolate and represent the task knowledge that is necessarily to solve the problem.

4. Choose the best problem solving technique(s) and apply it to the particular problem.

Define the problem as a state space search

The state space representations forms the basis of most of the AI methods. Its structure corresponds to the structure of problem solving in two important ways.

- It allows for a formal definition of a problem as the need to convert some given situation into some desired situation using a set of permissible operations.
- It permits us to define the process of solving a particular problem as a combination of known techniques (each represented as a rule defining a single step in the space) and search, the general technique of exploring the space to try to find some path from the current state to a goal state. Search is a very important process in the solution of hard problems for which no more direct techniques are available.

Example : A water jug problem : You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug ?

The state space for this problem can be described as the set of ordered pairs of integers (x, y), such that $x = 0, 1, 3, 4$ and $y = 0, 1, 2, 3$. x represents the number of gallons of water in the 4-gallon jug and y represents the quantity of water in 3 gallon jug.

The start state is (0, 0) and the goal state is $(2, n)$ where n may be 0, 1, 2 or 3.

The operator to solve the problem is given in figure 2.1. There are several sequence of operators that solve the problem. One sequence is shown in figure 2.2.

1.	(x, y) if $x < 4$	$\rightarrow$	$(4, y)$	Fill the 4-gallon jug
2.	(x, y) if $y < 3$	$\rightarrow$	$(x, 3)$	Fill the 3-gallon jug
3.	(x, y) if $x > 0$	$\rightarrow$	$(x - d, y)$	Pour some water out of the 4-gallon jug
4.	(x, y) if $y > 0$	$\rightarrow$	$(x, y - d)$	Pour some water out of the 3-gallon jug
5.	(x, y) if $x > 0$	$\rightarrow$	$(0, y)$	Empty the 4-gallon jug on the ground
6.	(x, y) if $y > 0$	$\rightarrow$	$(x, 0)$	Empty the 3-gallon jug on the ground
7.	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow$	$(4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full
8.	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow$	$(x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full

9.	(x, y) if $x + y \le 4$ and $y > 0$	$\rightarrow$ $(x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10.	(x, y) if $x + y \le 3$ and $x > 0$	$\rightarrow$ $(0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11.	$(0, 2)$	$\rightarrow$ $(2, 0)$	Pour the 2-gallons from the 3-gallon jug into the 4-gallon jug
12.	$(2, y)$	$\rightarrow$ $(0, y)$	Empty the 2-gallons in the 4-gallon jug on the ground

- **Problem Statement:** Three missionaries and three cannibals want to cross a river. There is a boat on their side of the river that can be used by either one or two persons.
  How should they use this boat to cross the river in such a way that cannibals never outnumber missionries on either side of the river? If the cannibals ever outnumber the missionaries (on either bank) then the missionaries will be eaten. How can they all cross over without anyone being eatent?
- State space for this problem can be described as the set of ordered pairs of left and right bank of the river as (L, R) where each bank is represented as a list [nM, mC, B].
  n is the number of missionaries M, m is the number of cannibals C, and B represents boat.
- **Start State:** ([3M, 3C, 1B], [0M, 0C, 0B]),
  1B means that boat is present and 0B means it is not there on the bank of river.
  Any State: $([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$, with constraints/conditions as $n_1 (\neq 0) \ge m_1; n_2 (\neq 0) \ge m_2; n_1 + n_2 = 3, m_1 + m_2 = 3$
  Goal State: ([0N, 0C, 0B], [3M, 3C, 1B])
  By no means, this representation is unique.
  In fact one may have number of different represenatations for the same problem.
  The table on the next side consists of production rules based on the chosen representation.

Set of Production Rules

Applied keeping constraints in mind

RN	Left side of rule		Right side of rule
Rules for boat going from left bank to right bank of the river			
L1	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([(n_1-2)M, m_1C, 0B], [(n_2+2)M, m_2C, 1B])$
L2	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([(n_1-1)M, (m_1-1)C, 0B], [(n_2+1)M, (m_2+1C), 1B])$
L3	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([n_1M, (m_1-2)C, 0B], [n_2M, (m_2+2C), 1B])$
L4	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([(n_1-1)M, m_1C, 0B], [(n_2+1)M, m_2C, 1B])$
L5	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([n_1M, (m_1-1)C, 0B], [n_2M, (m_2+1C), 1B])$
Rules for boat coming from right bank to left bank of the river			
R1	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([(n_1+2M)M, m_1C, 0B], [(n_2-2)M, m_2C, 0B])$
R2	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([(n_1+1M)M, (m_1+1)C, 1B], [(n_2+1)M, (m_2-1)C, 0B])$
R3	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([n_1M, (m_1+2)C, 1B], [n_2M, (m_2-2)C, 0B])$
R4	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([(n_1+1)M, m_1C, 0B], [(n_2-1)M, m_2C, 0B])$
R5	$([n_1M, m_1C, 1B], [n_2M, m_2C, 0B])$	$\rightarrow$	$([n_1M, (m_1+1)C, 0B], [n_2M, (m_2+1)C, 0B])$

One of the possible paths

Start $\rightarrow$	**([3M, 3C, 1B], [0M, 0C, 0B])**	
L2:	([2M, 2C, 0B], [1M, 1C, 1B])	1M, 1C $\rightarrow$
R4:	([3M, 2C, 1B], [0M, 1C, 0B])	1M $\leftarrow$
L3:	([3M, 2C, 1B], [0M, 3C, 1B])	2C $\rightarrow$

