Artificial Intelligence

03/01/2025

- Imparting normal intelligence into a machine is called Artificial Intelligence.
- constraint of time & space → which intelligence is to be applied when and where.
- Thinking power → not possessed by machines.

Eg: 2001×1999 → diff. to calculate

$(2000+1)(2000-1)$ ← intelligence, only applicable here; not everywhere.

Brain is divided into 2 hemispheres: —

Left
classical AI.

Right
statistical power (primitive; existed before AI)

Book: Stuart Russell: Artificial intelligence: A modern approach

Motivations:

- AI is related to philosophy, psychology and intuitive thinking.
- AI problems are mostly non-deterministic → it contains uncertainity with approximate solution.
- AI is breadthwise not depth: AI is applicable more on a general level rather than specific cases.
- The specific part is not entirely discarded, and a part of it is taken along with the general part.

Heuristic → guessing power of human beings (not deterministic)
↳ Random search
↳ contains both risk and gain
↳ more thinking power to obtain better solution.

Eg) Block wall problem
A on top of B on top of A.
Planning to remove B (first as its top is empty.

Reinforcement learning → is not data driven. Based on function generating reward/penalty and transit on states accordingly.

Markov model → Next state depends on previous state.

Reinforcement learning involves one or more actions to move from one state to another and choosing the best one. (it is more than Markov learning).

Agent → Human/Robot

Turing Test
→ To determine whether a computer can think like a human.
→ Natural language, knowledge, reasoning power and learning power :- criterion to pass the test.

Expert system :- Human reasoning system through rules which are provided to machine.
Eg: If it is sunny, do not take the umbrella.

- Mycin → expert system used to prescribe antibiotics for diseases.
- Apple had developed AI that defeated human in chess (1997)

Phases of AI
- First → logic based → deterministic
- Second → probabilistic based
  ↳ involves uncertainty of real world.
- Third → neural n/w based
  ↳ requires huge amount of data.

Intro
AI → Simulation of human intelligence processes by developing computing systems that have the ability to perform tasks like humans

Humans	AI
→ Not always rational - biased / emotion driven	→ Always rational (doing the right thing)
→ Doesn't take max. goal always, takes risk sometimes.	→ Takes maximum goal

278. P-9

Logic + Domain knowledge is deterministic.

Knowledge representation → Rule, → Graph, → Vector } Also involves relation between the components of knowledge

Robots are sent where there is crisis to human life (mining/war/etc.)

AI → computation, logic, probability

~~Logic~~, searching

- Simple searching → no opponent; just follow a path
- game searching (used in AI) → multiple no. of opponents (agents) whose actions/ moments are known to each and every agent.

Search

Heuristic search is the fundamental technique of AI.

- Blind (Uninformed) → All paths need to be explored. → Info (knowledge) not available. e.g. BFS
- Informed (Heuristic search) → Guess what is ahead & that info is used. e.g. expand a particular node which is best at that level.

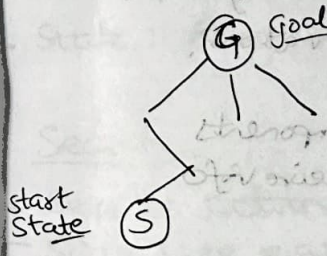
Knowledge

- Declarative → Based on facts & questions on it
- Procedural → Based on 'How' → Needs a fact to reach to sol ∴ → embedded with declarative.

planning

Constructing a sequence of actions that achieves a given set of goals.

eg. LISP, PROLOG → programming lang; goal is given, have to reach to start state. (Backward search)
↓
treates everything as lists



AI planning

Tackles uncertainty in planning that could lead to non achievement of goals.

Predicate → Quantifiable; Relational

Proposition → Yes/No → not simultaneously

Modifying the environment involves removing obstacles.

Forms of knowledge
- Empiricism
- Scientific
- Intuitions
- Authoritative

Logical Reasoning
- Deduction: From given facts infer another fact
- Abduction: The another fact deduced is possibly true from a given set of facts.
- Induction: Involves generation of rule from facts & observations.

AI divides the system into 2 components
- Agent : perform action / perceive info.
- Environment.

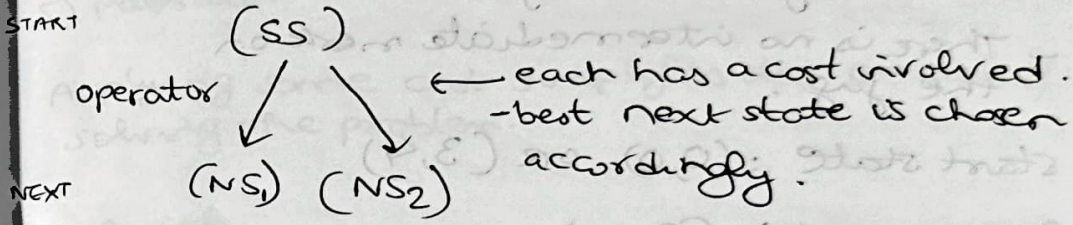
Artificial Narrow Intelligence (Weak AI)
Artificial General Intelligence (Strong AI)
Artificial Super Intelligence (Hypothetical)

Problem Solving

Lec 2

I/P : Set of states
Operator
Start state

$(S_1, S_2, S_3, \ldots)$



Problem Representation
- Broad, generalisation approach to solve a large class of problems.
- State space search ⇒ to model the problem
- State : Any configuration (valid).

Search
- Several actions possible from SS to NS (many)
- Simulate each sequence of actions to reach the goal
- Calculate the cost of each of these paths from goal to start state.
- Finding such a best sequence is called search. (problem-solver does this)

⇒ If action is not driven by objective fn, it may fail to reach the goal, like infinite loop.

Eg:- Water jug problem

(x,y)

- Two jugs contain x & y units of water.
- Let J1 allow 3 gallons & J2 allows 4 gallons at full
- There is no intermediate marking on the jug.

Start state : (0,0) or (3,4)

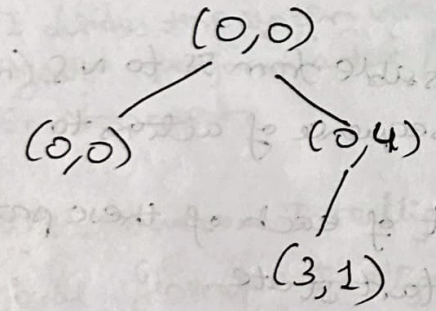
Suppose goal : (2,y)

Not measurable due to lack of marking.

Let start state = (0,0)

Intermediate states : (3,0), (0,4)

Goal state : (2,y)



actions

1) fill
2) empty
3) pour from one to another.

Lab: one such standard problem.

Problem solver

1) Initial state
2) Set of Actions
3) Successor fn → Explicit / Implicit.
4) Goal test
5) Path cost

Applying some cost simplifies the process of solving the problem.

Search features:-

Completeness
Time complexity
Space complexity
Optimality

In BFS, time complexity is more
In DFS, space complexity is less, but it is not guaranteed to find the solution.

Uninformed Search

Problem Formulation

→ Configuration/state: Representation like (x, y) for water jug problem.
→ Constraints
→ Rules: Actions to be performed like fill or empty
→ Initial state
→ Goal state.
→ Implicit state: Intermediate.
→ Valid sol^{n}
→ General Algorithms → all algos of trees are applicable to search

→ Problem of implementation in memory (IV) and time.
→ Incompleteness: Sol^{n} possible so whether it can be achieved

Eg:- If path cost is given as a constraint, the problem gets more difficult.
⇒ Change search problem

→ Uncertainty: Explicitly not mentioned, vague ~~constraints~~ representations.
Eg: Self-driving car (in adverse weather conditions)
Try to remove most uncertainity → hence most problems involve approximation.

Searching

→ Search tree :- Possible action sequences starting from initial sequence
→ Unexpanded node ← Node that has not been verified → needs to be expanded.
→ Set of actions are applied to current state to generate a new set of states.

Strategies

→ Uninformed
→ BFS
→ Uniform cost
→ DFS
→ Depth limited.

[Game problem is irreversible]

Disadv of BFS (Application)

→ Redundant paths can't be avoided & actions are reversible (in most cases)

→ Frontier → ~~set~~ set of leaf nodes available for expansion at any given point.

Disadv can be combatted through maintaining a data structure which remembers the explored paths.

Process

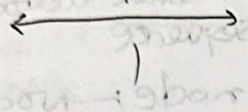
Generated → Expanded → Examined → Discarded

Graph Search

Issue: child may have multiple parents.
State space graph is divided into explored and not explored.

In graph

Black → explored
White → frontier
grey → unexplored



Systematical examination is necessary to avoid exploration issues

Combinatorial explosion :- By checking all the paths it is difficult to find a solution.

