1. Describe different types of environments applicable to AI agents

An environment in artificial intelligence is the surrounding of the agent. The agent takes input from the environment through sensors and delivers the output to the environment through actuators. There are several types of environments:

- Fully Observable vs Partially Observable
- Deterministic vs Stochastic
- Competitive vs Collaborative
- Single-agent vs Multi-agent
- Static vs Dynamic
- Discrete vs Continuous

1. Fully Observable vs Partially Observable

- When an agent sensor is capable to sense or access the complete state of an agent at each point of time, it is said to be a fully observable environment else it is partially observable .
- Maintaining a fully observable environment is easy as there is no need to keep track of the history of the surrounding.
- An environment is called **unobservable** when the agent has no sensors in all environments.
- **Example:**
    - **Chess –** the board is fully observable, so are the opponent's moves
    - **Driving –** the environment is partially observable because what's around the corner is not known

2. Deterministic vs Stochastic

- When an uniqueness in the agent's current state completely determines the next state of the agent, the environment is said to be deterministic.
- Stochastic environment is random in nature which is not unique and cannot be completely determined by the agent.
- **Example:**
  Chess – there would be only few possible moves for a coin at the current state and these moves can be determined
  Self Driving Cars – the actions of a self driving car are not unique, it varies time to time

3. Competitive vs Collaborative

- An agent is said to be in a competitive environment when it competes against another agent to optimize the output.
- Game of chess is competitive as the agents compete with each other to win the game which is the output.
- An agent is said to be in a collaborative environment when multiple agents cooperate to produce the desired output.
- When multiple self-driving cars are found on the roads, they cooperate with each other to avoid collisions and reach their destination which is the output desired.

4. Single-agent vs Multi-agent

- An environment consisting of only one agent is said to be a single agent environment.
- A person left alone in a maze is an example of single agent system.
- An environment involving more than one agent is a multi agent environment.
- The game of football is multi agent as it involves 10 players in each team.

5. Dynamic vs Static

- An environment that keeps constantly changing itself when the agent is up with some action is said to be dynamic.
- A roller coaster ride is dynamic as it is set in motion and the environment keeps changing every instant.
- An idle environment with no change in it's state is called a static environment.
- An empty house is static as there's no change in the surroundings when an agent enters.

6. Discrete vs Continuous

- If an environment consists of a finite number of actions that can be deliberated in the environment to obtain the output, it is said to be a discrete environment.
- The game of chess is discrete as it has only a finite number of moves. The number of moves might vary with every game, but still, it's finite.
- The environment in which the actions performed cannot be numbered ie. is not discrete, is said to be continuous.
- Self-driving cars are an example of continuous environments as their actions are driving, parking, etc. which cannot be numbered.

2. Define blind search and informed search. Hence discuss the merits and demerits of each.

Informed Search: Informed Search algorithms have information on the goal state which helps in more efficient searching. This information is obtained by a function that estimates how close a state is to the goal state.
Example: Greedy Search and Graph Search

Uninformed Search or blind search : Uninformed search algorithms have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and length of actions.
Examples: Depth First Search and Breadth-First Search

Informed Search vs. Uninformed Search:

Informed Search	Uninformed Search
It uses knowledge for the searching process.	It doesn't use knowledge for searching process.
It finds solution more quickly.	It finds solution slow as compared to informed search.
It is highly efficient.	It is mandatory efficient.
Cost is low.	Cost is high.
It consumes less time.	It consumes moderate time.
It provides the direction regarding the solution.	No suggestion is given regarding the solution in it.

Informed Search	Uninformed Search
It is less lengthy while implementation.	It is more lengthy while implementation.
Greedy Search, A* Search, Graph Search	Depth First Search, Breadth First Search

3. Compose your opinion about heuristic function

A **heuristic function**, also called simply a **heuristic**, is a **function** that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution.

The heuristic function is a way to inform the search about the direction to a goal.

It provides an informed way to guess which neighbor of a node will lead to a goal. It must use only information that can be readily obtained about a node

The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand.

This solution may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.

The representation may be approximate cost of the path from the goal node or number of hopes required to reach to the goal.

The heuristic function that we are considering, for a node n is, h (n) =estimated cost of the cheapest path from the state at node n to a goal.

Example: For Travelling Salesman Problem, the sum of the distances traveled so far can be a simple heuristic function.

It is of two types: Maximize or Minimize function.

In maximization, greater the cost of node, better is node while in minimization, lower the cost better is the node.

4. Explain any two Informed Search Strategies

5) Define Artificial Intelligence and list the task domains of Artificial Intelligence.

Artificial intelligence (or AI) is basically the capability of a machine or a computer program to think and learn like a human, in short, mimicking human intelligence processes. Algorithms are created and applied into a dynamic computing environment, enabling machines to learn and make decisions with the data.

As there are many different types of AI, it can be classified into 3 main categories.

Narrow Artificial Intelligence

Also known as Weak AI, it is developed and designated for a particular task. The single task it performs is within a limited context, being excellent at routine jobs, be it physical or cognitive.

Artificial General Intelligence

Also known as AGI, it has generalized cognitive ability to analyse and decide, in which the AI provides solutions for an unfamiliar task it encounters. It can applies understanding and reasoning like how a normal human would in the specific environment.

Artificial Super Intelligence

Also known as ASI, it exceed human abilities, being be able to mimic human thoughts and achieve an accuracy higher than normal humans. ASI proves superior to humans in terms cognitive ability.

Now, AI is dependent on the tasks they carry out. It can range from simple task, to complex analytics. There are a few domains on how AI carry out their functions.

Formal Tasks- Tasks which requires logic and constraints to function.

(For eg. Mathematics, Games, verification)

Mundane tasks- Routine everyday tasks that requires common sense reasoning

(For eg. Perception, robotics, natural language, Vision, Speech)

Expert tasks- Tasks which requires high analytical and thinking skills, a job only a professionals can do

(For eg. Financial analysis, medical diagnostics, engineering, scientific analysis, consulting)

So having these types of domains, there are different types of works and information AI can interpret. Usually an AI can approach them in the following ways.

Symbolic approach is how an AI application interprets human intelligence through symbol manipulation and interpretation. Basic concepts are broken down into symbols and the AI processes them into something understandable to solve the problem. These symbols are arranged in such a way that the structures pertaining them can relate to each other. It can be through lists, hierarchies, or even networks to determine it.

Sub-symbolic approach is how an AI recognizes things or patterns without any specific knowledge. Using a connectionist system method, the AI processes a cluster of high-dimensional quantitative sensory data to determine the relevancy of each node of data and determine the output of it. It functions on limited information, gradually adjusting to finally to best fit a solution.

Statistical approach is how AI solve specific problems using mathematics as tools. The AI will gather, organize, analyze, synthesize and interpret numerical information from data and deliver measurable forumlative results. This helps retrieve important data and trends which are vital and necessary.

6) State and explain algorithm for the Best First search with an example

Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic

function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

1. f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

1)Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

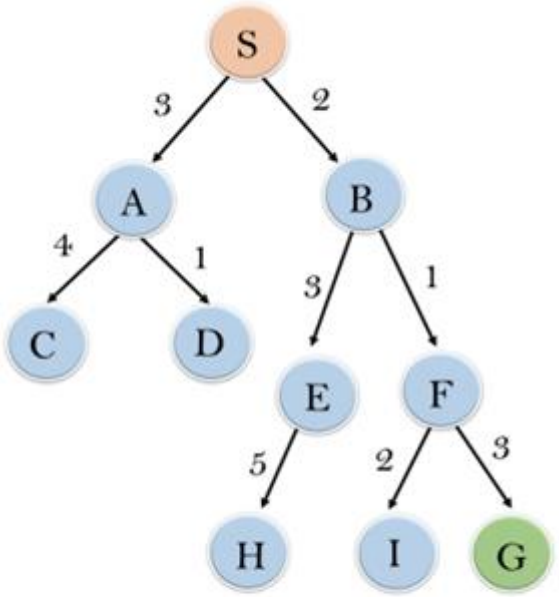
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

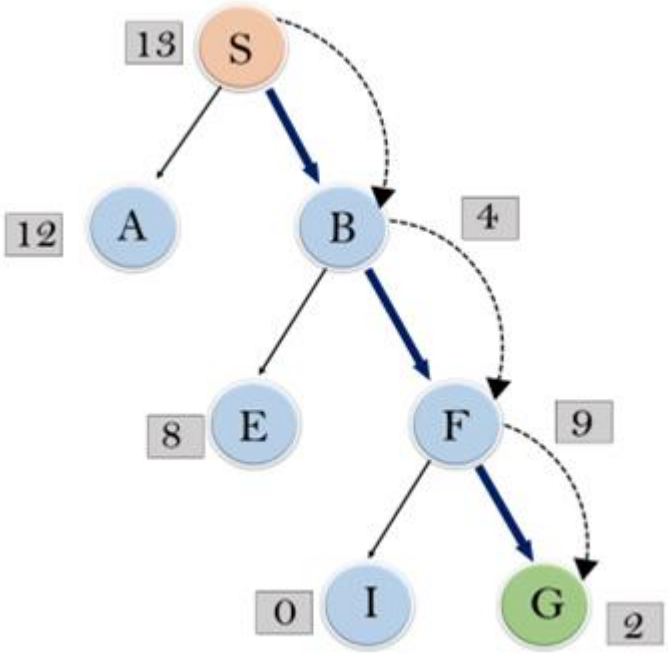
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

