Week 01(Intro to AI)

Environments

There are 7 types of environments:
For each type of environment the first is less complex, the second is more complex.

1. **Fully vs Partially observable**: Fully observable environments are those where the agent has all access to all relevant sensor information to make its desired choice. No need to keep the internal state in memory. Eg: Chess,Tic Tac Toe,Ludo, 8 way puzzle

Partially observable environments are where agents dont know everything about the environment due to faulty sensors, changing complex scenarios. Eg: Taxi driving,Poker, Vacuum Cleaner, PUBG

2. **Single vs Multi Agent:** Single Agent environment are those where only a single agent interacts with the environment to make a positive change. For eg: Crossword, 8 Puzzle

Multi Agent environments are those where at least two different agents are interacting with the environment and other agents to maximize their own output while minimizing others output. For eg: Chess, Poker, Taxi Driving, Tic Tac Toe, PUBG, Games with bot.

3. **Deterministic vs Stochastic:** Deterministic environment is where the agent's current state and selected action COMPLETELY determines the next state. In another way agents can accurately predict the outcome of the action, no uncertainty. These are environments which are fully observable.For eg: Chess, Tic Tac Toe,8 Puzzle,Crossword

Stochastic environment is where the agent can not be entirely sure what the outcome is as there is a lot of uncertainty. For eg: Poker, Taxi Driving, PUBG

4. **Static vs Dynamic:** Static environments are those environments which only change when agents interact with it and deliberately change them. For eg: Chess, Tic Tac Toe, Poker, Crossword, 8 Puzzle

Dynamic environments change on their own when agents interact with them. For eg: Taxi Driving, Vacuum Cleaner, PUBG, Real World

Uncertain outcome & Structure Change = Stochastic and Dynamic
Uncertain outcome = Only stochastic
Structure Change = Only Dynamic

5. **Episodic vs Sequential:** Episodic environments are those where the task performed now has no effect on the next task. For eg: Supply chain robot, defect checking robot, Vacuum Cleaner

Sequential environments are those where the task performed does have an effect on the next task. For eg: Chess, Tic Tac Toe,Poker

6. **Known vs Unknown:** Known environments are those where the agent has full knowledge of the environment.

Unknown environments are those where the agents DO NOT have full knowledge about the environment.

7. **Discrete vs Continuous:** Discrete environments have a finite number of percept and actions. For eg: Chess, Tic Tac Toe, Poker

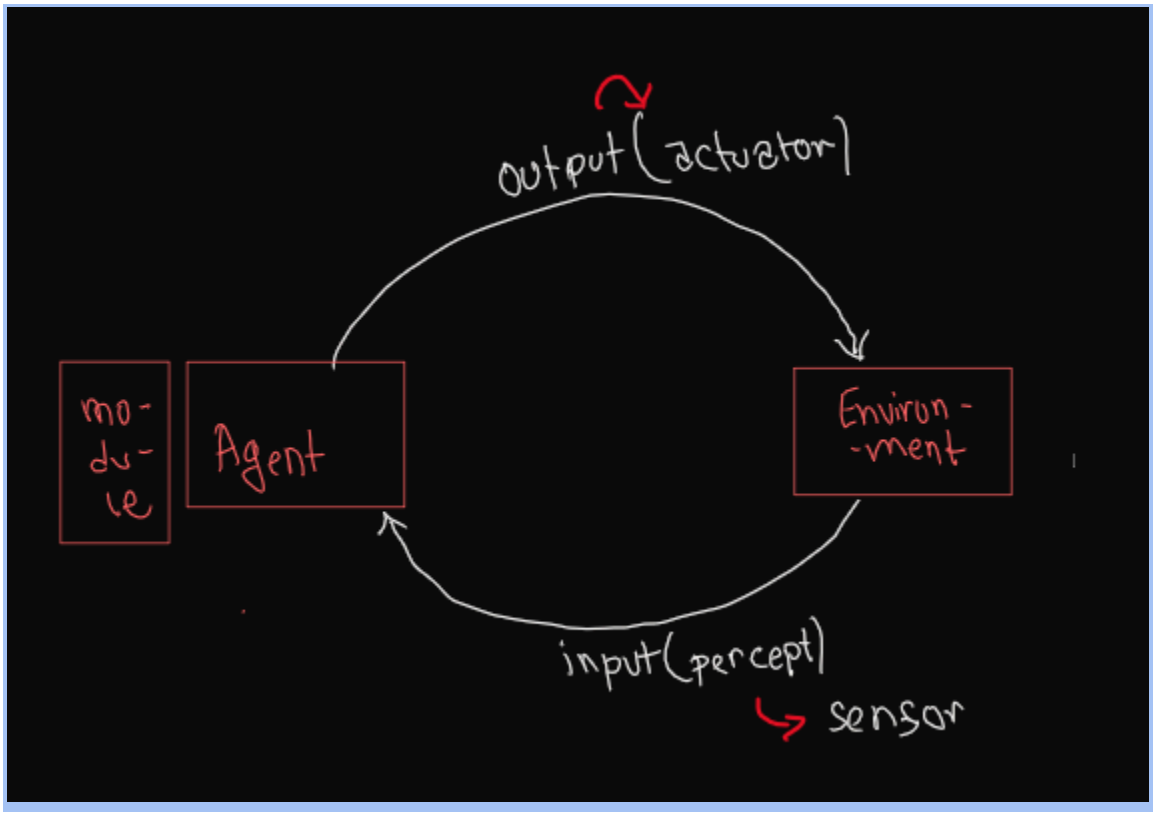
Sequential environments have an infinite number of percept and action. For eg: Taxi Driving