R4:	([3M,1C, 1B], [0M, 2C, 1B])	1C ←
L1:	([1M, 1C, 0B], [2M,2C, 1B])	2M →
R2:	([2M, 2C, 1B], [1M,1C, 0B])	1M,1C ←
L1:	([0M, 2C, 0B], [3M,1C, 1B])	2M →
R5:	([0M, 3C, 1B], [3M,0C, 0B])	1C ←
L3:	([0M, 1C, 0B], [3M,2C, 0B])	2C ←
R5:	([0M, 2C, 1B], [3M,1C, 0B])	1C ←
L3:	**([0M, 0C, 0B], [3M,3C, 1B])**	**2C → Goal state**

Example : The 8–Puzzle problem

Problem Statement

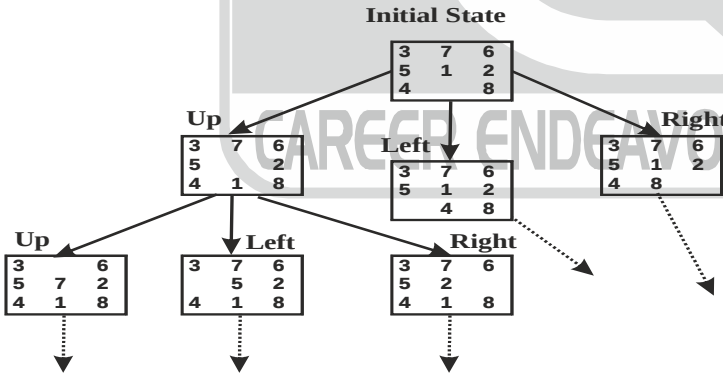
The eight puzzle problem consists of 3 x 3 grid with 8 consecutively numbered tiles arranged on it.
Any tile adjacent to the space can be moved on it.
Solving this problem involes arranging tiles in the goal state from the start state.

Start State Goal State

3	7	6	⟶	5	3	6		
5	1	2		7		2		
4		8		4	1	8		

Solution by State Space Method:

The start State could be represented as :
[[3,7,2], [5, 1, 2], [4, 0, 6]]
The goal State could be represented as :
[[5, 3, 6], [7, 0, 2], [4, 1, 8]]
The operators can be thought of moving {up, down, left, right}, the direction in which blank space effectively moves.
[[3,7,2], [5, 1, 2], [4, 0, 6]]



In order to provide a formal description of a problem, the following steps has been performed:

1. Define a state space that contains all the possible configurations of the relevant objects (and some impossible ones). It is, of course, possible to define this space without explicitly enumerating all of the states it contains.
2. Specify one or more states within that space that describes possible situations from which the problem solving process may start. These states are called the initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These states are called *goal states*.
4. Specify a set of rules that describe the actions (operators) available. Doing this will require giving thought to the following issues:

- What unstated assumptions are present in the informal problem description ?
- How general should the rules be ?
- How much of the work required to solve the problem should be precomputed and represented in the rules ?

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

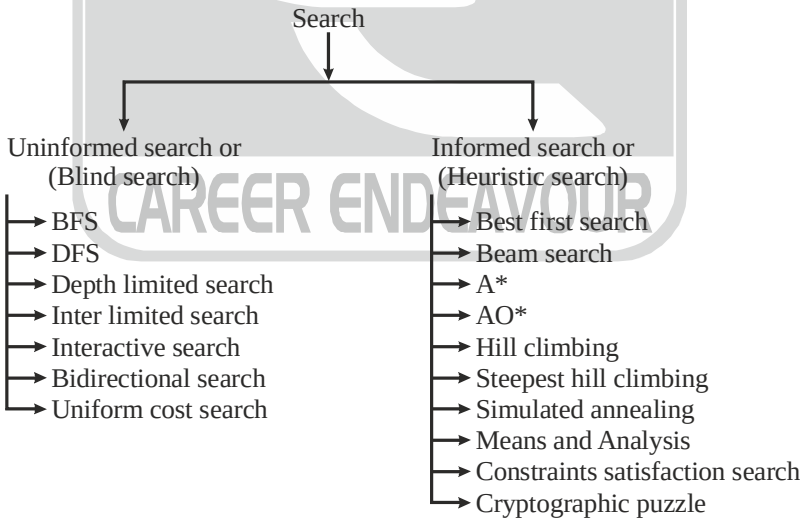
Production system : It is useful to structure A.I. programs in a way that facilitates describing and performing the search process. A production system consists of :

- A set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied (For forward rules).
- One or more knowledge/databases that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.
- A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
- A rule applier.

It also encompasses a family of general production system interpreters, including:

- Basic production system languages, such as OPS5 and ACT*.
- More complex, often hybrid systems called expert system shells, which provide complete (relatively speaking) environments for the contruction of knowledge based expert systems.
- General problem solving architectures like SOAR [Laired *et al*., 1987], a system based on a specific set of cognitively motivated hypotheses about the nature of problem solving.

Problem solving by search:



Problem solving by search:

The problem (search space) can defined as four components (S, s, O, G)

Where, $S \rightarrow$ set of states represented as node

$s \in S \rightarrow$ initial/starting states from where the agent initiates its search process.

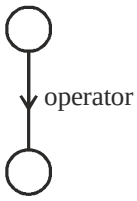
$O \rightarrow$ the operator/action that is used to move the agent from one states to next state (successer state)

$O : S \rightarrow S \qquad O(P) = q$

where, P is a state on which operator action is applied.

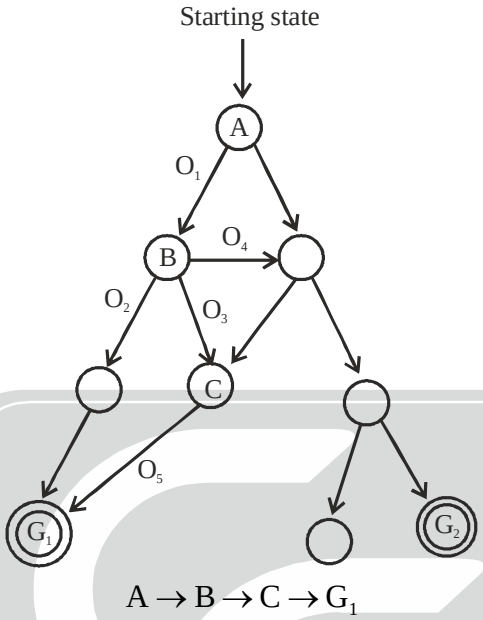
O is a function where takes S (i.e. state) to the initial node.

