Date 08 Aug
Page

AI

Definitions

① We call programs intelligent, if they exhibit behaviours, that can be regarded as intelligent, if they were done by human beings.

– Herbert Simon

② AI is the study of techniques for solving exponentially hard problems in polynomial time eventually by exploiting the knowledge about problem domain.

– Elaine Rich

1. Thinking humanly	1. Thinking Rationally
2. Acting humanly	2. Acting Rationally

3 dimensions :–

1. Introspection
2. Experiment (observation)
3. brain imaging

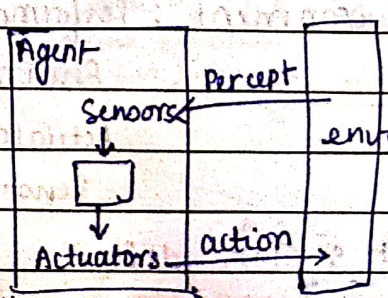
} Thinking ability of a human.

- Computer vision
- NLP (natural lang. processing) : interpret the lang & give response
- ML abilities : based on input, ⇒ Gaining knowledge.
- Knowledge representation /r : storing data on the basis of some knowledge
  : Reasoning
- Robotics.

neural networks

* Intelligent agents :

1. Agent
2. Percept : i/p from environment
3. Percept sequence : i/p history
4. Agent func. : abstraction
5. Agent program. : implementing abstraction using some program.



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

[A, clean] [A, clean] Right
[A, clean] [B, dirty] Suck

function

function vacuum-agent ([location, status])
return action
if status = dirty
return suck
else if location = A
return right
else if location = B
return left

* Rationality
performance measure
(1) success
(2) prior knowledge of environment
(3) Actions, based on perception history. Actions are given to actuators.
(4) Percept sequence : provided by sensors.

* Rational Agent
Task environment : Performance measure
Environment
Actuator
Sensor
} PEAS

eg. Agent : Taxi driver
Performance measure : safe, fast, legal, trip, maximize profits
Environment : Roads, traffic, customers,
Actuator : Gear, Steering, Clutch, ~~Steering~~, horn, display,

Date / /
Page

input sequence

Sensors :- camera, Sonar, speedometer, GPS, Keyboard,

PEAS for
★ medical Diagnosis System
Agent : System.
Performance measures : accuracy, cost effective,
Environment : Lab, Hospital, patient,
Actuators : Display the Question, Referral, Test Suggestion
~~Test~~ Sensors :- screen, K/B,

9 Aug
★ AI
Write PEAS / task environment for an interactive tutor.
(i) Agent : Tutor

Performance measure : ~~Proficient lang~~, ~~knowledge~~ result & test & score,
Environment : Class Room, Student, Questionnaires, Test
Instructor/ Actuators :- Displaying of Marks, Remarks, Suggestion
Sensors :- Questionnaries, K/B (enter the info & store it).

★ Different ~~Pr~~ Properties of task Environment.
1. Fully observable vs Partially Observation
- Agent doesn't know certain info. of environment (having some abstraction)
eg Vacuum cleaner (don't know 2nd room info)
Taxi driver (what's going on in the mind of other drivers) : "which route they would choose"

no sensor → unobservable

2. single vs multi agent.
eg. cross word puzzle
eg chess ~~ludo~~ ludo

+ = uncertain environment
3. Deterministic vs Stochastic — based on I/P can't determine next step.
eg chess, vacuum cleaner
Dynamic environment is there
eg Taxi driver

4. Episodic Vs Sequential — taxi driver.
no dependencies
- checking whether machine part is ~~defect~~ defective or not
- all actions are related, actions are performed in a time frame.
- one after other

classification eg of episodic

route will be given which driver taxi have to follow hence sequential

Date / /
Page

★ (5) Static : and Dynamic : define w.r.t (time) & (actions).
/ semi dynamic → environment (chess) based on agent performance, envi is changing & time is constant.

for definite i/p & definite action is there

Vacuum cleaner, puzzle | taxi driver (surrounding will change in the time frame).

(6) Discrete vs continuous. (depends on how many states are there in the environment).
finite no of states.
*(action & input will be discrete in the environment) (eg chess, move after each move you will define your action with finite states)

↳ synonym to sequential

(easy) (7) known vs Unknown environment. (hardest environment) can be fully observable eg. Video games we know all the button but don't know how will it work
deterministic can be partially observable. eg. solitaire
(what are future states not known)

★ Example.
(1) crossword puzzle
Fully, Sngl, D, sequent, static, Discrete
(2) chess with a clock
F, M, D, Seq, semiDy, Dis.
(3) Image analysis — use matrices
F, S, Det, Ep, St, conti
(4) Part-picking robot
P, S, Sto, Ep, dy, conti. — what you do with part.
only part is given to u
(5) Refinery controller
P, S, Sto, Seq, Dy, continuous — depend on prev. condn
only one mach. what will be next move

★ Agent = Prgm + architecture
MAP (i/p, actions) — model used to program agent
To write MAP func, we need to consider architecture.

Date / /
Page

action: organised in a look up table
↳ size is depends/constrainted by lifetime of agent, T and no. of percept input, P.

