Consistant too.

MMI

L-02

Intelligent Agents

Russel & Norvig Ch-2

Linear time algo develop করতে চাই,
User perspective থেকে requirement of algorithm কেমন?

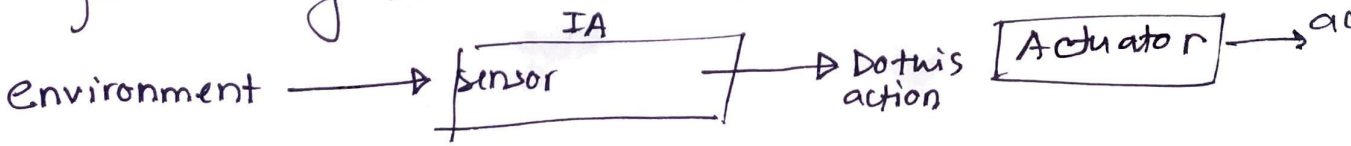
ATM machine এর security threat বেশী than issuing money with checkbooks.
We can ensure security by multiple way authentication an expensive hardware.
But multilevel authentication দিলে beats the purpose of such ATM machine.
If an algorithm is not full-proof, that does not mean it's not usab
we can improve the performance, but we don't remove them.

Intelligent agent: Intelligent Agent

Agent collects info from environment and takes a decision.

Car + driver → an example of intelligent agent.

decision to move is generated by driver, but ~~where~~ ac is generated by the car itself.



Agent should decide by themselves.

* Self driving Car: Tries to be intelligent.

Look up table use: Sensor val ⟶ action. Using this it is generating action. This is $\boxed{\text{Automatic}}$ System

Autonomous: Works according to the previous experience and generate decisions based on this.

* Traffic light is an automatic system
* Dynamic system, whose pattern is unknown to us, this ~~is an auto~~ needs an autonomous system.
* Industrial sector can use automatic system as patterns and workflows are known.
* Space and Oceans/sea ⟶ Cannot design ~~autonomous syste~~ automatic systems.

* These agent program which works good for autonomous System are data driven. We supply data and these data will be used for learning the pattern within data.

* HW must have sensors, actuators and effectors
* inside there is an agent program having processing power.

Homework: CMU → Automated taxi driving

L-03

MMI

 # Rationality: rational agent should for each possible percept sequence, do whatever actions will maximize it's expected performance measure based on:

1) the percept seq and
2) its built-in and acquired knowledge.

Sensors placed on roads, based on sensor reading, car ~~the~~ when is understood and based on that the ~~car will~~ traffic light will switch and show lights.

 # Table driven agent: Sensory information পাবো, Table look up করে corresponding action will be implemented. unwise, because memory ~~#~~ will be too big to generate and store. Besides everytime we ~~m~~ need to look up and do an action.

Non perceptual parts: একটা agent এর খুব পর্যন্ত চেনেই না, so, those information is not present in the look-up table.

Now if we see that for a certain sensory perception, the action is reducing performance of the agent, we need to update the table, but can't do so easily as fixed table.

 # Simple reflex agent: Sensory info → action না হয়ে multiple sensory info → action of table. Otherwise same drawbacks by table-driven agent.

memory Based agent: past experience is kept inside the memory of agent. Because changes in environment will actually assist the agent to make better assumptions

#VVI **Brooks Subsumption Architecture.** complex behaviour কে অনেক গুলো ছোট ছোট independent simple behaviour এ ভাগ করে নিবো।

SB → simple behav
CB → complex "

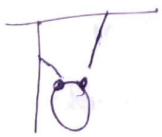
$SB_1 + SB_2 + SB_3 = CB$

we will divide behaviour in such a way that perception and action will be associated with the simple behaviour. If a situation ~~is~~ can be done by SB, CB এর part দিয়ে we can solve the problem.

Now what is the diff betn this and previous arch?
Previously each task was considered as a b. 5000 tasks → 5000 b. But Now a complex b is thought of a ~~simple~~ combination of simple b.

Goal Based agent: Previously, perceived info → action. But now we are doing this. perceived info + goal → action. Each action will take me closer to the goal.

