Heuristic Search Techniques

1. It is a type of a search performed by AI that looks to find a good solution, not necessarily a perfect one, out of the available options.

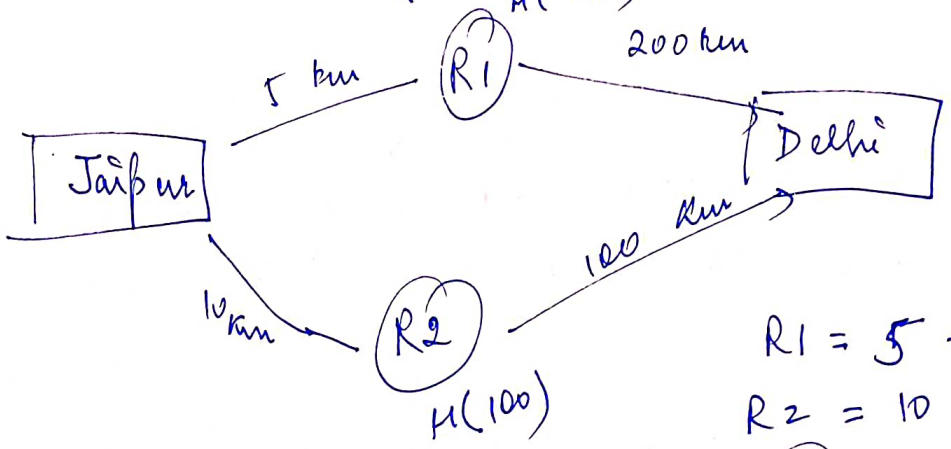
2. This technique makes decisions by ranking every available choices at each branch of a search and then chooses the best option of those presented.

3. Rather than focusing on finding an optimal solution like other search methods, heuristic searching is designed to be quick, and therefore finds the most acceptable option within a reasonable time limit. There are several types of heuristic search Algorithms that can be utilized depending on the task.

- It is a technique designed to solve a problem quickly.
- Tries to optimize a problem using Heuristic function. (tries to solve problem in minimum steps/cost.)

Heuristic function

- It is a function $H(n)$ that gives an estimation on the cost of getting from node 'n' to the GOAL STATE
- Helps in selecting optimal node for expansion.

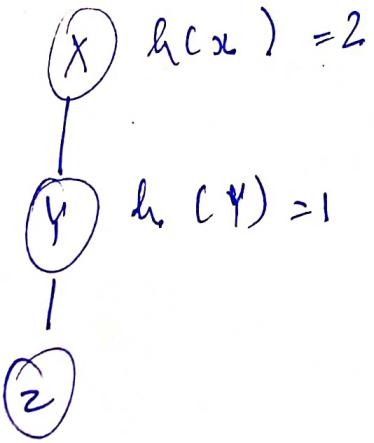


R1 = 5 + 200 = 205

R2 = 10 + 100 = 110

R2

→ It informs that how far would be your goal state



Types of Heuristic search :

1) Admissible : In this heuristic function, never overestimates the cost of reaching the goal.
It underestimates the the cost of reaching the goal.

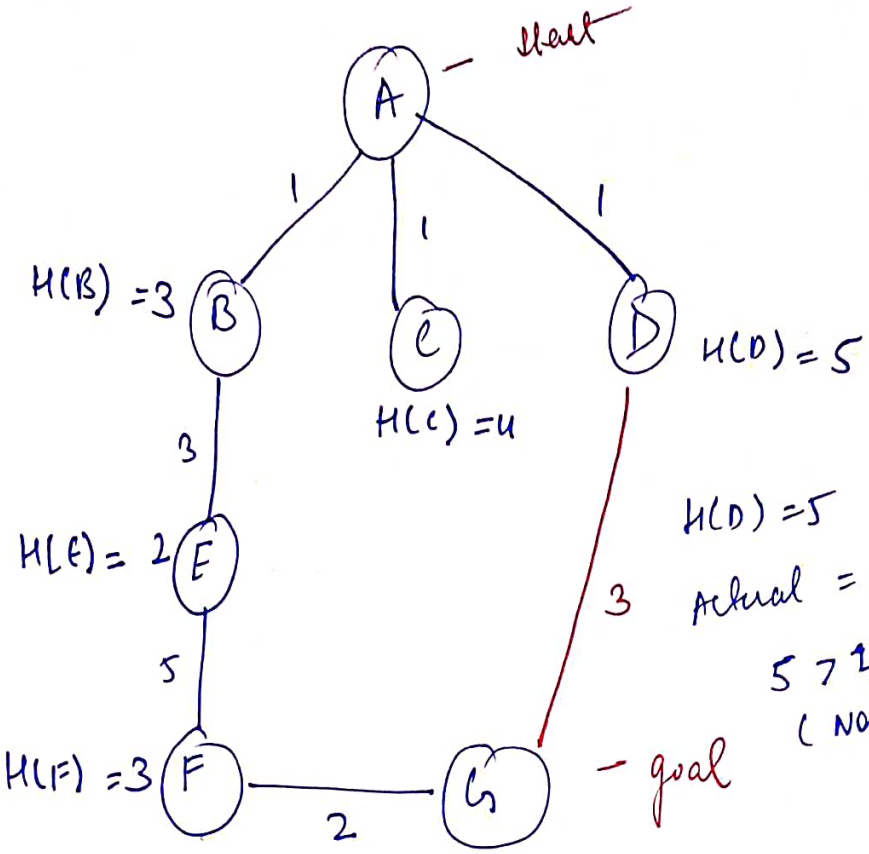
→ $h(n)$ is always less than or equal to actual cost of lowest cost path from 'n' to goal.

$$H(n) \leq H'(n) \quad \{ \text{goal} \}$$

2) Non Admissible : It overestimates the cost of reaching the goal.

→ $h(n)$ is always greater than Actual cost of lowest cost path from 'n' to goal.

$$H(n) > H'(n)$$



H(D) =5

Actual = 1+3 = 4

5 > 4

(Non - Admissible)

$F(n) = H(n) + G(n)$

B = 3 + 1 = 4 (select)

C = 4 + 1 = 5

D = 5 + 1 = 6

Actual = 1 + 3 + 5 + 2 = 11 -

3 < 11 (Admissible)

- Particularly useful in solving large and complex problems, solutions of which require infinite time. i.e. far longer.

Generate and Test

1) It is an informed search technique or heuristic search technique.

2) Uses DFS with backtracking.

Steps Involved / Algorithm : Generator / Test

1) Generate a possible solutions
2) Test to see if this is a Actual solution.
3) If a solution is found, quit. Otherwise go to step 1.