The operators are represented by an edge in the state space graph.



$G \subseteq S$: The set of goal states|derived states.

Agent is an object who searches the goal node.



Plan: The sequence of action i.e. (O_1, O_3, O_5)

Solution plan: The sequence of action which leads you from initial to goal state i.e. (O_1, O_3, O_5).

Cost of plan: Sum of cost of all the operators in the plan.

$$\text{Cost}(O_1, O_3, O_5) = \text{cost}(O_1) + \text{cost}(O_3) + \text{cost}(O_5)$$

General Frame Work for search:

(1) Initialize : OPEN = {&} CLOSED = {}

(2) Fail : If OPEN is empty terminate with failure

(3) Select : Pick a state m from OPEN and place to the CLOSED.

(4) Terminate : Terminate with success.

(5) Expand : Find the successor n of m and if $n \notin$ OPEN U CLOSED then place n into OPEN list.

(6) Loop: goto to step 2.

BLIND SEARCH/UNINFORMED SEARCH

When there is no information available regarding the search space, such type of search is known as blind/ uninformed search.

1. Breadth First Search (BFS):

We can implement it by using two lists called OPEN and CLOSED.

The OPEN list contains those states that are to be expanded and CLOSED list keeps track of states already expanded.

Here OPEN list is used as a **queue.**

Algorithm (BFS):

Input : Two states, START and GOAL

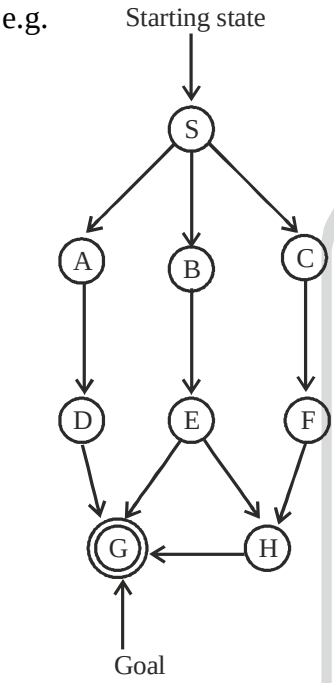
Local Variables: OPEN, CLOSED, STATE–X, SUCCESSORS

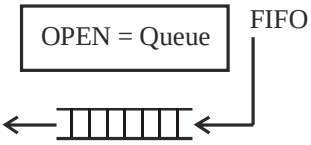
Output : Yes or NO

Method :

Initially OPEN list contains a START node and CLOSED list is empty, Found = false;
While (OPEN $\neq$ empty and found = false)
Do {
Remove the first state from OPEN and call it STATE-X;
Put STATE-X in the front of CLOSED list;
If STATE-X=GOAL then **Found = true** else
{-perform EXPAND operation on STATE-X, producing a list of SUCCESSORS;
- Remove from SUCCESSORS those staes, if any, that are in the CLOSED list;
- Append SUCCESSORS at the end of the OPEN list/*queue*/
} } /* end while */
If found = true then return **YES** else return **No** and Stop

e.g.





OPEN ← S Closed { }

OPEN ← A B C Closed {S}

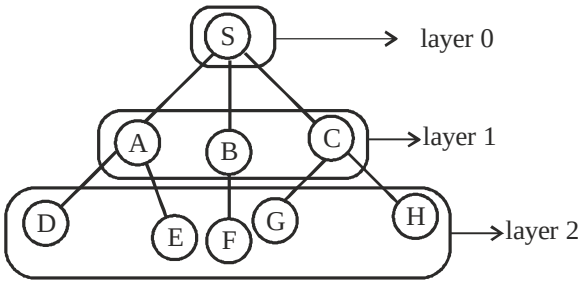
OPEN ← B C D Closed {S,A}

OPEN ← C D E Closed {S, A, B}

OPEN ← D E F Closed {S, A, B, C}

OPEN ← E F G H Closed {S, A, B, C, D, E, F, G}

It is BFS (Breadth First Search). Graph in a layer wise manner

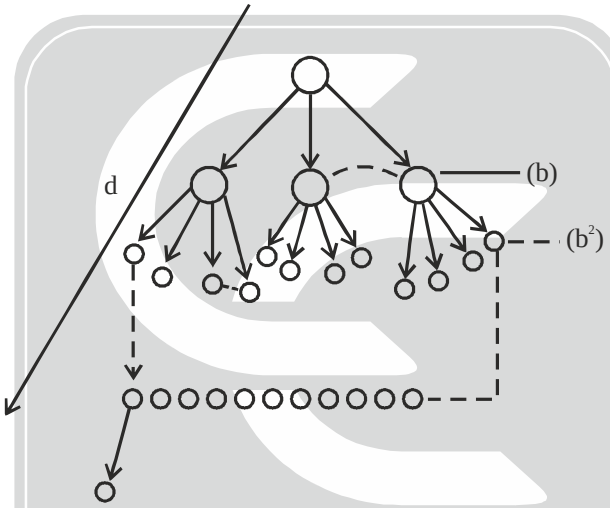


If the branching factor = b and depth of the goal = d

Branching factor (b) is a max number of child

Time complexity = # of node accessed during the traversal = $O(b^d)$

Space complexity: Max size of the OPEN list.



