03/08/2021 Notes – Group 5 (106119004,106119096)

AIML Notes 03-

AI/ML Notes 03/08/2021

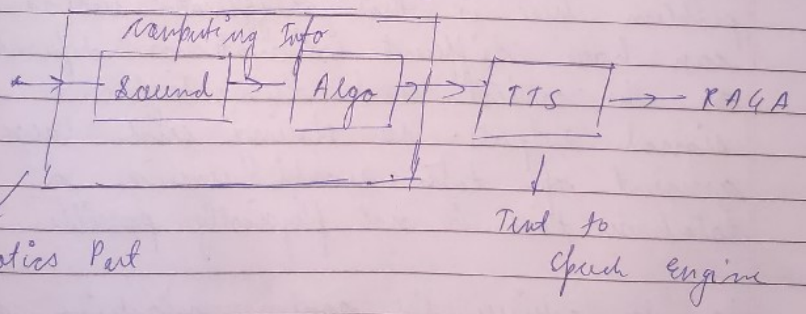
By Group 5

Various Capabilities / disciplines of an Artificial Intelligence (AI) bot

1) Machine Learning
2) Problem Solving (Search, constraint Analysis)
3) NLP (Machine Translation, Speech Recognition)
4) Decision Making (Logic, knowledge, engineering)
5) Reasoning (Probabilistic & Uncertainity quantification)
6) Robotics (Perception, & Actuation)
7) Object Recognition (Vision, Image Processing)

Example

Identify Raga of Music



What is Artificial Intelligence?

According to John McCarthy it is

machine and intelligent computer programs. It is related to similar task of using machines to understand human intelligence but AI does not have to confine itself to methods that are biologically observable.

Example

Text to Speech Engine

One can think of that, we can store various words and their corresponding speech signals in database and for translating a sentence, signals of different words can be merged. But the problem here is that different words can have different intonations depending on the context and thus different signal outputs. So storing such large amount of data would require a database that is not physically possible.

So to achieve the goal, machine are trained continuously and certain rules are constructed. Thus the

intelligence is not only restricted to methods that are biologically observable.

Intelligence

Intelligence is a fuzzy term
It is computational part of the ability to achieve goals in the world.
example
Deep Blue, a computer designed to play chess and defeated Gary Kasparov after learning for long period of time.

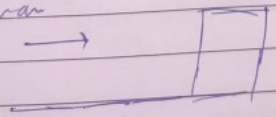
Views of AI fall into 4 categories

1) Acting Humanly

* Turing Test Approach
  - NLP

- Knowledge representation
- Automated Reasoning
- Machine learning

any.



If human pose a question and if it can human is convinced by the answer and not able to differentiate the answer comes from human or machine.

* Computer vision to perceive
* Robotics to manipulate objects

2) Thinking Humanly

Cognitive modelling approach. Combination of Techniques from AI & Psychology.

3) Thinking rationally

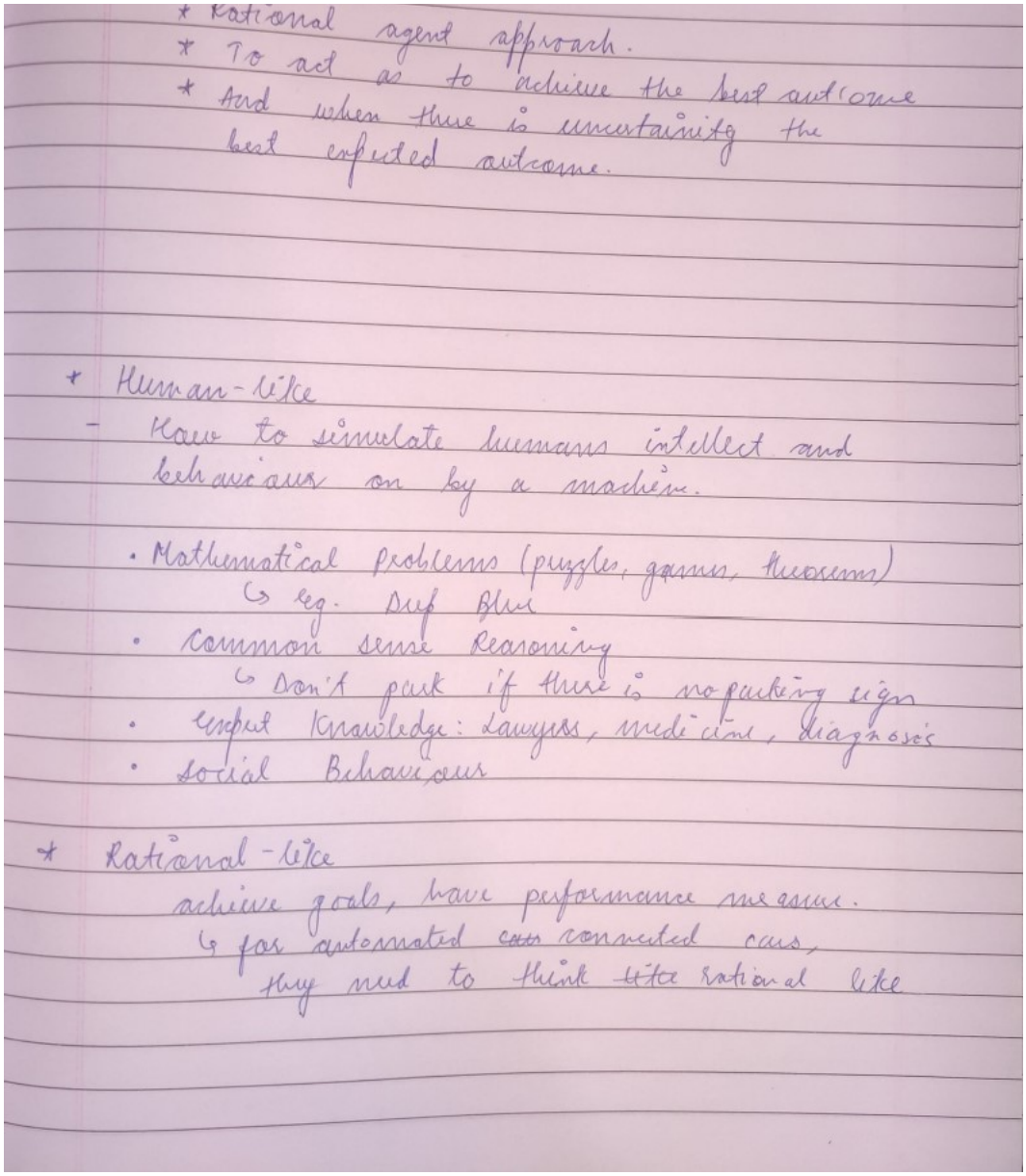
* Laws of thought process.
* Gave way to logic
* Argument structure that yielded correct results based on the premises
* Predicate, Propositional, First Order logic

Ex:

If there is smoke
It can be because of fire
but it can be wrong as
it can be because of dry ice or fog.

