Artificial Intelligence.

Ch 2 - complete ✓
Ch 3 - 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8 *
Ch 4 - 4.1, 4.2, 4.3, 4.4, 4.7
Ch 5 - 5.2, 5.3, 5.4, 5.5, 5.7, 5.8, 5.9, 5.10, 5.11
Ch 6 - 6.3

→ Chapter 2.

2.1 Generate & Test
2.2 simple search
2.3 Depth first search (DFS)
2.4 Breadth first search (BFS)
2.5 comparison of BFS vs DFS
2.6 Quality of soln
2.7 DB DFS = depth bound DFS
2.8 DFID = depth first Iterative deepening

Intro
- model the problem solving process as → traversing a state space.

STATE ≠ description of the world in which problem operates.
Set of choices avebl to us ⇒ define the space in which this process operates.

2.1. High level search algo
1. generate a candidate from the state space
2. To test whether the candidate is the goal state.

Chapter 3.

Heuristic = guide the search.
pick the node most likely to be on path.

To make NP → P
we do heuristic search
i.e. informed.

solving problem quickly ← what, why, how.

why → To reduce time. → only explore nodes w the best heuristic value.

blind search – NP problem.

how → some ways :-
- Euclidean distance
- Manhatten distance (8 puzzle)
- Rule of thumb basically.

→ Always gives the good soln, however not optimum.

→ 8 puzzle with heuristic (Informed search)
↓
use no. of misplaced tiles as heuristic function
- BFS (best first)
- Greedy
- A*
- Hill climbing

→ Generate & Test (Heuristic Technique, DFS w BACKtracking)
1. generate a possible soln
2. Test to see if this is actual soln
3. If a soln is found, quit. Otherwise go to step 1.

Properties of good generators :-
1. Complete
2. Non Redundant
3. Informed

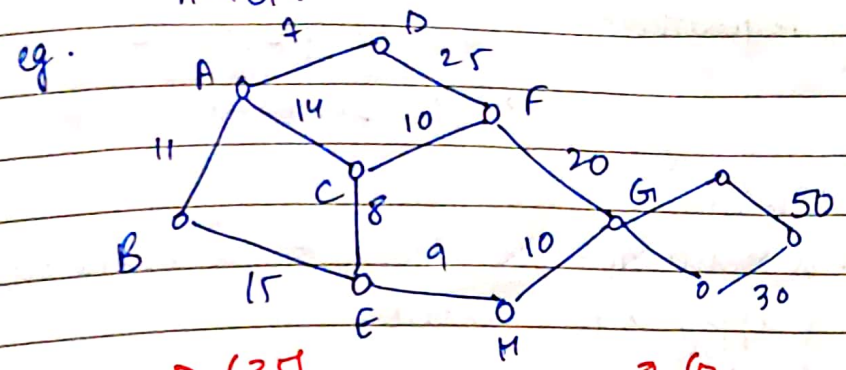
→ Best First Search Algorithm. (Informed, Heuristic)

Algorithm :-
→ Let OPEN be a priority queue containing initial state
(↳ on heuristic value)

LOOP
if OPEN is empty, return failure
NODE ← Remove. first (OPEN)
if Node is a goal
then return the Path from Initial to node
else generate all successors of node
put newly generated Node into OPEN
according to their f values
END LOOP

Euclidian formula

eg. A → G.



straight line dist

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

A → D (35) → F (17) → G
A → C (25) → F (17) → G
A → B (32) → E (19)

again arranged acc to priority & take least f values

open = ~~A~~ ~~C~~ B D F E

acc to f values

A → C → F → G

cost = 44

this is why H gives ← ACE H G would have given 41 cost.
good soln, not optimal.

→ Beam search Algorithm.

- Take care of space algorithm (constant)
- Beam width (β) is given

considering the same eg as above.

↓

if beam width is 2 then it says to only keep 2 nodes in OPEN list, why keep all

~~A~~ ~~C~~ B D F E

~~A~~ C B (don't keep D) → ~~B~~ F E

remove CB.

→ Improves space complexity → you're pruning nodes.
Time also improves because priority queue takes O(nlogn)

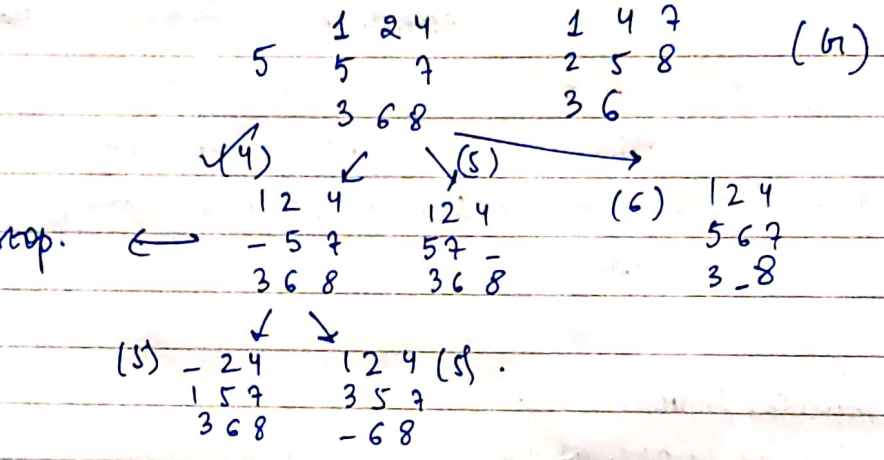
Beam Search NOT COMPLETE.

if β = 1 then it's hill climbing method

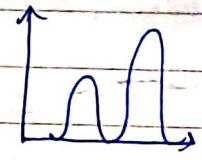
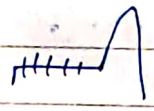
→ Beam width 1.

