* Humanoids

→ Nadine : A humanoid social robot modelled after prof. Nadia

→ Asimo :

* Non-humanoid robots

→ Paro
→ Aibo
→ Miro

* Is AI only about robots?
No.

* Some domains where AI is infused with -
- Social Media
- Spam Filters
- Finance
- Healthcare

* What is AI?

AI is a branch of computer science by which we ~~are~~ can create intelligent machines which can behave like a human, think like humans, and able to make decisions.

AI is the study of the systems that —
- Think Humanly
- Think Rationally
- Act/Behave Humanly
- Act/Behave Rationally

• Acting Humanly : Turing Test

Humans are intelligent

To be called intelligent, a machine must produce responses that are indistinguishable from those of a human.

If interrogator can't reliably distinguish human from a computer, possesses intelligence.

AI techniques: Required Capabilities to act humanly (Turing Test)

- Machine Learning
  Adapting to new circumstances & to detect & extrapolate

- NLP
  Enabling the computer to communicate successfully in English.

- Knowledge Presentation
  Storing what the computer knows or hears

- Robotic

- Automated Reasoning

- Computer Vision

• Think Humanly:
  Cognitive learning Approach

3 ways to get inside the mind:
- Introspection - trying to catch our own thoughts as they go by.
- Psychological Experiments - observing a person in action
- Brain Imaging

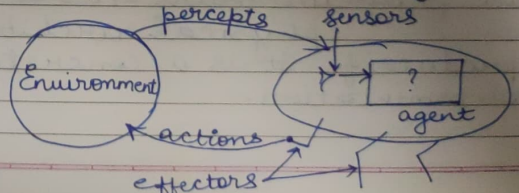
• Thinking Rationally:
  The laws of thought approach

  laws of thought → govern the operation of the mind

• Act Rationally:
  The rational Agent Approach

★ Agents < Robot / Software App.

An agent is something that acts.



An agent is anything that can be viewed as perceiving its environment through sensors & acting upon that environment through actuators.
↓
n-defectors

~~Integ~~

Intelligent Agent —

* AI Agent Components

- Action: An operation involving an actuator is called an action.
- Agent function (Agent's behaviour) maps from percept ~~to~~ histories to actions [f: p* A]
- Agent program: runs on the physical architecture to produce f. It is a concrete implementation.

- Percept: The complete set of inputs at a given time for an agent is called percept.
- Percept Sequence: Complete history of everything the agent has received
- Sensors: An agent perceives the environment through sensors.
  Eg: Camera, IR range finders.
- Actuators: Agent can change the environment through actuators/effectors.
  Eg: Motors.

* Vaccuum Cleaner World

Percepts: location & contents
Eg:- Dirty

Actions: left, Right, Top.

PAGE No.
DATE / /

★ Properties of Agents

→ Rationality
→ Autonomy
→ Reactivity

→ Rationality

- not the best decision but the best decision according to the situation.

→ Autonomy

The autonomy of an agent is the extent to which it's behaviour is determined by its own experience. (in dynamic unpredictable environments)

→ Reactivity

PAGE No.
DATE / /

★ PEAS Analysis

→ Performance Measures (P)
Environment (E)
Actuators (A)
Sensors (S)

- Must first specify the setting for intelligent agent design.
- Agents can be described by their PEAS
- Formally described as PAGE → Percepts, Actions, Goals, Environment
- A rational agent maximizes the performance measures for their PEAS.
- Specifying the task environment is always done first.

• Performance Measures.

- Objective criterion for success of an agent's behaviours
- depends on agents functions.

Eg: Vacuum-cleaner agent:
amt of dirt cleaned up, time taken, electricity consumed, noise generated.

: Self-driving cars:
time to reach destination, safety, predictability of behaviour, reliability

Eg: Game-playing agents:
win/loss %, robustness, unpredictability.

* Job of AI is to design the AI Agent

* AI Agent Environment

Types –

1) Fully observable (v/s partially observable) (Eg: Self driving car)
(Eg: Chess, Snakes & ladders)

2) Deterministic (v/s Stochastic)

3) Episodic (v/s sequential)

4) Static environment (v/s Dynamic)

5) Discrete (v/s continuous)

6) Single agent (v/s multiple agent)

2) Deterministic – Env is completely determined by the current state & the action executed by the agent (agent's action uniquely determines the outcome).
Env → deterministic except for actions of other agents → then env → strategic

~~Sto~~ Stochastic Env → Wherever probability is involved
Eg: Snakes & Ladders
(Probability of dice involved)

3) Episodic – current state & action of agent determines the outcome. Previous state does not affect the outcome. ∴ all states are divided into episodes as they are isolated events – independent of each other.
Eg: Stone, Paper, Scissors → previous outcome does not affect the current outcome.
Whereas in Chess & S&L are not Episodic.

1) Static – The Environment is always static → Ex: Chess –
The chess board will remain the same, S & L board. (does not change w time)

Dynamic – Env. changing w time
Eg: Self-driving cars

5) Discrete – S&L board has discrete set of places where the outcome can be.

Continuous – Self driving car
The outcome is continuous as they go on roads moving continuously so at every instance the outcome (location of car) is different.

6) Single agent – In a game of chess, a human is playing against a single AI Agent.

*AI Agent Environment types are according to the env perceived by the agent & not by us.

Eg: Stone, Paper, Scissors
→ Fully Observable
→ Stochastic
→ Episodic
→ Static
→ Discrete
→ Single Agent

Justify why so?

* Agent Types

Basic types in order of increasing generality –

1) Simple Reflex agents
2) Reflex agents with state/model (Model based)
3) Goal based agents
4) Utility based agents
5) Learning agents

1) Works by finding a rule ~~to~~ whose condition matches the current situation (as defined by the percept) & then doing the action associated with that rule.

Eg: If car in front is breaking, then inline breaking will be initialed

3) Goal based agent

AI agent needs to attain some goals.

4) Utility

Goal based — not enough — for high quality behaviour.

Eg: Self driving cars

Goal to reach from A to B
But utility → goal attained w some performance measures like did it reach safely

5) learning

- agents which can learn from past experience or it has learning capabilities.

- learning element
- critic
- Performance element
- Problem Generator

★ Problem Formulation

Need to define/formulate the problem in such a way that searching/sorting algorithms can be applied to it.
Need to be represented in State/Space Representation.

Outcomes represented as states & all possible states form the state space.

After every action, check the state attained is desired or not and accordingly apply the searching algo or not to get the desired state.

★ State-Space Representation

Each agent — abstract representation of the agent's environment.
It is an abstraction that denotes configuration of agent.

Initial State: description of the starting configuration of the agent.

An Action/Operator — takes the agent from one state to another.

Plan - Sequence of actions

Goal - set of desirable states of the world specified by a goal test which any goal state must satisfy

Path cost -
path cost → positive no. = sum of step costs

Apply permissible actions until goal state is reached

State space
- Intial state
- Goal state
- Permissible actions to go from one state to other
- Path cost

Examples:

1) Vacuum cleaner

Let the world contain just 2 locations

Each location may/may not contain dirt & the agent (vacuum cleaner) may be in any room.

8 possible states

State: one of the 8 states

Operators: move left, move right, suck

Goal test: no dirt left in any square

Path cost: Each action costs 1.

Intial State: Both rooms are dirty & vacuum cleaner is in any one of the rooms.

2) Missionaries & Cannibals

3 missionaries & 3 Cannibals • need to cross a river.

1 boat can carry 1 or 2 people

Constraint: No. of Cannibals should not outnumber the missionaries at any given point in time or at any place otherwise they will be eaten.

State: $\langle m, c, b \rangle$ (m: no. of missionaries, c: cannibals, b: pos. of boat)

Intial state: $\langle 3,2,1 \rangle$

Goal state: $\langle 0,0,1 \rangle$

3) Cryptarithmatic

letters stand for digits
Aim - to find a substitution of digits for letters such that resulting ~~sum~~ sum is arithmatically correct.

Usually each letter must stand for a digit which is different.

Ex:

FORTY	29786	F=2,
+ TEN	850	O=9,
+ TEN	+ 850	R=7,
SIXTY	31486	etc.

States: a cryptarithmetic puzzle with some letters replaced by digits

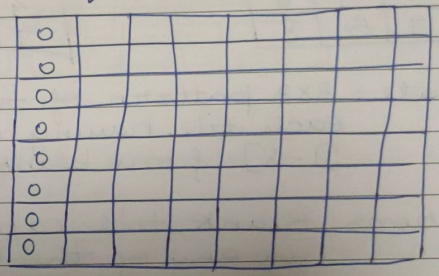
Operators: replace all occurrences of a letter with digit not already appearing in the puzzle.

Goal test: puzzle contains only digits & represents a correct sum

Path Cost: Zero. All sol^{n} equally valid

4) 8-Queens Problem

State: 8×8 chessboard (Empty, or one queen placed in first column in each row)



o = Queen

Goal: all Queens on chessboard such that no 2 queens attack each other.

