AI

Date
Page No

Unit I

Introd" to AI

→ AI refers to the simulation of human intelligence in machines that are programmed to think and act like humans.
→ It involves development of algorithms and computer programs that can perform tasks that typically require human intelligence such as visual perception, speech recognition, decision-making, and language translation.
→ AI systems are more generic (rather than specific), can "think" and are more flexible.

→ Intelligence is the ability to acquire and apply knowledge.
Knowledge is the info acquired through experience.
Experience is the knowledge gained through exposure (training).
We get AI as the "copy of something natural (i.e., human beings) 'WHO' is capable of acquiring and applying the information it has gained through exposure.

Intelligence is composed of →
* Reasoning
* Learning
* Linguistic Intelligence
* Problem-Solving
* Perception

Date:
Page No:

Uses of AI → ① Healthcare – medical diagnosis, drug discovery, predictive analysis of diseases
② Finance – credit scoring, fraud detn, financial forecasting
③ Retail – product recommendations, price optimizn, supply chain management.
④ Manufacturing – quality control, predictive maintenance, production optimization.
⑤ Transportation – autonomous vehicles, traffic prediction, route optimization
⑥ Customer Service – chatbots (to answer frequently asked queries)
⑦ Security – facial recognition, intrusion detn, cyber-security threat analysis.
⑧ Marketing – targeted advertising, customer segmentation, sentiment analysis.
⑨ Education – personalized learning, adaptive testing, intelligent tutoring systems.

Need for AI → ★ To create expert systems that exhibit intelligent behaviour with the capability to learn, demonstrate, explain and advise its users.
★ Helping machines find solutions to complex problems like humans do and applying them as algorithms in a computer-friendly manner.

Approaches of AI → There are total 4 approaches

① Acting humanly (The Turing Test Approach)
→ In this approach a computer passes the test if a human interrogator, after asking some

written questions, cannot identify whether the written responses come from a human or from a computer.

(2) Thinking humanly (The cognitive modelling approach)
→ In this we determine whether the computer thinks like a human.

(3) Thinking Rationally (The "laws of thought" approach)
→ In this we determine whether the computer thinks rationally i.e. with logical reasoning.

(4) Acting Rationally (The rational agent approach)
→ In this approach we determine whether the computer acts rationally i.e. with logical reasoning.

Applications of AI → NLP, Gaming, Speech Recognition, Vision Systems, Healthcare, Automotive.

forms of AI →

(1) Weak AI → * AI created to solve a particular problem or a specific task.
* it is not a general AI and only used for specific purpose
* Ex → AI used to beat chess grandmaster.

(2) Strong AI → * difficult to create than Weak AI
* It is a general purpose intelligence that can demonstrate human abilities such as learning from experience, reasoning, etc.

Date:
Page No:

③ Super Intelligence – ★ It is an AI that is much smarter than the best human brains in practically every field.
★ It ranges from a moderate being just smarter than a human to a machine being trillion times smarter than a human.
★ It is the ultimate power of AI

About AI system –
An AI system is composed of ← agent, environment

Agent → (human/robot) — anything that can perceive its environment through sensors and acts upon that env. through effectors.
→ Intelligent agents must be able to set goals and achieve them.
→ In some problems if the agent is the only actor then it can be certain of the consequences of its actions.
However if the agent is not the only actor, then it requires that the agent can reason under uncertainty.
So we need an agent that cannot only assess its env ~~but also evaluate its predictions~~ and make predictions but also evaluate its predictions and adapt based on its assessment.

Drawbacks of AI – ① Bias & Unfairness → carrying of existing biases
② Lack of transparency & accountancy
③ Job displacement

Date
Page No:

④ Security and privacy tasks ⑥ Regulation.
⑤ Ethical concerns (autonomous weapons).

Technologies based on AI – ① Machine learning
② NLP ③ Computer Vision
④ Robotics ⑤ Neural Networks (on funct^n of Brain)
⑥ Expert Systems (mimic decision-making ability of human)
⑦ Chatbots.

Problem Solving

→ Reflex agent of AI directly maps states into actions. When agent fails where state of mapping is too large then the stated problem dissolves and sent to a problem solving domain which breaks the large stored problem into smaller storage area and resolves one by one. The final integrated action will be the desired outcome.

→ On the basis of the problem and their working domain, different types of problem-solving agent defined and use at an atomic level without any internal state visible with a problem-solving algorithm. The problem solving agent performs precisely by defining problems and several solutions. Hence a problem-solving agent is a result-driven agent and always focuses on satisfying the goals.

