Chapter-1

AI can be defined in four catagory =>

1. Thinking humanly → AI means making machines that think like humans (robot)
2. Thinking rationally → AI means making machines that reason logically. (calculator) like solving math problem step by step.
3. Acting humanly → AI means making machines that act like a human. (Chatbot.)
4. Acting rationally → AI means making machines that do the right thing based on what they know. A self driving car.

Acting humanly

Turing test => machine is intelligent if it can fool human in conversation.

To pass turing test machine needs;

Natural Language processing → to communicate in human language.

Knowledge representation → to store information.

Automated Reasoning → to use knowledge and answer logically.

Machine learning → to improve with experience.

Thinking humanly
Cognitive modelling is the study human thought to copy it in machines. Three ways to study →
Introspection → observing own thoughts.
Psychologic experiment → see behaviour in different situation.
Brain studies. (neuroscience) → using machines to see how brain works

Thinking rationally (Law of thoughts)
use logic and reasoning rules.

Acting rationally (Rational agent approach)
Agent ⇒ something that acts
Rational agent ⇒ acts to give best possible outcomes based on what it knows.

Syllogism is a kind of logical reasoning where you draw a conclusion from two or more statements (called premises)

Materialism is a view that says ⇒ everything that exists is made up of matter (physical stuff) and all things that happen can be explained by interaction of matter and physical laws.

Rationalism ⇒ Reason (logical thinking) is the main source of knowledge.

Empiricism ⇒ All knowledge comes mainly from experience through our senses.

Logical positivism ⇒ It combine logic + science + empiricism

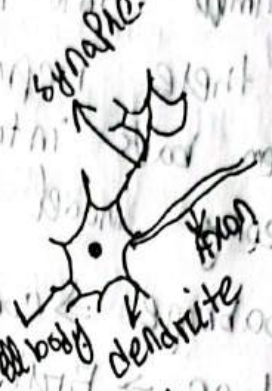
Utilitarianism ⇒ It is a moral philosophy as AI is used in medical science.

Economics is about how people make choices to get what they want (maximize payoff)

Decision Theory ⇒ Combines probability and utility to make the best choice under uncertainity.

Sequential (operations research) ⇒ Focuses on decisions that happen in sequence not just one time choices.

Neuron structure



Cell body (soma): Contains nucleus.
Dendrites: Receive signals.
Axon: sends signals far distance
Synapses: Junctions with other neurons.

Brain Vs computer

Feature	Human brain	Supercomputer
#neuron/CPU	10^{11} neurons	10^{4} CPUs
Storage	10^{14} bits RAM	10^{14} bits RAM
cycle time	10^{-3} sec	10^{-9} sec
connectivity	10^{14} synapses	Lower than brain

Computers are faster but less interconnected
Brains excel at parallel processing and adaptability.

283
164
7 5

1 2 3
8 4
7 6 5

Psychology:- It is used to model intelligent behavior in machines.

#Cognitive psychology views the brain as an information-processing machine.

Craik's small-scale model of agent =>

① Translate stimuli into internal representation.
② Manipulate these representation to predict outcomes.
③ Convert them back into actions.

cognitive theory model the mind like computer programs

Charles Babbage => compute mathematical table

Ada Lovelace → First programmer, wrote programs for analytic engine.

AI influenced time sharing, interactive interpreters, graphical interfaces, data structures, symbolic/functional/object oriented programing.

Control theory and cybernetics study how systems can regulate themselves to achieve goals automatically.

Norbert father of cybernetics. It's idea is to minimize errors in systems.

Homeostasis => Maintaining stability through feedback loops.

Stochastic optimal control => Design systems to maximize performance over time.

Control theory use calculus and matrix algebra → continious variable.

AI use logic and computation → handles language, vision, Planning.

Understanding and Processing of language is central to intelligence.

Skinner → behaviorist theory of language - learning by reinforcement.

Chomsky → criticise skinner → human can create new sentences they have never heard. Introduced syntactic structures.

NLP ⇒ Natural language processing combines AI and linguistics to Process language.

AI became a separate field, because ⇒

It aimed to duplicate human faculties like creativity, language, self improvement. It needed autonomous machines on complex environments, unlike control theory or operation research.

AI needs background knowledge to handle meaning and context.

SOAR ⇒ Example of a complete agent architecture.
Can integrate different AI subfields: reasoning, planning, learning and memory.

Modern AI focuses on less on algorithms and more on data - large datasets allow simpler algorithms to perform very well.

SOAR is a cognitive architecture that helps agent think, learn and solve Problems using rules and goals.

Period	Focus	Points
1956–69	⇒ Birth of AI	⇒ AI as new field, started journey
1960	⇒ Early enthusiam	⇒ First NLP, but hardware was too
1969–79	⇒ knowledge base system	⇒ Focus on knowledge.
1980–87	⇒ Expert systems	⇒ commercial in business and ind
1987–93	→ AI winter	→ little progress, low confidence.
1986–95	→ Return of Neural network	⇒ Backpropogation algorithm
1995–Present	⇒ intelligent agents	⇒ SOAR, web-bots.
2001–Present	⇒ Big data era	⇒ Data matters more than algori

Chapter-3

- Problem solving agents that make decision and take action to reach a specific goal.
- Goal formulation is the process of deciding what the agent wants to achieve.
- Problem formulation is the process of deciding what actions and states the agent should consider to reach the goal.
- Problem is a situation where the agent has to make decision to achieve its goal.
- Search is the process of finding a sequence of action that leads to goal.
- Solution is the sequence of actions found by the search that actually achieves the goal.
- Execution is carrying out the chosen sequence of actions.
- Open loop problem is a situation where the agent follows a fixed plan without paying attention to feedback from the environment.
- Initial state is the starting point of the agent before solving problem.
- Transition model is a function that tells us what happens after doing an action in a given state.
- Successor is the new state that comes after doing an action.
- State space are all the possible states that can be reached from initial state by applying action.
- Path is a sequence of connected state and actions leading from the start to a goal.

Path cost is the total cost of following a path; step cost is the cost of a single action.

Optimal solution is the best solution among all the path with the lowest path cost

Abstraction is the process of removing unnecessary details from a problem so that only the important parts remain.

A problem will go through this steps ⇒

States, initial state, Actions, Transition model, Goal test, Path cost.

Search Tree: An action sequence that reaches the goal state.

Root node → initial state.
Branches → actions.
Nodes → states in the problem state space.

types of nodes ⇒

Parent node ⇒ node from which another is generated.
child node ⇒ result of applying an action.
Leaf node ⇒ node with no children
Frontier → set of all leaf nodes available for expansion at a given time

Process of search steps

① start at the initial state
② Test if it's goal state.
③ Expand the node → apply legal actions to generate successors
④ Add succession to frontier.
⑤ select one node from the frontier and repeat.

Repeated state in the same generated multiple times

Arad → Sibiu → Arad.

* Loopy path is a path that revisits a state
search tree infinite.
* Redundant path is different paths leading to same state, but some are worse. Arad → Sibiu
Arad → XXX → YYY → Sibiu

Algorithms

TREE-SEARCH

- Expand nodes without remembering where it has been.
- Can revisit the same state multiple times. (loopy path, redundant paths remain)

GRAPH SEARCH

- Add an explored set (closed list) to remember visited states.
- Avoids repeated and redundant search.
- Frontier always separates explored region from unexplored region

Node

A node is like a record card that stores information about one step in each search.

STATE - The situation you are currently in.

PARENT - The previous step that brought you to here.

ACTION → The move you made to reach this place.

PATH-COST → How much it cost to reach this place from starting point

* Every node has a parent pointer which points back to previous step.
* When you take an action from one state, you get a new child node.

State = Just the situation/place.
Node = state + extra information.

Queue: we need a place to keep nodes before expanding them. This place is called the frontier or queue.

FIFO => First in first out => Breadth First search (BFS)
LIFO => Last in first out => Depth First search (DFS)
Priority queue => Always take out lowest cost node first => Uniform Cost search / A*
We use explored set to avoid repeating work.

Algorithm performance =>

Completeness => will it always find a solution if one exists?
Optimality => will it always find the best solution?
Time complexity => How long it takes
Space complexity => how much memory it uses.

b = Branching Factor (b) => how many choices you have at each step.
d = Depth (d) => Length of the shortest path to the goal.
Search Cost => The resources spent searching. (time, memory)
Total cost => search cost + path cost
Path cost is the actual cost of path found (money, fuel)

|V| = number of vertices (states/places)
|E| = number of edges (connection between states)

Uninformed Search:- The search method which does not know extra about the problem except the starting point, the rules/actions, the goal test.

BFS:- It starts from initial state or root then expand the root first. Then expand all children and grandchildren and so on.

It always expand the shallowest (nearest) node first.

It uses FIFO ⇒
new nodes are added to the back
oldest nodes expanded first.

BFS Function:-

Function BFS (Problem):
Create a node for the start state
if start state is goal → return it
Put start node in a FIFO queue (frontier)
explored = empty set.

loop:
if frontier is empty → fail (no solution)
take node from frontier (shallowest)
add node.state to explored
for each possible action:
generate child node
if child.state not in explored or frontier:
if child.state is goal → return solution
put child into frontier.

BFS Properties:

Completeness: Yes, because it checks all levels one by one, it will never miss the goal.

Optimality: BFS guarantees the shortest solution only if all steps have equal cost. But if some roads take more time, it does not guaranteed to be optimal.

Time complexity: If the solution is at depth d, branching factor = b (choices of each step)

Time = $O(b^d)$

Space complexity: BFS keeps all nodes in memory.

space = $O(b^d)$. It is worse than time as you must store all frontier nodes.

Uniform Cost Search

UCS expands the node with the lowest total path cost g(n) from the start to that node. It chose the priority queue and always chooses the cheapest path explored so far.

BFS ⇒ Choose route with fewest turns.
UCS ⇒ Choose route with best cost/time, even if it has more turns.

Works as:

① Start from initial node with cost = 0.
② Put it into priority queue (order by path cost).
③ Remove the node with lowest path cost.
if it is goal → return solution.
expand it and add children to queue with path cost.
If a child already exists in frontier but with a higher cost replace it with a cheaper one.

Pseudocode:

```
function UCS (Problem):
    start_node = state with cost 0.
    frontier = Priority queue (ordered by path cost)
    explored = empty set.

    while frontier not empty:
        node = frontier.pop_lowest_cost()
        if node is goal:
            return solution
        add node to explored
        for each child of node:
            if child not in explored or frontier:
                add child to frontier.
            else if child is in frontier with higher cost:
                replace with cheaper child
```

UCS Properties:

Completeness: Yes, As long as step costs are positive.

Optimality: Yes, It guarantees the cheapest solution.

Time complexity: cost of optimal solution (C^*), minimum step cost (ε)

so in worst case, $O(b^{(1+C^*/\varepsilon)})$

Space Complexity: same as time and also exponential. It keeps many nodes in memory because of priority queue.

Depth First Search DFS:-

DFS always expands the deepest node first in the search tree.
It goes as far down as possible and when it hits a dead end, it backtracks and tries another path.

DFS Works:- Start at the root (initial state)
Always expand the most recent node (deepest in the current path)
If goal found → stop, and if stuck → backtrack and try a different branch
Uses a LIFO (Last in, First out) stack.

Pseudocode:-

```
function DFS(Problem):
    start_node = initial state
    frontier = stack with start_node
    explored = empty set
    while frontier not empty:
        node = frontier.pop()
        if node is goal:
            return solution
        add node to explored
        for each child of node (in order):
            if child not in explored or frontier:
                push child into frontier
```

Time: $b^{(1+C^*/\varepsilon)}$ C^* = optimal solution, ε = minimum cost

Tree Search DFS:- Does not check repeated states. Can get stuck in loops.

Graph Search DFS:- Avoids repeated states (keeps explored set)

DFS Properties:-

Completeness:- Tree search DFS ⇒ Not complete ⇒ may loop forever in infinite
Graph search DFS ⇒ complete ⇒ if the state space is finite

Optimality:- Not optimal. DFS might find a solution that's not the best.

Time complexity:- maximum depth = m, branching factor = b
$O(b^m)$

Space complexity:- Main advantage: Needs only $O(bm)$. (much less than BFS)

Depth Limited Search (DLS):-

DFS but with a maximum depth limit. Avoids infinite loops by stopping at limit.

Properties:-

Completeness:- If depth limit < d (goal is deeper) ⇒ incomplete ⇒ might miss
If depth limit ≥ d ⇒ complete.

optimality:- Not optimal. If goal is at depth 3 but you set limit to, you might return a deeper solution first.

Time complexity:- $O(b^l)$ (where l = deep limit)

space complexity:- $O(b*l)$ → very low memory (like DFS).

Iterative Deepening DFS:-

It is a search method where we keep increasing the depth limit step by step (0, 1, 2, ...) and run depth-limited search each time until we find the goal.

Properties:

Completeness: Yes. If branching factor is finite.

optimality: yes. if Path cost grows with depth.

Time complexity: $O(b^d)$ same as BFS.

Space complexity: $O(b*d)$ same as DFS.

Algorithm	Use in	Complete	Optimal	Time	space
BFS	~~Yes~~	Yes	Yes	$O(b^d)$	$O(b^d)$
DFS	memory tight	X	X	$O(b^m)$	$O(bm)$
DLS	when know a safe depth limit	sometime	X	$O(b^l)$	$O(bl)$
IDDFS	solution depth unknow but memory limited	Yes	Yes	$O(b^d)$	$O(bd)$
Uniform	with varying distance	Yes	Yes	$O(b^{(1+C^*/\epsilon)})$	$O(b^{(1+C^*/\epsilon)})$
Bidirectional	Routing	Yes	Yes	$O(b^{(d/2)})$	$O(b^{(d/2)})$

Bidirection search method:

It is a search method where we start searching forward from the start state and backward from goal state at the same time. When two searches meet in the middle, we have found the solution path.

Properties: Time $O(b^{(d/2)})$; much faster than BFS.

Space complexity: $O(b^{(d/2)})$ → much smaller than BFS, but still requires storing one frontier.

completeness: Yes, if branching factor is finite.

Optimality: yes, if both searches are BFS.