Actions: Placing one queen in one square

Path cost: No. of moves

5) 8-Puzzle Problem

Intial State

1	2	3
	4	6
7	5	8

Goal State

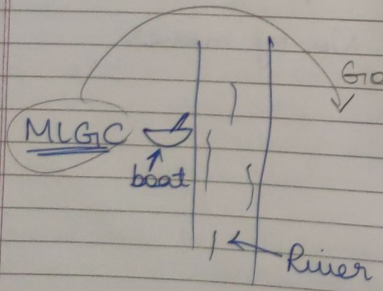
1	2	3
4	5	6
7	8	

State: 3×3 pattern of tiles with each tile having a number [1-8] & one blank tile.

Actions: Blank tile can move in any of the 4 dirⁿ, not diagonally & take one step at a time.

Cost: Can be no of moves.

6) Man, Lion, Goat, Cabbage (MLGC)



Constraint: LG ✗ GC ✗

<(M,L,G,C)< 7

7) Bottle-jug problem

m litre jug & n liter jug initially empty

Use jugs to measure d liters of water where d<n

State: (x, Y) — X= amt of water in jug1
Y= " " " " jug 2

★ Search Strategies

Search Strategies evaluation in terms of four criteria:

- Completeness: Strategy gurantead to (most essential) find a solⁿ when there is one?
- Time complexity: how long does it take to find a solⁿ?
- Space complexity: how much memory does it need to perform the search?

- Optimality: does the strategy find the highest quality solution when there are several different solutions.

  Able to solve the sol^{n} ~~is b~~ in the best possible way.

1/02/23

Module 2:

★ Uninformed Search (Blind Search)

→ searching with no additional information

→ Goal may be hard to find:
  - ↦ Shortest path from initial to Goal
  - ↦ Any path " " " "

Given state space Graph, how to obtain

- Search all possible paths
- Eliminate cycles from path

Let fringe be a list containing the initial state loop.
if fringe is empty, (→ all nodes have been explored but goal not found)
return failure

Node ← remove-first(fringe) (first node in a fringe, check if it's the goal node)
if Node = goal,
then return the path from initial state to node (found goal)
else generate all successors of Node and merge the newly generated node into fringe. (find new successors & merge into fringe)
End loop.

(how to merge will be different for diff blind search techniques/ algo)

★ fringe = open = Queue = sequence = front tire

★ BFS

Expand shallowest nodes first adding generated nodes to the back of the fringe"

(How newly generated nodes are merged to fringe)

→ also called EN-QUEUE-at-End

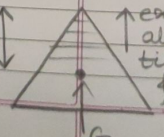
Put this in place of prev algo.

BFS Summary:
→ Uninformed (finds the path of min length to the goal)
→ FIFO (Queue) (as nodes at back)
→ expanding shallowest
→ complete/optimal (if all costs are same)

(complete: then finite & soln exists; optimal: only)

let d = depth of the shallowest goal
m = depth of search tree = max depth
b = every node as atmost b children or exactly b children (branching factor)

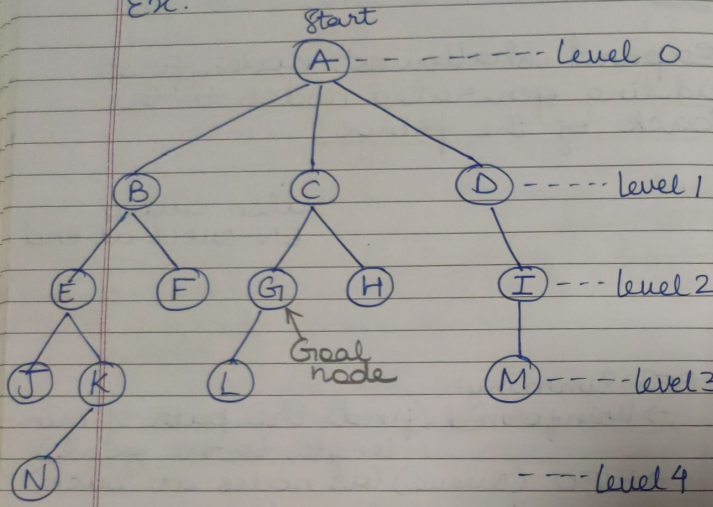
Dis-adv: has exponential time & space complexity → $O(b^d)$



expands all nodes till depth (d-1) & few nodes till depth d.

b: branching factor - no. of children every node has
d: depth of soln

Ex:



Fringe ~~A~~

BCD → Remove A and add successors back

~~B~~DEF Remove first element & add its successors at the back of the queue.

~~BE~~FGH

~~E~~FGHI

~~F~~GHIJK

~~G~~HIJK

↓

G1 = Goal

★ General algorithm for graph search

Let fringe be a list containing the initial state

Let closed be initially empty. (or visited)

★ Extra step ★

Loop

If fringe is empty, return failure

Node <- remove-first (fringe)

If Node is goal
then return the path from initial state to Node S.

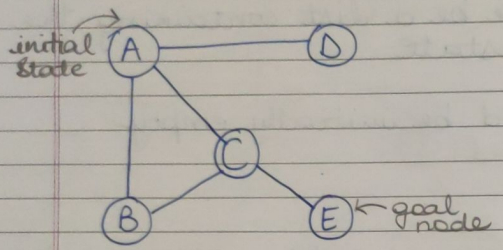
Else put Node in closed

Generate all successors of Node S

For all nodes in S
if m is not in closed
Merge ~~it~~ m into fringe

End loop.

Ex: BFS Graph traversal



Fringe/Open	Visited/Closed
A	
BCD	A
~~DC~~	AB
DCE	ABC
CE	ABCD
E	ABCD
	ABCDE

We will not add A because it is in the visited list

Drop C, because C is already in the visited list

E = Goal — E entered the fringe because of C & C entered because of A.

∴ Path : A → C → E [minimum length path]

★ DFS

Expand the deepest node first add the generated nodes to the front of the fringe.

→ Uninformed Search
→ Uses LIFO Queue (Stack)
→ Deepest node
→ non-optimal (explores left tree first)
→ Complete? NO (because if it is an infinite tree, impossible to find solution)

Graph Search
That avoids repeated states & redundant path is complete in Finite State Spaces.

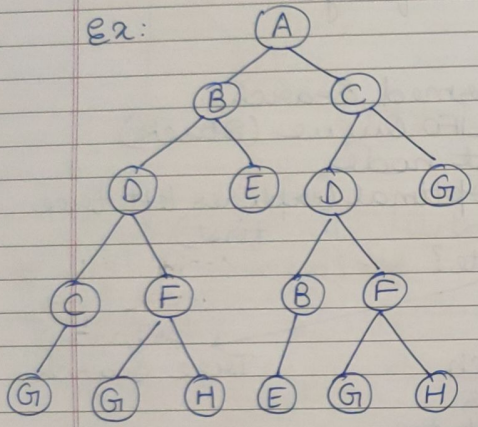
Tree Search

Time Complexity:
$O(b^m)$
b = branching factor
m = max depth of any node (m = ∞ if tree is unbounded)

Adv of DFS

→ Space complexity: O(b.m)

Ex:

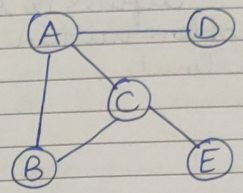


A

BC

DEC	DG
CFEC	BFG
GFEC	EFG
FEC	FG
GHEC	GHG
HEC	H
EC	G
C	
DG	

Ex 2:



A	–
BCD	A
CCD	AB
~~ABECD~~	ABC
~~CD~~	ABCE ← If E = goal - Stop
–	ABCED ← for full graph traversal

★ DLS

Depth Limited Search

→ it avoids the pitfalls of DFS which is ∞ path by imposing a cutoff on the maximum depth of a path.

→ DFS with a depth limit l. Algo treats the node at l as it has no successors nodes further.

→ In this algo, DLS can be terminated with 2 conditions of failure.

– Standard failure value: indicates that problem does not have any solution.

- Cutoff failure value: ~~indicates~~ defines no sol^n for the problem within a given depth limit.

Adv:
1) Memory Efficient

Properties:
1) Complete unless goal node is within the depth l.

2) Time
$O(b^l)$

3) Space
$O(b \cdot l)$ [Keeps all nodes in memory]

4) Optimal?
No (depending upon search algo)

★ Iterative Deepening Search

→ hard part about DLS is picking a good limit; known as diameter of the state space.

→ Iterative Deepening search is a ~~stat~~ strategy that solves choosing l by iteratively increasing the value of l till you find the goal node.

→ It appears like BFS but it ~~is~~ follows DFS.

→ Properties:
- Complete? Yes.
- Time: $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d$
  $= O(b^d)$
- Space: $O(bd)$
- Optimal? Yes, if step cost = 1
  Otherwise not.

