

Searching & Search Algos

[Module - 1 EMD
ADI]

Search = find a path from start state to goal state.

Performed $\Rightarrow$ Search is performed in state space, all possible world.

State captures everything needed to predict outcomes of action, assuming we are deterministic about actions, we know our goal state and we can observe the state space.

Ex:- Route Mapping & NQueens.

• How to model solution to the search problem??

$\Rightarrow$ Given search problem definition

- Initial state
- Actions
- Transition model
- Goal state
- Path cost

$\Rightarrow$ Construct search tree for state space graph.

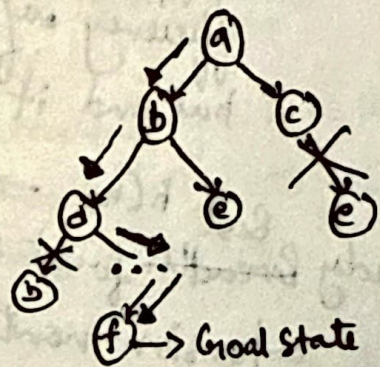
• Root Node = Initial state

• No cycles (to be deleted)

• Child Node reachable state using transition model

• path through the (corresponding sequence of action).

• Solution path ending at goal state.



• How different is AI Tree/Graph Search??

$\Rightarrow$ whole tree can't be fit in memory, builds part of the tree upon reaching the state or node using transition function, memory management is crucial.

Cycles are detected and prevented but are costly to be prevented and crucial.

BFS - Breadth first search :-

$O(b^m)$ $O(b^d) \Rightarrow$ maximum time complexity

is it complete $\Rightarrow$ If shallowest goal is at finite depth.

If b is finite

is it optimal $\Rightarrow$ cost is non-decreasing level by level

Depth first Search (DFS) :-

$O(b^m) \rightarrow$ maximum time complexity

is it complete $\Rightarrow$ if m is finite.

** no loop should be there **

Is it optimal $\Rightarrow$ If finds left most solution.

- Heuristic function $\Rightarrow$ Estimation function to scale how close is a state to the goal state, improves efficiency of search. May lead to invalid path but end it gives the desired path.

Best $h(n) \rightarrow$ symbol
Greedy ~~Breadth~~ first Search :- At each state after expansion next state to be expanded is decided using heuristic function. Greedy Search.

A-Star search $\rightarrow$ expands node with lowest combined cost with path cost + heuristic cost.

Priority Queue of $g(n) + h(n)$ is complete if $h(n)$ is admissible.

Iterative Depth First Search $\Rightarrow$ Performing Iterative DFS, where at each iteration depth maximum can be reached is increased. Starting from 0 to $D \rightarrow$ max Depth or until Goal state.

Local Search :- Start a current solution, improve the solution by moving state to better neighbor state (local moves).

**** assumption that we know utility of all possible states ****

Used for $\Rightarrow$ General optimization, Greedy Agent, Goal Based Agent identifying the goal state.

State Space, State representation, Objective function

Local Neighborhood

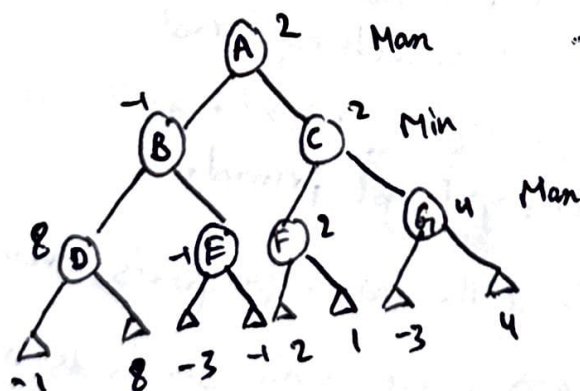
N-Queens as local search problem. State $\Rightarrow (q_1, q_2, q_3, q_4)$
 $\downarrow$
 queen positions

Greedy Local Search :- Greedy Best first search, just focusing on the local optima and moving forward this may result getting stuck at local

Simulated Annealing $\Rightarrow$ accepting some worse state trans. to get over getting stuck at local max/min.