Name	Full vs Partial Observable	Single vs Multi agent	Deterministic vs Scholastic	Static vs Dynamic	Episodic vs Sequential	Discrete vs Continuous
8 way puzzle	Full	Single	Deterministic	Static	Sequential	Discrete
Crossword	Full	Single	Deterministic	Static	Sequential	Discrete
Chess	Full	Multi	Deterministic	Static	Sequential	Discrete
Tic Tac Toe	Full	Multi	Deterministic	Static	Sequential	Discrete
Go	Full	Multi	Deterministic	Static	Sequential	Discrete
Ludo	Partial	Multi	Scholastic	Static	Sequential	Discrete
Poker	Partial	Multi	Scholastic	Static	Sequential	Discrete
Part Picking Robot	Partial	Single	Scholastic	Dynamic	Episodic	Continuous
Taxi Driving	Partial	Multi	Scholastic	Dynamic	Sequential	Continuous
Vacuum Cleaner	Partial	Single	Scholastic	Dynamic	Episodic	Continuous
Medical Diagnosis	Partial	Single	Scholastic	Dynamic	Sequential	Continuous

NB: Red Color, more complex.

Go is more complex than chess as its state space is larger.

Agents



The agent takes in input(percept) from an external environment using sensors and processes inside processing modules and generates outputs with actuators which will bring about a positive change in the environment which is calculated using performance measurement.

Performance measurement(P), Environment(E), Actuator(A) and Sensors(S) - PEAS

Coffee Making Robot:

P: Customer Rating
E: Kitchen
A: Wheels, legs, Robotic arms
S: IR sensor, Camera

Search Engine:

P: Search result relevance
E: WWW
A: Computer Display
S: Keyboard, mic

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

1.Table Driven Agents:

- Lookup table for every possible percepts to map every possible percepts-actions
- Requires huge STATIC memory to store the large table. (Major Disadvantage)
- No dynamic memory, so looping might occur. As no record of previous state is stored.
- Not adaptive to changes in the environment. That is can not change the table
-

2.Simple Reflex Agents:

- No dynamic memory so no record of the past. So can't make decision based on the previous state
- Immediate response, dependent on current state using if-then logic.
- Only a single if condition not multiple.
- Environment must be fully observable
- No knowledge of the non perceptual state of the environment as there is no memory.

3.Model Based Agents(Agent with memory):

- Has a dynamic memory, so can store the previous states. As a result can make a decision based on the previous state
- Can work in a partially observable environment as the memory stores the previous states.
- Can adapt to the environmental change as can make changes in the memory.
- Can guess what the outcome of any action will be.

4.Goal Based Agents:

- On top of model based agents, here we will consider the goal of the system. We will target the goal. Searching and planning to reach my goal.
- Deliberative not reactive.
- For eg:

5.Utility Based Agents:

- On top of goal based agents, here the agent will consider how well the goal was achieved based on what way was taken to reach the goal. For eg: if we have an autonomous vehicle and it drives so fast to the goal that many bystanders are killed, then utility is low for that particular agent.

Week 02(Uninformed Search)

Uninformed Search Strategies	Informed Strategies
Start state, end state	
The position of end state is NOT known	The position of end state is known
No domain Knowledge(no knowledge if a path will take to the goal state or even if it does is it the most efficient way)	Domain Knowledge
No Path cost or step cost	Calculate path cost or

There are 5 types of uninformed searches:
1. Breadth First Search
2. Depth First Search
3. Depth Limited Search
4. Iterative Depth Limited Search
5. Uniform Cost Search(UCS)

Only UCS can work on weighted graphs.

Branching Factor(b): The maximum number of children each node/vertex has.
Depth(d): The maximum number of levels of the tree. Maximum number of vertex from root to leaf in a continuous straight line

State Space: Possible configuration state given in a tree.

Completeness = Can guarantee that the goal can be found.
Optimum = Can find the best goal node if multiple goal nodes exist.

1. **Breadth First Search(BFS):**

- Fringe = Frontier = Queue = First in First out (FIFO)
- Level Wise Traversal. First shallowest unexpanded node is selected for expansion.
- Shortest path from Start to End
- Only for BFS is the goal test carried out when the goal node is put into queue i.e. node generated. .
- Complete. Will visit all the nodes so will find the goal node even if it takes time.
- Optimum. As this is a level wise traversal. So the node which is closer to the root node will have a lower level value compared to the node which is further away.
- Time Complexity: b^d-goal visit/generated/put into queue or b^(d+1)-goal expand. This is exponential as in AI the number of nodes is huge.
- Space Complexity : b^(d+1)
- Better with: Depth is greater than branching

2. **Depth First Search(DFS):**

- Fringe = Stack = Last in First Out (LIFO)
- Depth Wise Traversal. Expand the node which is selected in the most recent time.
- Incomplete. Runs into a loop when iterating through loops.
- Not Optimum. There is a possibility that the lowest goal node is on the leaf where dfs visits first compared to the nearest goal on the other end of traversal.
- Time Complexity: b^(d+1). Goal test is done when the goal node is expanded.
- Space Complexity: bd. So it takes less space than BFS.
- Better with: Branching factor is greater than depth

3. **Depth Limited Search(DLS):**

- Fringe = Stack = Last in first out(LIFO)
- DFS with a pre-determined depth limit, l. The algo traverses through to l and then backtracks dfs.
- Fixed depths means it will remove the infinite loop issues from DFS.
- Incomplete. As the depth of the goal is beyond the depth limit
- Not Optimum. For l>actual depth,d. As it acts like dfs to some extent.