Date:
Page No:

Types of problem in AI -

(1) Ignorable - here sol^n steps can be ignored

(2) Recoverable - sol^n steps can be undone

(3) Irrecoverable — sol^n steps cannot be undo

Steps problem-solving in AI (as AI directly related to human nature & activities)

Steps Include -

* Problem definition - Detailed specification of inputs and acceptable system sol's.

* Problem Analysis - Analyse the problem thoroughly

* Knowledge Represen^n - collect detailed info about the problem and define all possible techniques.

* Problem - Solving - selection of best techniques

Components to formulate the associated problem

* Initial State - starts the AI agent towards a specified goal.

Knows methods initialize problem domain solving by a specific class.

Date
Page No

* Action – works with func" with a specific class taken from initial state and all possible actions done in this stage.

* Transition – Integrates actual action done by the previous action stage and collects the final stage to forward it to their next stage.

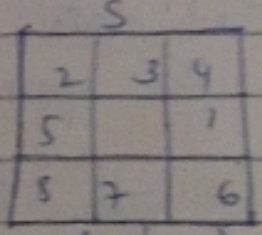
* Goal Test – determines that the specified goal achieved by the integrated transition model or not, whenever the goal achieves stop the action and forward into the next stage to determine the cost to achieve the goal.

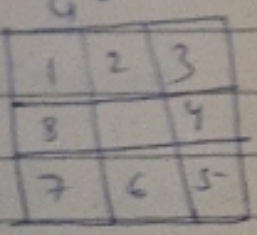
* Path costing – assigned what will be the cost to achieve the goal.
→ requires all hardware, software & human working cost.

State - Space Search

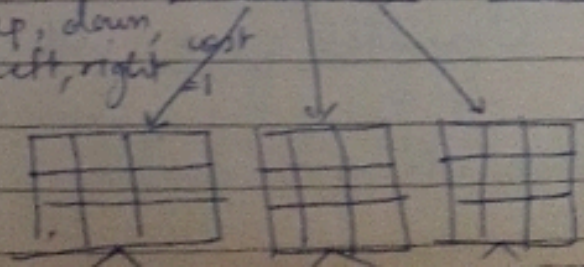
→ used to locate a goal state with the desired feature.

S: { S, A, Action(S), Result(S, a), Cost(S, a) }





→ pre use
→ analyse



Date:
Page No:

→ A state space is a set of all possible states that it can reach from the current state.
→ The nodes of a state space represent states, and the arcs connecting them represent actions.
→ A path is a set of states and the actions that link them in the state space.
→ A problem's soln is a node in the graph representing all possible states of the problem.
→ Most AI techniques based on it.

State Space Represen"
→ consists of identifying an Initial State (to begin) and a Goal State (final destn) and then following a specific sequence of actions (States).

State - AI problems can be represented as set of well-formed states. Initial, Goal & others generated by applying rules betn them.

Space - exhaustive collec" of all possible states

Search - techniques that moves from beginning state to desired state by applying valid rules while traversing the space of all possible states.

Search Tree - tree-like depic" of search issue.
↳ provides agent with description of all avl actions.

Transition Model - describes what each action does

Date
Page No.

Path cost — func" that assigns cost value to each path.

→ It is an activity sequence that connects beginning node to the end node

→ Optimal sol" is the one with lowest cost among all alternatives.

Search Algorithms in AI

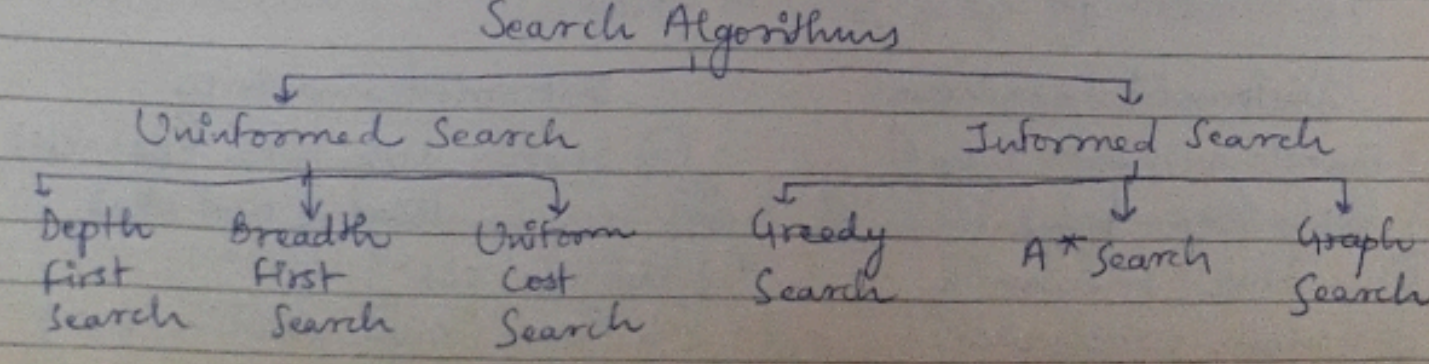
A search problem consits of -

* A State space — Set of all possible actions where you can be
* A Start space — state from where search begins
* A Goal Test — a func" that looks at the current state returns whether or not it is the goal state.