Time complexcity:

$$1 + b + b^2 + ... + b^{d-1} + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

Space complexcity:

$O(b^d)$

It increases exponentially

2. Depth First Search

- In depth-first search we go as far down as possible into the search tree/graph before backing up and trying alternatives.
- It works by generating a descendent of the most recently expanded node until some depth cut off is reached and then backtracks to next most recently expanded node and generates one of its descendants.
- It avoids memory limitation as it only stores a singles path from the root to leaf node along with the remaining unexpndable siblings for each node on the path.
- Again use two lists called OPEN and CLOSED with the same conventions explained earlier.
- Here OPEN list is used as a **stack.**
- If we discover that first element of OPEN is the **Goal** state, then search terminates successfully.

- We could get track of our path through state space as we traversed but in those situations where many nodes after expansion are in the closed list, then we fail to keep track of our path.
- This information can be obtained by modifying CLOSED list by putting pointer back to its parent in the search tree.

Algorithm (DFS)

Input : Two states, START and GOAL

Local Variables: OPEN, CLOSED, RECORD–X, SUCCESSORS

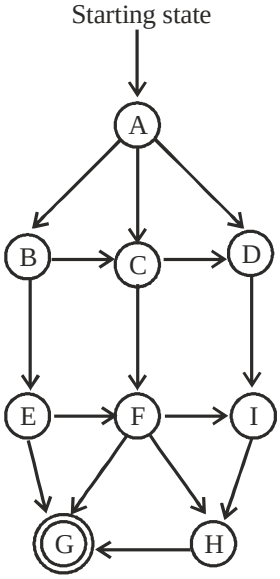
Output: A path sequence, if one exists, otherwise return No.

Method: OPEN, CLOSED, RECORD–X, SUCCESSORS

Form a stack consisting of (START, nil) and call it OPEN list.
Initially set CLOSED list as empty;
Found = false;
While (OPEN $\neq$ empty and found = false) DO
{
Remove the first state from OPEN and call it RECORD-X
Put RECORD-X in the front of CLOSED list;
If the state variable of RECORD-X=GOAL,
then **Found- true**
Else
{
Perform EXPAND opertation on STATE-X, a state variable of RECORD-X, producing a list of action records called SUCCESSORS;
create each action record by associating with each state its parent.
Remove from SUCCESSORS any record whose state variables are in the record already in the CLOSED list.
Insert SUCCESSORS in the front of the OPEN list /*Stack*/
}
} /*end while*/
If Found = true
then return the plan used /* find it by tracing through the pointers on the CLOSED list */else retum **No**
Stop

DFS:

OPEN = STACK



Here, we can also follow the path from A→B→C→D→I..... then the cost may be difference it we follow difference path.

OPEN	A →	Closed { }
OPEN	D C B →	Closed {A}
OPEN	D C E →	Closed {A, B}
OPEN	D C F G →	Closed {A, B, E}
OPEN	D C F →	Closed {A, B, E, G}

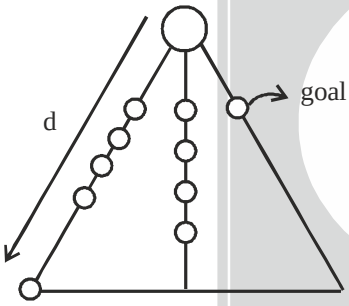
G is the goal node

This is the depth first search (DFS).

In DFS we go in on direction deeper untill we hit to a dead end. After hitting to the dead end we need to back track.

If the branching factor is b and depth of the tree is m

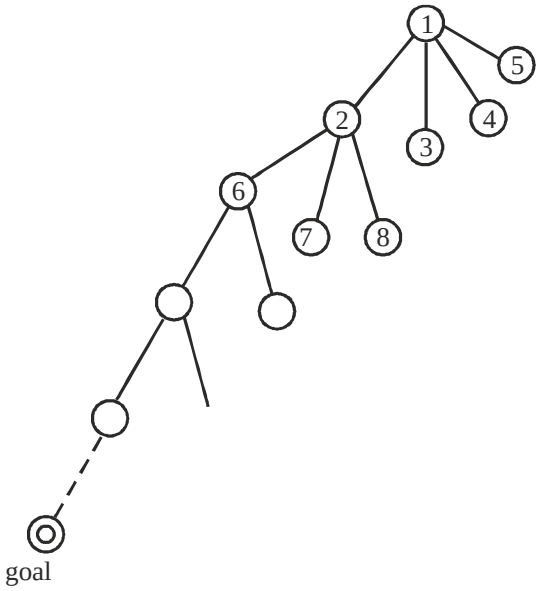
Time complexity: $O(b^m)$

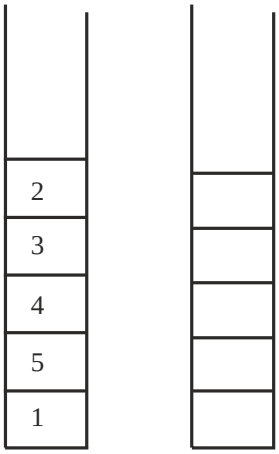


Space complexity : Maximum number of nodes in the open list at any moment.

$\Rightarrow$ b + b + b + ... m times.

$\boxed{O(bm)}$





Comparisons : (These are unguided/blind searches, we can not say much about them)

- DFS is effective when there are few sub trees in the search tree that have only one connection point to rest of the states.
- DFS is best when the GOAL exist in the lower portion of the search tree
- DFS can be dangerous when the path closer to the START and further from the GOAL has been chosen.
- DFS may no find solution, even if it exists
- DFS may go to infinite loop if rules applied are left recursive.
- BFS is effective when the search tree has a low branching factor.
- BFS required a lot memory as number of nodes in level of the tree increases exponentially as compared to DFS.
- BFS is superior when the GOAL exists in the upper right portion of a search tree.