- Also known as British Museum search Algorithm finding an object in the British museum by wandering randomly

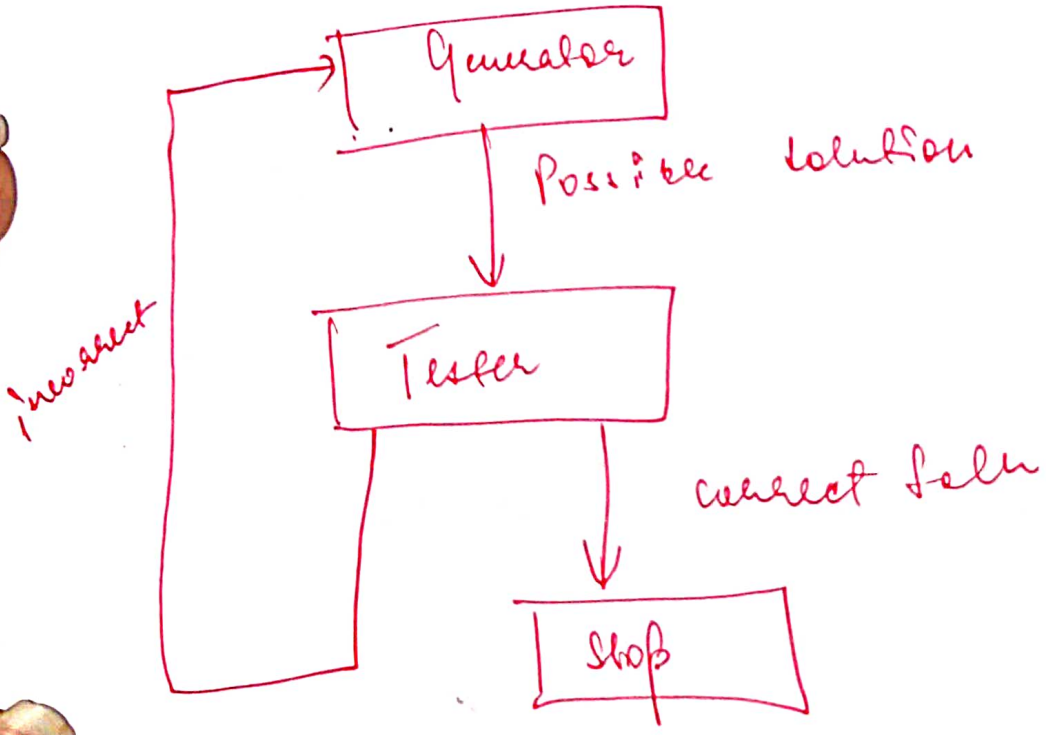
Properties Of Good Generator:

1) Complete
2) Non-redundant
3) Informed

- If problem space is large, it may take very long time.

Systematic Generate and test

- While generating complete solutions and generating random solutions are 2 extremes there exists another approach that lies in b/w.
- The Approach is that the search process proceeds systematically but some paths that unlikely to lead the solutions are not considered.
- This evaluation is performed by heuristic fn.
- DFS with Backtracking can be used to implement Systematic Generate and test.



Hill Climbing

- Local search Algorithm which continuously moves in the direction of increasing elevation / value to find the peak of mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- It is a technique used for the mathematical problems.
- It is also known as greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- Variant of Generate and test in which feedback from test procedure is used to help generator decide which direction to move in search space.
- No Backtracking
- Always moves in single direction.

not better – stop

if new state is better than current state, new state = current state

Type of hill climbing →

1) Simple hill climbing : It only checks its one neighbour or successor state, and if it finds better than the current state, then move else be in the same state.

*Features

→ less time consuming

→ less optimal soln and soln is not guaranteed

Algorithm for simple hill climbing.

1. Evaluate the initial state, if it is a goal state then return success and stop.
2. loop until a soln is found or there is no new operator left to apply.
3. Select and apply an operator to the current state.
4. Check new state
   a. If it is goal state, then return succes and quit
   b. Else if it is better than current state then assign new state as a current state.

c. Else if not better than the current state then return to step 2.

Step 5 Exit

Steepest - Ascent Hill climbing

- Variation of simple hill climbing.
- It examines all neighbouring node of current state and selects one Neighbour node which is closest to the goal state.
- It consumes more time as it searches for multiple Neighbours.

Algorithm for steepest Ascent hill climbing

Step 1: Evaluate the initial state, if it is a goal state then return success and stop, else make current state as initial state.

Step 2: loop until a soln is found or the current state does not change.

a. let succ be a state such that any successor of the current state ~~does not change~~. will be better than it.

b. For each operator that applies to current state.

a. apply a new operator and generate a new state

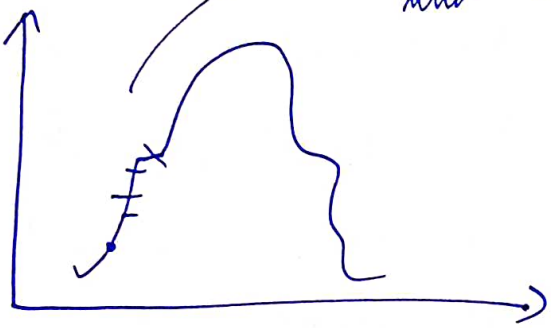
b. Evaluate the New state.

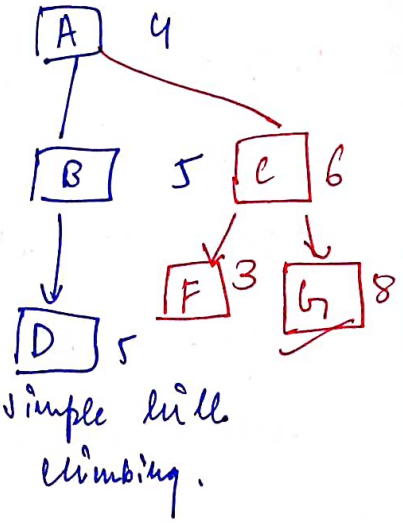
c. If it is a goal state, then return it and quit, else compare it to to succ.

d. If it is better than succ, then set new state as succ.

e. If the succ is better than the current state then set current state to succ.

Step 5 : exit.



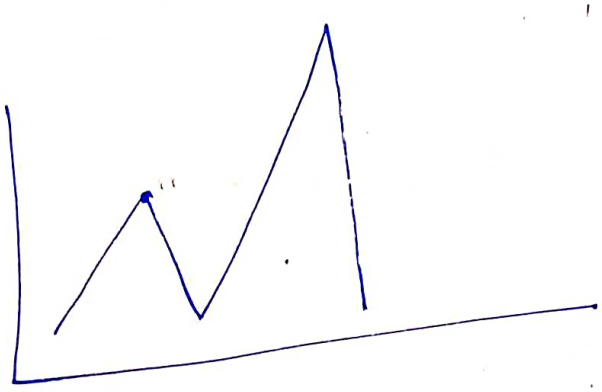


simple hill climbing.

-Deep mind / Alpha go

Problems in Hill Climbing:

1) local Maximum: peak state in the landscape which is better than each of its neighbouring states, but there is another state also present which is higher than the local maximum



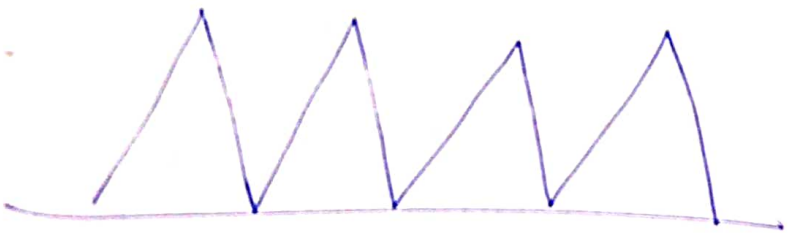
Solution: Backtracking technique → Maintain a list of visited states. If the search state reaches the Undesirable state, it can backtrack to the previous configuration and explore a new path.

2) Plateau : Flat area of search space in which all Neighbour states of the current state contain the same value, because of this algorithm does not find any best direction to move.

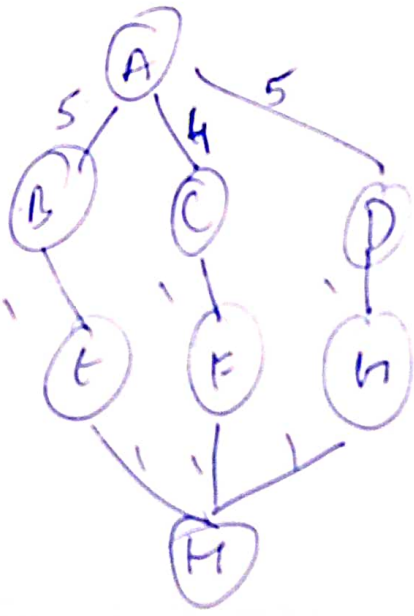
Solution : Make a Big jump.

