INDEX

NAME: Preuitha R STD.: SEC.: ROLL NO.: SUB.:

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Introduction		
		Introduction- Evolution of AI		
		State of Art		
		Different Types of Artificial Intelligence		
		Applications of AI		
		Subfields of AI		
		Intelligent Agents		
		Structure of Intelligent Agents		
		Environments		
		Problem Solving based on Searching		
		Introduction		
		State space Search		
		Uninformed search methods		
		Uniform cost Search		
		Breadth First Search		
		Depth First Search		
		Iterative deepening depth-first		
		Informed Search Methods		
		Best First Search		
		A* search		
		Depth limited Search		
		Local Search and Adversarial Search		
		Local Search Algorithms -		
		Hill climbing Search		
		Simulated annealing		
		Genetic Algorithm		
		Adversarial Search -		
		Game Trees & MinMax Evaluation		
		Elementary 2 player games - XO		
		MinMax and $\alpha\beta$ pruning		

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Logic and Reasoning		
		Introduction to logic & Reasoning		
		Propositional logic		
		First order logic		
		Inference in First Order logic		
		Unification		
		Forward Chaining		
		Backward Chaining		
		Resolution		
		Uncertain Knowledge & Reasoning		
		Quantifying uncertainity		
		Bayes Rule		
		Bayesian Belief Network		
		Approximate inference in BN		
		Planning		
		Classical Planning		
		Planning as State Space Search		
		Forward Search		
		Backward Search		
		Planning Graphs		
		Hierarchical Planning		
		Planning & acting in non-det. dom.		
		Sensorless planning		
		Multiagent Planning		
		Communicating, Perceiving and Acting		
		Communication		
		Fundamentals of Language		
		Probabilistic Language Processing		
		Information Retrieval		
		Information Extraction		
		Perception		
		Image Formation		
		Object Recognition		

ARTIFICIAL INTELLIGENCE

Intelligence =

Ability to interact with the world - speech, vision, motion, manipulation

Ability to model the world and reason about it.

Ability to learn and adapt.

AI: Branch dedicated to create intelligent machines that work like humans.

- Machine/System + Software + Internet connectivity

AI is design of intelligence in an artificial device.

John McCarthy, Alan Turing, Marvin Minsky, Allen Newell, Herbert A. Simon

History

1941 - Development of electronic computer. (solve linear equations) - John Atanasoff & Clifford Berry - ABC

1943 - McCulloch & Pits propose model of artificial neurons

1946 - John Machly & Pres Eckelt - digital computer - ENIAC

1956 - Minsky & Edmonds - first neural network computer - SNARC

- The Dartmouth conference (2 months by John McCarthy)

1958 - LISP language developed by McCarthy

1963 - Advanced Reasearch Projects for DOD.

Enthusiasm → Reality → Knowledge based systems

Expert system is interactive and reliable decision making system that uses both facts & heuristics to solve complex problems.

1970 - first expert system.

MYCIN - identify bacteria causing infections & recommend drugs.

DENDRAL - predict molecular structure

PXDES - predict type of lung cancer

CADET - identify cancer at early stages.

1972 - PROLOG language - facts & rules representation.

1990 onwards - Increase in computational power.

Note:

1966 - first chatbot - ELIZA

1972 - first intelligence robot - WABOT-1

Types

1. Limited memory - uses past experience + present data for decision making.
2. Reactive machines - do not have ability to hold memories - specific jobs.
3. self awareness - Super intelligent
4. Theory of mind - Socialite & understand emotions

Applications

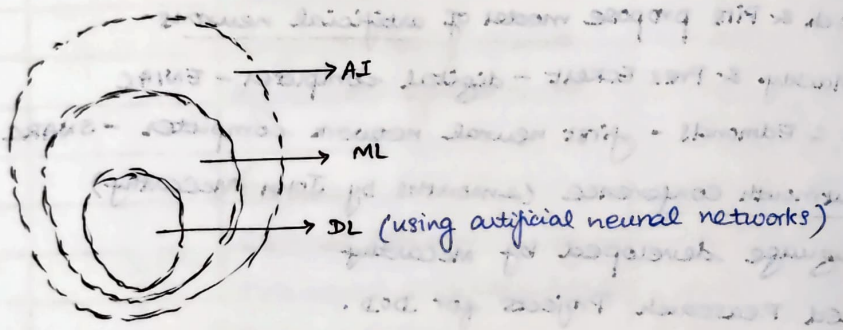
- Military
- Science
- Industrial
- Business
- Entertainment

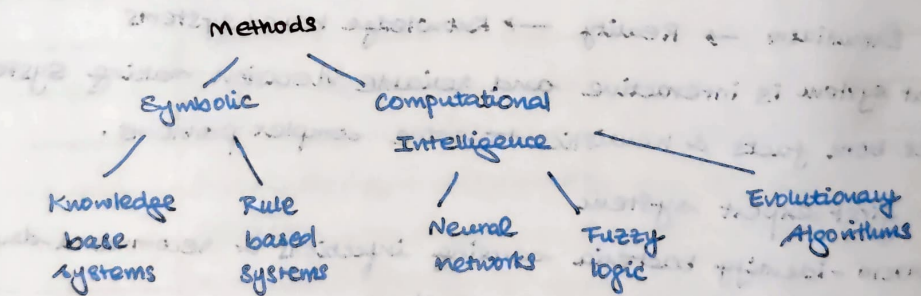
* Expert systems
* Natural Language Processing
* Speech Recognition
* Computer vision
* Robotics

Note:

Humans = Memory + Perception + Anticipation + Problem Solving + Decision making

Machines = Natural Language + computer vision + Dialog & Conversation + Domain Data + Machine learning





Human centered approach
|
hypothesis and experimental confirmation

vs

Rational approach
|
mathematics and engineering.

Turing Test - 1950

Types

- **Criteria**
  - Strong AI
  - Weak AI
- **Ability**
  - Narrow AI
  - General AI
  - Super AI
- **Functionality**
  - Reactive — doesn't learn
  - Limited memory - learn from history
  - Theory of mind - understands emotions
  - Self awareness - have own emotions

Agents

Perceiving its environment through sensors and acting upon that environment through actuators.

(Sensors) Agent ← perceive ← Env. ⇒ Have goals

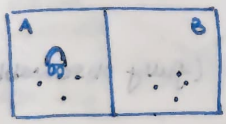
(effectors) → take action →

Autonomous agent

Agent decides autonomously which action to take in the current situation to maximize progress towards its goal.

Peyormance (criterion) measure

eg:



Agent function tabulation

Percept seq.	Action
[A, clean]	Right
[A, dirty]	Suck
[B, clean]	Left

Rational agent

For each possible percept sequence, a rational agent should select an action that is expected to maximize its peyormance measure.

Intelligent agent

Must sense

Must act

Must be autonomous

Must be rational

PEAS

- Performance measure
- Environment
- Actuators
- Sensors

Agent Environment

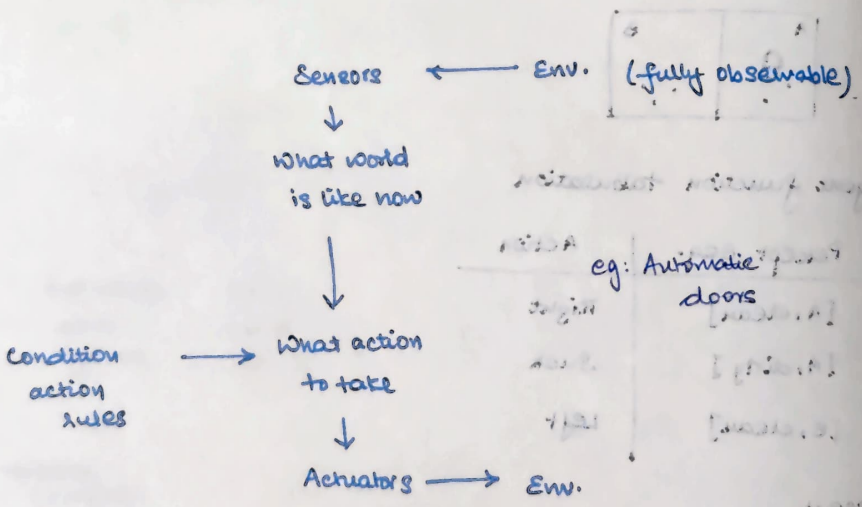
- Fully observable vs Partially observable
  - access to complete state of env. at each point in time.
- Deterministic vs Stochastic
  - next state depends on current state & action
- Episodic vs Sequential
  - atomic episodes each independent
- Static vs Dynamic
  - env. is unchanged while agent is deliberating.
- Discrete vs Continuous
  - limited no. of percepts and actions
- Single vs Multiagent
  - only one agent

Classes of Intelligent Agents

① Simple Reflex Agents

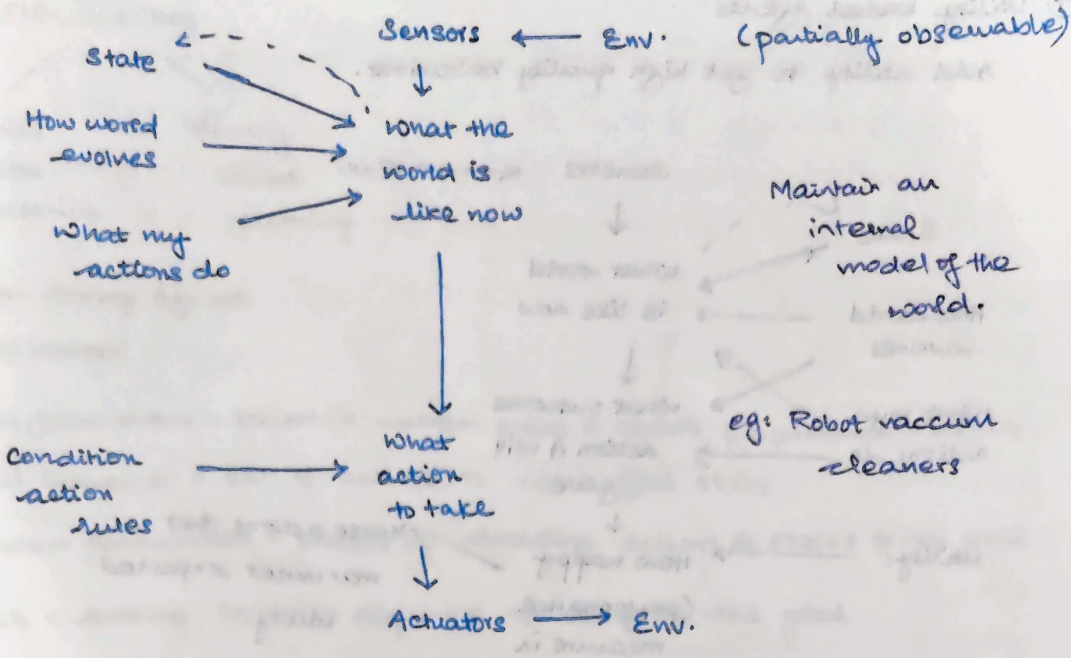
Act only on the basis of the current percept.

if condition then action



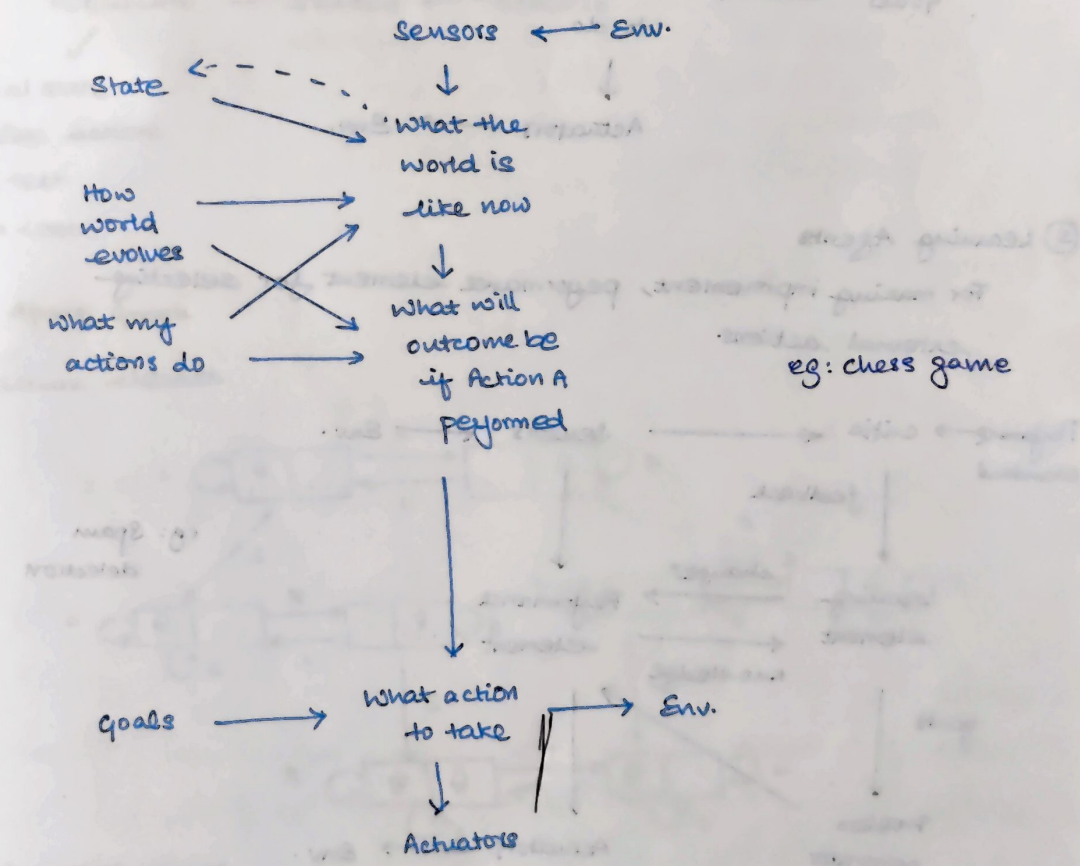
② Model based Reflex Agents

Keeps track of current state describing part of the world which cannot be seen.