Eg :- Travelling Salesman problem with large no. of cities.

For each node (attributes)

- State
- parent
- action
- path cost

In uninformed search there is non-variable path cost, in general.

→ Inclusion of time introduces constraints to the problem.

Node & state (diff)

Eg: Airport → can be a node
Physical location → state

Node is a representation.

Features of AI (slide 19)

- → Graph is represented through initial state, actions, transition model (it is infinite)
- → complexity is expressed through:
  - b → branching factor : 2 for BFS.
  - d → depth of shallowest node.
  - m → max. length of any path.

Shallow → General
Depth → Specific

Explored : $O(b^{d-1})$

~~Unexplored : $O(b^d)$~~

Frontier : $O(b^d)$

Space complexity : $O(b^d)$

Issue at depth d, takes a long time for debugging / finding out

> It is exponential search problem → hence it is very rarely able to solve uninformed problem.

> Since we are exploring each layer at a time, it is complete (though memory & ~~time~~ space complexities would be high)

- Why is the term $b(b^d - 1)$ added to time complexity?

Ans) Because time complexity takes worst case sol ∩.

If we find the goal at a depth it needs to be expanded and then needs to found,

So this expansion accounts for the extra term after b^d in true complexity

BFS is optimal as we systematically explore the nodes layer-by-layer. Hence Soln is always obtained & it is a complete problem.

> BFS does not involve any cost

Uniform cost search

- Nodes closer to the goal are assigned lower cost.
- Find path where cumulative sum of costs is least.
- No. of steps is not considered, but the total cost is adjudged.

properties

→ Infinite loop may be created if there exists path(s) with zero cost, equivalent to No Op. → some min cost must thence be provided. This guarantees completeness.

→ Search is guided by cost rather than depth (b & d)

$O(b^{1+\lfloor C/\epsilon \rfloor})$; C → cost of optimal soln

if all costs are equal, $d = 1 + \lfloor C/\epsilon \rfloor$; $\epsilon \leftarrow$ min cost.

Uniform cost does more work as it checks all paths from S to G.
S → start ; G → goal

prob

Time Complexity

slide 28 $b = 2$

path : A B G

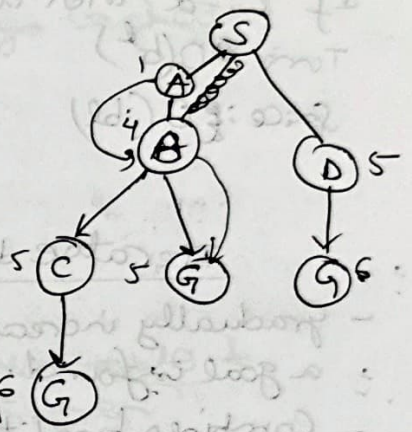
path cost = 1 + 4 + 5 = 10.

$\epsilon = 1$

T.C. $= O(b^{1 + \lfloor C/\epsilon \rfloor})$

$= O(b^{1 + \lfloor 10/1 \rfloor})$

$= O(2^{11}) \simeq O(2048)$

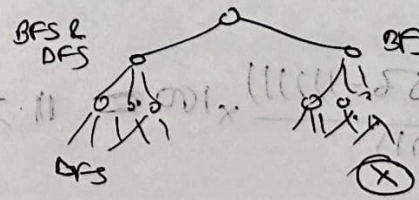


DFS

- If any rule becomes redundant, backtracking is done → hence it is not optimal.
- Depends on choice of depth

Time: $O(b^m)$ ← if solutions are deep, it is faster than BFS. ($m > d$?!)

Space: $O(bm)$ ← linear



Depth-limited search (DLS)

- DFS with limit l
- Tackles drawback of infinite path.

If $l != d$ then it is not complete.

Time: $O(b^l)$

Space: $O(bl)$

Iterative deepening (IDS)

- Gradually increasing the limit, until a goal is found.
- Combines benefits of BFS & DFS.

memory: $O(bd)$

EXAM ✱ Slide 47

No. of nodes generated in DLS :-

$$N_{DLS} = b^0 + b^1 + \cdots + b^{d-1} + b^d$$

$$N_{IDS} = (d+1)\,b^0 + d\,b^1 + (d-1)\,b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1\,b^d$$

$b = 10, d = 5$

$N_{DLS} = 1 + 10 + 100 + 1000 + 10000 + 100000$
$= 111111$

$N_{IDS} = 6 + 50 + 400 + 3000 + 20000 + 100000$
$= 123456$

$$\text{Overhead} = \frac{123456 - 111111}{111111} \times 100 \approx 11\ \%$$

based on b & d (2 parameters) } numerical exam (4)

DLS vs DFS

DLS vs IDS

Q) If b is more, which algo to choose?
If d is less, which algo to choose?

IDS

Time $= O(b^d)$

Space $= O(bd)$

» In exam, create comparative table & if values of b & d are given then compute values.

Informed search

imp

Heuristic function → Based on knowledge we have to develop heuristic function.

Randomly without information → uninformed search.

Heuristic fn, if not designed properly, may fail to find the sol^n due to limited knowledge. (mis-utilisation of knowledge)

Provides an estimate of solution cost.

→ Exploring the path by exploiting the knowledge.

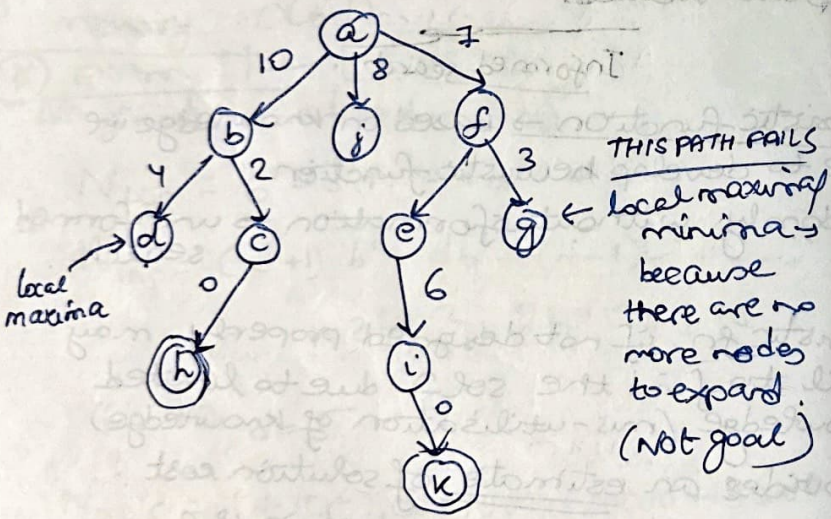
Hill climbing Algo

→ Generate & Test Algo: Generate a fn → if test fails then generate another.
→ from initial state use heuristic fn to generate another state.

f(start) ----→ f(n)

if f(n) < f(start) then expand and check if f(n) is the goal state.

At goal state, value of heuristic function is zero.



Disadv

→ Since we are expanding only one node it may get trapped at local maxima.

Gradient Descent (Steepest Hill climbing)

As opposed to simple hill climbing, it checks all successor states.

At a local maxima/minima, every move around it produces worse results (Trapped).

Soln

→ Give up
→ move from the trap through a big jump → in order to find another place where slope is available. Jump means to randomly select another step (which may be trapped again).

In a valley (plateau) → flat line → no slope is available & the algo does not move.

Ridge → multiple local maxima → multiple steps are required to jump from a local maxima/minima.

Soln

→ Backtrack to earlier node in diff-direction.

Block world problem

if a block is above another desired block as per goal state then +1
for wrong block it is -1

Since we are only considering block below it, we are using local heuristic.

This may lead to many wrong goal states.

due to incorrect heuristic function selection.
- Changing the heuristic function to using global info, then it gives better path with unique states.

Simulated Annealing. (SA)

Depends on how a metal is heated & based on its properties it reaches maximum energy level.

Here, heat → knowledge.
After taking it to highest energy level, it is cooled to make it crystallised.
This is the philosophy behind SA.

The cooling down (i.e. change of energy) is done probabilistically to the point where no more change is possible (global minima)
= [Controlled Randomness]
It enables more thorough exploration

From start state (top), the downward move is made to some state, before the randomness is applied.
→ Layer wise search in controlled manner.

If change of weight is high, it may overshoot.
If it is low, trieteration is more, but chance of overshoot is less.
Accordingly, energy is modified in such a controlled manner.

$p = e^{-\Delta E/T}$: probability for transition to higher state.

ΔE → change in energy
T → Temperature (Random) → based on annealing schedule

Generate random no. (or) between $[0, 1]$. if $r < p$, then proceed to change.

Best-first search (OR graph)

why?
⇒ Each path is equally important.