4. **Iterative Deepening Search(IDS):**

- DLS with an increasing predetermined depth limit,l starting from 0.
- Behaves like BFS as it technically searches level wise.
- Complete. It will visit all the nodes in a particular level before moving to the next level.
- Optimum. Will first iterate through an entire level before it goes to the next level so will always find the shallowest goal first.
- Time Complexity: b^(d+1)
- Acts like BFS if all paths cost the same and queue is used.

5. **Uniform Cost Search(UCS):**

- Also works with weighted graphs(the rest don't)
- Expands on the basis of the lowest distance between current node and start/root node.
- If all paths cost the same, act like BFS for fringe = queue. DFS for fringe = stack
- Search is terminated when the goal node is expanded.
- Complete. If there is infinite loop, path cost will increase to a point another branch has lower cost
- Optimum. Lowest path cost is found from start to goal if multiple paths exist.
- Time complexity: b^(c*/epsilon) + 1--goal node expansion. c* = the total cost from start to goal, and epsilon is the path taken.

Week 03(Informed Search)

Heuristic = It is an estimation of the cheapest path cost from current node and goal node. It must be an underestimation (less than or equal to) compared to the original path cost. It is problem specific. We get a heuristic value from a heuristic function. Eg:The calculation of estimated path cost between two points in cities (i.e. heuristics) along a straight line connecting two points(Euclidean distance) will always be the same or less than any present available actual path cost.

Admissible Heuristics = Heuristics where <u>all</u> the estimated values are lower than actual cost to reach the goal node from any nodes.

GreedyBest First Search :

- The algo only traverses the nodes(vertex) where the heuristic value(not actual cost) is the least.
- In average cases not all vertices are visited so the time and space complexity is lower compared to uninformed search.
- The heuristic value of the goal node is ZERO(0).
- Not optimal. There is a chance we will get a path with the lowest heuristic value but not least actual path cost due to how heuristic was chosen.
- Not complete. As we might choose a path with lowest heuristic value which loops in a circle and does not go to the goal state. [Fig 3.1]
- Time and space complexity = b^m. Most of the case

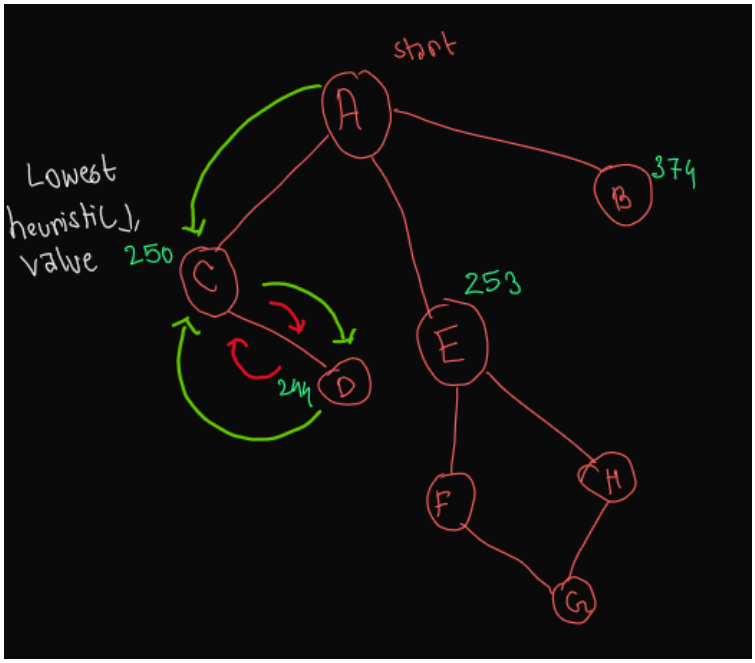


Fig: 3.1

A* Search:

- f(n) = g(n) + h(n). g(n) is the UCS part and h(n) is the greedy best first search part .
- It is a combination of uninformed search(UCS) and informed search(best first search)
- If all g(n) = 0 A* acts like greedy best first search, and if all h(n) = 0 its acts like UCS
- Time complexity: b^d
- Space complexity: b^d
- Complete. If there is infinite loop, path cost will increase to a point another branch has lower cost
- Optimal if heuristic admissible. If not admissible then not optimal.

Week 04(Local Search)

In uninformed and informed search, we had to keep track of the parent for each of the nodes, so we could actually see the path the algo took to reach the goal. This takes up a lot of **memory**.