Disadv of Bidirectional:-

① Backward search is not optimal or make sense for "no queen attack another queen"

② still need space as we must store all visited nodes for intersection checks.

Informed Search:- Informed search is a search method that uses extra problem specific knowledge (heuristics) to guide the search towards the goal more efficiently than blind search.

A heuristic is a guess of how close we are to the goal.

h(n) heuristic function is the estimated cost of cheapest path from node n to goal.

If n is the goal h(n) = 0.

The heuristic is not exact but it gives useful estimate.

Greedy Best First Search:-

It is a search method that always expands the node which looks closest to the goal, based only on the heuristic estimate h(n), without caring about path cost so far.

Properties:-

Completeness:- Not always, can get stuck in loops, like DFS.

optimal:- No. chooses nearest looking node, not always shortest path.

Time & Space:- Worst case O(b^m) like DFS.

A heuristic is a smart guess or shortcut to help solve problems faster.

* A heuristic is admissible if it never overestimates the cost to reach the goal.
Guaranteed finding optimal solution.
* An algorithm is complete if it always finds a solution. (if one exists)
* If we have two heuristics, h_1 and h_2 and if h_1 always gives better estimates than h_2 then h_1 dominates h_2.
* An algorithm is optimal if it is admissible, complete, uses the best heuristic.
A* with a consistent heuristic is always optimal.

A* search Algorithm:

It is an informed search algorithm that finds the shortest path from a start node to a goal node using both:

1. cost so far from start node to current node → $g(n)$.
2. Estimate cost to reach goal from current node → $h(n)$

$f(n)$ = total estimated cost of the path going through n.

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

A* search Algorithm solve with 8 puzzle.

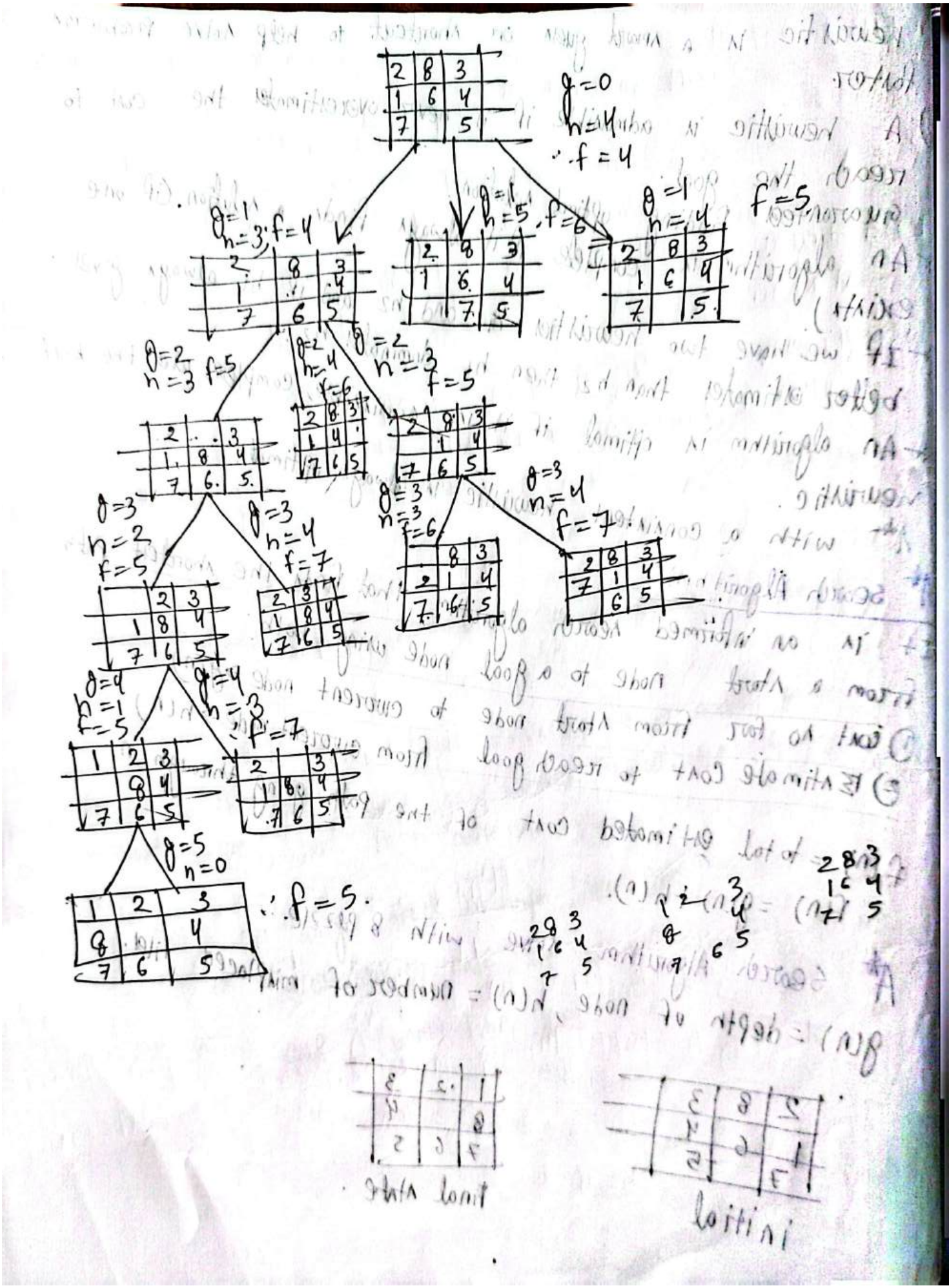
$g(n)$ = depth of node, $h(n)$ = number of misplaced tiles.

2	8	3
1	6	4
7		5

initial

1	2	3
8		4
7	6	5

final state.

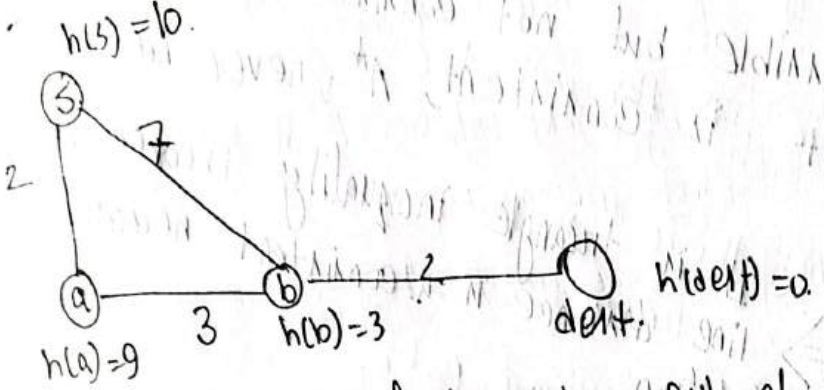

g=0
h=4
∴ f=4
g=1
h=3
f=4
g=1
h=5
f=6
g=1
h=4
f=5
g=2
h=3
f=5
g=2
h=4
f=6
g=2
h=3
f=5
g=3
h=2
f=5
g=3
h=4
f=7
g=3
h=3
f=6
g=3
h=4
f=7
g=4
h=1
f=5
g=4
h=3
f=7
g=5
h=0
∴ f=5.

Optimality of A* algorithm

h(n) > actual cost, optimum solution can be overlooked or not possible.

h(n) = actual cost, Best case, if h(n) approximates actual cost, searching will minimum of node to the goal

h(n) < actual cost [Admissible, Consistent Heuristics]



From node s. we start to node a, Push at priority queue 2+9(h) = 11

From node s we start to node b, Push at priority queue 7+3(h) = 10

So from b we will explore of neighbour b which are destination and (a)

As (a) node is visited we will Pop dest now the cost to dest = 7+2+0 = 9. This is the solution of A* algorithm.

But the optimal solution could be

s → a → b → dest = 2+3+2 = 7.

A* does not provide the optimal solution as heuristic cost is more than actual cost.

A heuristic is consistent if your guess about the distance (or cost) to the goal does not jump unrealistically when you move from one place to another.

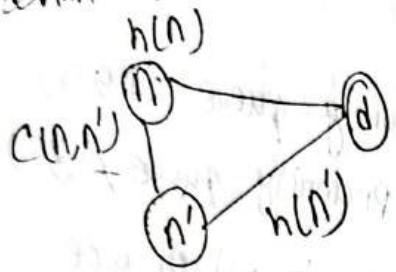
when moving from node n to its neighbour n' the estimated cost from n to the goal is not greater than n'.

$$h(n) \leq c(n,n') + h(n')$$

$c(n,n')$ = actual step cost from n to n'

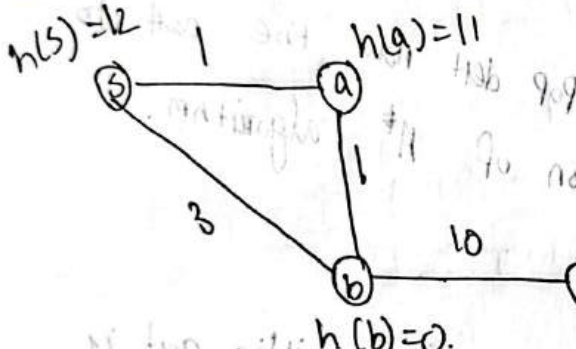
$h(n')$ = estimated cost from n' to goal.

If heuristic is admissible but not consistent A* might need to revisit but if it is consistent, A* never need to revisit it.



⇒ If triangle inequality theorem, straight line distance is a consistent heuristic function.

Check if the A* is consistent Heuristic



	h(n)		c(n,n')		h(n')
s-a	12	≤	1	+	11
s-b	12	≥	3	+	0

(it is not consistent heuristic)

when we pop some node from priority list it goes to close list

All consistent heuristics are admissible.

Here starting node S, has two successor a and b. S push two value into priority queue $a = 1+11 = 12$

$b = 3+0 = 3$

we pop the value b and take it to close list and put a into open list.

Now b has two successor dett and a but a is already visited so $d = 3+10 = 13$

but here optimal solution is $= 1+1+10 = 12$

so for optimal solution heuristic must be consistent.

The tree search version of A^* is optimal if h(n) is admissible.

The graph search version of A^* is optimal if h(n) is consistent.

Proof

if h(n) is consistent, then the values of f(n) along any path are nondecreasing.

$h(n) <= c(n,a,n') + h(n')$ where n' is successor of n. ——①

From A^* algorithm,

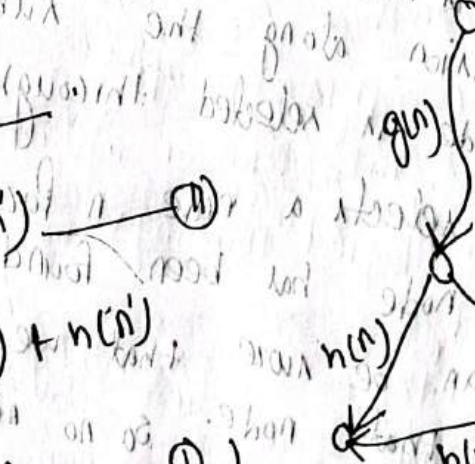
$f(n') = g(n') + h(n')$ ——

$g(n') = g(n) + c(n,a,n')$ ——②

$f(n') = g(n) + c(n,a,n') + h(n')$

$f(n') \geq g(n) + h(n)$ (From ①)

$f(n') \geq f(n)$.



Whenever A* selects a node n for expansion, the optimal path to that node has been found.

Optimal means g-value along the path will be less than the g-value along the sub-optimal path.

Let's select the Node n through the non optimal path.

But this can't happen as there must be a node n' in the optimal path which in the frontier list.

f_1 = f-value of node n along optimal path.

f_2 = f-value of node n along sub optimal path.

f_3 = f-value of node n'

$f_3 <= f_1$, $f_1 < f_2$ $\therefore f_3 < f_2$

As long as there is a node n' which belongs to the optimal path and which is in the frontier list, node n will not be selected for expansion along the sub-optimal path. Thus it cannot happen a node is selected through the sub optimal path.

whenever A* selects a node n for expansion, the optimal path to that node has been found. If A* picks a node to expand, we can be sure that we already have the cheapest path to that node. So no need to revisit node later.

the sequence of nodes expanded by A* using graph-search is in non decreasing order f(n).