The nodes that were not expanded in a previous iteration remains in contention for expansion in the subsequent iterations. (the node having lowest value of heuristic fn is expanded).
→ Combination of DFS and BFS.

Make 2 lists: open & closed.
↑ already examined.

Evaluation function

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

$g(n)$ = exact cost of cheapest path ~~to~~ so far

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

$f(n)$ = estimated total cost of cheapest path

At goal state $h(n) = 0$

In UCS → path cost is considered, $h(n) = 0$.

In Greedy (best first), $g(n) = 0$.
path cost is not important

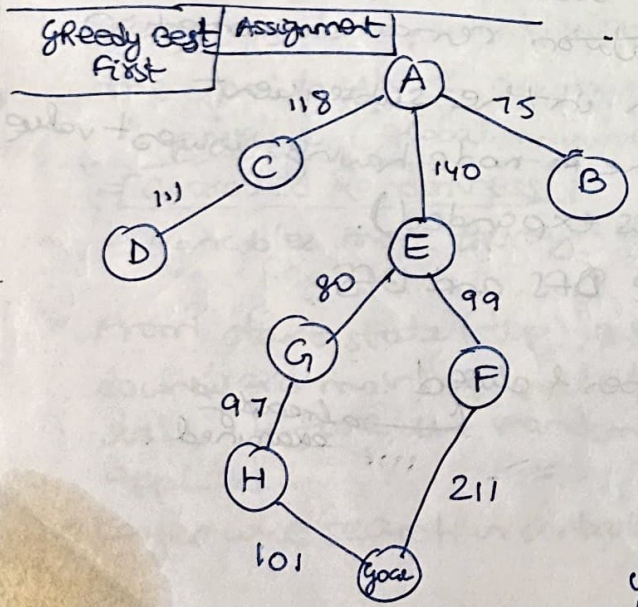
Admissible property → it never overestimates or underestimates the path cost.

Greedy cost first

Time: $O(b^m)$

Space: $O(b^m)$

A**



State	h(n)
A	366
B	374
C	329
D	244
E	253
F	178
G	193
H	98
I (goal)	0

$f(n) = h(n)$ ≥ stands for that the cost

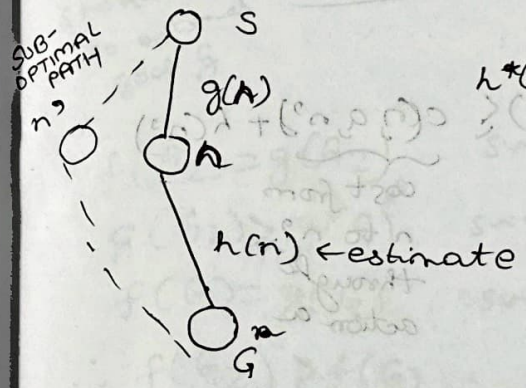
A* algorithm

17/01/2025

Used for Robot Path planning.

Considers the best of both Djikstra's and best first search (greedy).

Measures
→ Distance from start
→ Smart estimate of remaining distance to goal



Graph search algo

Open List : Nodes to be eval. Sorted by f(n) in asc.

Closed List : ~~Goal~~ Already eval. nodes.

If no path exists the reconstruct the heuristic algorithm.

PROPERTIES

I. Admissibility property → Never overestimate the cost to reach the goal

h(n) is an estimate.

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

↑ exact cost to reach n

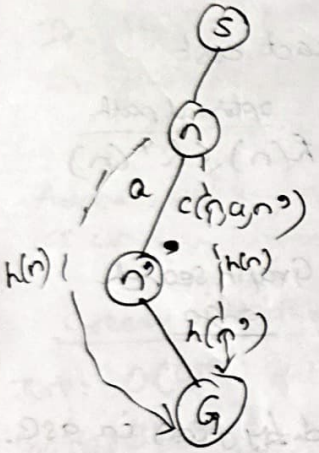
↳ estimated cost to goal from n.

G → Goal state.

Admissible

1. $h(n) \leq h^*(n)$
2. $h(n) \geq 0$ & $h(G) = 0$
3. $h_{SLD}(n)$ ← straight line distance
   ← minimum distance (no overestimation)

II. Consistency/Monotonicity



$h(n) \leq \underbrace{c(n, a, n')}_{\text{cost from } n \text{ to } n' \text{ through action } a} + h(n')$

proof ✍ using triangular inequality

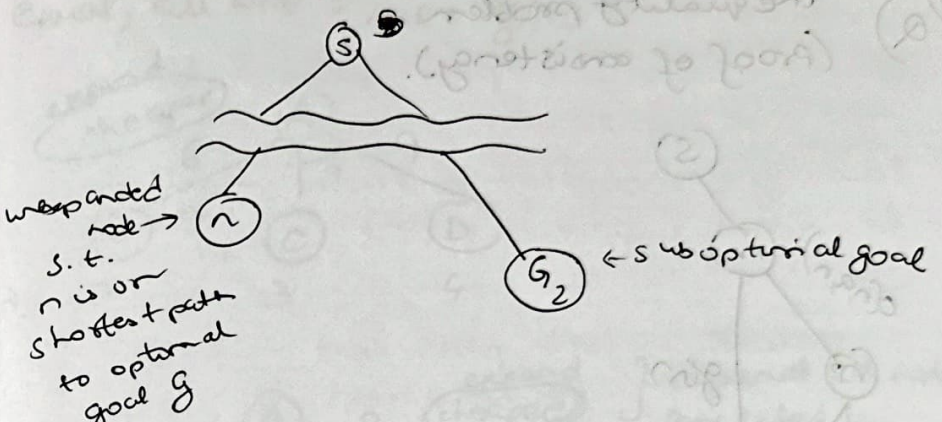
Start state has zero path cost.

All nodes remain in contention for expansion

Q) In A* is 418 or 450 the suboptimal solution? (in TSP)

HW

Optimality of A* (proof)



$f(G_2) = g(G_2)$ since $h(G_2) = 0$

$g(G_2) > g(G)$ since G_2 is suboptimal

$f(G) = g(G)$ since $h(G) = 0$

$\therefore f(G_2) > f(G)$

Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion.

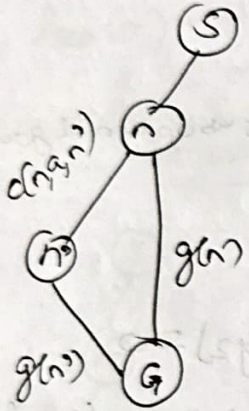
$f(G_2) > f(G)$

$h(n) \leq h^*(n)$ — since h is admissible.

$g(n) + h(n) \leq g(n) + h^*(n)$

$f(n) \leq f(G)$

(HW) Q) >> Consistency proof using inequality problem. (Proof of consistency). Triangular

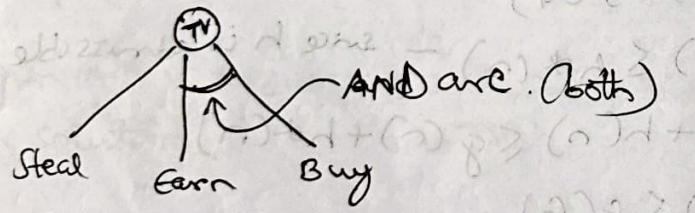


AO* (AND-OR) Search

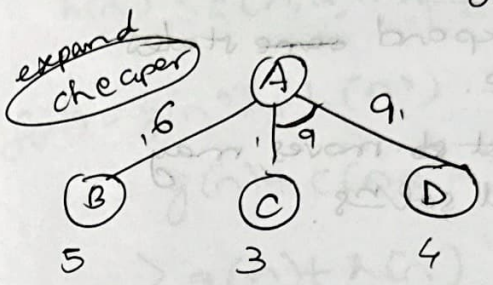
- used for problem reduction.
- Informed search algo.

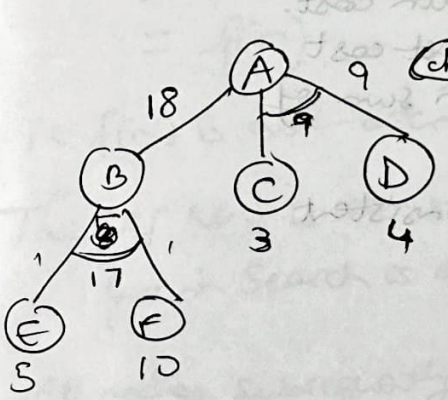
AND OR graph

AND → all probs to be solved.
OR → any prob to be solved



In example, slide 63, path costs are equal, all are 1, ie. g(n) is 1.





} expand all nodes connected by AND arc together.

Expansion of path takes place of lower weights at root nodes which have been propagated upwards.

Adv
- Complete
- No inf loop
- less memory

Disadv
- Not optimal as it doesn't explore paths when it finds a sol?