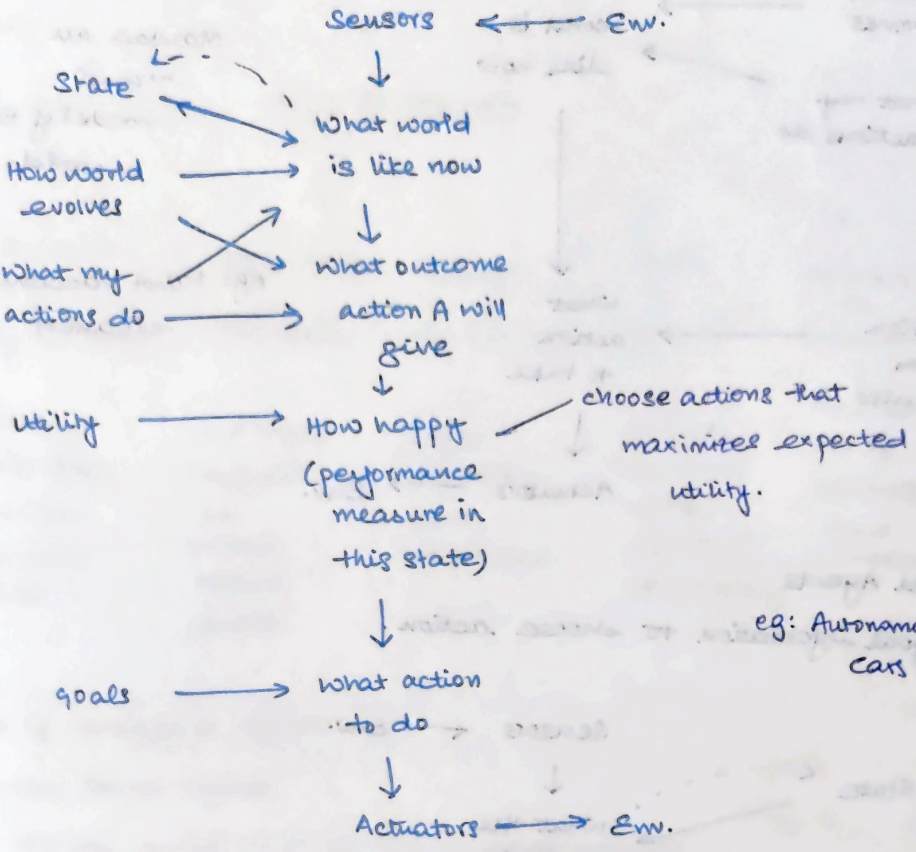
③ Goal based Agents

use goal information to choose action.



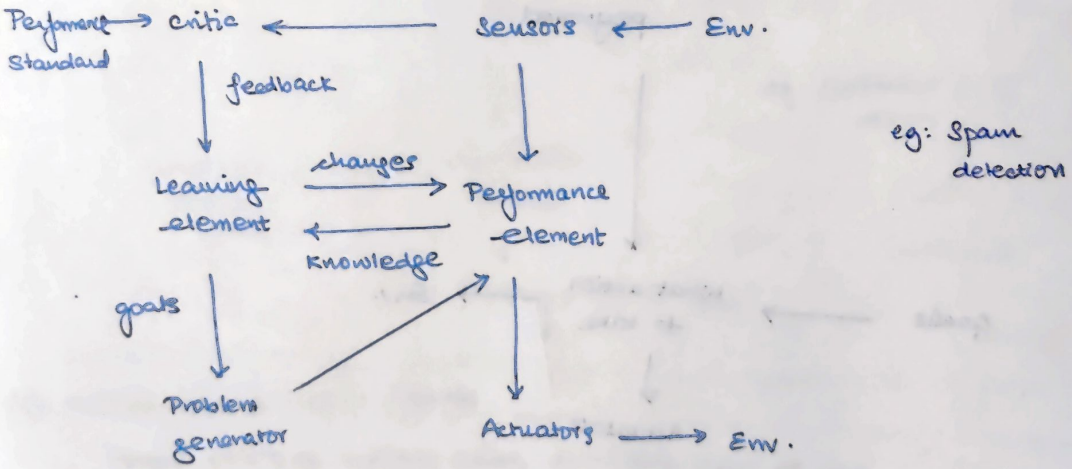
④ Utility based agents

Add utility to get high quality behaviour.



⑤ Learning Agents

For making improvement, peformance element for selecting external actions.



PROBLEM SOLVING

- Model based reasoning
- Memory based reasoning

Problem Solving Agents

(Goal based)

1. Goal formulation - based on current state & agent's performance measure.
2. Goal sequence - set of actions to reach goal state.
3. Problem formulation - process of deciding actions & states given goal.

Search - looking for best sequence to achieve the goal.

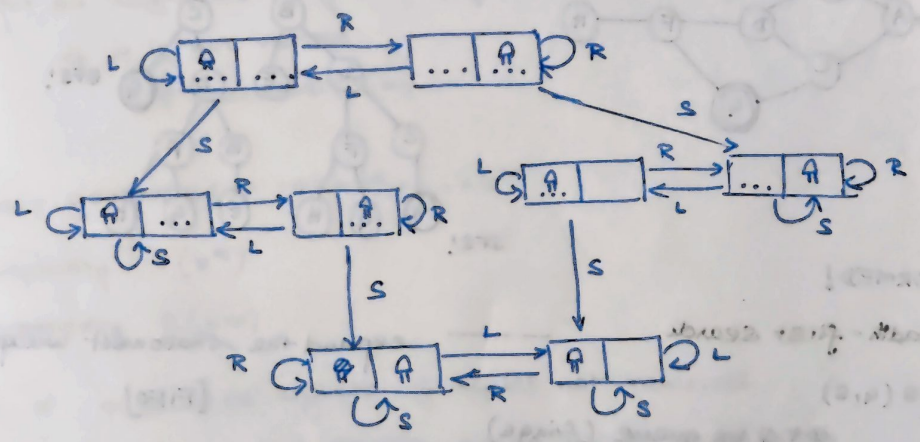
4. Search algorithm takes problem and produces action sequence.
5. Execution - actions recommended carried out.

FORMULATE — SEARCH — EXECUTE

- Initial state
- Possible actions
- Goal test
- Path cost

State Space Graph

eg: Vaccum cleaner



8 possible states

Initial state: any

Possible actions: L, R, S

Goal test: Both rooms clean

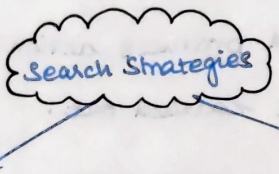
Path cost: 1 per action

General Search Algo:

- Initialize search tree with initial state as root node.
- Until queue is empty
  check if goal state if not expand current state by generating successors and adding them to search tree as leaves.
- Goal found = SUCCESS
  else = FAILURE

Strategies criteria:

1. completeness - always find solution when there is one.
2. Time complexity
3. Space complexity
4. Optimality

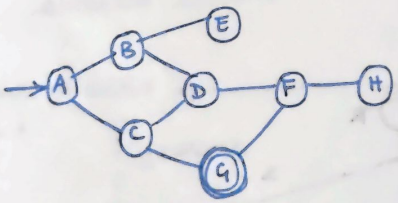


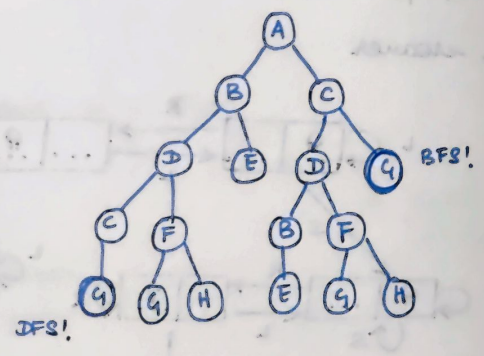
Blind Search/ Uninformed

use ONLY info. in problem definition.

Heuristic Search/ Informed

State space (Graph) → Search Tree





UNINFORMED!

① Breadth-first search ——— expand the shallowest unexpanded node [FIFO]

```
BFS(G,S)
    let Q be queue (fringe)
    Q.enque(s)
    mark s as visited
    while (!Q.empty):
        v = Q.deque(), if v==goal then return path from s to node.
        for all neighbours w of v in graph G:
            if w not visited ↙        ↘ if w==goal
                Q.enque(w)              return path s to w.
                mark w as visited
```

complete ✓

optimal ✓ Search goal node even in lowest level.

No. of nodes generated = $1+b+b^2+\ldots+b^d$

b = branching factor

d = depth of shallowest node goal

m = maximal depth of leaf node

Time complexity $O(b^d)$

Space complexity $O(b^d)$

} on the higher end.

② Depth-first search

```
DFS(G, S):                                    [LIFO]
    let S be stack
    S.push(s)
    mark s as visited
    while (S is not empty):
        v = S.top()
        S.pop()
        for all neighbours w of v in Graph G:
            if w is not visited:        ↙ ↘ if w == goal
                S.push(w)                   return path from s to w.
                mark w as visited
```

```
DFS-Recursive(G, S):
    mark s as visited
    for all neighbours w of s in Graph G:
        if w is not visited:
            DFS-recursive(G, w)
```

No. of nodes generated = $1+b+b^2+\ldots+b^m$

Time complexity $O(b^m)$

Space complexity $O(bm)$

complete ✗ m could be infinite, if cycles not removed.

optimal ✗ finds leftmost solution regardless of depth/cost.

(3) Uniform cost search — Dijkstra Algo variant

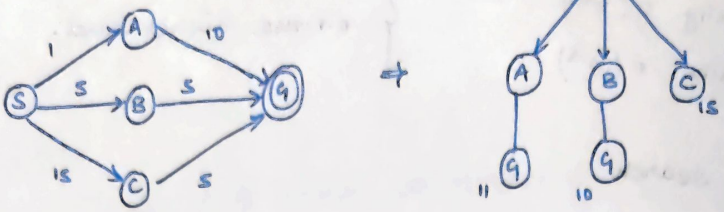
Works by expanding the node n with the lowest path cost.

If step cost are equal then - BFS.

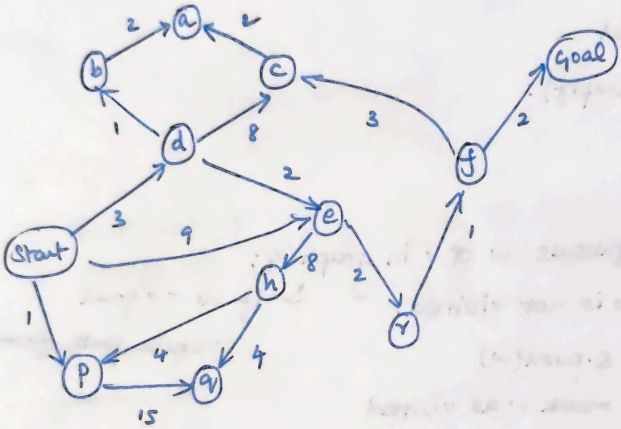
$g(N) = \Sigma$ costs of arcs

— gives lowest cost path to goal node

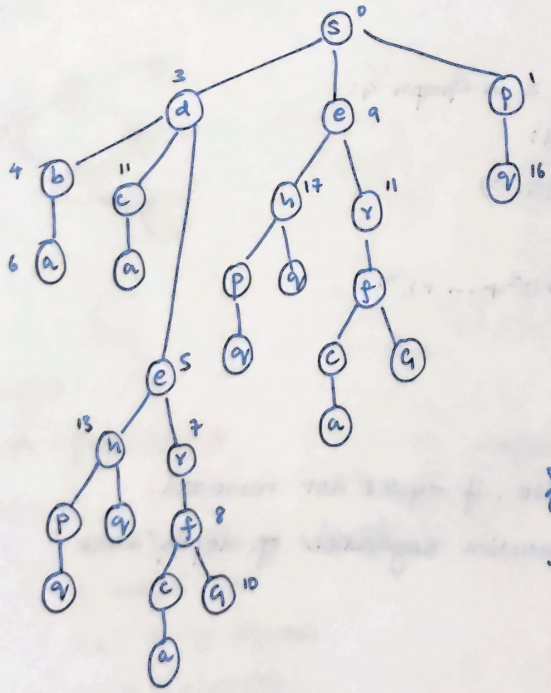
eg:



eg:



State space graph:



{S p d b e a r f e q}

expansion order.

```
UCS (start, goal):
    frontier = PriorityQueue()
    frontier.push ((start, 0))
    explored = set()

    while not frontier.isEmpty():
        curr_node, curr_cost = frontier.pop()
        if curr_node == goal:
            return reconstruct_path (curr_node)
        explored.add (curr_node)

        for neighbour, step_cost in neighbors (curr_node):
            if neighbour not in explored:
                total_cost = curr_cost + step_cost
                frontier.push ((neighbor, total_cost))

    return "Goal not found"

reconstruct_path (goal):
    path = [goal]
    curr_node = goal
    while curr_node.parent is not None:
        path.append (curr_node.parent)
        curr_node = curr_node.parent
    return reverse (path)
```

Complete ✓ when step cost > ε (+ve no.)

Optimal ✓ Nodes expanded in increasing order of path cost

Time complexity $O(b^{C^*/\varepsilon})$

Space complexity $O(b^{C^*/\varepsilon})$

C^* = cost of optimal solution

(4) Depth Limited Search

DFS with a depth limit L.

* choice of depth parameter is important.

Too deep - waste time / space

Too shallow - Might not reach goal.

Completeness ✗ when depth limit < depth of shallowest solution in search space.

optimal ✗ does not find shortest path

Space complexity $O(bl)$

Time complexity $O(b^l)$

b = branching factor

l = max depth - depth limit

(5) Iterative Deepening Search

Provides best of both BFS and DFS.