The first goal node selected for expansion must be optimal as

d node have $h=0$, so $f(n)=g(n)$. Later goal nodes will only
e more expensive.
If C^* is the cost of the optimal solution, it may expand some
des with $f(n)=C^*$ before reaching the goal.
will always find a solution if step costs are positive and
anching factor b is finite.
only expands nodes with $f(n) \leq C^*$

Ch-4

Genetic Algorithm for 8 queen Problem =>

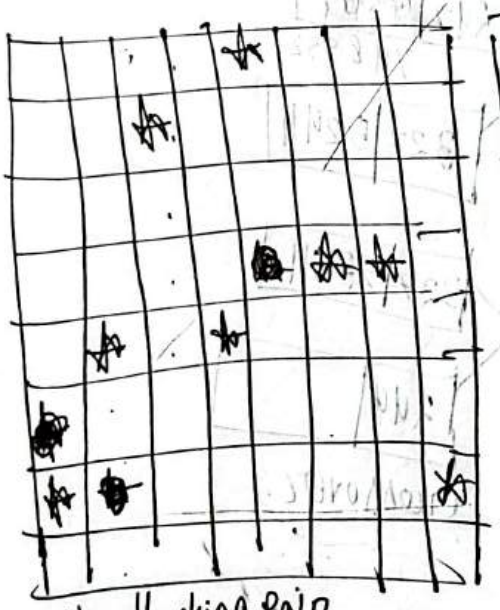
Given 4 initial population →

1st one → 2 4 7 4 8 5 5 2

2nd one → 3 2 7 5 2 4 11 → fitness value 23

3rd one → 2 4 4 1 5 1 2 4 → 11 11 20

4th one → 3 2 5 4 3 2 1 3 → 11 11 11



Non attacking pair

for Q1 → 6 (cause 1 to attack)

for Q2 → 5

for Q3 → 4

Q4 → 4

Q5 → 3

Q6 → 1

Q7 → 1

Q8 → 0

24 → fitness value.

fitness percentage for population 1,

$\frac{24}{(24+23+20+11)} = 31\%$

Population 2 $= \frac{23}{78} = 29\%$

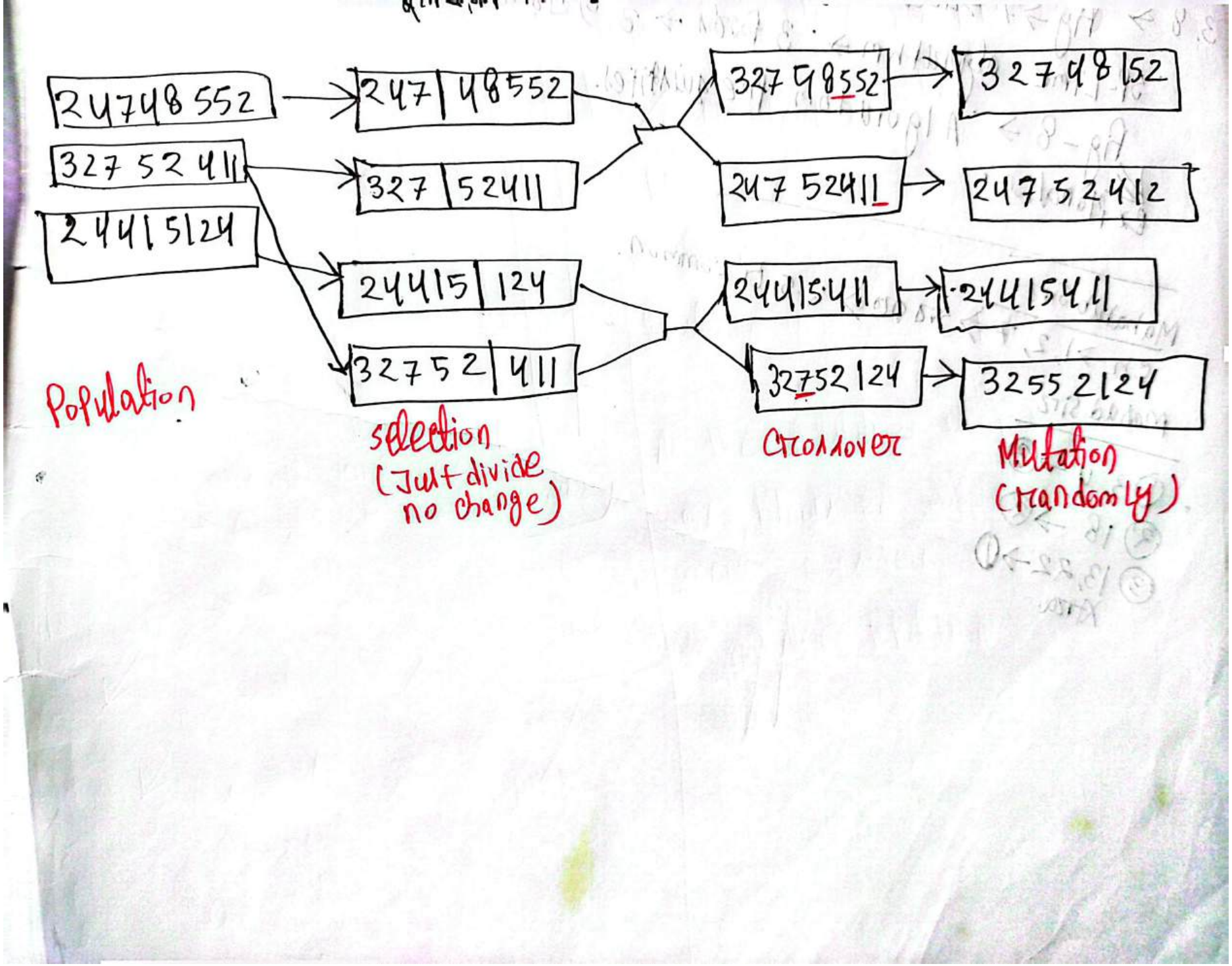
Population 3 $= \frac{20}{78} = 26\%$

Population 4 $= \frac{11}{78} = 14\%$

As the population 4 has less than 20% fitness so we are not gonna consider it,

① initialize population

② select


24748552
32752411
24415124
Population
247|48552
327|52411
24415|124
32752|411
selection
(Just divide
no change)
32748552
24752411
24415411
32752124
Crossover
32748152
24752412
24415411
32552124
Mutation
(randomly)

Ch-4

Local search algorithm is a type of optimization algorithm that is used to find an optimal solution for a particular problem with small search space. Reasons to use local search algorithm ⇒

① large search space
② Continious and discrete optimization.
③ Efficiency ④ Greedy nature ⑤ local optima
⑥ Adaptability ⑦ Online and real time optimization. ⑧ Stochastic nature.

two very advantage ⇒

① little memory ② find reasonable solution in large or infinite state space.

4 Queen ⇒ Colum ⇒ 2-4-1-3 → Goal, state space, state transition, local search

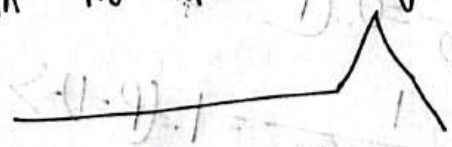
+ local maxima is the higher peak that each of it's neighbour but still has less peak than it's global maxima.

Hill climbing method problem :-

① local maxima ② cannot recover from failure ③ Multiple local maxima
④ stuck on ridge and ~~flat~~ Plateaux.

Ridge :- There are sequence of local maxima making it difficult for algorithm to navigate

Plateaux :- As there is no uphill to go the algorithm often gets lost.



Escape local maxima by allowing some bad moves but gradually decrease their probability.

→ Probability controlled by parameter is called temperature.

→ higher temp allow more bad moves than lower temperature.

Simulated annealing solution is to start by shaking hard and then gradually reduce the intensity of shaking.

If the schedule lowers T slowly enough, the algorithm will find a global minimum with probability approaching 1.

Algorithm:

Function Simulated Annealing (Problem, schedule)
return a solution state.

input: Problem, a problem schedule, a mapping from time to temperature

make node → current

for t = 1 to ∞ do

If T=0 then return current.

next → a randomly selected successor of current

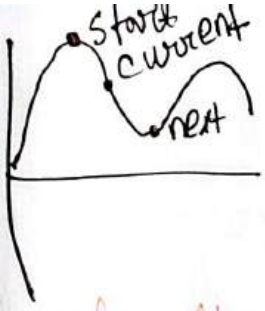
ΔE → next values − current values

if ΔE > 0 then next → current

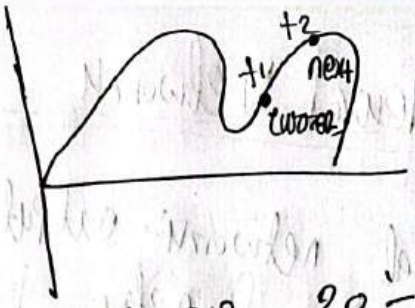
else current → next only with probability $e^{\Delta E/T}$.

$T=0$; $e^{-\Delta E/T} = \frac{1}{e^{\Delta E/T}} = \frac{1}{\infty} = 0$. $(P=0) \Rightarrow$ Bad move

$T=1$; $e^{\Delta E/T} = \frac{1}{e^{\Delta E/\infty}} = \frac{1}{e^{\Delta E/\infty}} = 1$. $(P=1) \Rightarrow$ Temp is high.



current = t_1.
next = t_2
$t_2 - t_1$ = annealing.



$t_2 - t_1$ = random access.

Genetic Algorithm =>

24 - 23 - 20 - 11

1. Initialize population.
2. selection according to fitness
3. crossover between selected chromosomes.
4. perform mutation
5. Repeat cycle till the condition of stop is free.

Chapter-13

Agents often need to act under uncertainity. uncertainity come from partial observability, Nondeterminism.

Agent combines probabilities (how lively something will happen), utilities (how good or bad each possible outcome is for the agent) and this combination is called decision theory.

utility = usefulness / value of an outcome for the agent.

decision theory says a rational agent should always pick the action that gives the highest average utility.

Probability of outcome1
utility of outcome1

Expected utility = (Probability of outcome1 x utility of outcome1) + (Probability of outcome2 x utility of outcome2) + ...

A decision theoretic agent that selects rational actions =>

1) Belief state → what the agent thinks the world looks like
2) update beliefs → After each action or new perception
3) calculate probabilities → It is outcomes for possible actions.
4) select the action with highest expected utility.
   given probabilities of outcomes and utility information

return action.

The set of all possible worlds is called the sample space (Ω)

Each possible world is one outcome (ω)

Sets of world is called event

Probability of an event = Sum of probabilities of the worlds inside it

Event ⇒ total 11 from 2 dice,

two worlds (5,6), (6,5)

$P(\text{total}=11) = 1/36 + 1/36 = 1/18$

Unconditional / Prior Probability → Belief before extra information.

Conditional / Posterior Probability → Belief after seeing some evidence

"|" means "given that"

P(A|B) ⇒ The probability that event A happens given that B has already happened

Conditional Probability	Logical implication: B→A
Quantitative (number between 0 and 1)	qualitative (True / False)
Allows uncertainity	centainity if B, then A guarantee.

Conditional Probability

$$P(a|b) = \frac{P(a \wedge b)}{P(b)} \quad (\text{if } P(b) > 0)$$

$a \wedge b$ = both a and b happen together.

P(a|b) = Probability of a restricted to the universe where b already happened.

For both dice same (event 1)

dice1 = 5 (event 2)

P (doubles ∧ dice1=5) =p

both dice same and dice1 =5 → the only possibility is (5,5)

Total outcome =36.

P (double ∧ dice1=5) = $\frac{1}{36}$.

if dice1 = 5 and dice 2 can be anything,

P(dice1 = 5) = $\frac{6}{36}$ = $\frac{1}{6}$.

Now applying conditional formula,

P(double | dice1=5) = $\frac{1/36}{1/6}$ = $\frac{1}{36} \times \frac{6}{1}$ = $\frac{1}{6}$.

Product rule,

$P(a \wedge b) = P(a|b) \cdot P(b)$

To find Probability of a and b, find Probability of b, the multiply by Probability of a given b.

Proposition is a statement about variable ~~assig~~ assignments.

Combining Proposition: where Proposition can be combined with logical Connectivities.

Full Joint distribution is the complete probability table over all variables. Total entries = 2x2x4=16

cavity ∈ (Yes, No) → 2 value

toothache ∈ (yes, No) → 2 value.

weather ∈ (sunny, Rain, cloud, snow) → 4 value,

Inclusion - Exclusion Principle disjunction,

$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$.

Normalization Axiom,

$$\sum_{w \in \Omega} P(w) = 1$$

The total probability over all possible outcomes is 1. So something must happen the sum of all possibilities is 100%.

Additivity Axiom,

$$P(a) = \sum_{w \in a} P(w)$$

The probability of an event a is the sum of probabilities of all outcomes (worlds) where a is true.

If an event can happen in multiple ways, you add up the probabilities of all those ways.

Full joint distribution is a table of probabilities for all combinations of all variables

Marginalization,

often we are interested in one variable and want to ignore the others. This is done by marginalization also called summing out.

$$P(\text{cavity}) = \sum_{\text{Toothache, catch}} P(\text{Cavity, Toothache, Catch})$$

$$P(Y) = \sum_{z \in Z} P(Y, z)$$

Y = variable of interest

z = all other variables

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

$$P(X|e) = \frac{P(X,e)}{P(e)}$$

X = query variables, Y = other unobserved variables

e = observed evidence

$$P(\text{cavity} \mid \text{toothache}) = \frac{P(\text{Cavity} \wedge \text{toothache})}{P(\text{toothache})}$$

$P(\text{toothache}) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$

$P(\text{Cavity} \wedge \text{toothache}) = 0.108 + 0.012 = 0.12$

$P(\text{Cavity} \mid \text{toothache}) = \frac{0.12}{0.2} = 0.6$

result and non result

$P(\neg\text{Cavity} \mid \text{Toothache}) = 0.4$

∴ So the total sum $= 0.6 + 0.4 = 1$

Normalization Constant (α) $\quad \alpha = \frac{1}{P(e)}$

we skip computing α or $P(e)$. But we compute unnormalized probabilities for all values of X and then divide each by sum to normalize.

$\text{Cavity} = 0.108 + 0.012 = 0.12$

$\neg\text{Cavity} = 0.016 + 0.064 = 0.08$

∴ $\text{sum} = 0.12 + 0.08 = 0.2$

Normalize, $\frac{0.12}{0.2} = 0.6$

$\frac{0.08}{0.2} = 0.4$

$\alpha = \frac{1}{\text{Sum of unnormalized Probabilities}}$ = Normalization Constant.

From this table compute P(Cavity | toothache) using general inference procedure. steps including, marginalization and normalization

$P(\text{Cavity} \mid \text{toothache}) = \alpha \sum_{y \in \{\text{catch}, \neg\text{catch}\}} P(\text{Cavity}, \text{Toothache}, y)$

when toothache yes,

Cavity = 0.108 + 0.012 = 0.12

¬Cavity = 0.016 + 0.064 = 0.08.

These are unnormalized probability.

Total Probability of Toothache = yes = 0.12 + 0.08 = 0.2.

Normalize the probability,

probability of cavity = yes = $\frac{0.12}{0.2} = 0.6$

Probability of = No = $\frac{0.08}{0.2} = 0.4$

∴ Normalization = 0.6 + 0.4 = 1.

In full joint distribution,

The summation is over all possible combinations of Y.

X, E, Y together cover all variables in the domain.

P(X, e, Y) comes directly from full joint distribution

For n Boolean variables the full joint distribution table has 2^n entities. So $O(2^n)$ time for n→∞ is impossible and not scalable.

$P(A,B) = P(A|B) \cdot P(B)$

$P(\text{Toothache}, \text{Catch}, \text{Cloudy}, \text{Cavity}) = P(\text{Cloudy} | \text{Toothache}, \text{Catch}, \text{Cavity}) \cdot P(\text{Toothache}, \text{Catch}, \text{Cavity})$

This is Product rule

$$P(\text{Cloudy} | \text{Toothache}, \text{Catch}, \text{Cavity}) = \frac{P(\text{Cloudy and Toothache and Catch and Cavity})}{P(\text{Toothache and Catch and Cavity})}$$

$$= \frac{P(\text{Cloudy}) \cdot P(\text{Toothache and Catch and Cavity})}{P(\text{Toothache and Catch and Cavity})}$$

$$= P(\text{Cloudy})$$

This is absolute (marginal) independence.