→ Disadv:
wasteful as it generates states multiple times.

★ Bidirectional Search

Starts searching from both the ends.
We can use diff algos in both the direction according to your convenience.
Branching factor should be same for both sides.

Main problem: Don't know the back side – where to start from

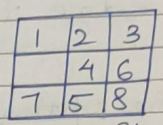
Only works for finite spaces

Complete? Yes
Optimal? Yes

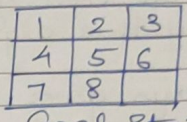
Time & Space: $O(b^{d/2})$

Disadv:
Requires a lot of memory space because we need to store nodes from both sides.

Q. 8 puzzle problem

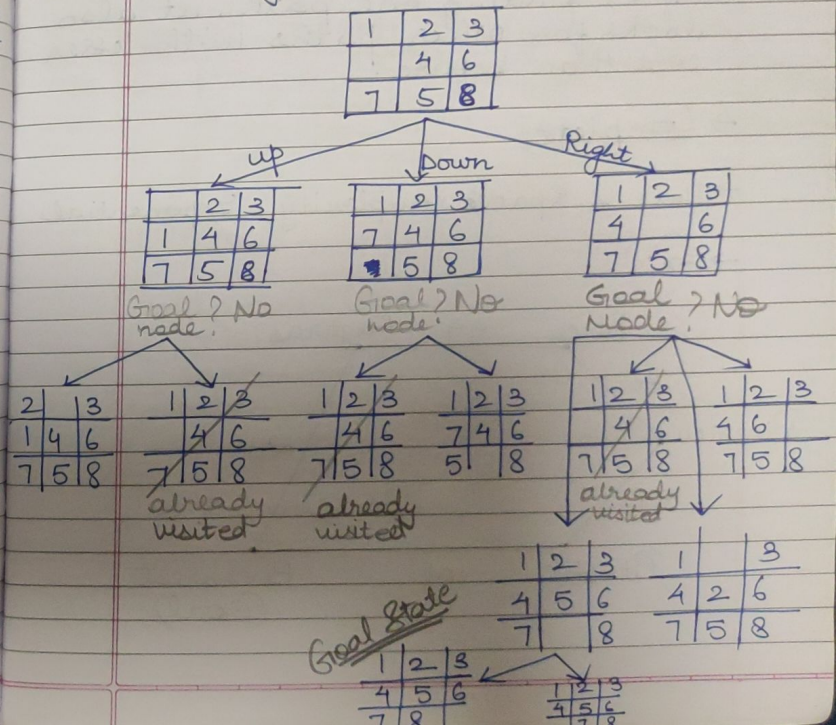


Initial State



Goal State

Blank tile can move up, down, left or right.



★ Uniform Cost Search (UCS)

→ Need the cost of the edge to apply this algo.

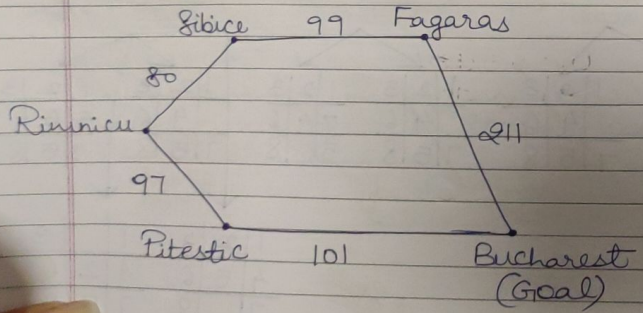
→ always gives you the optimal soln because we are expanding (min-cost) on the basis of cost & keep on going till the goal node is found

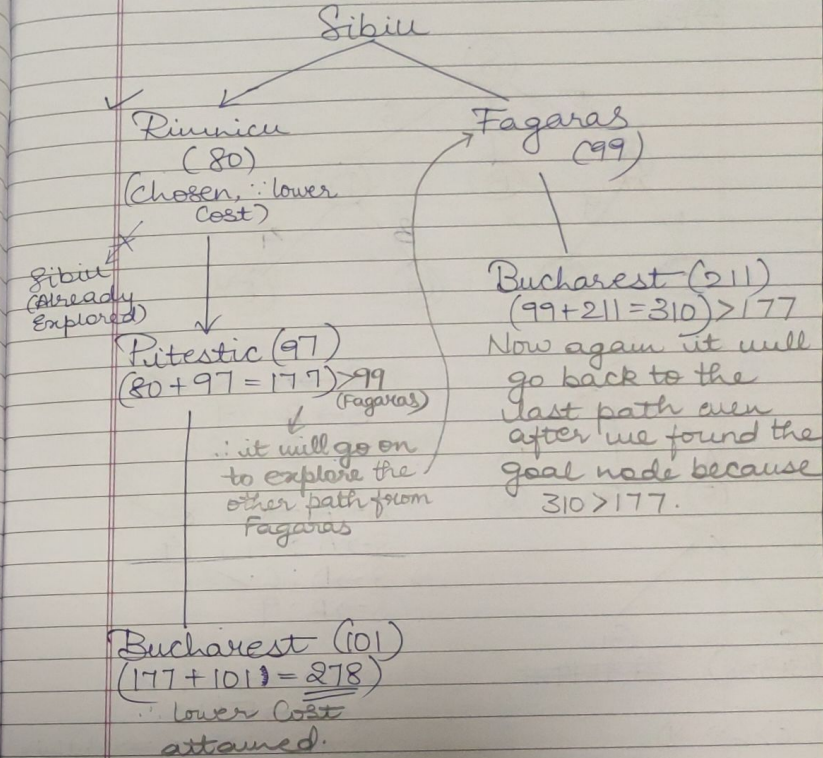
→ After finding one path, it also looks for other paths with lesser cost than before.

→ Complete

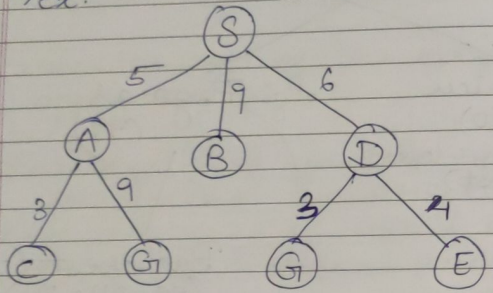
→ Time & Space complexity: Exponential

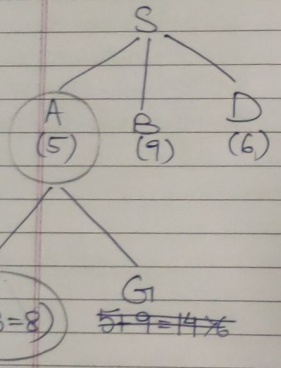
→ Ex:





→Ex:





8 > 6 ⇒ ... X

⇒ S → A : 5 ✓
S → B : 9
S → D : 6

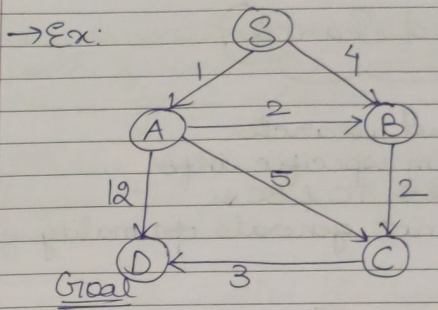
⇒ S → A → C : 8 ✓
S → A → G : 14

⇒ S → D : 6 ✓
S → B : 9

⇒ S → D → G : 9 ✓
S → D → E : 10
S → B : 9 ✓ ⇒ Nothing found
S → A → G : 14

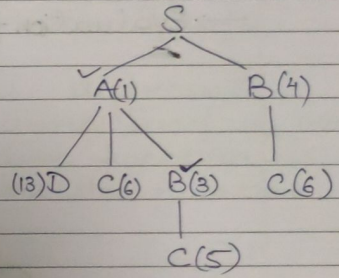
⇒ S → D → G : 9 (Goal node found)

→Ex:



S → A : 1 ✓
S → B : 4

S → A → C : 6
S → A → D : 13
~~S → B : 4~~
S → A → B : 3 ✓
S → B : 4



S → A → C : 6
S → A → D : 13
S → B : 4 ✓
S → A → B → C : 5
~~S → B → C : 6~~

S → A → C → D : 9
S → A → D : 13
S → B → C : 6

S → A → C : 6
S → A → D : 13
S → A → B → C : 5 ✓
S → B → C : 6

Explore both

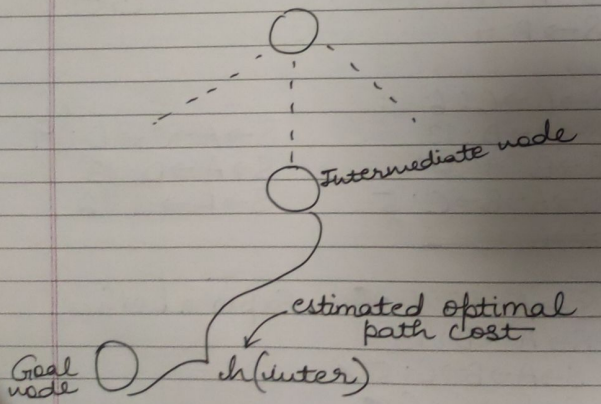
★ Informed Search

- → guided search
- → problem specific info
- → quick ⇒ TC & SC ↓
- → does not generate optimality

→ Evaluation fⁿ – f(n) – to calculate the ~~est~~ estimate cost

↓

in most cases – called the Heuristic funcⁿ. h(n)



The estimated cost of the cheapest path from the state at node n to a goal state....