By iteratively increasing the depth limit with each iteration until d is reached (depth of shallowest goal node).

Do DFS for path length = 1 → (No goal found) → Do DFS for path of length = 2 → ...

complete ✓

Optimal ✓ step cost = 1

Time complexity = $(d+1)(1) + db + (d-1)b^2 + (d-1)b^3 + \cdots + (1)b^d = O(b^d)$

Space complexity = $O(bd)$

(6) Bidirectional Search

Finds smallest path from source to goal.

2 simultaneous searches

Forward search $S \rightarrow G$

Backward search $G \rightarrow S$

Search terminates when 2 subgraphs intersect.

complete ✓

Optimal ✓ if search space is evenly distributed & b is low.

Time complexity $O(b^d)$

Space complexity $O(b^d)$

INFORMED!

Informed/ heuristic search methods use problem specific knowledge, more efficient.

Heuristics - use heuristics to identify most promising search path.

eg: For an 8 puzzle problem

i) Heuristic based on number of moves

ii) Heuristic based on Manhattan distance

2 8 3	1 2 3	i) 5
1 6 4	8 4	ii) 1+2+1+1+1 = 6
7 5	7 6 5	
Current state	Goal state	

① Best First Search

Node selected for expansion based on $f(n)$: Evaluation function

$f(n)$ provides an estimate for the total cost.

Expand the node n with smallest $f(n)$.

Does not consider cost of reaching current node from the start state.

It focuses solely on the estimated cost from current node to goal state.

A. Greedy Best First Search

Specific variant of best first search.

Selectes the node to expand based on estimated cost to reach goal state from current node.

$f(n)$ = Estimate of cost from n to goal

Complete ✗ Can get stuck in loops.

Time Complexity $O(b^m)$

Space Complexity $O(b^m)$ keeps all nodes in memory

Optimal ✗ Can get stuck in local optima / find suboptimal solutions.

+ Can be used in applications where finding a solution quickly is more important than finding the optimal solution.

B. A* Search

- Hart, Nilsson, Rafael, 1968

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

$g(n)$ = sum of edge costs from start to n

$h(n)$ = estimate of lowest cost path from n to goal

Algorithm:

```
OPEN   // set of nodes to be evaluated
CLOSED // set of nodes already evaluated
add the start node to OPEN

loop
    curr = node in OPEN with lowest f-cost
    remove curr from open
    add curr to CLOSED

    if curr is the target node
        return // path found

    for each neighbour of the current node
        if neighbour is not traversable / neighbour is CLOSED
            skip to next neighbour
        if new path to neighbour is shorter / neighbour not in OPEN
            set f-cost of neighbour
            set parent of neighbour to current
            if neighbour is not in OPEN
                add neighbour to OPEN.
```

(1) Simple reflex, Model based reflex, Goal based, Utility based, Learning agents

To develop - route finder for cannon Missile to attack enemies.

Efficiently navigate through varying terrains & enemy positions to reach the desired targets.

Simple reflex → Not suitable for complex tasks like route finding in dynamic environments, where decisions need to be made based on both current & anticipated states.

Model based reflex → Can be used for route finding considering the consequences of actions to plan accordingly.
NOT suitable since creating accurate model of env. & updating in real time can be challenging.

Goal based → Effective for route finding - focus on reach destination considering factors like terrain, enemy positions and obstacles. By continuously updating goals and evaluating paths, it can adapt to changing conditions.

Utility based → Make decisions that optimize a combination of objectives balancing the importance of reaching the target quickly with minimal risk.

Learning agent → Trial and error, cannot afford this in a warfare.

(2) Fully observable — if user data (browsing history, purchase history, demographics, preferences) & product info (images, description, reviews, availability), market dynamics, platform analytics are available.

Stochastic — outcomes are influenced by both randomness & actions of the agent

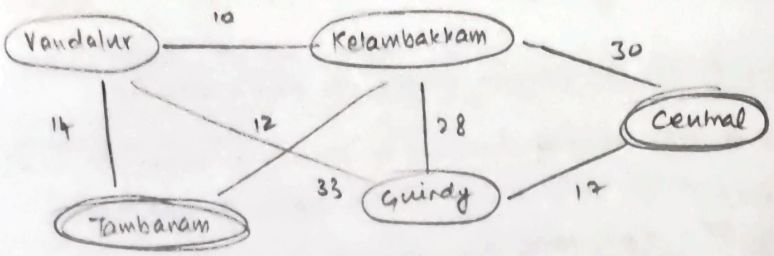
Sequential — Requires agent to consider long term consequences of actions and maintain a memory of past interactions.

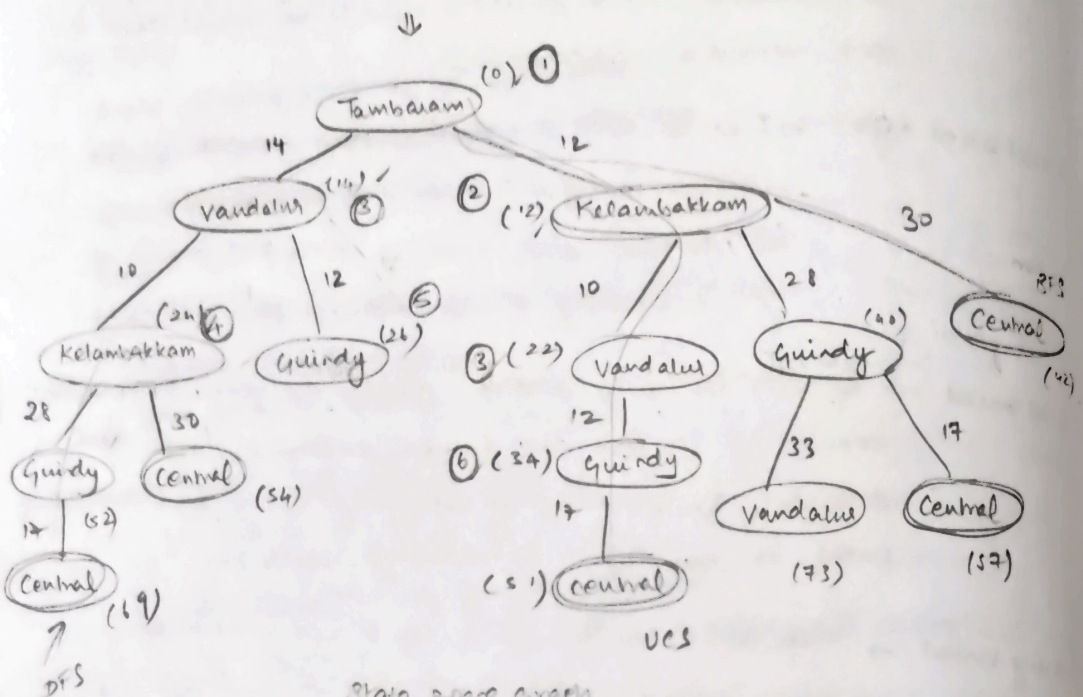
Dynamic — Frequent changes in product inventories, pricing strategies and customer preferences.

Continuous — infinite no. of percepts and actions.

Multiagent — customers, sellers, platform. Each agent has its own objectives, preferences and constraints leading to complex interactions and decision making process.

3





State Space Graph

DFS

T — V — K — G — C

14 + 10 + 28 + 17 = 69

BFS

T — K — C

12 + 30 ✓ Bell!

= 42

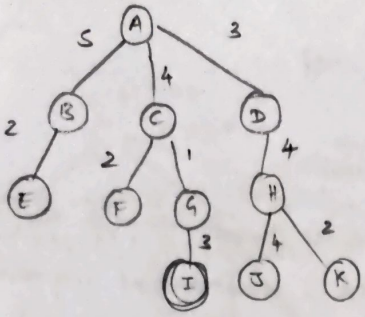
UCS

T — K — V — G — C

12 + 10 + 12 + 17

= 51

(4)

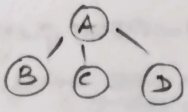


DFS: A-B-E-C-F-G-I

Iterative DS: $l=0$

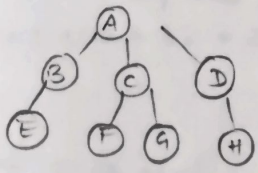
(A) A

$l=1$



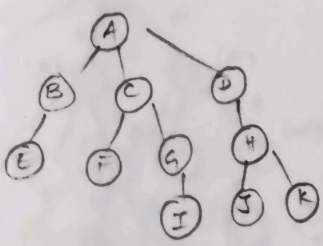
A-B-C-D

$l=2$



A-B-E-C-F-G-D-H

$l=3$



A-B-E-C-F-G-I

(same as DFS)

Benefit over BFS → Improved memory efficiency

IFS only keeps track of current path being explored.

Downside over BFS → Involve redundant work, revisiting same states across multiple iterations.

(5) Greedy best first search – faster

A* search – optimal with minimum cost.

---

(1) Learning agent for social media since trends constantly evolving, pattern recognition, personalization, continuous improvement.

Env → Fully observable (if all data avail.) - rarely the case, so partially observable

Stochastic - users may interact in unpredictable ways

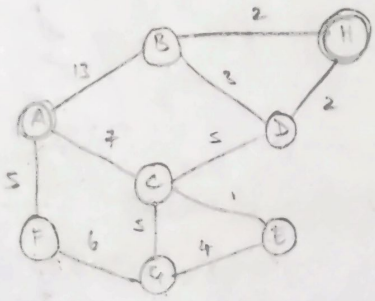
Sequential

Dynamic

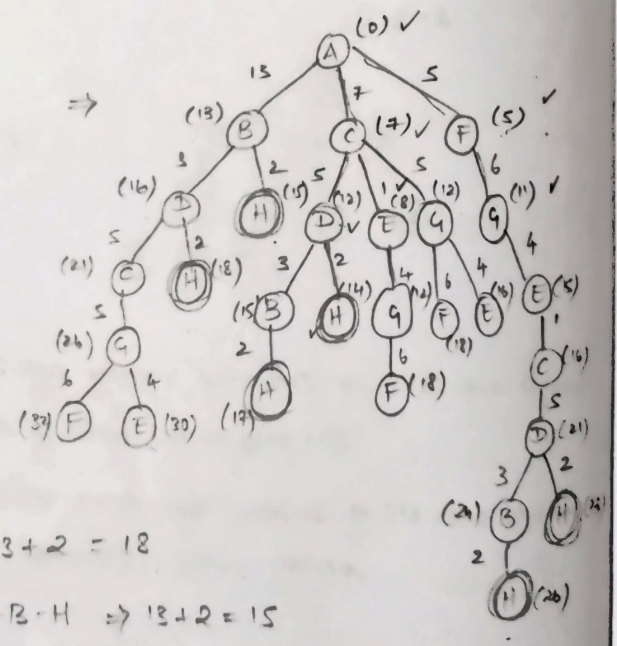
Continuous

Multiagent - users, content creators, advertisers, platform. The AI system must model and optimize these interactions to effectively identify trends & user requirements while balancing conflicting objectives and priorities.

(2)



State space graph



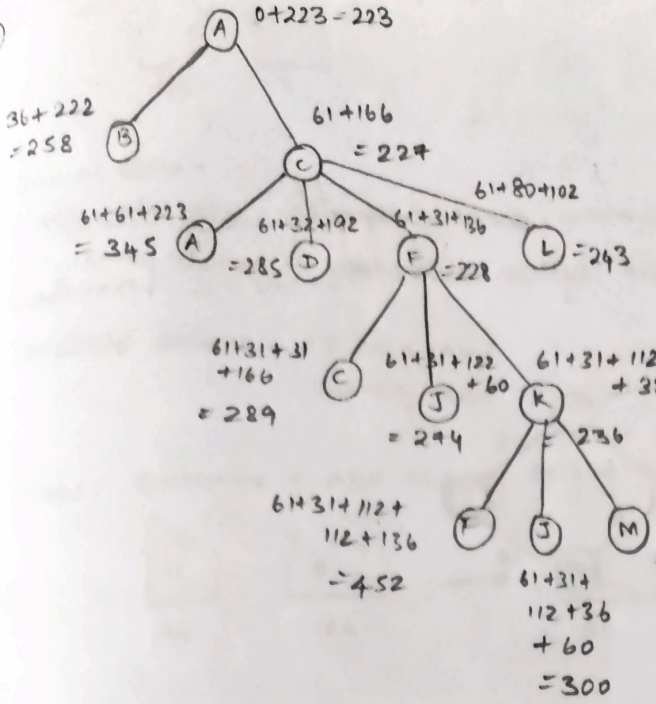
DFS → A-B-D-H ⇒ 13+3+2 = 18

Iterative DS → level 2 : A-B-H ⇒ 13+2 = 15

UCS →

Sequence : A, F, C, E, G, D, H : A-C-D-H ⇒ 7+5+2 = 14

Best : UCS



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

$g(n)$ = sum of edge cost from start to n

$h(n)$ = estimate of lowest cost path from n to goal. Straight line distance to M.

A, C, F, K → order of expansion.

A-C-F-K-M

61+31+112+32 = 236 //

Open list

~~A~~

~~C~~

~~B~~

~~F~~

~~L~~

~~B~~

~~D~~

~~A~~

~~K~~

~~L~~

~~A~~

~~B~~

~~J~~

~~C~~

M

L

A

B

J

D

C

J

F

---

① Model based reflex agent - water molecule has well understood chemical properties and molecular interactions. can leverage the domain knowledge to analyse the structure and behaviour of water molecules in liquid form. By maintaining an internal model of chemical bonds, molecular geometry, intermolecular forces, the agent can simulate the behaviour of water molecules in various environmental conditions and predict their structural configurations. Predefined rules - VBT, MOT to interpret input data and generate predictions of molecular structure of liquid water.