Adversarial Search in Competitive Environments

MiniMax Algorithm $\Rightarrow$ Backtracking Algorithm, Max player will try to maximize the utility & Min will try to minimize the utility.



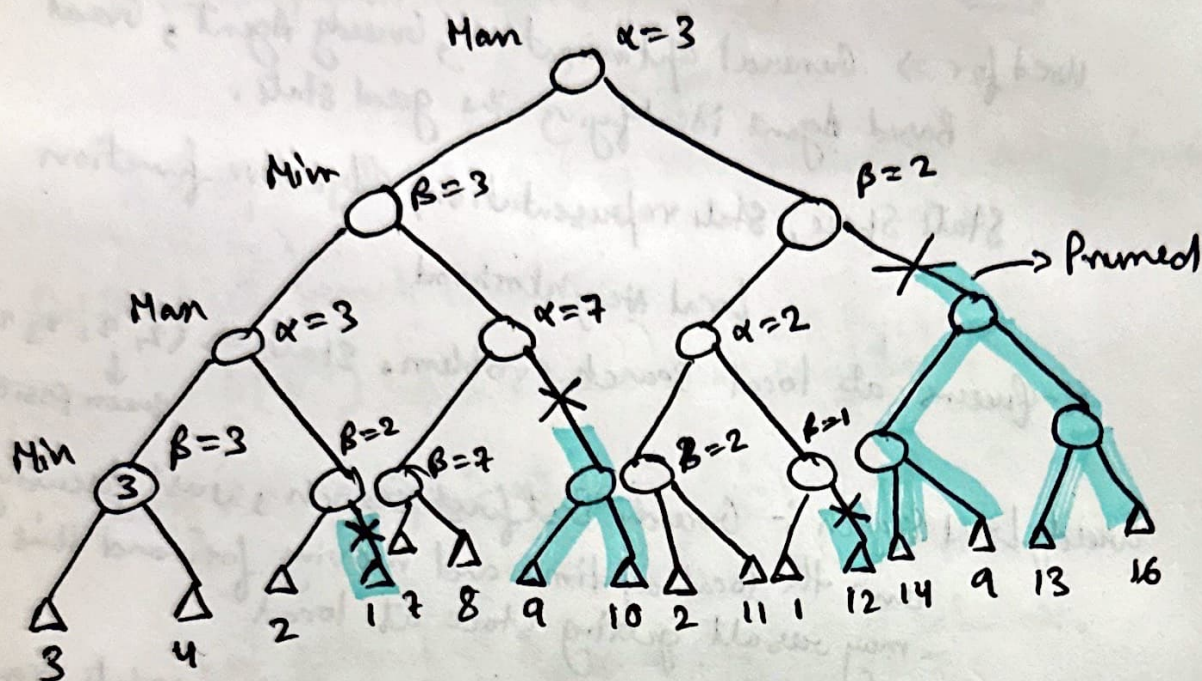
Time Complexity $\Rightarrow O(b^d)$
 $\downarrow$
 branching factor

The Algorithm explores this tree by simulating all possible moves by both players.

Evaluate terminal nodes using heuristic or utility function and Back propagate

Alpha-Beta Pruning (α - β) $\Rightarrow$ cut-off search by exploring less no. of nodes.

$\alpha \rightarrow \text{Max}$ $\beta \rightarrow \text{Min}$



- Explored 3 and $\beta=3$.
- Explored 4 and $\beta=3$ still stays.
- $\alpha=3$ at Max.
- Right subbranch explored.
- $\beta=2$ at right.
- 1 terminal got pruned.
- $\beta=3$ at Min level.
- Right sub-branch explored.
- $\beta=7$ at left of right, $\alpha=7$ accordingly
- Branch right got pruned.