But sometimes we do not require the way we reached the solution but just the solution itself. So point in keeping the nodes in memory. In comes Local Search. For eg: 8 queen problem,

There are two advantages of local search:
i. It takes up less memory as there is no need to store the path traversed or nodes visited.
ii. They can find solutions in an infinitely large state space with low space complexity.

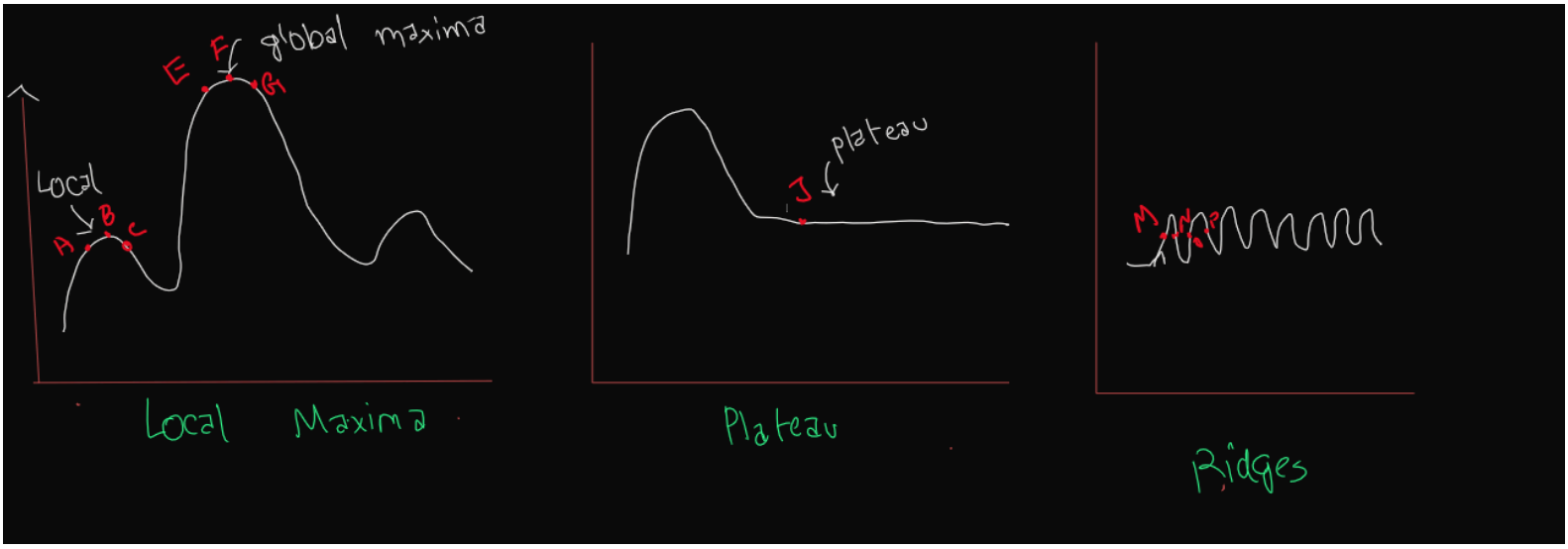
Hill Climbing

- Does not maintain a search tree. Only remembers the value of the current node and value of the object function.
- Does not look beyond the immediate neighbours of the current node just like going through a fog on a hill with amnesia.
- Also called greedy local search as it takes the immediate better state in sight(ie neighbour) not thinking what's ahead.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

Problems regarding Hill climbing



The higher the peak = Higher h value = Closer to 0 = Better goal(solution)

Local Maxima = When we are at point A and we go to the nearest neighbour ie B we have to move right and go uphill to B(h increases) from B if we move right we go to C which is downhill (h decreases). So from B if we move either way our h decreases meaning we will be moving away from our goal if we move from B thus making B our peak. But we can see our maximum peak is at F which is our global maxima and B is our local maxima

Plateau = If we move towards J from left and keep moving towards right our h value does not change hinting. We know we move forward if we get lower value. Here at J, we have no idea of where to go so this is known as PLATEAUX

Ridges = If the local maxima is really close to one another we get this problem. If we move right to M, we are moving uphill, now if we move right to N, we are moving downhill, then further right on O we are moving uphill and moving right to P, we are moving downhill.

How to resolve the drawbacks of Hill Climbing

Week 05+06(Constraint Satisfaction Problem)

It is a type of configuration search. In configuration search the path taken to reach a goal is not important rather we need to make sure we reach the goal.

There are 3 components:
1. A set of variables
2. A set of domains(values each variable can take)
3. Constraint (what combination of values a variable can take)

The flowpath of CSP:
1. Initial state. Here no variables are assigned any value yet.
2. Successor function. To assign values to variables.
3. Goal Test. Solution is consistent(no constrain broken by any variables) and complete(all variables have value assigned)

Constraint can be written in two ways:
1. Implicit: Generalized. Eg: No same color side by side
2. Explicitly: Broken down for every variable and every case. Eg: Color(A) != Color(B), Color(B) != Color(C), where A and B are side by side and B and C are side by side.