(2) Stochastic

Partially observable

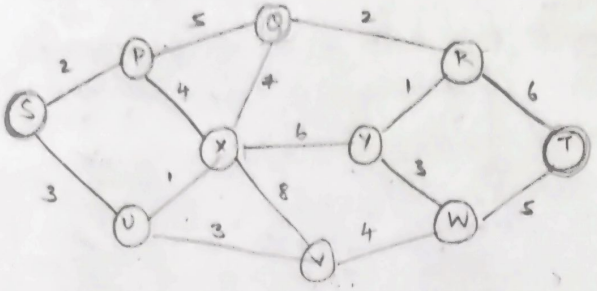
Episodic

Discrete

Dynamic - lighting, subject movement, scene composition, user interaction

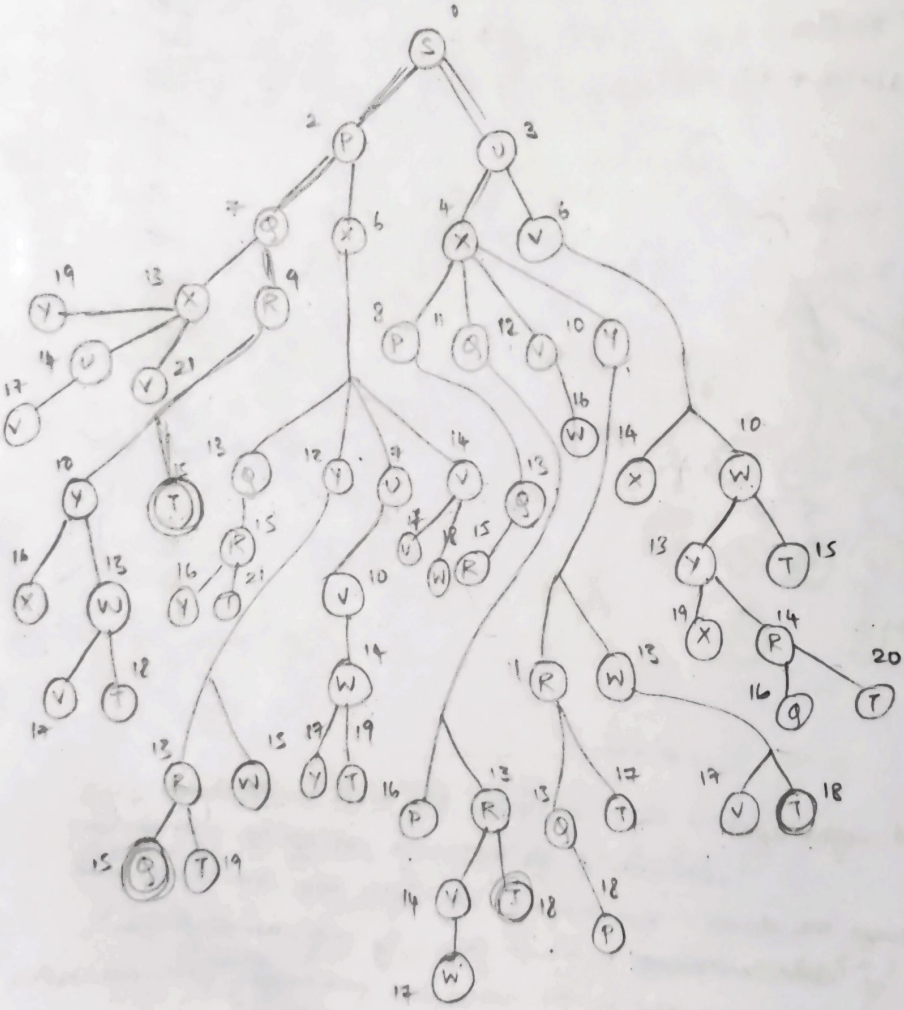
Single Agent - AI enabled camera system embedded in mobile phone

(3)



↓

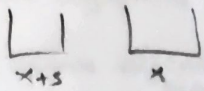
State space graph



S-P-Q-R-T and S-U-X-Q-R-T

(15) (18)

④



$x + 5 + x = 19$

$2x = 14$

$x = 7$

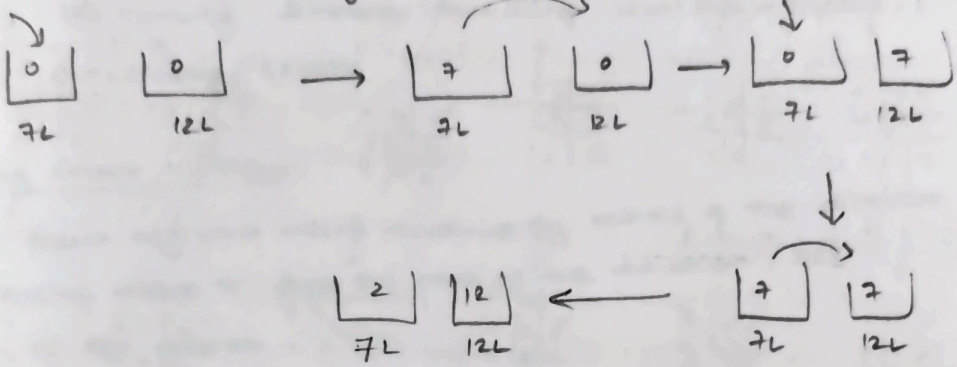
Initial state -

capacity of both unknown.

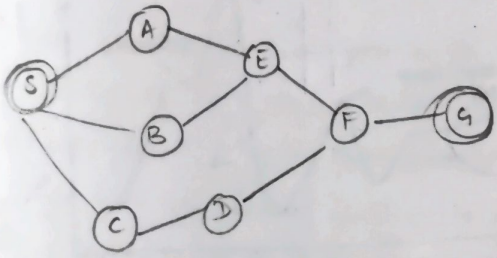
Goal test - 2L in one can

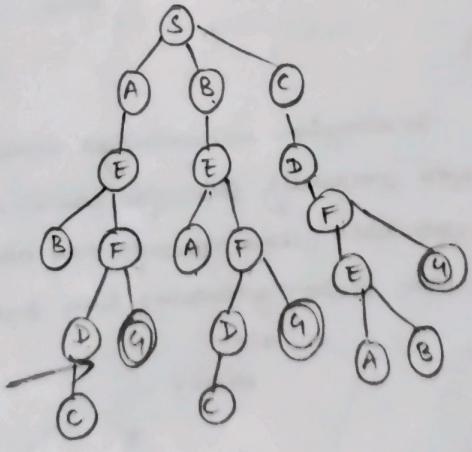
Possible actions - fill each can/
empty can

Steps - Determine x and x+5 by solving the equation.



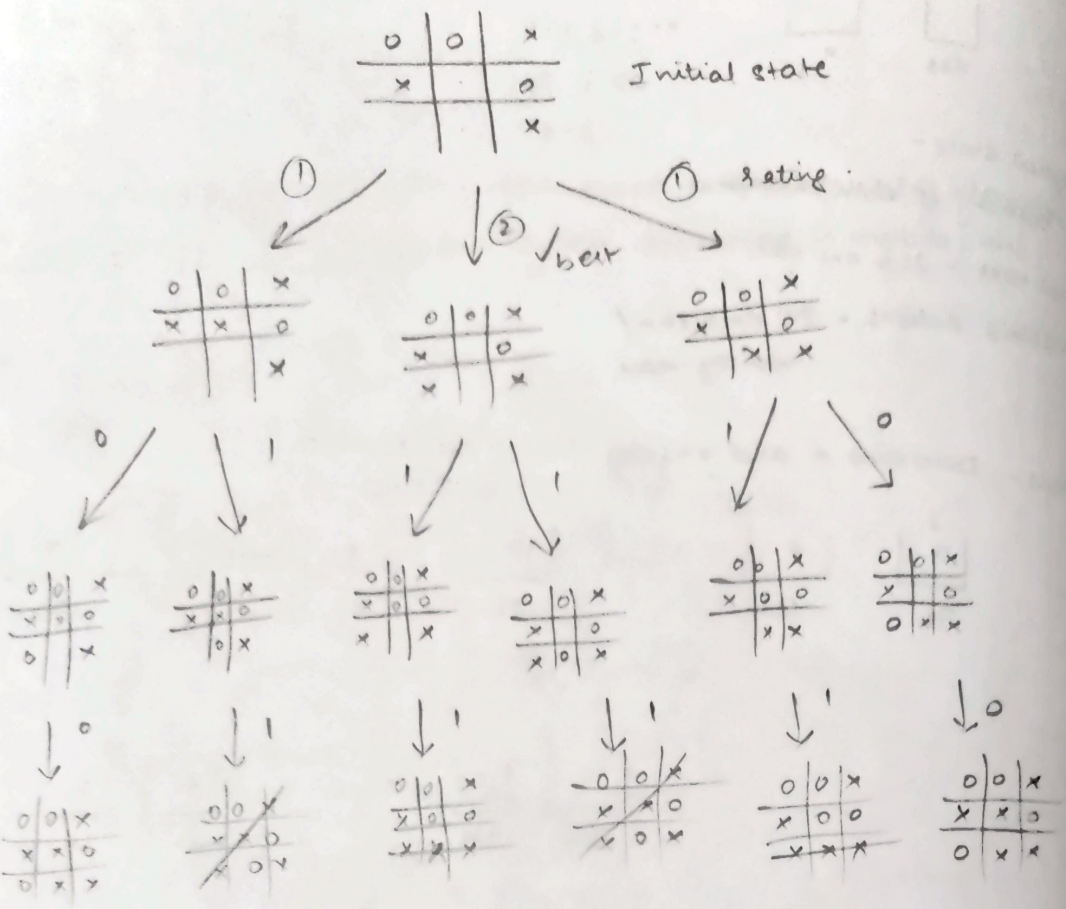
⑤ A* algorithm





Adv - Memory

Disadv - Time Complexity



8 queens problem and heuristic function (h_3) that returns no. of attacking queens.

61514381 row no. for each col.

Local Search Algorithms

The local search algorithm searches only the final state, not the path to get there.

eg: 8 queens problem

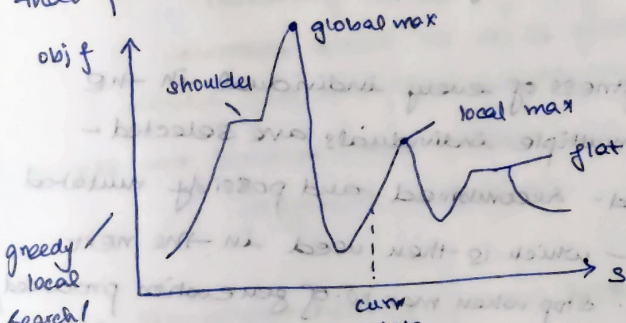
They operate by searching from a start state to neighbouring states, without keeping track of the paths, nor the set of states that have been reached.

They are not systematic.

Algorithms - Hill climbing, Simulated Annealing, Local Beam Search,
Evolutionary - Genetic

Hill Climbing Search Algorithm:

- Heuristic search algorithm which continuously moves in the direction of increasing value to find the peak of the mountain / best solution of the problem.
- Keeps track of one current state and on each iteration moves to the neighbouring state with highest value - heads in direction that provides the steepest ascent:

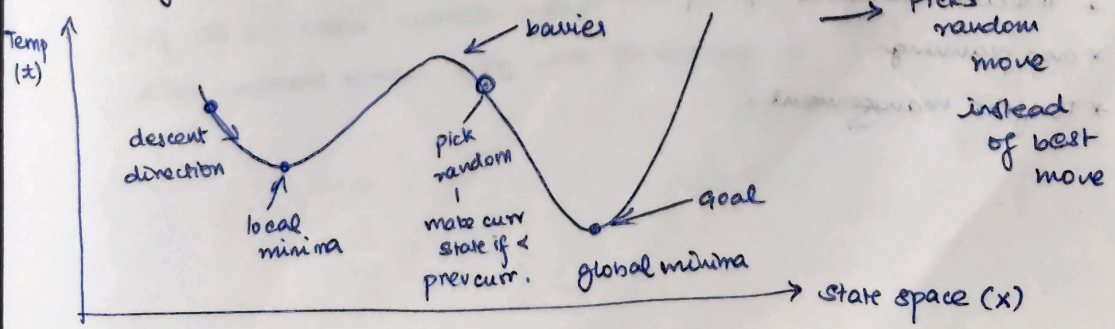


idea $P(\Delta E) = e^{-\Delta E/kt}$

$e^{-\Delta D/T} > R(0,1)$

Simulated Annealing:

- Stochastic global search optimization algorithm.
- Appropriate for non-linear objective functions where other local search algorithms do not operate well, used when lot of local minima.
- Start by shaking hard and gradually reduce intensity of shaking.



Beam Search Algorithm :

- Heuristic search algorithm that examines a graph by extending the most promising node in a limited set.
- Beam width = No. of nodes n
- Only keeps the lowest number of nodes on open list.

Components :

1) Problem - rep. as graph with set of nodes, one or more nodes represents a goal.
2) Set of heuristic rules for pruning - rules to prune unfavorable nodes.
3) Memory - limited capacity (beam), if memory is full and a node is to be added to the beam, the most costly node will be deleted, if memory limit is not exceeded.

Genetic Algorithm (GA) :

- Used to find true/approximate solutions.
- Categorized as global search heuristics.
- Use techniques from biology - inheritance, mutation, selection, crossover (recombination).
- Starts from a population of randomly generated individuals and happens in generations.

In each generation, the fitness of every individual in the population is evaluated & multiple individuals are selected - parents. These are modified - recombined and possibly mutated to form a new population - which is then used in the next iteration of the algorithm. Stop when max no. of generation produced/ satisfactory fitness level has been reached for the population.

Applications :

* IC design
* Factory floor layout VLSI } - Simulated Annealing best!
* Job shop scheduling
* Automatic programming
* Telecommunications N/w optimization
* Crop planning
* Portfolio management.