→ The Sol" to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state.

→ This plan is achieved through search algos.

Types of Search Algos -



↳ Gives more optimal soln than informed search

Date:
Page No:

Uninformed / Blind Search

→ Here the search algos have no additional info on the goal node other than the one provided in the problem definition.
→ The plans to reach the goal state from the start state differ only by the order and/or length of actions.
→ It can only generate the successors and differentiate betn the goal state and non goal state.

Each algorithm in this will have -
* A problem graph, containing start node S and goal node G.
* A strategy, describing the manner in which the graph will be traversed to get to G.
* A fringe, which is a data structure used to store all possible states (nodes) that you can go from the current states.
* A tree, that results while traversing to the goal node.
* A soln plan, which is the sequence of nodes from S to G.

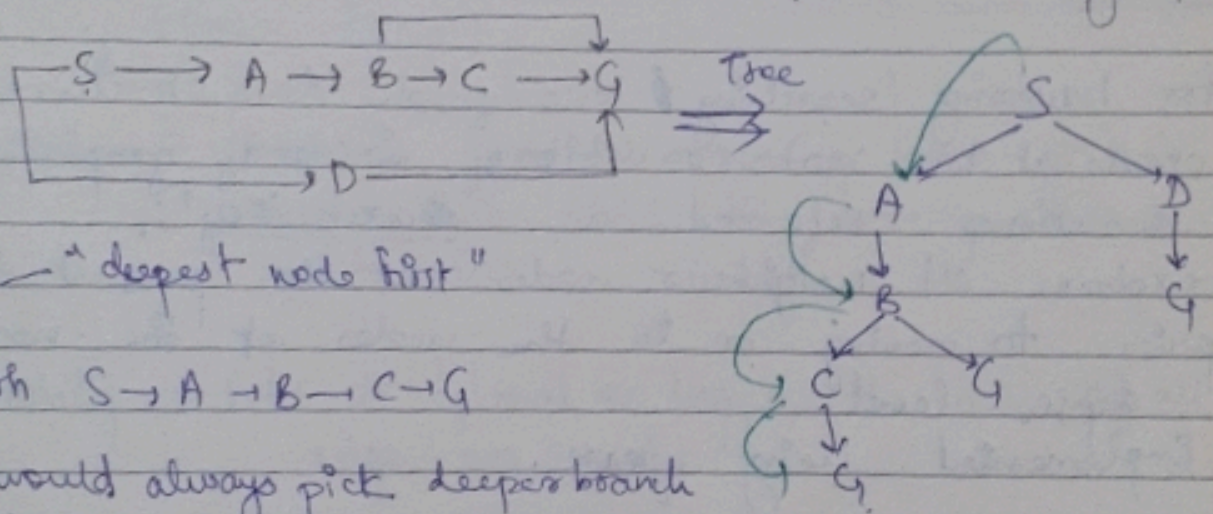
Uninformed Searching	Informed Searching
(1) Search without Info	(1) Search with Info (Heuristic)
(2) No knowledge	(2) Use knowledge to find steps
(3) Time consuming	(3) Quick soln
(4) More complexity	(4) Less Complexity (Time, Space)

Date
Page No.

Depth first Search

→ for traversing / searching tree or graph data structures.
→ starts at root node (arbitrary in case of graph)
→ & explores as far as possible along each branch before backtracking.
→ uses last-in-first-out and implemented using stack.

(Q) to move from node S to node G for following Graph



(Sol) "deepest node first"

Path S → A → B → C → G

! It would always pick deeper branch until it reaches the soln (or it runs out of nodes and goes to the next branch).

d = depth of search tree = no. of levels of search tree

n^i = no. of nodes in level

Time complexity = Equivalent to no. of nodes traversed in DFS.

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

Space complexity = Equivalent to how large can the fringe get

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

Date:
Page No:

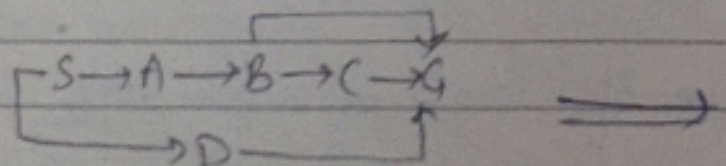
Completeness = DFS is complete if the search tree is finite, meaning for a given finite search tree, DFS will come up with a sol^n if it exits.