3) Ridges : Special form of local maximum. It has an area which is higher than its surrounding areas, but itself has a slope and cannot be reached in single move

Solution : Move in different direction.



- Ridge



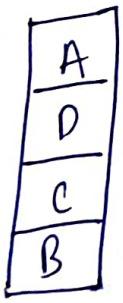
Block World problem using Hill climbing

=) What is Blocks World?

The World comprises of

- A Flat surface such as Tabletop.
- An Adequate set of identical blocks which are identified by letters
- The Blocks can be stacked one on one to form towers of Apparently unlimited height.
- The stacking is acheived using a robot Arm which has fundamental operations and states.
- The Robot can hold one block at a time and only 1 Block can be moved at a time.

Initial State

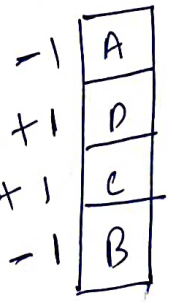


Goal State

› Local Heuristic function

- +1 for each Block that is resting on the thing it is supposed to be resting on.
- -1 for each Block i.e resting on a wrong thing.

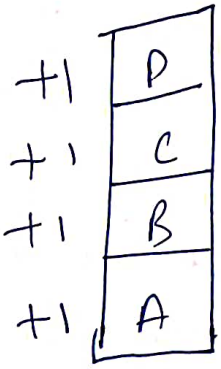
Initial State



Initial State
h = 0

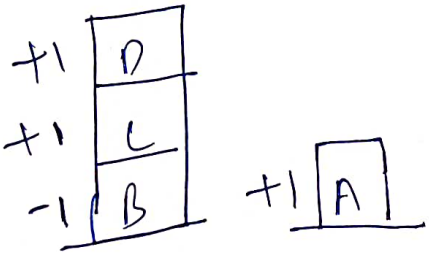
→ After Applying simple hill climbing Algo. then the initial state will become current state.

Goal State



Goal State
h = +4.

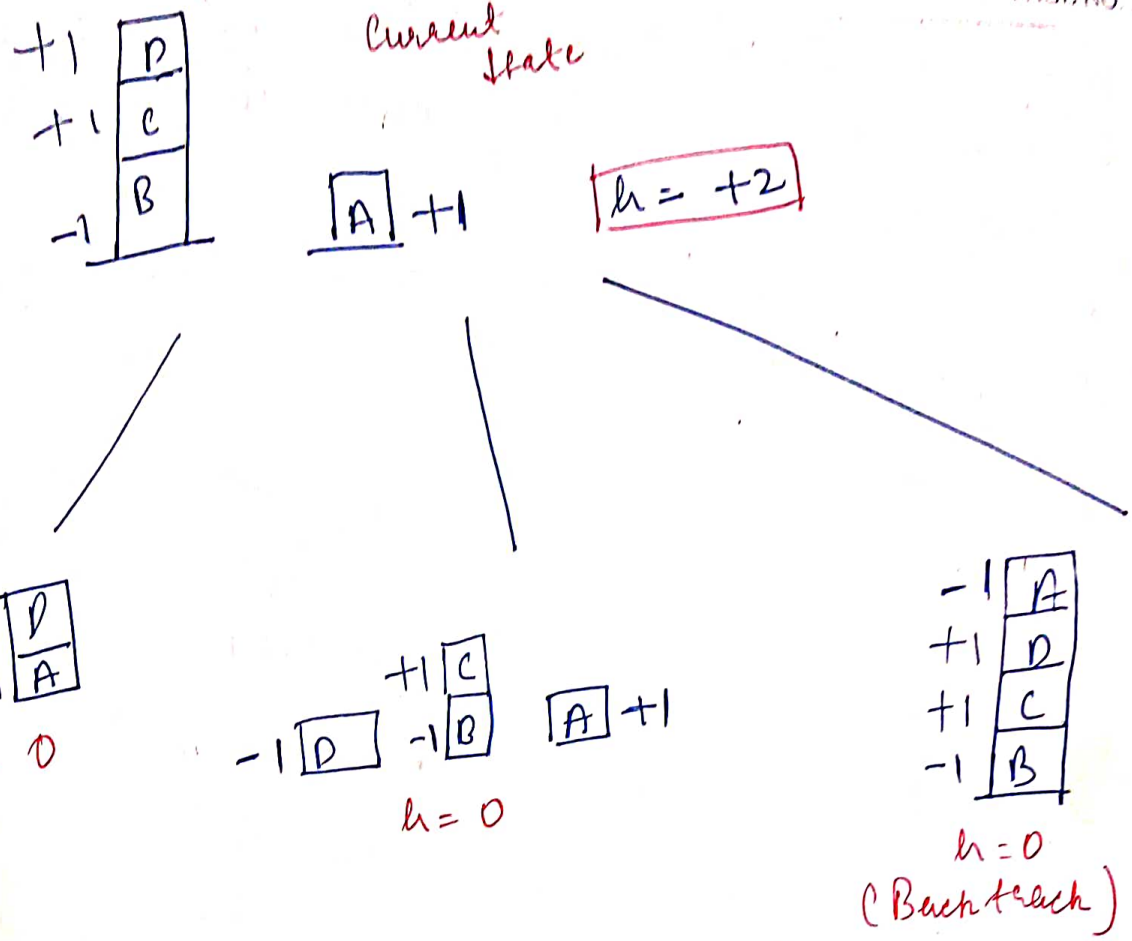
New State



h = +2
→ Better

→ Then we will make it as current state

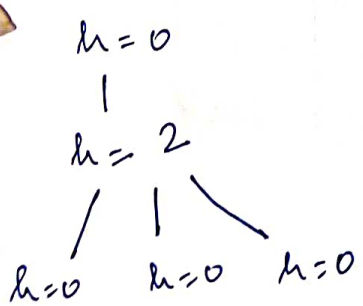
New state is better than current state or Not?

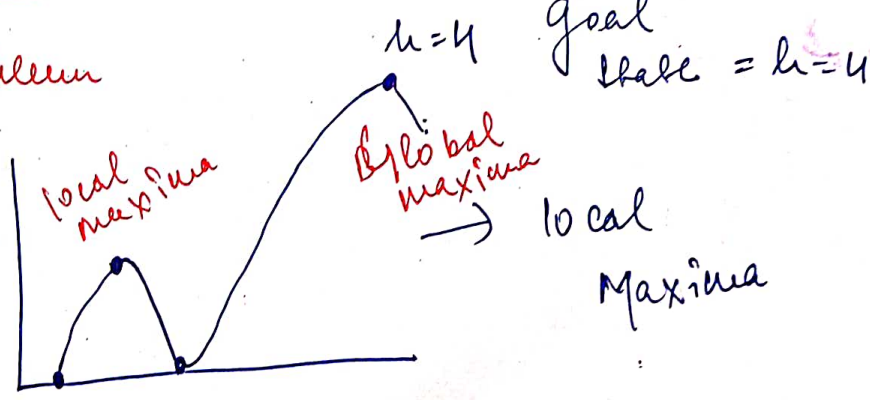


Which one is Better choice?

→ Trapped

Overall Problem



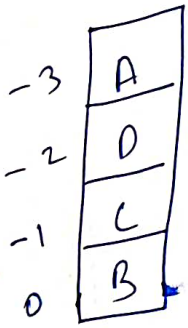


So now we need to see a New path / Backtrack.

Note: while designing Heuristic fn it should not consider local parameter but it should consider Global parameter.