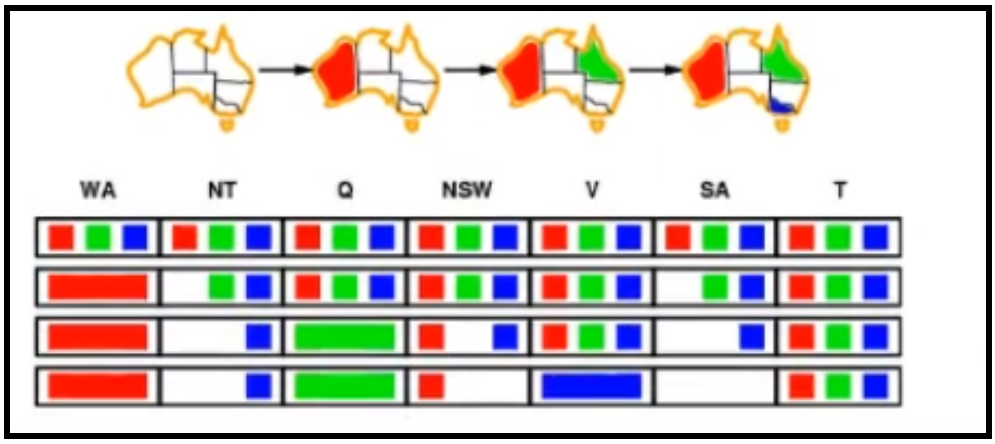
Backtracking means there will be many unwanted states we visited. We can improve our search by

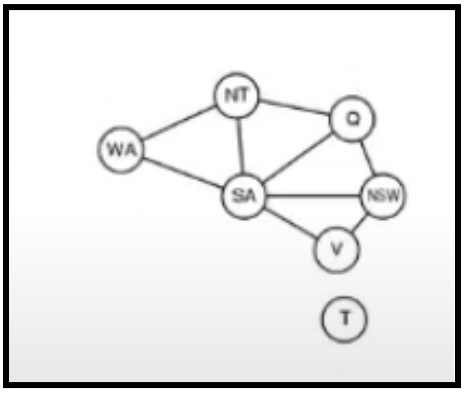
LEGAL VALUES = VALUES In DOMAIN
Nodes = Variables

Forward Checking

Also known as Node Consistency. As we are considering every node.

Two things we have to keep in mind.
1. Keep track of remaining legal values from unassigned values.
2. Terminate search when any variables do not have any legal values.
3. More efficient than backtracking but less efficient than Arc Consistency.



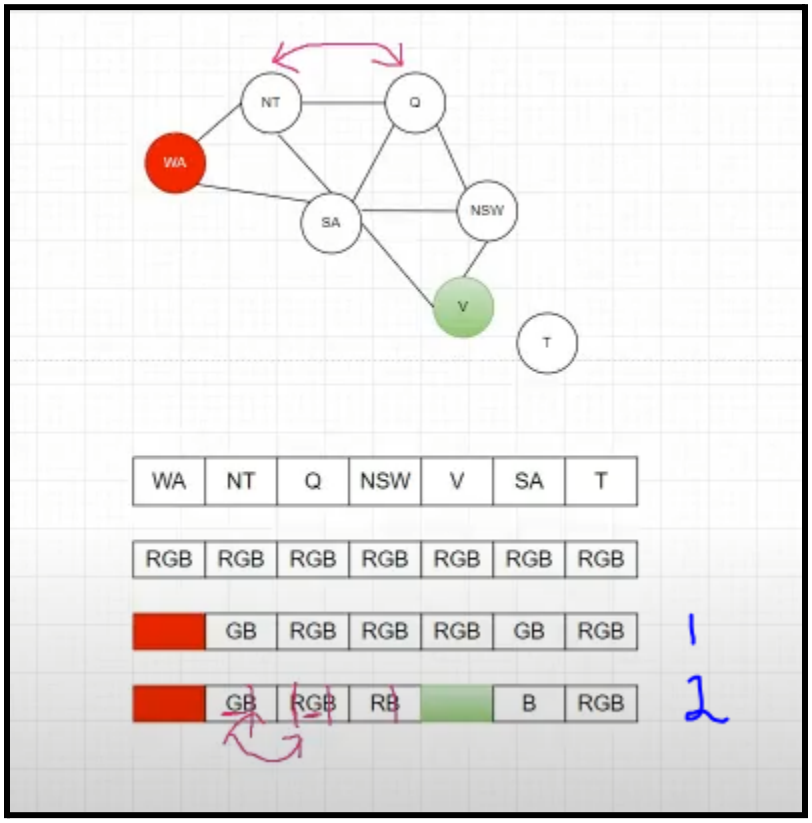


Degree huertics +FC > Degree heuristics

In step 1 we are initiating the search.
In step 2. We chose RED for WA and removed red from the legal values of adjacent nodes. And check to see if any variables(nodes) have zero legal values. There are not any so we continue.
In step 3. We chose GREEN for Q and removed green from the legal of adjacent nodes.And check to see if any variables(nodes) have zero legal values. There are not any so we continue.
In step 4. We chose BLUE for V and removed blue from the legal of adjacent nodes.And check to see if any variables(nodes) have zero legal values. And Yes SA has zero legal values so we stop the search

If we use normal backtracking we will have to color a lot of other nodes before we reach a dead state so in backtracking there are a number of useless states.