Advantages:

- use very little memory
- often find reasonable solutions in large/∞ state spaces – where classical search is not suitable.
- solve optimization problems.
- find best state according to an objective function.

Note:

function HillClimbing(problem) returns a state that is local maximum

current ← problem.INITIAL

while true do

neighbour ← a highest valued successor state of current

if VALUE(neighbour) ≤ VALUE(current) then return current

current ← neighbour

Problems: — Soln: Random Restart Hill Climbing

1) Local maximum → solution: Backtracking technique – select another initial state and then restart the search.

2) Plateau → solution: Take big steps while searching, select randomly a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.

3) Ridges → solution: use bidirectional search, by moving in different directions.

function SimulatedAnnealing(problem, schedule) returns a solution state

current ← problem.INITIAL

for t=1 to ∞ do - - - -> unknown env.

T ← schedule(t)

if T=0 return current

next ← a randomly selected successor of current

ΔE ← VALUE(current) - VALUE(next)

if ΔE > 0 then current ← next

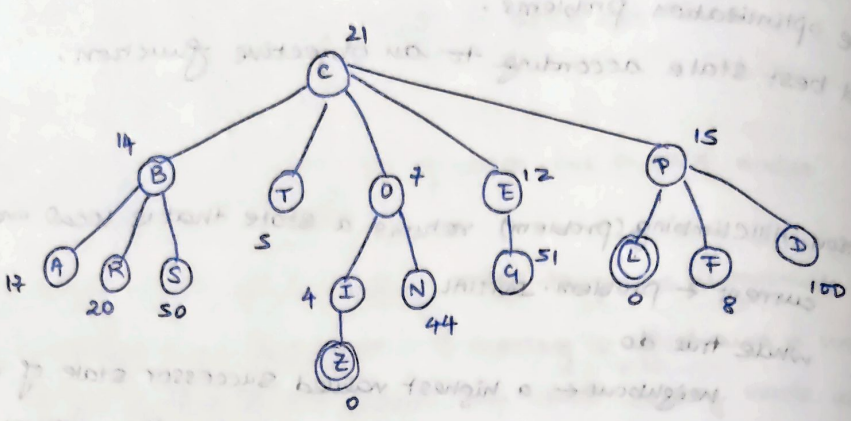
else current ← next only with probability $e^{-\Delta E/T}$

Example of Beam Search Algorithm -

Start state = C

Goal state = Z and L

n=2 (beam size)



Iteration 1

open list = {C}

Iteration 2

Successor of C = B T O E P

open list = {T, O}

Iteration 3

T has no successors, remove T from open list.

Find successor of O and replace.

open list = {I, N}

Iteration 4

Find successor of I

Z is goal.

Path ⇒ C → O → I → Z

This algorithm is not complete.

It is not optimal.

Time complexity = O(B * m) ← Beam width

Space complexity = O(B * m) ← max depth of any path in the search tree.

If B=1 then algorithm is Best First Searching.

When compared to A* search, TC and SC is very much reduced.

Steps in genetic algo.

- Select initial population
- Fitness function
- Parent selection
- Crossover
- Mutation
- Recombine

Mixing number = p
= No. of parents that come together to form offspring.

Either select from all individuals based on the fitness score / select from n individuals.

Random select of crossover point to split each of parent strings.

Adversarial Search

- Search in competitive environments – agent's goals are in conflict.
- Games are good examples of adversarial search.

eg: chess and tic tac toe. deterministic, fully obs. env. 2 agents action alternate & utility vals are equal & opposite.

Game →

s_0 : The initial state

Player (s) : Defines which player has the move in state (s)

Actions (s) : Returns the set of legal moves in a state

Results (s, a) : The transition model which defines result of move.

Terminal-Test (s) : True when game is over, false otherwise.

Utility (s, p) : Objective / Payoff function.
Defines the numeric value for a game that ends in a terminal state s for player p.

s_0, Actions (s), Results (s, a) defines a game tree.

Nodes – game states

Edges – moves

Optimal decisions in games →

For a normal search problem, optimal solution is sequence of actions leading to a goal state.

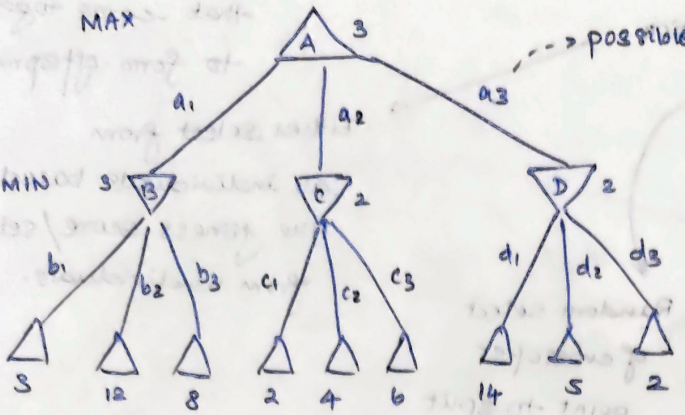
For adversarial search problem, one player (MIN) interferes with the sequence of actions.

Strategy for (MAX) player – specify moves in initial state, observe every possible response by MIN, specify moves in response.

minmax value

Calculating utility value of nodes (leaves) is easy since the outcome is visible. But how to calculate utility value for non-terminal nodes?

eg:



Optimal strategy can be determined using minimax value of each node MINIMAX(n).

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Mini Max Search :

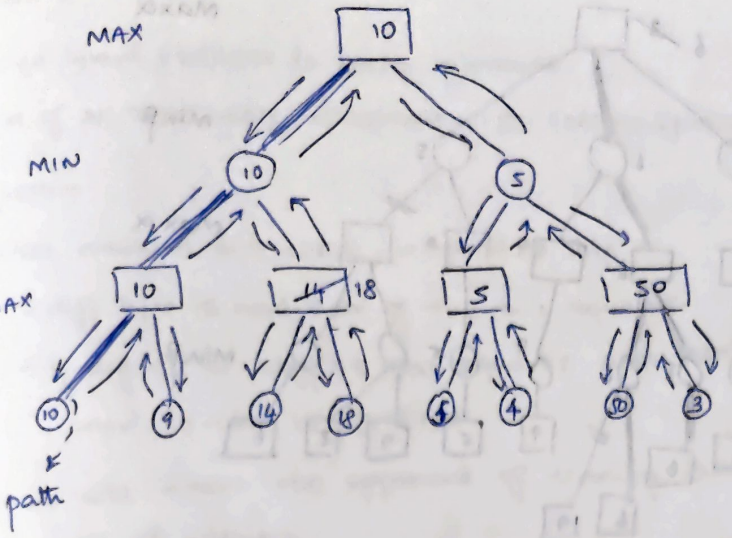
Given 2 player game tree in which the static scores are given from the first players point of view.
Compute value at root node and find the most convinient path for MAX node.

Steps:

1. Generate the whole game tree to leaves.
2. Apply utility (payoff) function to leaves
3. Uses DFS for expanding the tree.
4. Back up values from leaves toward the root.
   - max node computes maximum value from its child values.
   - Min node computes minimum value from its child values.
5. When value reaches the root - optimal move is determined.

Complete - Yes (if game tree is finite)
Optimal - Yes
Time Complexity - $O(b^d)$
Space complexity - $O(bd)$

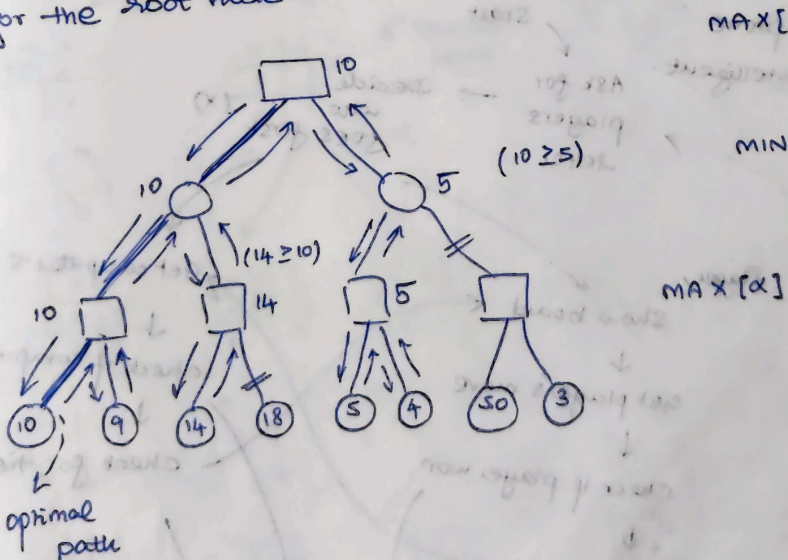
eg:



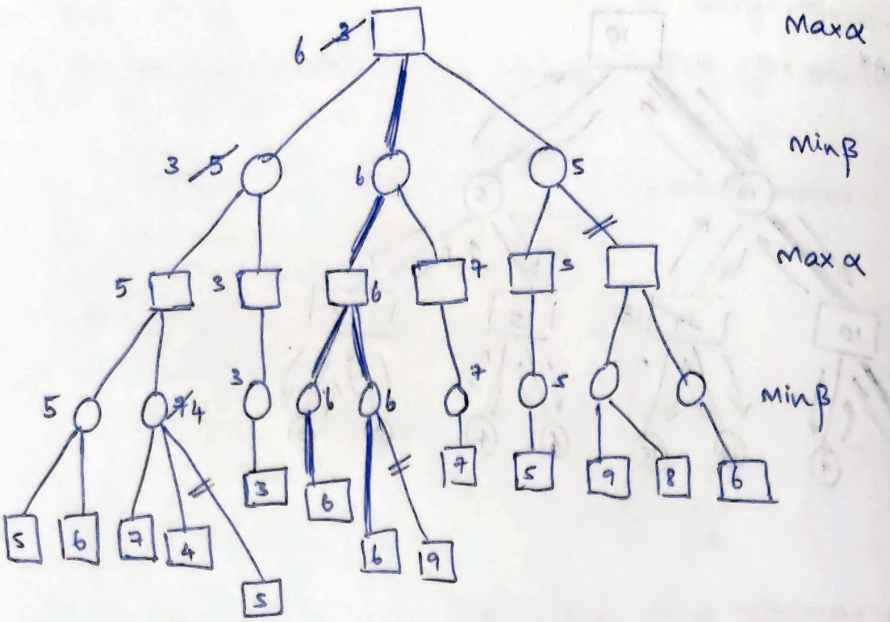
Alpha Beta Pruning :

- The problem with minimax search is that we need to search each and every path - the number of game states it has to examine is exponential in the number of moves.
- αβ propaposes to find the optimal path without looking at every node in the game tree.
- Max contains α
  Min contains β
- In both MIN and MAX node we return when $\alpha \geq \beta$ which compares with its parent node only.
- Both minimax & αβ cutoff give same path.
- αβ gives optimal solution as it takes less time to get the value for the root node.

eg:



eg:



Note:

Tic Tac Toe Game Playing:

Approach-1:

Data structures - Board and move Table

9 elements vector

0 for blank
1 for X
2 for O.

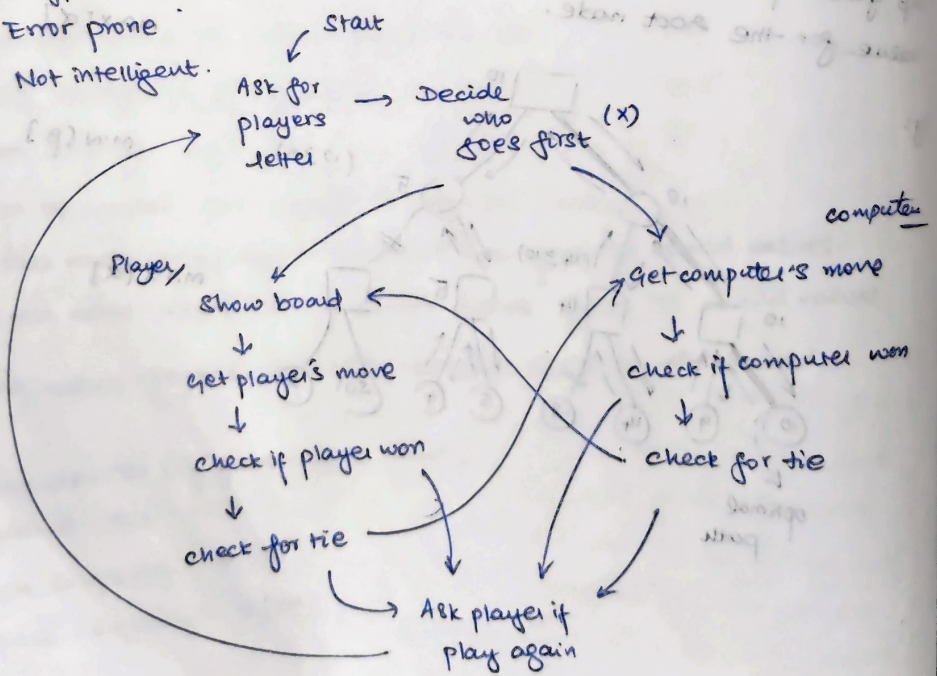
Vector of 3^9 elements = 19683,
each element is a
nine-element vector
representing board position.

Curr board pos → New Board pos.

(Lookup table)

Adv: Time efficient.

Disadv: Space complexity ↑.
Difficult to extent.
Error prone
Not intelligent.