OR

It's the estimate of optimum cost from the current node n to the goal node....

.....is called the heuristic funcⁿ h(n)

[Problem Specific]

Different algos use different Evaluation functions and in most cases the Evaluation function is given as the heuristic function.

Choosing the correct heuristic value/Evaluation function is the most imp step to find the optimal path quickly.

Ex:

Minimum dist b/w the cities will be a straight line → which will be used to calculate the lowest estimate cost

∴ h(n) → Euclidean dist.

Ex: 8 puzzle problem

initial/intermediate state

2	8	3
1	6	4
	7	5

Goal state

1	2	3
8		4
7	6	5

Can be many ways/ functions can be used

① $h(n)$ → how many tiles are out of posⁿ?

$h(n) = 1+1+1+1+1 = 5$

②

$h(n)$ → Cityblock dist (2D dist) or Manhattan dist

$h(n) = 1+2+0+1+1+0+1+0 = 6$

① ⑥ ④ ⑦ ⑤

2 →	8	3
1 ←	6	4
←	7 ↓	5

★ Best First Search

→ Algo:

Let fringe be a priority queue containing the initial state.

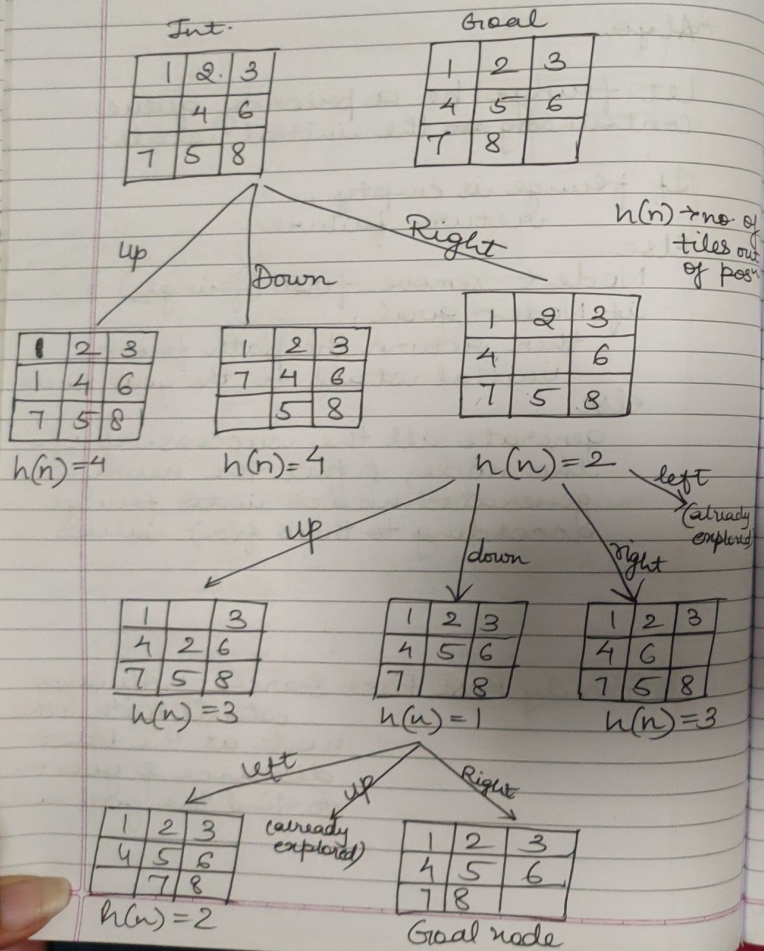
If fringe is empty,
    return failure.

Else
    Node ← remove_first (fringe)
    if Node = goal
        then return the path from initial state to the goal node
    else
        generate all the successors of the node, & put the newly generated nodes into fringe according to their $f(n)$ values

↑ Evaluation funcⁿ

◉ Greedy Best First Search → Always starts with the node at the least distance & goes on to find the goal node

Ex:



Cost (here) can be taken as the no. of hops because it is not given.
∴ cost = 5

★ A* Algorithm

→ Complete & Optimal

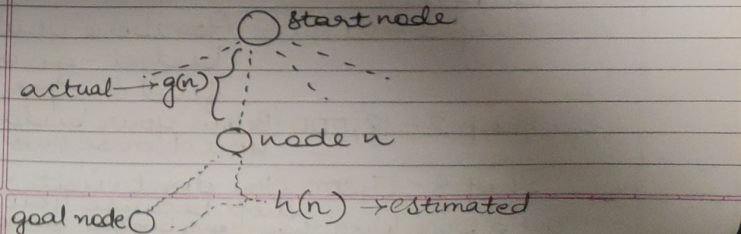
→ $f(n) = g(n) + h(n)$ [whereas Best F.S. → $f(n) = h(n)$]

→ like uniform cost search

→ $g(n)$ = actual cost of the path from the start node to the node n

→ $h(n)$ = estimated cost of cheapest path from node n to goal node

→ $f(n)$ = estimated cost of cheapest solution through node n (start – goal)



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

↓

* props. of $h(n)$

If these props. aren't satisfied, the A* is not ~~compo~~ optimal (only complete)

1) Admissibity

An Admissible heuristic is one that never overestimates the cost to reach the goal node.

The estimated cost should always be underestimate of the actual cost b/w node n to goal node.

If you overestimate — you don't get the optimal solution.

Eg: You want to buy a mobile phone and with your desired features, it costs Rs. 20,000 (estimated)

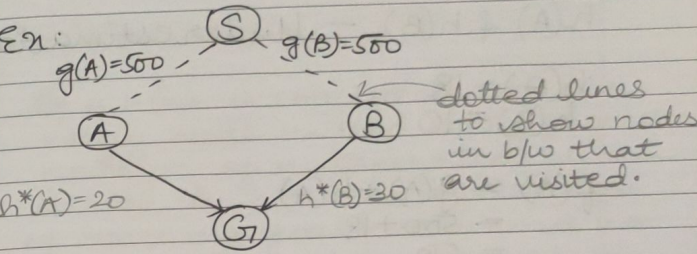
You go on to find it in different shops.

Shop 1 - 21,000 (~~too~~ above your budget - go on to find cheaper sol^n)

Shop 2 - 18,000 (Below your budget - you choose this option)

Shop 3 - 17,200 (You don't visit this shop, because you already got a price lower than your budget ⇒ not optimal)

Ex:



$h^*(A)$ & $h^*(B)$ → Actual values of the path, we assume these here so as to understand the $h(A)$ & $h(B)$ values.

$h(A)$ & $h(B)$ — Overestimate

$h(A) = 60$

$h(B) = 50$

~~h(n)~~

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

$= 500 + 60$

$= 560$

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

$= 500 + 50$

$= 550$ ← Explore this

$f(G_1) = g(G_1) + h(G_1)$

$= (500 + 30) + 0$

$= 530$

Now, $f(G_1) < f(A)$

∴ we won't explore $f(A)$

∴ The Sol^n will be S - B - G_1.

$h(A)$ & $h(B)$ — Underestimation

$h(A) = 18$
$h(B) = 10$

$f(A) = g(A) + h(A)$
$= 500 + 18$
$= 518$

$f(B) = g(B) + h(B)$
$= 500 + 10$
$= 510$ ← Explore this

$f(G_1)_{\text{through A}} = (500+20) + 0$
$= 520$

$f(G_1)_{\text{through B}} = 500 + 30) + 0$
$= 530$

here $f(G_B) > f(B)$
∴ it goes on to explore $f(A)$

Now $f(G_A) > f(A)$

and $f(G_A) < f(G_B)$ ⇒ ∴ We will choose A→G as the solution

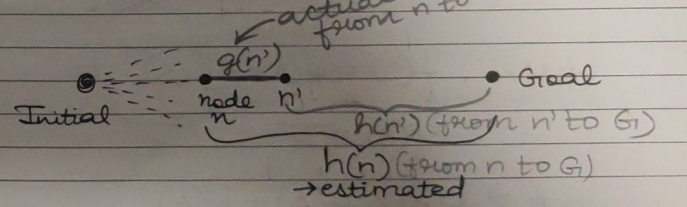
$h(n) \leq h^*(n)$
Underestimate → using a $h(n)$

2) Consistency

~~If~~ a $h(n)$ is consistent, if for every node n and every successor node n' of n generated by an action A, the estimated cost of reaching goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'

Every consistent $h(n)$ is also admissible but not vice versa.