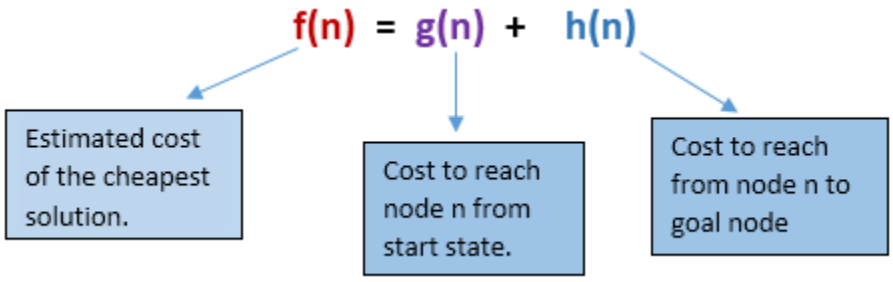
Optimal: Greedy best first search algorithm is not optimal.

7) Explain A* algorithm and write its pseudo code.

A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

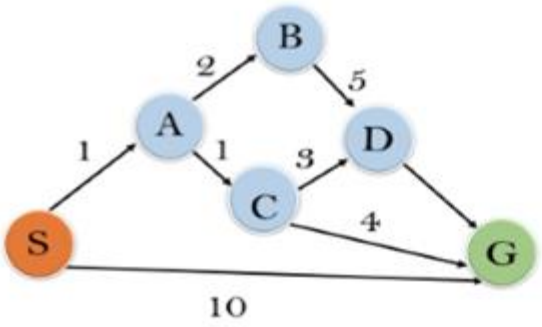
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

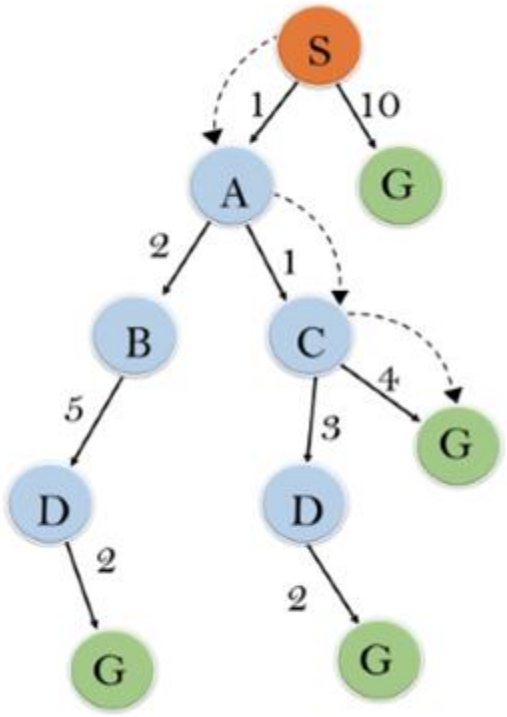
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

8) Explain hill climbing algorithm. Explain plateau, ridge, local maxima and global maxima.

hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.
Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, **mathematical optimization problems** implies that hill-climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-Travelling salesman problem where we need to minimize the distance traveled by the salesman.
- 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in **reasonable time.**
- A **heuristic function** is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

Features of Hill Climbing

1. **Variant of generate and test algorithm :** It is a variant of generate and test algorithm. The generate and test algorithm is as follows :

   1. Generate possible solutions.
   2. Test to see if this is the expected solution.
   3. If the solution has been found quit else go to step 1.

   Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from the test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

2. **Uses the** Greedy approach **:** At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.

Types of Hill Climbing

1. **Simple Hill climbing :** It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node.
   Algorithm for Simple Hill climbing :
   Step 1 : *Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.*
   Step 2 : *Loop until the solution state is found or there are no new operators present which can be applied to the current state.*
   a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

   b) Perform these to evaluate new state
   i. If the current state is a goal state, then stop and return success.
   ii. If it is better than the current state, then make it current state and proceed further.
   iii. If it is not better than the current state, then continue in the loop until a solution is found.

   Step 3 : *Exit.*

2. **Steepest-Ascent Hill climbing:** It first examines all the neighboring nodes and then selects the node closest to the solution state as of next node.

   Step 1 : *Evaluate the initial state. If it is goal state then exit else make the current state as initial state*
   Step 2 : *Repeat these steps until a solution is found or current state does not change*
   i. Let 'target' be a state such that any successor of the current state will be better than it;

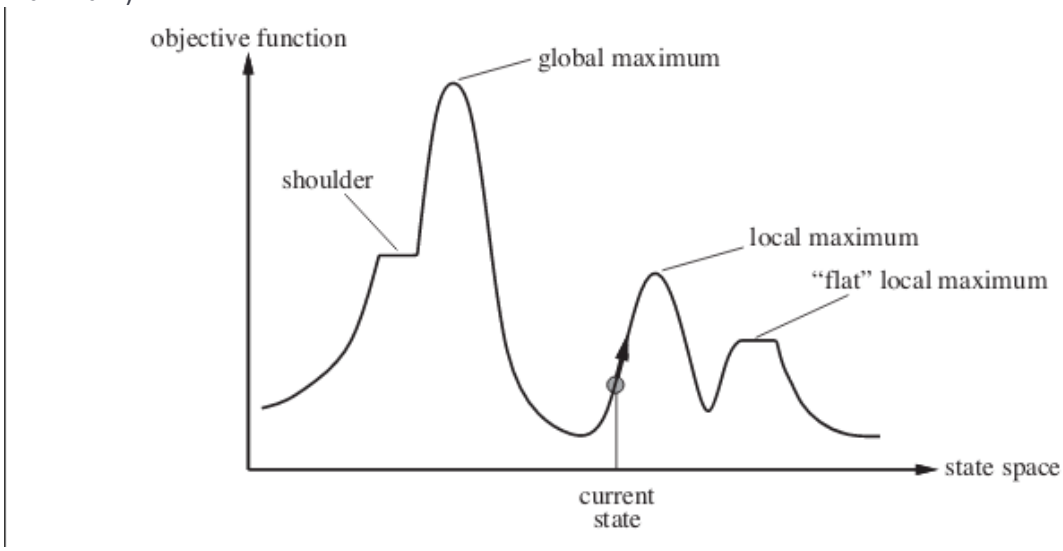
   ii. for each operator that applies to the current state
   a. apply the new operator and create a new state
   b. evaluate the new state
   c. if this state is goal state then quit else compare with 'target'
   d. if this state is better than 'target', set this state as 'target'
   e. if target is better than current state set current state to Target

Step 3 : *Exit*

3. **Stochastic hill climbing :** It does not examine all the neighboring nodes before deciding which node to select .It just selects a neighboring node at random and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

State Space diagram for Hill Climbing

State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximize).
X-axis : denotes the state space ie states or configuration our algorithm may reach.
Y-axis : denotes the values of objective function corresponding to a particular state.
The best solution will be that state space where objective function has maximum value(global maximum).



Different regions in the State Space Diagram

1. **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.
2. **Global maximum :** It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
3. **Plateua/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
4. **Ridge :** It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
5. **Current state :** The region of state space diagram where we are currently present during the search.
6. **Shoulder :** It is a plateau that has an uphill edge.

Problems in different regions in Hill climbing

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

1. **Local maximum :** At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
   To overcome local maximum problem : Utilize backtracking technique. Maintain a list

of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

2. **Plateau :** On plateau all neighbors have same value . Hence, it is not possible to select the best direction.
   To overcome plateaus : Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region
3. **Ridge :** Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.
   To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

9) Explain simulated annealing.

Simulated Annealing (SA) is an effective and general form of optimization. It is useful in finding global optima in the presence of large numbers of local optima. "Annealing" refers to an analogy with thermodynamics, specifically with the way that metals cool and anneal. Simulated annealing uses the objective function of an optimization problem instead of the energy of a material.

Implementation of SA is surprisingly simple. The algorithm is basically hill-climbing except instead of picking the best move, it picks a random move. If the selected move improves the solution, then it is always accepted. Otherwise, the algorithm makes the move anyway *with some probability* less than 1. The probability decreases exponentially with the "badness" of the move, which is the amount deltaE by which the solution is worsened (i.e., energy is increased.)

Prob(accepting uphill move) ~ 1 - exp(deltaE / kT))

A parameter T is also used to determine this probability. It is analogous to temperature in an annealing system. At higher values of T, uphill moves are more likely to occur. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization, T starts high and is gradually decreased according to an "annealing schedule". The parameter k is some constant that relates temperature to energy (in nature it is Boltzmann's constant.)