Prune on Beta $\Rightarrow$ If the maximizer finds a value $\geq \beta$

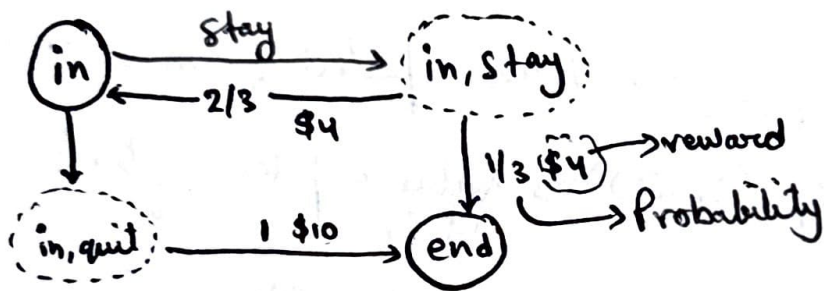
Prune on Alpha $\Rightarrow$ if the minimizer finds a value $\leq \alpha$

Reduces $O(B^m) \rightarrow O(B^{m/2})$ w/o Best case

Markov Decision Process

As far we have seen search problem with deterministic outcomes, every action or edge had probability 1. For uncertainty i.e. probabilistic transformations, Markov Decision Process is used for.

Ex: Dice game $\Rightarrow$ for each round $r=1, 2, 3, \dots$
you choose stay or quit.
If quit you get \$10.
If you stay you get \$4 & Dice is rolled $\Rightarrow$ (1,2) Dice game is quit
(3,4,5,6) game is same.



States: The set of states

$s_{\text{start}} \in \text{States}$

Actions(s): possible actions we can take at state s .

$T(s, a, s')$: Probability of s' at s with action a .

Reward(s, a, s'): reward for the transition $s \xrightarrow{a} s'$

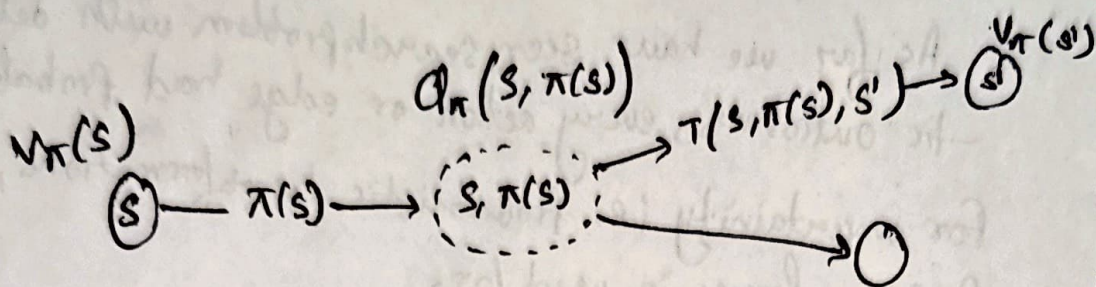
IsEnd(s): whether at the end.

Policy: (π) is a function that maps states to actions.

Utility is sum of rewards on the path.

Policy evaluation $\Rightarrow$ finding out how good it is to follow that policy starting from each state.

Building a recurrence to evaluate the policy $\Rightarrow$



$$V_{\pi}(s) = \begin{cases} 0, & \text{if } \text{isEnd}(s). \\ Q_{\pi}(s, \pi(s)), & \text{otherwise.} \end{cases}$$

$$Q_{\pi}(s, \pi(s)) = \sum_{s'} T(s, \pi(s), s') [\text{Reward}(s, \pi(s), s') + \gamma V_{\pi}(s')]$$

$V_{\pi}(s) \Rightarrow$ Value expected from state s

$Q_{\pi}(s, \pi(s)) \Rightarrow$ Value expected after taking action at s .

Ex:- Dice game $\Rightarrow$ Policy $(\pi) = \text{stay}$

$$V_{\pi}(\text{end}) = 0$$

$$V_{\pi}(\text{in}) = \frac{1}{3} (4 + V_{\pi}(\text{end})) + \frac{2}{3} (4 + V_{\pi}(\text{in}))$$

$$\Rightarrow V_{\pi}(\text{in}) = \frac{1}{3} \times 4 + \frac{2}{3} (4 + V_{\pi}(\text{in}))$$

$$\boxed{V_{\pi}(\text{in}) = 12}$$

Similarly $V_{\pi}(\text{end/quit}) = 10$.