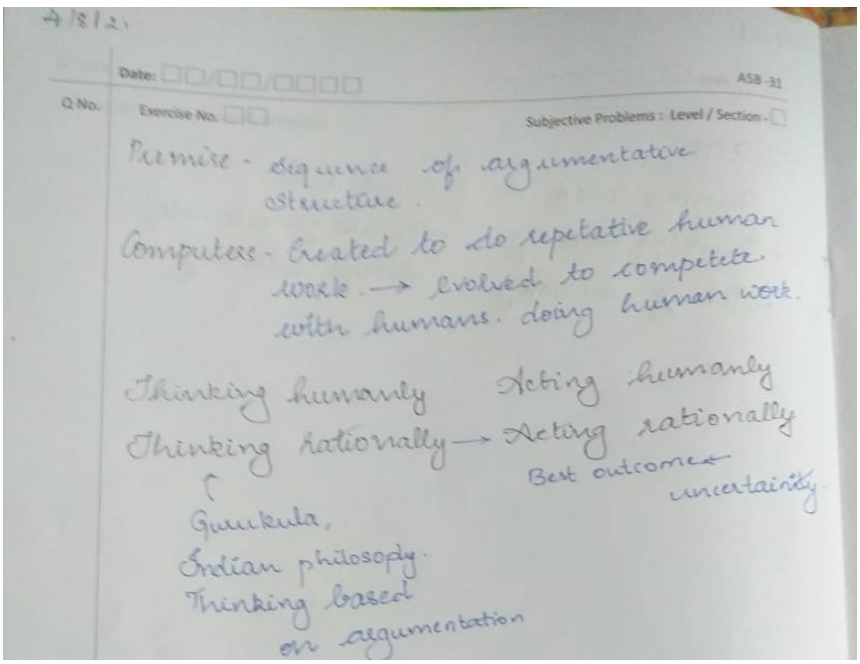
So we need some more input.
and then we can structure the argument.

4) Acting rationally



Speech from one lang to another lang ex is missing though raga example is explained, the robotic speech example is missing (how it is stored - why diphones are preferred over words, phonene or syllable), Lombard effect example is missing. Other than the above examples, the notes has all other topics and proper explanation.
-by 106119120 (Sruthi P) and 106119110 (Samiksha Shekhawat)

04th Aug 2021. Class notes prepared by members of group 2 (106119050(H KAILASH) && 106119052(INDRESH P))
Link to pdf: https://drive.google.com/file/d/1E2otsj7JW6FfR_rKewthhwVpUj15epxg/view?usp=sharing



Thought process & Behaviour.

Turing test

Interrogater —asks Question→ Machine.

Machine —Answers→ Cannot find it was a machine.

(passed turing test).

∴ Thinks like human.

Date: □□/□□/□□□□

Q No. Exercise No. □□ ASB-32

Subjective Problems : Level / Section - □

Requires NL, Knowledge rep., automated reasonning, ML.

Turing —introduced→ Structured programming

Thinking humanly

Cognitive Science — Psycology, AI

Thinking rationally

Logic, Problems.

Acting

- Perceive ← get input from envi.
- Act

Seq of algo → Output to get acted upon.

Perceived ↓ ↑ Converted to represented.

AI examples

- Common Sense reasoning.

Smoke means fire.

fire ⇒ Smoke.

Date: □□/□□/□□□□

Q No. Exercise No. □□ ASB-33

Subjective Problems : Level / Section - □

Tweety is a bird.

All bird fly ⇒ Tweety can fly.

Exception : Tweety ← ~~Ostrich~~.

Inference X.

Yale shooting problem

Initial — Gun is loaded (0) —Fired→ State change (1)

Alive (4) —Shot→ Alive (3).

Designing games.

Search problem.

Target :- Alive (0).

Update vs revise:

A or B → C

Find C rather than A & B.

Training Theories of actions

Looks. like (P) → is (P)

Make-looks like(P) → looks-like (P)

. . . . make-looks-like (P) → is (P)?

Date: □□/□□/□□□□

Q No. Exercise No. □□ ASB-34

Subjective Problems : Level / Section - □

Bench mark

8-puzzle. Automated parking.

History

- Neural network - 1943.
- Wasn't popular - GPU - 15 yrs back (unavailable).
- 1956 - AI. name coined.

Logic theorem

principia mathematica theorem

- Somany examples of history.

1956 - Fathers of AI met.

1966-1974. Problem with computation.

1969 - 1979 - Knowledge base.

Weak vs Strong.

Recent :- DL / GPU architecture

Bayesian belief network - Uncertainity.

Many more in ppt :(

Q No. Exercise No. □□ Subjective Problems : Level / Section - □

State of art

Proverb solves crossword.

AI can't replace humans.

Dynamic aspect missing.

Program :

Tic-tac-toe

- Data structure approach.

1	2	3
4	5	6

7	8	9

Blank - 0 X - 1 O - 2.

19063 X 9 element vector to work.

3^9

- Ternary — Decimal.

Fast in time

Space increasing.

Lot of unnecessary work.

Extend - not possible.

Date: □□/□□/□□□□ ASB -36

Q No. Exercise No. □□ Subjective Problems : Level / Section - □

- Data struct approach - 2

Stored as vector.

- Return middle.
- Returns non-corner.

Manipulating only on one vector.

go() ← To fetch value. to make move.

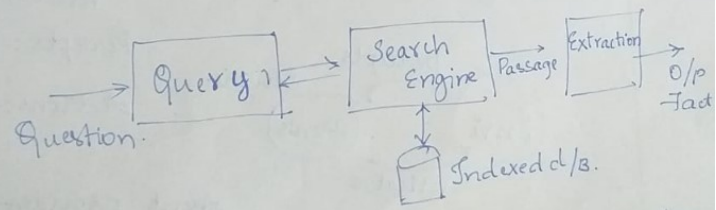
Better than table. (previous).

- State space tree: $\alpha \rightarrow \beta$.

Doesn't make move which will end losing.

Question Answering.

Question — Definition
— Factoid.



Eg:- What did Russia do? — How AI approaches.