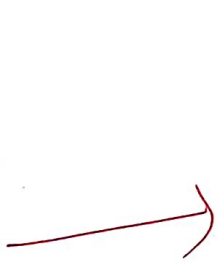
Global Heuristic function:

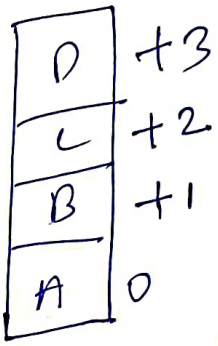
- For each block that has the correct support structure: +1 to every block in the support structure.
- For each block that has a wrong support structure: -1



h = -6

Initial state

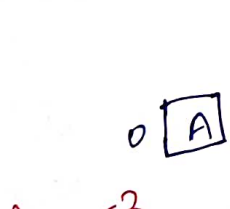




h = +6

Goal state

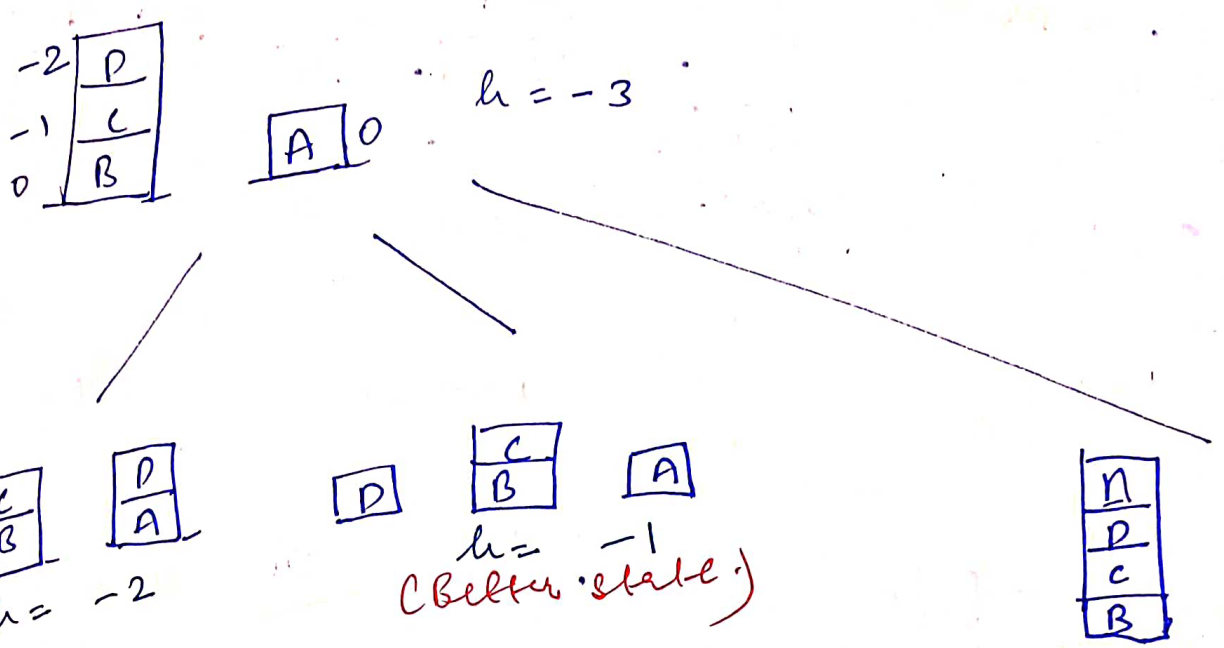
This will become as current state

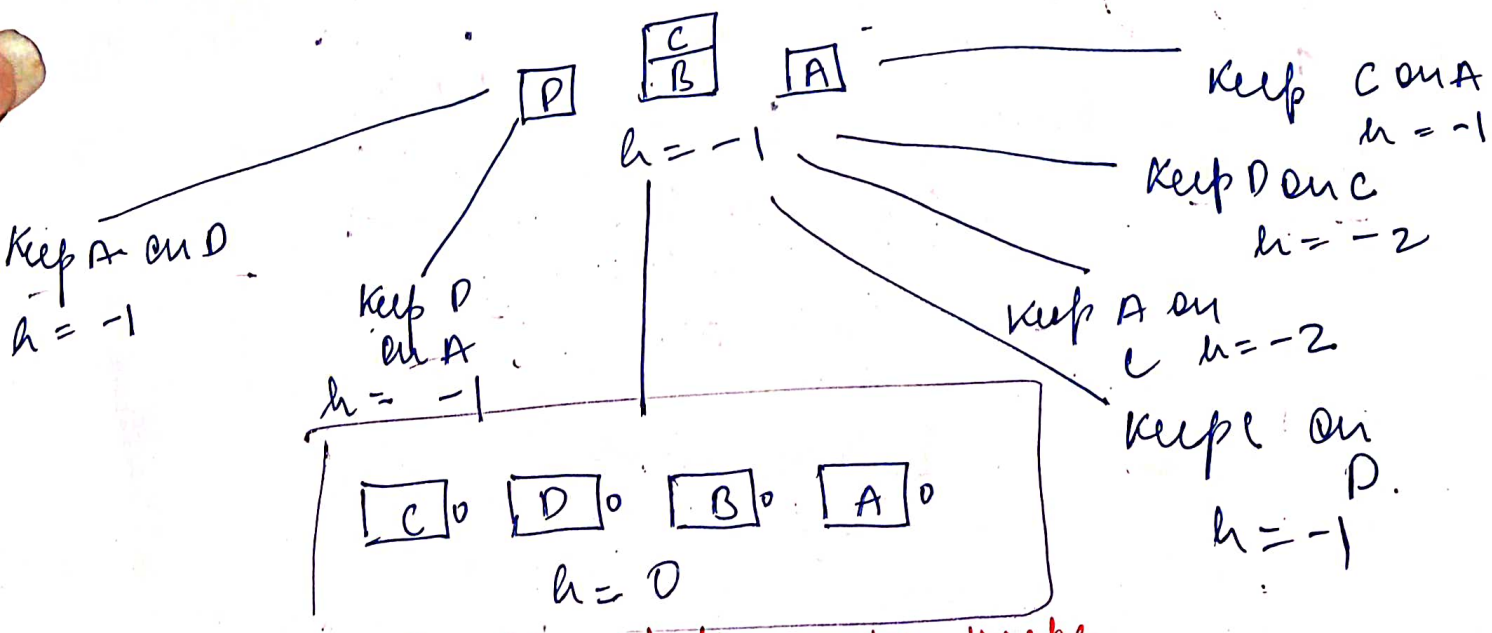


h = -3

→ New State is Better?
C then make this as current state.