$\Rightarrow$ * To sum up Markov Decision Process is a mathematical framework used to model decision making where states and outcomes are uncertain, means probabilistic and have different rewards.

Reinforcement Learning

Reinforcement Learning agent acts on the environment, it receives some evaluation of its action. Agent is not exposed directly to the end goal state. It is rewarded with some value for each action.

Training data $\Rightarrow (S, A, R)$
 $\swarrow \quad \downarrow \quad \searrow$
 State Action Reward

End goal is to create optimal policy (π) to maximise long term reward.

Elements :- policy $\Rightarrow \pi: S \rightarrow A$

 reward $\Rightarrow R(s)$

 value $\Rightarrow V_{\pi}(s, \pi(s))$

Supervised, Unsupervised, and Reinforcement Learning of Neural Networks

- Supervised Learning :- for every input vector $\vec{x}$ we have output $\vec{y}$. After each prediction $\hat{y}$, loss is calculated.

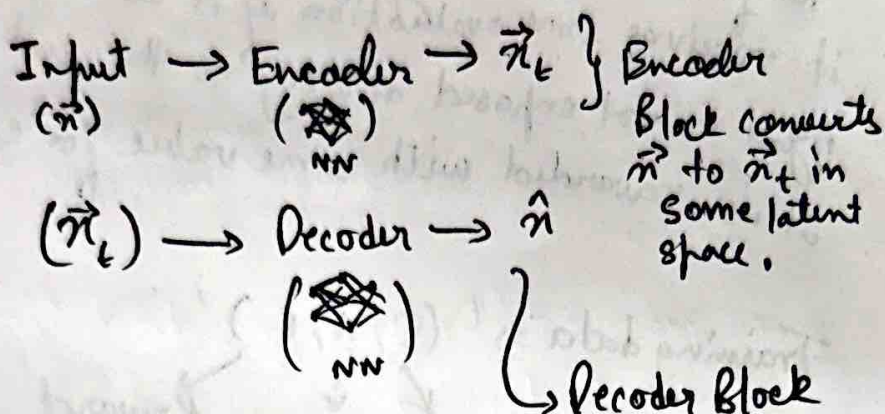
$l = (y - \hat{y}) \rightarrow$ loss this propagated back accordingly.

Ex:- Regression, Classification

- Unsupervised Learning :- for every training we only have input vector $\vec{x}$. No labels. Network tries to find patterns & structure in the records.

Ex:- Training of Autoencoder.

Training of Auto-Encoder $\Rightarrow$

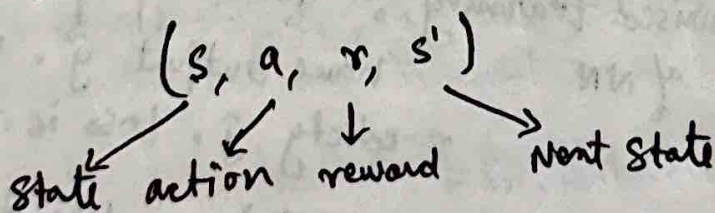


$$\text{Loss} \Rightarrow |\vec{n} - \hat{n}|$$

$\hookrightarrow$ This difference is backpropagated for training and used for learning patterns.

- Reinforcement Learning :- No labels or fixed dataset. There are some given to action in environment.

Agent Learns from $\Rightarrow$



Some Evaluation methods $\Rightarrow$ Policy Gradient, Q-Learning.

Fuzzy Logic

Reasoning with partial truths between 0-1.

handles uncertainty using human like language :-

not raining - example

fuzzification using membership function, if then statements.

Reasoning with Uncertainty

[Module 2 - AOI]

Reasoning in Uncertainty $\rightarrow$ Probabilistic Answering

Bayesian Networks $\Rightarrow$ Graphical Model showing dependencies between vars.