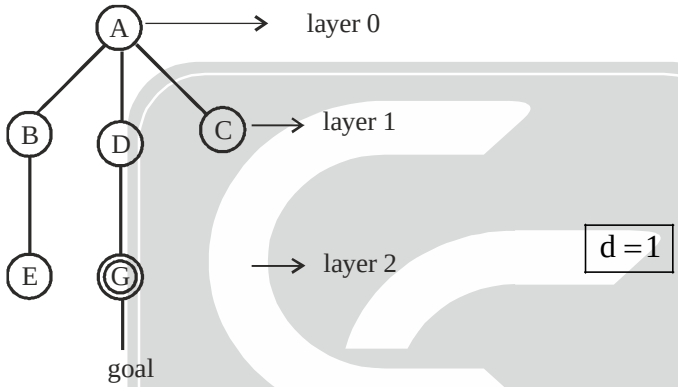
3. Depth Limited Search:

Apply the search only to the specified depth = d. It may follow BFS and DFS.

Successful: If the goal is within the depth

Unsuccessful: If the goal is at depth > d.

DFS :



It can only search at the level 1 because depth (d) = 1. It goes $A \rightarrow B$ then backtrack A and then $A \rightarrow D$ then $A \rightarrow C$.

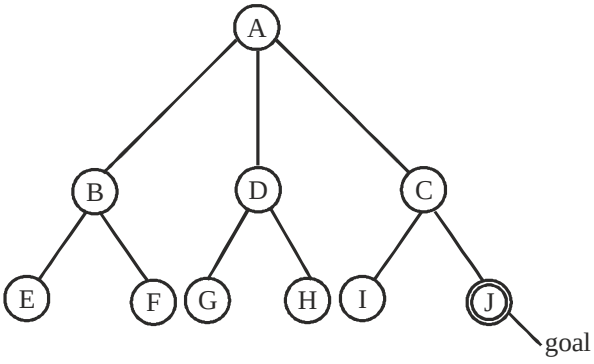
Problem with this search is that if the goal is not with in specified search space then it does not find the goal.

Solution for this search is that we increase the depth (d) by 1 i.e. search is called iterative deepening search.

Iterative Deepening Search: It is the extension of depth limited search.

We increase the depth in each iteration if the goal is not achieved.

DFS follows:



$d = 0 \Rightarrow A \rightarrow$ unsuccessful

$d=1 \Rightarrow A,B,D,C \rightarrow$ unsuccessful

$d = 2 \Rightarrow A, B, E, F, D, G, H, C, I(J) \rightarrow$ unsuccessful

Here, we visit the root node d + 1 times and next level is d times.

If the branching factor = b and goal is at depth d.

Time complexity = $1.b^d + 2.b^{d-1} + 3.b^{d-2} ..... + (d+1) \times 1$

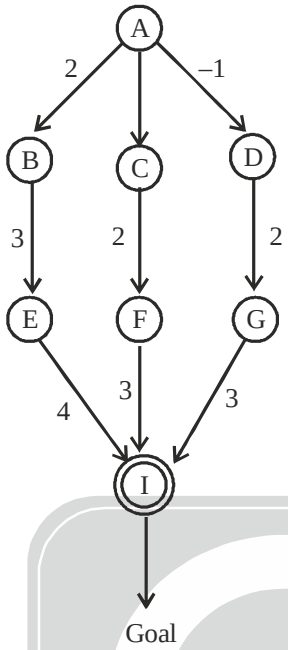
$$\boxed{\text{Time compl.} = O(b^d)}$$

Solution Plan:

(1) $A \xrightarrow{2} B \xrightarrow{3} E \xrightarrow{4} I = 9$

(2) $A \xrightarrow{1} C \xrightarrow{2} F \xrightarrow{3} I = 6$

(3) $A \xrightarrow{-1} D \xrightarrow{2} G \xrightarrow{3} I = 4$



Path with cheapest Cost is the shorted path.

4. Uniform Cost search: It uses only historical value.

(1) Initiative: OPEN = { } CLOSED = { } C[&] = 0

(2) Fail: If open is empty the terminate the search with failure.

(3) Select : Select a state n with minimum C value and put it into closed.

(4) Terminate : If n is a goal then terminate with success.

(5) Expand : For all the successor of n for a succesor m of n if $m \notin$ OPEN $\cup$ CLOSED C[m] = C[n] + w[n,m] and put n inito open.

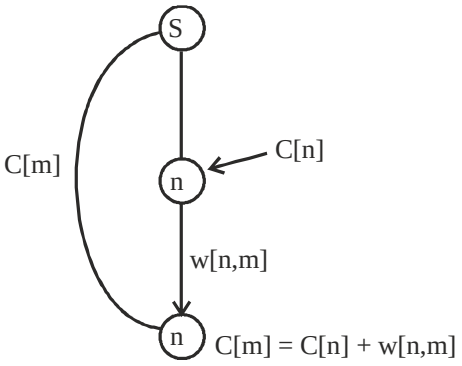
If $m \in$ OPEN $\cup$ CLOSED then set

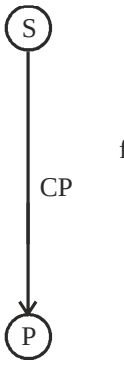
$$C[m] \leftarrow \min\{C[m], C[n] + w[n,m]\}$$

If C[m] has declared and $m \in$ closed then remove m from closed and put it into the OPEN.