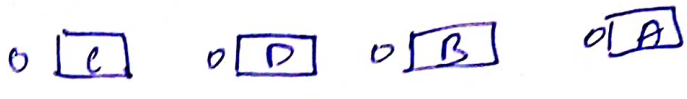
Current State





(Better state, so make this as current state

Current State

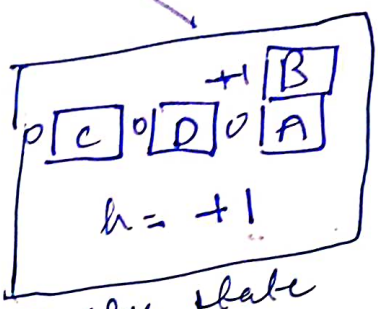


$h = 0$

Keep C on D or B on A ($h = -1$)

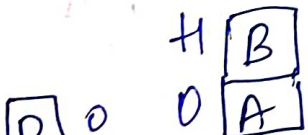
Keep D on C or B on A ($h = -1$)

Keep B on C or D ($h = -1$)



Better State

Keep A on B or D or C ($h = -1$)



Keep B on Table ($h = 0$)

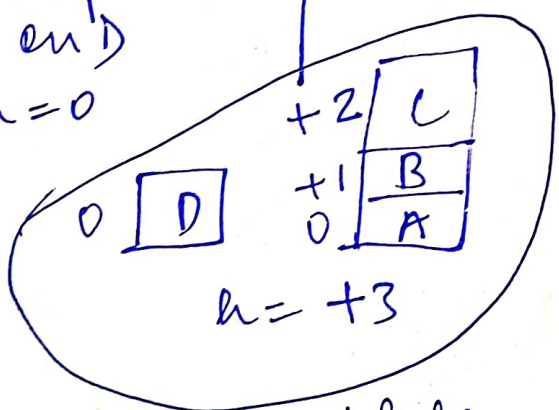
Keep D on C ($h = 0$)

Keep D on B ($h = -1$)

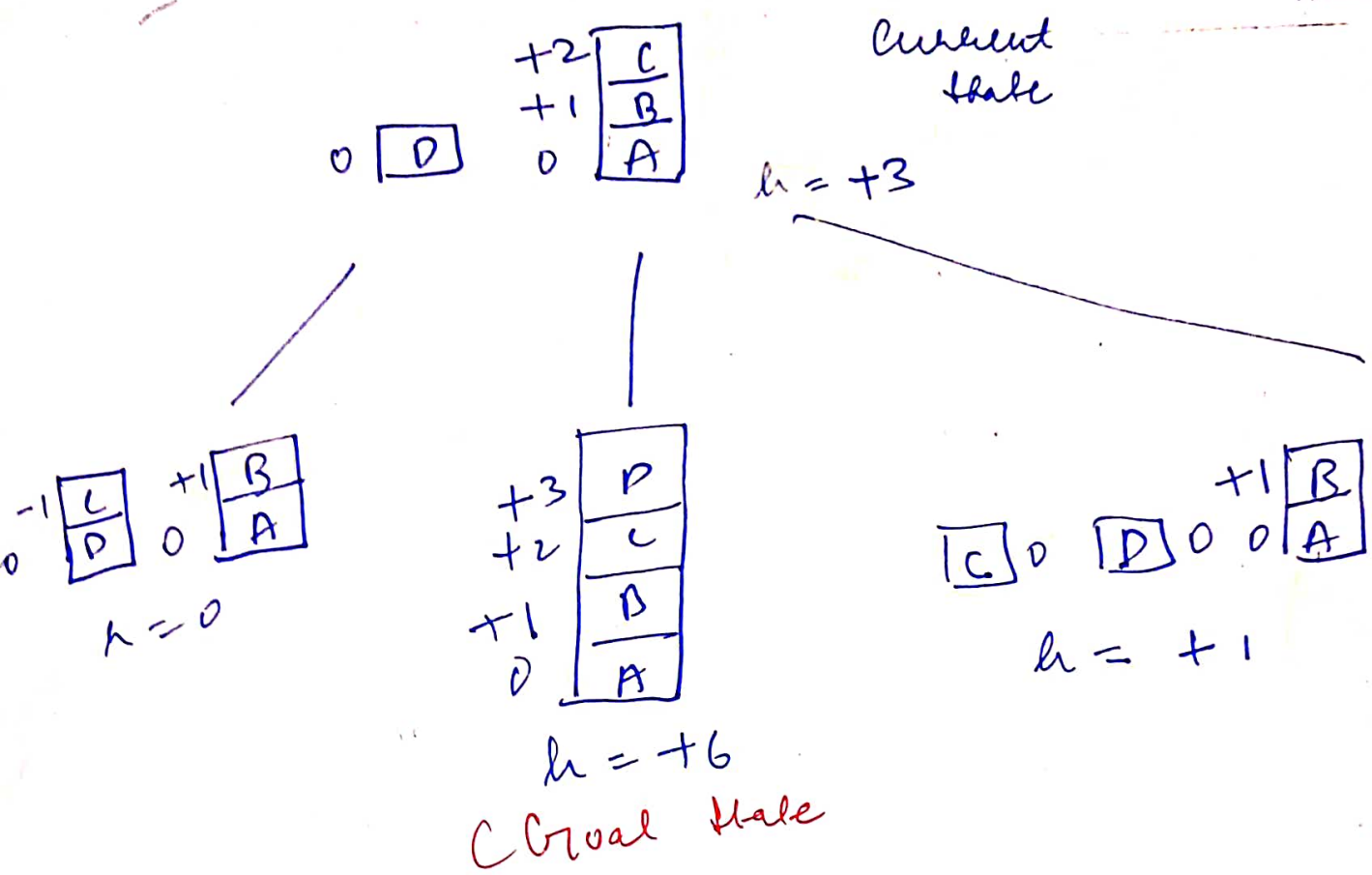
Keep C on D ($h = 0$)

Keep B on D ($h = -1$)

Keep B on C ($h = -1$)



Better State


Current state
h = +3
+2 C
+1 B
0 A
0 D
-1 C
0 D
+1 B
0 A
h = 0
+3 D
+2 C
+1 B
0 A
h = +6
Goal state
+1 B
C 0 D 0 0 A
h = +1

PAGE NO. .....

1) Best - First Search Algorithm (Greedy Search)

- Greedy Best first search Algorithm selects the path which appears best at moment.
- It is the combination of DFS and BFS.
- It uses Heuristic f^n and search.
- With the help of Best first search, at each step, we can choose the most Promising node.
- In the Best First search Algorithm, we expand the node which is closest to goal node and the closest cost is estimated by Heuristic f^n i.e.,

$$f(n) = g(n).$$

- Or we can also say that it uses Evaluation f^n to decide which Adjacent Node is most promising and Then Explore.
- lies in the Category of Heuristic or Informed search.
- Priority Queue is used to store cost of Nodes.

Algorithm

Priority queue PQ — ordered order containing initial states

loop

if PQ = Empty Return FAIL

else

NODE ← Remove_First (PQ)

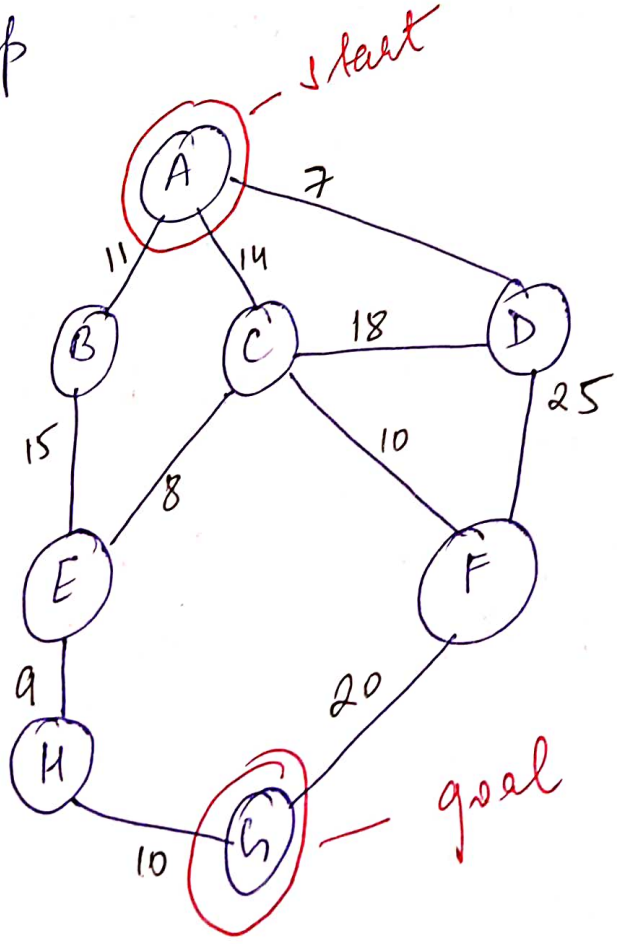
if NODE = GOAL

Return Path from Initial to NODE

else

Generate all Successors of NODE and insert newly generated NODE into PQ according to cost Value

End loop



Straight line Distance

A → G = 40
B → G = 32
C → G = 25
D → G = 35
E → G = 19
F → G = 17
G → G = 0
H → G = 10

open

[A]

[C, B, D] // here we will see straight line distance and put it in sorted order.