Hidden Markov Model $\Rightarrow$ Model sequential or Temporal Uncertainty.

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

$P(A|B) \rightarrow$ Posterior Probability

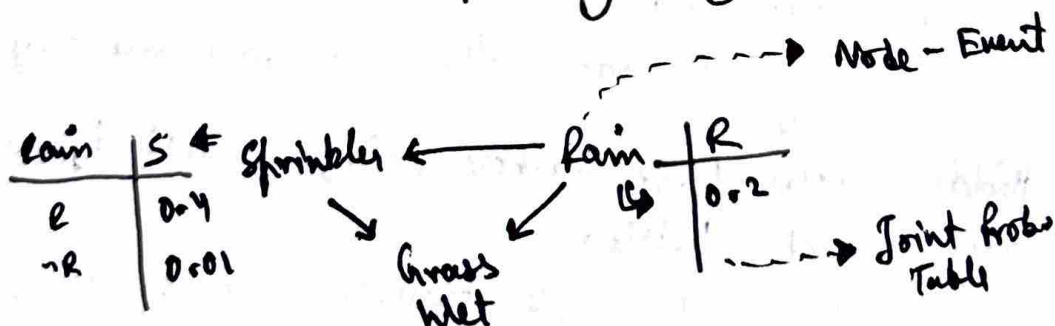
$P(B|A) \rightarrow$ Likelihood

$P(A) \rightarrow$ Prior Probability

$P(B) \rightarrow$ Evidence Probability

Bayesian Network

Probabilistic model that represents a set of variables and their conditional dependency using DAGs.



Sprinkler	Rain	W
$\sim S$	$\sim R$	0.0
$\sim S$	R	0.8
S	$\sim R$	0.88
S	R	0.99

$$P(w, s, r) = P(w|sr) \times P(s|r) \times P(r)$$

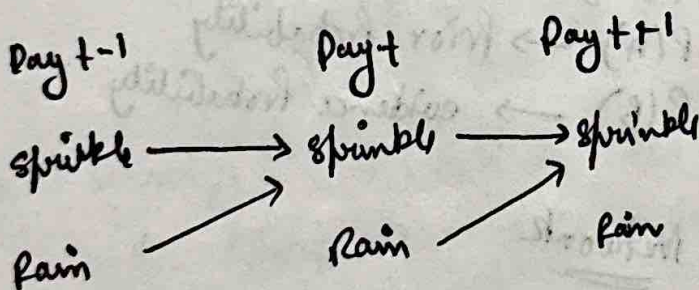
$$= 0.99 \times 0.4 \times 0.2$$

$$= \underline{\underline{0.0792}}$$

Dynamic Bayesian Network

Temporal probabilistic model which reasons using both uncertainty and temporal dependency.

Consists of repeated "time slices", each representing the system at a specific time step, with edges showing dependencies within and between slices.

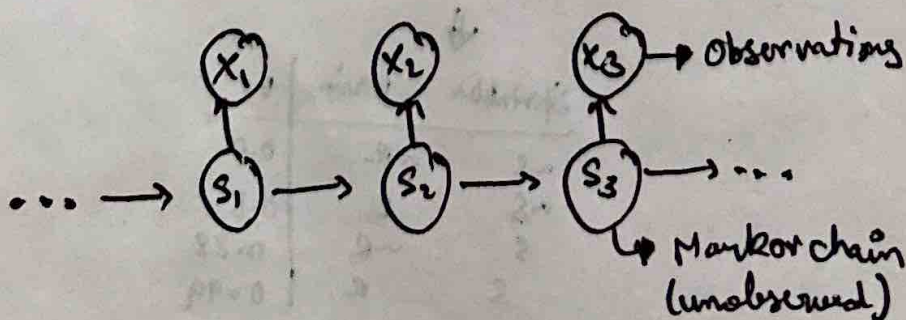


Hidden Markov Model

A hidden markov model is a statistical model for sequences where the system hidden states and observable outputs

Hidden states form markov chain, next state depends on current state.