Arc Consistency



In step 2. Green is assigned to V. Now we remove G from legal values of SA and NSW. Now we are left with B in SA. So we remove B from SA's adjacent nodes. So B is removed from legal values of NT,Q and NSW. Then G is left in NT and Q so as they are adjacent so we have hit a dead end.

Ordering Heuristics

1. Most constrained variable- Also known as Minimum remaining values(MRV). An unassigned variable with the least number of legal values.(possible variable domain).
2. Most constraining variable- Also known as Degree heuristic. The variable with the most constraint on remaining variables. That is the variable with the most number of connected(neighbouring) unassigned variables.
3. Least constraining value- Choose the value for an unassigned variable which rules out the least number of values for the remaining connected variables.

First we choose a Most Constrained Variable, if there is a tie we choose a most constraining variable. If there is a tie again we choose one of the tied variables randomly and give the Least constraining value to it(if possible). Still there might be a chance of a dead end.

NB: If we have a node which is isolated or not connected to any other nodes, then we will choose at the very end.

Week 07(Probability)

not A = !A = ¬ A

Joint (Common) Probability

$$P(A \cap B) = P(A \wedge B)$$

$$= P(A \text{ and } B) = P(A, B)$$

Union Probability

$$P(A \cup B) = P(A \vee B)$$

$$= P(A \text{ or } B)$$

$$= P(A) + P(B) - P(A \cap B)$$

Conditional Probability

P(A) given P(B)

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

- $P(A \cap B) = P(B) \times P(A|B) = P(A) \times P(B|A)$

Independence

Complete Independence

- $P(A \cap B) = P(A) \times P(B)$
- $P(A|B) \cdot P(B) = P(B|A) \cdot P(A) = P(A) \times P(B)$

Conditional Independence

- $P(A \wedge B|C) = P(A|C) \cdot P(B|C)$
- $\frac{P(A \wedge B \wedge C)}{P(C)} = P(A|C) \cdot P(B|C)$
- $P(A \wedge B \wedge C) = P(A|C) \cdot P(B|C) \cdot P(C)$

↑ A & B independent given C

Joint Probability Table for 3 Variables

	+b		-b	
	+s	-s	+s	-s
+h	1	2	3	4
-h	5	6	7	8

$P(+h \cap +b) = 1+2$

$P(+b \cap +s) = 1+5$

$P(+h \cap +s) = 1+3$

$P(+h \cap +s \cap +b) = 1$

$P(+b) = 1+2+5+6$

$P(+h) = 1+2+3+4$

$P(+s) = 1+5+3+7$

$$P(+h \mid +b) = \frac{P(+h \cap +b)}{P(+b)}$$

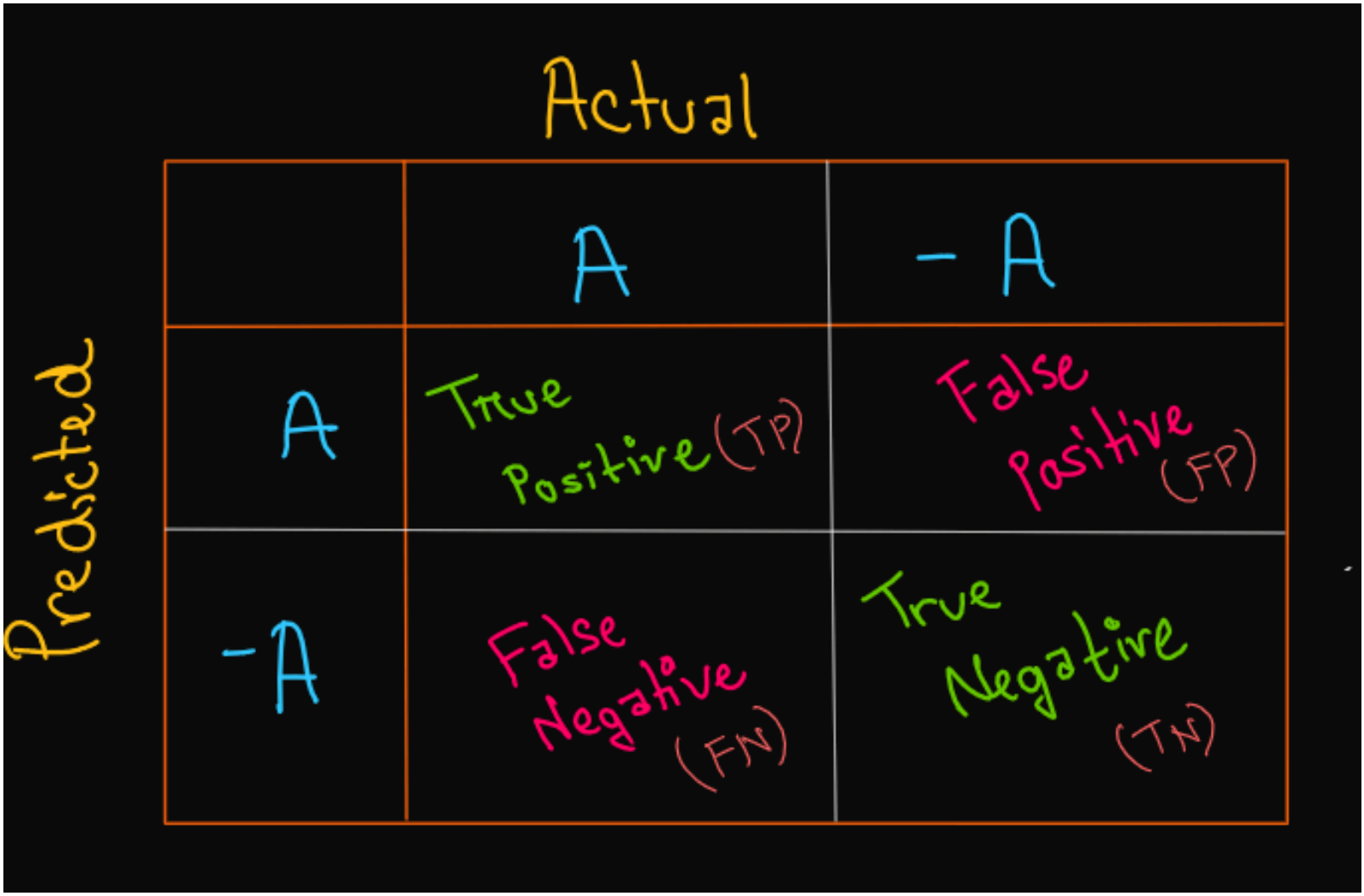
$$P(+h \cap +s \mid +b) = \frac{P(+h \cap +s \cap +b)}{P(+b)}$$