Simulated annealing is an algorithm that combines hill-climbing with a random walk in someway that yields both efficiency and completeness.
Choosing based on condition.
Here try to minimize value of objective function

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency.

The schedule input determines the value of the temperature T as a function of time.

- At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

- When T is decreased slowly enough it always reaches the best state x^* because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T.
(Is this necessarily an interesting guarantee?)

Delta E = +VE moving up, right path ,current state is good state

The probability that a transaction to a higher energy state will occur and so given by a function:

$$P = e^{-\Delta E/kT}$$

- E is the +ve level in the energy level
- T is the temperature
- k is Boltzmann's constant

To explain simulated annealing, we switch our point of view from hill climbing to **gradient descent** (i.e., minimizing cost) and imagine the task of getting a ping-pong ball into the deepest crevice in a bumpy surface. If we just let the ball roll, it will come to rest at a local minimum. If we shake the surface, we can bounce the ball out of the local minimum. The trick is to shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from the global minimum. The simulated-annealing solution is to start by shaking hard (i.e., at a high temperature) and then gradually reduce the intensity of the shaking (i.e., lower the temperature).

Simulated annealing

function SIMULATED ANNEALING (*problem*, *schedule*)
returns a solution state

inputs:
problem, a problem
schedule, a mapping from time to "temperature"
local variables:
current, a node
next, a node

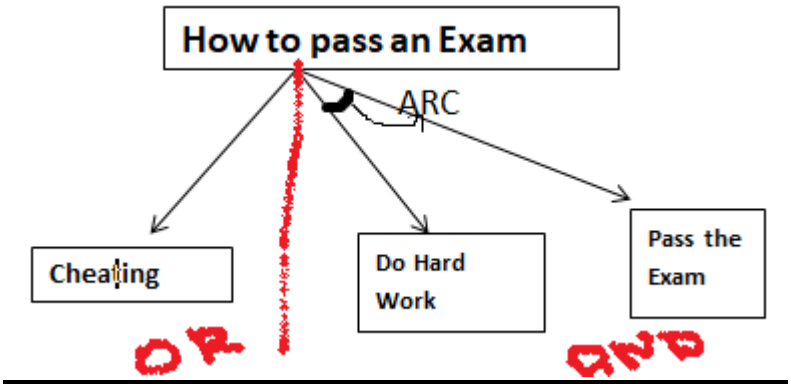
current ← MAKE-NODE(*problem*.INITIAL-STATE)
for $t = 1$ **to** ∞ **do**
$T \leftarrow$ *schedule*(*t*)
if *T=0* **then return** *current*
next ← a randomly selected successor of *current*
$\Delta E \leftarrow$ *next*.VALUE - *current*.VALUE
if $\Delta E > 0$ **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E/T}$

10. Explain problem reduction with respect to AND OR graph

AO* Algorithm

AO* Algorithm basically based on problem decompositon (Breakdown problem into small pieces) When a problem can be divided into a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution, **AND-OR graphs** or **AND - OR trees** are used for representing the solution.
The decomposition of the problem or problem reduction generates AND arcs.

AND-OR Graph



The figure shows an AND-OR graph

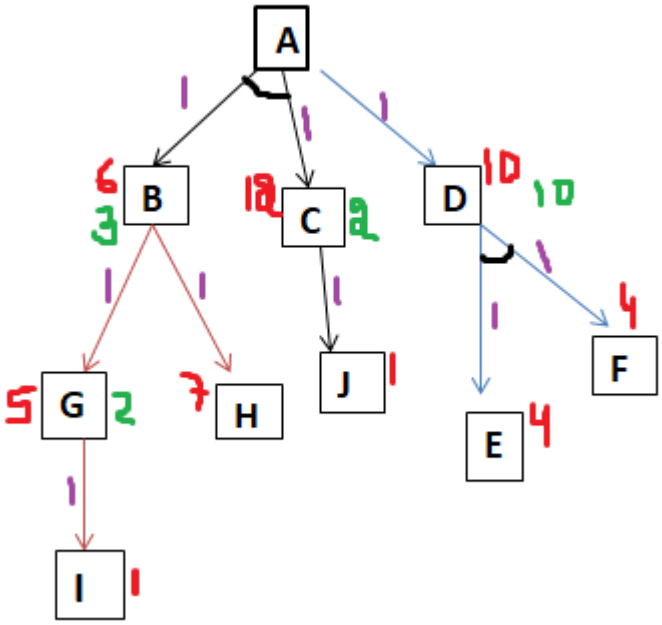
1. To pass any exam, we have two options, either cheating or hard work.
2. In this graph we are given two choices, first do cheating **or (The red line)** work hard and **(The arc)** pass.
3. When we have more than one choice and we have to pick one, we apply **OR condition** to choose one.(That's what we did here).
4. Basically the **ARC** here denote **AND condition**.
5. Here we have replicated the arc between the work hard and the pass because by doing the hard work possibility of passing an exam is more than cheating.

A* Vs AO*

1. Both are part of informed search technique and use heuristic values to solve the problem.
2. The solution is guaranteed in both algorithm.
3. A* **always** gives an **optimal solution** (shortest path with low cost) But It is not guaranteed to that **AO*** always provide **an optimal solutions**.
4. **Reason:** Because AO* does not explore all the solution path once it got solution.

How AO* works

Let's try to understand it with the following diagram



The algorithm always moves towards a **lower cost value.**
Basically, We will calculate the **cost function** here **(F(n)= G (n) + H (n))**
H: heuristic/ estimated value of the nodes. and **G:** actual cost or edge value (here unit value).
Here we have taken the **edges value 1** , meaning we have to focus solely on the **heuristic value.**

1.The **Purple color** values are **edge values (here all are same that is one).**
2.The **Red color** values are **Heuristic values for nodes.**
3.**The Green color** values are **New Heuristic values for nodes.**

Procedure:

1.In the above diagram we have two ways from **A to D** or **A to B-C** (because of and condition). calculate cost to select a path
2.**F(A-D)= 1+10 = 11** and **F(A-BC) = 1 + 1 + 6 +12 = 20**
3.As we can see **F(A-D)** is less than **F(A-BC)** then the algorithm choose the path **F(A-D).**
4.Form D we have one choice that is **F-E.**
5.**F(A-D-FE) = 1+1+ 4 +4 =10**
6.Basically **10** is the cost of reaching **FE from D.** And **Heuristic value of node D** also denote the cost of reaching **FE from D**. So, the new Heuristic value of D is 10.
7.And the Cost from A-D remain same that is **11**.

Suppose we have searched this path and we have got the **Goal State**, then we will never explore the other path. (this is what AO* says but here we are going to explore other path as well to see what happen)

Let's Explore the other path:

1.In the above diagram we have two ways from **A to D** or **A to B-C** (because of and condition). calculate cost to select a path
2.**F(A-D)= 1+10 = 11** and **F(A-BC) = 1 + 1 + 6 +12 = 20**
3.As we know the cost is more of **F(A-BC)** but let's take a look
4.Now from B we have two path G and H , let's calculate the cost
5.**F(B-G)= 5+1 =6** and **F(B-H)= 7 + 1 = 8**
6.So, cost from **F(B-H)** is more than **F(B-G)** we will take the path B-G.
7.The Heuristic value fro**m G to I is 1 but let's calculate the cost form G to I.**

8.**F(G-I) = 1 +1 = 2.** which is less than **Heuristic value 5**. So, the new **Heuristic value form G to I is 2.**

9.If it is a new value, then the cost from **G to B** must also have changed. Let's see the new **cost form (B to G)**

10.F(B-G)= 1+2 =3 . Mean the **New Heuristic value of B is 3.**

11.**But A is associated with both B and C .**

12.As we can see from the diagram **C only have one choice or one node to explore that is J.** The Heuristic value of C is 12.

13.Cost form C to **J= F(C-J) = 1+1= 2** Which is less than Heuristic value

14.No**w the New Heuristic value of C is 2.**

15.**And the New Cost from A- BC that is F(A-BC) = 1+1+2+3 = 7 which is less than F(A-D)=11.**

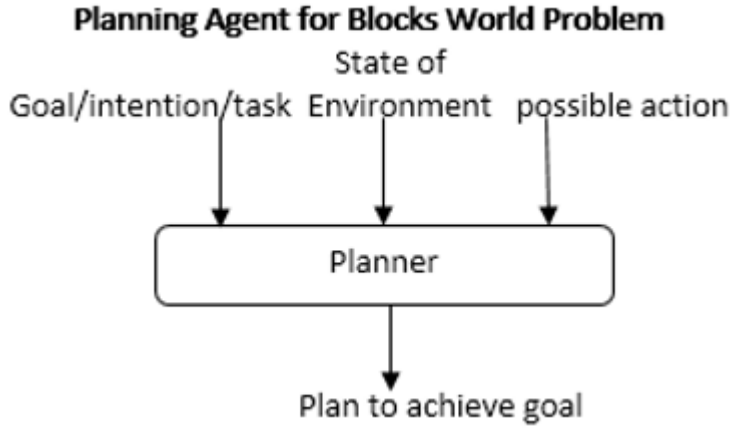
16.In this case Choosing path A-BC is more cost effective and good than that of A-D.