→ Hill climbing Algorithm. → local search algo
→ greedy approach
→ NO Backtrack

1. Evaluate the initial state
2. loop until a solution is found or there are no operators left
   - Select & apply a new operator
   - Evaluate the new state:
     goal then quit
     → if better than current state then it is new current state

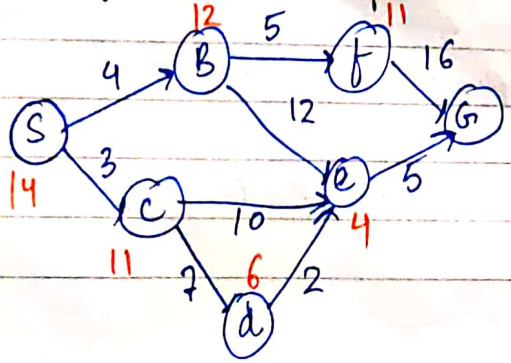


(*) Problems in hill climbing
1) local Maximum
2) Plateau (Flat maximum)
if all moves give some value, it stops.
3) Ridge

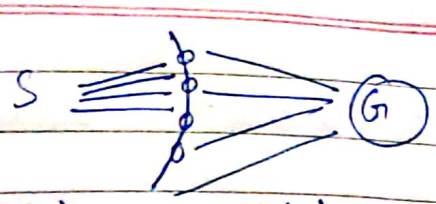



doesn't change dir^n (special case of local max)

→ A* algorithm → Informed searching



$f(N) = g(N) + h(N)$
↓ Actual cost from start node to n
↓ Estimation cost from n to goal node



$g(n)$ $h(n)$ → estimated. (in range).

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

S → B or S → C

$4 + 12 = 16$ | $3 + 11 = 14$.

SC → E $3 + 10 + 4 = 17$

SC → D $3 + 7 + 6 = 16$

Now they're equal you can explore both.

$SB + f = 4 + 5 + 11 = 20$

$SB + e = 4 + 12 + 4 = 20$

Sed → e $3 + 7 + 2 = 16$

Now this is chosen again.

$12 + 5 + 0 = 17$

Time complexity = $O(V+E) = O(b^d)$.

SC - $O(b^d)$

→ Admissible

estimated $h(n) \leq h^*(n)$ → underestimation

$h(n) \geq h^*(n)$ → overestimation

S — n — G

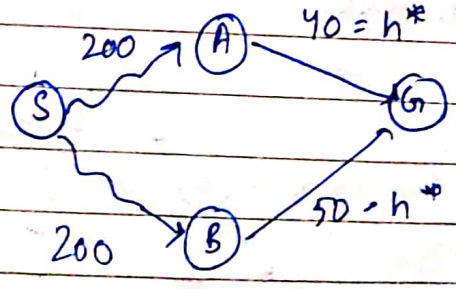
$g(n)$ $h(n)$.

your estimation = 20K

actual = 30K

↓

you'll go to next step.



Case 1 = overestimation

estimated ← $h(A) = 80$, $h(B) = 70$ } $> h^*$ → actual.

$f(A) = 200 + 80 = 280$

$f(B) = 200 + 70 = 290$.

better than both

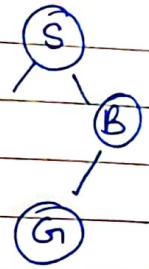
$f(G) = g(G) + h(G)$

$= 250 + 50 = \boxed{250}$

so this stops.

Case 2 = underestimation

$h(A) = 30$, $h(B) = 20$ } $< h^*$



$f(A) = g(A) + h(A) = 200 + 30 = 230$

$f(B) = g(B) + h(B) = 200 + 20 = 220$

$f(g) = g(G) + h(G) = 250 + 0 = 250$

won't stop here

$f(G) = g(G) + h(G) = 240$

NOW it stops.

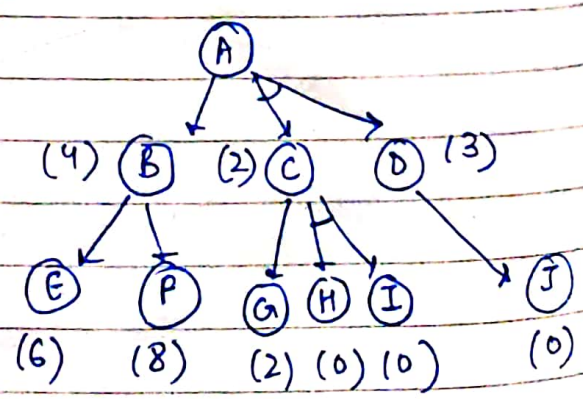
→ AO* (And / OR) → problem decomposition (breakdown into smaller pieces)

Want to pass in exam

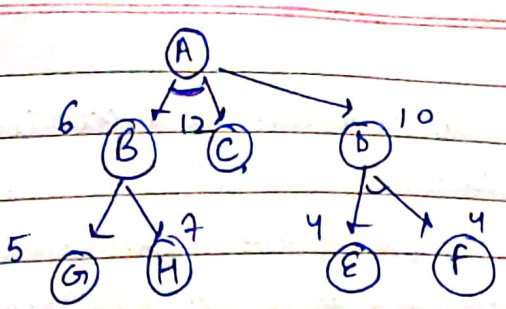
denotes AND

Do cheating — Hard work — Pass it

AO* does not explore all the solution paths once it got a solution



(*) question.