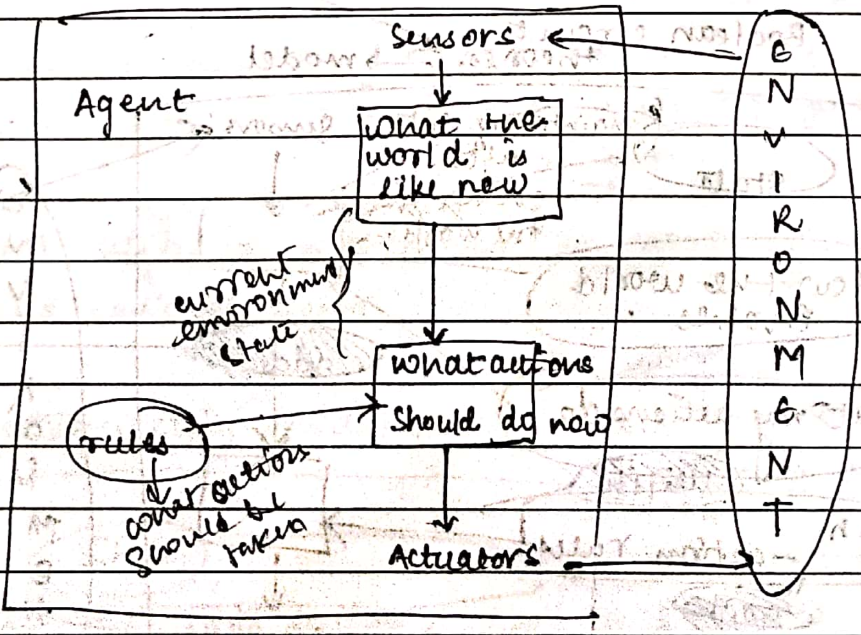
for the entire lifetime of agent: # of Actions = Lookup Table → # of entries in = $\sum_{t=1}^{T} |P^t|$ (P → no. of percepts) : very large no. (so not much suitable.

Types of Agent Program
★ 1) Simple Reflex Agent
2) Model based Reflex agent : take actions based on Model
3) Goal based Agent : Goal define prior to i/p & action
4) Utility based agent : follow some utility function useful.
useful func(s) to reach goal,
only useful func(s) will be considered

models diff. of agents

★ Imp Fig

For every action there is a specified rule :
action — rule



: implemented using logic gates

function SIMPLE_REFLEX AGENT (percept)
return an action.

Scanned by CamScanner

* no history only on current situation state

persistent : rules, a set of condition-action rules;

state ← INTERPRET-INPUT (Percept)

rule ← RULE-MATCH (State, Rules)

action ← rule . ACTION

return action;

Suitable for Fully observable environment, otherwise in other case it will go into infinite loop/state; hence not act rational. To avoid/tackle such condn we use randomized technique.

Agent — Randomized approach in case of multiagents; Deterministic suitable to make a rational agent single user

(1) Problem : Partial Observability.

(2) Model Based Reflex Agent [having some provision to store/saving info about history of environment from some state, called internal state]

[all info should not be stored bcz it require more storage like earlier tables, only some history info.]

What should be stored in internal state

(1) World evolves : independent

(2) Effect of actions of the agents : dependent on agent's action.