But this will only happen when the algorithm explores this path as well. But according to the algorithm, algorithm will not accelerate this path **(here we have just did it to see how the other path can also be correct).**

But it is not the case in all the cases that it will happen in some cases that the algorithm will get optimal solution.

10**. Explain problem reduction with respect to AND-OR graphs.**

11. Design a planning agent for a Blocks World problem. Assume suitable initial state and final state for the problem



Designing the Agent

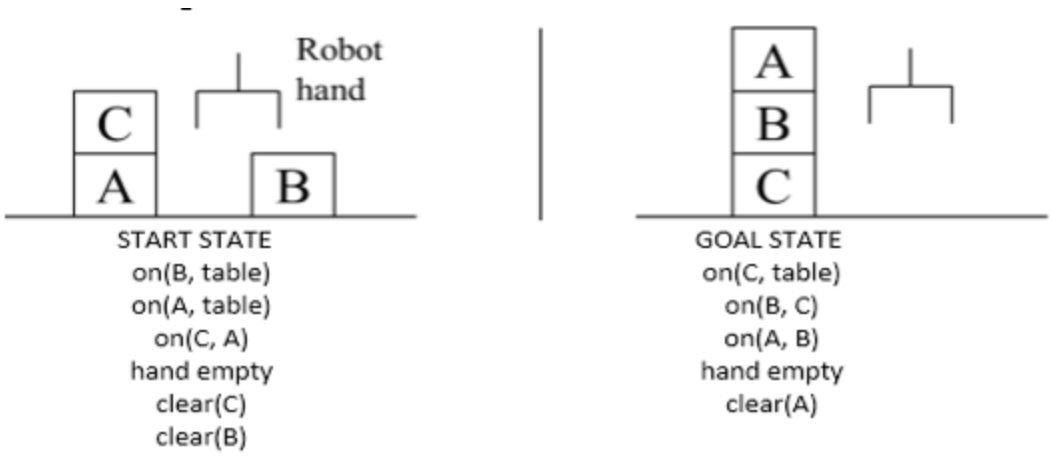
Idea is to give an agent:

- Representation of goal/intention to achieve
- Representation of actions it can perform; and
- Representation of the environment;

  Then have the agent generate a plan to achieve the goal.

  The plan is generated entirely by the planning system, without human intervention.

  Assume start & goal states as below:



a. STRIPS : A planning system – Has rules with precondition deletion list and addition list

Sequence of actions :

b. Grab C

c. Pickup C

d. Place on table C

e. Grab B

f. Pickup B

g. Stack B on C

h. Grab A

i. Pickup A

j. Stack A on B

Rules:

k. R1 : pickup(x)

1. Precondition & Deletion List : hand empty, on(x,table), clear(x)
2. Add List : holding(x)

l. R2 : putdown(x)

1. Precondition & Deletion List : holding(x)
2. Add List : hand empty, on(x,table), clear(x)

m. R3 : stack(x,y)

1. Precondition & Deletion List :holding(x), clear(y)
2. Add List : on(x,y), clear(x)

n. R4 : unstack(x,y)

1. Precondition & Deletion List : on(x,y), clear(x)
2. Add List : holding(x), clear(y)

Plan for the assumed blocks world problem

For the given problem, Start →→ Goal can be achieved by the following sequence:

1. Unstack(C,A)
2. Putdown(C)
3. Pickup(B)
4. Stack(B,C)
5. Pickup(A)
6. Stack(A,B)

12. Compare forward and backward state search algorithms with figure.

Forward Chaining:
Forward Chaining the Inference Engine goes through all the facts, conditions and derivations before deducing the outcome i.e When based on available data a decision is taken then the

process is called as Forwarding chaining, It works from an initial state and reaches to the goal(final decision).

Example:
A
A -> B
B
————————————
He is running.
If he is running, he sweats.
He is sweating.

Backward Chaining:

In this, the inference system knows the final decision or goal, this system starts from the goal and works backwards to determine what facts must be asserted so that the goal can be achieved, i.e it works from goal(final decision) and reaches the initial state.

Example:
B
A -> B
A
————————————
He is sweating.
If he is running, he sweats.
He is running.

Difference between Forwarding Chaining and Backward Chaining:

	Forward Chaining	Backward Chaining
1.	When based on available data a decision is taken then the process is called as Forward chaining.	Backward chaining starts from the goal and works backward to determine what facts must be asserted so that the goal can be achieved.
2.	Forward chaining is known as data-driven technique because we reaches to the goal using the available data.	Backward chaining is known as goal-driven technique because we start from the goal and reaches the initial state in order to extract the facts.
3.	It is a bottom-up approach.	It is a top-down approach.
4.	It applies the Breadth-First Strategy.	It applies the Depth-First Strategy.
5.	Its goal is to get the conclusion.	Its goal is to get the possible facts or the required data.
6.	Slow as it has to use all the rules.	Fast as it has to use only a few rules.

7.	It operates in forward direction i.e it works from initial state to final decision.	It operates in backward direction i.e it works from goal to reach initial state.
8.	Forward chaining is used for the planning, monitoring, control, and interpretation application.	It is used in automated inference engines, theorem proofs, proof assistants and other artificial intelligence applications.

13) Explain planning and acting in nondeterministic domains.

15. Describe Hill climbing search algorithm, what are the problems face by Hill climbing search?

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

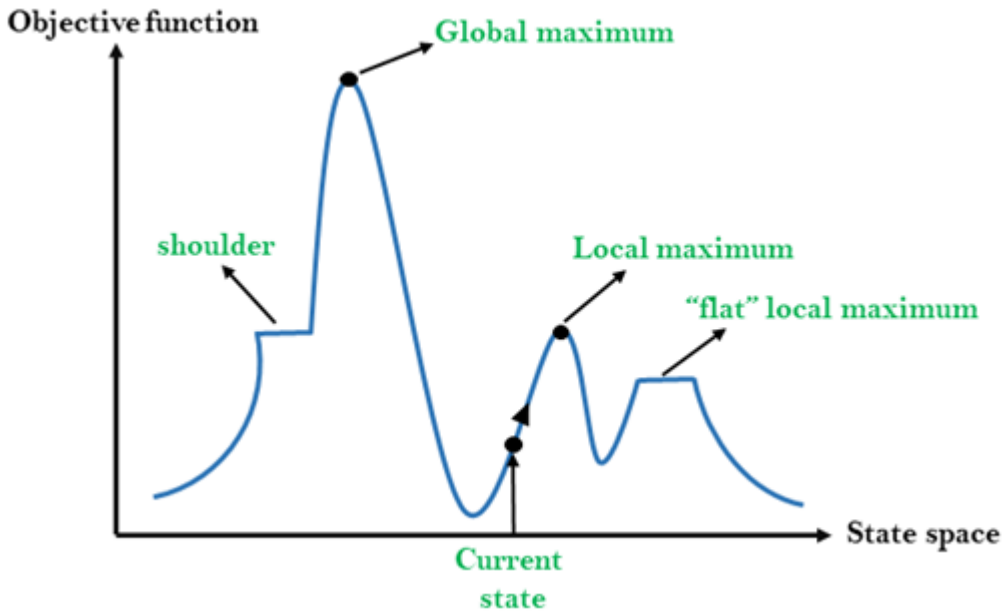
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    a. If it is goal state, then return success and quit.
    b. Else if it is better than the current state then assign new state as a current state.
    c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    a. Let SUCC be a state such that any successor of the current state will be better than it.
    b. For each operator that applies to the current state:
        a. Apply the new operator and generate a new state.
        b. Evaluate the new state.
        c. If it is goal state, then return it and quit, else compare it to the SUCC.
        d. If it is better than SUCC, then set new state as SUCC.

e. If the SUCC is better than the current state, then set current state to SUCC.

- **Step 5:** Exit.