A* algo

Problem of repeated states

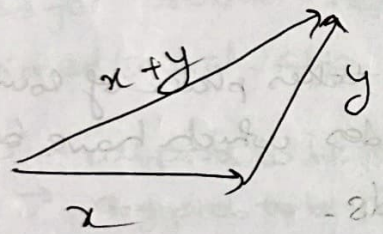
we don't want to expand ~~same~~ states that are repetitive.

Pruning of redundant ~~sb~~ moves, may lead to suboptimal sol^ns.

Sol^n

- To assume uniform path cost.
- Select path having least cost.
- To avoid duplicacies in sum of $g(n) + h(n)$.

→ ensure heuristic is consistent

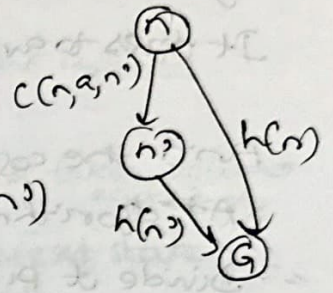


More no. of parameters increases flexibility. (features to extract)

Lesser no. of parameters means lesser no. of paths and more expensive.

Consistent heuristics

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



$$f(n') = g(n') + h(n')$$
$$= g(n) + c(n,a,n') + h(n')$$
$$\geq g(n) + h(n)$$
$$= f(n).$$

i.e- $f(n)$ is non-decreasing along any path.

Th: If $h(n)$ is consistent, A* using Graph Search is optimal.

All nodes subsequently form contours increasing f value → expansion.

A*

Complete

Time : $O(b^d)$

Space : All nodes in memory.

Optimal : yes

e) y vs x, Robot path planning problem.

It tries to avoid obstacles

Find the cost using traditional A* algorithm.

Divide it piecewise

Draw it in a grid structure.

Each grid is a state.

Check obstacles in each state.

Take variable heuristic fn with variable parameter.

Agent and Environment

Agent → Human / Robot

Environment → consisting of many states and obstacles thru which agents learn

- There will be a feedback to agent to follow a path
- Do not need huge amt of data

Calculate 2 signals
→ Reward
→ Penalty
} Feedback signal

Intelligent Agents

→ Rationality

→ Difficulty nature of environment affects behaviour of agent.

→ Agent percieves entire sequence & takes action accordingly

→ Markov decision process (MDP) :— Next state depends on previous state only.

~~Evaluate~~

Multiple actions can lead to a state.

Evaluation of best from those actions.

Agent → sensor
→ actuator.

Agent for

$f : P^* \rightarrow A$

P* → percept histories

A → actions.

Agent = Arch + Prog.

Arch → actuator, sensors

Prog → I/p from sensors & direct action.

State of env changes thru action.

Mapping a state to action → Condition–Action Rule

lookup table → sequence & ~~the~~ actions.

Vacuum cleaner

State : [loc, condition]

eg [A, dirty]

Rule

If (Condition) then (Action)
↑ percept

↑ from perception through sensors like camera, etc.

Rationality

Taking action at the right point of time

→ Reasonable: Reasoning power
→ Sensible: Able to explain it
→ Good sense of judgement.

Performance measured through objective fn.
(Eg: amount of dirt cleaned up OR amt of time taken / electricity consumption) noise generated for the machine.

performance P is a fn of various params
$P = f(a, b, c, d)$
Need to learn the parameters to maximise performance.

percept sequence → previous short term, long term.

~~Collection~~
Collections of many rules is called knowledge.

performance is expected, not absolute.
Enhanced through perfection.

→ Knowledge base & sensor to perceive.

Application of Rationality

PEAS: Performance measure, Env, Actuator, Sensors

Environment → Surrounding conditions
Actuator → takes actions
Sensors → for perceiving
Performance measure → obtained through learning

exam Q) A problem can be given: Design the PEAS and knowledge base for that case.

Q) Env types

→ Fully observable: Nothing else needed to take optimal decision.
→ Partially observable: → can take internal rep. of state thru knowledge base.

→ Deterministic: current state affects next state.
→ Stochastic: Current state doesn't fully influence next state.
→ Strategic: Depends on actions of other agents
Collaborative & Competitive. (Games)
eg: Robo-soccer eg: chess

→ Episodic: like AND-OR graph
↳ Can apply searching algo.

A sequential setting (episode) depends on prev episodes / seq. steps

Eg) seq-setting → word
episode → sentence

⇒ Multi agent systems → how agents collaborate?

→ Static: Env remains unchanged
→ Dynamic: keeps track of changes
→ Semi-dynamic → Env doesn't change
→ Agent perf. score changes

→ Discrete → No. of actions & states is finite
→ Continuous → total actions cannot be perceived.

→ Single-Agent: One agent
→ Multi-Agent: perf. of one agent affects the other.

⇒ Taxi-driving is none of the env. types as mentioned in the slides.

Agent fns

Percept seq. —maps→ Action.

Rule: If (Percepts) Then (Action)

Agent fn is designed as a Lookup table.
→ states & actions specified in advance by designer.

Drawbacks of table
→ Cont. env. makes huge table
→ long time to build & learn

→ iid → Identically Independently Distributed

All samples are independent & need not be learned in sequence.
Identical features are present for each sample.

Disadv of lookup table

P ← set of possible percepts
T ← lifetime of agent (total no. of percepts to be received)

$$\sum_{t=1}^{T} |P|^t \leftarrow \text{no of entries}$$

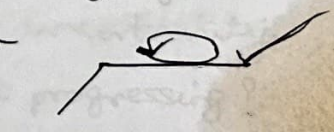
exam

Types of Agent → Acts only on current percept, no percept history

→ Simple Reflex: Not model based. Reinforcement.
→ model based Reflex
→ Goal based
→ Utility based } clubbed together
→ Learning agents.
for sequence agents we use (deep) reinforcement learning.

Types of Agents (based on user):
- Human
- Robot
- Software

→ Simple in partially observable env creates infinite loops. (unavoidable)
Make a big jump or backtrack any state → if stuck in loop at plateau



Not Data-driven approach

*Why do we need huge amount of data? What will machine obtain/learn from data?

-SVM, Bayesian
↳ not much stochastic; mathematics based

more data gives better performance through non-redundant approach

More data leads to more variability (through more input)

Same i/p or less no. of i/p doesn't induce variability.

Reinforcement learning does not require huge amt of data. It learns the env & how to change state based on current position.

Deep Reinforcement learning → ~~many~~
Actions are restricted.
Many states allows learning of different actions.

Knowledge base involves condition-action rules.

Disadv of simple
→ very limited intelligence.
→ Response to changes in env
→ Fully observable.

Eg: Generate rules for self driving car (PEAS)

Model-Based Reflex agent

→ guided by a model
→ Maintains internal state → representation of unobserved aspects of current state depending on percept history.

Depends on:
→ change of env on its own
→ Change of env through actions / ~~actors~~ agents.

Eg: ⇒ Rules for "driving back home" (model based)

Goal-based agent

→ goal-driver; previous ones only based on state
→ objective: Reduce distance from goal.
↳ Searching and planning.
→ ~~Search~~ perf. measure thru reward & penalty

⇒ What happens and is it nearer to goal?

Eg: Not only applying brake but checking light also.
→ Setting of goal in addition to previous.
→ Reasoning under uncertainty.

Utility Based Agent (Also goal based)

- Mapping to real no. (associated degree of happiness) is a utility function.
- Multi Object optimisation (MOOp)
- Everything is approximated under uncertainty.

Eg: Diff b/w types of agent & how is it progressing?

learning agent

→ learn from past experiences; not in prev.

components

→ learning elenet → Learning
→ Critic → feedback
→ Perf. elenet → select action
→ Prob Generator: → suggestive actions

Multi Agent System (MAS)

→ multiple interacting agents
→ multiple knowledge bases
→ common goal / objective.

Eg: Game theory, MAS agent-based modelling.
↓
same board, agents diff.

Hierarchal agents

eg: Robo soccer.
Position of robots fixed.
just differs in position from goal → closer or farther.

Artificial Intelligence

→ To test S/W prog is working or not.
→ Program vs art. int. is checked against human evaluation.
⇒ Through Turing Test (Generative Adversary Model GAN)

Turing test.

Check b/w human & machine response & if it can be determined separately.

25/01/2025

Planning

- Read to reach a particular destination.
- Finding best route: Tasks at particular time & why.
- Logical side; Domain independent.
- Sequence of actions; next action depends on prev. one based on pre-conditions.

Types → BSSP
Types → FSSP

FSSP
Forward State Space Planning (Progression)
$S \rightarrow S' \rightarrow \dots \rightarrow G$.
- Large branching factor.
- Sound Algorithm.