[B, D]

[F, E, B, D]

[G, E, B, D]

[E, B, D]

closed

[ ]

[A]

[A, C]

[A, C]

[A, C, F]

[A, C, F, G]

Path = A → C → F → G = (44)

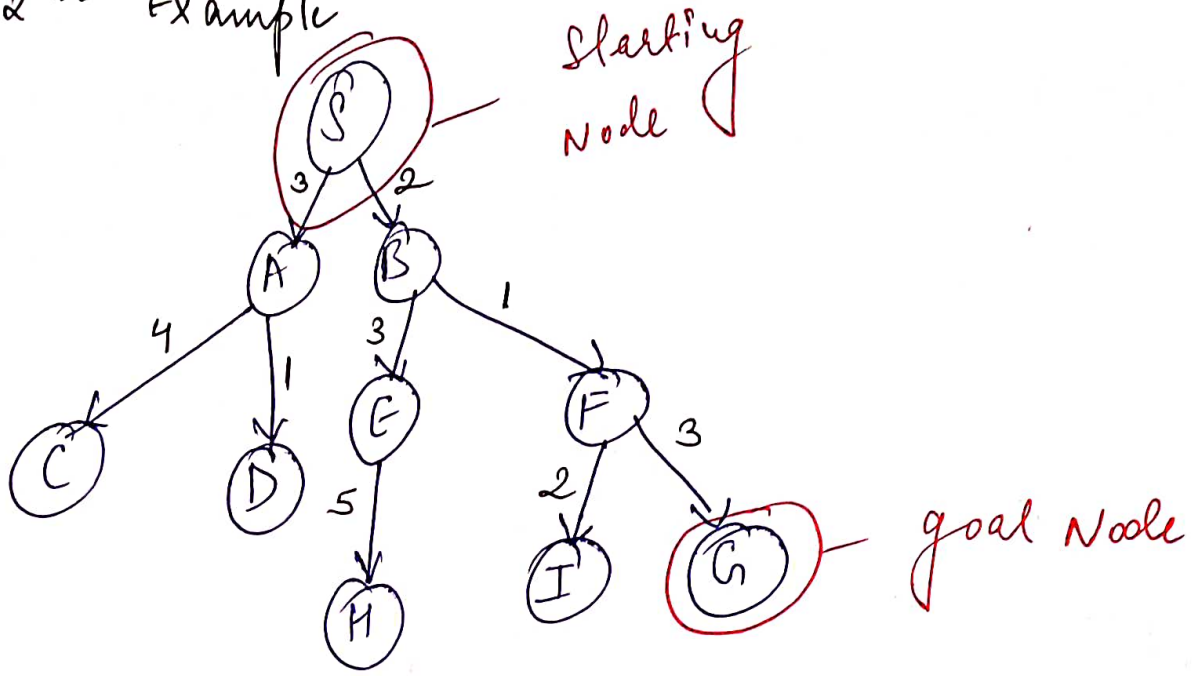
Space complexity = $O(b^d)$

Time complexity = $O(b^d)$

b: branching factor

d: depth.

2nd Example



Node	H(n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Open	closed
[S]	[ ]
[B, A]	[S]
[F, E, A]	[S, B]
[G, I, E, A]	[S, B, F]
[I, E, A]	[S, B, F, G]

Path → S → B → F → G.

Best First Search Algorithm

1) Place the starting Node into the OPEN List.
2) If the Open list is Empty, stop and Return Failure.
3) Remove the Node n, from the open list which has the lowest value of $f(n)$ and places it in the CLOSED list.
4) Expand the Node n, and generate the successors of Node n.
5) Check each successor of Node n, and find whether any Node is a Goal Node or not. If any successor Node is a Goal Node, then Return Success and terminate the search, else proceed to step 6.
6) For each successor Node, Algorithm checks for Evaluation f^n and then check if a Node has been in either OPEN or CLOSED list. If the No has Not been in both list, then add it to open list.
7) Return to step 2.

Advantages:

- Best First search can switch b/w BFS and DFS by gaining the Advantages of both the Algorithms
- More Efficient than BFS and DFS.

Disadvantages

- Stuck in loop as DFS.
- Not Optimal.

PAGE NO. .....

Unit - 1 : Topic : A* Search Algorithm

- Uses Heuristic function $H(n)$ and cost to reach the node 'n' from start state $g(n)$
- Finds shortest path through search space
- Fast and optimal Result.

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

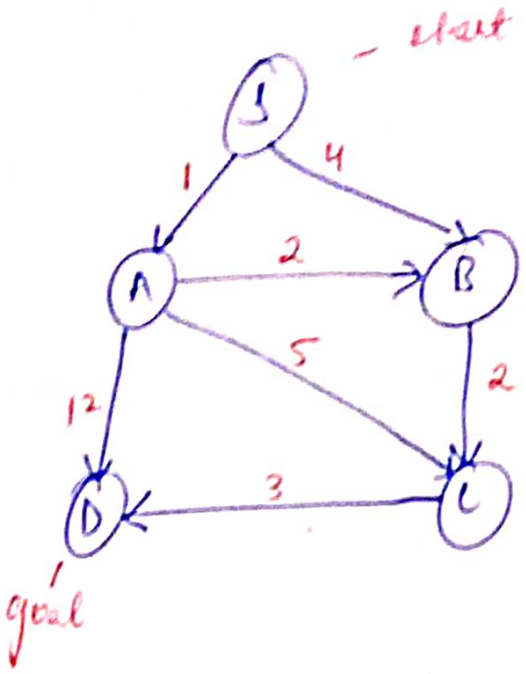
- It is a form of Best First Search.

$f(n)$ → Fitness No: Estimated cost of the cheapest soln.

$g(n)$ → cost to reach node 'n' from start state

$h(n)$ → cost to reach from node n to goal node.

Note: At each point in the search space, only those Node is Expanded which have the lowest Value of $f(n)$, and the Algorithm



Heuristic → h(n)
S → 7
B → 2
C → 1
A → 6
D → 0

S → A = 1 + 6 = 7

S → B = 4 + 2 = 6 ← cost

S → B → C = (4 + 2) + 1 = 7 ← closed

S → B → C → D = (4 + 2 + 3) + 0 = 9

↓

We will keep this to compare it with other paths.