Boolean circuit theories. → model

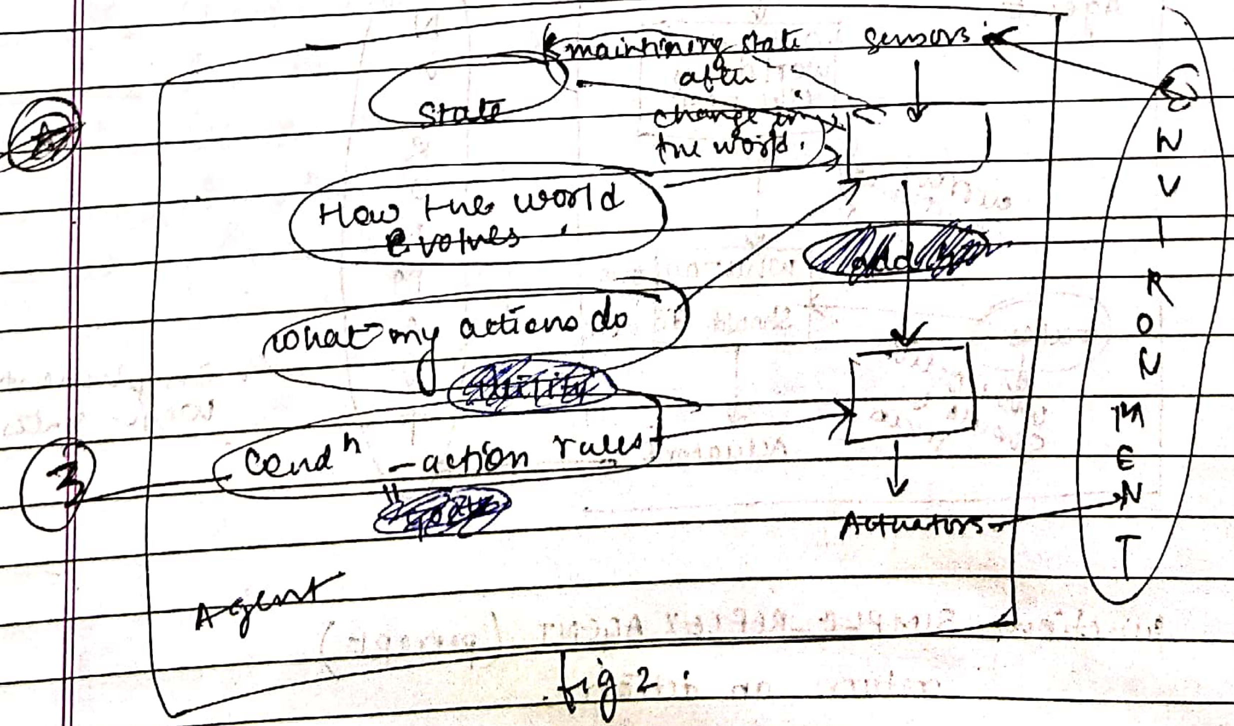


fig 2 :

Date / /
Page

RANKA

function MODAL-BASED-REFLEX-AGENT (percept)
return action
persistent: state — (current state of the agent)
models —
rule — (what is next action based on current state)
action —

* internal state always get updated (algorithm)

state ← UPDATE-STATE (state, action, percept, model)
rule ← RULE_MATCH (state, rules).
action ← rule.ACTION.
return action

(3) Goal-based ! we should know what should be achieved by agents.
* action performed by agent should leads to some goal

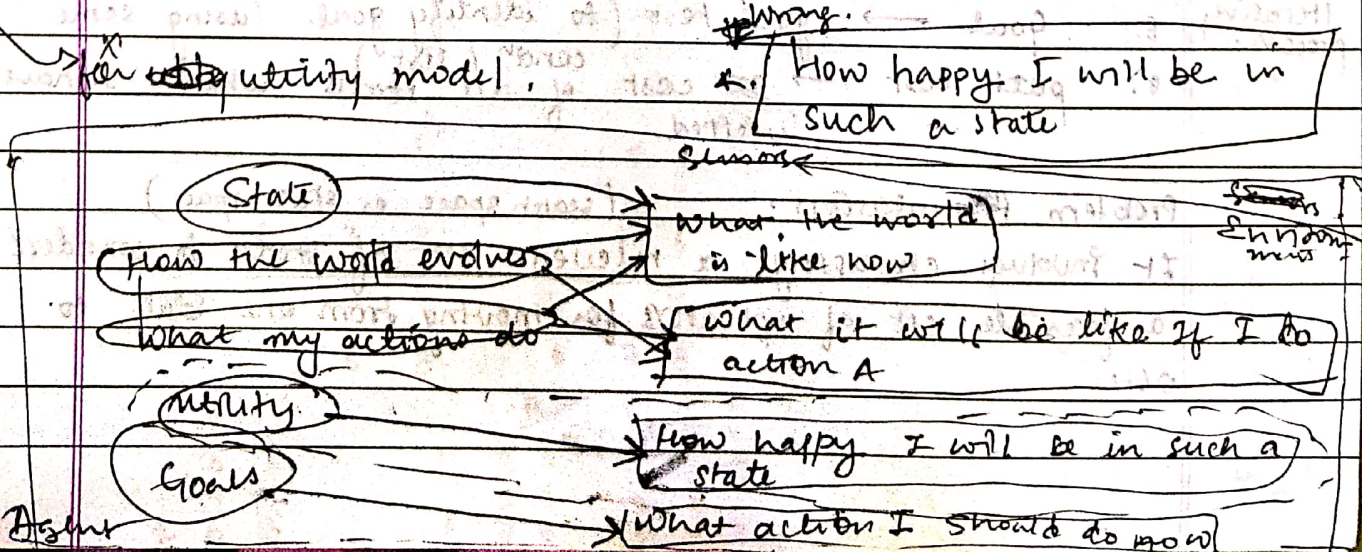
→ replace (3) by Goals in fig 2

Goal based = searching + planning — AI subfields.
- finding the different routes & then do the planning.
maximise performance measure

(4) Utility-based ! considering only those actions which maximise utility func in performance &, not all action.

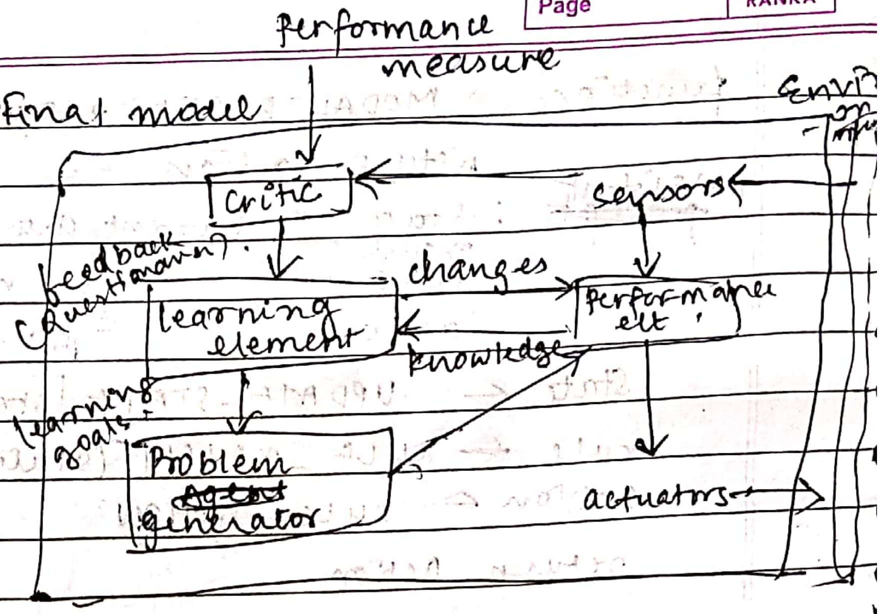
If this utility is maximized by some agent then that agent will be called rational agent.