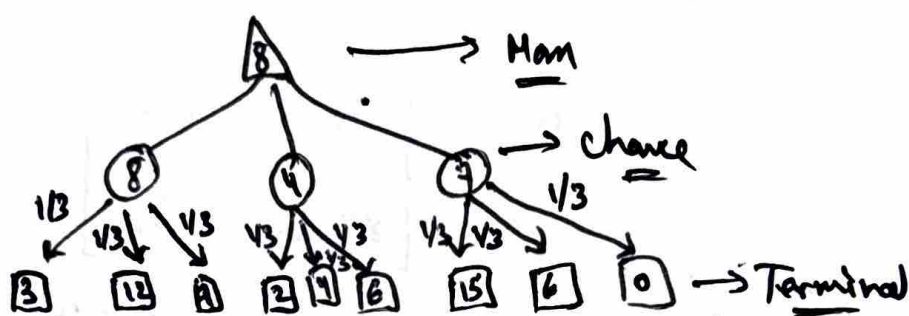
Each hidden state emits observation according to PDF.



Expectimax

Expectimax extends minimax by handling chance nodes, which represents random events, in addition to max or min player.

At chance nodes, instead of taking minimum or maximum weighted expected value is taken using PDF.



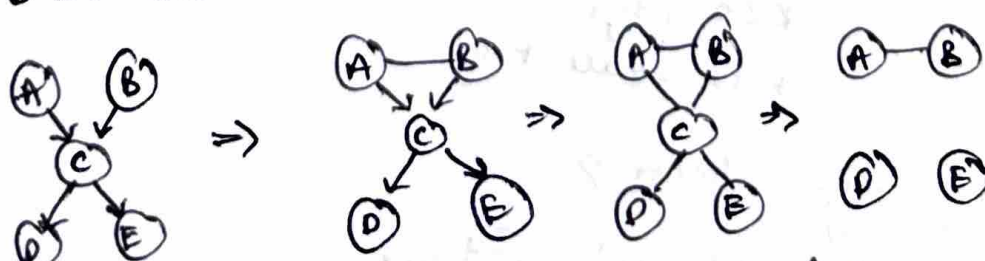
Dependence in Bayesian Network (D-Separation)

- Steps $\Rightarrow$
- Draw Ancestral Graph
 - Marginalize
 - Disorient
 - Delete Observed

for questions like $P(D|CEG) = P(D|C)$, we have to check if both have only path by D if Given C.

Logically Dependent.

Marginal Dependent means there exists path between A & B and vice-versa.



D & E are conditionally independent.

Feature Detection Technique

* Harris Corner Detection $\Rightarrow$

- Use Sobel Filters $\Rightarrow$
$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$
$$I_y \quad I_x$$

- Compute I_x & I_y .
- Correlation Matrix $\Rightarrow$

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

$$R = \det(M) - K \cdot (\text{trace}(M))^2$$

- if R is high Corner.
- $R < 0$ is Edge.
- R is small Flat.

* Shi-Thomasi Corner Detector $\Rightarrow$

- Compute Correlation Matrix with Sobel filters.
- λ_1, λ_2 are Eigen values.
- $R = \min(\lambda_1, \lambda_2)$
- if R is high Corner
- $R < 0$ Edge
- R is small Flat

* Canny Edge Detector $\Rightarrow$

- Gaussian Filters.

- Gaussian Gradient using Sobel Filters.
- Maximum local Only.
- Double Thresholding.
- Connect weak edges to strong else discard.

Feature Extraction Technique

* SIFT (Scale Invariant Feature Transform) $\Rightarrow$

Identifies keypoints at multiple scale and describes each keypoint by constructing a histogram of local gradients orientations.

Features are immune to scale, orientation & distortions.

* SURF (Speeded Up Robust Features) $\Rightarrow$

Identifies Keypoints at multiple scale which are built using integrals for efficient scale computation.

Immune to viewpoint changes & lighting.

* Oriented FAST and Rotated BRIEF (ORB) $\Rightarrow$

Combines FAST corner detection & BRIEF descriptor for creating features in real time with low latency.