S → A → B = (1 + 2) + 2 = 5

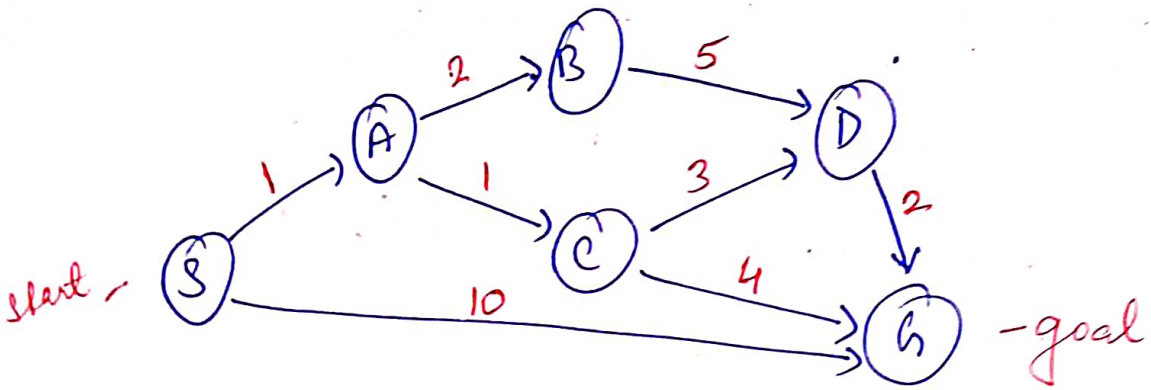
S → A → C = (1 + 5) + 1 = 7 ← closed

S → A → D = (1 + 12) + 0 = 13 → ~~2~~ 2

S → A → B → C = (1 + 2 + 2) + 1 = 6

closed # 2 — S → A → B → C → D = (1 + 2 + 2 + 3) + 0 = 8

closed # 4 — S → A → C → D = (1 + 5 + 3) + 0 = 9



State	H(n)
S	5
A	3
B	4
C	2
D	6
G	0

$S \to A = 1+3 = 4$

$S \to G = 10+0 = 10$ — ✗1

$S \to A \to B = (1+2)+4 = 3+4 = 7$

$S \to A \to C = (1+1)+2 = 2+2 = 4$

$S \to A \to C \to D = (1+1+3)+6 = 5+6 = 11$

✗2 $S \to A \to C \to G = (1+1+4)+0 = 6$

$S \to A \to B \to D = (1 + 2 + 5) + 6 = 14$

Now here we exploring this path are :

$S \to A \to C \to D \to = (1 + 1 + 3 + 6 = 11$

$S \to A \to C \to D \to G = (1 + 1 + 3 + 2) + 0 = 7.$ A, B

$S \to A \to B \to D \to G = (1 + 2 + 5 + 2) + 0$
$= 10$

We will get as

$S \to A \to C \to G = (1 + 3 + 4) + 0 = 6.$

Algorithm of A * Search

Step 1 : Place the starting node in OPEN LIST.

Step 2 : Check if the OPEN list is empty or not, if the list is empty then return failure and stop.

Step 3 : Select the node from the OPEN list which has the smallest value of $(g + h)$. If the node n is the goal node then return Success and stop, otherwise.

Step 4 : Expand Node n and generate all of its Successors, and put n into the closed list. For each Successor n, check whether n' is already

in the OPEN or CLOSED list. If not then compute valuation $f'n$ for n' and then place into Open list.

Step 5: Else if node n is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest $g(n')$ value.

Step 6: Return to Step 2.

Advantages:

- Best searching Algorithm
- Optimal and complete
- Solving Complex problems.

Disadvantages:

- Does not always produce the shortest path as it mostly based on Heuristics and Approximation.
- Some complexity issues.
- Requires memory

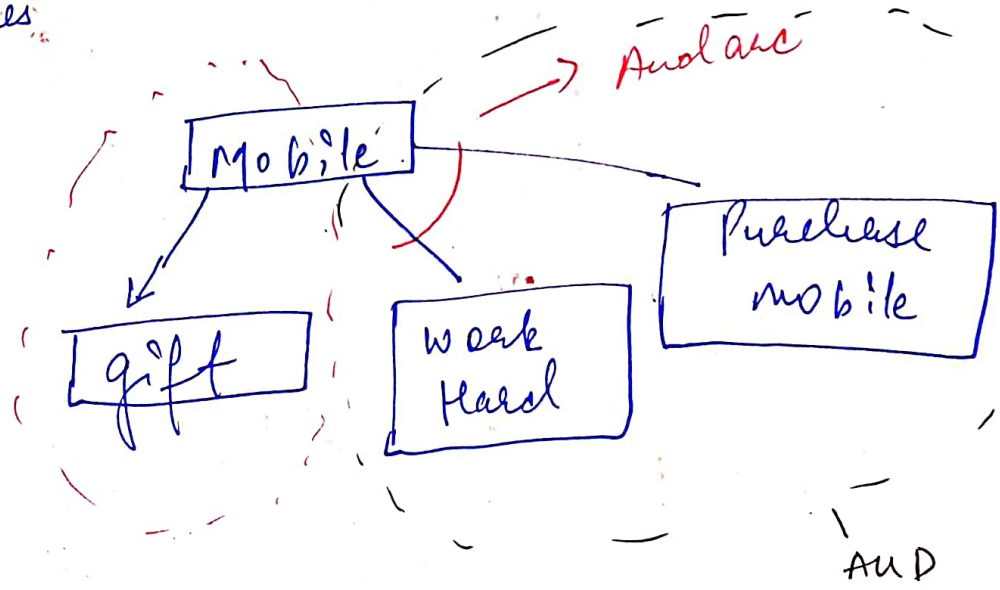
AO * Algorithm / AND-OR Graphs

- And-Or graphs is useful for Representing the Problem Solution that can be solved by decomposing them into Smaller set of problems, all of which must be solved.

ANDARCS : May point to 'n' no. of Successor Nodes.

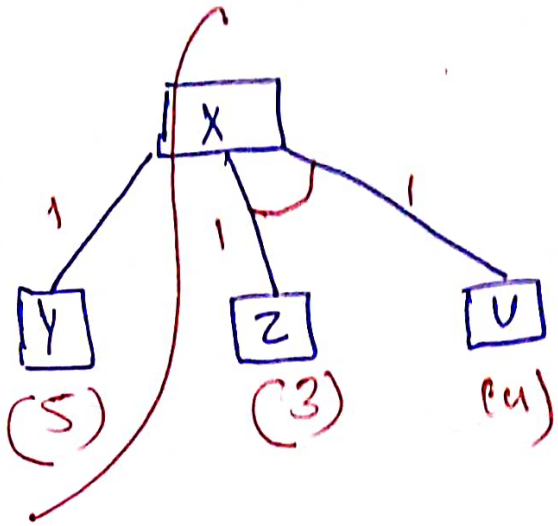
- It is an Algo like Best-fit search can be used that has the Ability to handle AND Arcs.

Examples :



* FUTILITY : Should be chosen to correspond to a threshold value.