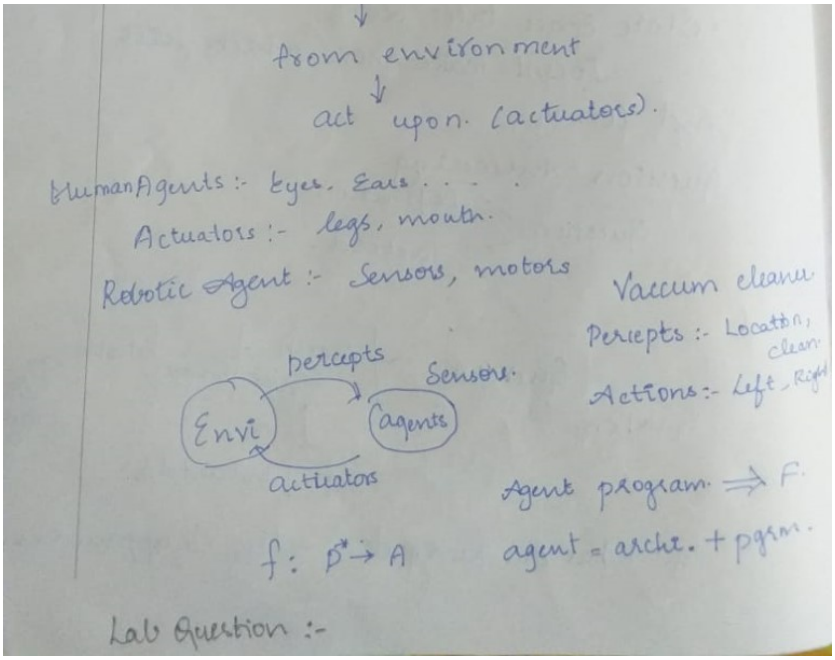
Date: □□/□□/□□□□ ASB -37

Q No. Exercise No. □□ Subjective Problems : Level / Section - □

- Agent & Environments
- Rationality
- PEAS.
- Environment types.
- Agent types → Search.

Agent

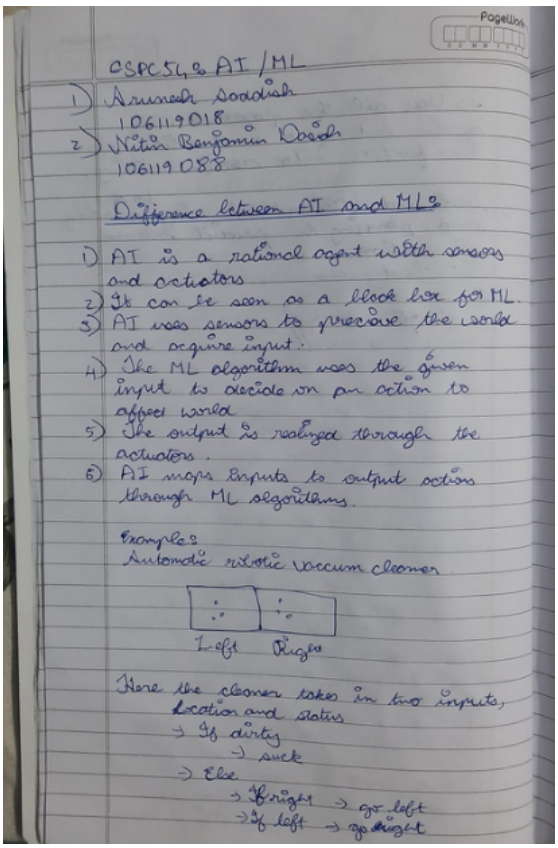
Perceive info. (sensors)



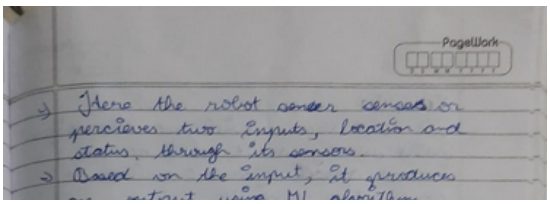
4th August 2021 – Notes Review by Group 6

Ragavi Vijayaragavan- 10611098
Shruthi Kumaravel- 106119116

- Acting rationally and the process could have been elaborated further. Information is perceived and converted to a representation. This information is processed by the algorithm and output is delivered in a decipherable form. An example is the AI to find raga.
- In Yale shooting problem, the underlying concept is that a sequence of actions is used to determine if a goal can be reached, with the help of state transition. This wasn't mentioned.
- Expert Systems was not mentioned. Computer systems designed to make decisions like industry experts are called expert systems.
- Otherwise, the notes are concise and cover all the points.



https://drive.google.com/file/d/1xeSoU7qPy_dNah19pMyU8ec4-q1sU97X/view?usp=sharing



→ The output is carried out through wheels to the sucking agent, i.e.

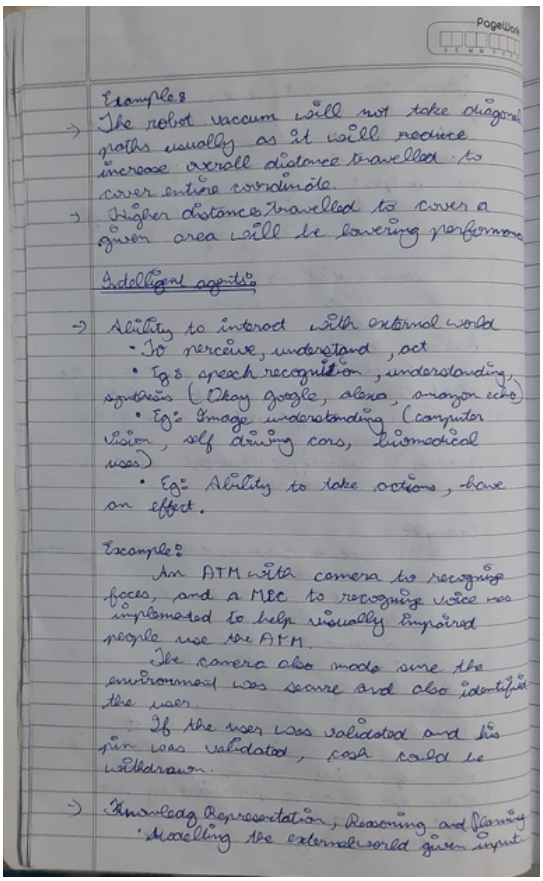
sense dirt → suck
Else → move

Rational agent

→ A rational agent is one who does the right thing even in uncertain situations.
→ It picks the right action causing the most success.
→ Success is usually calculated using a performance measure.

Example:
The vacuum cleaner can use performance measure based on
→ amount of dirt cleaned
→ Energy consumed
→ Distance travelled
→ Even differentiating between useful and useless objects.

Formal definition of rational agent:
A rational agent is one that picks the output that maximize performance, given the evidence provided by the input or percept sequence and the inbuilt knowledge the agent has.



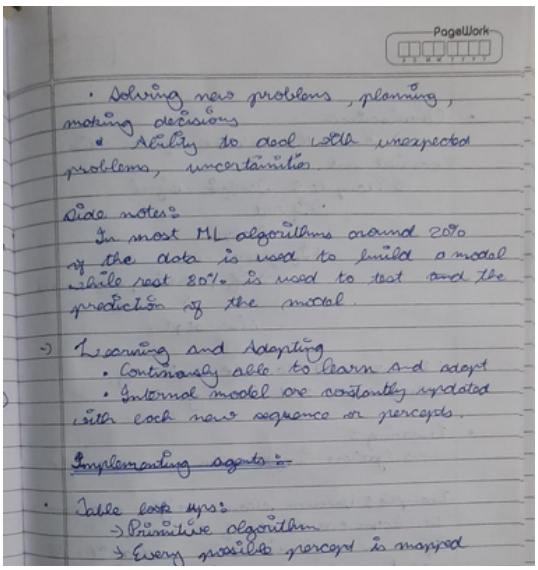
Example:
→ The robot vacuum will not take diagonal paths usually as it will reduce increase overall distance travelled to cover entire coordinate.
→ Higher distance travelled to cover a given area will be lowering performance.