BSSP
Backward.
$G \rightarrow \dots \rightarrow S$ (Regression)
- Check subgoals for consistency.
- Small branching factor
- Not sound algorithm

Stack planning → (bidirectional search Algo) → combines BSSP & FSSP
→ Planning system

Eg: Sussman anomaly.
[Block problem]

Interleaving → whether to go back a step at some iteration, i.e. undo the operation.

Closer to goal state so that loop / undo operation is minimised.

Precondition for block transfer, both need to be clear above.

- Planning → such that subtasks are in sequence and linked & maintain optimal cost.
- Intermediate moves follow a specific sequence, not randomised.

Partial-order planning

→ Delay during search
→ sequence to be established.

In sock & shoe planning, sock on either leg is independent but shoe on a particular leg is dependent on sock on that leg beforehand.
The parallel plans are non-commitment on which comes first.

Target stack plan

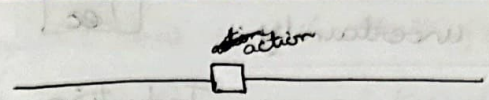
→ Knowledge base req'd for current situation and actions.
→ Planning Graph: data structures used in automated planning.

○ State level → Nodes with logical propositions
□ Action → Influences state levels.

Edges → State node to Action
Arcs → Action to state Node.

– Mutex Relationship ⇒ cannot coexist simultaneously, reduces complexity.

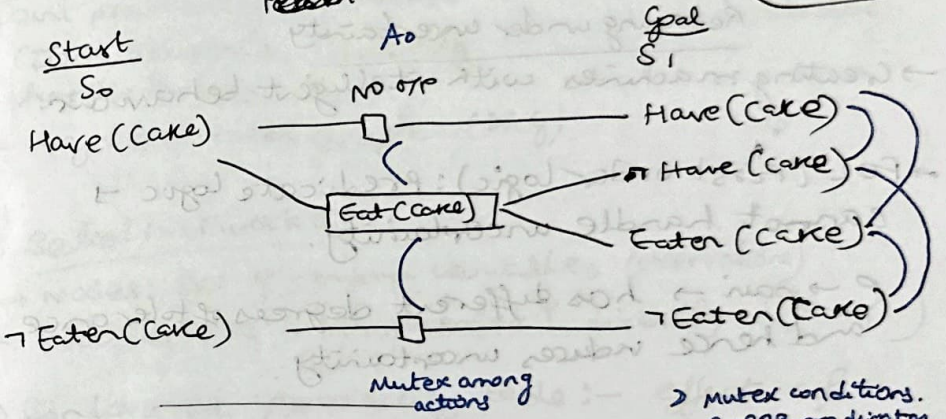
In graph
– many levels
– Level → Actions & literals



Persistence action
[It is carried forward to the next state].

Start: Have(Cake)
Goal: Have(Cake) ∧ Eaten(Cake)

PLANNING GRAPH



→ Mutex conditions among predicates that have been derived.

Mutex Cond

→ Interference
→ Inconsistent effects
→ Competing nodes.

Mutex link defines which literals can't exist together.

Have(Cake) and Eaten(Cake) are mutually exclusive.

→ Generate nodes with further actions till goal literals does not have mutex links. (Termination condition)

Reasoning Uncertainty

Deduction
- Formal logic
- Certain
- Known premise → Conclusion

Induction
- Informal: does not follow some rules of probability
- Uncertain (probabilistic)
- Plausible, believable, probable, reasonable

Reasoning under uncertainty
→ Creating machines with intelligent behaviour.

→ FOL (First order logic): Predicate logic → cannot handle uncertainity.

P → pair → has different degrees of tolerance and hence reduces uncertainity.

Predicting
→ Probabilistic reasoning to determine chance.
→ Cause to effect

Diagnosing
→ Effect to cause
→ Backward
→

Paradigms: Bayesian & Non-Bayesian

Bayesian Approach

$$P(H|E) = \frac{P(H)}{P(E)} P(E|H)$$

Disadv: Variables are uncorrelated (independent)

- Determines chance of truth or falsity of event.

Belief → based on ~~personal~~ knowledge & perception
→ It is a personal value & depends on above 2 parameters. (Subjective value)
→ New evidence affects the belief.
→ Prior belief: prior to new evidence

Bayesian depends on
→ Joint probability distribution & Conditional probability distribution
(Joint occurences, relatively harder to determine)
- Acyclic directed graph (DAG).

Belief Networks
→ Nodes: Set of random variables (observations)
→ Directed links: - X influences Y directly. ($X \to Y$)

→ Conditional probability table: - effects of parents on each node

Joint probability $P(X_i | X_i, X_j, \ldots)$
X_i → prior
$X_i, X_j, \ldots$ → immediate parent(s) in graph (DAG)

→ Only actual variable is given

→ Revise tree until difference is greater than threshold.

Knowledge base
If X then Y with some probability p.
Probability of E is normalised with H and H':

$$P(E) = P(E|H)\,P(H) + P(E|H')\,P(H')$$

propagating belief

28/03/2023

Knowledge-based systems

Knowledge is represented in IF-THEN format
(uncertainty is induced by assigning probability 'p' to the rules).

Abductive reasoning → giving more emphasis on observations.

(E) (H)
IF X then Y. with prob p.

eg: If sneeze then have cold

We need to determine Y without knowing anything about X. (using Bayes' rule), i.e. P(H|E)

$$P(H|E) = P(E|H) \cdot \frac{P(H)}{P(E)}$$

$$P(H|E) = \frac{P(E|H)\,P(H)}{P(E|H)P(H) + P(E|H^c)P(H^c)} \quad ; \text{standardised } = P(E)$$

Propagation of Belief

→ Each piece of evidence affects only one hypothesis
→ In real life, multiple Evidences $(E_1, \ldots E_n)$ affect multiple hypothesis $(H_1, \ldots, H_m)$

$$P(H_i | E_{j_1} \cap E_{j_2} \cap \ldots E_{j_k}) = \frac{P(E_{j_1} \cap E_{j_2} \cap \ldots E_{j_k}) \cdot P(H_i)}{P(E_{j_1} \cap E_{j_2} \cap \ldots E_{j_k})}$$

Posterior probs

Assumptions

→ All hypothesis are mutually exclusive.
→ There is at least one hypothesis
→ Hypothesis are collectively exhaustive
→ Evidences are conditionally independent.
↓
Hence we use Bayesian method

$$P(E_{j_1} \cap E_{j_2} \cap \ldots E_{j_n}) = P(E_{j_1}) \cap P(E_{j_2}) \cap \ldots P(E_{j_n})$$

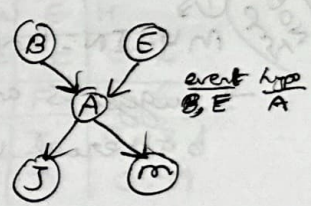
Prior prob of hypothesis given. $P(H_i)$
Conditional prob of matrix given for each evidence. $P(E_i|H_i)$
Single and combined evidence
$P(H_i | E_1 \cap E_2)$: find hypothesis for combination of evidences

Burglar alarm at home problem

Hypothesis → Alarm is sounded @ Harry's house
Evidences → No Burglary
No Earthquake
John & Mary both called Harry

Joint probability distribution.

J → John calls
m → mary calls
A → Alarm sounded.
B → Burglary
E → Earthquake



Draw the network of in question.

$$p[J, m, A, B, E]$$
$$= p[J|m, A, B, E] \cdot p[m, A, B, E]$$
$$= p[J|m, A, B, E] \cdot p[m|A, B, E]\, P[A, B, E]$$
$$\vdots$$
$$\Rightarrow p[J|m, A, B, E]\, P[m|A, B, E]\, P[A|B, E]\, P[B|E]\, P[E]$$

$$P(J \wedge m \wedge A \wedge \neg B \wedge \neg E)$$
$$= P(J|A)\, P(m|A)\, P(A|\neg B \wedge \neg E)\, P(\neg B)\, P(\neg E)$$

parent

→ all values given in table

Combinations of factors for alarm

$P(A|BE)\,P(BE) \cap P(A|B'E)P(B'E) \cap P(A|BE')P(BE') \cap P(A|B'E')P(B'E')$

$P(A'B) = P(A'BE) + P(A'BE')$

Inference from belief n/w
→ cause to effect → Effect to cause

Adv & Disadv of Bayesian
→ Sound theoretical foundation
→ Requires large size of samples to obtain acc. probs.
→ Consistency & Comprehensiveness of values

MYCIN: Uncertainty Handling. (Stanford 1972)
→ Suggests antibiotics (cost) and dosages of bacterial infections

LISP program

- Knowledge Base
- Patient Database (large sample size)

• Product of probabilities not great option

Utility Theory

→ Mathematical framework
→ Determines how AI systems make choices among options based on perceived utility.