Example (MM/ sir):



→ Two sensors are giving same data. Table driven হলে, ~~ব~~ decision will be random, but goal based better performance দিবে।

Utility based agent: utility use করবে।

Properties of environment: Agent has relation to env.

Fully observable: env এর সব info agent sensor use করে collect করতে পারবে। Agent যে position এ থাকুক না কেন, it can do it. and collect info from the environment.

Partially observable: can not take full info using sensor from the env

Deterministic: controlled env, যেখানে ~~e~~ Scenerio change হবে না।

Stochastic: ~~Ato~~ uncontrolled env, যেখানে scenerio change হবে।

Industry — Deterministic
Real-life scenerio — Autonomous vehicle — Stochastic.

MMI

L-04

Properties of Environments II

Episodic: An episodic environment means to accomplish a particular environment's task, there is no dependency with previous task.

Sequential: Actions of agents are connected sequentially. (previous ১ৰ সৈতে connected).

Discrete agent/multi agent:

Single agent/multi-agent system:

Multi-agent systems might have other agents and there can be competition or cooperation among the agents.

Single agent – There is only a single agent in env.

Ch-03

Problem Solving and Search

Agent: Sensor
Actuator

=> Information perceived by sensors → Process info (soft part) (AI algorithm to process info) → Action implemented by Actuators.

W1 # Q. Why do you formulate goal before problem formulation?

*** Ans: Because goal জানা থাকলে, if it is a problem for a particular algorithm or not, that can not be known.

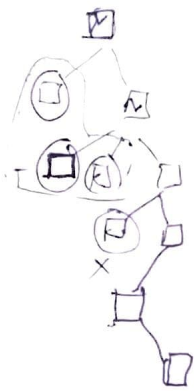
Goal formulation এর জন্য কোন কিছু requirements আছে।

① Firstly formulate goal

② Secondly formulate problem — বর্তমান scenerio + goal এর respect এ problem formulate করতে পারব।

Since state is changing, the problem formulation changes.

Example: Romania:



MMI

L-05

Problem-Formulation Problem is defined by 4 items.

initial state :

Successor function : $S(x)$ set of action-state pairs

path cost :

Tree-search ~~algothi~~ algorithms:

f Tree_search(problem, strategy)

initialize the search tree using initial-state of problem.

loop do

no candidate to expand - return false.

if candidate is goal - return true

if not, we expand a children and ~~then ac~~ we choose according to strategy.

state is physical config

Node is data structure for ~~int~~ implementation

fringe: fringe is a list consisting of nodes to be visited.

we pop a node from fringe and again expand the node if it's not a goal state.

How we keep and structure the nodes in the data structure depends on the strategy that we are using.

we consider that cost of traversing from one state to another as unit.

Search Strategy

it's decided, first node কে remove করব, so search strategy makes up the data structure.

completeness : does strategy always find a solution if exists?

~~# time complexi~~

optimality : does it always find an optimal solution?

These two are new metrics for strategy evaluation alongside with time and space complexity.

b - maximum branching factor

d - " depth of least cost soln

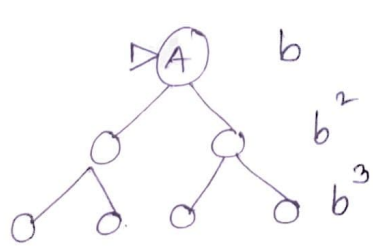
m - " "

Uninformed search strategies

BFS
UCS
DFS
DLS
IDS

They do not use any information regarding the solution, so they are called uninformed / blind search

In every step, b এর value add হচ্ছে।



Every time branching হচ্ছে and value will increase.
if depth is d, then branching factor is b^d.

* BFS is not optimal.

we cannot remove any node in bfs → a major drawback

so, space becomes a huge issue. in this case.

<u>Uniform - Cost search</u>: fringe = Q ordered by lowest path cost.

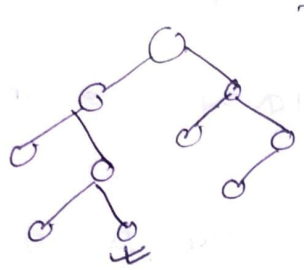
If path cost are same it is BFS.