For two event X and Y.

$P(X|Y) = P(X)$, $P(X|Y) = P(Y)$, $P(X,Y) = P(X)\,P(Y)$

Independence reduce the size of joint distribution.

Product rule and Bayes Rule

$$P(a \wedge b) = P(a|b)\,P(b) = P(b|a)\,P(a)$$

$$\therefore P(b|a) = \frac{P(a|b)\,P(b)}{P(a)}$$

$P(a|b)$ = the probability of observing a if b is true.

$P(b)$ = Prior probability of b

$P(a)$ = Probability of a (normalizing factor)

with additional evidence e,

$$P(Y|x,e) = \frac{P(x|y,e)\,P(Y|e)}{P(x|e)}$$

We know,
$P(\text{symptom}|\text{disease})$ ⇒ disease for symptom.
But we need to find $P(\text{disease}|\text{symptom})$

$P(s|m) = 0.7$, $P(m) = \frac{1}{50,000}$, $P(s) = 0.01$.

$$P(m|s) = \frac{0.7 \times \frac{1}{50000}}{0.01}$$

$$= 0.0014.$$

$\alpha = \frac{1}{P(x)}$ using normalization constant α,

$$\alpha = \frac{1}{P(s|m)\,P(m) + P(s|\neg m)\,P(\neg m)}$$

unnormalize: $P(s|m)\,P(m)$, $P(s|\neg m)\,P(\neg m)$

$P(M|s) = \alpha\, P(s|m)\,P(m)$, $P(\neg M|s) = P(s|\neg m)\,P(\neg m)$

$P(m|s) + P(\neg m|s) = 1$.

Bayes Rule in Robust

direct diagnostic probabilities $P(\text{disease}|\text{symptom})$ can be fragile
Causal probabilities are stable. $P(\text{symptoms}|\text{disease})$ are stable
It is independent of how frequent the disease is.
Bayes Rule automatically updates the probability when the prior $P(\text{disease})$ changes.

Using Bayes Rule combining evidence,

P(cavity | toothache ∧ catch) ⇒ What is the probability that a Patient has a cavity if they have toothache and dentists probe catch in tooth.

$$P(\text{cavity} \mid \text{toothache} \wedge \text{catch}) = \alpha\, P(\text{toothache} \wedge \text{catch} \mid \text{cavity})\, P(\text{cavity}) \quad \text{—— (1)}$$

By having conditional independence by conditioning in cavity the evidence variable becomes independent so,

$$P(\text{toothache} \wedge \text{catch} \mid \text{cavity}) = P(\text{toothache} \mid \text{cavity})\, P(\text{catch} \mid \text{cavity}) \quad \text{——→ (2)}$$

So Putting this in equation 1,

$$P(\text{cavity} \mid \text{toothach} \wedge \text{catch}) = \alpha\, P(\text{toothach} \mid \text{cavity})\, P(\text{catch} \mid \text{cavity})\, P(\text{cavity})$$

For n evidence variables,

$$P(C \mid E_1 \wedge E_2 \wedge E_3) = \alpha\, P(C) \prod_{i=1}^{n} P(E_i \mid C)$$

here E = effect.

C = cause.

* Bayes rule provides a way to update the probability of a hypothesis based on new evidence.
* Naive Bayes is a simple model that predicts probability based on past data, assuming all features are independent.
* Random variable is the quantity whose value is not certain and can vary due to chance.

Design a naive Bayes model Bayesian classifier:

Bayes rule is a method which update the Probability of a hypothesis based on new evidence,

$$P(B|A,e) = \frac{P(A|B,e) \cdot P(B,e)}{P(A,e)}$$

For the cavity example:-

	toothache		¬toothache	
	Catch	¬Catch	catch	¬Catch
Cavity	0.108	0.012	0.072	0.08
¬Cavity	0.016	0.064	0.144	0.576

$P(\text{Cavity} \mid \text{toothache} \wedge \text{catch}) = \alpha\, P(\text{toothache} \wedge \text{Catch} \mid \text{cavity}) \cdot P(\text{cavity})$ —— ①

Each variable cavity and toothache directly caused by cavity

$P(\text{toothache} \wedge \text{Catch} \mid \text{cavity}) = P(\text{toothach} \mid \text{cavity})\, P(\text{Catch} \mid \text{cavity})$ —— ②

This gives the information of catch and toothache given by cavity.

From ① and ②

$$P(\text{Cavity} \mid \text{toothache} \wedge \text{Catch}) = \alpha\, P(\text{tootache} \mid \text{cavity}) \cdot P(\text{Catch} \mid \text{cavity}) \cdot P(\text{cavity}) \quad \text{—— ③}$$

The conditional independence refers the fact that two variables are independent based on the other event. For two independ variable X, Y for a third variable z,

$$P(X, Y \mid Z) = P(X \mid Z) \cdot P(Y \mid Z)$$

For the dentist example conditional independence of toothache, catch, cavity

$$P(\text{toothache}, \text{catch} \mid \text{cavity}) = P(\text{toothache} \mid \text{cavity}) \cdot P(\text{catch} \mid \text{cavity})$$

We can derive decomposition:

$P(\text{Toothach}, \text{catch}, \text{cavity})$.

$= P(\text{toothache}, \text{catch} \mid \text{cavity}) \cdot P(\text{cavity})$ [Product rule]

$= P(\text{toothache} \mid \text{cavity}) \cdot P(\text{catch} \mid \text{cavity}) \cdot P(\text{cavity})$

The Patterns of this example is a single cause directly influences many event which all are conditional independent,

The full joint distribution can be written as →

$P(\text{Cause}, \text{Effect}_1, \text{Effect}_n) = P(\text{Cause}) \prod P(\text{Effect}_i \mid \text{Cause})$

This Probability distribution is called naive Bayes' model.

$P(\text{Cause}, \text{Effect}) = P(\text{Cause}) \prod P(\text{Effect}_i \mid \text{Cause})$

The chance of all things happening together.

= chance of the cause × chance of the effects given the cause.

Naive Bayes: The mathematical method assumes all effects are independent from each other when the cause is unknown.

Marginal Probability: Marginal Probability means the Probability of one event happening without caring about any other related event.

If we have data about cavity, toothache, and catch then $P(\text{cavity})$ is marginal probability cause the cavity can be happen no matter someone have toothache or not.

$$P(X) = \sum_Y P(X, Y)$$

We sum other variable to get only X's Probability.

Conditional Probability:- Conditional Probability tells us the chance of one event given that another event is already known to happen. It shows how one event depend on other.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$\text{Probability A given to B} = \frac{\text{Both A and B happening}}{\text{chance of B.}}$$

Conditional independence:- Two event are conditionally independent if knowing one of them does not change the probability of other, once we know the main cause.

When toothache and catch are conditionally independent given cavity,

$$P(\text{toothache}, \text{catch} | \text{cavity}) = P(\text{toothach} | \text{cavity}) \cdot P(\text{catch} | \text{cavity})$$

Chapter-18

An agent to learn => means it improves it's performance in the future after observing / experiencing something new.

- Trivial learning → memorizing a phone number
- Profound learning → Einstein developing new theory.

An agent learn instead of being fully programmed =>

1. cannot anticipate everything → World has too many situations.
2. World changes over time.
3. some programs are hard to program directly.

Components an agent can learn ⇒
condition ⇒ Action mapping : learning when to break.
Recognization ⇒ identifying buses
Effects of actions ⇒ breaking distance on wet road.
Utility ⇒ smooth ride = more tips.
Action value ⇒ knowing which actions are better.
Goals ⇒ safe and fast rides.

Types of learning ⇒
Inductive learning ⇒ From examples to general rule!
Deductive learning ⇒ From general rule to specific case!
All humans are mortal so Chandra is mortal.

Feedback types ⇒ Learning types.
Supervised learning ⇒ Learns from input-output pairs. It is more likely to label.
Unsupervised learning ⇒ no labels, finds pattern. The agents task is clustering.
Reinforcement learning ⇒ Learns by rewards / punishments.
Semisupervised Learning ⇒ some data labeled, most data unlabeled.

Supervised learning ⇒
Supervised learning is a machine learning task where an agent learns from labeled input-output pairs to approximate an unknown true function. The goal is to predict the correct output for unseen inputs.
input = photo of animal, output = cat or Dog.

Training set ⇒ use to learn pattern.

Test set ⇒ use to check performance on new data.

Generalization is the ability of the hypothesis to work well on unseen data, not just on training examples.
It is like understanding the topic rather than memorizing the answer.

Classification ⇒ output is label - weather is sunny

Regression ⇒ output is a real number - Temperature is 32.5°C

Hypothesis space ⇒ The set of all possible function we consider.

A hypothesis is consistent if it fits all training data exactly.

Occham's Razor : prefer the simplest hypothesis that explains the data.

Complex models when hypothesis fit training data perfectly but may fail on unseen data (overfitting)

simple models may not fit all training data, but usually predict better (generalize well)

Realizable Hypothesis space A problem is realizable if the true function exists inside the hypothesis space H.

Using all possible hypothesis causes ⇒ too much complexity and computation.

ayesian Supervised learning

t is a method where we assign probabilities to each possible ypothesis instead of just saying possible or impossible.

$h^* = \arg\max P(h|data)$

t the hypothesis h^* that maximizes the probability after seeing he data.

ule, $P(h|data) \propto P(data|h) \cdot P(h)$

P(h) => Prior Probability, what you believe about the hypothesis before seeing the data.

P(data|h) => Likelihood, how well this hypothesis fits the data you have.

P(h|data) => Posterior Probability, the updated probability after seeing the data.

= Simple hypothesis have higher prior probability.

Decision Tree learning:-

Decision Tree learning is a machine learning method where an agent takes input features and produces a decision (output), usually in Boolean form. It works by asking a sequence of questions about attributes and following branches until reaching a leaf node with a final decision.

: how a decision tree works =>

internal nodes => test on attributes (Pardon = Full?)

Branches => Represent possible values of the attributes.

Leaf nodes => final decision (Positive / negative)

Decision trees may not use all attributes. only relevant ones affect the final decision.

The tree starts at the root node and moves down by following branches until a leaf node is reached.

Expressiveness of Decision Tree:-

It refers to how powerful it is and what types of logical problems it can represent.

A decision tree represents Boolean logic rules in disjunctive normal form (DNF) an OR of ANDs.

- Boolean decision trees are equivalent to propositional logic rules.
- Each path from root to leaf = conditions joined by AND.
- Multiple paths = OR of their AND conditions.

Number of possible functions:-

For n attributes (True / False)

Truth table rows = 2^n.

ecision tree

weather	Humidity	wind	play
sunny	High	weak	No
sunny	Normal	weak	No
sunny	Normal	strong	No
cloudy	High	weak	No
cloudy	High	strong	No
cloudy	Normal	strong	Yes
cloudy	Normal	weak	Yes
Rainy	High	weak	Yes
Rainy	Normal	strong	Yes
Rainy	Normal	weak	Yes

steps:

① At first choose a target attribute

② calculate Information gain of target attribute

③ calculate Entropy of other attributes using the following formula:

$$E(A) = \sum_{i=1}^{N} \frac{P_i+N_i}{P+N} \times I(P_i N_i)$$

④ subtract E(A) from IG of each attribute to find out Gain

$$IG = \frac{-P}{P+N} \log_2 \frac{P}{P+N} - \frac{N}{P+N} \log_2 \left(\frac{N}{P+N}\right)$$

For Play information gain,

$$IG = \frac{-P}{P+N} \log_2 \frac{P}{P+N} - \frac{N}{P+N} \log_2 \frac{N}{P+N}$$

Yes = 5, No = 5

$$= \frac{-5}{5+5} \log_2 \frac{5}{5+5} - \frac{5}{5+5} \log_2 \frac{5}{5+5}$$

$$= 1$$

For weather,

	No	Yes
sunny	3	0
cloudy	2	2
Rainy	0	3

$$I.G(\text{sunny}) = \frac{-3}{3} \log_2\left(\frac{3}{3}\right) - 0/3 \log_2(0/3) = 0$$

$$\text{Entropy(sunny)} = I.G(\text{sunny}) \times \text{Probability(sunny)}$$

$$= 0 \times 3/10 = 0.$$

I.G (Cloudy) $= \frac{-2}{4} \log_2(2/4) - 2/4 \log_2(2/4) = 1$

Entropy (Cloudy) $= 1 \times 4/10 = 0.4$

I.G (Rainy) $= \frac{-0}{3} \log_2(0/3) - 3/3 \log_2(3/3) = 0$

Entropy (Rainy) $= 0 \times$ Probability (Rainy) $= 0 \times 3/10 = 0$

Entropy (weather) $=$ E(Sunny) + E(Rainy) + E(Cloudy)

$= 0 + 0.4 + 0 = 0.4$

∴ Information gain $= 1 - 0.4 = 0.6$

For humidity,

	No	Yes
High	3	1
Normal	2	4

I.G.(High) $= -\frac{3}{4} \log_2 3/4 - 1/4 \log 1/4 = 0.8113$

∴ Entropy (High) $= 0.8113 \times \frac{4}{10} = 0.325$

IG.(Normal) $= \frac{-2}{6} \log_2(2/6) - 4/6 \log_2 \frac{4}{6}$

$= 0.9183$.