{ Objective → Certain
Subjective → depends on perception.

x → outcome
U(x) → fn. x → utility value
P(x) → prob.

Maximum Expected Utility (MEU): optimises the decision-making principle.

- Orderability: if $U(x) > U(y)$, then x preferred over y

(*) exam Q: write short note on 'utility'.

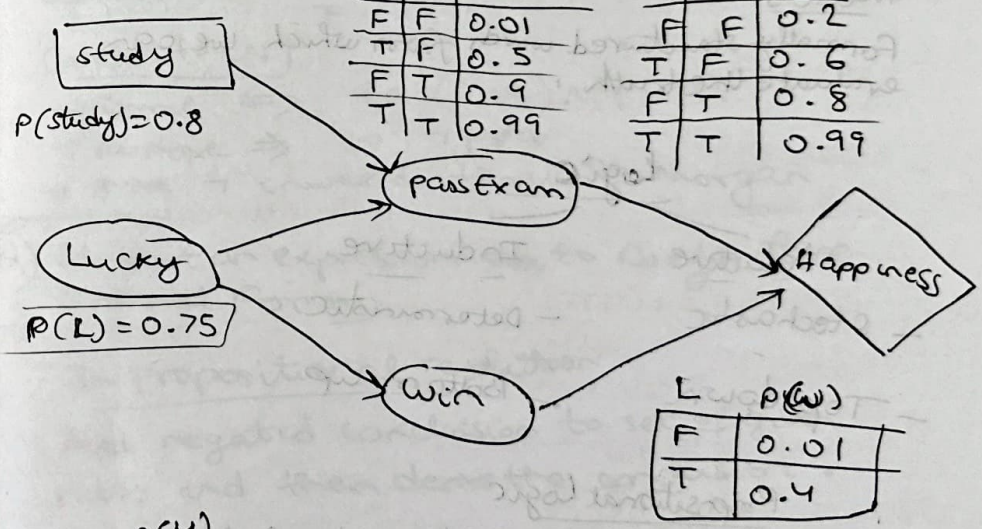
- Transitivity
- Continuity
- Substitutability
- Monotonicity
- Decomposability

(2)

Assignment

L	S	P(E)
F	F	0.01
T	F	0.5
F	T	0.9
T	T	0.99

W	E	H
F	F	0.2
T	F	0.6
F	T	0.8
T	T	0.99



L	P(W)
F	0.01
T	0.4

Find P(H)
Cont.
Probability of happiness for all conditions

Lec

Propositional Logic

Proof or validation behind any reason provided.

Inference engine ← Domain-independent
Knowledge base ← Domain specific

Logic

Formal languages for representing info so that conclusions can be drawn.

Entailment
$KB \models a$ } one thing follows another

- Relations based on sentences based on semantics.

Models

Formally structured words from which we can evaluate the truth.

Logic

- Deductive
- Stochastic
- Top down

Inductive.
- Deterministic
- Bottom up

Propositional Logic

Proposition: declarative statement which is either true or false.

Symbolic logic → because it uses symbols

Tautology → always True
Contradiction → always false

propositions

Atomic
- True/false sentences

Compound
- Combination of many atomic propositions.

$P \rightarrow Q \equiv \neg P \vee Q$

validity → Sentence is true in all models.
Satisfiable → Sentence is true in some model, at least one
Unsatisfiable → Sentence is true in no model

CNF)

Product of Sum:

clauses individually true makes whole true.

Convert to CNF
→ Remove $\Leftrightarrow$ to $(p \rightarrow q) \wedge (q \rightarrow p)$
→ Remove $\Rightarrow$ to $\neg p \vee q$
→ move $\neg$ inwards thru De-Morgan

Q) Convert an expression to Conjunctive Normal Form.

In propositional resolution

Add negated conclusion to set of given rules and then derive the conclusion.

Game Search

31/01/2025

Prev search: - Agent completely controls environment.
- Deterministic
- State doesn't change (unless changed by robot)
- Single path from start to goal.

Game Search – Multi-agent
– Stochastic
– Non-deterministic path.

Every agent considers actions of other agents (Cooperative / Competitive) & optimises its own benefit

> Adversarial search problem → Game Search
- Impact of one agent significantly impacts other agents; in a Multi Agent Environment (MAE)
- At the end, utility values are equal and opposite, for 2 agents with alternating actors.

Eg: DeepBlue by Apple.
- Easy game state
- Limited action rules for agents
- Precise rules define outcomes.

Eg: Chess
- Avg branching factor = 35.
- 50 moves each player, search tree has 35^{100} nodes.

Brute force becomes inefficient here.

Typical Search
→ No opponent

Game Search
→ Unpredictable opponent.

Zero sum → Sum of player & opponent's move is zero.
win → +1, lose → −1, draw → 0.

Eg:- Rock, Paper, Scissors

Sum at each cell is zero, so zero-sum.
If draw, both zero, else +1/−1 or vice versa.

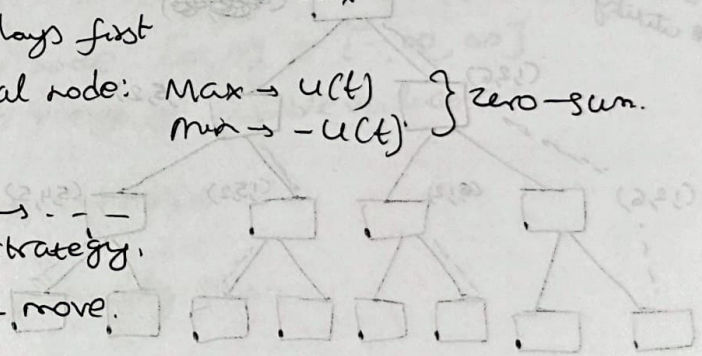
Eg: Battle bots

State-space representation

0,0	10,8
8,10	0,0

Tic-tac-toe

→ 2 players: MAX & MIN
→ MAX wants to maximise utility & vice-versa
→ MAX plays first



a → b, c, d → . . .
Recursive strategy.

Ply: Half-move.

From first max node expand 3 possible min nodes, then moves to terminal. Backtrack using smallest value (MIN) and evaluate.
MAX will choose largest of above obtained smallest values.
Using DFS → This computation takes place.
(Minimax Algorithm)

Minimax Algo –

uses DFS

Depth ← m

Legal moves at each point ← b.

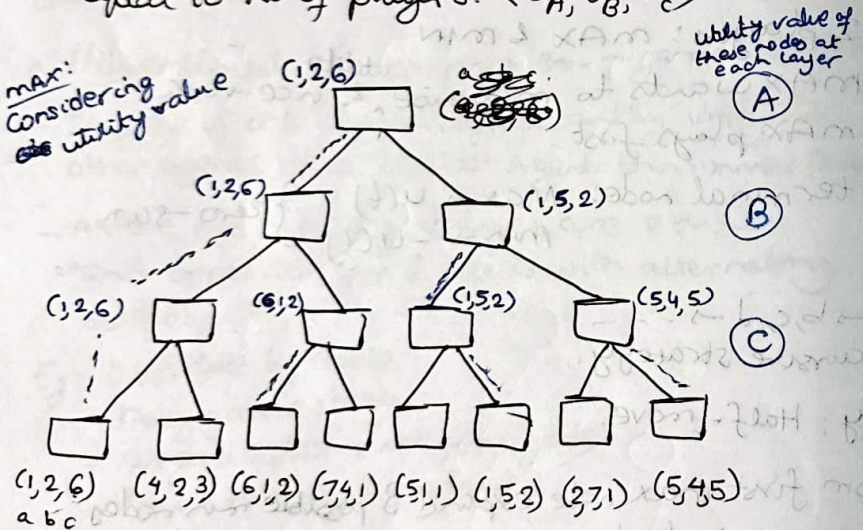
Time complexity: $O(b^m)$.

Space: $O(bm)$ ← all actions together.

: $O(m)$ ← actions one at a time

In multiplayer games

→ Single value of each node is replaced with a vector of values having dimensions equal to no of players. $\langle v_A, v_B, v_C \rangle$



minimax

Complete: Yes (if tree is finite)

optimal: Yes (against optimal opponent)

Time = $O(b^m)$

Space = $O(bm)$

Alpha-Beta Pruning Algo

→ prune branches that need not be expanded again, i.e. need no more examination.

→ For MIN, pruning trees having higher than that value (atmost) ← β cuts

→ For MAX: (atleast) ← α cuts (prune lower values)

After generating value for only some of n's children, we can never reach 'n' in a minmax strategy.

Alpha: highest value found till any point. Starts with $-\infty$

Beta: lowest value choice found at any point. Starts with $+\infty$.