→ for utility model,



(A learning agent: function defined internally ≠ external performance, then rational agent.)

(4) Final model



4 components of a learning agent.

* 3 diff ways in which these components are working

1. atomic : individual action not combined (eg. reflex action)
2. factored : every action can be factored using propositional logic
3. Structured : ← First order logic (FOL)

23 Aug) **Searching :**

* Problem solving using searching / state space search (all set of state is given which will be needed to consider).
— Goal based agent.

assumptions : choose a sequence of action to reach ~~the~~ a desired goal.

1. initial state : (initial configuration of agent.)
2. set of states
3. action / operator (following which states can be change. to move from one state to another)
4. plan — on reaching every new test, goal test will be performed.
5. Goal → goal test (to identify goal / using some condn / test)
6. path cost → cost of all states which u have visited.

Iterative process.

Problem formulation : (search space or state space)
It involves choosing a relevent set of states to consider & a feasible set of operators for moving from one state to other.

Date / /
Page

- **Search** is a process of imagining sequences of operators applied to the initial state and checking which sequence of state will reaches the goal state(s).

- Search frameworks :
  1) Uninformed search / Blind search (no info. abt Problem domain) [eg. BFS, DFS]
  2) Informed search / Heuristic search eg. Hill climbing, LS, GA
     known info abt prob. domain ; Heuristic func. represents.

- Search Problem Formulation — initial state
  Four main components : [S, s_0, O, G] → set of goals (there can be multiple no. of goals).
  S: set of states; O: set of operators

8-Puzzle Problem.

initial state s_0 :

1	3	5
2	7	4
6		8

G:

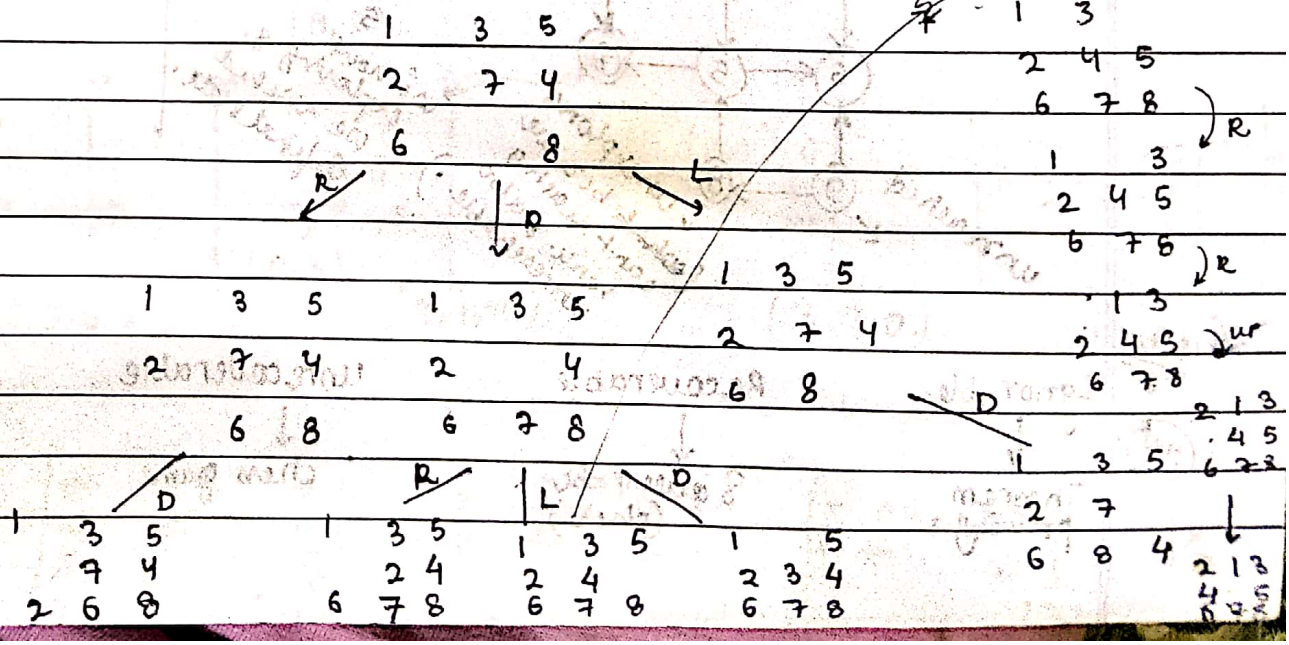
1	2	3
4	5	6
7	8	

state description(s): location of each of the eight slides & blank

s_0 = start symbol

operators : 4 operator : L, R, U, D

goal : one or more goal configuration.