Intelligent agents:

→ Ability to interact with external world
• To perceive, understand, act
• Eg: speech recognition, understanding systems (Okay google, alexa, amazon echo)
• Eg: Image understanding (computer vision, self driving cars, biomedical uses)
• Eg: Ability to take actions, have an effect.

Example:
An ATM with camera to recognize faces, and a MIC to recognize voice was implemented to help visually impaired people use the ATM.
The camera also made sure the environment was secure and also identified the user.
If the user was validated and his pin was validated, cash could be withdrawn.

→ Knowledge Representation, Reasoning and Planning
• Modelling the external world given input



• Solving new problems, planning, making decisions
• Ability to deal with unexpected problems, uncertainties.

Side notes:
In most ML algorithms around 20% of the data is used to build a model while rest 80% is used to test and the prediction of the model.

→ Learning and Adapting
• Continuously able to learn and adapt
• Internal model are constantly updated with each new sequence or percepts.

Implementing agents:-

• Table look ups:
→ Primitive algorithm
→ Every possible percept is mapped

is an output.
→ AI perceives a percept, appends to sequence of percepts, and looks up mapped output on the table and acts on it.
→ Only possible for limited cases of inputs.
Example: Vaccum cleaner example.

- Autonomy:
→ All actions are specified.
→ no need in saving and no autonomy
→ self sufficient

- Omniscience:
→ Knows the outcome of an action carried out
→ Example: Traffic signal
→ If red
→ stop
→ If green
→ If previous person moves
→ move
→ else stop.

We need AI to have autonomy, omniscience, and learning.

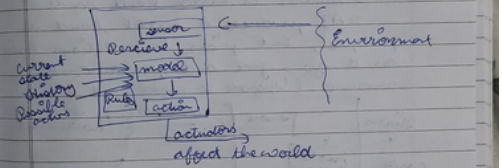
- Learning:
→ Gathers information from percepts

Example: Vaccum cleaner might use camera to learn coordinates of a room.
If a new furniture is added it will adapt and adjust the coordinates to learn.
It decides on actions based on coordinates and percepts.

Reflex agent: (Simple)
An AI that works solely based on current input is a reflex agent based AI:
Eg: Vaccum cleaner
If dirty → suck
else if left → go right
if right → go left.

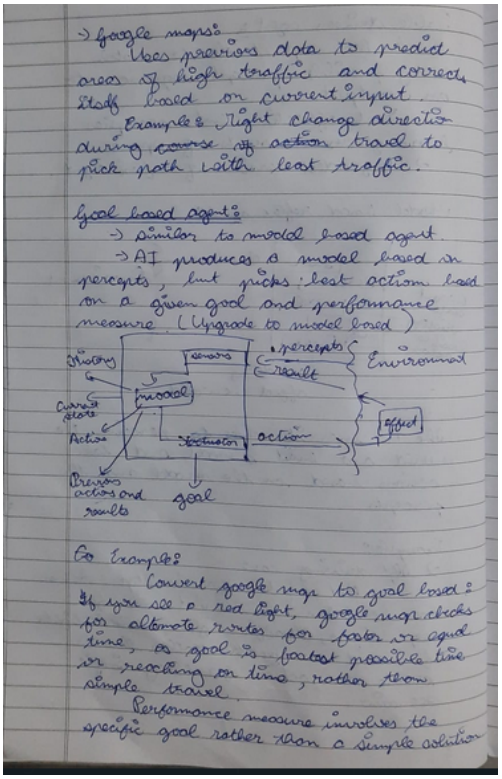
Table driven agent:
→ A look up table to match every possible percept sequence to an output.
→ Most effective but occupies humongous amounts of space
→ Upgraded version of reflex agent.

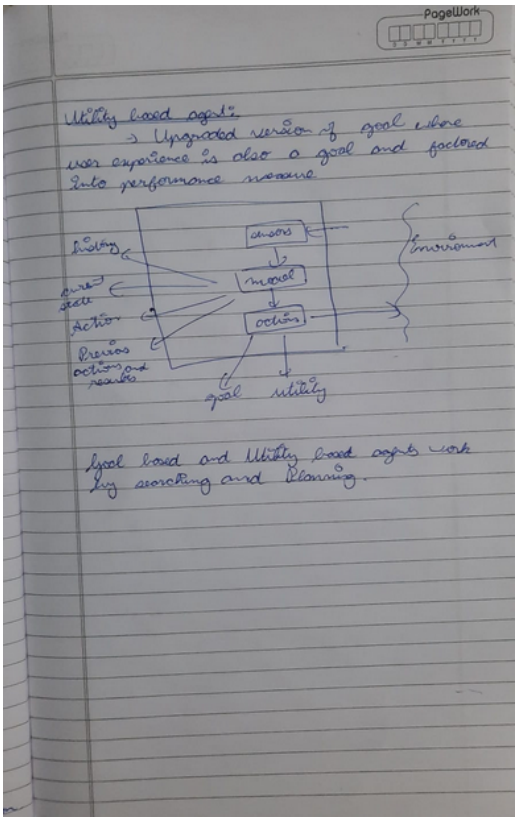
Model based reflex agent:



The AI uses its sensors to perceive the world and build a model and take actions based on the model and the percepts.

Example:
→ If driving car:
Red light refers to stop. Action can be taken as blanket rule for stop signs, traffic lights, vehicle lights if based on model rather than create an action for every single scenario.





Group 13 evaluation 9th August
Anirudh VS - 106119012
Rinish Sam I - 106119102

- Explanation is missing for agents with autonomy: The agent should be able to design and act, given a new environment.
- Example for agents with autonomy w.r.t the vacuum cleaner scenario is missing.
- For model training 80% of data is used to train the model and 20% is used to test the model. These values are mentioned in correctly.
- Example for utility based model is missing.
- Apart from this all the points are covered.

Class notes of 10th August 2021. Prepared by Group 15: Madhavan B(106119068) and Sakthi Prasath S(106119108).

Link to PDF:
https://drive.google.com/file/d/1Mi1boRfKvp7Z05ULsJit8pT0aH375X-J/view?usp=sharing

0/8/2021

AI/ML

பக்கம்:
நாள் : 1 1

* Goal, Utility based Agents require lot of searching and planning.

* Fully observable environment v.s Partially
  - Complete environment is available
  - Partial env.
* The other one is ~~most~~ Un-observable env.

→ Google Maps: Fully observable env. (it senses complete env. at any particular point of time.
→ Autonomous driving vehicle: Fully observable env.
→ Vacuum cleaner: Partially observable env.