* ⇒ admissable

find new heuristic value at every stage & update till root

Unit 3 -

3.4 - local maxima

3.5 - solution space search

3.6 - Variable nbd descent

3.8 - Tabu search.

3.4

Heuristic fn defining a terrain over the search space. with each state having a heuristic value.

strategy → steepest gradient ascent or descent if heuristic fn thinks lower values are better

performance depends on the terrain being defined.

Smooth terrain → unhindered.

undulating " → can get stuck on local optimum

3.5

problems → constructive search problems.

↓

we incrementally build the soln.

we extend given partial soln's.

(*) read again

can be both

global - best first search (entire search space at hand)

local - hill climbing (confined to choosen path)

→ Perturbation search

↳ each node in search space is a candidate soln.

may be local or global.

hill climbing → looks for local improvements & stops when it cannot find a better candidate.

local perturbation search methods AKA neighbourhood search algos.

3.6 variable neighbourhood descent

↳ extension of hill climbing algo.

* different neighbourhood fns can be defined

can get stuck on local optimum

- more sparse – less costly · quick movement
- more dense – costly.

VDN tries to get best of both the worlds.

↓

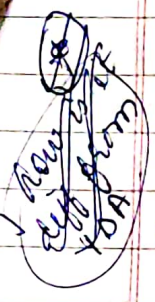
starts searching with sparse nbd fn

↓

once it reaches an optimum → switches to a dense fn.

based on the logic that most of the movement would be done in earlier rounds.

↑ diversify search.



3.8 Tabu search

- follows heuristic as long as better choices are presented.
- when no better choice, instead of terminating local search, it gives in to exploratree tendency & continues searching.
- modifies the termination criterion, does not terminate on reaching a max, continues searching beyond until some other criteria is also met.
- to drive search away from maxima, keep a finite CLOSED list implemented by circular queue.

Hill climbing & Tabu search - work with bounded memories.

Chapter 4

4.1 - Iterated Hill climbing
4.2 - Simulated Annealing
4.3 - Genetic Algorithms.
4.4 - The TSP
4.7 - Ant colony optimization.

4.1

In hill climbing,
start node - no particular significance
↳ simply a candidate to begin with.
starting pt of steepest gradient ascent (or descent).
↓
terminates when gradient = 0.
if surface monotonic = global optimum
else, may be only a local optimum.

Iterated Hill Climbing, → does a series of searches from a set of randomly selected different starting points.
Hope that :- best value found by one of them will be global optimum.

- constant amt of space to run.
- diff runs on same problem → IHC will return diff solns.

4.2

↳ Hill climbing variation.
- Simulated Annealing allows downward steps.
(do kadam peeche like lambi chhalaang maarna).
↳ checks all neighbours.
→ easy to code for complex problems.
→ gives good solns → guarantees optimal soln.

→ slow process, can't tell if optimal soln is found.
↓
for this some other method rqd.

→ annealing schedule is maintained
→ move to worse state may be accepted.
→ maintain the best state so found.

(instead of random starting points, random moves are being allowed).

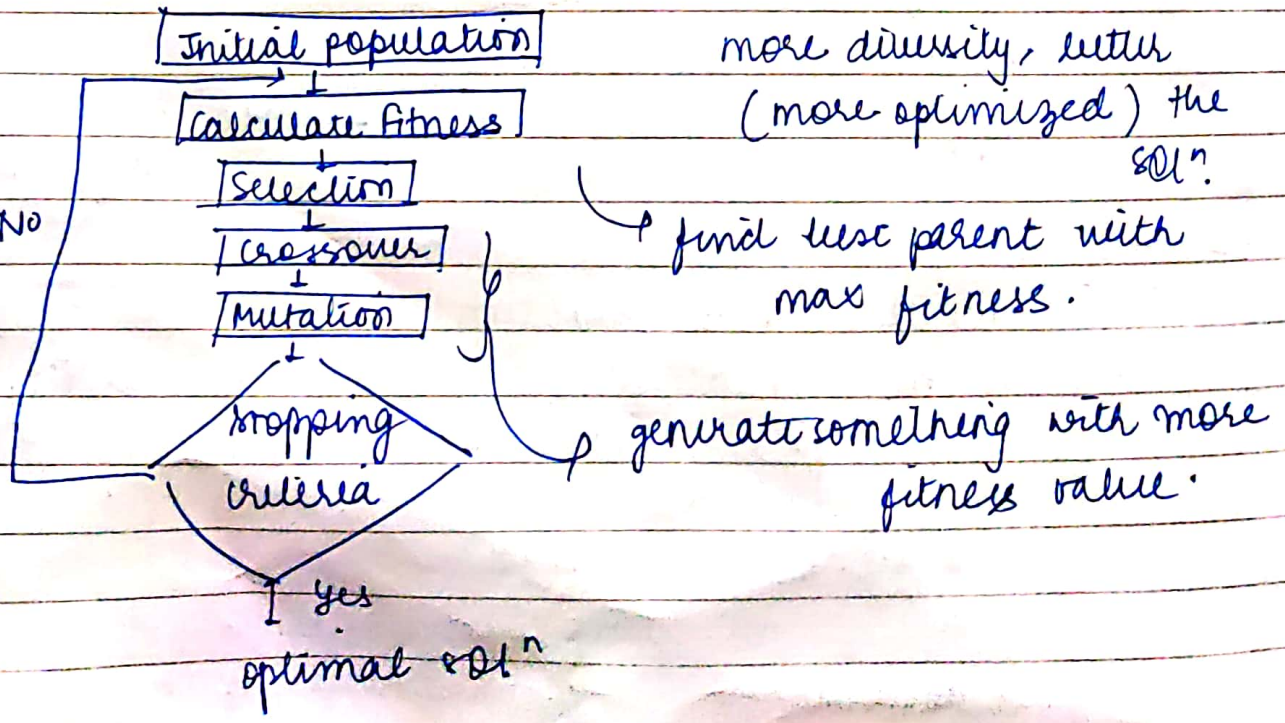
Pg 125 – A randomized algo . . . .