MMI

L-06.

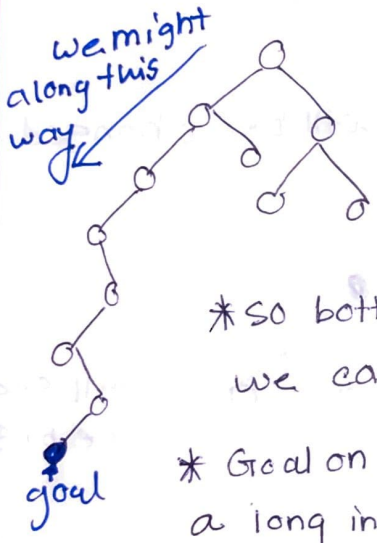
DFS: Two variations — depth limited search, Iterative deepening search.

Benefit of dfs is that we can keep part of the search tree outside of the memory. So, memory এর মধ্যে we get a benefit.



This part is kept out of the memory. So dfs is better than bfs in this case.

But this might not always happen. Suppose we have a non-uniform tree.



Here if we go along the deepest path/longest path to get the goal, so, lots of nodes to be kept in the memory in this case. So it is a major drawback of dfs.

* So bottomline is এমন depth এ যাওয়া off করতে পারলে, we can get better results of dfs.

* Goal on right side, in shallow depth but if we go along a long infinite path, we won't reach goal.

Space is linear, for DFS.

* DFS has a large search space for a graph.

* DFS with depth limit l,

certain depth পর্যন্ত search করব, considering solution is at depth l, set by the user themselves.

* user interactivity should be less.

We search for a certain l, next l এ আমরা আবার search করতে চাই, so we can device a user defined "l".

*Now what if limit is not set by the user, but set by the algorithms themselves?

* DFS is preferable for it's space requirements (linear) than bfs. So DFS variation is being discussed.

* BFS has a strength. If soln is in shallow part of tree, we can get the solution faster

* DFS strength is that deeper part এ থাকলে we can get a better result.

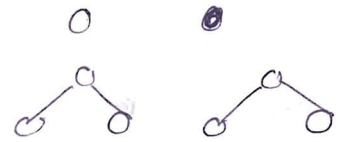
* So, we want best of boths: shallowest part এ থাকলে algo faster solution দিবে, (like BFS). Deep এ থাকলে DFS এর মত solution দিবে, so we get best of both. This IDS. (Iterative depening search).

VVI
IDS

- Limit is not fixed, over the time limit will be changed.
- Limit start ~~করতে~~ করা difficult.

So, let us start from L = 0

L = 1



will search like bfs.

* L যত কম, looks like bfs

* L যত বেশী, look like dfs

Time: $O(b^d)$

Space: $O(bd)$

optimal: yes if unit step cost.

*DFS complete না? Explain: Soln shallowest part, but a leaf is at a large depth and so, deep এ রেখে থাকবে but goal পাবে না, Cam R দিয়ে we cannot get a soln.

* **Repeated states:** Failure to detect repeated states might turn a linear problem to exponential one. So we should address this problem also.

MMI

L-07

Informed Search Algothirms.

In uninformed search, we just try to search for a node without any given information.

Informed search এ tree কিভাবে তৈরী করব, এই information টাও save করা যাবে। search এর সময় এই additional information আমরা input দিচ্ছি।

Now this measurement we supply will have an evaluation function, that would give us information based on the problem domain.

* desirability → additional evaluation f() use করে every node এর desirability বের করব, to test whether it is fit for our goal / not.

Best-first search:

Evaluation function is an heuristic function.

evaluation function h(n) heuristic = estimate of goal cost from n to goal.

* Let us consider that the distance is a straight line distance from node "n" to goal.

$h_{SLD}(n)$ = Straight line distance from n to Bucharest.

Straight line distance (an estimate. $h_{SLD}(n)$ is never larger than actual distance (actual road distance).