∴ Entropy (Normal) $= 0.9183 \times 6/10 = 0.551$

Entropy (humidity) $=$ E(High) + E(Normal)

$= 0.325 + 0.551$

IG (humidity) $= 1 - 0.876 = 0.124$

For wind,

	No	Yes
weak	3	3
strong	2	2

$I.G(weak) = \frac{-3}{6}\log_2(3/6) - 3/6 \log_2(3/6) = 1$

$Entropy = 1 \times 6/10 = 0.6$

$I.G(strong) = \frac{-2}{4}\log_2(2/4) - 2/4 \log_2(2/4) = 1$

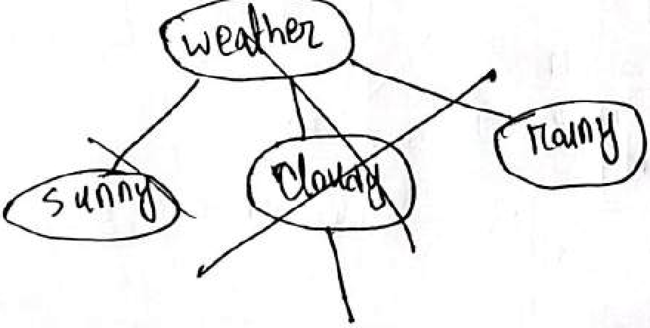
$Entropy(strong) = 1 \times 4/10 = 0.4$

$Entropy(wind) = 0.6 + 0.4 = 1$

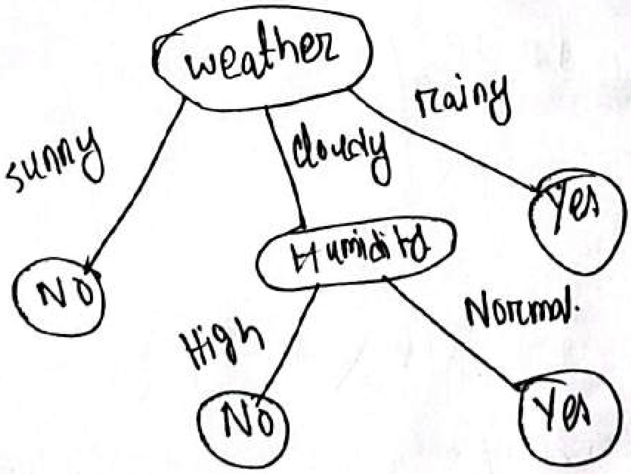
$\therefore I.G(wind) = 1 - 1 = 0$

Decision tree,

Target attribute বাদে যার gain সবচেয়ে বেশি তাকে root node বানাতে হবে।



weather এর sunny আর rainy তে mix নেই result (yes/No) এই তাই ওইখানে node use হয় নাই। একারণেই cloudy তে Yes/no mix তাই ওকে highest IG র node বানাবো।



Ch-22 → Natural Lan

*Grammar means the set of rules that tells how to form valid sentence.
*Semantic means the meaning of sentence.
*Natural language is huge, full of changing always ambiguous

N-gram Model:- An N-gram model is a type of probabilistic language model that predicts the next word of sentence based on previous (N-1) words.

Unigram:- use 0 previous words; predicts words based only on word frequency.

Bigram:- use 1 previous words; predicts next word using one before it.

Trigram:- use 2 previous words.

$$P(w_1, w_2) = \prod_{i=1}^{n} P(w_i | w_{i-2}, w_{i-1})$$

Corpus:- It is a large collection of text we for learning or analyzing a language.

N-language Application:-

(1) Language identification:- we build trigram model for each language using corpus. for the unknown text we check which language gives the highest probability.

$$l = \text{argmax} [P(l) \times P(C_{1:N} | l)]$$

$P(l)$ = Prior probability of language means how often the language we use.

$P(C_{1:N}|l)$ ⇒ Probability of the sequence of character from 1 to N given that language.

② Spelling Correction.

③ Genre classification → detects type of text.

④ Name-Entity recognition → which is what.

Smoothing in N-gram model:-

smoothing means giving a small non zero probability to words or character sequences that never appear in training data. So the model does not fail for new or unseen data.

① laplace smoothing ⇒ even if something never happened, assume it could happen once.

$$P(word) = \frac{n+1}{N+V}$$

N = total word, V = vocabulary size. If ht never appear among 1000 sample

$P("ht") = \frac{0+1}{1000+2} = \frac{1}{1002} \neq 0$; it is easy to understand but not very accurate.

② Backoff model ⇒ use a smaller model if detailed data is not available

If higher order N-gram is missing we back off to simpler model.

If we don't have data for

'dog bark loudly' → trigram

we use 'bark loudly' → bigram

but if we don't even have that we use

'loudly' → unigram

~~***~~

Linear interpolation smoothing:- Combines all models → trigram, bigram and unigram together using weights (λ)

$$P(C_i | C_{i-2}, C_{i-1}) = \lambda_3 P(C_i | C_{i-2}, C_{i-1}) + \lambda_2 P(C_i | C_{i-1}) + \lambda_1 P(C_i)$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

λ_3 = weight of trigram
λ_2 = " " bigram
λ_1 = " " unigram.

if trigram are frequent → give more weight to λ_3.
if data is rare → depend more on λ_2 and λ_1.

It a weighed average of all models, giving smoother and accurate result

Model Evaluation:-

To check which model (unigram, bigram, trigram) is better we need cross-validation. We divide dataset into two parts →

① Training Corpus
② Validation Corpus.

So we train the model with one part of data and test it with the unseen part of data - it prevents the overfitting.

"For the first word we use unigram model"

There are two evaluation type:-

① Task specific → Test on real application task.
② Task independent → test by seeing how much probability the model gives to new text.

Perplexity: Perplexity measures how well a language model predicts a sample of text.

Low perplexity means the model is less confused so Good model! When we multiply many small probability for each word or character the number becomes extremely tiny that computer can't store it.

$$\text{Perplexity}(C_{1:N}) = P(C_{1:N})^{-1/N}$$

Perplexity is the inverse of the probability of whole text.

$C_{1:N}$ = sequence of N characters.

$P(C_{1:N})$ = Probability of that model assign to the sequence

N = total number of token or words.

$P(\text{the}) = 0.2$

$P(\text{dog}|\text{the}) = 0.4$

$P(\text{barks}|\text{the dog}) = 0.5$

$P(C_{1:3}) = 0.2 \times 0.4 \times 0.5 = 0.04$

$\text{Perplexity} = (0.04)^{-1/3} = 2.92$ so good model

The perplexity sequence of model:

Unigram > Bigram > trigram

Better model trigram.

Text classification: Text classification means automatically deciding which category or class a piece of text belongs to using example learned from data.

① SPAM → use marketing words or use strange symbols

② HAM.

Computer learn using two approach:-

① Language model approach based on Naive Bayes idea'
Build one language model for each class (Spam / Ham).

steps ⇒

ⓐ Train model on all spam message ⇒ P(message | spam)

ⓑ Train another model on all ham message ⇒ P(message / Ham)

ⓒ For new message check which model gives higher probability.

P(C | message) = P(message | C) × P(C)

C = class.

P(message | C) = how likely this model fits class.

P(C) = how common the class is.

② Machine learning Approach ⇒

Represent the text as a list of words and their counts.

Each words become feature ⇒ number of times it appears.

The model ignore word order ⇒ so it is called Bag of words.

we can also use informative feature rather than all feature cause every feature is not important.

We use algorithm ⇒ Naive Bayes, Logistic regression, decision trees, KNN ⇒ K-nearest neighbours.

Classification by data Compression :-

Here we classify text using compression algorithm.

① Combine all spam message and Compress them: (Spam.gz)

② Combine all ham message and Compress them (ham.gz)

③ Take new email and append it to each file.

④ See which file's compression size increase less.
if the new text compress better with spam → classify as spam
n " " not " " " " ham ⇒ " as ham

Information Retrival:

Information retrival means finding useful or relevant documents based on what user wants → just like how google searches the web!
main parts of IR system.

① Corpus document → the entire collection of text or files, where the system searches.
② Query language → The way user write their search requests - [AI, book] [AI and Book]
③ Result set → The list of documents found relevant to query. List of matching pages.
④ Presentation → How results are shown → ranked list, cards, maps etc.

* Early IR were boolean form either true or false. But it was not useful as Boolean logic is confusing for normal user. only true or false no ranking. It is too strict as small changes can give zero results.

modern IR use BM25 (Best matching 25) algorithm.
BM25 gives each document a numeric score showing how relevant it is to the query. High score → more relevant document
It use 3 main factor.
① Term frequency (TF) → how many times a word appear in the document.
② Inverse document frequency → how rare a word is across all documents

(5) Document length → short, focused documents score higher than very long one.

BM25 = f(TF, IDF, doc length)

Index = the word list or dictionary to find document quickly.

hit list = Hit list is the list of all documents where a specific word appears.

Evaluation of Information retrival system:

After building a system we must measure how good it is. We test the system using unown data and count how many results are relevant or irrelevant.

	result set	Not in result set
relevant	30	20
Not relevant	10	40

(1) Precision: Precision measures how many of the found documents are actually relevant.

$$P = \frac{\text{Relevant in result}}{\text{total in result}} = \frac{3}{30+10} = 0.75$$

75% of retrived documents were useful. Precision check quality of search result.

(2) Recall: measure how many of all relevent documents the system managed to find.

$$R = \frac{\text{Relevant in result}}{\text{Total relevant docs}} = \frac{30}{30+20} = 0.6$$

Recall says the coverage of search result.

Trade off Between Precision and recall:

show more result	show few result
Recall ↑	Recall ↓
Precision ↓	Precision ↑

we can't maximize both at same time we need trade off!
So we use F1 score formula.

$$F_1 = \frac{2PR}{P+R}$$

Higher F1 → better overall performance.

In web search, as web are huge we can't use Recall we only use precision.

P@10 means precision in top 10 search result. If 7 of the top 10 Page were relevant P@10 = 0.7.

Pagerank Algorithm → It is an algorithm created by Larry Page to rank web page by importance. Before Page rank we used Term frequency. But a random blog can use the searched term without having any meaning; Pagerank solved this issue by using 'links' as vote for importance.

Pagerank works as →

1. The web is a network of pages connected by links.
2. When one page links to another, it's like giving a vote.
3. Pages with many votes are important, but votes from important pages count more.

recursive logic →

Pagerank is the self-referencing. Popular pages increases the page rank to which they are link to, Google recalculates

the score again and again until they stabilize or Convergence.

$$PR(P) = \frac{1-d}{N} + d \sum \frac{PR(in_i)}{C(in_i)}$$

$PR(P)$ = Page rank score of Page P.

N = Total number of Pages.

d = Damping factor (usually 0.85)

$\frac{1-d}{N}$ ensures that every page gets at least a small score.

in_i = Pages that link into Pages P.

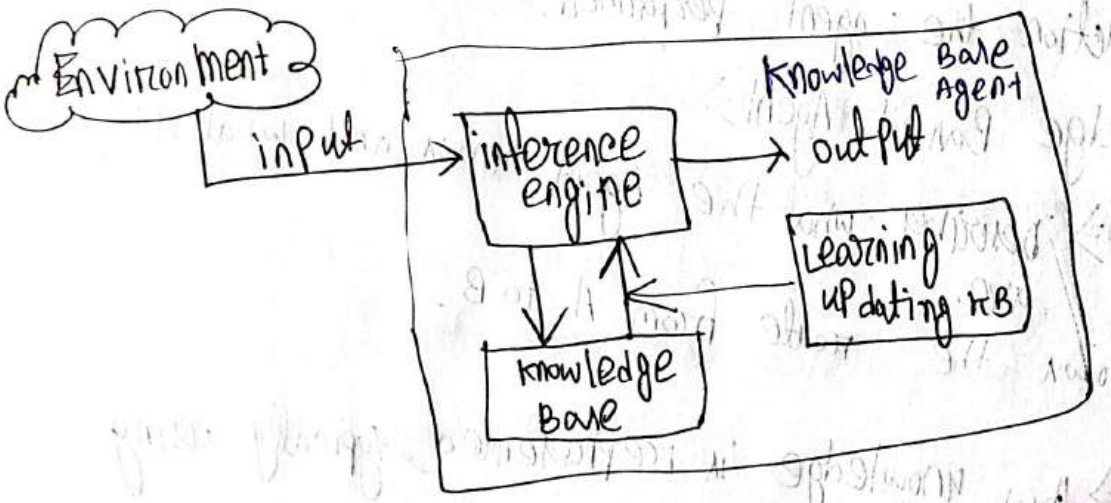
$C(in_i)$ = Number of outgoing links from Page in_i

Propositional Logic

Knowledge base agent are those who have the capability of maintaining an internal state of knowledge, reason over their knowledge, update their know ledge after observations and take actions. These agents can represent the world with some formal representation.

It composed of two main parts ⇒

① Knowledge base ② Inference system.



Knowledge Base is a collection of sentence or real world fact. Inference system generates new facts so that an agent can update the KB. Two rules ⇒

① forward chaining

② Backward chaining.

Operations perform by KBA ⇒
TELL, ASK, Perform.
The knowledge base agent takes percept as input and return action as output.

KBA Concepts ⇒

(I) MAKE PERCEPT SENTENCE → Generates a sentence stating what percept the agent received at a specific time.

(II) MAKE ACTION QUERY → Generates a question asking what action the agent should take at the current time.

(III) MAKE-ACTION-SENTENCE → Generates a sentence that asserts which action the agent performed.

LEVELS of knowledge Based Agent ⇒

(1) Knowledge Level ⇒ Describes what the Agent knows and what its goals are.

A taxi agent knows the route from A to B.

(2) Logical Level ⇒ How knowledge is represented, typically using logic sentences.

Taxi agent knowledge of the route is encoded logically.

(3) Implementation level ⇒ Describes how logic and knowledge are physically implemented and how actions are executed.

Taxi agent follows the logic to reach the destination.

design approaches for KBA =>

① Declarative Approach =>
start with an empty knowledge Base and adds facts sentences manually. Focus on what the agent should know.

② Procedural Approach =>
Encode behaviour directly in the program and focus on how the agent should act.

A good real world agent start with declarative knowlege and optimize using procedural techniques.

Knowledge representation and reasoning

focus on how AI agents think and how that thinking enables intelligent behavior. It is not just storing data, it helps AI systems learn and behave intelligently like humans.

Things need to be represented in AI =>

facts, objects, events, performance, meta knowlege (knowledge about what is known), knowledge Base.

knowledge in AI is awareness gained through experience facts, data, situations.

Types of knowledge in AI =>

① Declarative knowledge
what is known about something. It is expressed in declarative sentence and simple than procedural knowledge.

(II) Procedural knowledge.

Knowing how to do something. it is imperative knowledge and apply directly to task. and includes rules, strategies, procedures, algorithm and it varies with the type of task.