A tree search version of A^* is optimal if ~~it~~ $h(n)$ is admissible; while graph search ~~it~~ version ~~is~~ of A^* is ~~optimal~~ if $h(n)$ is consistent



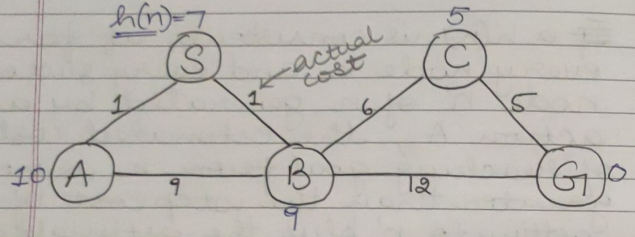
$h(n) \leq c(n, a, n') + h(n') = g(n') + h(n')$

Cost from n to n' ~~through~~ after performing action a.

(We don't use g(n') because that is from start to goal.)

(for understanding only, don't use this formula)

A* Example



S → initial
G1 → goal

Step 1: Start at node S;
Successors — A & B
S-A ⇒
S-B ⇒

Expanded node	Open
—	S7

actual estimated

here, $f(S) = g(S) + h(S)$
$= 0 + 7$
$f(S) = 7$

S-A ⇒ $f(A) = g(A) + h(A)$
$= 1 + 10$
$= 11$

S-B ⇒ $f(B) = g(B) + h(B)$
$= 1 + 9$
$= 10$

Expanded Node	Open
—	S7
S7	B10, A11

Step: 2: S-B
Successors of B are A, C, ~~B~~ G1

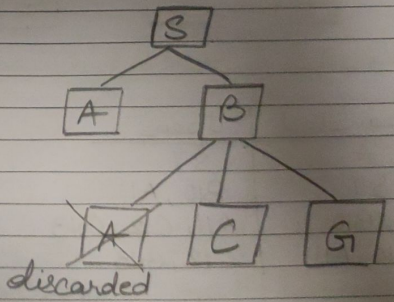
S-B-A ⇒ $f(A) = (1+9) + 10 = 20$

We will discard this as we have a node in the fringe with a lesser value (A11)

S-B-C ⇒ $f(C) = (6+1) + 5 = 12$

S-B-G1 ⇒ $f(G1) = (1+12) + 0 = 13$

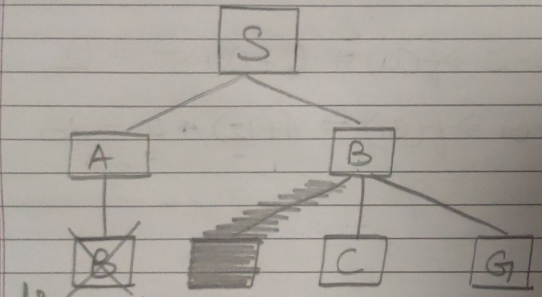
S-A path from step1 chosen as min f(n).



Expanded Node	Open
–	S7
S7	B10, A11
S7, B10	A11, C12, G13

Step 3: S-A
Successor is B
S-A-B $\Rightarrow f(B) = (1+9) + 9 = 19$ *Discard*

S-B-C chosen as min f(n)

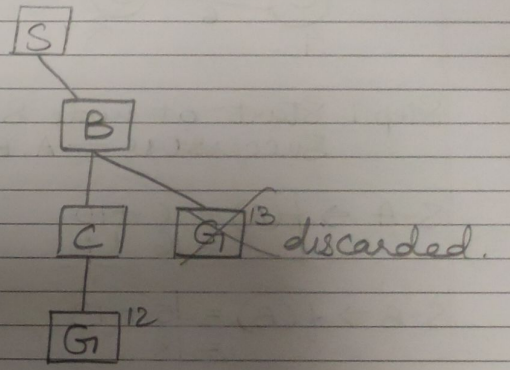


Expanded	Open
–	S7
S7	B10, A11
S7, B10	A11, C12, G13
S7, B10, A11	C12, G13
C12	G12
G12	Empty

Step 4: S-B-C
Successor is G1

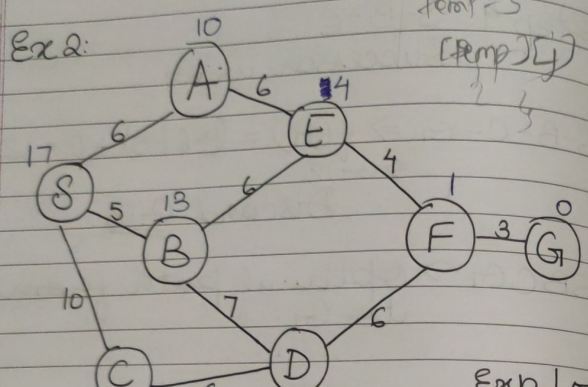
S-B-C-G1 $\Rightarrow f(G1) = (1+6+5) + 0 = 12$
$\Rightarrow$ G1 12
$\therefore$ Discard G1 13

SBCG1 → Optimal path from S to G1.



Expanded Node	Open
–	S7
S7	B10, A11
S7, B10	A11, C12, G13
S7, B10, A11	G12, ~~G13~~
S7, B10, A11, C12	G12
S7, B10, A11, C12, G12	Empty.

Ex 2:



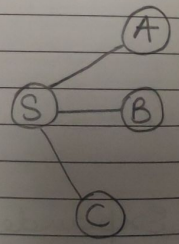
Exp	Open
–	S17

Step: 1: Start at node S;
Successors — A, B, C

S-A ⇒ f(A) = 6 + 10
= 16

S-B ⇒ f(B) = 5 + 13
= 18

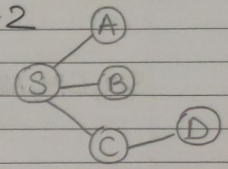
S-C ⇒ f(C) = 10 + 4
= 14



Exp	Open
–	S17
S17	C14, A16, B18

Step: 2: S-C
Successors — D

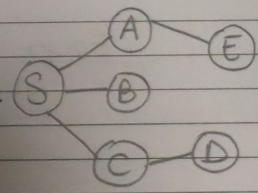
S-C-D ⇒ f(D) = (10+6) + 2
= 18



Exp	Open
–	S17
S17	C14, A16, B18
S17, C14	A16, B18, D18

Step: 3: S-A
Successors — E

S-A-E ⇒ f(E) = (6+6) + 4
= 16



Exp	Open
S17, C14, A16	E16, B18, D18

Step: 4: S-A-E
Successors — B, F

S-A-E-B ⇒ f(B) = (6+6+6) + 13
= 31 (Discarded :: of B

S-A-E-F ⇒ f(F) = (6+6+4) + 1
= 17

Exp	Open
S17, C14, A16, E16	B18, D18, F17

Step 5: S-A-E-F
Successors - D, G

S-A-E-F-D ⇒ f(D) = (6+6+4+6) + 2
= 24 (Discarded)

S-A-E-F-G ⇒ f(G) = (6+6+4+3) + 0
= 19

Exp	Open
S17, C14, A16, E16, F17	B18, D18, G19

Step 6: ~~SAEFB~~ S-B
Succ - E, D

S-B-E ⇒ f(E) = (5+6) + 4
= 15

S-B-D ⇒ f(D) = (5+7) + 2
= 14

Exp	Open
7) S17, C14, A16, E16, F17, B18	E15, D14, G19

Step 7: S-B-D
Succ - C, F

S-B-D-C ⇒ f(C) = (5+7+6) + 4
= 22 (Discard)

S-B-D-F ⇒ f(F) = (5+7+6) + 1
= 19 (Discard)

Step 8: S-B-E
Succ - [redacted], F

S-B-E-F ⇒ f(F) = (5+6+4) + 1
= 16

Exp	Open
7) B18	
8) D14	E15, G19
9) E15	F16, G19

Step 9: S-B-E-F
Succ – D, G

SBEFD ⇒ f(D) = (5+4+6+6)+2
= 23 (Discard)

SBEFG ⇒ f(G) = (5+6+4+3)+0
= 18

Exp	Open
10) F16	G18
11) G18	Empty

Step 10: SBEFG is optimal path from S to G

★ A* Algorithm

Graph G →

Cost of Edges —

h(n) of nodes

Calculate f(n) = g(n) + h(n)
↑ actual
↑ estimated

22/2/23

★ Adversarial Search and Game playing

Common terms in game theory

→ Game: Any interaction b/w 2 or more players in which each player's payoff is affected by their decision & the decisions made by others

→ Players

→ Payoff

→ Zero-Sum: Eg: Win=1, Loss=-1
Sum = 0.