In backtracking node values pass to upper nodes

$\alpha \geq \beta$ for pruning

At starting node $[\alpha, \beta] = [-\infty, \infty]$

This is the range of possible values

Root layer is MAX

Layer 1 → MIN

and alternatingly so on

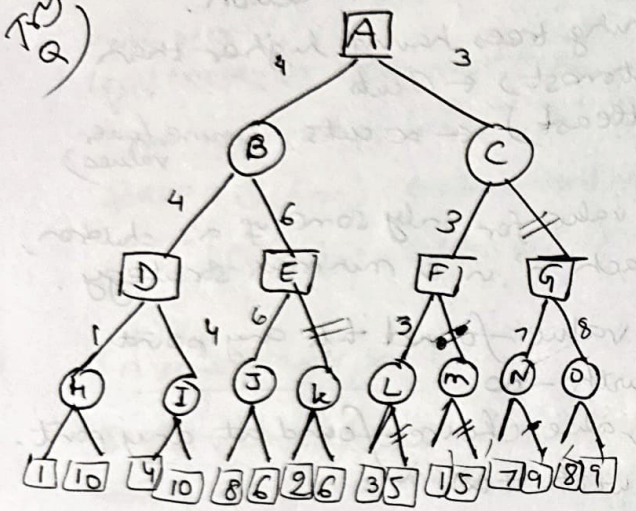
The branch having lower value outside range is pruned and not examined

- minmax decision is independent of the values of the pruned leaves

- Dependent on order in which states are examined.

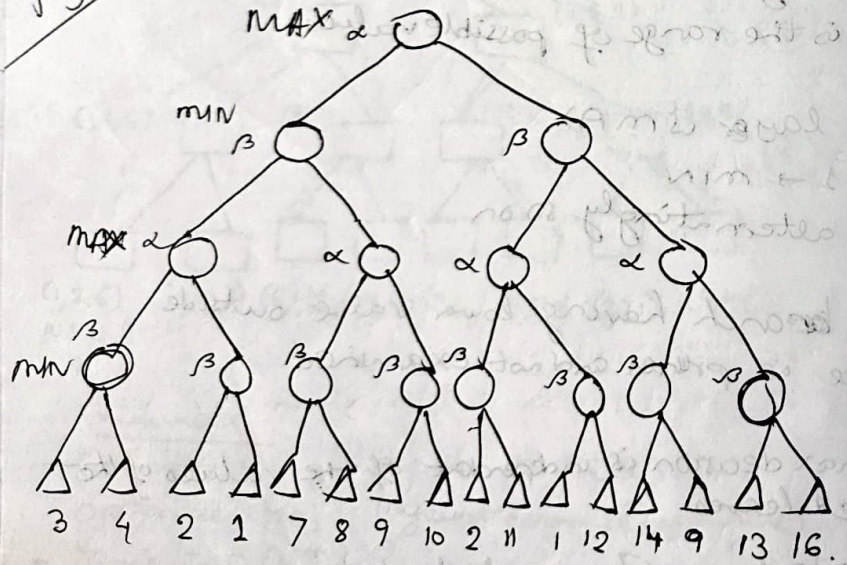
Alpha-Beta-Pruning Procedure

Try Q)



Pruning only affects time complexity; not final result.

Try Q)



Idea

Reinforcement Learning

03/02/2025

Unsupervised learning → No output values, hence no change of weights; no error calculation

Reinforcement learning is a Trial and Error method. Error is modified through experience, i.e. we learn from errors.

Eg) Cycle riding :-

Agent : Person
Environment : Cycle, Road
Error : Falling down.
; Penalty : Pain
; Reward : Compliments from friends & parents

We obtain reward/penalty through error. This is called evaluation. It can be mapped to the cost function of supervised learning.

State : Start state
Action : Paddling
State (next) : Forward / Backward / Falling.

→ State$_1$, Action, State$_2$

Evaluation / Payoff → method of obtaining reward / penalty.

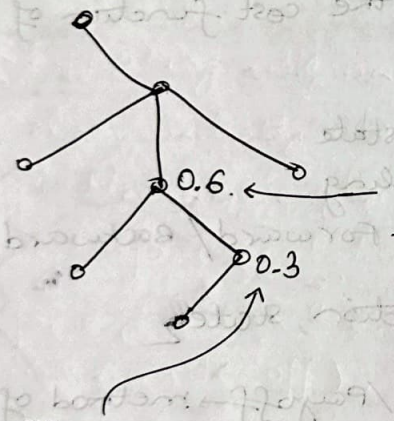
→ • Learning procedure : gathering experience.
→ Not much data
→ Computational resource very less (GPU not reqd) → Hence less energy required.

⇛ YT Lecture:
Prof. Swagatam Das (ISI)

→ Reward obtained through Reinforcement learning is not immediate reward, rather intermediate.

After taking an action, if we determine that a previous action is not good, then we go back to modify the action.

Reward is always expected reward, not exact reward, due to dynamic environment



If prob to expand this node is 0.6 then chance of winning prob becomes 0.6

If prob of winning decreases after a layer, i.e. towards the goal (& there is less scope of adjustment), then the prev. value is updated through experience.

- Rather than immediate reward, it is cumulative reward (or penalty).
- upon reaching a goal state, the path is traced back and learnt.
- Evaluation takes place after a certain outcome.
- Close to the goal we are more confident to win.

Temporal Difference Learning

Temporal → Time-variant

~~Action 'A(t+1)' depends on A_t~~

→ $A_{(t+1)}$ (Action at t+1) depends on A_t (action at t)

→ Prediction at time (t+1) is more accurate than prediction at time t.

→ From feedback based on error, the previous state has to be modified.

Expected Reward

- Non-deterministic / Stochastic / Random

[K-means clustering has initialisation problem due to randomness in initialising]

How to solve randomness

Run the algorithm many times and then take the average.

for expected-reward,
play the game from start to finish
multiple times and take average
reward over each such path.
(since winning is not always guaranteed).

> Exploitation: using the available (repeated) information only for learning. But there is no scope for learning outside that knowledge. It cannot give incremental knowledge. It may lead to missing of some better optimal result/opportunity.

> Exploration: Taking risk and searching (new paths) something else. Initially when knowledge is not available, then exploration is done. After obtaining info, exploitation is done.
The limit for exploration & exploitation is a dilemma (trade-off).

→ If opponent is random, then that creates noise (which creates problem)

→ If opponent is not random, and also plays according to fixed rules and if we are playing greedy to win, then we will be just playing a very very small fraction of the game (tree/graph), without exploration → it is the best choice.

So, exploration may not gives us the immediate benefit.

Eg: Bandit Problem or chess.

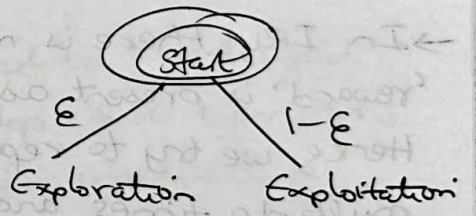
Book > Richardson

Greedy agent can get stuck in sub-optimal state.

Epsilon Greedy Algo

choose random option with probability of epsilon. Set less probability to start exploration, initially, After some information is available, exploitation is done.

Exploration : ϵ
Exploitation : $1-\epsilon$



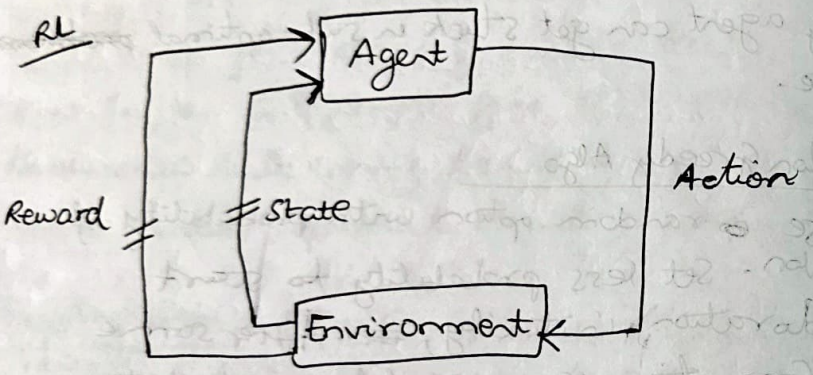
A Reward Signal

(IRL) Immediate Reinforcement Learning

11/02/202

In normal reinforcement learning, there is a delay in obtaining reward/payoff. It takes some time to take decision.

→ In IRL, evaluation happens after performing each action, i.e. there is no delay.



→ In IRL, there is no 'state' input, only 'reward' is present as input.
Hence, we try to repeat the same action multiple times, and it may not always be succesful → hence, the nature of the problem is stochastic.

Since the problem is stochastic, we determine expected reward, not exact reward.