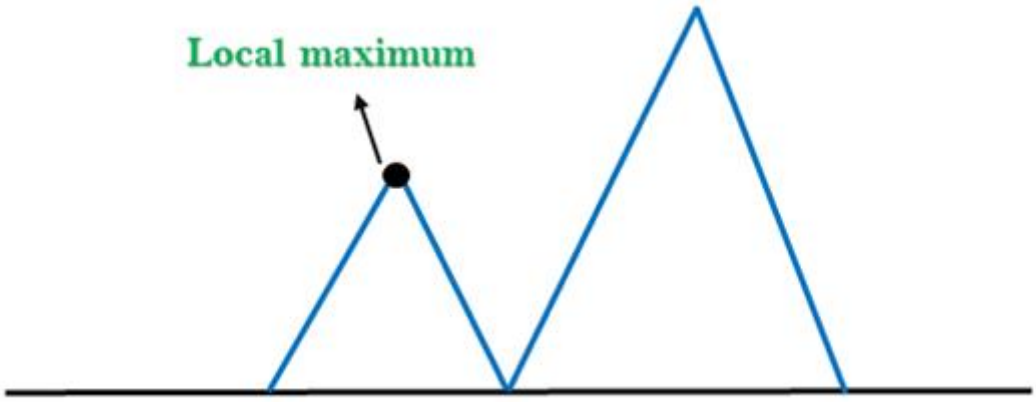
3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm:

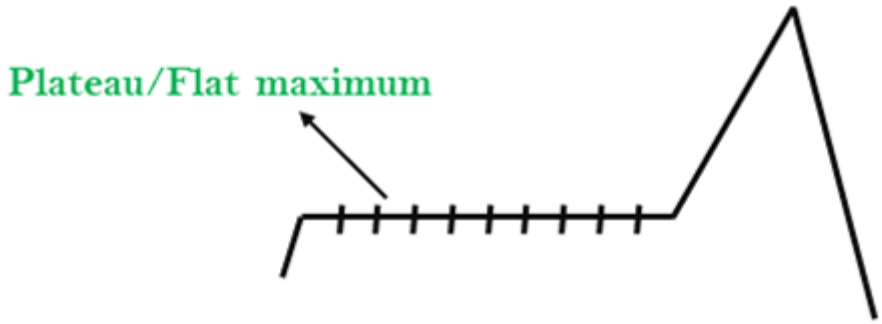
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

1. **Local maximum :** At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
   To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
2. **Plateau :** On plateau all neighbors have same value . Hence, it is not possible to select the best direction.
   To overcome plateaus : Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region
3. **Ridge :** Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.
   To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

4.

18. Discuss about the following: i) Greedy best-first search. (ii) A* search (iii) Memory bounded heuristic search.

Refer 6th and 7th qn

Simplified Memory Bounded A* is a shortest path algorithm based on the A* algorithm. The main advantage of SMA* is that it uses a bounded memory, while the A* algorithm might need exponential memory. All other characteristics of SMA* are inherited from A

SMA* has the following properties

- It works with a heuristic, just as A*
- It is complete if the allowed memory is high enough to store the shallowest solution
- It is optimal if the allowed memory is high enough to store the shallowest optimal solution, otherwise it will return the best solution that fits in the allowed memory
- It avoids repeated states as long as the memory bound allows it
- It will use all memory available
- Enlarging the memory bound of the algorithm will only speed up the calculation
- When enough memory is available to contain the entire search tree, then calculation has an optimal speed

Like A*, it expands the most promising branches according to the heuristic. What sets SMA* apart is that it prunes nodes whose expansion has revealed less promising than expected. The approach allows the algorithm to explore branches and backtrack to explore other branches.

19. Explain Alpha-Beta pruning in Min-Max search. Why it is suitable for two player game?

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

1. α>=β

Key points about alpha-beta pruning:

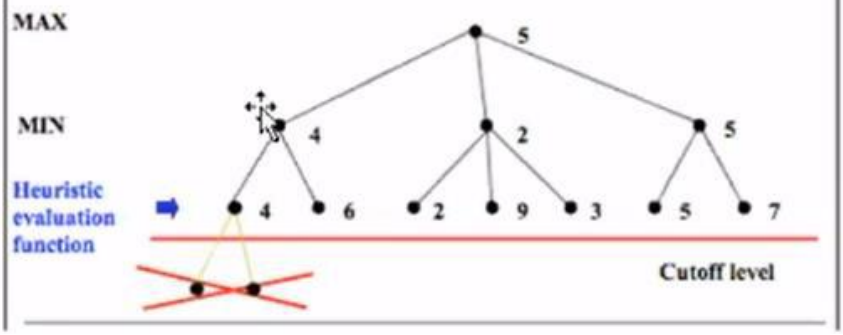
- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
---------------------------------------------------
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
---------------------------------------------------
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Using minimax value estimates

- **Idea:**
  - Cutoff the search tree before the terminal state is reached
  - Use imperfect estimate of the minimax value at the leaves
    - Evaluation function



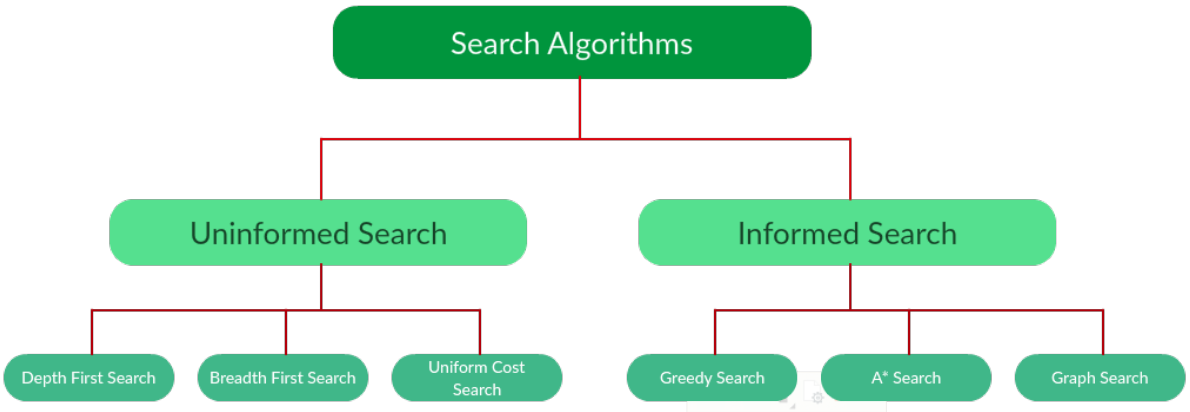
19.

20. **Show the performance measure of various search algorithms.**

Artificial Intelligence is the study of building agents that act rationally. Most of the time, these agents perform some kind of search algorithm in the background in order to achieve their tasks.

A search problem consists of:

- **A State Space.** Set of all possible states where you can be.
- **A Start State.** The state from where the search begins.
- **A Goal Test.** A function that looks at the current state returns whether or not it is the goal state.



Depth First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

Time complexity: Equivalent to the number of nodes traversed in DFS.

$$T(n) = 1 + n^2 + n^3 + ... + n^d = O(n^d)$$

Space complexity: Equivalent to how large can the fringe get.

$$S(n) = O(n \times d)$$

Completeness: DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.
Optimality: DFS is not optimal, meaning the number of steps in reaching the solution, or the cost spent in reaching it is high.

Breadth First Search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root, and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

Time complexity: Equivalent to the number of nodes traversed in BFS until the shallowest solution.

$$T(n) = 1 + n^2 + n^3 + ... + n^s = O(n^s)$$

Space complexity: Equivalent to how large can the fringe get.

$$S(n) = O(n^s)$$

Completeness: BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.
Optimality: BFS is optimal as long as the costs of all edges are equal.

Uniform Cost Search

UCS is different from BFS and DFS because here the costs come into play. In other words, traversing via different edges might not have the same cost. The goal is to find a path where the cumulative sum of costs is least.

Time complexity:

$$T(n) = O(n^{C/\varepsilon})$$

Space complexity:

$$S(n) = O(n^{C/\varepsilon})$$

Advantages:

1. UCS is complete.
2. UCS is optimal.

Disadvantages:

1. Explores options in every “direction”.
2. No information on goal location.

Greedy Search

In greedy search, we expand the node closest to the goal node. The "closeness" is estimated by a heuristic h(x).

Heuristic: A heuristic h is defined as-
h(x) = Estimate of distance of node x from the goal node.
Lower the value of h(x), closer is the node from the goal.

Advantage: Works well with informed search problems, with fewer steps to reach a goal.
Disadvantage: Can turn into unguided DFS in the worst case.

A* Tree Search

A* Tree Search, or simply known as A* Search, combines the strengths of uniform-cost search and greedy search. In this search, the heuristic is the summation of the cost in UCS, denoted by g(x), and the cost in greedy search, denoted by h(x). The summed cost is denoted by f(x).

Heuristic: The following points should be noted with respect to heuristics in A* search.

Here, h(x) is called the **forward cost**, and is an estimate of the distance of the current node from the goal node.

And, g(x) is called the **backward cost**, and is the cumulative cost of a node from the root node.

A* search is optimal only when for all nodes, the forward cost for a node h(x) underestimates the actual cost h*(x) to reach the goal. This property of A* heuristic is called **admissibility**.

Admissibility:

$$0 \leqslant h(x) \leqslant h^*(x)$$

A* Graph Search

A* tree search works well, except that it takes time re-exploring the branches it has already explored. In other words, if the same node has expanded twice in different branches of the search tree, A* search might explore both of those branches, thus wasting time.