(III) Meta knowledge.

knowing about other knowledge. An knowing that approach to Problem solving is based on procedural steps.

(IV) Heuristic knowledge

Expert level knowledge based on experience. It is useful for making decision under uncertainity.

(V) structural knowledge.

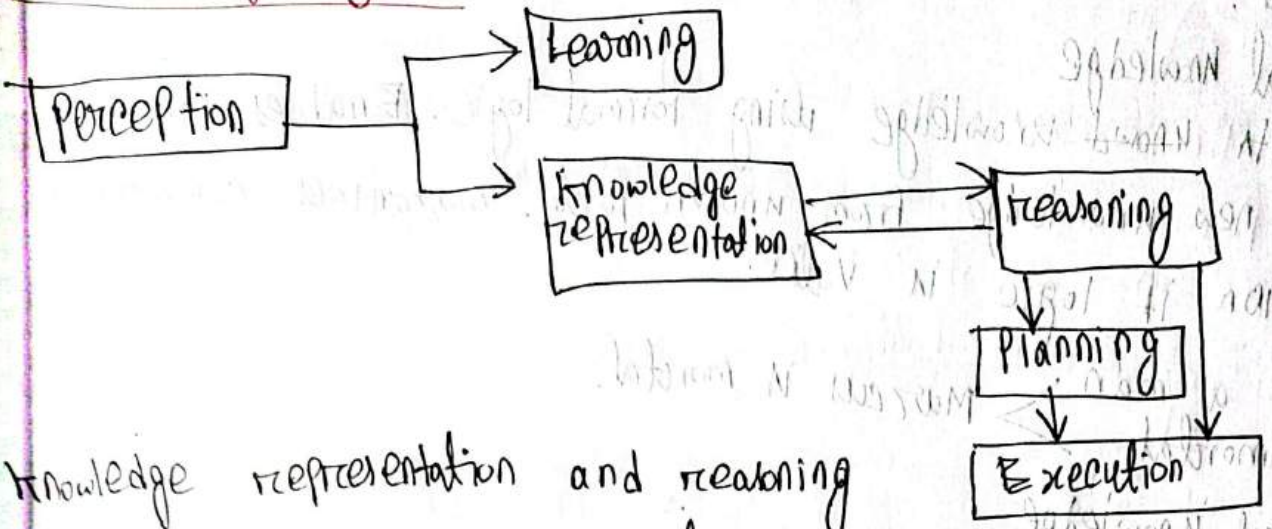
It is knowledge of relationships between concepts. It helps in problem solving and describes how concepts are related.

relation between knowledge and intelligence

knowledge is essential for demonstrating intelligent behavior. An AI agent can only act appropriately if this has relevant knowledge or experience.

An intelligent agent. sense the environment, use knowledge to decide actions, acts accordingly and without knowledge the agent cannot behave intelligently.

AI Knowledge cycle:



Knowledge representation and reasoning are involved in showing the inteligence of machine like human. They are independent but also coupled together.

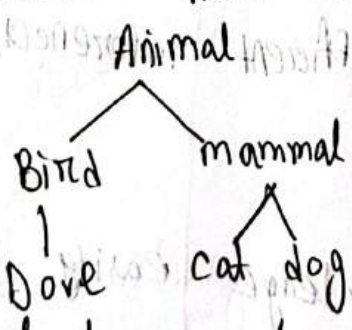
Four knowledge representation in AI =>

① Simple relational knowledge.

The most basic method of storing facts using tables (like in database). Facts are organized in rows and columns. Commonly used in relational database. But has limited reasoning/inference capability

② Inheritable knowledge.

Stores knowledge in hierachical class structure, use inheritance where subclasses inherit properties from parent class. use arrow show the relation b. from object to value.



If Animal has property 'can move' then cat, dove automatically is

inherit that.

(3) Inferential knowledge

represents ~~knowd~~ knowledge using formal logic. Enables the system to infer new knowledge from known facts. Guarantees correctness of conclusion if logic is valid.

Marcus is a man
Man is mortal > Marcus is mortal.

(4) Procedural knowledge

Defines how to do things using code or rules. Use if-then rules and in LISP, Prolog. and it is good for domain specific or heuristic knowledge. Not all scenarios can be effectively represented.

If it is raining then take an umbrella.

Requirements for knowledge representation system.

① representation accuracy.

Must represent all ~~the~~ kinds of required knowledge precisely.

② inferential adequacy

Must support deriving new knowledge from existing knowledge

③ Inferential Efficiency

should be able to make efficient inferences, using shortcut or guides.

④ Acquisitional Efficiency

Should acquire new knowledge easily, preferably automatically

Challenges in KR (Knowledge Representation)

① Ambiguity and uncertainty
words can have multiple meaning and data can be incomplete or imprecise.

② Scalability of representation.
Volume of knowledge and complexity of interrelation.

③ Balancing expressiveness vs Efficiency
Expressiveness,
Rich detail = better understanding, but slower processing.
Efficiency,
fast computation may skip details.

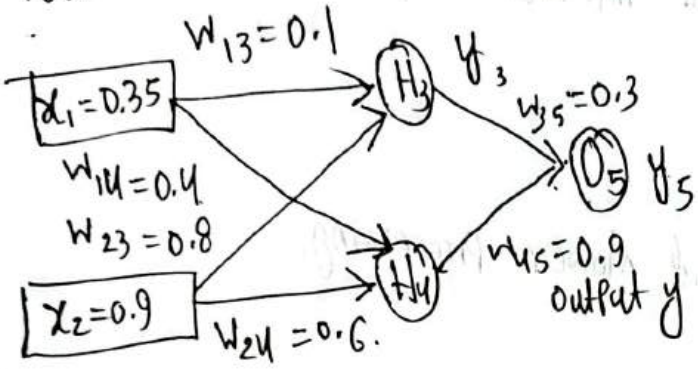
④ Dynamic update and maintenance.

Application of KR.

① Problem solving and decision making → use logic and rules.
② Robotics and Autonomus → semantic networks.
③ Healthcare industry → rules and ontologies.
④ search engine → use knowledge graph to connect queries to entities.

Mahabub Sir's Forward and Backward Propagation:

Assume that the neurons have a sigmoid activation function, perform a forward Pass and a backward Pass on the network. Assume that the actual output Y is 0.5 and learning rate is 1. Perform another forward Pass.



Ans

Using sigmoid function,

For a_1,

$(W_{13} * x_1) + (W_{23} * x_2)$

$= (0.35 \times 0.1) + (0.9 \times 0.8) = 0.755.$

$Y_3 = \frac{1}{1+e^{-0.755}} = 0.68$

For a_2,

$(W_{14} * x_1) + (W_{24} * x_2)$

$= (0.4 \times 0.35) + (0.6 \times 0.9) = 0.68$

$Y_4 = \frac{1}{1+e^{-0.68}} = 0.6637$

$a_3 = (W_{35} * y_3) + (W_{45} * y_4)$

$= (0.3 * 0.68) + (0.9 * 0.6637)$

$= 0.801$

$y_5 = \frac{1}{1+e^{-0.801}} = 0.69$

Error $= Y_{target} - y_5$

$= 0.5 - 0.69$

$= -0.19$

Xtra)

Each weight changed by

$\Delta w_{ji} = \eta \delta_j o_j$

$\delta_j = o_j(1-o_j)(t_j - o_j)$ if j is an output unit

$\delta_j = o_j(1-o_j) \sum \delta_k w_{kj}$ if j is a hidden unit

where η is a constant called the learning rate

t_j is the correct teacher output for unit j.

δ_j is the error measure for unit j.

For output unit, Y_5,

$\delta_{Y5} = Y(1-Y)(y_{target} - y)$ → first equation

$= 0.69 * (1-0.69) * (0.5 - 0.69) = -0.0406$

For hidden unit,

$\delta_8 = Y_3(1-Y_3) w_{35} * \delta_5$ → got from first thing.

$= 0.68 * (1-0.68)(0.3 * -0.0406) = -0.00265$

$\delta_4 = Y_4(1-Y_4) w_{45} * \delta_5$

$= 0.6637 * (1-0.6637) * (0.9 * -0.0406) = -0.0082$

XAI Explainable artificial intelligence means building AI system that can explain how and why they make a certain decision in a way that humans can understand.

A Black Box model is a system that you can see the input and output but you can't see or understand the process inside
XAI helps build transparency, accountability and trust.
XAI tries to:-
1) Make models themselves more understandable. (transparent model)
2) Use external tool to explain complex model. (Post hoc explainability)

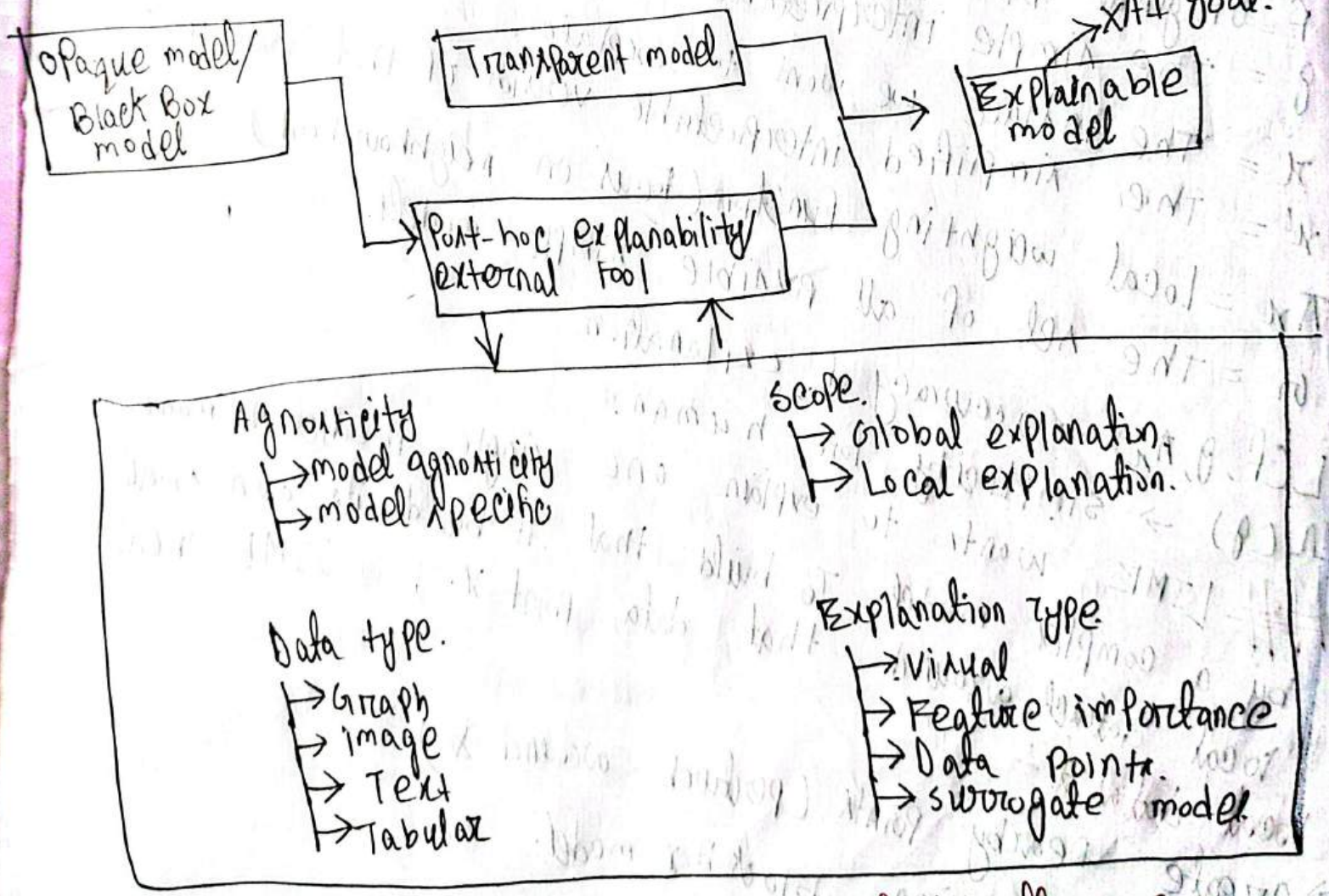
XAI Taxonomy:- XAI introduced terms that describe how explainable or interpretable a model is.

transparency:- A model is transparent if it is inherently easy to understand just by look at it.

Interpretability:- Interpretability is the ability to describe why a model made a certain decision in human understandable terms.

Explanability:- It connects the human and AI through clear, understandable explanations of individual predictions.

All explainable models are interpretable but not all interpretable models are fully explainable. Cause interpretable tells how easily we can understand the model but explainable focus how well we can explain individual decision. A neutral network is not interpretable by itself but. using explainability tools we can explain why it predicted Yes/No.



General ontology of explainable artificial intelligence.

LIME:- Lime is a technique that explains the prediction of any machine learning model by building a simple interpretable local model around the instance you want to explain. Lime works at →

① Choose the instance to explain.
② Zoom into a local region.
③ Create perturbed samples.
④ Interpret the local model.

① local
② interpretable
③ model agnostic

$$g = \underset{g \in G}{\operatorname{argmin}}\ L(f, g, \pi_x) + \Omega(g)$$

f = original black box model.
g = the simple interpretable model built locally.
x = The instace we want to explain
x' = The simplified interpretable version of that instance.
π_x = local weighting function (Focus on neighbourhood)
G = The set of all possible simple models.

L(f, g, π_x) → accuracy of explanation.
$\Omega(g)$ → simplicity for human.

LIME wants to explain one single prediction made by a complex model. To build that it builds its own small local dataset around that data point x. how LIME creates local dataset :-

① Create nearby points (perturb around x).
② Label them using black box model.

③ Weight each neighbour by Proximation.
Not all neighbours are equally important.

$$\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$$

$D(x,y)$ = distance between x and z.