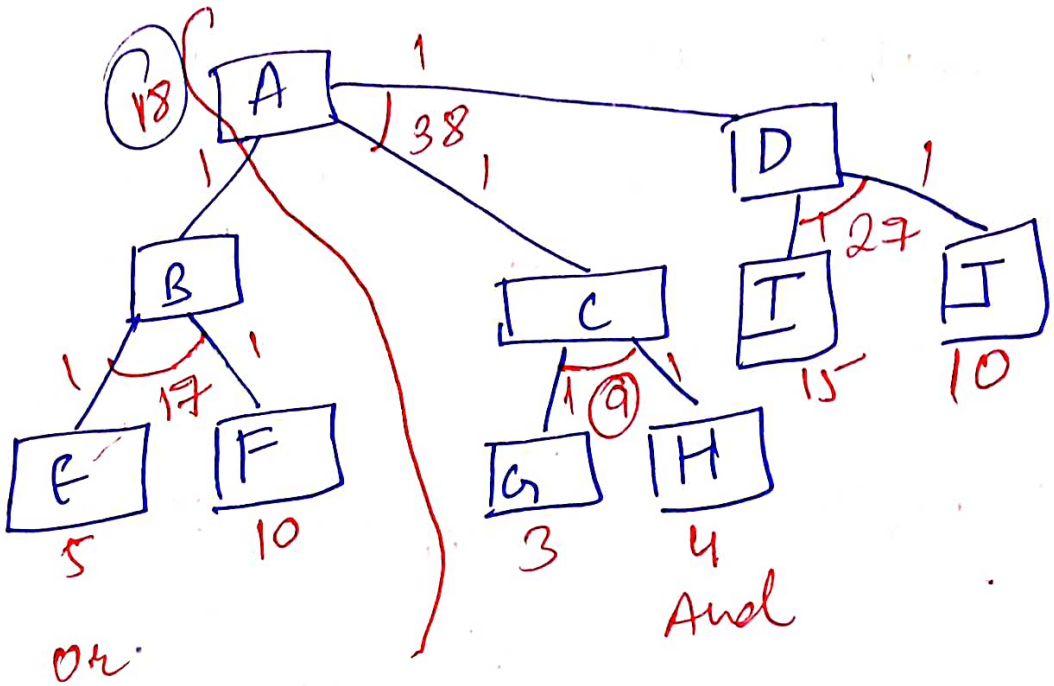
→ If Est. cost > Futility ⟹ Stop Search



(OR) $x \rightarrow Y \quad 1 + 5 = 6$

(AND) $x \rightarrow z \rightarrow x \rightarrow U = 1 + 3 + 1 + 4 = 9$

Example 2



$A \rightarrow B \rightarrow E$ and F. $217 + 1$ 18

- It is useful for Representing the Solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must be solved.
- The Decomposition, or reduction generates arcs that we call AND Arcs. One AND arc may point to any no. of successor nodes all of which must be solved in order for the Arc to point to a Soln.
- Just as in an OR graph, several arcs may emerge from a single node, indicating a variety of ways in which the Original problem might be solved. That is why the structure is called Not simply an AND Graph but rather an AND-OR Graph.

Constraint Satisfaction Problem (CSP)

- It is a search procedure that operates in a space of constraint sets.
- Constraint satisfaction Problems in AI have goal of discovering some problem state that satisfies a given set of constraints

Process

i) Constraints are discovered and propagated throughout the system.

ii) If still, time is No terminations, search Begins

↳ A guess is made about something and added as a new constraint.

CSP → V [Variable list { V1, V2, V3 ... Vn }

→ D [Domain { D1, D2 .... Dn }

↳ one domain for each Variable.

→ C (constraints)

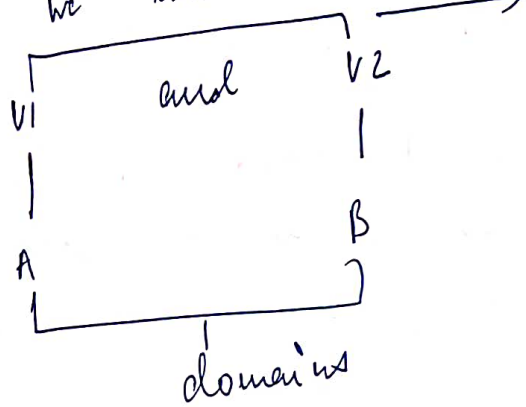
→ Specify allowable combination of values

$C_i = (\text{Scope}, \text{Relation})$

Scope → set of variables that participate in constraint

Relation → defines values that a variable can take.

- Suppose we have 2 variables → Variables.



Constraint

1) Values of V1 and V2 can be same.

$C_1 = \{(V_1, V_2), V_1 \neq V_2\}$

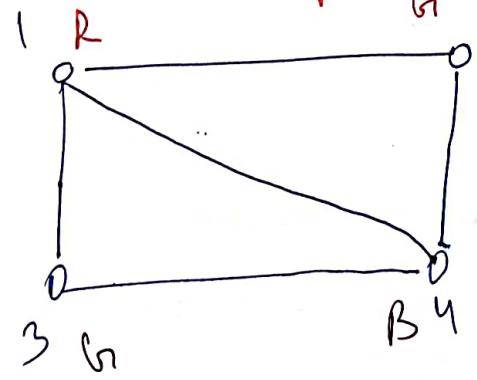
(Scope) , (Relation)

- To solve constraint Satisfaction Problem, we use Intelligent Backtracking Method,

only Backtrack where conflict occurs

Example of Constraint Satisfaction Problem

Node Colouring



Variables = {1, 2, 3, 4}

D = {Red, Green, Blue}

C = {adjacent Nodes cannot have same color}

	1	2	3	4
Initial	R, G, B	R G B	R G B	R G B
1 = R	R	G, B	G, B	G, B
2 = G	R	G	G, B	B
3 = B	R	G	B	B (error)

This particular state is giving me an error

For this conflict we need to Backtrack to where conflict has occured

3 = G	R	G	G	B

Example 2 of constraint satisfaction problem

Crypt Arithmetic Problem

- It is an example of type of constraint satisfaction problem. (CSP)
- Constraints

→ Digits that can be assigned to a word/ alphabet (0-9)
Range

→ No 2 letters have same value

→ Sum of digits must be as shown in the problem.

→ There should be only 1 carry forward

Letter	Digit
T	2
O	1
G	8
U	0

```
  TO
+ GO
-----
 OUT
```

start from left most digit = (1)

(0-9)

9+8 = 17

(Here you will get maximum carry as 1)

- Also leftmost digit cannot be 0.

2	1	
8	1	
1	0	2

$2 + 8 = 10$ select.

$2 + 9 = 11$

$2 + 8 = 10$

$2 + 8 = 10$ this can't be taken as it value will become 1 and we have already assigned 1 to O.

- Second example of constraint satisfaction problem.

	C3(0)	C2(1)	C1(1)	
	S 9	E 5	N 6	D 7
+	M 1	O 0	R 8	E 5
M O N E Y	M 1	O 0	N 6	E 5	Y 2

- Now we will check $S + M$, if $m = 1$

$S + M \geq 10$ $\quad S = 9$

- Now $E + 0 = N$
  $E + 0 = N$] if $c2 = 0$ X $\rightarrow c2 = 1$

- let E = 5

$$E + O + C_2 = N$$

$$5 + 0 + 1 = 6$$

- let $N + R = E$

$6 + R = 5$

$6 + 9 = 15$ carry.

let $R = 9$

] Not possible as we already given the value as 9 to S.

- ~~Trial~~ So for this we can do as

$$6 + 8 + 1 =$$

(C1)

- Now $D + E = Y$

$7 + 5 = 12$ 1 goes to carry

$y = 2$.

P — 5 X E

\ 6 X N

/ | \ 7 ✓

9 X 8 X

S R

3) Example of Cryptarithmetic Problem.

```
  B A S E
+ B A L L
---------
G A M E S
```

(1)

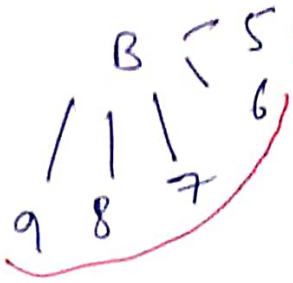
	C3	C2	C1	
	B 7	A 4	S 8	E 3
+	B 7	A 4	L 5	L 5
G 1	A 4	M 9	E 3	S 8

→ • E + L = S — (4) No carry
• E + L = S + 10 carry or S − E = L
S − E = 5
↓
(5, 0) ✗ L
(6, 1) ✗
(7, 2) ✗
(8, 3)
(9, 4)

E = S − L + 10

→ S + L = E
S + L = S − L + 10 or
2L = 10
L = 5

B + B = A + Carry



let S, E = 7, 2 (discards)

eg

B = 6

6 + 6 = 12

- let S, F = 8, 3

B = 7

Q. Try Cross + Roads = DANGER