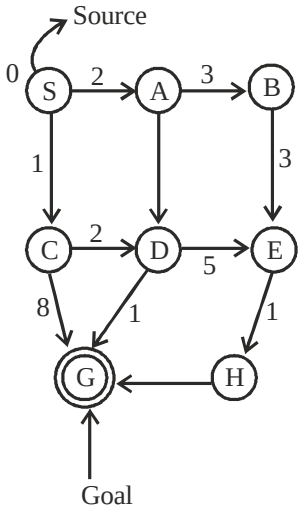
Note : Uniform cost search is the best algorithm for a search problem. It can solved in any general graph for optimal cost.

Loop goto step # 2:





C(P) : the cost of the vertices from source vertex path to vertices P



$\text{Open} = \{S^0\}, \quad \text{closed} = \{\ \}$

$\text{Open} = \{A^2, C^1\}, \quad \text{closed} = \{S^0\}$

$\text{Open} = \{A^2, D^3, G^9\} \quad \text{closed} = \{S^0, C^1\}$

Out the node which has min cost value i.e.

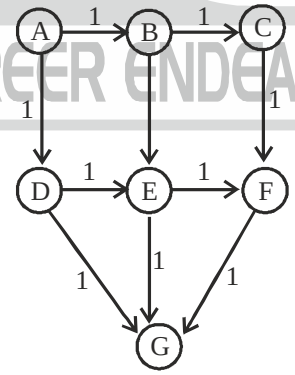
$\text{Open} = \{D^3, G^9, B^5\}, \quad \text{closed} = \{S^0, C^1, A^2\}$

$\text{Open} = \{G^4, B^5, E^8\}, \quad \text{closed} = \{S^0, C^1, A^2, D^3\}$

$\text{Open} = \{B^5, E^4\}, \quad \text{closed} = \{S^0, C^1, A^2, D^3, G\}$

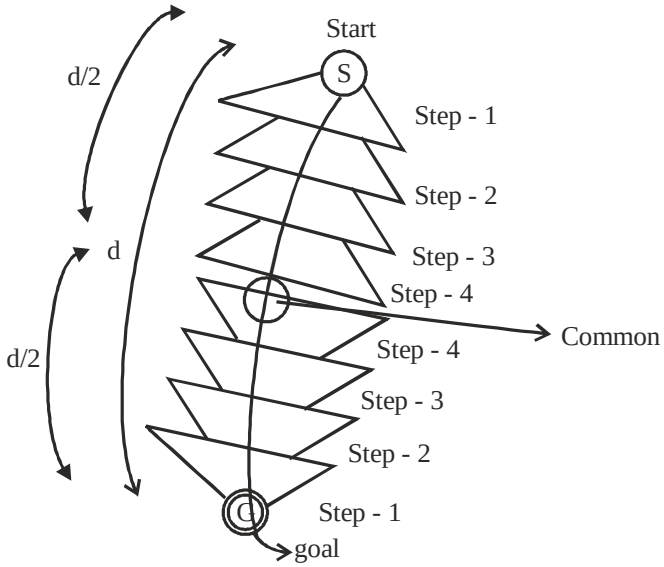
G^4 = cost of the path = 4

Note: If all the cost of edges in a graph are equal then UNIFORM cost search behaves like BFS or also in case if cost are absent. It uses priority queue.

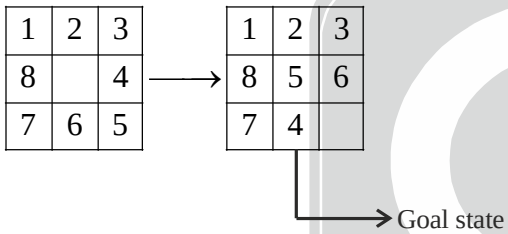


Bi-Directional Search

- For those problems having a single goal state and single start state, bi-directional search can be used.
- It starts searching forward from initial state and backward from the goal state simultaneously starting the states generated until a common state is found on both search frontiers.
- DFID can be applied to bi-direction search for k = 1, 2, ... as follows :
- K^{th} iteration consists of a DFS from direction to depth k storing all states at depth k, and DFS from other direction : one to depth k and other to depth k+1 not storing from forward direction. /*The search to depth k+1 is necessary to find odd-length solutions */.

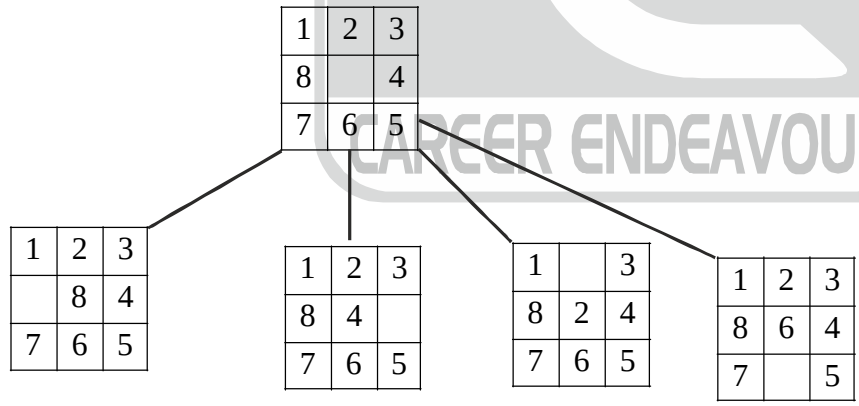


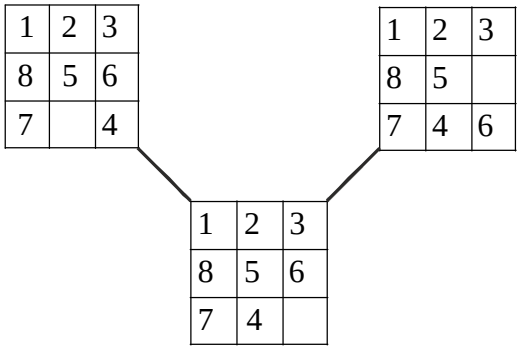
It is useful for solving 8 Puzzle.