A* Graph Search, or simply Graph Search, removes this limitation by adding this rule: **do not expand the same node more than once.**

Heuristic. Graph search is optimal only when the forward cost between two successive nodes A and B, given by h(A) - h (B) , is less than or equal to the backward cost between those two nodes g(A -> B). This property of graph search heuristic is called **consistency**.

Consistency:

$$h(A) - h(B) \leqslant g(A \rightarrow B)$$

21. Formulate the four necessary things to solve a problem

A problem can be defined formally by five components:

• The **initial state** that the agent starts in.

• A description of the possible **actions** available to the agent. Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s. We say that each of these actions is applicable in s.

• A description of what each action does; the formal name for this is the **transition model**, specified by a function RESULT(s, a) that returns the state that results from doing action a in state s. We also use the term successor to refer to any state reachable from a given state by a single action.

Together, **the initial state, actions, and transition model** implicitly define the state **space** of the problem—the set of all states reachable from the initial state by any sequence of actions. The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.

A path in the state space is a sequence of states connected by a sequence of actions.

• The **goal test**, which determines whether a given state is a goal state. Sometimes there is an explicit set of possible goal states, and the test simply checks whether the given state is one of them.

Sometimes the goal is specified by an abstract property rather than an explicitly enumerated set of states. For example, in chess, the goal is to reach a state called "checkmate," where the opponent's king is under attack and can't escape.

• A **path cost** function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure. We can assume that the cost of a path can be described as the sum of the costs of the individual actions along the path.

The **step cost** of taking action a in state s to reach state s is denoted by c(s, a, s).

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
              state, some description of the current world state
              goal, a goal, initially null
              problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
      if seq = failure then return a null action
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

22. What are properties of good system for the representation of knowledge?

To be considered good, a Knowledge Representation system must have the following features:

*1**. Representational adequacy***

It should be able to represent the different kinds of knowledge required.

*2**. Inferential adequacy***

The Knowledge Representation system should be able to come up with new structures or knowledge that it can infer from the original or existing structures.

*3. **Inferential efficiency***

It should be able to integrate additional mechanisms to existing knowledge structures to direct them toward promising directions.

*4**. Acquisitional efficiency***

The Knowledge Representation system should be able to gain new knowledge through automated methods instead of relying on human intervention. However, it should also allow for the injection of information by a knowledge engineer.

To date, no single Knowledge Representation system has all of these properties.

23. Explain different approaches to knowledge representation.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

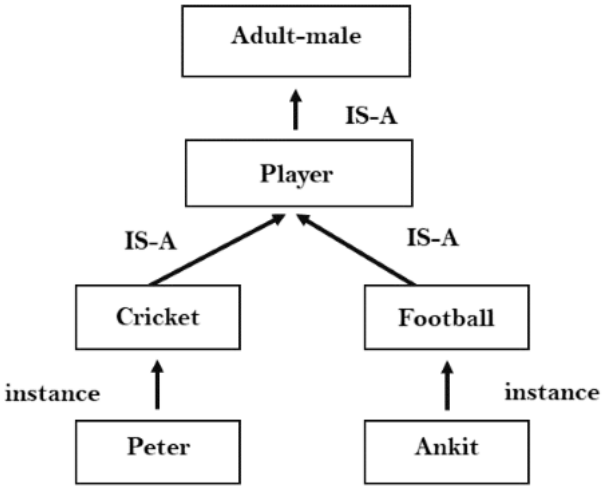
- This approach has little opportunity for inference.

Example: The following is the simple relational knowledge representation.

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.

Example:



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.

Example: Let's suppose there are two statements:

a. Marcus is a man

b. All men are mortal

Then it can be represented as:

man(Marcus)

∀x = man (x) ----------> mortal (x)s

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

24. Distinguish forward and backward reasoning explain with example.

Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A, T1) .....(2)

  Missile(T1) .....(3)
- All of the missiles were sold to country A by Robert.

  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)
- Missiles are weapons.

  Missile(p) → Weapons (p) .....(5)
- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) .....(6)
- Country A is an enemy of America.

  Enemy (A, America) .....(7)
- Robert is American

  American(Robert). .....(8)

Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)**

  Owns(A, T1) .....(2)
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .....(4)**

- **Missile(p) → Weapons (p) .....(5)**
- **Enemy(p, America) →Hostile(p) .....(6)**
- **Enemy (A, America) .....(7)**
- **American(Robert). .....(8)**

Forward Chaining and backward chaining in AI - Javatpoint, First order Logic in Artificial Intelligence | first order logic in ai | FOL | (Eng-Hindi) | #3 - YouTube, Propositional Logic in Artificial Intelligence | propositional logic examples | PL (Eng-Hindi) | #2 - YouTube (to understand)

25. Write in detail about the various steps of knowledge process.

Knowledge engineering projects vary widely in content, scope, and difficulty, but all such projects include the following steps:

1. *Identify the task.* The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be available for each specific problem instance. The task will determine what knowledge must be represented in order to connect problem instances to answers.

2. *Assemble the relevant knowledge.* The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract what they know—a process called knowledge acquisition. At this stage, the knowledge is not represented formally. The idea is to understand the scope of the knowledge base, as determined by the task, and to understand how the domain actually works.

3. *Decide on a vocabulary of predicates, functions, and constants.* That is, translate the important domain-level concepts into logic-level names. This involves many questions of knowledge-engineering style. Like programming style, this can have a significant impact on the eventual success of the project. Once the choices have been made, the result is a vocabulary that is known as the ontology of the domain. The word ontology means a particular theory of the nature of being or existence. The ontology determines what kinds of things exist, but does not determine their specific properties and interrelationships.

4. *Encode general knowledge about the domain.* The knowledge engineer writes down the axioms for all the vocabulary terms. This pins down (to the extent possible) the meaning of the terms, enabling the expert to check the content. Often, this step reveals misconceptions or gaps in the vocabulary that must be fixed by returning to step 3 and iterating through the process.

5. *Encode a description of the specific problem instance.* If the ontology is well thought out, this step will be easy. It will involve writing simple atomic sentences about instances of concepts that are already part of the ontology. For a logical agent, problem instances are supplied by the sensors, whereas a “disembodied” knowledge base is supplied with additional sentences in the same way that traditional programs are supplied with input data.

6. *Pose queries to the inference procedure and get answers.* We can let the inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing. Thus, we avoid the need for writing an application-specific solution algorithm.

7. *Debug the knowledge base.* The answers will be correct for the knowledge base as written, assuming that the inference procedure is sound, but they will not be the ones that the user is expecting. For example, if an axiom is missing, some queries will not be answerable from the knowledge base. A considerable debugging process could ensue. Missing axioms or axioms that are too weak can be easily identified by noticing places where the chain of reasoning stops unexpectedly. Incorrect axioms can be identified because they are false statements about the world. For example, the sentence ∀ x NumOfLegs(x, 4) ⇒ Mammal(x) is false for reptiles, amphibians, and, more importantly, tables. The falsehood of this sentence can be determined independently of the rest of the knowledge base.

26. Translate the following into First Order Logic.

i. Everyone who saves money earns interest.

ii. If there is no interest, then nobody saves money.

(refer 30.)

27. Describe A* algorithm with merits and demerits.

It is the combination of Dijkstra's algorithm and Best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A* search. In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula.

$$f(n) = g(n) + h(n)$$

Where

g (n) : The actual cost path from the start state to the current state.

h (n) : The actual cost path from the current state to goal state.

f (n) : The actual cost path from the start state to the goal state.

For the implementation of A* algorithm we will use two arrays namely OPEN and CLOSE.

OPEN: An array which contains the nodes that has been generated but has not been yet examined.

CLOSE: An array which contains the nodes that have been examined.

Algorithm:

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

Implementation: The implementation of A* algorithm is 8-puzzle game.

Advantages:

- It is complete and optimal.
- It is the best one from other techniques. It is used to solve very complex problems.
- It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages:

- This algorithm is complete if the branching factor is finite and every action has fixed cost.
- The speed execution of A* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute h (n).
- It has complexity problems.

28. How do you examine about Backtracking search for CSP?

Commutativity - A problem is commutative if the order of application of any given set of actions has no effect on the outcome.

CSPs are commutative because when assigning values to variables, we reach the same partial assignment regardless of order. Therefore, we need only consider a single variable at each node in the search tree.