4.3 Genetic Algorithm.

4 best genes travel in evolution

→ abstraction of real biological Evolution
→ Solve complex problems (like NP hard)
→ focus on optimization
→ population of possible solns for a given problem.
→ from a group of individuals, the best will survive

Phenotype ⇄ genotype (encode / decode)

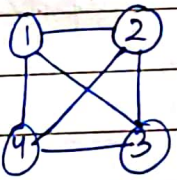


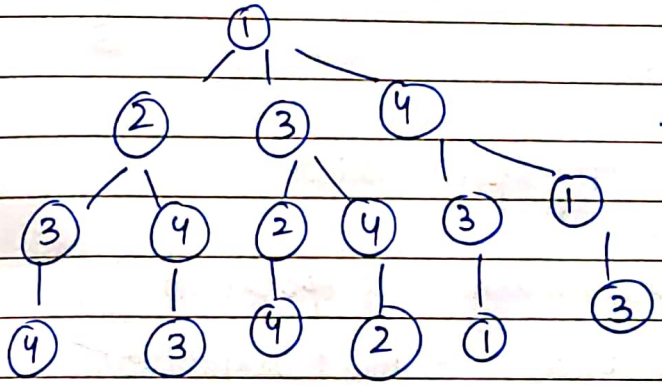
5.2 Branch & bound.

↳ branching is the process of generating subproblems.

- bounding means ignoring partial soln that cannot be better than the current best soln.
- → eliminate those parts of search space that don't contain better solution (pruning).
- → we basically extend the cheapest partial path.

4.4 Travelling Salesman Problem using B&B.





State Space Tree

for every node find cost, discard those that exceed upper bound.

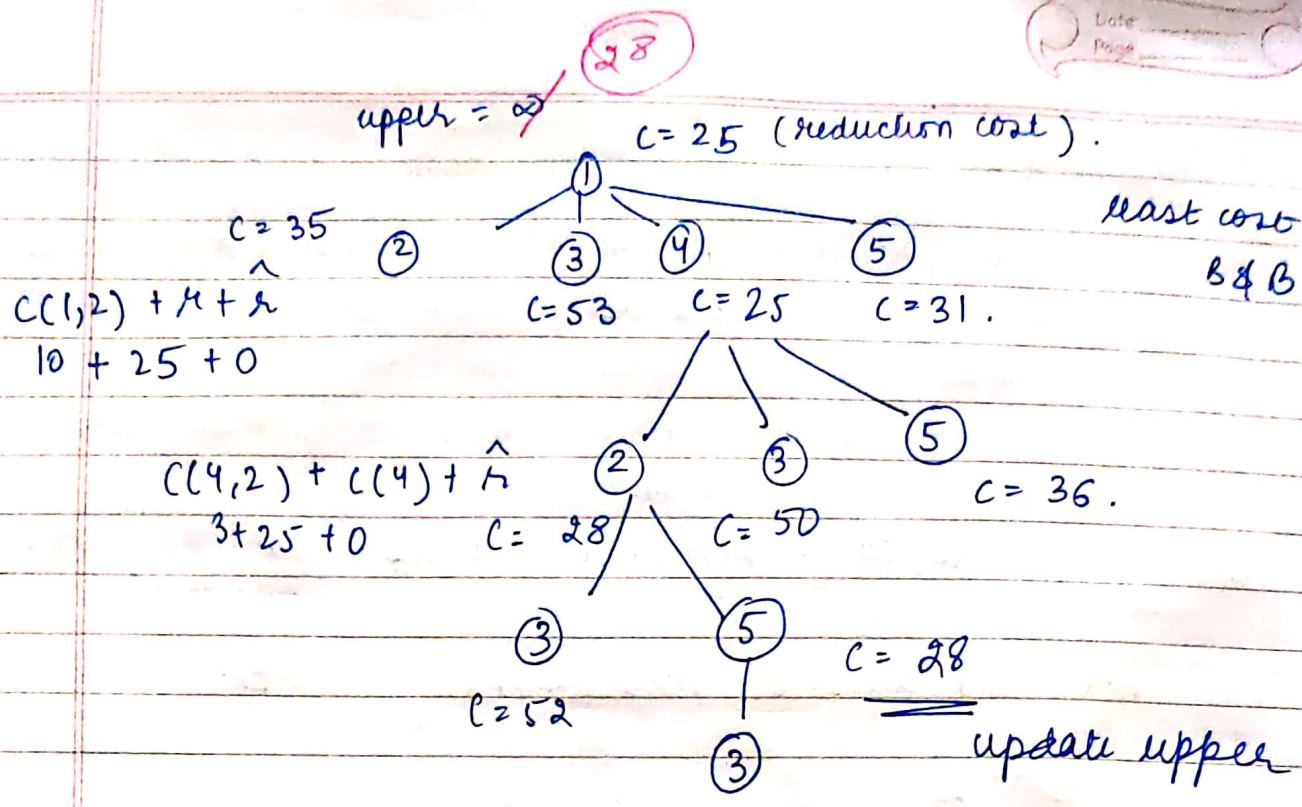
backtracking - for permutation problems.