Operator :

(1) Left slide (2) Right slide (3)Up (4) Down



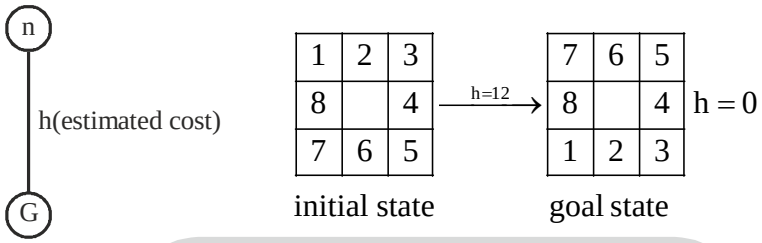


INFORMED/HEURISTIC SEARCH

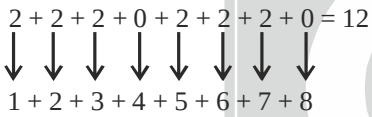
A heuristic function is a function that maps from problem state description to measures of desirability, usually represented as numbers.

Well defined heuristic functions can play an important role in efficiently guiding a search process toward a solution. The purpose of heuristic function is to guide the search process in the most profitable direction by suggesting which part to follow first when more than one is available.
It uses the domain specific information to estimate the quality or the potential of the partial solution.
Search specific means estimated cost from current node to goal node.
Every node is assocaited with a Heuristic value.

Heuristic Value: The estimated cost of moving from the current node to the goal node.

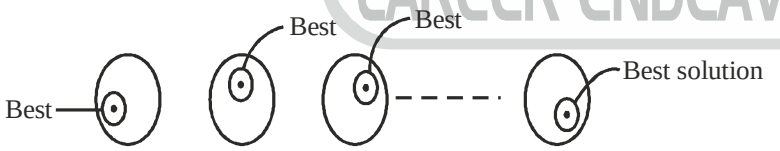


(1) h = # of digits or tiles which are out of place = 6
(2) Manhatton distance:

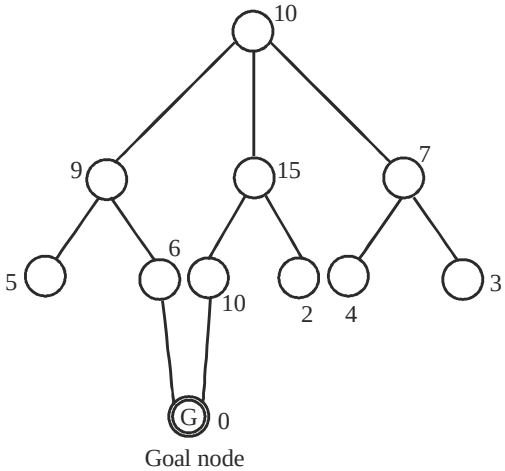


Two types of heuristic:
(1) Accurate: Estimated cost is right
(2) Inaccurate: Estimated cost is not correct.
Both DFS and BFS blindly explore paths without considering any cost function.

1. Best first search: In this, next successor is choosen as a node having Best Heuristic value.
It considers only the future (Greedy approach)
The idea of best first search is to use an evaluation function to decide which adjacent is most promising and then explore.



In the greedy aproach we consider locally best solution.

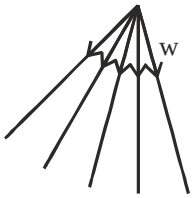


Algorithm: Best-First Search:

1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes left as OPEN do:
   - Pick the best node on OPEN
   - Generate its successors.
   - For each succesor do:
     - If it has not been generated before, evaluate it, add to OPEN, and record its parent.
     - If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

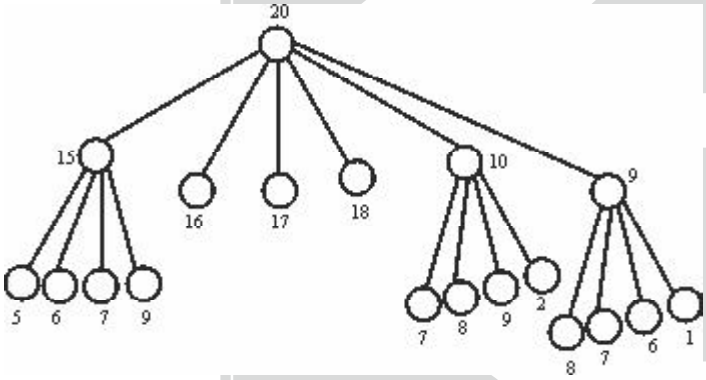
Note: Best-First Search is Hill Climbing where we compare Heuristic value not only among same level node but also with lower level nodes.

2. Beam Search: Here each possibility is checked for the desired solution. It keeps track of k states rather than just 1 state.



width of the beam = 5

OPEN : It is used for storing traversing element (frontier/fringe).