Approach-2:

Assign board positions to vector elements.

sum of all rows, cols, diagonals = 15 (Magic Square)

8	3	4
1	5	9
6	7	2

Algorithm:

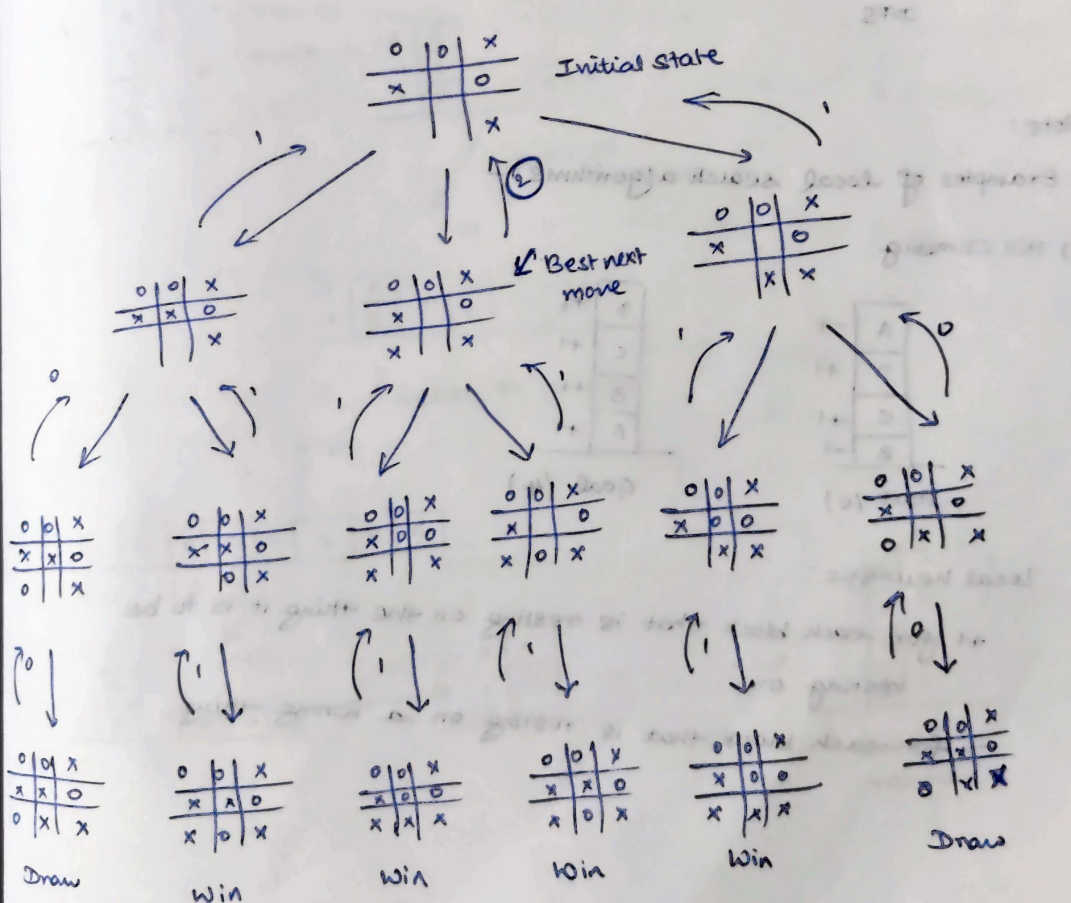
* First machine will check, chance to win
  - diff b/w 15 and sum of the two squares.
  - if diff < 0 or diff > 9 the squares were not collinear and so can be ignored.

* or it will check the opponent of winning and block the chances of winning.

Approach-3:

Data structures - Board position is a structure containing

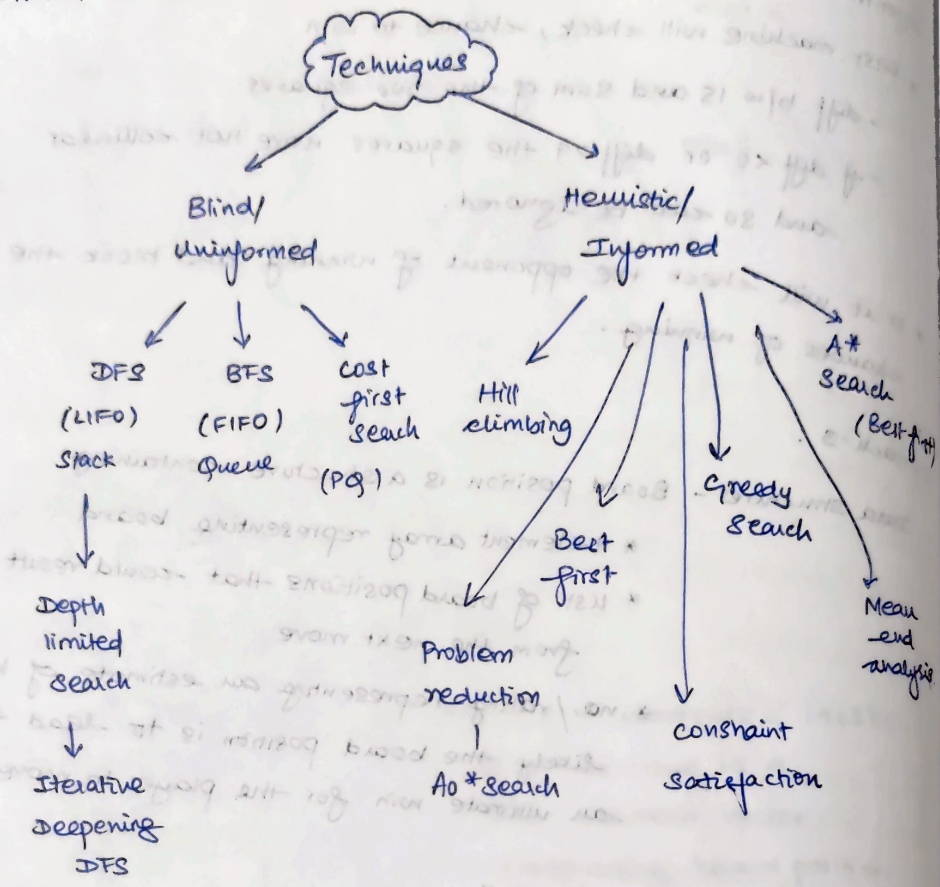
* 9 element array representing board
* list of board positions that could result from the next move.
* no./rating representing an estimate of how likely the board position is to lead to an ultimate win for the player to move.



Note:

Search - systematic examination of states to find path from start/root state to the goal state.

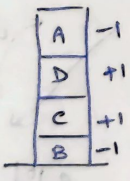
Should cause motion and must be systematic.



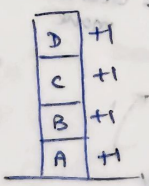
Note:

Examples of local search algorithms:-

1) Hill Climbing



Start (0)

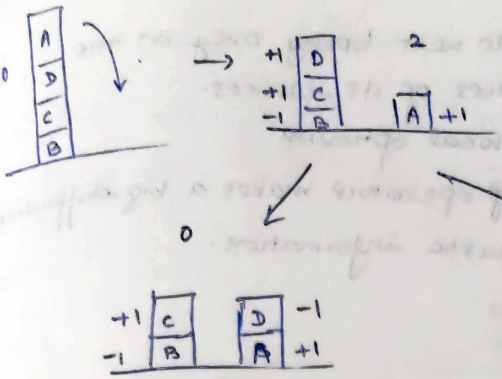


Goal (4)

Local heuristic:

+1 for each block that is resting on the thing it is to be resting on.

-1 for each block that is resting on a wrong thing.

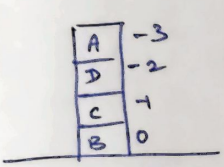


Local optimum, cannot reach goal state!

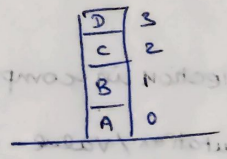
Global heuristic:

For each block that has the correct support structure +1 to every block in the support structure.

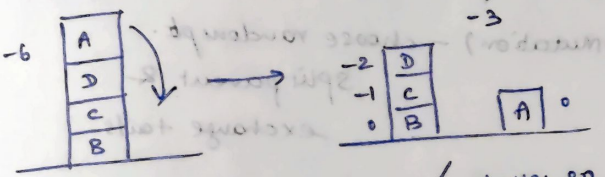
For each block that has a wrong support structure -1 to every block in the support structure.



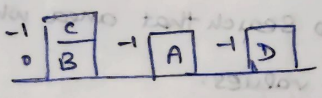
Start (-6)



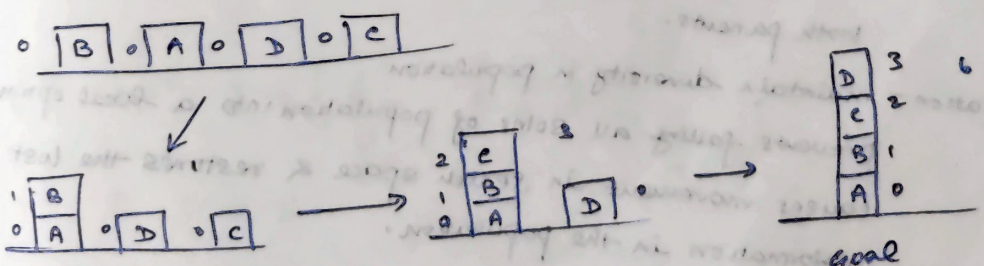
Goal (6)



better SD continue



better SD continue



Goal reached.

Note:

Local information - decides what to do next looking only at the immediate consequences of its choices.

Will terminate when at local optimum.

The order of application of operators makes a big difference.

Global information - encoded in heuristic information.

Note:

Hill climbing	VS	Steepest Ascent Hill climbing
Basic hill climbing first applies one operator and gets new state. If it is better then becomes current state.		This considers all moves from the current state. Selects the best one as the next state.

Note:

Simulate process of natural selection in computer:

- Encoding - Binary / Permutation / value
- Evaluation - Fitness func.
- Selection criteria - Roulette wheel / Rank / Tournament / Steady state
- Reproduction (cross over & Mutation) - choose random pt. split parent & exchange tails.
- Insertion & Termination

Significance of genetic operators:

Reproduction - force the GA to search that area which has highest fitness values.

Crossover - ensures search progresses in the right direction by making new chromosomes that possesses characters of both parents.

Mutation - maintain diversity in population.
prevents falling all soln. of population into a local optima.
causes movement in search space & restores the lost information in the population.

Unification

Unification is an algorithm for determining the substitutions needed to make two FOL expressions match.

To apply the rules of inference, the inference system must be able to determine when two expressions match - here unification algorithms are used.

UNIFY algorithm is used, it takes two atomic sentences and returns a unifier for those sentences (if any exist)

eg: Find Most General Unifier (MGU) for Unify {King(x), King(John)}

→ α_1 = King(x)

α_2 = King(John)

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution & both expressions will be identical.

constant or Skolem function of the enclosing [illegible] quantified variables.

$\forall x$ Person(x) $\Rightarrow$ $\exists y$ Heart(y) $\wedge$ Has(x, y)

↓

$\forall x$ Person(x) $\Rightarrow$ Heart(H(x)) $\wedge$ Has(x, H(x))

(5) Drop universal quantifiers

$\forall x$ Person(x) becomes Person(x)

(6) Distribute $\wedge$ over $\vee$

$(\alpha \wedge \beta) \vee \delta \equiv (\alpha \vee \delta) \wedge (\beta \vee \delta)$

FAT preparation :-

Uncertainity — FOL fails

* Laziness
* Theoretical ignorance
* Practical ignorance

causes :-

* Info from unreliable sources
* Experimental error
* Equipment failure
* Temp. variation
* climate change

Reasoning :-

Degree of belief ⇒ Probability theory

$$P(A) = \frac{\text{No. of desired}}{\text{Tot. no. of outcomes}} \Rightarrow [0,1]$$

Prior / Unconditional probability ⟶ $P(A)$

Posterior / conditional probability ⟶ $P(A|B) = \frac{P(A \cap B)}{P(B)}$

A and B are independent,

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

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

Joint probability distribution

Bayes Rule :-

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

$$\therefore P(A \cap B) = P(A|B) * P(B)$$

$$P(A \cap B) = P(B|A) * P(A)$$

posterior (likelihood (prior

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

(marginal

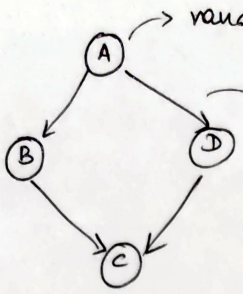
Bayesian Network / — DAG

belief n/w /

decision n/w

Prob. graphical modeling

— prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, decision making under uncertainity.



→ random var.

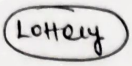
→ relationship/ conditional probability

DAG

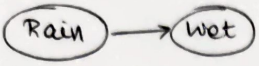
No parent - prior prob.
other - cond. prob.

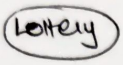
One path - poly tree.

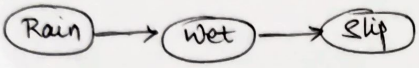
eg:



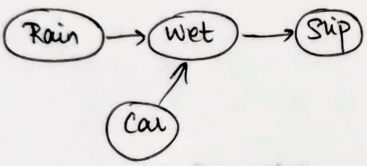
$P(L,R,W) = P(L) \cdot P(R) \cdot P(W|R)$







$P(L,R,W,S) = P(L) \cdot P(R) \cdot P(W|R) \cdot P(S|W)$



$P(R,W,C,S) = P(R) \cdot P(C) \cdot P(W|R,C) \cdot P(S|W)$

Planning :-

Deciding set of action to perform based on perception to achieve desired goal.

- Domain specific
- Domain independent

Domain description + Task/Action specification + Goal description

Planning = set of actions

Action = { Precondition - for action to be performed
Effect - result of action performed.

BLOCK WORLD problem:

Actions : unstack(A,B)
stack(A,B)
pickup(A)
putdown(A)

Predicates : ON(A,B)
ONTABLE(A)
CLEAR(A)
HOLDING(A)
ARMEMPTY

Planning Languages :-

States, Goals, Actions

STRIPS : Stanford Research Institute Problem Solver

ADL : Action Description Languages

PDDL : Planning Domain Definition Language

→ State :-

conjunction of ground, function free & positive literals

Goal :-

Conjuction of literals/variables.