σ (sigma) = kernel width → controls how large the neighbour-hood is

$\pi_x(z)$ = weight = how important z is for explaining x.

exp এর ভিতরে, D বড় হলে exp(D) ছোট

z is very close to x → $D(x,z)$ is small → exp() is large → high weight

z " " far from x → $D(x,z)$ is large → exp() is small → low weight

small σ ⇒ very local explanation

Large σ ⇒ broader neighbourhood.

local interpretable model agnostic

local interpretable model agnostic

④ Train a simple surrogate model $g(z)$

$$L(f, g, \pi_x) = \sum_{z \in Z} \pi_x(z)\left(f(z) - g(z)\right)^2$$

⑤ Keep it simple using LASSO (L1 Penalty)

$$\Omega(g) = \lambda \sum |w_i|$$

Large λ (lambda) → more weight become zero → simple explanation.

small λ → more features kept → more detailed explanation.

Submodular Pick-LIME (SP LIME)

LIME explains only one prediction at a time. But sometimes we want to understand the whole model. So SP LIME extends LIME to give a global explanation by choosing a small set of representative examples that together show the main behavior of model.

SP LIME works in 3 steps ⇒

① RUN LIME on all data points.
W[i,j] = how important feature j for instace i.

② Find Global feature importance.

$$I_j = \sum |W_{gij}|$$

If a feature appear often and strongly in many LIME explanation means it's total I_j score will be high.

③ Pick few representative instance.
we can't show every explanation to user, so SP LIME choose some example that covers as many important features as possible without repeating the same ones again and again. $C(V,W,I) = \sum_{j=1}^{p} 1[\exists i \in V : |W_{gij}| > 0] \times I_j$

Algorithm :-

① For each datapoint x_i : $W_i \rightarrow$ run LIME(x_i)

② For each feature j. I_j = sum of $|W_{ij}|$

③ start with empty set V.
while $|V| < B$:
Add instance that covers most new important feature.

④ Return final selected set V.

Adv of LIME

① Model Agnostic
② Local interpretability
③ human readable result
④ Fast for single instance

Dis Adv of LIME

① unstable explanation
② unrealistic samples
③ choice of neighbourhood is trick
④ Not truly global.

How to make heuristic function:-

1. Understand the Problem
2. define the state space
3. Determine the goal.
4. Choose a heuristic function like manhattan distance.
5. Admissibility: ensure that the function never overestimate the true cost to reach goal.
6. Fine tune and test
7. integrate with AI algorithm
8. Iterative refinement.

Admissible heuristic function is

$$h(n) \leq h^*(n)$$

$h^*(n)$ = heuristic cost, $h(n)$ = estimated cost.

** A heuristic function $h(n)$ estimates the cost or distance from a given node n to the goal in a search problem.

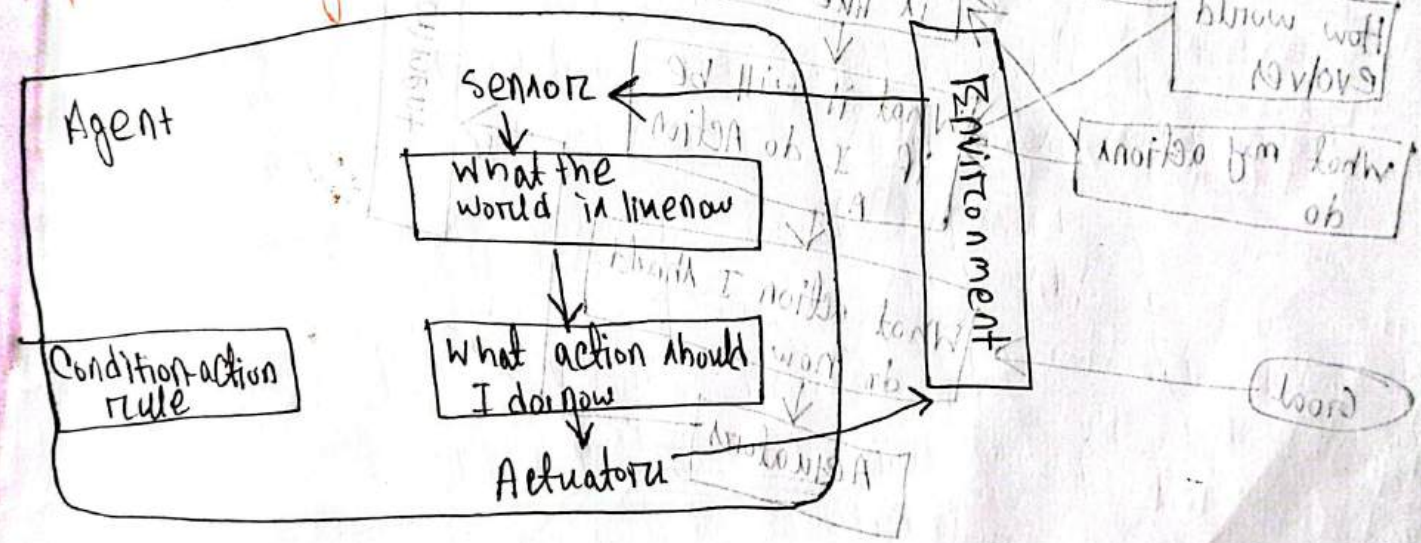
Algorithm of knowledge base AGENT

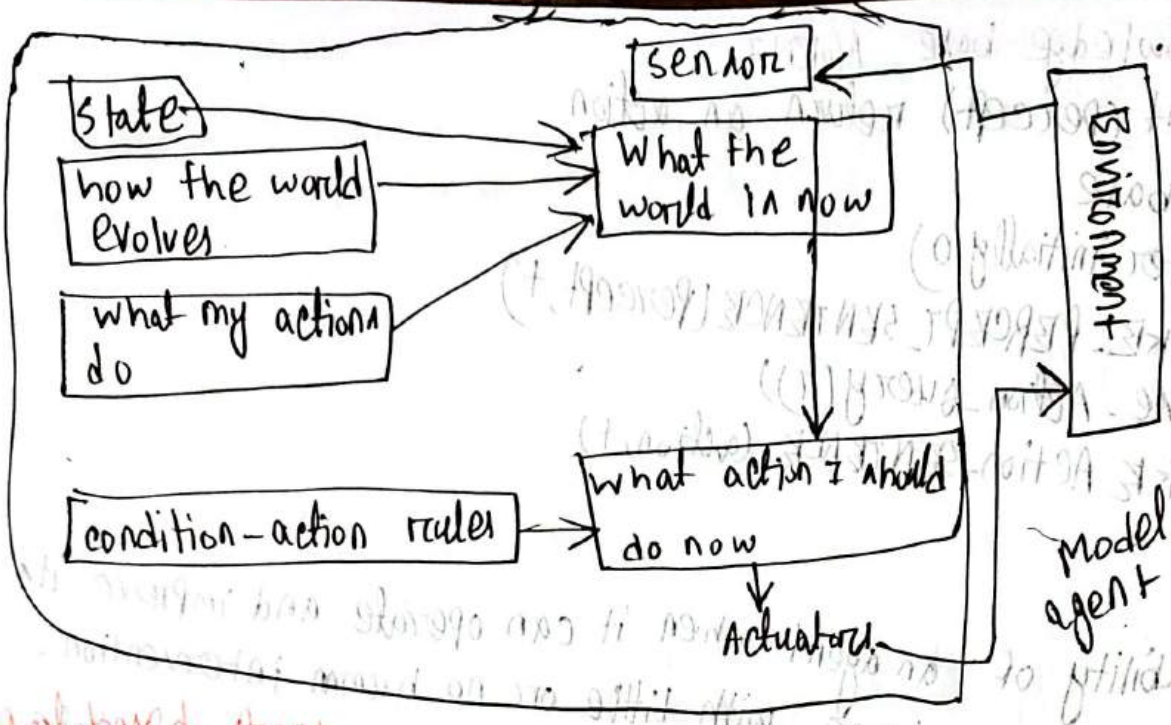
function KB-Agent (percept) return an action
KB: knowledge base
T: time (counter initially 0)
TELL (KB, MAKE_PERCEPT_SENTENCE (Percept, t))
action (KB, Make_Action_Query (t))
TELL (KB, MAKE ACTION_SENTENCE (action, t))
t = t+1
return action

Autonomy is the ability of an agent when it can operate and improve its behaviour using its own experience with little or no human intervention.

* when the next state is fully determined by the current state and the taken action then it is deterministic.

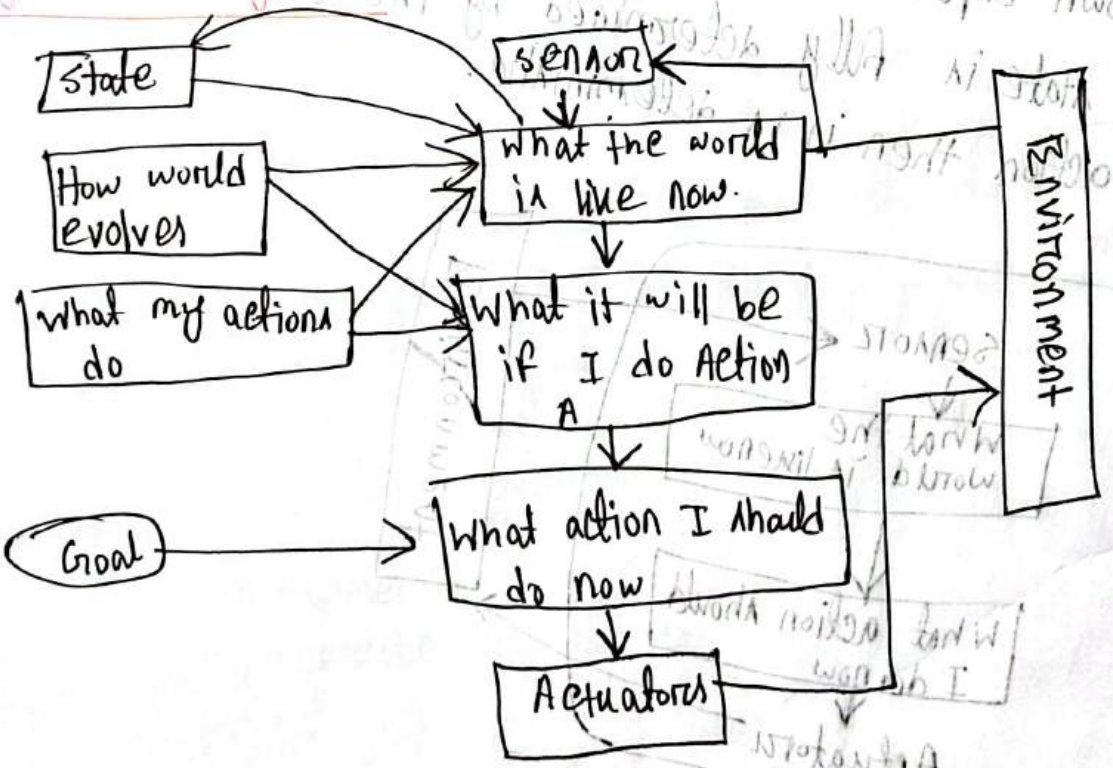
simple reflex agent

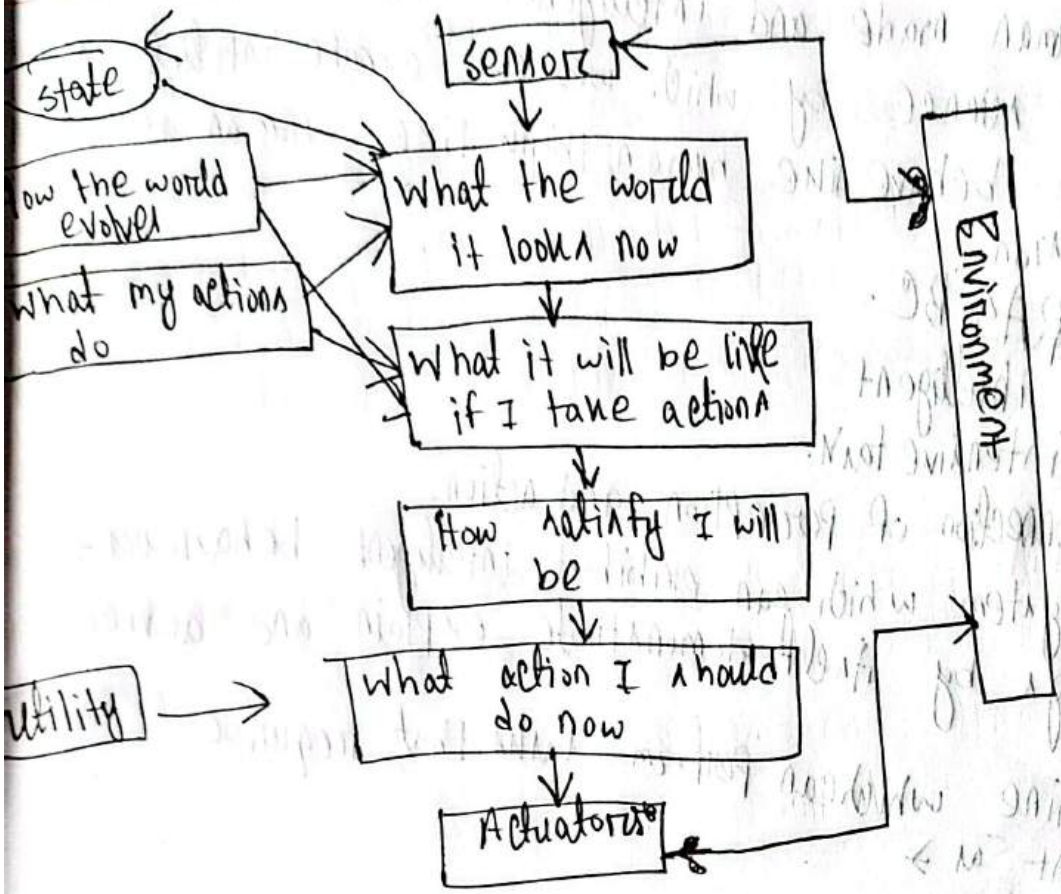


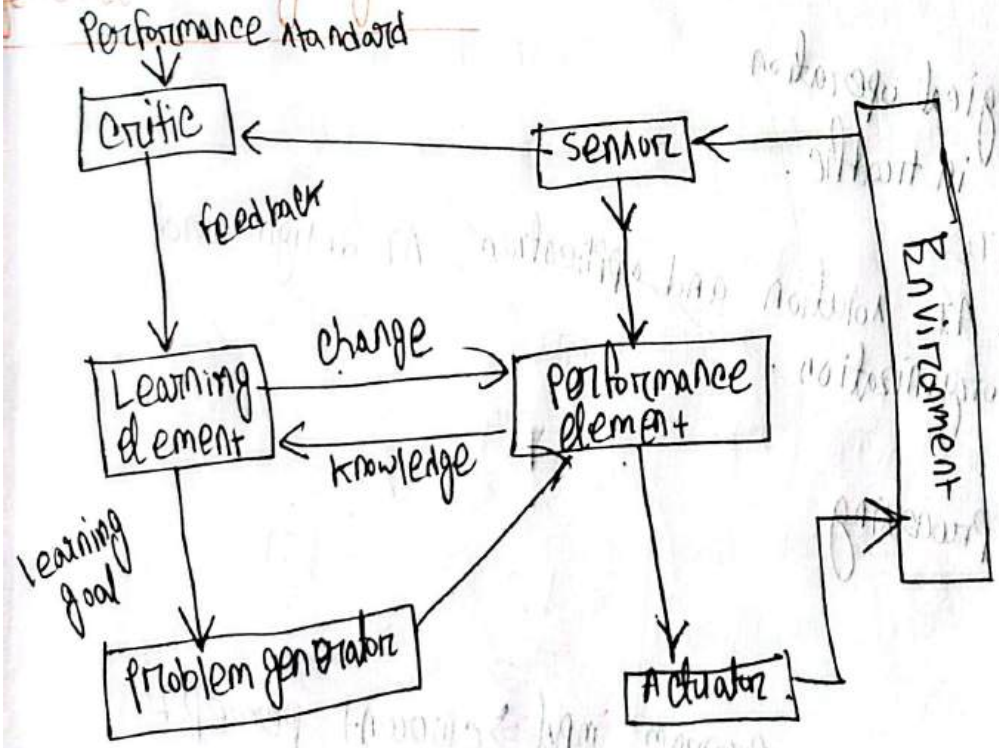


Model Based agent.