Date / /

use DFS (since not exploring the state present in the same level)

- 8-queens problem : 8 queens should be arranged on chess board in such a way that no queen attack other queen

5 operators : U, D, L, R, D

1) Iterative refinement improvement approach : placing all 8 queens at same posn at the same time & do improvisation (random approach; some rules diagonal, up, right, left, down; moving operators)

2) placing one queen at a time : long process (involves lot of steps)

→ Outline of search algorithm

Search algorithm (Basic) — contain all the state of search/State space.

(1) Initialize : set OPEN = {S} : Data structure | Stack | Queue
  (Stack ↑ DFS → Dense graph → adjacency matrix; Queue ↑ BFS → efficient for sparse graph → adjacency list)

(2) Fail :

  If OPEN = { } Terminate with failure

(3) Select : Select a state n, from OPEN

(4) Terminate :

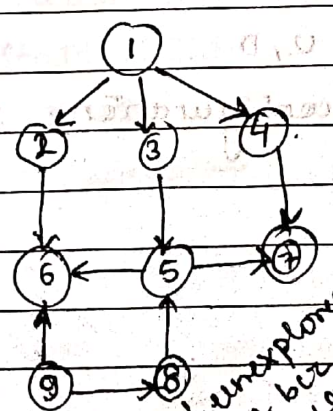
  If n ∈ G, terminate with success

(5) Expand :

  Generate the successors of n using O (operators) and insert them in OPEN

(6) Loop : go to step 2.

eg



unreached

unexplored but we can't reach 9 hence 8 also unreachable.

without exploring 8, we reached final state.

DFS (efficient)

| 1 | 2 | 3 | 4 | 7 |

Stack 7

5

BFS

| 1 | 2 | 3 | 4 | 6 | 5 | 7 |

Queue

sets

- Ignorable → Theorem proving.
- Recoverable → 8 queen problem (chess)
- unrecoverable → chess game

(HW)

DS to Implementation

Date 26 Aug
Page

RANKA

★ Uninformed Search

(Blind) BFS ← Priority Fn for UCS : uniform cost → Path cost (sum of cost of paths taken)

DFS

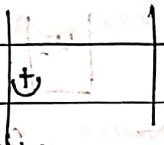
DLS : Depth limited Search

IDDFS : Iterative Deepening DFS } Bomb^n of BFS & DFS

BS : Bidirectional Search }

Prob!) Missionaries and Cannibal Problem.

m: 3 c: 3



State (#m, #c, 1/0)

#m : no. of missionaries at the first bank

#c : no. of cannibals " " " "

last bit : whether the boat is in the first bank

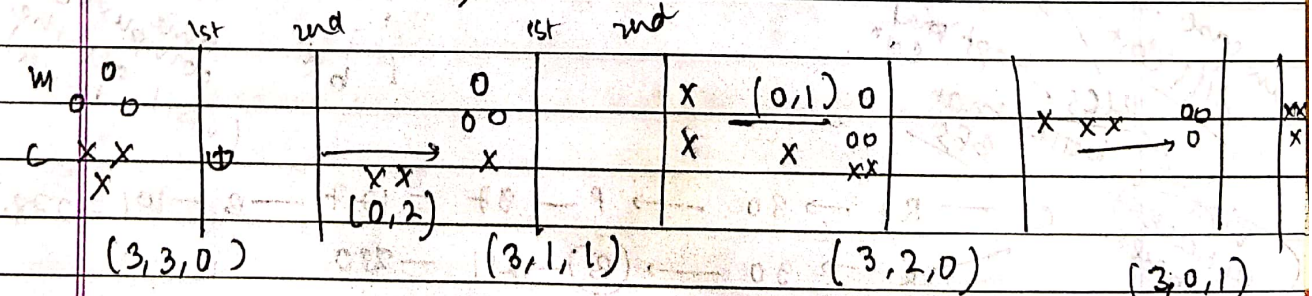
Start state : (3, 3, 1) Goal State : (0, 0, 0)