	1	2	3	4	5	
1	∞	12	20	10	0	10
2	13	∞	16	11	0	2
3	0	5	∞	12	2	2
4	15	6	18	∞	0	3
5	11	0	0	12	∞	4
	1	0	3	0	0	21

⇒ 4 ⇒ 21+4 = 25

reduced cost = 25

↳ consider reduced matrix

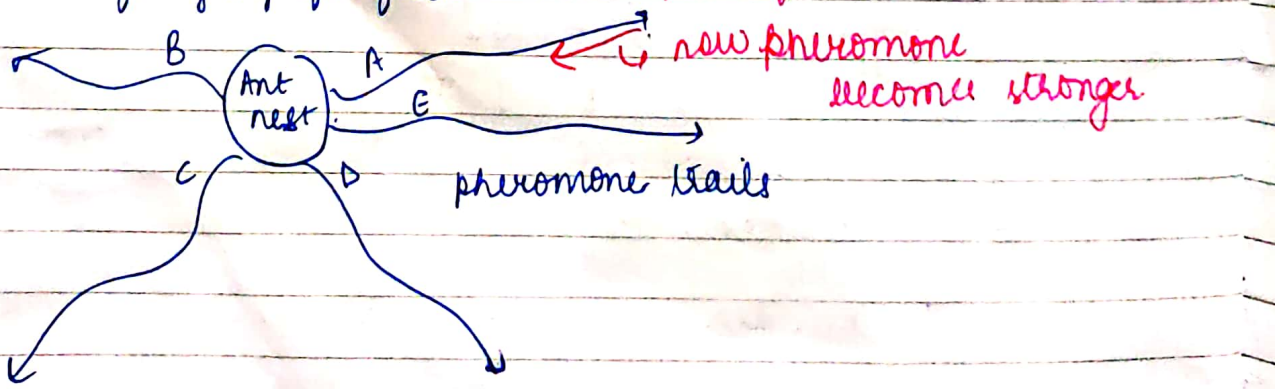


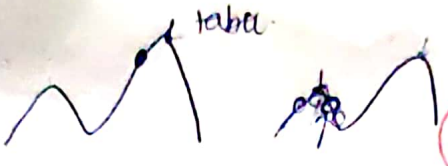
4.7 Ant Colony optimization

↓

Inspiration :- Co-operation

- Biosemiotics = bio + semiotic
  - ↳ prelinguistic meaning-making
- Ant Colonies.
  - ↳ complex social colonies
  - Hierarchy.
  - cooperation & div of labour, well developed comm sys.
  - ↓
  - allows ants to utilise their env better.
- Ants foraging for food





→ each ant constructs a solution using a stochastic greedy method using a combination of heuristic fn & pheromone trail following.

• Applicable to problems that can be reduced to graph search

deposit pheromone $\propto \frac{1}{\text{tour-length on edge } (u,v)}$

→ decides a tour & updates Pheromones.

TABU SEARCH.

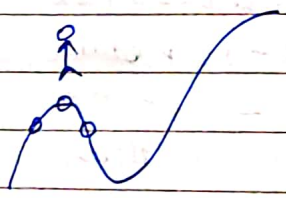
Hill climbing ⇒ EXPLOITATION of gradient

c = current
n = next.

if n = Best (MoveGen (c)) is better than c

n ← c

need to do EXPLORATION.



n ← Best (allowed (MoveGen(c))

↓

Till some termination criteria

(store best solution found)

Use circular CLOSED Queue

↳ keep track of next moves. eg SAT = if a bit is changed in last t moves disallow it.

↳ Tabu tenure.

eg. 9 bit SAT

0 0 0 0 0 0 0 0 0 t = 1

↓

0 0 0 4 0 0 0 0 0 t = 2

↓ ↓ |

3 4 |

↓ ↓ 4 . . .

2 3

↓ ↓

1 2

↓

ASPIRATION CRITERIA

↓

IF all allowed are bad & a tabu move leads to n which is better than best

THEN

ALLOW the (tabu) move.

FREQUENCY

F = | 18 7 6 7 2 11 14 18 | ← if we want to push algo towards nodes that have been visited less).

modify heuristic evaluation fn by :-

eval'(n) ← eval(n) − K F(Dn).