$$P(+h \mid +s \cap +b) = \frac{P(+h \cap +s \cap +b)}{P(+s \cap +b)}$$

$$P(+h \vee +s \mid +b) = \frac{P(+h \cap +b) + P(+s \cap +b) - P(+h \cap +s \cap +b)}{P(+b)}$$

Week 08.1(Bayes Theorem)

Confusion Matrix



Column = Actual Value of the Event
Row = Predicted Value of the Event. The outcome we got from our experiment.
Since we have Actual on the column and predicted on the row. Changing the heading will change the input in the boxes.

-A = negative of A

True Positive(TP) = We are supposed to get A, we got A
True Negative(TN) = We are supposed to get -A, we got -A
False Positive (FP)= We are supposed to get -A(neg), we got A(pos)
False Negative(FN) = We are supposed to get A(pos), we got -A(neg)

Sensitivity/Recall = The ratio of A(pos) which were correctly identified.

$$= \frac{TP}{TP + FN}$$

*The denominator is the row with the Actual Positive(Actual A).

Specificity = The ratio of -A(neg) which were correctly identified.

$$= \frac{TN}{TN + FN}$$

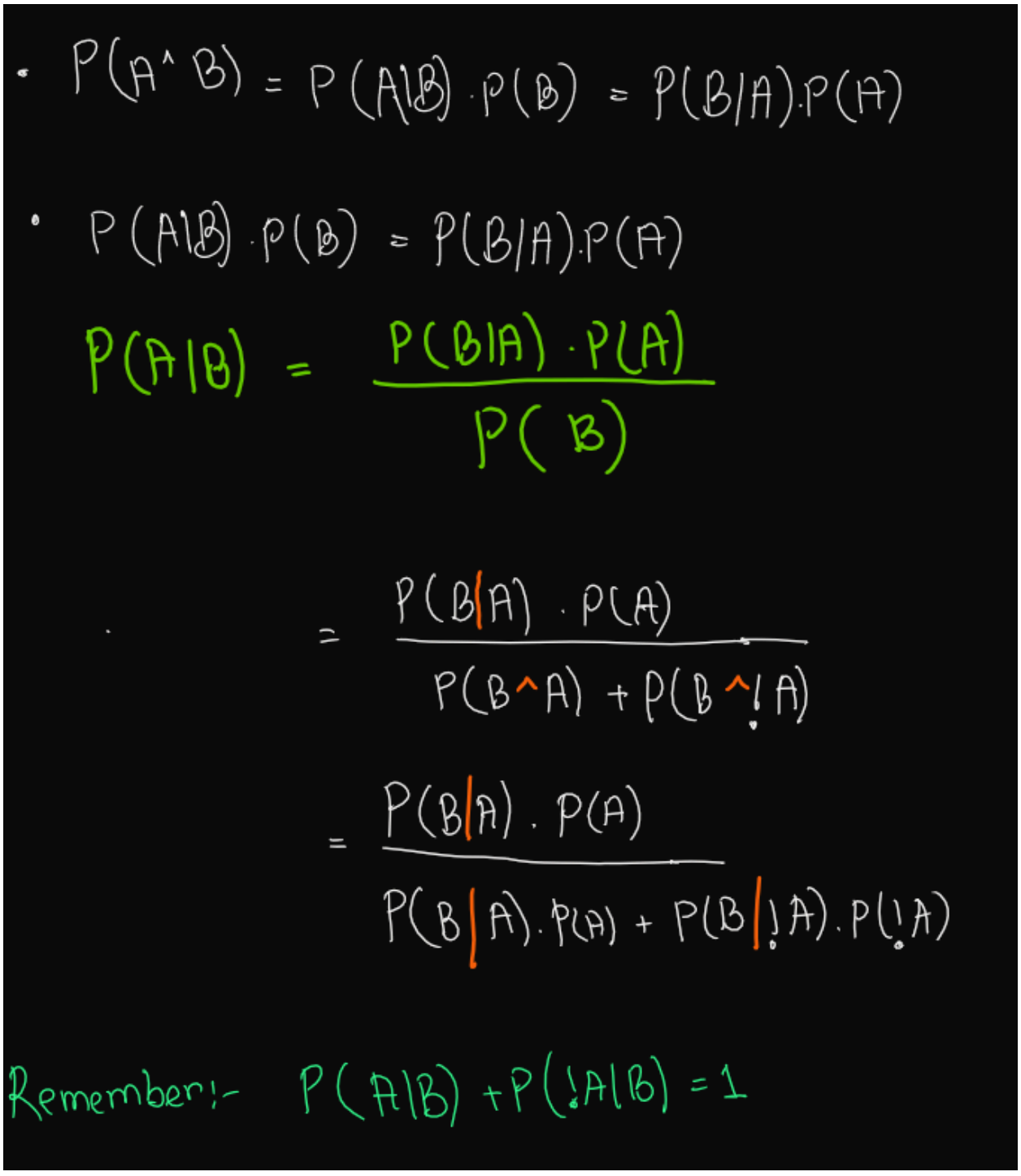
Accuracy = The ratio of values which we got correct.