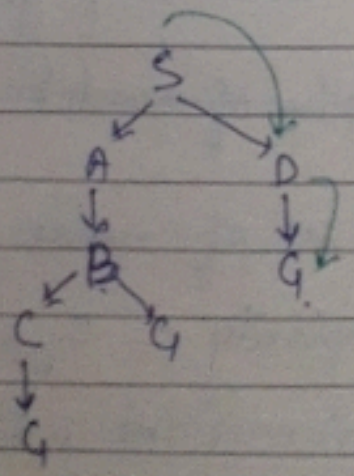
Optimality = DFS is not optimal, meaning the no. of steps in reaching the sol^n, or cost spent in reaching is high.

Breadth first Search

→ for traversing / searching tree / graph data structures
→ starts at tree root (or arbitrary in case of graph, sometimes referred as 'search key').
→ explores all neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.
→ implemented using queue.

(Q) move from node S to node G.





(Sol)
"shallowest node first"
→ always pick shallower branch until it reaches the sol^n (or it runs out of nodes, & goes to next branch).

Date:
Page No.

Path ⇒ $S \rightarrow D \rightarrow G$

s = depth of shallowest sol^n
n^i = no. of nodes in level
Time complexity = Equivalent to no. of nodes traversed in BFS until shallowest sol^n
$T(n) = 1 + n^2 + n^3 + \ldots\ldots + n^s = O(n^s)$

Space complexity = Equivalent to how large can the fringe get.
$S(n) = O(n^s)$

Completeness = BFS is complete, meaning for a given search tree, BFS will come up with a sol^n if it exists.

Optimality = BFS is optimal as long as the costs of all edges are equal.

Informed Search

→ Here algorithms have info on the goal state, which helps in more efficient searching.
→ This info is obtained by something called a heuristic.

Search Heuristics – A heuristic is a func^n that estimates how close a state is to the goal state.
Ex → Euclidean distance (lesser the distance, closer the goal.)

→ Different heuristics are used in different informed algos.

Heuristic function – If there are no specific answers to a problem, or the time required to find one is too great, a heuristic func^n is used to solve the problem.
The aim is to find a quicker or more approximate answer, even if it is not ideal.
→ Or utilizing a heuristic means trading accuracy for speed.

→ The distance formula is an excellent option if one needed a heuristic func^y to assess how close a loc^n in a 2-dimensional space was to the objective point.

Properties – * Admissable Condition – If an algorithm produces an optimal result, it is considered admissable.

$$h(n) <= h^*(n)$$

($h(n)$ → heuristic cost, $h^*(n)$ → estimated cost)

* Completeness – If an algo ends with a sol^n, it is considered complete.

* Dominance Property – If A1 and A2 are 2 heuristic algos and have h1 and h2 heuristic functions, resp, then A1 will dominate A2 if h1 is superior to h2 for all possible values of node n.

* Optimality Property – If an algo is thorough, allowable, and dominates the other algos,

that'll be the optimal one and will unquestionably produce an optimal result.

Different Categories of Heuristic Search Techniques (HST)

Direct H.S.T
can also be called as blind control strategy / blind search / uninformed search.
→ utilize arbitrary sequencing of operations & look for soln throughout entire state space.

Weak HST
can also be called as Heuristic control strategy / informed search / Heuristic Search
→ successful when used on appropriate tasks & require domain-specific knowledge.
→ users req additional info to compute preferences across child nodes
→ A heuristic func is connected to each node

Hill Climbing Search

→ heuristic search
→ Solves the problems where we need to maximize / minimize a given real func by choosing values from the given inputs.

Features — ① Variant of generating & test algo.
* Generate possible soln's.
* Test to see if this is the expected soln.
* If soln is found quit else go to step 1.

called so as it takes feedback from the test procedure.

Date:
Page No:

→ Then this feedback is used by the generator in deciding the next move in search space.

(2) Uses Greedy approach
→ at any pt in state space, the search moves in that direc^n only which optimizes the cost of func^n with the hope of finding the optimal sol^n at the end.

Hill Climbing / Greedy local Search

→ local search algo. which continuosly moves in the direction of ↑ value to find the best sol^n to problem. It terminates when it reaches a peak value where no neighbour has a higher value.
→ only looks to its good immediate neighbour state and not beyond that.
→ Its nodes has 2 components which are. state and value.