* Autonomous vehicle is a dynamic agent. it must be fully observe the env. to avoid risk.

* If autonomous vehicle ~~gets of data~~ ~~senses~~ ~~many~~ may not ~~have~~ sense the full env. So it ~~may~~ is partially at some times.

→ Single agent env. vs Multi agent. env

- Crossword puzzle solver. (All the clues and other data are present with itself)
- Autonomous vehicle (Different agents give input to an agent)
- ~~Invoto~~ chess

→ Deterministic vs stochastic

- If the next state depends upon the current state and the action
- If the next state doesn't depend upon the current state and the action.

Stochastic can be thought as partially observable

Deterministic can be thought as fully observable

* Taxi driving can be ~~base~~ called as stochastic environment.

பக்கம்:
நாள் : 1 1

* Episodic v.s Sequential
  - Action is divided into atomic components * next action doesn't depend on previous action. *Individually perceived and acted. Ex: Small components integrated into a big one. (Ex: Garden sprinklers).
  - Current action depends on previous action. (Ex: gaming, vacuum cleaner)

* Static v.s Dynamic
  - Ex: Crossword puzzle, garden sprinkler
  - Change in * Ex: games

* Discrete vs Continuous

Let us consider the environments have states and time is a factor.

பக்கம்:
நாள் : 1 1

Environment Types

Fully

continuously

* If env is ~~depicted~~ at some points in time
* Ex: Chess, games,

* Known vs Unknown

Agent knows
* Outcomes are available for all actions.
* Ex: Tic Tac Toe game with all possibilities recorded.

Agent doesn't know
* Outcomes / Actions are dynamically decided
* Ex: Taxi driving

* Taxi driving

	Backgammon	Internet Shopping	Taxi
Observable	Yes	No	~~No~~
Deterministic	No	Partially	No
Episodic	No	No	No
Static	Semi	Semi	No
Discrete	Yes	Yes	No
Single Agent	No	No (except auction)	No

	Medical Diagnosis	Interactive Tutor
Observable	Partial	No
Deterministic	No	Stochastic
Episodic	No	sequential
Static	Dynamic	Dynamic
Discrete	Continous	Discrete
Single Agent	Single	Multiagent

To evaluate an agent, we have a measuring scale known as PEAS.

P- Performance measure
E- Environment
S- Sensors

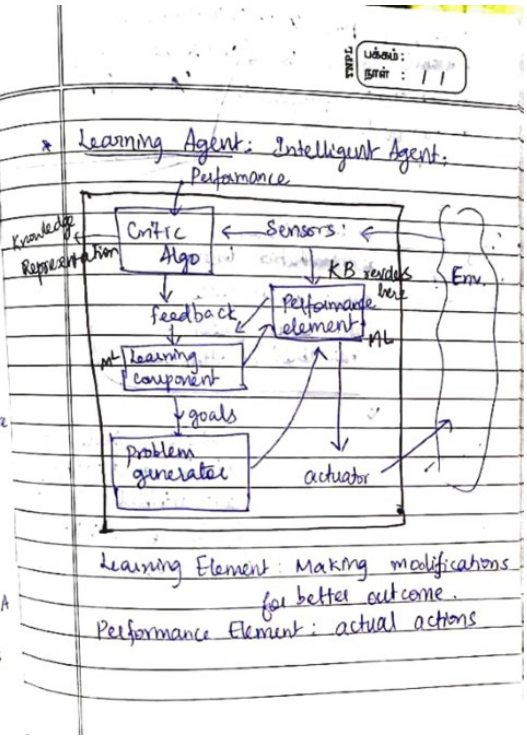
* The real world is (of course) partially observable, stochastic, sequential, dynamic, continous, multiagent

PEAS for English Tutor

Performance Measure: Student's performance - Oral, written
Env : set of students, Testing agency
A : Marks,
S : Keyboard Entry / Oral English capture

PEAS for Medical Diagnostics

P: Healthy patient, Low cost
E: Hospital, Patient, staff, Lab.
A: Diagnostic report, Treatment, Refund, Q&A
S: Entry of symptoms, Patient's answers

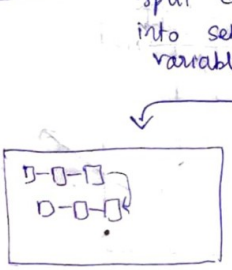


Learning Element: Making modifications for better outcome.
Performance Element: actual actions

Types of components in environment
* Atomic components
* Structured components
* Factored.

One to one correspondence A → B of components

Huge component → where factored representation tries to split each state into set of fixed variables or attributes.



where a state includes objects and each object have attributes and each objects are linked to each other.

Examples for atomic representation : Game playing | Hidden map models . This is based upon (HMM) Markov Decision Process (MDP) [will learn about it later.]

Example for factored representation : i) Constraint Satisfaction Problems (CSP) ii) Problems based on propositional logic iii) Bayesian networks iv) ML Algorithms

Examples for structured representation: i) First order logic. ii) Natural Language Understanding (NLU) iii) Knowledge based learning (KBL).

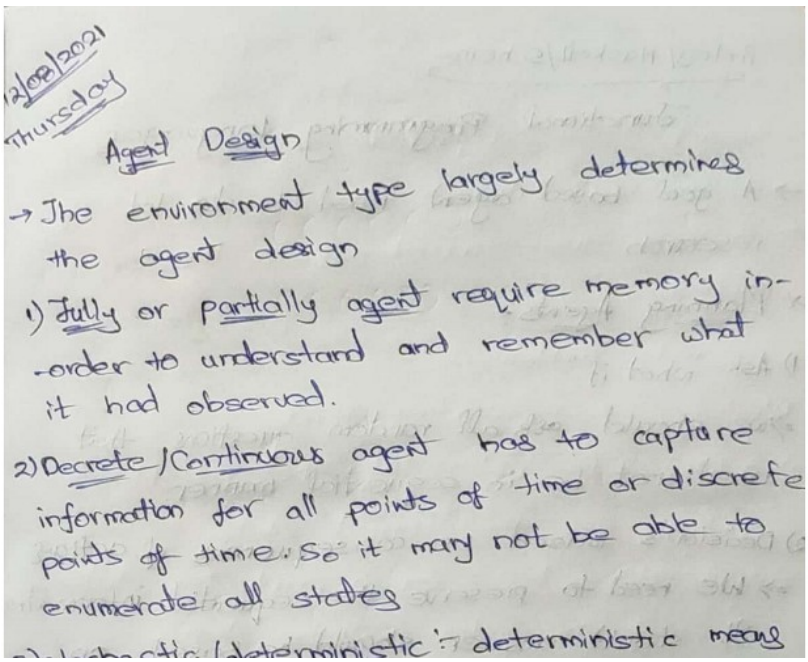
In structured representation knowledge representation plays a major role. We would work with 'ontology'.

Ontology is the connection between entities and relationships.