$$= \frac{TP + TN}{TP + TN + FN + FP}$$

Precision = The ratio of actual positive from the total predicted positive.

$$= \frac{TP}{TP + FP}$$

Basic Of Bayes Theorem



Week 08.2(Naive Bayes)

“Naive” Bayes is Bayes Theorem with the conditional independence when multiple variables are present for the given part.
Here Getting A^B^C is a tough task as the data set needs to be huge so there is a chance we might get zero as the probability, so we ASSUME conditional independence.

$$P(D \mid A \wedge B \wedge C) = \frac{P(A \wedge B \wedge C \mid D) \cdot P(D)}{P(A \wedge B \wedge C)}$$

$$\text{'Naive' Bayes} \rightarrow = \frac{P(A \mid D) \cdot P(B \mid D)\, P(C \mid D) \cdot P(D)}{P(A \wedge B \wedge C)}$$

$$P(\text{effect} \mid \text{cause}) = \frac{P(\text{cause} \mid \text{effect}) \times P(\text{effect})}{P(\text{cause})}$$

$$P(\text{effect} \mid \text{cause}_1 \wedge \text{cause}_2 \wedge \text{cause}_n) = \frac{P(\text{cause}_1 \mid \text{effect}) \times P(\text{cause}_2 \mid \text{effect}) \times P(\text{cause}_n \mid \text{effect}) \times P(\text{effect})}{P(\text{cause}_1 \wedge \text{cause}_2 \wedge \text{cause}_n)}$$

B. Consider the dengue prevalence in Bangladesh as 0.01. Despite the good performance of the NS1 Antigen test, it also has 95% Specificity and 97% Sensitivity as well. A patient performed a first dengue test and he got the positive test result. He performed a second dengue test and again he got the test result as positive. What is the probability of having dengue of that patient? (4.5 marks)

$P(D) = 0.01 \quad P(-D) = 0.99$

Sensitivity: $P(T|D) = 0.97$

Specificity: $P(-T|D) = 0.95$

$P(-T|D) = 1 - 0.97 = 0.03$

$P(T|-D) = 1 - 0.95 = 0.05$

$$P(D|T_1T_2) = \frac{P(T_1 \wedge T_2|D)P(D)}{P(T_1 \wedge T_2)} = \frac{P(T_1|D).P(T_2|D).P(D)}{P(T_1 \wedge T_2)}$$

$$= 0.97 \times 0.97 \times 0.01$$

$$= 0.009409$$

$$P(-D|T_1T_2) = \frac{P(T_1 \wedge T_2|-D).P(-D)}{P(T_1 \wedge T_2)} = P(T_1|-D).P(T_2|-D).P(-D)$$

$$= 0.05 \times 0.05 \times 0.99$$

$$= 0.002475$$

The probability the patient having dengue is greater than not having dengue, after two tests.

Q) What is the problem with Bayes theorem? How can we solve that?

A) Let's say we want to find the probability of P(A^B^C | D). Now we need to have a huge dataset(2^4 unique row combinations) to get the correct combination of A,B,C,D otherwise we end up with a probability of zero. For this reason we use Naive Bayes because it assumes the conditional independence so instead of P(A^B^C | D) we get P(A|D)P(B|D)P(C|D). In this case the chances of getting a zero value of probability from the combination is far LOWER than the previous case.

Q) Why is Naive Bayes called “Naive”? /Why can Naive Bayes not be used in the real world?

A) In naive Bayes we assume conditional independence of the feature variables for the real world scenarios. In the real world the feature variables are not always conditionally independent of one another, so this approach is called ‘Naive’’ as it is an assumption.

Q)Would you Naive Bayes classifier for a large dataset?

A) Yes. Naive Bayes classifier is based on Naive Bayes which assumes conditional independence of feature variables. In a large dataset we will have multiple feature variables in the table so while training the model we might not get all the possible combinations of the feature and target variables we need which will result in us having the answer to probability in some cases zero. But with Naive Bayes classification we will very rarely face such a situation.