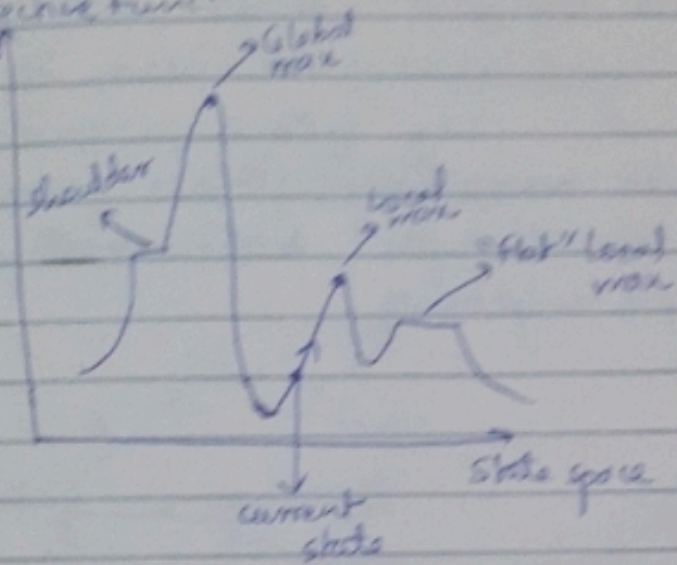
Features —
* Generate & Test variant — produce feedback which helps to decide which direction to move in the search space.

* Greedy Approach — moves in the direc^n which optimizes the cost.

* No backtracking — does not backtrack the search space, as it does not remember the previous states.

Date
Page No.

State - Space Diagram (various states of algo & Objective fun")



→ If fun" on y-axis is cost fun", the goal of search is to find the global min and local minimum.

→ If the fun" on Y-axis is objective fun", then the goal of search is to find global max and local max.

Local max - state better than its neighbour states.
Global max - best possible state of state space landscape. It has highest value of objective fun"
Current state - state where agent is currently present.
Flat local max - flat space where all neighbour states of current states have the same value.
Shoulder - plateau region which has an uphill edge.

Types of Hill Climbing Algo.

Simple HC	Steepest-Ascent HC	Stochastic HC
→ only evalues neighbour node state at a time and selects the 1st one which optimizes current cost and set	→ examines all neighbouring nodes of current state and selects one neighbour	→ does not examine for all its neighbour before leaving. Rather selects one neighbour node

Simple HC	Steepest-Ascent HC	Stochastic HC
it as a current state → It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state → less time consuming → less optimal sol^n & sol^n not guaranteed	node which is closest to the goal state. → consumes more time as it searches for multiple neighbours	at random & decides whether to choose it as a current state or examine another state.

Date:
Page No.

Problems

* Local max - here all neighbouring state have value worse than current state. Algo will not move to worst state & terminate itself. A better sol^n may exist
Sol^n → backtracking (maintain list of visited states)

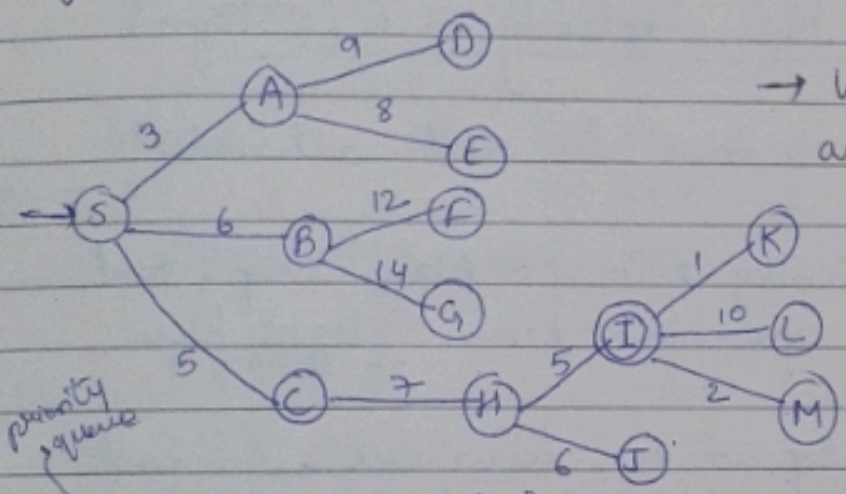
* Plateau - here all neighbour have same value. Hence not possible to select best direct^n
sol^n → Randomly select a far away state from current state.

* Ridge - Any pt on ridge can look like peak because movement in all possible directions is downward.
sol^n → move in several directions at once.

Date
Page No.

Best First Search (Informed search)

→ It uses an evaluation func" to decide which adjacent is most promising and then explore.
→ uses a priority queue or heap to store the costs of nodes that have the lowest evaluation func" values.



priority queue

→ We start from source "S" and search for goal "I" using given costs and BFS

* pq initially contains S
→ we remove S from pq and process unvisited neighbours of S to pq
→ pq now contains {A, C, B} (C is put before B because C has lesser cost)

* We remove A from pq & process unvisited neighbours of A to pq
→ pq now contains { C, B, E, D }

* We remove C from pq and process unvisited neighbours of C to pq.
→ pq now contains { B, H, E, D }

* We remove B from pq & process unvisited neighbours of B to pq.