STOCHASTIC / RANDOMIZED methods.

Random Walk

randomly choose next neighbour

random − neighbour (c).

Algos b/w Hill search & Stochastic methods

(purely exploitative) (purely explorative)

Make a random move with a probability ∝ improvement in eval(n)

c —random→ n

move with a probability on basis of eval(n) − eval(c).

if this ↑↑ probability ↑↑ ⟷

if ↓↓, doesn't stop, just p ↓↓

Unit 5 :-

5.2 - Branch & bound
5.3 - Refinement search
5.4 - Dijkstra Algo
5.5 - Algo A*
5.7 - Iterative Deepening IDA*
5.8 - Recursive best first search RBFS
5.9 - Pruning CLOSED list
5.10 - Pruning OPEN list
5.11 - divide & conquer Beam stack search

5.4. Dijkstra

↳ shortest path algo on graphs
• solves single source problem. → finds shortest dist. to all nodes from src.
• needs the complete graph for the algo

5.7 Iterative deepening A*.

↓ added more qualities to A*.

↳ combination of DFS + BFS.

Adv (sort of)
• less space
• incomplete, infinite disadv

↳ Adv :-
• Quality sol^n
• complete
disadv
• space

↓

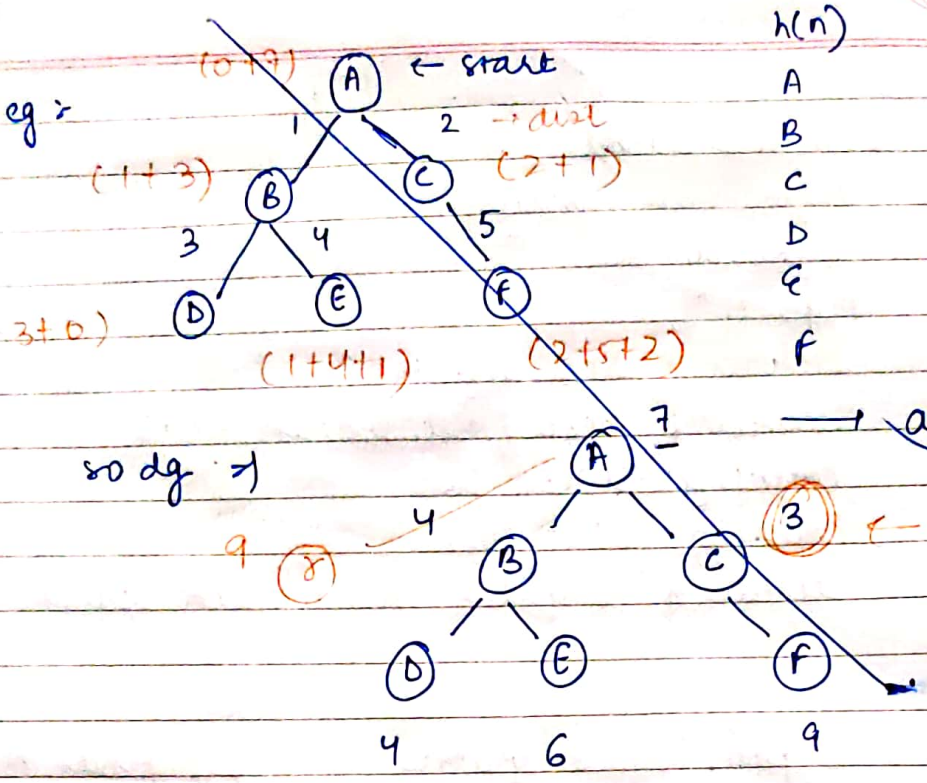
• It provides depth search which is limited to some F-bound.
• uses $F(n) = g(n) + h(n)$

algo :-

step 1 :- first set the threshold as the least value where $F(n) = g(n) + h(n)$ F(n).

step 2 :- Now by checking if the threshold ≥ the child then expand.
else, check for the least F(n)

step 3 :- continue till you reach goal node



least value = 3

hence threshold = 3.

start At A → 3 < 7 X

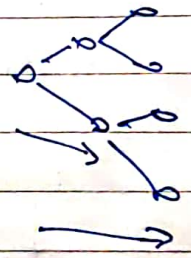
next threshold = 4

start at A → 4 < 7 X

5 → 5 < 7 X

6 → 6 < 7 X.

so, we choose A.



IDA* is to A* what IDFS is to DFS.

↳ converts A* to linear ~~search~~ algorithm space (at expense of ↑ TC)

Algorithm.

(i) set certain threshold (cutoff).

(ii) if f(node) > threshold . prune it.

(iii) Now, set threshold = min (all nodes that were pruned).