→ At every time (t) instance, pick up an action (a_t) to get reward (R_t)

IRL is a simpler version of RL.

→ It does not have temporal nature
→ Reward is not a scalar evaluation, because evaluation is stochastic
→ Keep trying any action multiple times before figuring out which is the right action.
→ IRL exploits the same action; it does not involve exploration.

Eg:- IRL is useful in Drug Testing
Apply drug-1 to patent-1 and patient-2
Observation: P-1 is cured, P-2 is not cured.
Can it be concluded that D-1 is effective for P-1 and ineffective for P-2?
It cannot, because P-2 may have other symptoms.
It is repeated. D-1 is applied to P-1 again. However, P-1 may develop newer symptoms and the D-1 may render ineffective.
Here, the environment is stochastic in nature
Always applying drug-1 is exploitation. Using drug-2 would mean exploration. But there is a dilemma in the no. of times D-1 and D-2 should be applied.
Patient being cured is a reward.

Stopping exploration may lead to missed info, too much exploitation leads to no new testing.

Formalism

: calculating reward which is stochastic in nature.

set of actions

$A = \{1, 2, \ldots, n\}$ eg:- drug-1, drug-2, ...

Say, Associated with each action there is a coin.

i.e If there are 'n' actions, there are 'n' coins.

- Upon taking an action, we toss a coin (not the same for two actions). The coin is fixed for a particular action.

After tossing, some probability to coming up for Heads, and some probability to coming up for Tails.

Let, for action 1, tossing a coin, if it is Heads, then evaluation (payoff/reward) is 1, and if it is Tails then reward is 0.

Probability Distribution

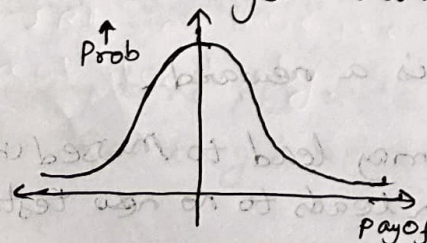
Tossing of a coin is Bernoulli distribution.

Acc to central limit theory, it is Gaussian distribution.

The probability distribution gives the expected reward.

Process

→ Generate a uniform random number; [0,1]



→ Keep integrating the area under the curve, called CDF (Cumulative Distribution Function) gives the probability that the random variable X is less than or equal to the probability.

⇒ when CDF reaches the random no., then the sample is considered.

For instance, let us have four possible outcomes (payoff)

We want to generate one of those 4 outcomes at random. Each of these outcomes have probability P_1, P_2, P_3 and P_4.

Step1: We generate uniform random no.(r); [0,1]

Step2: Check if $r < P_1$, then generate/select Outcome-1

Step3: check if $P_1 < r < P_1 + P_2$

↳ since it is cumulative reward.

Then generate Outcome-2

Step4: check if $P_1 + P_2 < r < P_1 + P_2 + P_3$

Then generate Outcome-3

Step5: check if $P_1 + P_2 + P_3 < r <$ 1

Then generate Outcome-4

Challenge of IRL

- We do not know the distribution (exactly).
  To tackle this, we go for pure learning. i.e. where the mathematical modelling fails, we go for algorithms in iterative forms. (this is same as Machine Learning).
- However, we know that there is some distribution, which we are assuming according to which the payoffs are generated. (i.e. we do not know the parameters of the distribution)
- For this, we assume that there is an expected payoff, denoted by $q^*(a)$

  $q^*(1) \rightarrow P_1$; for the previous example.
  $q^*(2) \rightarrow P_2$, and so on.

  For Gaussian distribution,

  $q^*(1) = \mu_1$

  $q^*(2) = \mu_2$

For IRL, since there is no state, we need to repeat iteratively.

Multi-arm Bandits

14/02/2025

- Bandits are used in slot machines.
- used to understand tradeoff b/w exploration and exploitation.
- Distribution of reward is unknown.
- Slot machine: pull lever, if symbols match then earn money else lose.
- The choice of lever is random and can be picked in any sequence.

Initial findings :- No. of winnings per 100 attempts per machine,
using this knowledge for choice of lever is exploitation, i.e. using same known knowledge, i.e. we are not learning new knowledge.
Hence, we need to go for exploration on the machine having lower win rate,

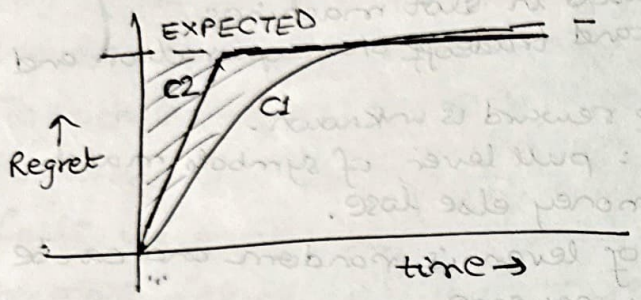
- Feedback → Reward / Penalty
- Stochastic : due to uncertainity in distribution of reward. Hence, we seek for expected reward.
- Selecting action value is unknown, hence estimate reward is computed through sample-average.

Value for at time t for action a

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)} \quad ; \; Q_t(a)$$

Objective : Over repeated action selections for 1000 times, we need to maximise winnings. by concentrating actions on the best lever.

Regret → Diff b/w optimal selection of levels.



Regret for C2 < Regret for C1

⇒ Choose b/w multiple options, each having unknown payoffs. maximise reward over time.

Trade-off
↳ Balancing b/w diff arms exploration to learn about their reward distribution.
↳ Exploitation of known arms already having high rewards.

This is an iid problem (Independently Identically Distributed)

- K independent arms
- Each arm 'i' has reward 'R' from unknown distribution with expected value μ.
- Maximise cumulative reward over T trials.

Naive method.

→ Probability distribution of rewards is diff for each lever.

Conflict

→ Exploration → Improves current knowledge for long term benefit
→ Exploitation → greedy action → exploiting current action-value estimates.

Epsilon-greedy method

→ Balances exploration & exploitation
→ ε ← probability of choosing to explore
1-ε ← prob. of choosing to exploit

Action at time t: { max $Q_t(a)$ @ 1-ε ; any action a @ ε }

Algorithm

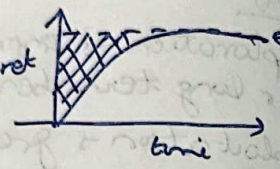
```
p ← random value

if p < ε :
    random action
else :
    current best action
```

If ε = 0, i.e. greedy approach, it leads to saturation due to lack of exploration → gets stuck for suboptimal actions like local optima.

For value of ε: 0.01 performs better than 0.1.
→ 0.1 reaches optimality earlier but winning chance does not exceed 91%.
→ 0.01 increases slowly, but eventually performs better. (i.e. improvement is continuous)

Regret minimisation

Trying to minimise area in graph



Regret = Payoff of possible action − Payoff of action taken.

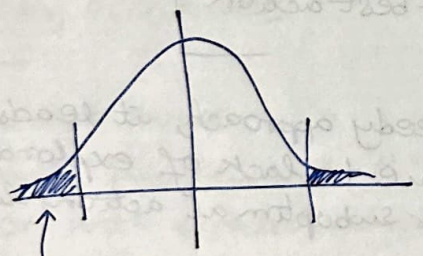
= u(poss. action) − u(action taken)

→ Asymptotic correctness:- Playing infinite times would lead to expected reward.

→ Regret optimality:- knowing best arm apriori and selecting that always.

→ Probably Approximately Correct (PAC) gives lower bound of regret = log T.

Upper Confidence Bound (UCB)



Deviation from mean by this amount is acceptable.

select actions based on:—

→ Estimated rewards

→ Uncertainity / confidence in estimates (to be reduced)

t ← time

a_t → action at time t

y_t → reward at time t

$$UCB_i(t) = \bar{\mu}_i t + \sqrt{\frac{2 \log t}{n_i(t)}}$$

$\bar{\mu}_i t$: mean

$\sqrt{\frac{2 \log t}{n_i(t)}}$: uncertainity through entropy measurement.

Algo

Initialisation:- Pull each arm once to gain preliminary info.

- select arm that maximises UCB value

- update pull count & estimated reward for arm

Estimated reward: average of rewards

$$\bar{\mu}_i(t) = \frac{1}{n_i(t)} \sum_{s=1}^{t} r_{i,s}$$

$r_{i,s}$ → reward on ith arm at sth pull

(to be minimised) ↳ confidence level: $\sqrt{\frac{2 \log t}{n_i(t)}}$

↳ favours arms that have been pulled less no. of times, as they have larger confidence interval.

Adv

→ Theoretical guarantee: provides regret bound
→ Simple & Efficient
→ stationary environment :—

Each slot machine has a program.
The program remains the same during the evaluation process.

If it changes, then it becomes non-stationary.