Date:
Page No:

→ pq now contains { H, E, D, F, G }

* We remove H from pq.

* Since our goal "I" is a neighbour of H, we return.

→ ~~worst~~ worst-case time complexity = O(n*logn)

→ In worst case, we may have to visit all nodes before we reach goal.

→ pq is implemented using Min (or Max) Heap, and insert and remove operations take O(logn) time.

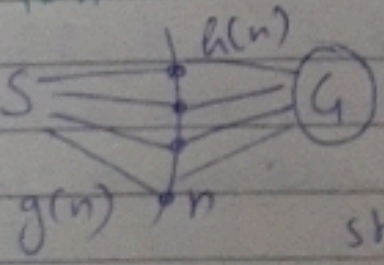
→ performance of algo depends on how well the cost or evaluation func" is designed.

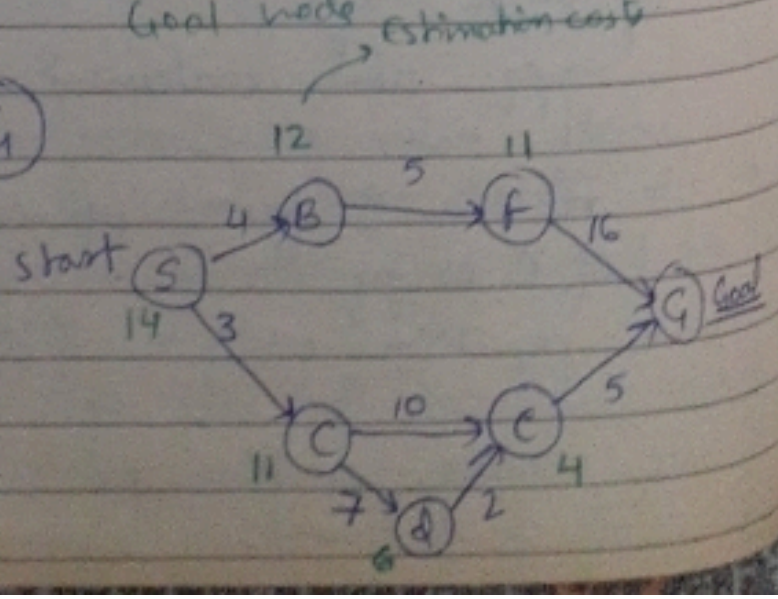
A* Search Algorithm (Informed Searching)

admissible

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

- g(N): Actual Cost from start node to n
- h(N): Estimation cost from n to Goal node





Date
Page No.

heuristic value

$f(S) = 0 + 14 = 14$

$S \to B$ or $S \to C$ ✓ we will choose $S \to C$ as it is less

$4+12 = 16$ $\quad$ $3+11 = 14$

$SC \to e$ or $SC \to d$

$3+10+4 = 17$ $\quad$ $3+7+6 = 16$ ✓ As $S \to B$ & $SB \to d$ both have similar costs so we will explore both

$SB \to f$ or $SB \to e$

$20 = 5+4+11$ $\quad$ $4+12+4 = 20$

Hence these values are greater than $SC \to d$ so we will explore it

$Scd \to e$ $\quad$ Hence 16 is lower than $SB \to f$ and $SB \to e$

$3+7+2+4 = 16$

$Scde \to G$

$12 + 5 + 0 = \boxed{17}$ ✓ (min value)

(By default heuristic value of Goal state is 0)

To check other path

$SBf \to G$

$9 + 16 + 0 = 25$ ✗

→ This algorithm finds the shortest path bet^n a starting node and a goal node in a graph or a grid.

→ The algo works by maintaining two lists: an open list and a closed list.

→ Open list contains nodes that have been visited

Date:
Page No:

but not yet been expanded
→ closed list contains nodes that have already been expanded.

→ To evaluate which node to expand next, the algo calculates a heuristic func^n that estimates the distance betn the current node and the goal node. (which is Euclidean distance or Manhattan distance).

→ Algo choose node with lowest estimated distance as the next node to expand. It then generates all the neighbouring nodes of this node and calculates their estimated distances. These nodes are added to the open list.

→ Algo repeats this process until it reaches the goal node. Once the goal node is reached it backtracks to find the shortest path from the starting node to the goal node.

→ Used widely in robotics, video games, route planning.

AO* Algorithm

→ extension of A* algo.
→ designed to handle problems where cost of moving betn nodes is not known in advance and may vary during the search proces.