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
    remove {var = value} and inferences from assignment
  return failure
```

Figure 6.5 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INFERENCE can optionally be used to impose arc-, path-, or k-consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are removed from the current assignment and a new value is tried.

The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then **BACKTRACK** returns failure, causing the previous call to try another value.

BACKTRACKING-SEARCH keeps only a single representation of a state and alters that representation rather than creating new ones.

To solve CSPs efficiently without domain-specific knowledge, address following questions:

1. Which variable should be assigned next (SELECT-UNASSIGNED-VARIABLE), and in what order should its values be tried (ORDER-DOMAIN-VALUES)?

2. What inferences should be performed at each step in the search (INFERENCE)?

3. When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

29. Explain the steps involved in converting the propositional logic statement

into CNF with a suitable example.

Steps to Convert to CNF (Conjunctive Normal Form)

Every sentence in Propositional Logic is logically equivalent to a conjunction of

disjunctions of literals. A sentence expressed as a conjunction of disjunctions of literals is

said to be in Conjunctive normal Form or CNF.

1. Eliminate implication '→'

a → b = ~a v b

~ (a ^ b) = ~ a v ~ b …….. DeMorgan's Law

~ (a v b) = ~ a ^ ~ b …….. DeMorgan's Law

~ (~a) = a

2. Eliminate Existential Quantifier '∃'

To eliminate an **independent Existential Quantifier**, replace the variable by a Skolem constant. This process is called as Skolemization.

Example: ∃y: President (y)

Here 'y' is an independent quantifier so we can replace 'y' by any name (say – George Bush).

So, ∃y: President (y) becomes President (George Bush).

To eliminate a **dependent Existential Quantifier** we replace its variable by Skolem Function that accepts the value of 'x' and returns the corresponding value of 'y.'

Example: ∀x : ∃y : father_of (x, y)

Here 'y' is dependent on 'x', so we replace 'y' by S(x).

So, ∀x : ∃y : father_of (x, y) becomes ∀x : ∃y : father_of (x, S(x)).

3. Eliminate Universal Quantifier '∀'

To eliminate the Universal Quantifier, drop the prefix in PRENEX NORMAL FORM i.e. just drop ∀ and the sentence then becomes in PRENEX NORMAL FORM.

4. Eliminate AND '^'

a ^ b splits the entire clause into two separate clauses i.e. a and b

(a v b) ^ c splits the entire clause into two separate clauses a v b and c

(a ^ b) v c splits the clause into two clauses i.e. a v c and b v c

To eliminate '^' break the clause into two, if you cannot break the clause, distribute the OR 'v' and then break the clause.

30. Translate the following into First Order Logic.

I. Everyone who saves money earns interest.

ii. If there is no interest, then nobody saves money.

Let us use the symbols:

S (x, y): x saves y,

M (x): x is money,

I (x): x is interest,

I (x): x is interest,

E (x, y): x Earns y.

Then the given statement on symbolization becomes:

(i) $(\forall x)((\exists y)(S(x, y) \wedge M(y)) \rightarrow (\exists y)(I(y) \wedge E(x, y)))$

(ii) $\sim(\exists x)\, I(x) \rightarrow (\forall x)(\forall y)(S(x, y) \rightarrow \sim M(y))$

31. Summarize your views about following.

i) Syntax of propositional logic ii) Semantics of propositional logic iii)

Simple knowledge base iv) Inference

Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

- **Atomic Propositions**
- **Compound propositions**

Atomic Proposition: Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

Example:

a) 2+2 is 4, it is an atomic proposition as it is a true fact.

b)"The Sun is cold" is also a proposition as it is a false fact.

Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

a) "It is raining today, and street is wet."

b) "Ankit is a doctor, and his clinic is in Mumbai."

Semantics Semantics provides the “meaning” of propositional logic formulae. It is defined very precisely in a mathematical way. The semantics allows us to identify correct inference rules, for instance transformations of formulae which preserve the meaning.

Intuitively, the meaning of “A ∧ B” is that “this is only true if both A and B are true”

Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world.

Inference:
In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.

Following are some terminologies related to inference rules:

• Implication: It is one of the logical connectives which can be represented as P → Q. It is a Boolean expression.

• Converse: The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as Q → P.

• Contrapositive: The negation of converse is termed as contrapositive, and it can be represented as ¬ Q → ¬ P.

• Inverse: The negation of implication is called inverse. It can be represented as ¬ P → ¬ Q.

---

32. Explain in detail about models for predicate logic?

Predicate Logic - Definition

A predicate is an expression of one or more variables determined on some specific domain. A predicate with variables can be made a proposition by either authorizing a value to the variable or by quantifying the variable.

Quantifier:

The variable of predicates is quantified by quantifiers. There are two types of quantifier in predicate logic - Existential Quantifier and Universal Quantifier.

Existential quantifier:

If p(x) is a proposition over the universe U. Then it is denoted as ∃x p(x) and read as "There exists at least one value in the universe of variable x such that p(x) is true. The quantifier ∃ is called the existential quantifier.

There are several ways to write a proposition, with an existential quantifier, i.e.,

(∃x∈A)p(x) or ∃x∈A such that p (x) or (∃x)p(x) or p(x) is true for some x ∈A.

Universal Quantifier:

If p(x) is a proposition over the universe U. Then it is denoted as ∀x,p(x) and read as "For every x∈U,p(x) is true." The quantifier ∀ is called the Universal Quantifier.

There are several ways to write a proposition, with a universal quantifier.

∀x∈A,p(x) or p(x), ∀x ∈A Or ∀x,p(x) or p(x) is true for all x ∈A.

Negation of Quantified Propositions:

When we negate a quantified proposition, i.e., when a universally quantified proposition is negated, we obtain an existentially quantified proposition,and when an existentially quantified proposition is negated, we obtain a universally quantified proposition.

The two rules for negation of quantified proposition are as follows. These are also called DeMorgan's Law.

Propositions with Multiple Quantifiers:

The proposition having more than one variable can be quantified with multiple quantifiers. The multiple universal quantifiers can be arranged in any order without altering the meaning of the resulting proposition.

Also, the multiple existential quantifiers can be arranged in any order without altering the meaning of the proposition.

The proposition which contains both universal and existential quantifiers, the order of those quantifiers can't be exchanged without altering the meaning of the proposition, e.g., the proposition ∃x ∀ y p(x,y) means "There exists some x such that p (x, y) is true for every y."

---

33. Relate first order logic with proposition logic and discuss in detail about the same.

Propositional logic is an analytical statement which is either true or false. It is basically a technique that represents the knowledge in logical & mathematical form. There are two types of propositional logic; Atomic and Compound Propositions.

Since propositional logic works on 0 and 1 thus it is also known as 'Boolean Logic'.

- Proposition logic can be either true or false it can never be both.
- In this type of logic, symbolic variables are used in order to represent the logic and any logic can be used for representing the variable.
- t is comprised of objects, relations, functions, and logical connectives.
- Proposition formula which is always false is called 'Contradiction' whereas a proposition formula which is always true is called 'Tautology'.

First-Order Logic is another knowledge representation in AI which is an extended part of PL. FOL articulates the natural language statements briefly. Another name of First-Order Logic is 'Predicate Logic'.

FOL is known as the powerful language which is used to develop information related to objects in a very easy way.

- Unlike PL, FOL assumes some of the facts that are related to objects, relations, and functions.
- FOL has two main key features or you can say parts that are; 'Syntax' & 'Semantics'.

Key differences between PL and FOL

1. Propositional Logic converts a complete sentence into a symbol and makes it logical whereas in First-Order Logic relation of a particular sentence will be made that involves relations, constants, functions, and constants.
2. The limitation of PL is that it does not represent any individual entities whereas FOL can easily represent the individual establishment that means if you are writing a single sentence then it can be easily represented in FOL.

PL does not signify or express the generalization, specialization or pattern for example ‘QUANTIFIERS’ cannot be used in PL but in FOL users can easily use quantifiers as it does express the generalization, specialization, and pattern

34. How would you identify an example for resolution?

Resolution in First-order logic - Javatpoint

35. Explain the algorithms for planning as state-space search.

36. Explain planning and acting in nondeterministic domains.

37. explain different forms of learning.

There are 4 types of learning

1. Supervised learning
2. Unsupervised learning
3. Semi-supervised learning
4. Reinforced learning

1. Supervised learning

Supervised machine learning can take what it has learned in the past and apply that to new data using labelled examples to predict future patterns and events. It learns by explicit example.

Supervised learning requires that the algorithm's possible outputs are already known and that the data used to train the algorithm is already labelled with correct answers. It's like teaching a child that 2+2=4 or showing an image of a dog and teaching the child that it is called a dog. The approach to supervised machine learning is essentially the same – it is presented with all the information it needs to reach pre-determined conclusions. It learns how to reach the conclusion, just like a child would learn how to reach the total of '5' and the few, pre-determined ways to get there, for example, 2+3 and 1+4.