→ Non zero-sum

→ Simultaneous

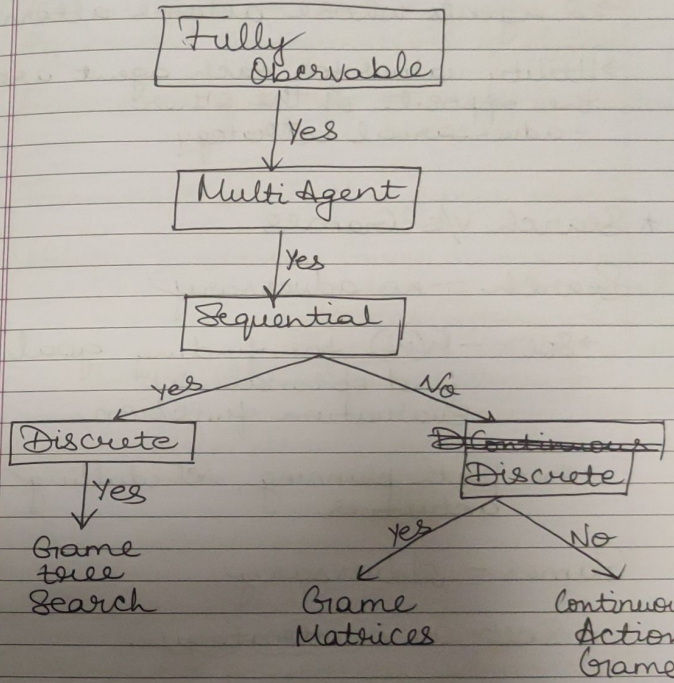
→ Sequential

→ Non-Cooperative: Players are playing against each other

→ Cooperative: forge alliances & play

→ Complete/Incomplete Information
→ Imperfect Information

★ Environment Type



* Typical AI Assumptions

→ 2 agents whose actions alternate

→ Utility values for each agent are the opposite of the other
- adversarial strategy

* Search v/s Games

Search — no adversary

→ Soln — h(n) for finding goal
- find optimal soln.
- Evaluation function

→ Ex - path planning, schedulling activities.

Games — Adversary

→ Soln is a strategy.

→ Optimality depends on opponent

→ Time limits force an approximate soln

→ Evaluation Function

→ Ex - Chess, checkers, Othello

* Game Setup : Games as Search

Initial State = Initial posn
Eg: board configuration of chess

Successor function : list of (move, state) pairs

Terminal test : Is the game over?

Utility function : Gives numerical value of terminal states (after which the tree can't grow)
Numerical Outcome for the game. Eg: win (+1), lose (-1) & draw (0) in tic-tac-toe or chess.

* Basic Strategy

- Grow a search tree
- One player can move at each turn
- Assign a payoff to each final posn called a utility.
- Can propogate values from the final positions
- Assume the opponent always makes moves worst for us
- Pick best moves on own turn.

* 2 player games.

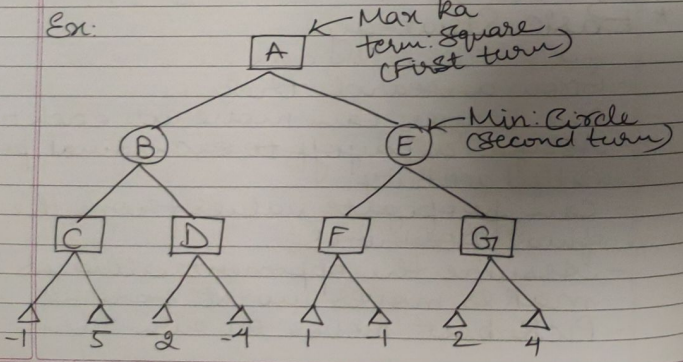
→ Two players
→ zero Sum
→ Perfect Information

2 players called: MAX & MIN
Assume MAX player makes the first move
If min/max not specified, assume the first node to max.

MAX & MIN take turns to maximize & minimize the utility func^n until one wins.

Ply of the node - no. of moves needed to reach that node.

Ex:



* Brute Force Search

Start at root node & generate entire search tree till the leaf positions assuming that the tree is finite.

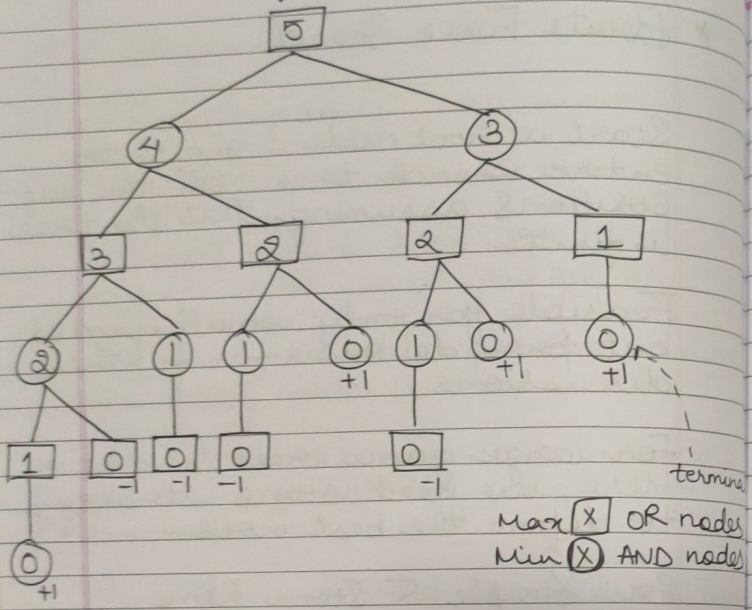
Feasible for only small games, but provide basis for further discussions.

For large game trees could be ∞. Here, we need some strategy to define the best move.

Ex: Simple 5-Stone Nim

- Played w 2 player & pile of stones
- Each player removes 1 or 2 stones from the pile
- Player who removes last stone wins the game.

Game tree for 5-Stone Nim



Grow the tree till all terminal nodes are 0
Now deciding the utility value.

If Max wins = +1
If Min wins = -1

★ Minimax Strategy (Min-max Algorithm)
: Look ahead & reason backwards

Minimax Theorem: Every 2-person zero-sum game is a forced win for one player, or a force-draw for either player, in principle, these optimal min-max strategies can be computed. Finding the optimal strategy for MAX assuming an infallible MIN opponent.

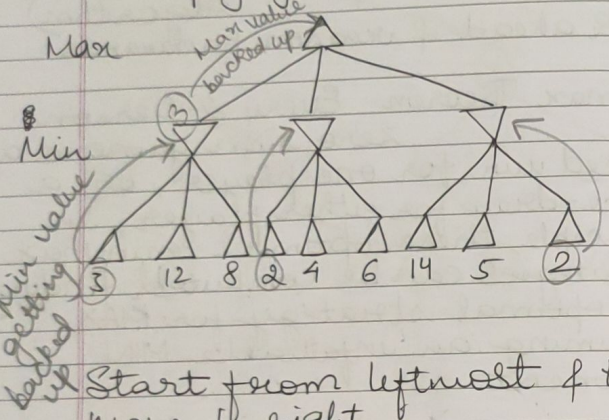
Need to compute this all down the tree.

→ If the backed up value at root is positive, if MAX plays judiciously, MAX can force the win.

→ is negative, no matter how well the MAX plays, if MIN plays extremely intelligent, MIN can force a loss on MAX.

Max node will select the Maximum value & Min node — minimum.

★ Two ply game tree



Start from leftmost & then move to right

★ Minimax Algorithm

Complete depth-first exploration of the game tree

Assumptions:
Max depth = d
b legal moves at each point

Eg: Chess: d - 100, b - 35

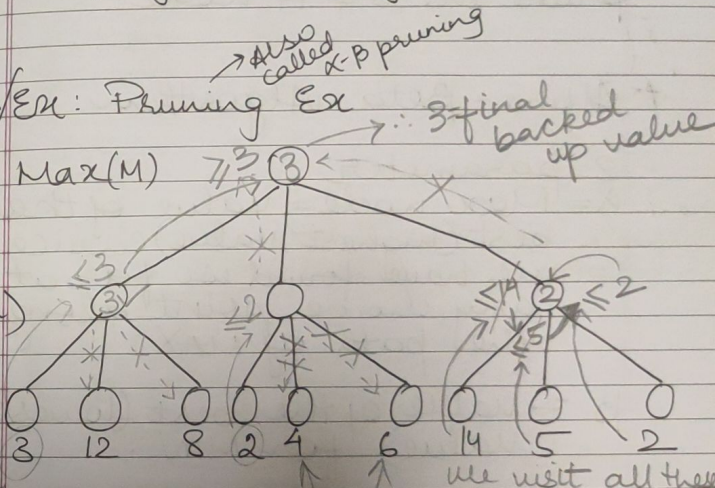
Time - $O(b^d)$
Space - $O(bd)$

★ Multiplayer games

Single minimax values become vectors (x, y, z)

All players will try to maximize.