Group 16 evaluation on 10th August
DharunSri - 106119028
sameer s – 106119114

- Peas for English Tutor
  Actuators : Exercise given to students, suggestion, speech, correctness (missing)
- Problem generator - knowledge reasoning skill (missing)
- Atomic components - (explanation)
  We cannot divide the environment further(missing)
- Apart from this all the points are covered

12/08/2021 Notes – Group 20 (106119136,106119132)


12/08/2021
Thursday

Agent Design

→ The environment type largely determines the agent design

1) Fully or partially agent require memory in-order to understand and remember what it had observed.

2) Decrete / Continuous agent has to capture information for all points of time or discrete points of time. So it may not be able to enumerate all states

3) Deterministic / Non deterministic : deterministic means

3) Stochastic / deterministic

the next state depends on the present state so the agent have to prepare for contingencies.

4) Single / multi agent :- In this case agent need to behave randomly.

→ Atomic structure :- we cannot divide the problem into further.

→ factored :- We can create sub problems. A huge problem can be factored into small sections.

→ structured :-

We will establish a link between the problem statements.

Scanned with CamScanner

Prolog / Haskell / Scheme

Functional Programming languages

→ A goal based agent need to do lot of research

* Planning Agents :-

1) Ask "what if"

→ we should ask all random questions that need not be in sequential manner.

2) Decision's based on consequences of actions

→ We need to preserve the sequential information. It should answer should be based on either previous question alone or sometime from the 1st question to previous question.

3) Must have a model of how the world evolves in response to actions.

→ We should try to know how much information to be maintained by the agent and we should formulate a goal.

→ Consider how the environment would be based on present and past scenario.

* Optimal vs. complete planning

→ Complete

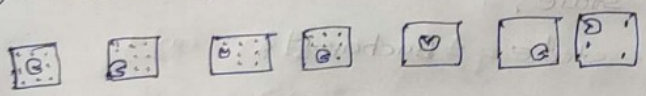
planning is not possible, because we are designing a agent and it is dynamic as it is keep learning and optimal is what should be remembered

* Goal test:
→ Test to verify the given state is final state or not
* Path cost:
→ Function to assign a numeric cost to each path
* Solution:
→ The one that leads to the final state from the goal state.

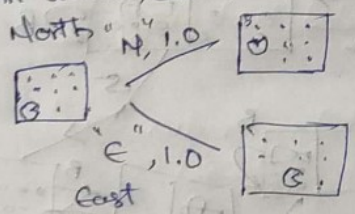
Search Problems

- A search problem consist of:

1) A state space



2) A successor function (with actions, costs)



3) A start state and a goal test

→ A solution is a sequence of Actions which transforms start state to goal state.

and what should not be

* planning vs. replanning:

→ Plan ahead and replanning the already planned information to do replan.

→ planning agent and replanning agent works together.

eg:- pac man :-

In replanning we reduce the time by making it not to move in the empty space much.

→ Search problems are used by Problem solving agents.

* Problem Solving Agents

→ Uses atomic representations :-

states are considered as whole as no internal structure visible to the problem. no possibility to split them further.

→ Goal formulation :- Based on current situation and agent's performance measure, it is done.

→ Problem formulation :- We have goal and current situation. We need to convert current situation to goal as a problem.

→ Environment is observable, discrete and deterministic

Scanned with CamScanner

→ Process of looking for a sequence of actions is called search.

→ Search take a problem as input and returns a solution in the form of action.

→ Actions are carried out in the execution phase

Search Problems :-

1) Initial State - starting Point

2) Actions :- Description of the possible actions give the current state.

3) Applicable action :- For a particular state some of the action will be done.

4) Transition Model :- A description of what each action does.

5) [illegible] state reachable from a

given state.

6) State space :- set of all states reachable from initial state by any sequence of actions.

7) Graph :- Nodes are states and the links between nodes are actions

8) Path :- Sequence of states.

* Goal test :

→ Test to verify the given state is final state or not

* Path cost :

→ Function to assign a numeric cost to each path

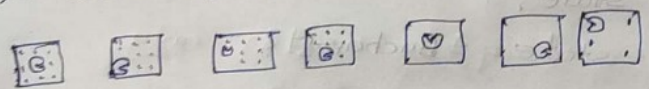
* Solution :

→ The one that leads to the final state from the goal state.

Search Problems

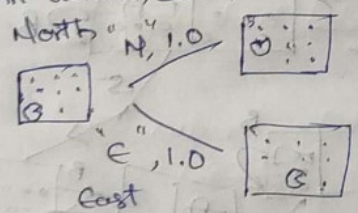
- A search problem consist of :

1) A state space



2) A successor function

(with actions, costs)



3) A start state and a goal test

→ A solution is a sequence of Actions which transforms start state to goal state.

Eg :- A map consisting of cities of

and edges as cost for travelling b/w them.

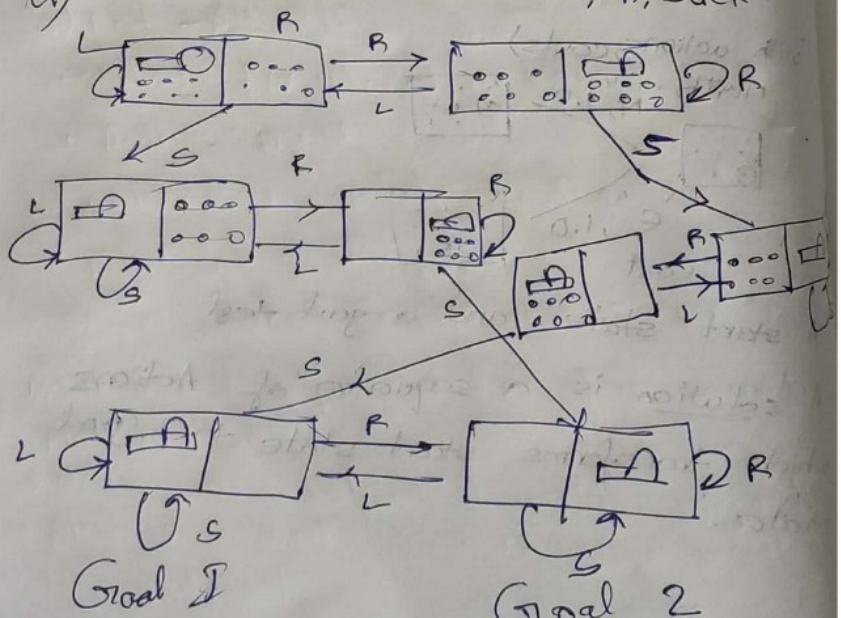
→ state space:
- cities

→ Successor function:
- Roads: Go to adjacent city with cost = distance.

→ start state:
- Arad.

→ Goal state:
- Is state == Bucharest?