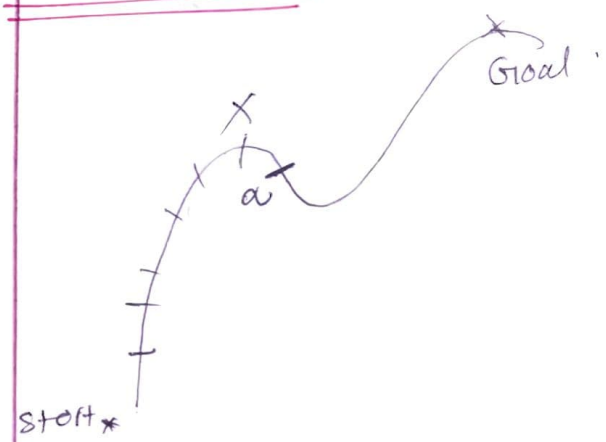
① We might need to go back (if we reach a state that does not look promising at all).

(2) আমরা certain stage এ দেখতে পারি কোনটা ভালো, মানে এখন যেটা ভালো, আমরা ওই অনুযায়ী আগাচ্ছি, but we cannot guarantee we will reach solution. This is a greedy choice, we are considering lowest at every stage and progressing.

Properties of Greedy Search:

Complete → No
Time → $O(b^m)$
Space → $O(b^m)$
Optimal → No.

Example by Sir:



~~কারণ~~ we want to climb up the hill (* start) এর respect এ next state এর height বেশী হওয়ায় উপরে উঠতে থাকবে and X এ এসে থেমে যাবে, because it's moving greedily, ~~it is~~ it won't move to a, so this is a problem. It ~~can~~ cannot go to goal due to this.

*So inherent drawback of a greedy algorithm is being

Stuck at local optima.

Soln: For sake of simplicity, lets assume our goal height is 5000 ft.
My current height is 3000 ft.

So, যেখানে আছি, we can start randomly from there from scratch.

Random restart: (1) from a diff start state, all parameters of search so far will be wasted.

* Euclidian distance from current to goal? what is ~~the~~ if this was a heuristic?

(2) Starting from such a state by the user, that would give me a better result.

(3) changing greedy property. Previous property was we only allowed ~~beio~~ climbing up. But when we are stuck at local optima, we consider that we would also go downward too.

MMI

L-08

* Misplaced tiles use করলে, relaxed problem এর, search space এর size অনেক বড় হয়ে যাবে। So, Number of visited nodes বেশী / number of explored nodes বেশী হবে।

Actual problem এর statespace অনেক বড় হবে, if we used misplaced tiles. So explored node count অনেক বেড়ে যাবে।

VVI [* CT time — Next monday]

Elements of heuristic approach:

Representation of solution space:

- Vector of binary values — 0/1 knapsack, 0/1 IP problem.
- Can be real too

* we want to color the whole map of world, with some condition. Then solution space is the color name of 4 colors.

* Defining the neighborhood and the neighbours:

Most ML/AI algorithms are iterative in nature. We cannot guarantee exact solution with it.
we want to make sure that we go to a better state than we are in, sometimes we may go along a path which is ~~bad~~ not good.
we take into account the presence of neighbors too.

← we consider they are '+' neighbors.

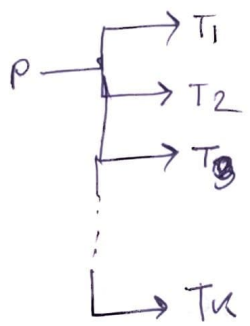
* Flip operator/ bits to pick up neighbors.

(3) * Defining the algorithm (3) → up to this common for all algorithms.

(4) Choosing the method.

— off the shelf or tailor made heuristic. ux কথা বলা।

Monir Sir example:



We take a techniane T_i that we know is good, and apply it. if it is not performing well, we choose another.

Everytime we take a Techniane, we start from initial ~~size~~ State. This creates wastage of resource from previous ~~past~~ tests. And even some ~~of~~ problems are hard to start from the starting state.

Population of soln → Starting from ~~a such a~~ point where we stopped.

Evaluation function: (H) এর বদলে বসাতে হবে

$$f = (1-S)(1-V)D$$

S = Sensor value

V = velocity difference betn 2w

D = distance.

* determines How far am I from object sensor object.

* as straight as possible হবে VD betn 2 wheels should be as small as possible.

→ * কত দূর distance যেতে পারে।