A goal based Agent




State
How the world evolves
What my actions do
Sensors
What the world is like now
What it will be like if I take actions
How satisfy I will be
What action I should do now
Actuators
Utility
Environment


Performance standard
Critic
Sensors
feedback
Learning element
change
knowledge
Performance element
Learning goal
Problem generator
Actuators
Environment

* Artificial intelligence =>

Artificial mean human made and intelligence means thinking power It is a branch of science by which we can create intelligence machine which can behave like human, think like human and able to make decision

Goals of AI => RAS BC.

1. Replicate human intelligent
2. Solve knowledge intensive task.
3. An intelligent connection of perception and action.
4. Creating some system which can exibit intelligent behaviours => learn new things by itself, demonstrate explain and advice to user.
5. Building a machine which can perform tasks that require human intelligent as =>
   1. Providing a theorem
   2. Playing game
   3. performing surgical operation
   4. driving a car in traffic.

Learning outcomes of AI =>

Human AI- interaction, AI solution and application, AI design and development, AI and organization.

1. Problem solving
2. Natural language Processing
3. Machine learning.
4. data analysis
5. decision support
6. Robotics.

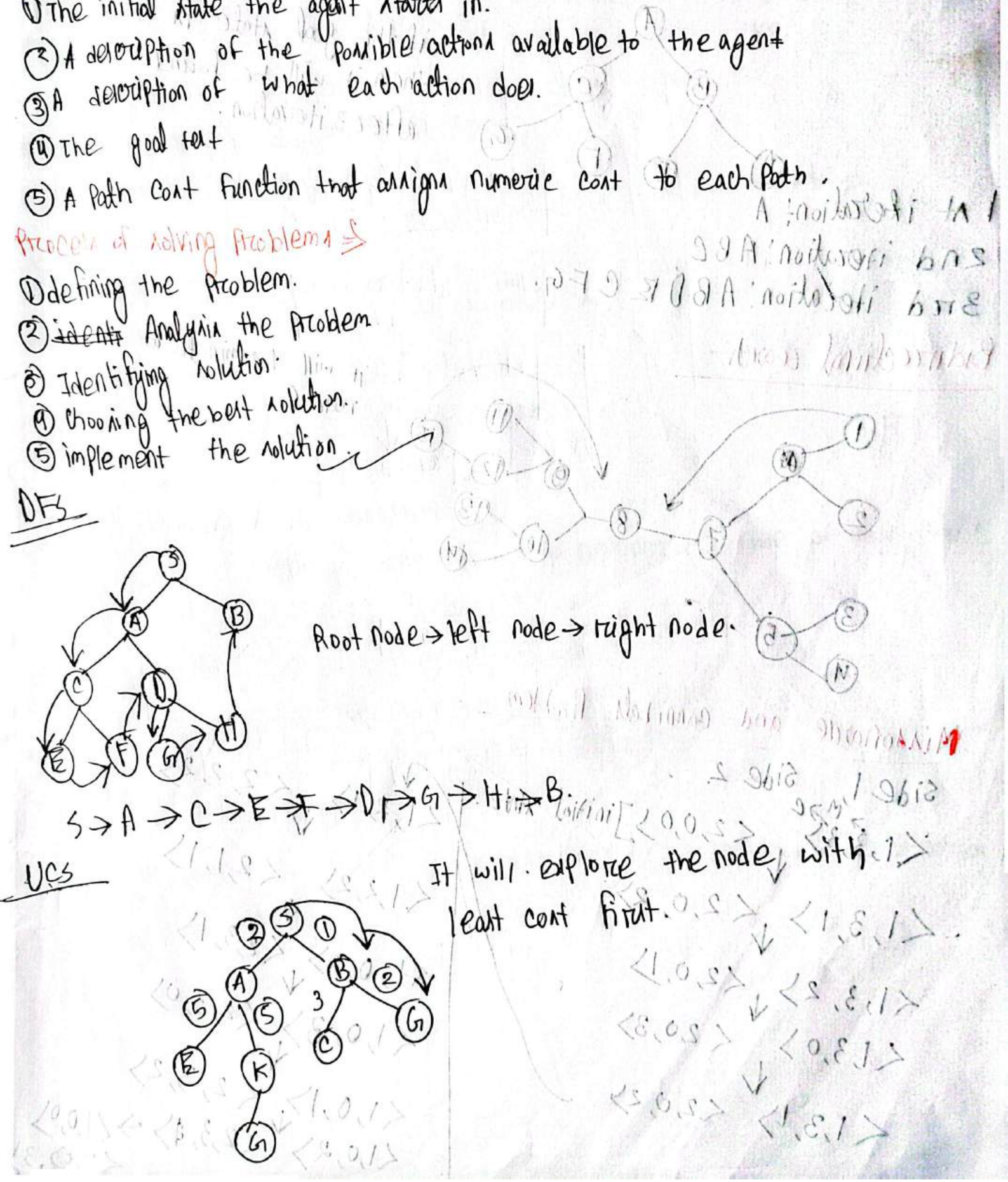
* Agent Program input => current percept
* Agent function input => full percept history.

A problem can be defined by five components ⇒ IAGP

1. The initial state the agent starts in.
2. A description of the possible actions available to the agent
3. A description of what each action does.
4. The goal test
5. A path cost function that assigns numeric cost to each path.

Process of solving problems ⇒

1. defining the problem.
2. ~~identi~~ Analysis the problem
3. Identifying solution.
4. Choosing the best solution.
5. implement the solution.



DFS

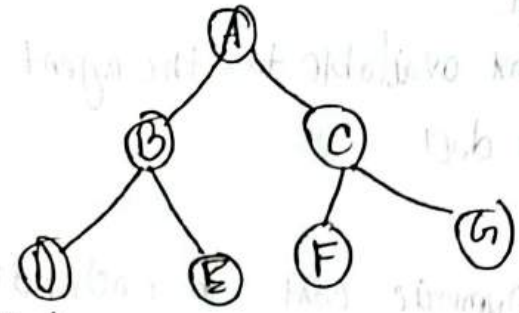
Root node → left node → right node.

S → A → C → E → F → D → G → H → B.

UCS

It will explore the node with least cost first.

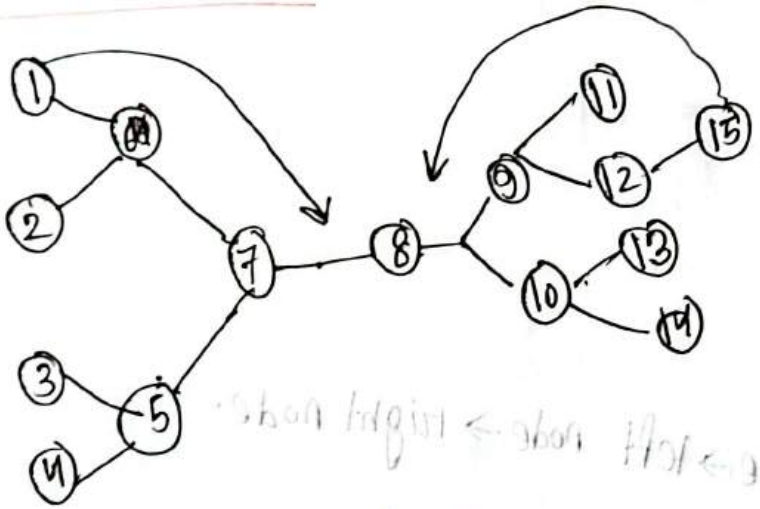
IDDFS



if the goal state is G then it will be found after 3 iteration.

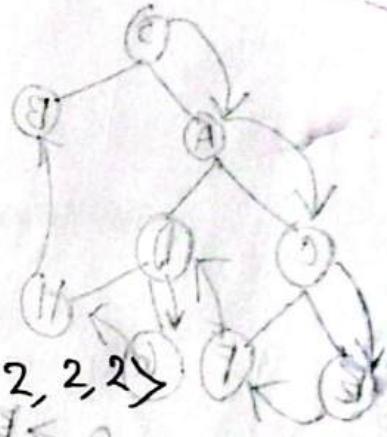
1 At iteration: A
2nd iteration: A B C
3rd iteration: A B D E C F G.

Bidirectional search:-



it will terminate at node 8.

Missionarie and Cannibals Problem



side 1, side 2.

$\langle 1,3,3 \rangle$ ($M \rightarrow$, $C \rightarrow$) $\langle 2,0,0 \rangle$] initial state

$\langle 1,3,1 \rangle$ $\langle 2,0,2 \rangle$

$\langle 1,3,2 \rangle$ $\langle 2,0,1 \rangle$

$\langle 1,3,0 \rangle$ $\langle 2,0,3 \rangle$

$\langle 1,3,1 \rangle$ $\langle 2,0,2 \rangle$

$\langle 1,2,1 \rangle$ $\langle 2,2,2 \rangle$

$\langle 1,2,2 \rangle$ $\langle 2,1,1 \rangle$

$\langle 1,0,2 \rangle$ $\langle 2,3,1 \rangle$

$\langle 1,0,3 \rangle$ $\langle 2,3,0 \rangle$

$\langle 1,0,1 \rangle$ $\langle 2,3,2 \rangle$

$\langle 1,0,2 \rangle$ $\langle 2,3,1 \rangle \rightarrow \langle 1,0,0 \rangle$

$\langle 2,0,3 \rangle$

* the actual cost of environment in reality
* state description is the abstract representation of world state.
* search node is representation of a state in search tree.

If heuristic is consistent it must be admissible ⇒

If a heuristic function is consistent,

$$h(n) \leq c(n, n+1) + h(n+1)$$

we begin at (n-1) node where n node is the goal node ⇒

$$h(n-1) \leq c(n-1, n) + h(n)$$

As n is the goal state so $h(n) = h^*(n)$ so,

$$h(n-1) \leq c(n-1, n) + h^*(n)$$

given that,

$$c(n-1, n) + h^*(n) = h^*(n-1)$$ we can see,

$$h(n-1) \leq h^*(n-1)$$

which is definition of admissibility.

To see if it is always true we now consider n-2 node of any path

$$h(n-2) \leq c(n-1, n-2) + h(n-1)$$

If h(n-1) is goal node then $h(n-1) = h^*(n-1)$

$$h(n-2) \leq c(n-1, n-2) + h^*(n-1)$$

we know that $c(n-1, n-2) + h^*(n-1) = h^*(n-2)$

$$h(n-2) \leq h^*(n-2)$$

By the inductive hypothesis this holds for all nodes proving consistency does imply admissibility.

First order logic

Syntax — Semantic

Symbol is the basic element of predicate logic.

Atomic " " most " sentence ⇒ Predicate (term1, term2 -- termn)

Quantifier → Universal, Existential Quantifier.

∀

All boys like cricket.

$\forall x$ boys(x) (like, cricket).

∃

$\exists x$ boys(x) (like, cricket).

Properties of quantifier:

$\forall x . \forall y = \forall y . \forall x$

$\exists x . \exists y = \exists y . \exists x$

$\forall x . \exists y \neq \exists y . \forall x$

Quantifier Duality

All boys like cricket

$\forall x$ boys(x) (like, cricket)

$\forall x$ like(x, Cricket) = $\neg \exists x$ $\neg$like(x, Cricket)

some boys like chocolate

$\exists x$ like(x, Chocolate) = $\neg \forall x$ $\neg$like(x, chocolate)

Given,

① Every student is sincere.

② All who are sincere and hard worker will succeed in their life

③ Meena is hard worker

④ Meena is a student

⑤ will meena succeed?

① $\forall x$ student(x) → sincere(x)

② $\forall x$ ↔ (sincere(x) ∧ hardworker) → succeed career(x).

③ Hardworker (meena)

④ student meena.

From eqn ① ⇒

student (meena) → sincere(x)

¬student (meena) ∨ sincere(x)

⇒ ¬T ∨ sincer (meena)

⇒ F ∨ sincere (meena)

⇒ sincere(meena)

From (3),

sincere(meena) ∧ hardworcher(meena) → succeed carer (x)

⇒ T ∧ T → succeed career (meena)

⇒ T → succeed career (meena)

⇒ ¬T ∨ succed carrer (meena)

⇒ F ∨ succed carreer (meena)

⇒ succeed career (meena)

∀ ⇒ use implies

∃ ⇒ use ∧.