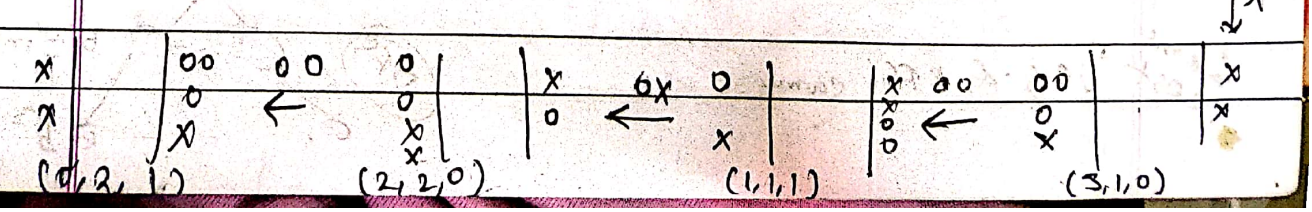
Operators : Boat carries

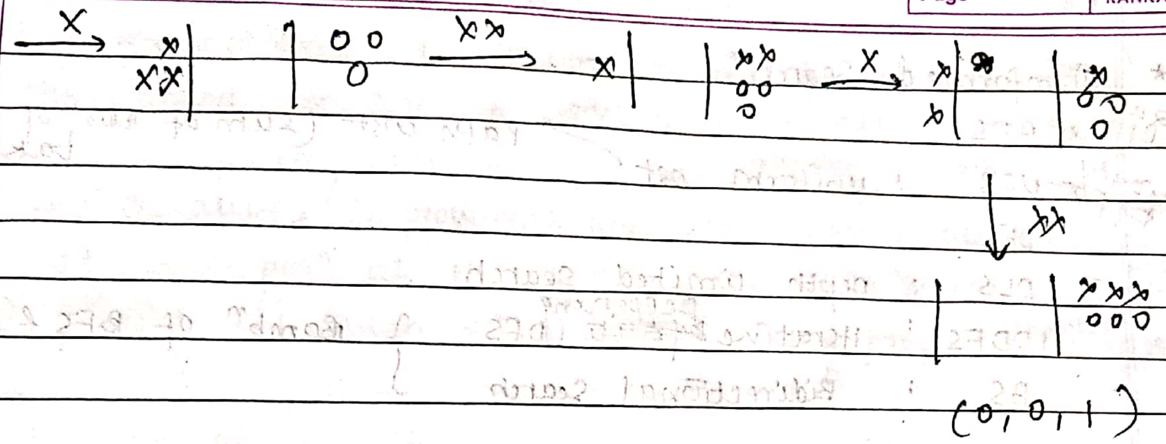
(1, 0)
(0, 1)
(1, 1)
(2, 0)
(0, 2)

} # of items can't be more than 2.

At the bank, cannibals can't be more than missionaries i.e. c ≤ m







Prob 2) Water Jug Problem

|4L| |3L| first Fill 3L container.

2L

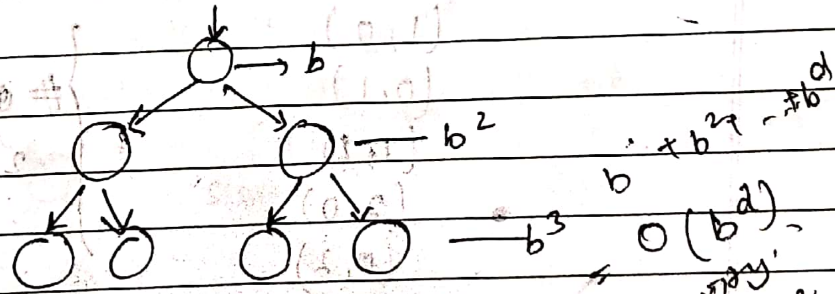
→ Search Tree : best efficient way to define a search space

complexity ← ③ Space ② Time ① completeness of an algo. → there exist a soln to the prob.

: check algo with incorrect i/p(s). (wrong case).

To examine efficiency of any algo:

① complete
② Optimal ← maximise problem objective
③ Time
④ Space



$b + b^2 + \dots + b^d$

= $O(b^d)$ time complexity (visit all the nodes)

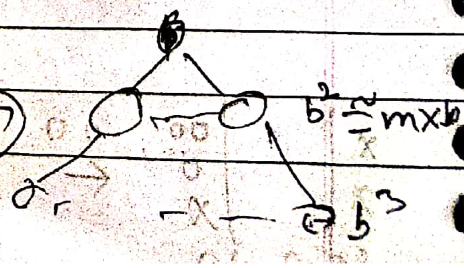
in case diff cost: UCS → optimal soln, better than BFS (a loop may occur) → visit unnecessary nodes.

S — R1 → 80 → P — 97 = 177 — B — 101 = 278.

→ R → 90 → (B) — 211 = 310

same cost : UCS & BFS : same

DFS



branching factor & depth of the tree

$O(b^m) < O(b^d)$

Date / /
Page

$b + b^2 + \dots + b^d$ = BFS } Space complexity.

$m \times b$ = DFS

and $(m \times b) < b^2$

when depth is not known then go for BFS

since BFS will be incomplete and not optimal when d is unknown

(★) DLS :- some limit depth, l : avoiding algo. to go into loop.

$l < d$: optimal

→ less time complexity than DFS.

(★) IDDFS : DFS + BFS at each level. 'l' value put the value on dfs'.

[l = 0] — DFS ← not optimal, not incomplete

\+ BFS → space, time ↓ also reduces

★ y hybridisation of algo

[a b]

(15, 15), 30 → a + b (20, 10)

a − b = 10

$d \cdot b + (d-1)b^2 + (d-2)b^3 + \dots$

$O(b^d)$

★ BS : Termination → (Goal State) — as a common state where they can meet

→ 2 search techniques — applied at root & one at bottom.