(iv) start again w new threshold

(v) stop once goal is reached

5.8 Recursive Best first search (RBFS)

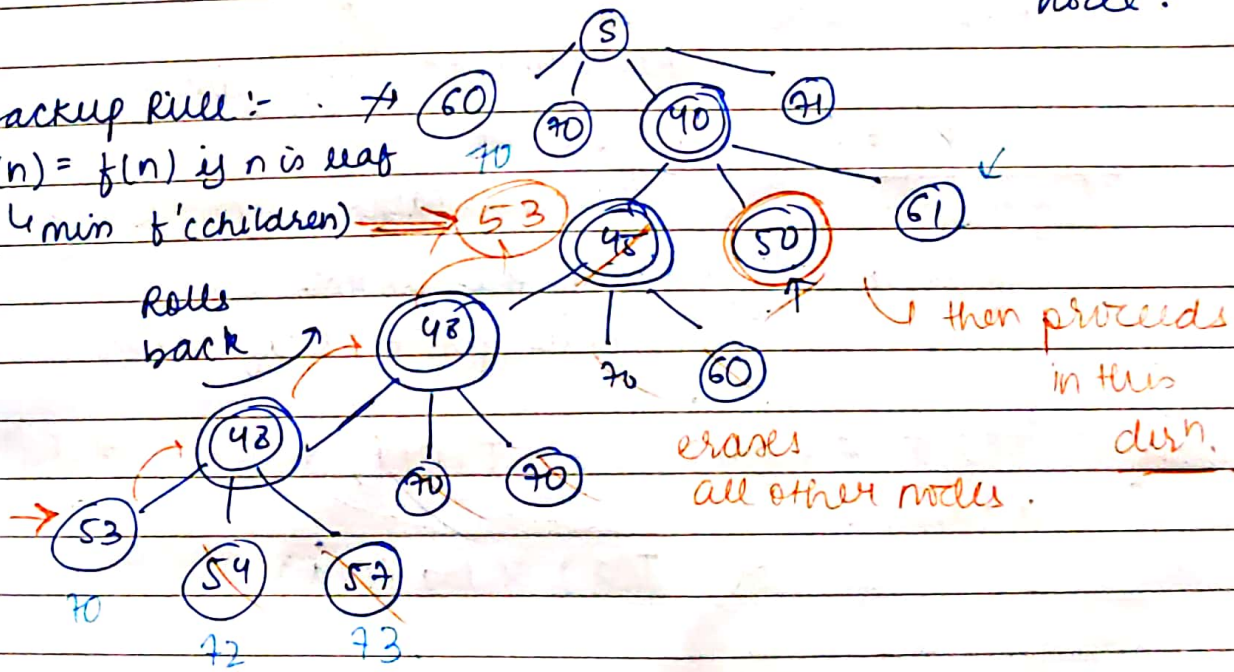
↳ modifications of A* that are space saving.

IDA*

no sense of direction within the threshold.

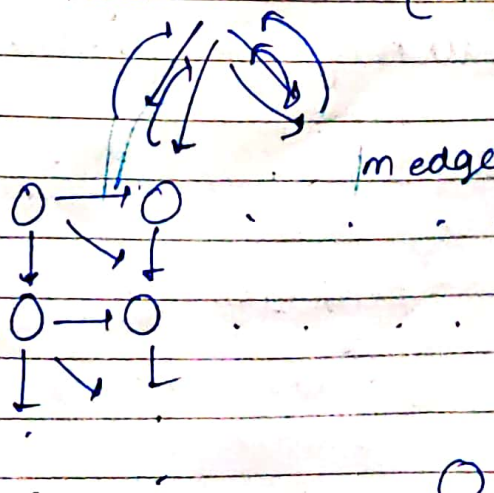
Recursive BFS → maintains a pointer to second best node.

Backup Rule:- f'(n) = f(n) if n is leaf
or ↳ min f' children)



erases all other nodes.

- at any pt = maintains only one path down the search tree
  ↳ linear SC
- THRASHING. (Ridges in Hill Climbing)



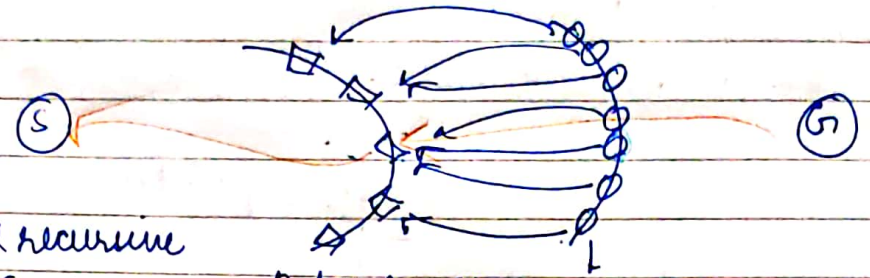
when only left & right

$= \frac{(m+n)!}{m!\,n!}$

→ Pruning Open & closed lists.

- PRUNING CLOSED. ① Stop search from leaking back
② Path Reconstruct.

* DCFS ⇒ Divide & conquer Frontier search



- then make 2 recursive calls to DCFS S→R & R→G. & aise...

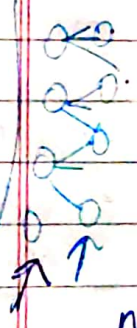
Relay layer ↓ halfway layer $g(n) \approx h(n)$

open list

- Throwing away most of the closed list.
↳ only open + Relay layers

- $T(d) + 2T(d/2) + 4T(d/4) + \cdots \times T(1)$.
d = depth.
$\sim T(d) \times \log(T/d)$

* Smart Memory Graph Search
↳ Run it like A*
keep a track of how much memory you're using
If you start running out of memory - prune it.