Q/Ex: Pruning Ex → Also called α-β pruning



only visited the first node ⇒ can't be greater/equal to 3

We visit all these nodes as ≤5, ≤14 can be greater than 3 ≥3.

We won't visit these nodes as we only want the value for the root node and these 2 won't change the decision as (≤2) can't be ≥3

Not visiting the nodes 4 & 6 ⇒ we are pruning them (cutting)

We can increase the efficiency of α-β pruning by something called "Node ordering" according to our convinience
In the last example, if we order the 2, 5, 14, we can prune the 5 & 14 node.

* Alpha-Beta Algorithm

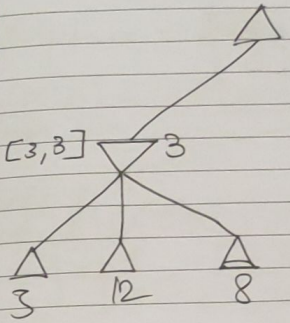
2 parameter
α = Max node = value of the best (highest value) choice that we have found so far at any choice point along the path of MAX

β = valle of the best (lowest value) for MIN.

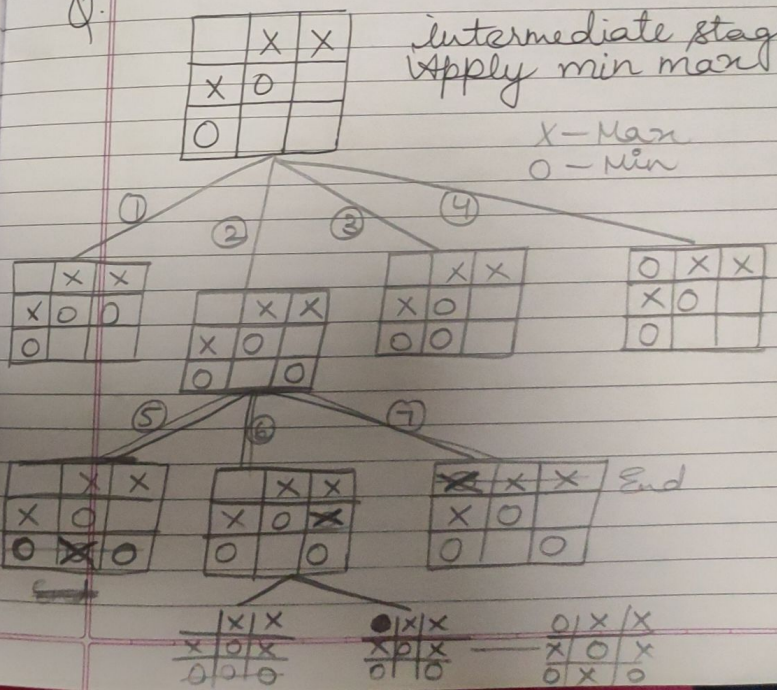
Depth First Search

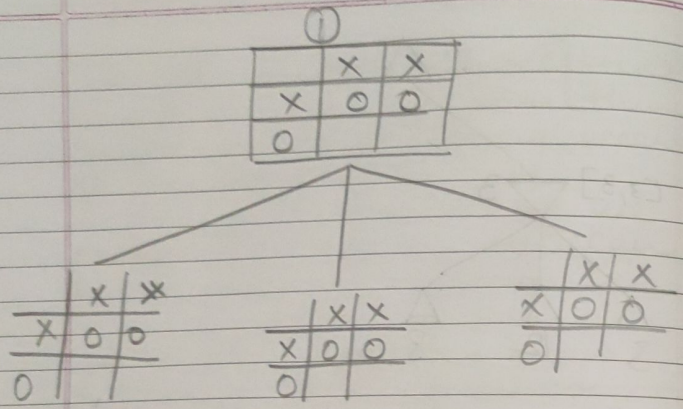
$(\alpha, \beta) = (-\infty, +\infty)$

↑
initialised as α goes to highest value so it starts from the min value.



Q.



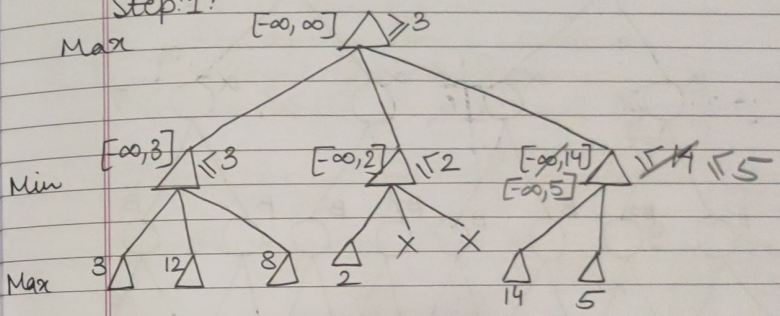


★ Alpha - Beta Pruning

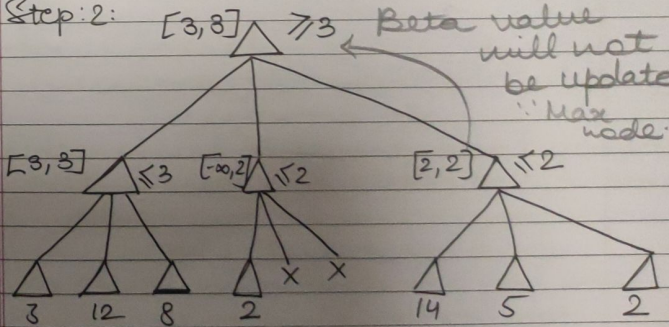
$(\alpha, \beta) \equiv (-\infty, \infty)$

Max Min

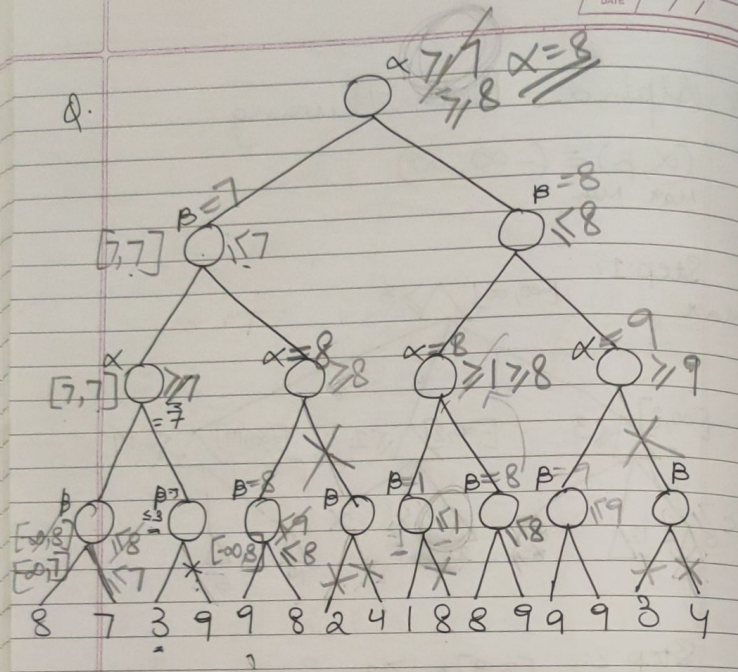
Step: 1:



Step: 2:



Q.



Saved computations of 8 nodes.

Deep Cut off

See these 2 values & comparing them we can cut off the other branch

★ Search Types

→ Backtracking State-Space Search

→ Local Search & Optimization
- ⇒ Hill Climbing
- ⇒ Escaping local Maxima: Simulated Annealing
- ⇒ General Algorithms

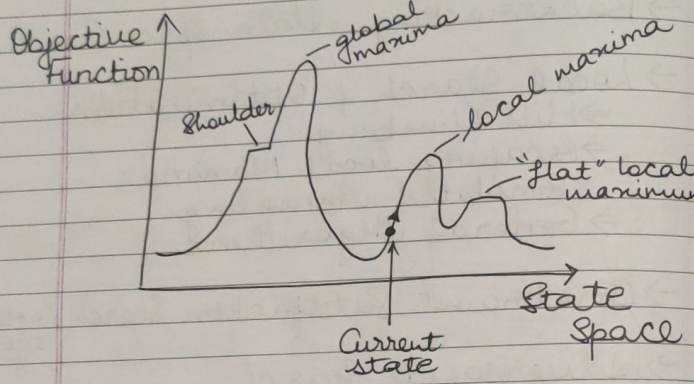
→ Constraint Satisfaction Search (after ISE)

→ Adversarial Search

★ Local Search & Optimization

- Local Search (decisions are local to that state)
- Idea: start with an intial guess at a sol & incrementally improve it until it is one
- You can maximize (Profit) or Minimize (Loss) ⇒ Useful for optimization.
- Adv:
  - Uses very little memory
  - Often reasonable sol^{n} in large or ∞ state space.