Ques

(1) what search techniques both are same, both are different.

b^d } search space, b, $b^{d/2}$, $b^{d/2}$ — time & space comp. reduced to $O(b^{d/2})$: i.e. half

$b^{d/2} + b^{d/2} = O(b^{d/2})$

(2) [Initial to goal] algo: traverse from

(3) is it possible to generate same successor from both the algo.

(★) OPEN list (algo using only one) : search, opp: Closed → list of all visited node.

chance of repeating the same state → avoid repetition

→ maintain list of all unexplored nodes

Date / /
Page

exercise: Identify where this algo will fail. for all algo.

A* algo: informed search: Next class.

Assignment: BFS, DFS and A* algo (10 marks) {next friday 6/9/19

29 Aug) Informed search algorithms:

Ques) 1) consider the following as a state-space search problem. choose a formulation that is precise enough to be presented.

(i) How would you represent a state.

(ii) Describe the initial state & the goal state/goal test

(iii) Describe the successor func or operator.

Problem1. You have to colour a planar map using only four colours in such a way that no 2 adjacent regions will have the same colour.

Problem2. In the TSP there is a map involving 'n' cities which are connected by roads. The aim is to find the shortest tour that start from a city visit all the cities exactly one and comes back to the starting city.

* Best-first search
→ greedy best-first search

(Cost function / evaluation / heuristic func. (h(n)) defined for each nodes.

• A* algo: first heuristic algo which gives optimal soln.
  • admissibility • consistency

* Best first search (↑ h(n)) (increasing order)

Date 30 Aug
Page

RANKA

- Basic Graph Search Algo.

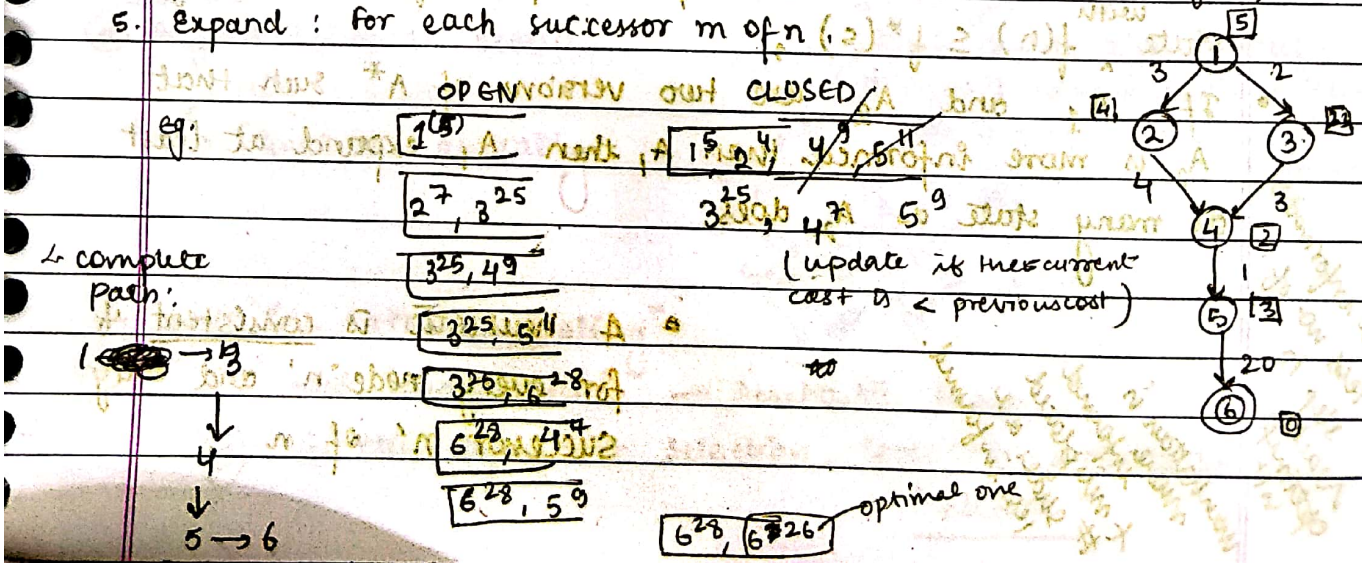
(1) Create a search tree, T_r consisting solely of the start node, N_0, put r_0 in OPEN.
(2) CLOSED = ϕ
(3) If OPEN = ϕ, exit with failure.
(4) select first node 'n' from OPEN, OPEN = OPEN − {n}, CLOSED = CLOSED ∪ {n}
(5) If n = goal node, exit successfully, path = arcs in T_r from n to n_0
(6) Expand n, generating (generally) m successors, ("not include any ancestor")
Install 'm' as successors of n in t_r by creating arcs from 'n' to each member of m.
(7) Reorder the OPEN
(8) Go to Step (3).