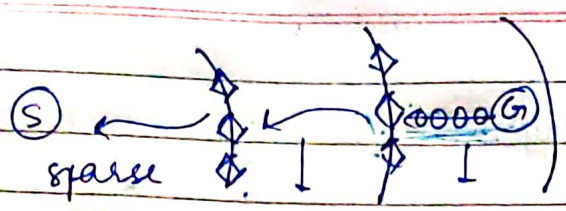


CLOSED

Kernel ↓ no children in OPEN list

Boundary ↓ at least one child in OPEN list

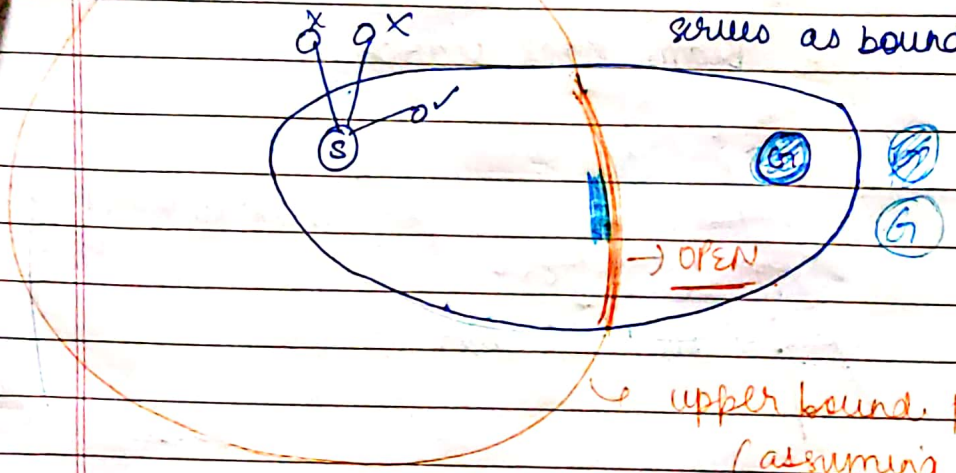
Prune the Kernel ↓ & convert boundary into Relay.



- Pruning OPEN.

  * Breadth First Heuristic search.

    - most rely on getting some upper bound on cost
      ↓
      serves as boundary



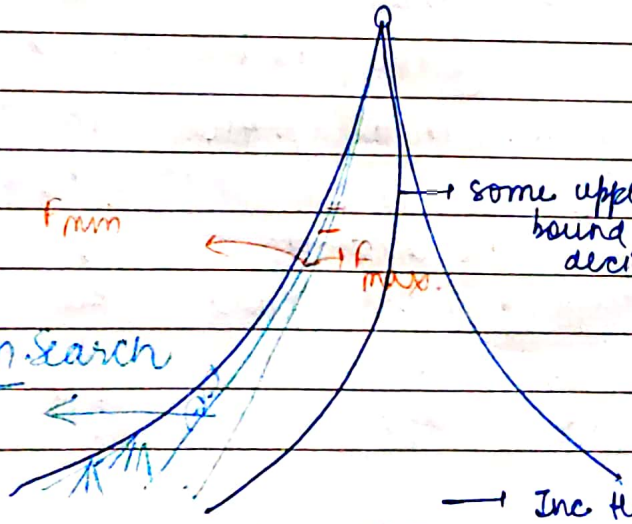
can be converted into Divide & Conquer BFHS.

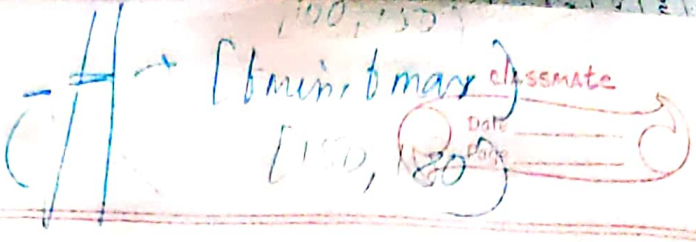
↳ upper bound for breadth first (assuming equal cost).

to prune it even further. → you fix a width essentially what beam search does.
↓
not complete.

  * Beam stack search.

Beam search + not complete.
↳ somehow needs to be backtracked
↓
- Beam stack.





Beam stack.

in every layer = $[f_{min}, f_{max})$

Initially every layer = $[0, \infty)$

We say

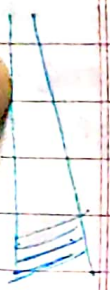
100 150

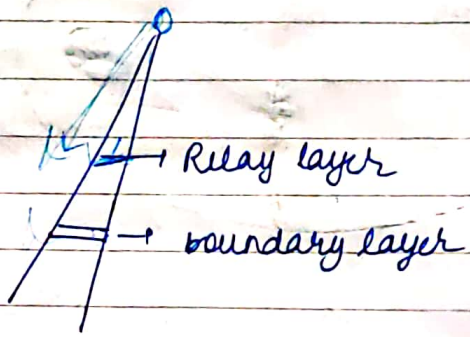
$[f_{min}, f_{max})$ → we update this

you've backtracked to this layer

[150, 180)

* Divide & Conquer Beam stack search.





how to backtrack now.

Constant amt of space.

eg TSP.

* Refinement search.

• start w/ set of all candidate solutions (represented by root node)

↓

every new node keeps on representing a subset of the possible solutions.

↳ can be seen as a partial soln.

B & B → selects one node & refines the partial soln it represents by specifying more factors (eg. CB, CB', MC).

↓

BGR

↓

GM

} for eg.

• Assuming that we have an algo to