Q) Vacuum world: L, R, Suck



Scanned with CamScanner

states: 2×2^2 Agent → 2

⇒ 2×2^2 possible states

(n) ⇒ $n \times 2^n$

Initial State:

Action: L, R, S

Transition Model:

Goal test: all the squares are clean

Path test: each step cost 1 ⇒ no of steps

Q)

7	2	4
5		6
8	3	1

⇒

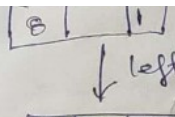
1	2	3
4	5	6
7	8	

Left, Right, Up, Down movement

↑ up

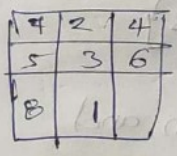
7	2	4
5	3	6

No of actions = 9×2^9 a lot



left

we should think of Applicable actions.



Q) 8-queens problem

Initial state: No queens

actions: Add a queen such that No queens attack each other

transition state:

Scanned with CamScanner

Q) → Route finding:

State: Location.

→ Initial state: user?

→ Actions: any mode of transport.

→ transition Model: state result on go.

→ from one state to next state

→ Goal test: Have I reached destination

→ Path cost: Petrol (or) Calories

Q) For Pac Man Problem

1) States: (x,y) location

2) Actions: NSEW

3) Successor: update location only.

4) Goal test: is (x,y) = END.

Q) Eat-All-Dots.

- States: {(x,y), dot booleans}
- Actions: NSEW
- Successor: update location and possibly a dot boolean

→ state space Graphs and search Trees

1) Mathematical representation of a search problem

→ Nodes are (abstracted) world configurations

→ Arcs represent successors (action results)

→ The goal test is a set of goal node

Scanned with CamScanner

→ In state space graph each state occurs only once.

→ We can rarely build this full graph in memory (it's too big), but it's a useful idea.

Search Trees :-

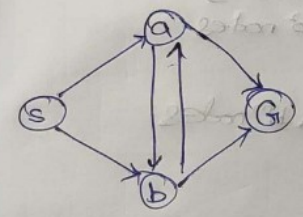
→ A "what if" tree of prans and their outcomes

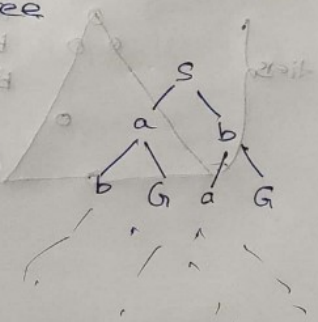
→ The start state is root node.

→ Children correspond to successors.

→ Nodes show states, but correspond to PLANS that acheive those states.

→ For most problems, we can never actually build the whole tree





* Searching with a search tree

→ Expand out potential plans (tree nodes)

→ Maintain a fringe of partial plans under consideration.

→ Try to expand as few tree nodes as possible

Depth-First Search

Strategy :- Expand a deepest node first.

Implementation :-

Fringe is a LIFO stack

Properties of Search Algorithm

1) Complete :-

2) Optinal :

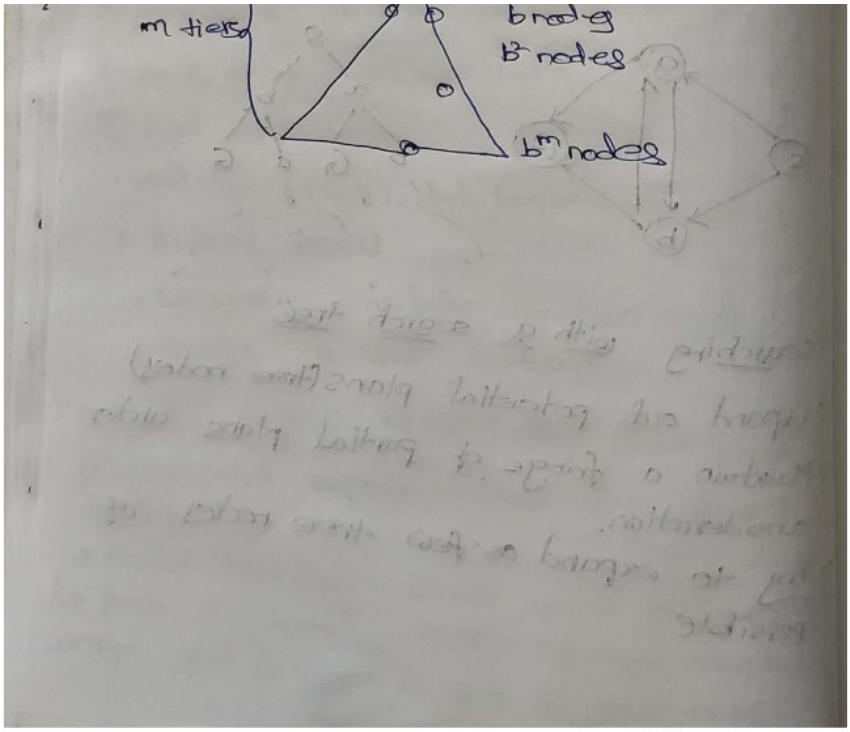
3) Time Complexity?

4) Space Complexity?

→ No of Nodes in entire tree

$= 1 + b + b^2 + \cdots b^m = O(b^m)$

↑ ↑ inode



Scanned with CamScanner

12th August 2021 – Notes Review by Group 18

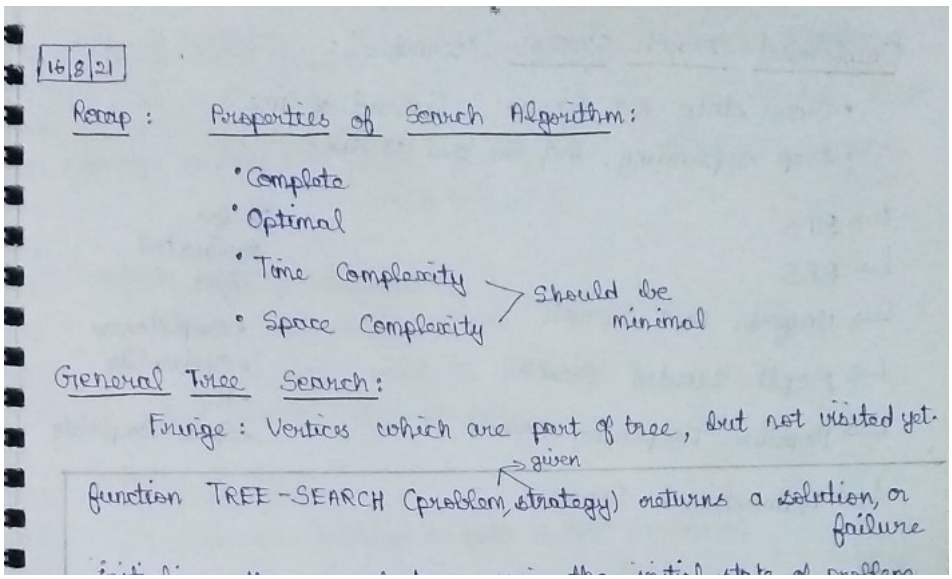
Pranav Somaiah- 106119094
Vaasu Gambhir- 106119140