★ State-Space features



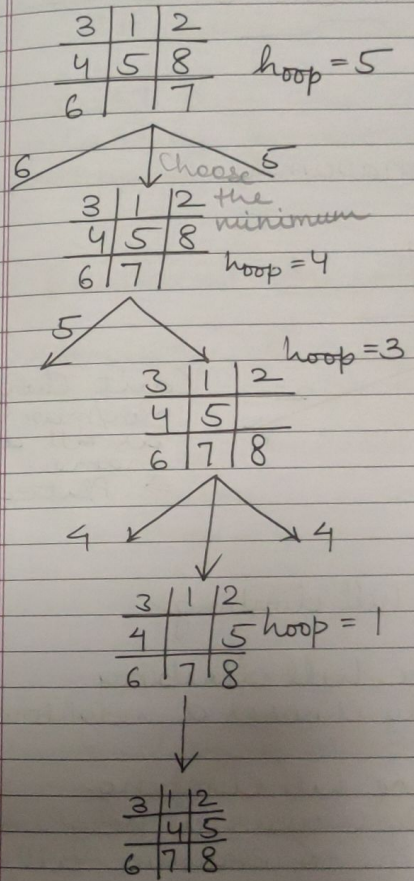
Intial guess will define if we get the local/global maxima/minima — Problem of local search algo. (→ we don't want the local value)

★ Hill Climbing (Greedy local Search)

→ Generate nearby successor states to the current state

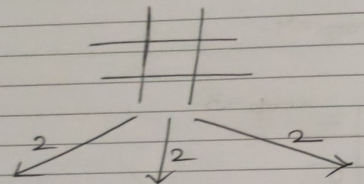
→ Pick best & replace the current state with that one

Ex: [Minize h]



hill climbing terms
→ local maximum
→ plateau
→ ridges

Ex - local maxima



Can't choose min/max value as all are same.
⇒ ∴ Plateau

Variants of hill climbing

→ Stochastic hill climbing
randomly chooses a neighbour

→ First Choice hill climbing
randomly chooses a neighbour & goes on to examine all.

Good strategy for thousands of neighbours.

Remedy to Drawbacks of Hill Climbing:
→ Random-restart hill-climbing

start diff searches from random starting posⁿ & stopping when a goal is found.

★ Simulated Annealing

↑ metallurgical process (a metal is heated at a high temp & then cooled down) to alter its properties

⇒ complete
⇒ efficient
⇒ Stochastic Optimization technique

↓ crucial parameter
→ cooling rate.

⇒ is a global opt. tech.

⇒ works well for discrete opt.

⇒ Based on a metallurgical metaphor

- Start with a temp (in this case - a variable) set very high & slowly reduce it.

⇒ Similar to hill climbing but will not always choose the best case but instead it will choose acc to the temp & how worse it is.

(POA)

⇒ Probability of acceptance is taken as of the worse case & then decided if it should be accepted or discarded.

⇒ This helps ~~to~~ us to minimize the probability of us getting stuck on local values

⇒ As we go on the temp decreases & therefore the POA decreases.

SA Algo [For minimization]

→ SA is a point method

→ begin at an initial point & high Temp T.

→ a second point is created in the vicinity of the initial point & ΔE is calculated.

- If ΔE = -ve (next < current), the new point is accepted (for minimization problems).
- Else the pt is accepted with a probability of $e^{\Delta E/KT}$ (K=1)
- This completes one iteration

In the next gen/iteration, again a new point is chosen, but now the temp is reduced according

Stopping Conditions:

→ till T becomes zero

→ Choose a small ∨ threshold & compare $E_{next} - E_{curr}$ & stop if its < threshold

→ A given minimum value of the temp has been reached

→ A certain no. of iterations (or) have passed w/o acceptance of a new sol^{n}.

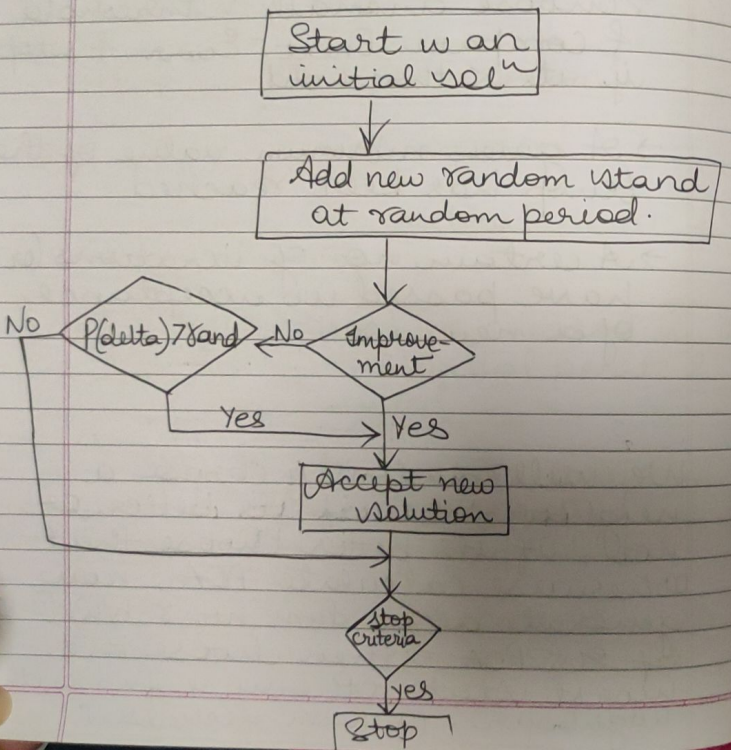
We will randomly choose a neighbour see if it's better or not, if it's better choose that otherwise calculate POA, now generate a random no. r b/w [0-1] if r ≤ POA ⇒ then choose the worst state otherwise find another neighbour. [If chosen, decrease T]

P(delta) = POA = 1 when c is very high.

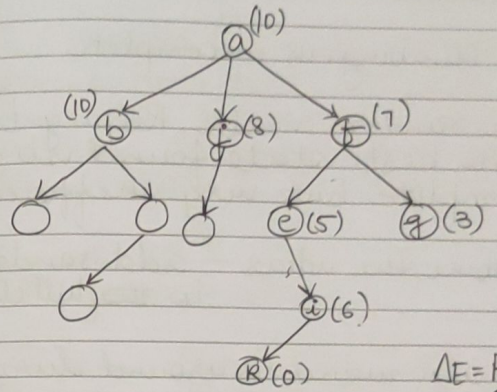
C = temp value (most probably)

P(delta) = POA = 0 when c is very small.

→ SA Flowchart



* SA Problem



$\Delta E = F_{next} - F_{curr}$ (objective func^n)

Current	Children	$\Delta E > 0$ → for minimization problem
a	—	
a	f7 f8 b10	ΔE = value(f7) − value(b10)
f	e5 g3	$\Delta E = -[7-(10)]$
	$\Delta E = -[5-7] = 2 > 0$	$\Delta E = 3 > 0$
e	i6	added the minus so that we get a true value
	$\Delta E = -[6-5] = -1 < 0$	
i	R0	
	$\Delta E = -[0-6] - 6 > 0$	
Goal node found.		

We randomly chose f7 no relation to its lowest value.

we chose e as $\Delta E > 0$

random number chosen = 0.5

$1 > 0.9 > 0.5$

prob = $\leftarrow$ Now, we have to calculate the probability to decide whether to choose this node or not

$= e^{\Delta E/KT}$

$p = e^{-1/10} = 0.9$ ↳ good prob ⇒ yes.

Intutions

* Hill Climbing is incomplete

* Pure random walk, Keeping track of the best state found so far is complete but very inefficient.

* Combine the ideas – add randomness to ~~the~~ hill climbing.

* the algo wanders around during the early parts of the search, & hopefully towards a good general region

* towards the end, the algo does a more forward search making a few bad moves.

* Theoretical Completeness:

if T lowers slowly enough, SA will find a global optimum with probability approaching 1.

Stochastic Algo (Random)

* Genetic Algorithms (GA)

are ~~sear~~ heuristic search & optimization techniques that mimic the process of natural evolution to find true sol^n to optimization problems.

* flowchart *
See from book

⇒ Representation of individuals
⇒ Selection Strategy

- Usually in binary
- individual is a string w each element in the string called a gene.

- Process of choosing 2 parents from the population for crossing

- Roulette wheel Selection, Tournament Selection, Elitist Selection, Rank Sel.

- ⇒ Reproduction

- Random Pairing of selected individuals

- Random selection of cross-over points.

Single-point
~~Single~~ cross over

Parent 1	10110 | 010	tails
Parent 2	10101 | 111	

Choose a point = 5

draw a line after that pt & exchange the tails to find the next generation

Child 1	10110111
Child 2	10101010

Each gene can be altered by a random mutation

↓

Changing 1 in 20 bits