1	2	6
3		7
5	8	4

h=10 →

1	8	7
2		6
3	4	6

8 puzzle problem goal state

0 + 2 + 1 + 1 + 2 + 1 + 1 + 2 = 10

↓Left

1	2	6
	3	7
5	8	4

h = 11

↓Right

1	2	6
3	7	
5	8	4

h = 11

↓Up

1		6
3	2	7
5	8	4

h = 9

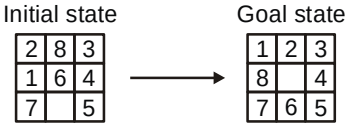
↓Down

1	2	6
3	8	7
5		4

h = 9

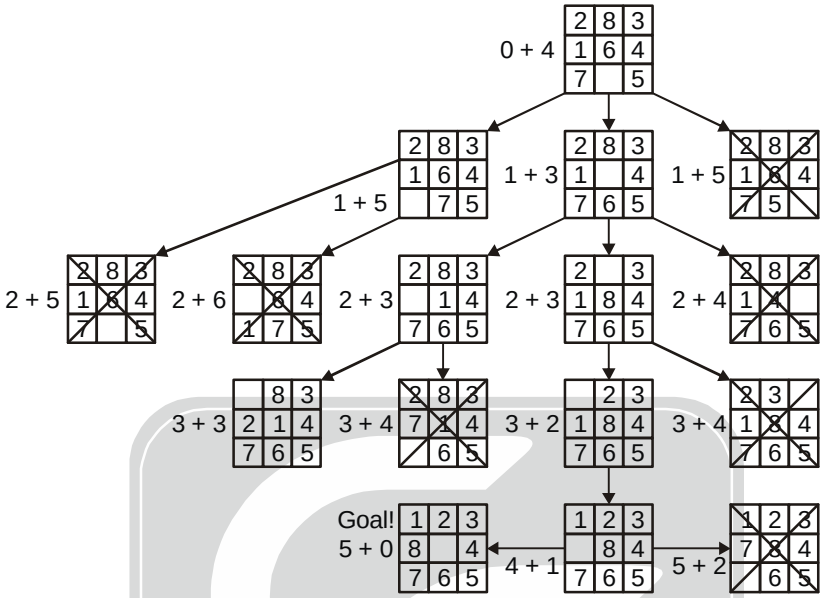
SOLVED EXAMPLES

1. We have Beam search with (limit = 2) where



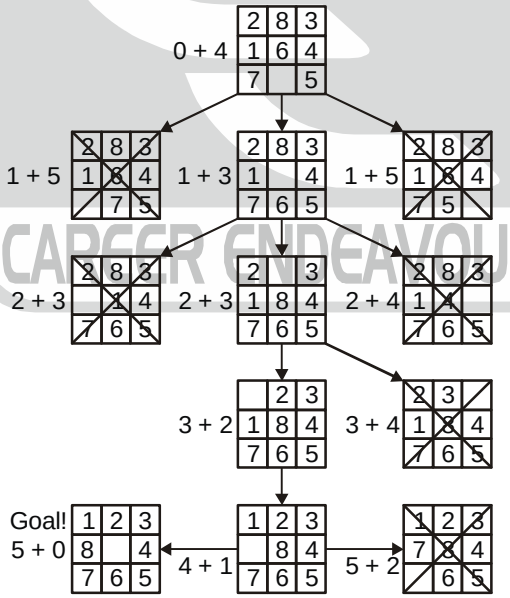
what will be the cost to reach from initial state to goal state ?

Soln.

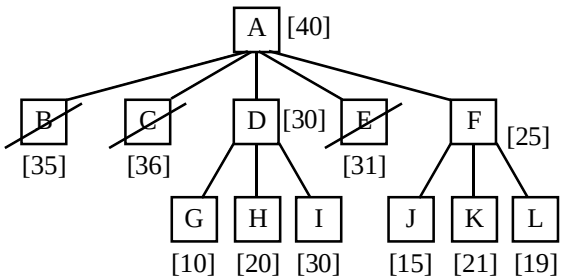


2. Example of Beam search algorithm when beam (k = 1)

Soln.



3. Which nodes will expands after D and F using beam search (Width = 2) algorithm ?

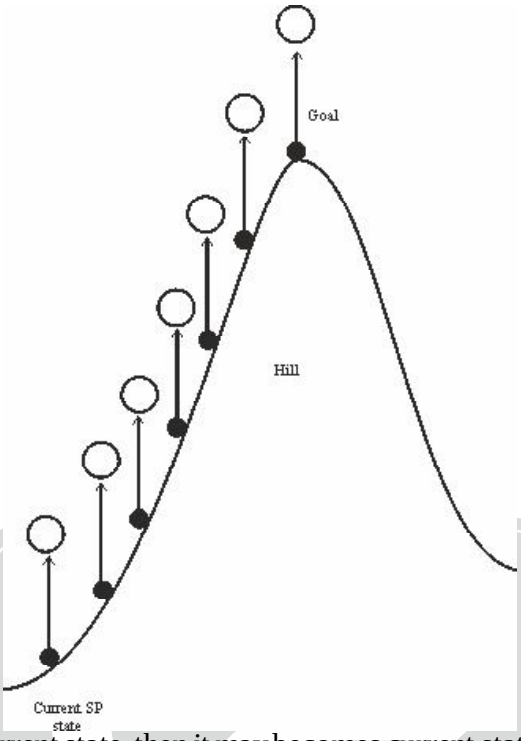


(a) HG (b) JK (c) GJ (d) HJ

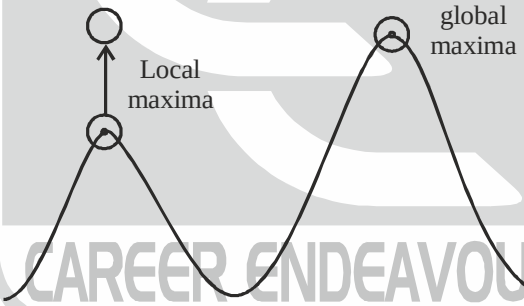
Soln. Since G, J are the cheapest nodes. Therefore, for the next level these nodes will expands.

Correct option is (c)

3. Hill Climbing:



If successor is better than current state then it may becomes current state. This is perform till we reach the goal.



Hill Climbing
Algorithm

Function Hill-Climbing(problem)
Returns a local maximum
Inputs: problem
Local variables: current (a node)
neighbour (a node)
Current <- make-node(initial-state[problem])
Loop do
Neighbour <- highest valued successor of current
If value(neighbor) <- value(current) return state(current)
current <- neighbour
End.

Note: So Hill Climbing is Heuristic DFS.