Can have more than one state.

Action :-

Schema -

Action name & parameters

Preconditions - conjunction of func. free +ve literals

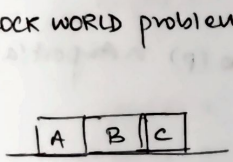
Effects - conjunction of func-free

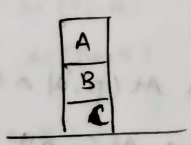
pos. (ADD list)

neg (DELETE list) literals.

Examples :-

① BLOCK WORLD problem

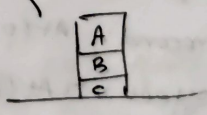




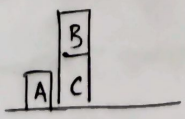
On(A, Table)
On(B, Table)
On(C, Table)
clear(A)
clear(B)
clear(C)

On(A, B)
On(B, C)

$\equiv q$



Move(B, Table, C)



move (A, Table, B)

Precondition = On(A, Table) ∧
On(B, Table) ∧
On(C, Table) ∧
clear(A) ∧ Clear(B) ∧ clear(C)

Effect = On(B, C) (add list)
On(B, Table) ∧ clear(C) (delete list)

On(A, Table) ∧
On(B, C) ∧ On(C, Table)
∧ clear(A) ∧ clear(B)

On(A, B) (add list)
clear(B) ∧ On(A, Table)
(delete list)

Define all actions →

R1: pickup(x)

Precond : handempty, on(x, table), clear(x)

Add list : holding(x)

R2: putdown(x)

Precond : holding(x)

Add list : handempty, on(x, table), clear(x)

R3: Stack(x, y)

Precond : holding(x), clear(y)

Add list : on(x, y), clear(y), handempty

R4 : unstack(x, y)

Precond : handempty, on(x, y), clear(x)

Add list : holding(x), clear(y)

② AIR CARGO TRANSPORT problem

Actions →

(load (c, p, a)

Precond : At(c, a) ∧ At(p, a) ∧ cargo(c) ∧ Plane(p) ∧ Airport(a)

Effect : ¬ At(c, a) ∧ In(c, p))

(Unload (c, p, a)

Precond : In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)

Effect : ¬ In(c, p) ∧ At(c, a))

(Fly (p, from, to)

Precond : At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)

Effect : ¬ At(p, from) ∧ At(p, to))

③ SPARE TIER problem

Remove (x, y)

Put (x, y)

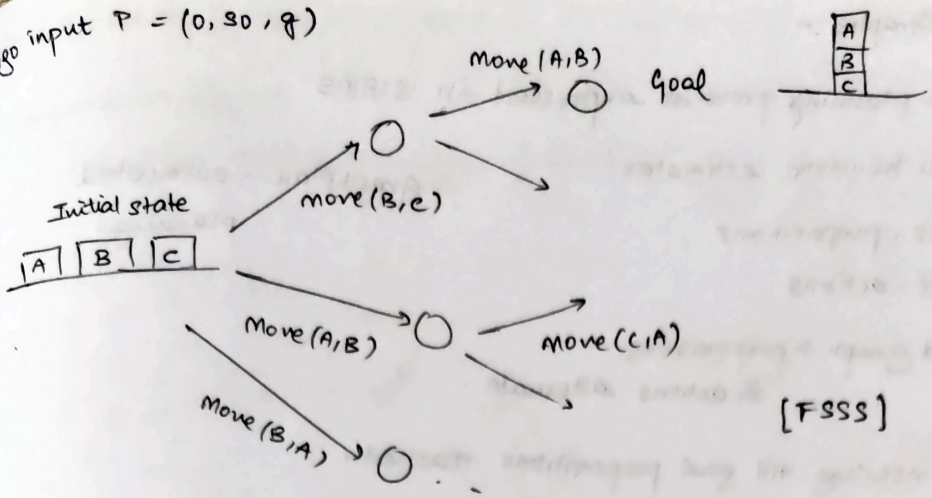
Planning with State-space search :-

Forward SSS/progression (initial state → final)

Backward SSS/regression (goal → initial)

Initial State, Actions, Goal Test, Step Cost

Algo input P = (O, s0, g)

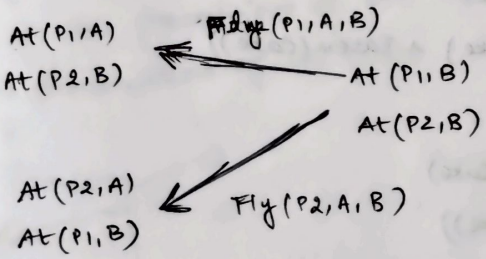


[FSSS]

(1) compute whether state is goal state
(2) find the set of actions that are applicable to a state
(3) compute successor state (result of applying action to state)

---

(1) Start at the goal, test if its initial state, otherwise
(2) Apply inverse of the planning operators to produce subgoals
(3) The algo. will stop if satisfies initial state.



[BSSS]

Planning Algorithms :-

① State space search / Total order Planning (A* - finite states) ← Soln → Path (seq actions)
   | need admissible heuristic.
② Partial order Planning (POP)
③ Planning graphs (graphPlan)

[Note]

Planning Graphs :-

Input → planning problem expressed in STRIPS

* better heuristic estimates

Graphplan - automated planning

Nodes - propositions

Edges - actions

Layered graph → propositions & actions alternate.

Keep inserting till goal propositions reached.

(mutually exclusive)

→ construct graph
→ search for solution
→ If not found extend graph & search again.

} until soln. found.

HAVE CAKE and EAT CAKE problem :-

Init (Have (Cake))

Goal (Have (Cake) ∧ Eaten (Cake))

Action (Eat (Cake)

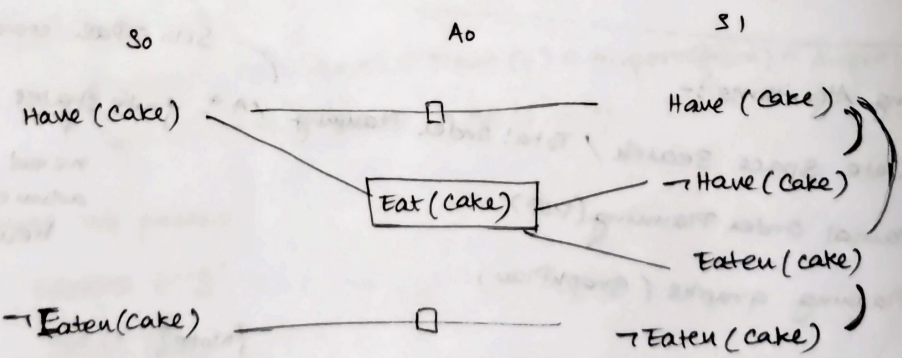
Precond : Have (Cake)

Effect : ¬ Have (Cake) ∧ Eaten (Cake))

Action (Bake (Cake)

Precond : ¬ Have (Cake)

Effect : Have (Cake))

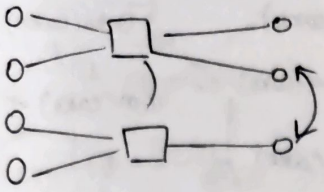


Identify mutual exclusion b/w actions & literals based on potential conflicts.

Mutual Exclusion (Mutex) b/w actions

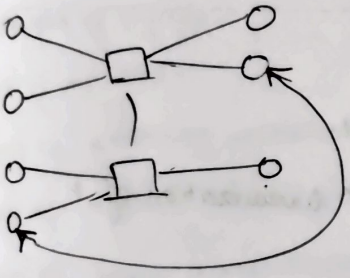
① Inconsistent effects

One action negates an effect of another.



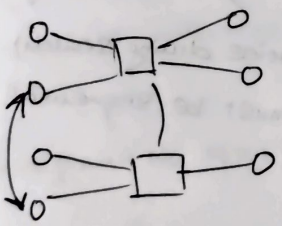
② Interference / Prec-effect

One of the effects of one action is the negation of precondition of other.



③ Competing needs

One of the preconditions of one action is mutually exclusive with precondition of the other.

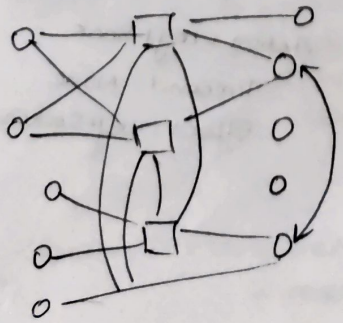


④ Inconsistent support

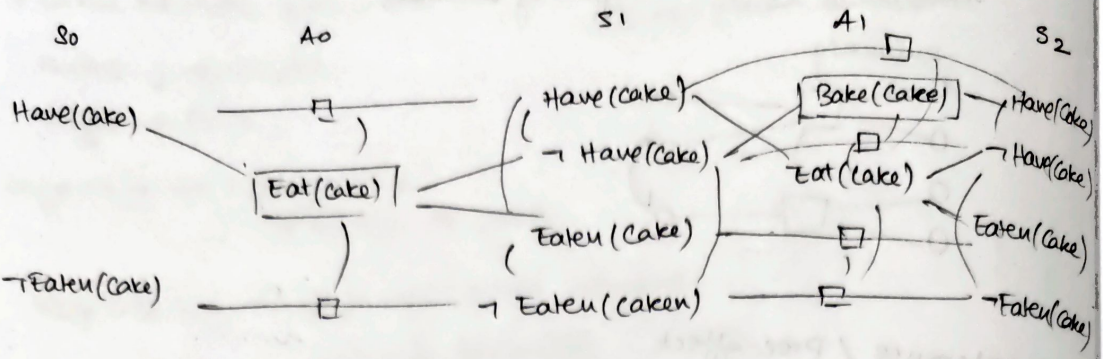
↳ b/w pair of propositions.

a proposition & its negation are mutex.

When only way of achieving the two is mutex.



You have to keep repeating process until 2 cons. levels are identical.
Contains same amount of literals.



* Literals increase monotonically.
* Actions increase monotonically.
* Mutexes decrease monotonically.

Partial Order Planning :-

Some steps ordered & some unordered.

Plan (P) ⟶ Totally ordered Plan ⇒ linearization of P.
+ ordering constraints

(1) Works on several subgoals independently
(2) Solves them with subplans
(3) Combines the subplans
(4) Flexibility in ordering the plans
(5) Least commitment strategy (delaying choice during search) / choice only about things you currently about.
(6) Leave actions unordered, unless they must be sequential.

PUTTING ON SHOES and SOCKS problem :-

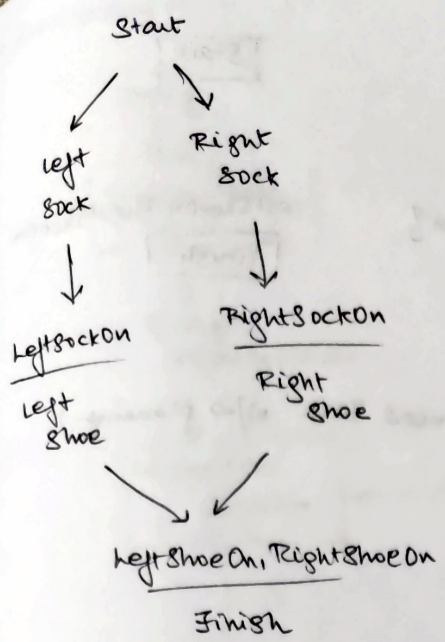
Goal (RightShoeOn ∧ LeftShoeOn)
Init()

Action : RightShoe
Precond : RightSockOn
Effect : RightShoeOn

Action : LeftShoe
Precond : LeftSockOn
Effect : LeftShoeOn

Action : RightSock
Precond : None
Effect : RightSockOn

Action : LeftSock
Precond : None
Effect : LeftSockOn



Order can be changed here.

But in total order planning — 6 ways.

Note

Practical prob are too complex to solve using simple ~~dumb~~ planner.

components of a plan :-

(1) A set of actions.

(2) A set of ordering constraints ($A \prec B$) — A before B

(3) A set of casual links / protection intervals b/w actions ($A \xrightarrow{p} B$) — A achieves p for B

(4) A set of open preconditions

Threat - New action that interferes with past decision.

Consistent Plan :-

* No cycle in the ordering constraints.
* No conflicts with the casual links.

Solution :-

consistent plan + NO open preconditions.

$A \xrightarrow{P} B$

C is a threat

demote: $C \prec A$ or promote: $B \prec C$

Steps :-

① Add dummy states

Start — $Literal_a, Literal_b, \ldots$ → No precondition. Effects are literals of initial state

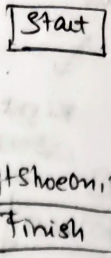
$Literal_1, \ldots Literal_n$ → Finish — Preconditions are literals of goal state. No effects.

Actions : {Start, Finish}

Ordering Constraints : {Start ≺ Finish}

Casual Links : {}

Open Preconditions : {LeftShoeOn, RightShoeOn}



Hierarchical planning :- (HTN - Hierarchical Task N/w planning)

Lower computational cost of planning

↓ Size of search space

(1) Identify hierarchy of conditions

(2) Construct a plan in levels, postponing details to the next level

(3) Perform hierarchical planning using Abstraction based ABSTRIPS

Multiagent planning :-

All agents needs RC-required cooperation.