Note: (ordering technique)
- OPEN — Stack (LIFO) (implemented using) → DFS
- OPEN — Queue (FIFO) → BFS

- A* algorithm

1. Initialise : Set OPEN = {S}, CLOSED = { }, g(S) = 0, f(S) = h(S)
2. Fail : If OPEN = { }, Terminate and fail.
3. Select : Select the min cost state, n, from OPEN, save 'n' in CLOSED.
4. Terminate : If n ∈ G, terminate with success & return f(n)
5. Expand : For each successor m of n

eg:



If $m \in [OPEN \cup CLOSED]$
set $g(m) = g(n) + c(n,m)$
set $f(m) = g(m) + h(m)$
insert m in OPEN.
If $m \in [OPEN \cup CLOSED]$
set $g(m) = \min\{g(m), g(n) + c(n,m)\}$
set $f(m) = g(m) + h(m)$
if $f(m)$ has decreased and $m \in$ CLOSED:
then move m to OPEN.

★ Properties of A*
1) Admissibility : guarantees optimal soln.
2) Consistency / monotonocity

- A heuristic is called admissible if it always underestimates i.e. we always have $h(n)$ [i.e. heuristic value of node 'n'.] $\leq f^*(n)$ where $f^*(n)$ denotes the minimum distance from a goal state to a node 'n'.
- A* is admissible ie. if there is a path from s to a goal state then A* terminates by finding an optimal path.
- For finite state space A* always terminate.
- At any time before A* terminate, there exist in OPEN a state n' with $f(n) \leq f^*(s)$ in an optimal path from s to a goal state.
- If A_1 and A_2 are two versions of A* such that A_2 is more informed than A_1, then A_1 expand at least as many state as A_2 does.

more information ⇒ less no of step to reach optimal soln

Uninformation is more informed ⇒ misplaced ⇒ tiles are less informed.

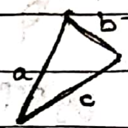
- A heuristic is consistent if for every node 'n' and every successor n' of n,

generated by any action 'a' the estimated cost of reaching the goal from 'n' is not greater than the step cost of getting to n' + the estimated cost of reaching the goal from n'.

i.e. $h(n) \leq c(n, a, n') + h(n')$ [followed from Δ inequality]

(a ↑ action)

$a + b \geq c$



- If search space is infinite, what will happen?

Problem with A*:
- Time complexity : Exponential
- Space complexity : keep all nodes in memory

] to reduce / solve this we go for IDA* (variation of DFS)

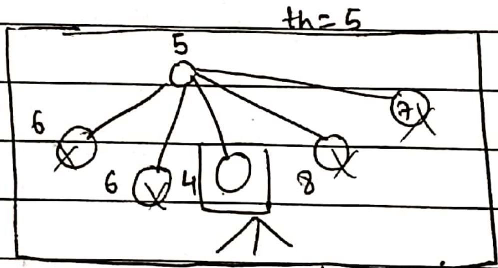
(Iterative deepening DFS + A*) ≡ Iterative deepening A*

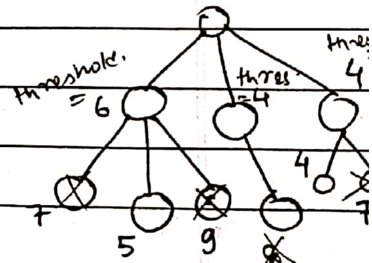
IDA* : ∂ (g+h) ≥ Threshold

repeat:
1. DFS, prune if
2. Initial threshold = estimate cost of the 'S'
3. Next iteration threshold = min (all cost ≥ Threshold)

eg:

th=5





Difficult Implementation

since algo is complex (since it is combⁿ of two algorithm).

* " can be implemented with less memory requirement bcz of pruning."

5th Sept

★ Beyond classical search
- Hill climbing
- Simulated annealing

" local search algo → specify complete configuration of search space ; fully observable " environment

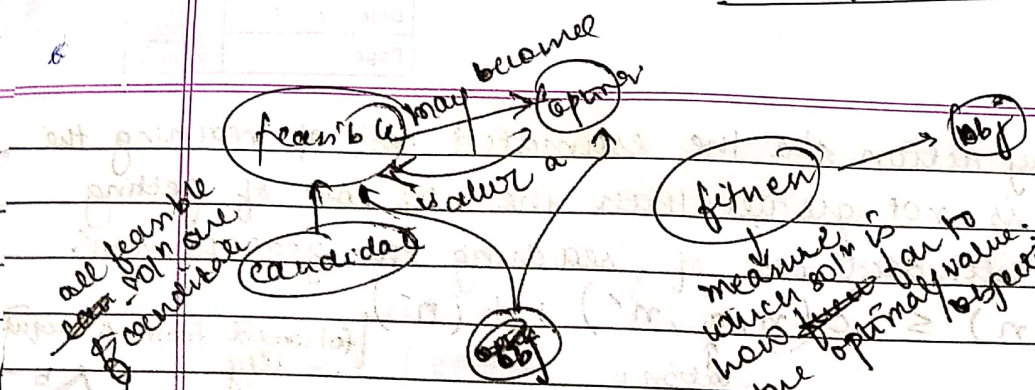
→ can be used for 1D, space since it always check the neighbour

heuristic algo. ↓ since highest value used for comparison heuristic func.

06 Sept

1. "Simulated annealing"
2. "local beam search" — stochastic beam
3. Genetic algo — selection, crossover & mutation
   "Elitism"

least fit soln gets eliminated during process!
*elitism



09 Sept

Constrained optimization

- Continuous (∞ state → finite states)
- to solve continuous domain problem.

Limit the search space.