Date:
Page No:

→ works by maintaining a set of alternative paths from the starting node to the goal node.
→ Each path has an estimated cost, which is calculated based on the heuristic func^n and the actual cost of the path.
→ Algo starts by initializing a single path from starting node to goal node, with an estimated cost equal to the heuristic func^n
→ Algo then repeatedly selects the path with lowest estimated cost and expands it by generating its neighbouring nodes.

→ For each new node, the algo calculates its actual cost and updates the estimated cost of the path that includes the new node.

→ If the new path has a lower estimated cost than any of the existing paths to the same node, it is added to the set of alternative paths.

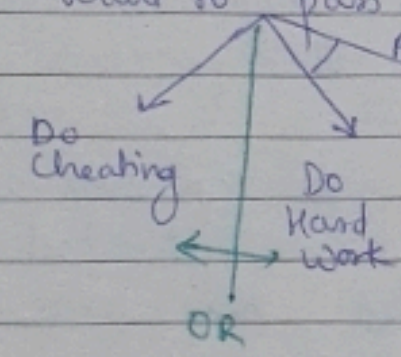
→ Algo continues until it finds a path to the goal node with an estimated cost that is lower than the estimated cost of all other path to the goal node. This path is then returned as optimal sol^n.

→ useful where cost of moving bet^n nodes is not known in advance or changes during process.

Date:
Page No:

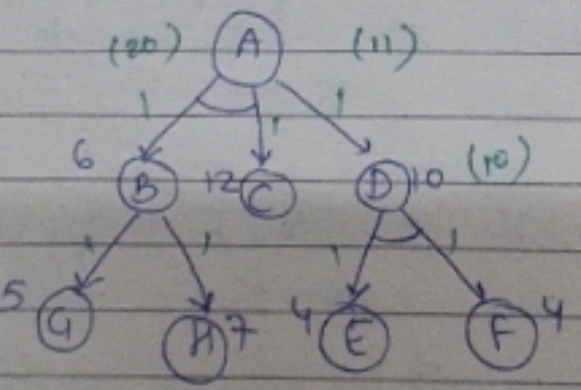
→ also useful where multiple optimal paths exist, as it can find and store all possible paths to the goal node.
→ however it is computationally expensive, especially in large graphs / grids. and not suitable for real-time applications.

AO* (AND/OR) → Problem Decomposition (Breakdown into smaller pieces)



AND ⇒ condition is compulsory which
OR is based on several decisions

→ AO* does not explore all the sol^n paths once it got a sol^n.



cost of every edge is 1

Did not explore this part.

A → BC
6 + 12 + 2 = 20

A → D
10 + 1 = 11 ✓

D → EF
4 + 4 + 2 = 10

→ Does not guarantee to find optimal path but provide a good path quickly and then improve upon it as more info becomes avl.

Date
Page No.

Constraint Satisfaction Problems

→ A CSP is a computational problem in AI where goal is to find a solⁿ that satisfies a set of constraints.

→ Here problem is usually defined by a set of variables, a domain of values for each variable, a a set of constraints that restrict the allowable combinations of values for the variables.

→ Here variable can represent anything from scheduling tasks to assigning resources to optimizing prodⁿ lines. The domain of values for each variable represents the set of possible values that the variable can take on.

→ Constraints define the relationships betⁿ variables and restrict allowable combinations of values for the variables.

→ Solving it involves finding a combⁿ of values for the variables that satisfies all the constraints. This can be done by using various search algos, such as backtracking, forward checking, and constraint propagation. These algos search by exploring the space of possible assignments and pruning away any assignments that violate the constraints.

Date:
Page No:

→ used in planning, scheduling, and optimizn.
→ used in NLP, computer vision, problems where it involves finding a soln that satisfies a set of constraints.

Game Tree

→ graphical represent^n of possible moves & outcomes of a game.
→ tree-like structure where each node represents a game state, and each edge represents a possible move from one state to another
→ starts with initial state of the game, and each subsequent level of the tree represents all possible moves that can be made from the previous state.
→ The leaves represents possible outcomes of the game, such as winning, losing or drawing.
→ used in game playing AI algos, mini-max and alpha-beta pruning.
These algo use it to search for best possible move to make at each turn, based on predicted outcomes of each possible move.
→ challenges include exponential growth in the no. of possible states as the game progresses. This can lead to an explosion in the size of game tree making difficult to search best move in a reasonable amt of time.

→ Hence pruning of branches used and are unlikely to lead to a good outcome.

Mini-Max Algo

→ backtracking algo used in decision making & game theory to find optimal move for a player, assuming that your opponent also plays optimally.

→ used in 2 player turn games (tic-tac-toe, chess)

→ Here 2 players are called maximizer and minimizer.

→ maximizer tries to get highest score possible while minimizer tries to do opp and get lowest score possible.