Supervised learning is further divided into:

Regression trains on and predicts a continuous-valued response, for example predicting real estate prices.

Classification attempts to find the appropriate class label, such as analyzing positive/negative sentiment, male and female persons, benign and malignant tumors, secure and unsecure loans etc.

2. Unsupervised learning

Supervised learning tasks find patterns where we have a dataset of "right answers" to learn from. **Unsupervised learning tasks find patterns where we don't.** This may be because the "right answers" are unobservable, or infeasible to obtain, or maybe for a given problem, there isn't even a "right answer" per se.

In unsupervised learning, neither a training data set nor a list of outcomes is provided. **The AI enters the problem blind – with only its faultless logical operations to guide it.** Imagine yourself as a person that has never heard of or seen any sport being played. You get taken to a football game and left to figure out what it is that you are observing. You can't refer to your knowledge of other sports and try to draw up similarities and differences that will eventually boil down to an understanding of football. You have nothing but your cognitive ability. Unsupervised learning places the AI in an equivalent of this situation and leaves it to learn using only its on/off logic mechanisms that are used in all computer systems.

3. Semi-supervised learning (SSL)

Semi-supervised learning falls somewhere in the middle of supervised and unsupervised learning. It is used because many problems that AI is used to solving require a balance of both approaches.

In many cases the reference data needed for solving the problem is available, but it is either incomplete or somehow inaccurate

4. Reinforcement learning

Reinforcement learning is a type of dynamic programming that trains algorithms using **a system of reward and punishment.**

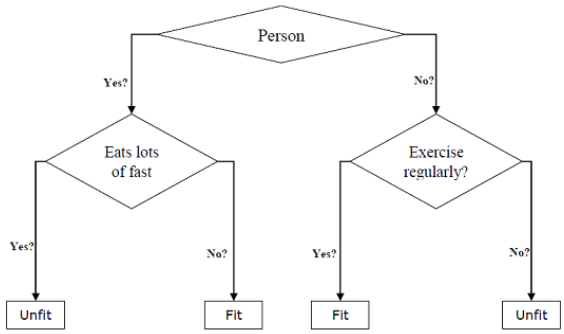
A reinforcement learning algorithm, or agent, learns by interacting with its environment. It receives rewards by performing correctly and penalties for doing so incorrectly. Therefore, it learns without having to be directly taught by a human – **it learns by seeking the greatest reward and minimising penalty**

38. explain decision trees

Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. Decisions tress are the most powerful algorithms that falls under the category of supervised algorithms.

They can be used for both classification and regression tasks. The two main entities of a tree are decision nodes, where the data is split and leaves, where we got outcome

Ex:



In the above decision tree, the question are decision nodes and final outcomes are leaves. We have the following two types of decision trees −

- **Classification decision trees** − In this kind of decision trees, the decision variable is categorical. The above decision tree is an example of classification decision tree.
- **Regression decision trees** − In this kind of decision trees, the decision variable is continuous.
-

Features

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees

39. Describe the following using fol

Kinship domain, numbers sets and lists, Wumpus world problem

Representing knowledge in FOL

Example:

Kinship domain

- **Objects:** people

  $John$, $Mary$, $Jane$, …
- **Properties:** gender

  $Male\ (x)$, $Female\ (x)$
- **Relations:** parenthood, brotherhood, marriage

  $Parent\ (x, y)$, $Brother\ (x, y)$, $Spouse\ (x, y)$
- **Functions:** mother-of (one for each person x)

  $MotherOf\ (x)$

Kinship domain in FOL

Relations between predicates and functions: write down what we know about them; how relate to each other.

- Male and female are disjoint categories

  $\forall x\ Male\ (x) \Leftrightarrow \neg Female\ (x)$
- Parent and child relations are inverse

  $\forall x, y\ Parent\ (x, y) \Leftrightarrow Child\ (y, x)$
- A grandparent is a parent of parent

  $\forall g, c\ Grandparent(g, c) \Leftrightarrow \exists p\ Parent(g, p) \wedge Parent(p, c)$
- A sibling is another child of one's parents

  $\forall x, y\ Sibling\ (x, y) \Leftrightarrow (x \neq y) \wedge \exists p\ Parent\ (p, x) \wedge Parent\ (p, y$
- And so on ….

Atomic proposition variable for Wumpus world:

- Let $\mathbf{P_{i,j}}$ be true if there is a Pit in the room [i, j].
- Let $\mathbf{B_{i,j}}$ be true if agent perceives breeze in [i, j], (dead or alive).
- Let $\mathbf{W_{i,j}}$ be true if there is wumpus in the square[i, j].
- Let $\mathbf{S_{i,j}}$ be true if agent perceives stench in the square [i, j].
- Let $\mathbf{V_{i,j}}$ be true if that square[i, j] is visited.
- Let $\mathbf{G_{i,j}}$ be true if there is gold (and glitter) in the square [i, j].
- Let $\mathbf{OK_{i,j}}$ be true if the room is safe.

Some Propositional Rules for the wumpus world:

(R1) $\neg S_{11} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{21}$

(R2) $\neg S_{21} \rightarrow \neg W_{11} \wedge \neg W_{21} \wedge \neg W_{22} \wedge \neg W_{31}$

(R3) $\neg S_{12} \rightarrow \neg W_{11} \wedge \neg W_{12} \wedge \neg W_{22} \wedge \neg W_{13}$

(R4) $S_{12} \rightarrow W_{13} \vee W_{12} \vee W_{22} \vee W_{11}$

40. inference rules for propositional logic.

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences.

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

1. Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

Example: Let's represent, P(c): "**A byte contains 8 bits**", so for **∀ x P(x)** "**All bytes contain 8 bits**.", it will also be true.

2. Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$.

Example:1.

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c

3. Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$

4. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$
- **Example: Let's say that,**

  "Priyanka got good marks in English."

  "Therefore, someone got good marks in English."

41. Decision tree learning. Decision rules?

42.how to use it to classify?

43. explain in detail about neural network architecture.

Neural Network: Architecture

Neural Networks are complex structures made of artificial neurons that can take in multiple inputs to produce a single output. This is the primary job of a Neural Network – to transform input into a meaningful output. Usually, a Neural Network consists of an input and output layer with one or multiple hidden layers within.
In a Neural Network, all the neurons influence each other, and hence, they are all connected. The network can acknowledge and observe every aspect of the dataset at hand and how the different parts of data may or may not relate to each other. This is how Neural Networks are capable of finding extremely complex patterns in vast volumes of data.

In a Neural Network, the flow of information occurs in two ways –

Feedforward Networks: In this model, the signals only travel in one direction, towards the output layer. Feedforward Networks have an input layer and a single output layer with zero or multiple hidden layers. They are widely used in pattern recognition.

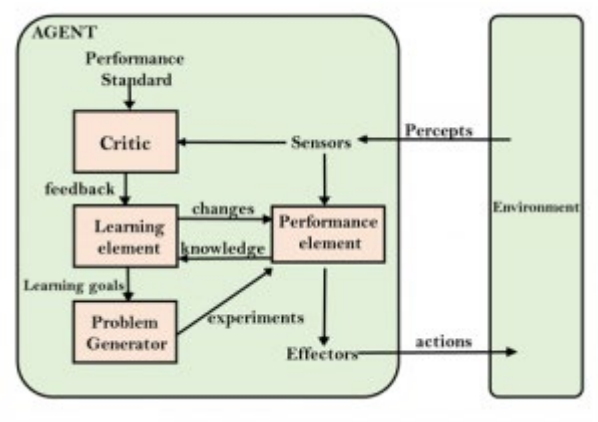
Feedback Networks: In this model, the recurrent or interactive networks use their internal state (memory) to process the sequence of inputs. In them, signals can travel in both directions through the loops (hidden layer/s) in the network. They are typically used in time-series and sequential tasks.

45,49. explain structure of learning agents. What Is the role of critic?

A learning agent in AI is the type of agent which can learn from its past experiences or it has learning capabilities.
It starts to act with basic knowledge and then able to act and adapt automatically through learning.
A learning agent has mainly four conceptual components, which are:

1. **Learning element :**It is responsible for making improvements by learning from the environment

2. **Critic:** Learning element takes feedback from critic which describes how well the agent is doing with respect to a fixed performance standard.
3. **Performance element:** It is responsile for selecting external action
4. **Problem Generator:** This component is responsible for suggesting actions that will lead to new and informative experiences.



Critic tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so. It is important that the performance standard be fixed. Conceptually, one should think of it as being outside the agent altogether, because the agent must not modify it to fit its own behaviour

46,47. Explain reinforcement learning: passive and active

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself

whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

Both active and passive reinforcement learning are types of RL. In case of passive RL, the agent's policy is fixed which means that it is ***told what to do***. In contrast to this, in active RL, an agent ***needs to decide what to do*** as there's no fixed policy that it can act on. Therefore, the goal of a passive RL agent is to execute a fixed policy (sequence of actions) and evaluate it while that of an active RL agent is to act and learn an optimal policy.

48,53: learning with hidden variables, em algorithm

50. explain nonparametric machine learning.

Explain any one nml algorithm

Algorithms that do not make strong assumptions about the form of the mapping function are called nonparametric machine learning algorithms. By not making assumptions, they are free to learn any functional form from the training data.

nonparametric methods seek to best fit the training data in constructing the mapping function, whilst maintaining some ability to generalize to unseen data. As such, they are able to fit a large number of functional forms.

K nearest neighbour

- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

52. applications of reinforcement learning

Manufacturing

Inventory management

Delivery management

Power systems

Finance sector

Self driving cars

Industry automation

Natural language processiong

Face and voice recognition

Health care

Gaming

CNCs and robotics