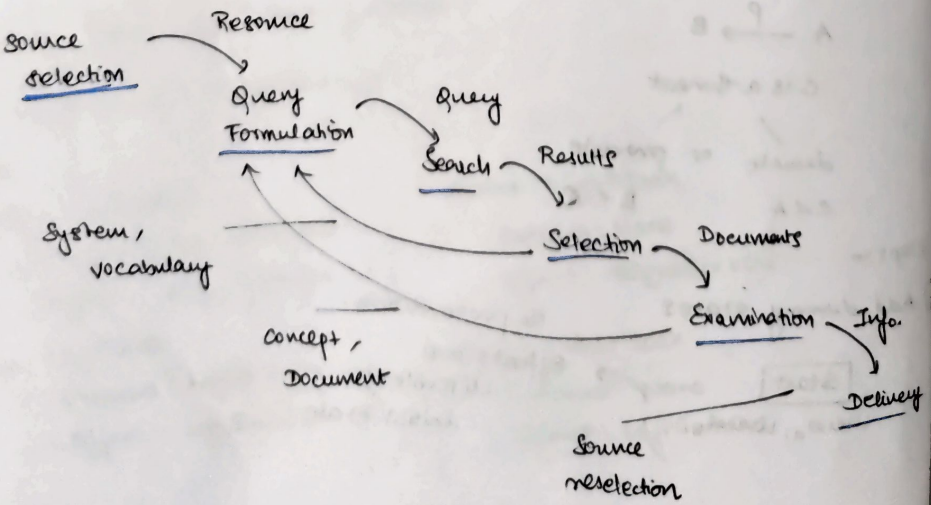
Information retrieval (IR) — rep., search, manipulate large collection of e-text. (Search for collections & find documents)

Corpus → repository of document

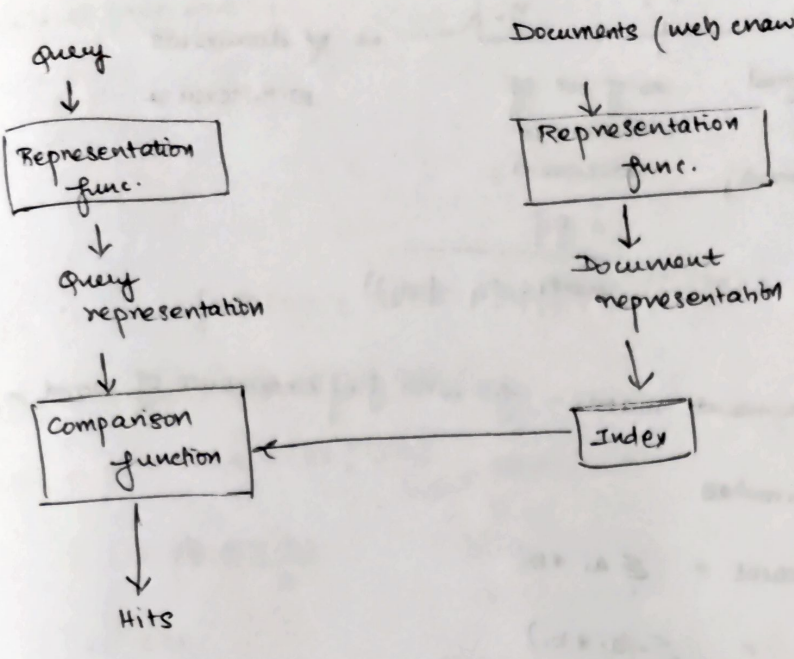
Info. need

Relevance

IR cycle :-



IR Architecture :-



Text representation :-

Bag of words → All words are index terms. Assign weight to each. (freq.)

Assumes term occurence & document relevance independent.

words are well defined.

IR Models :-

① Boolean Retrieval Model

V = {distinct words} = vocabulary

outcomes of q eval ⇒ T/F (exact match)

System retrieves every document that makes the query logically true.

② Vector Space Model

Documents that are close in vector space will be retrieved.

Document (d_j) & query (q) → vectors t-dim (index term)

$$\cos\theta = \frac{\bar{d_j} \cdot \bar{q}}{|\bar{d_j}| \cdot |\bar{q}|} = \frac{\sum_{i=1}^{t} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{t} w_{ij}^2} \times \sqrt{\sum_{j=1}^{t} w_{jq}^2}} \quad \in [0,1]$$

Term weights → TF or TF-IDF — statistical method in NLP, IR

Measures importance of term within a doc.

given a collection of documents.

Local : TF

Global : IDF

$$w_{ij} = tf_{ij} \cdot \log \frac{N}{n_i}$$

- w_{ij}: Weight assigned to term i in document j.
- tf_{ij}: no. of occ. of term i in document j.
- N: no. of documents
- n_i: no. of documents with term i.

$$= \frac{f_{ij}}{\max(f_{1j}, f_{2j} \cdots f_{|V|j})}$$

* Term-Document matrix – fill with f of occurence of word in document

Similarity formulas

$$\text{Dot product} = \sum_i a_i * b_i$$

$$\text{Cosine} = \frac{\sum_i (a_i * b_i)}{\sqrt{\sum_i a_i^2 * \sum_i b_i^2}}$$

$$\text{Dice} = \frac{2 \sum_i (a_i * b_i)}{\sum_i a_i^2 + \sum_i b_i^2}$$

$$\text{Jaccard} = \frac{\sum_i (a_i * b_i)}{\sum_i a_i^2 + \sum_i b_i^2 - \sum_i (a_i * b_i)}$$

③ Random Surfer model

PageRank as a model of user behavior (vote – link to page – importance of page)

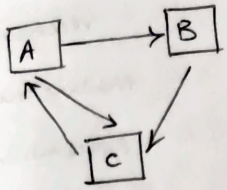
$$PR(A) = (1-d) + d\left[PR(T_1)/C(T_1) + \cdots + PR(T_n)/C(T_n)\right]$$

- $PR(A)$: page rank of page A
- d: damping factor b/w 0 & 1.
- $PR(T_i)$: of page T_i that links to A.
- $C(T_i)$: No. of outbound links in T_i

Pages with no outbound links – dangling links.

8) $d = 0.85$

Initially all page ranks = 1



$$PR(A) = (1-d) + d\left[PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n)\right]$$

iter1

$$PR(A) = (1-0.85) + 0.85\,(1/1) = 1$$

$$PR(B) = (1-0.85) + 0.85\,(1/2) \quad \text{(1/2} \rightarrow PR(A)/C(A))$$

$$= 1-0.85 + \frac{0.85}{2}$$

$$= 0.575$$

$$PR(C) = (1-0.85) + 0.85\left[PR(A)/C(A) + PR(B)/C(B)\right]$$

$$= (1-0.85) + 0.85\left[(1/2) + (0.575/1)\right]$$

$$= 1.06375$$

iter2

$$PR(A) = (1-0.85) + 0.85\,(PR(C)/C(C))$$

$$= (1-0.85) + 0.85\,(1.06375/1)$$

$$= 1.0541875$$

$$PR(B) = (1-0.85) + 0.85\,(PR(A)/C(A))$$

$$= (1-0.85) + 0.85\,(1.0541875/2)$$

$$= 0.598029\,6875$$

$$PR(C) = (1-0.85) + 0.85\,(PR(A)/C(A) + PR(B)/C(B))$$

$$= (1-0.85) + 0.85\,((1.0541875/2) + (0.598029\,6875/1))$$

$$= 1.06354922$$

Iter	A	B	C	D	...
0					
1					
2					

final results table

④ Probabilistic Language Models

Assigns probability to every sentence.

Picks one with higher probability.

$$P(S) = P(w_1, w_2, \ldots, w_N)$$

— Speech recognition
OCR & handwriting recognition
Machine translation
Generation
Context sensitive spell correction
Completion prediction

Language Model – NL (parse trees)
↑
morphology of words, syntax, semantics.

Implementation:

Input sentence → Language model → Predicts the next word.
↑
Knowledge grammar, sentence structure, domain

N-gram (Markov Modelling)
– Predict n from n-1 given words.

simplify – Markov assumption → compute probability of a word based on the probability of only the previous one / few words.

Toolkits
SRILM, KenLM, Google N-Gram

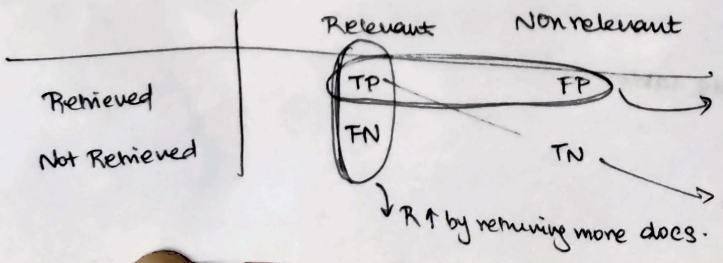
Evaluating IR :–

Precision – proportion of results that are relevant.

Recall – proportion of relevant docs that are in results.

ROC curve – plot false neg. vs false pos.

Web: Time to answer or 1/Rank of first relevant result.



$$F1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$

NL- very ambiguous

NLP:-
* human language
* understand & perform interesting tasks.
* I/N/o/p → Speech/ Text
→ lexical, syntatic, semantic, discourse info., real world knowledge
+ speech recognition & synthesis.

components:-
- Natural Language understanding
- Natural language generation

NLU:-
Knowledge of language

Phonology - words related to the sounds

Morphology - how words are constructed from more basic meaning units (morphemes).

Syntax - how words can be put together to form correct sentences.

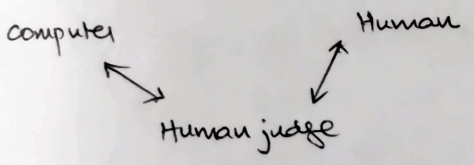
Semantics - what is the meaning of words

Pragmatics - how sentences are used in different situations.

Discourse - how the immediate preceeding sentence affect the interpretation of the next sentence.

World knowledge - general knowledge about the world.

Turing Test:-



NLG:-

Text planning - retrieve content

Sentence planning - form meaningful phrases

Text realization - map to sentence structures

NLP pipeline :-

(1) Sentence Segmentation (para → sentences)

(2) Word tokenization (sentence → words)

(3) Stemming (remove suffix, prefix from word & extract root form)

(4) Lemmatization (reduce words to canonical form (better → good))

(5) Stop word analysis (filter common - is, a, the, and)

(6) Dependency parsing (how words related - parse tree (main verb - root))

(7) Part-of-speech tagging (each word - adverb, adjective, verb, noun, ...)
preposition, conjunction, interjection

Sensing → Decision maker ← Knowledge

↓

Action

Knowledge engineering :

WFF - legal sentence

(1) Acquisition

(2) Representation (Logic) — Propositional / Sentinel
Predicate / Quantifier

(3) Validation

(4) Inference

(5) Explanation & Justification

Entailment

$P \models Q$

whenever P is true
so is Q.

Equivalence

$P \iff Q$

will be 1 if same.

$P \rightarrow Q$

will be Q if P == 1
else 1

$\forall \rightarrow$
$\exists \wedge$

$\exists x\,(P(x) \wedge S(x))$ Someone is sleeping

$\forall x\,(P(x) \rightarrow S(x))$ Everyone is sleeping

$\neg \exists x\,(P(x) \wedge S(x))$ None is sleeping

$\neg \forall x\,(P(x) \rightarrow S(x))$ Not everyone is sleeping

Unifier → Takes 2 sentences and returns a substitution that makes the 2 sentences look identical.

Conversion to CNF :-

1. Eliminate bicanditionals and implications.
2. Move ¬ inwards
3. Standardize
4. Skolemize (∃ with Skolem constant/function)
5. Drop ∀
6. Distribute ∧ over ∨

Prove predicate logic by resolution →

Negate the conclusion & add to clauses.

Convert to CNF

Make each conjunct a separate clause.

Standardize variables apart again.

∀x graduating(x) → happy(x)

∀x happy(x) → smiling(x)

graduating(JohnDoe)

¬ smiling(JohnDoe) ← From conclusion

↓

∀x ¬graduating(x) ∨ happy(x)

∀x ¬happy(x) ∨ smiling(x)

graduating(JohnDoe)

¬smiling(JohnDoe)

↓

∀x ¬graduating(x) ∨ happy(x)

∀y ¬happy(y) ∨ smiling(y)

graduating(JohnDoe)

¬smiling(JohnDoe)

↓

¬graduating(x) ∨ happy(x)

¬happy(y) ∨ smiling(y)

graduating(JohnDoe)

¬smiling(JohnDoe)

¬happy(y) ∨ smiling(y) — ¬smiling(JD)

{y/JD}

¬graduating(x) ∨ happy(x) — ¬happy(JD)

{x/JD}

graduating(JD) — ¬graduating(JD)

None

Convert to CNF :-

eg: $\forall x [\forall y \text{ Animal}(y) \rightarrow \text{Loves}(x,y)] \longrightarrow [\exists y \text{ Loves}(y,x)]$

$\forall x [\forall y \neg \text{Animal}(y) \vee \text{Loves}(x,y)] \rightarrow [\exists y \text{ Loves}(y,x)]$

$\forall x \neg [\forall y \neg \text{Animal}(y) \vee \text{Loves}(x,y)] \vee [\exists y \text{ Loves}(y,x)]$

$\forall x [\exists y \text{ Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists y \text{ Loves}(y,x)]$

$\forall x [\exists y \text{ Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists z \text{ Loves}(z,x)]$

$\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee [\exists z \text{ Loves}(z,x)]$

$\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee [\text{Loves}(G(z),x)]$

$[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$

$[\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)] \wedge [\text{Loves}(G(x),x) \vee \neg \text{Loves}(x,F(x))]$

Definite clause → Exactly 1 +ve literal

Horn clause → At most 1 +ve literal

Forward chain → Down up approach

Search
- Local
  - Hill climbing
  - Simulated Annealing
  - Genetic Algorithm (GA)
- Adversarial
  - Minimax
  - $\alpha\beta$ pruning

Game
- Adversarial
  - Utility vals are opp.
- Cooperative

- Initial state
- Successor func.
- Terminal test
- Utility func.

Minimax

Complete ✓ (finite)
Optimal ✓
TC = $O(b^d)$
SC = $O(bd)$

$\alpha\beta$ pruning

Dec-no of nodes that are evaluated.
Max node → α, Min node → β
($\alpha \geq \beta$)