→ Every board state has a value associated with it. In a given state if maximizer has upper hand then, the score of the board will tend to be some positive value.
If the ~~minimizer~~ minimizer has the upper hand in that board state then it will tend to be some -ve value.

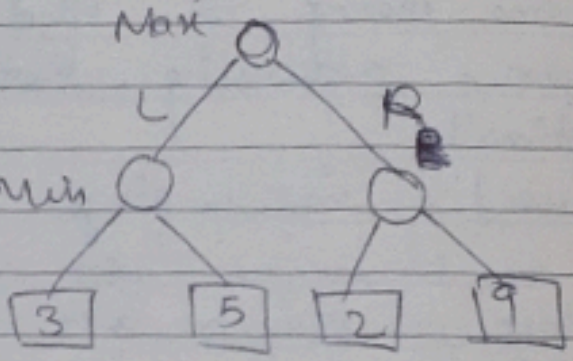
→ The values of the board are calculated by some heuristics which are unique for every type of game.

Date:
Page No:

Example → game → 4 final states & paths to reach final state are from root to 4 leaves of a perfect binary tree.

Assume you are maximizing player and get 1st chance to move i.e you are at root and opp at next level. Find move you will take optimally.

→ Since it is backtracking algo hence it will try all possible moves then backtracks and makes a decision.

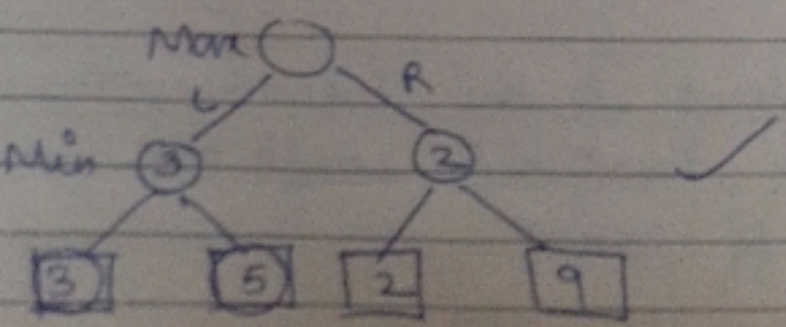


★ Max goes LEFT: It is now the min turn. The min now has a choice bet 3 & 5. Being min it will choose 3, which is least.

★ Max goes RIGHT: It is now min turn. Min now ~~choose~~ has a choice bet 2 & 9. He will choose 2 which is least.

→ Being max you will choose 3 which is larger. Hence optimal move for max is to go LEFT and optimal value is 3.

Now tree →



Alpha-Beta Pruning

→ optimization technique

→ modified / updated version of minimax algo.

→ to eliminate exponential states by pruning.

→ without checking each node of the game tree we can compute the correct minimax decision and it is known as pruning.

→ involves 2 threshold parameters Alpha & Beta for future expansion.

→ can be applied at any depth of a tree and sometimes it not only prunes the tree leaves but entire sub-tree.

Alpha → best (highest-value) choice we have found so far at any pt along the path of Maximizer. Initial value of alpha is $-\infty$.

Beta → best (lowest-value) choice we have found so far at any pt along the path of minimizer. The initial value of beta is $+\infty$.

→ returns the same move as the standard algo does, but it removes all the nodes which are not really affecting the final decision but making algo slow.

Date:
Page No:

condition for alpha-beta pruning →

$\boxed{\alpha >= \beta}$

→ max player only update value of α
→ min player only update value of β

→ while backtracking, node values will be passed to upper nodes instead of values of α & β.
→ We will only pass α, beta values to child nodes.

→ At each node algo evaluates posible moves & prunes branches that cannot lead to a better outcome for player.

→ The algo does this by comparing current alpha and beta values to the values of the nodes that have already been evaluated.
If current node's value exceeds the current beta value (in the case of max node) or falls below the current alpha value (in case of min node) the algo prunes that branch and does not evaluate any further nodes in that branch.

Date:
Page No:

Game of Chance

→ It refers to games where outcome is determined by random events, such as roll of dice or draw of cards.
AI techniques can be used to simulate and analyze these types of games.

→ Application include ~~bodi~~ building probabilistic models to predict the likelihood of certain outcomes.
Ex → in a game of poker AI can analyze the cards that have been played and betting patterns of the players to predict likelihood that a particular player has a strong hand.

→ Another app" include is to develop strategies for playing these games.

Ex → in a blackjack game AI could be trained to make decisions about when to hit or stand based on cards that have been dealt and dealer's up card, using reinforcement learning (reward based)

→ Also AI can be used to design new games of chance.
Ex → evolutionary algos can be used to generate new rules for games and to optimize the game mechanics to make them more engaging and fun to play