- Prolog was classified under functional programming languages while in reality it is a **logic** programming language, also the example of Lisp was missed out on.
- Under Planning agents, the chatbot example wasn't mentioned for asking "What if" and under Problem Solving Agents, the map example wasn't mentioned(Google Map to get directions).
- Lex and Yacc, and how a problem is looked upon in a rule-based programming language wasn't mentioned(like how we state the problem instead of approaching it sequentially).
- Fringe is a LIFO stack that contains vertices of the path that is currently being visited, this was not specified fully.
- Important ideas of the DFS search were not given fully (they are:-having fringe, an exploration strategy and expansion). That along with steps of finding an optimal solution using DFS weren't given.
  -> Initialise starting state with init state,
  -> If there is no way to expand and/or goal state is not reached, return failure, else return success
  -> Else keep exploring candidates which can be further expanded.
- It isn't specified that we want to complete the problem that consumes least time and space and the solution should be optimal(under DFS)
- Apart from this the notes are well written and cover major portions of lecture and PPT.

AIMLnot...

16th Aug 2021. Class notes by group 23 (106119042(G.Subhasree) && 106119130(Tanu Gangrade))

Link to pdf: https://drive.google.com/file/d/1qIE_K259BX4E1nWwhKOf3c7_DHjeCtkT/view?usp=sharing



Initialize the search tree using the initial state of problem

loop do

if there are no candidates for expansion (→ no soln.) return failure

choose a leaf node for expansion according to strgy.

if node contains goal state, return corresponding soln.

Else expand node & add resulting node to search tree

end

Imp ideas:

Fringe

Expansion

Explora^{n} strgy.

Main question: Which fringe nodes to explore??

eg. Tree Search [refr. last class]

↳ search strgy. depending on how we search

Uninformed Search Strategy [Technique]:

- Goal state not known [ahead of time]
- Keep expanding, hit the goal ⇒ done

↳ DFS

↳ BFS

↳ Uniform Cost Search

↳ Depth Limited Search

↳ Iterative Deepening Search

↳ Bidirectional Search

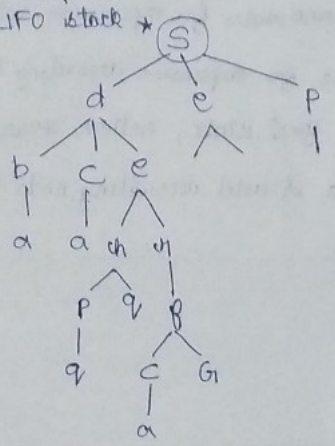
To be evaluated for:

- Completeness
- Optimality
- TC
- Space Complexity

DFS:

Last class eg. ★

- expand deepest node first
- Fringe: LIFO stack ★

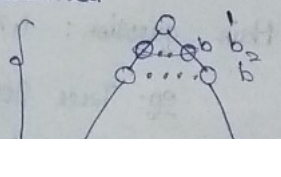


- Start from S [d, e, p are fringe vertex]
- Pick b, go till a ↓ not goal ↓ backtrack
- continue as on . . . . . .
- reach G: Goal state ↓ Soln.

Go as deeper as want

Eval: Soln. in least cost: not guaranteed [Complete & Optimal]

discussed [TC: order b^m



last class | SC: 0

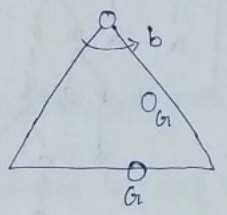
• No. of nodes: $1+b+b^2 \dots +b^m = O(b^m)$ ↓ upper bound [including repeating nodes]

b: branching fac.
m: max. depth

• fringe vertices: in stack ↳ grows big

→ Which nodes r expanded?

- Some left prefix of tree
- whole tree could be processed
- m: finite ⇒ $O(b^m)$ time



→ Space of fringe?

- Only has siblings on path to root ⇒ $O(bm)$

→ Is it complete? [get soln. for sure]

- Not always — DFS directed graph ↓ sometimes can't reach
- m could be ∞, ⇒ prevent cycles

→ Is it optimal?

- Not complete ⇒ Not opti
- finds leftmost, regdless of cost

Implemented using stack

Variation: Depth limited search → not guaranteed complete/opti.

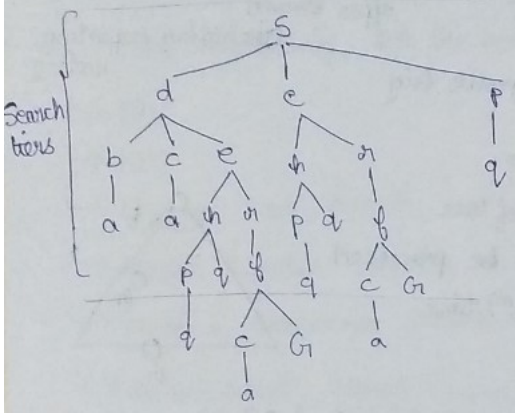
↳ limits ~~branching factor~~ depth

↳ depth pre fixed by user not till m, go till some l

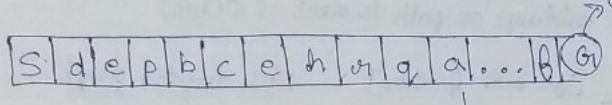
BFS:

- expand[a] shallowest node 1st
- Implementa[n]: Fringe is FIFO queue

Eg.



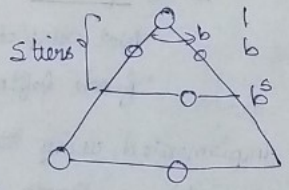
- Start from S
- expand to d, e, p → fringe
- put fringe vertex in Queue
- Visit unvisited vertices one by one [d] & put adj. vertices [b, c, e for d] in Queue
- See if Goal is reached @ shallower lvl.



inserted again

S → e → r → f → G

level by level search



→ What nodes expanded?

- all nodes above shallowest soln
- Let depth of " be s

⇒ Search time : $O(b^s)$

if s = m, $O(b^m)$

→ Fringe space?

~ last tier ⇒ $O(b^s)$

→ Is it complete? processes lvl by lvl till s

- s is finite if soln. exists ⇒ complete

→ Optimal? if all costs are same

else no

Modificⁿ of BFS : UCS

DFS vs BFS:

BFS outperform DFS:

- less connected graph, Goal @ shallower depth

DFS outperform BFS:

- If graph is completely connected, get soln. in left half itself in DFS

[every node connected to every other]

Iterative Deepening Search:

- Final strgy
- Combines BFS & DFS
- Idea : get DFS's adv of space with BFS's time / shallow adv.
- Deepen only after exploring shallowness
- Not O strgy of BFS, not stack strgy of DFS

Increases limit of algo : from 0, 1, ...

[uses depth limit concept] fix limit & do DFS

→ Run DFS with depth limit 1. If no soln. . . . .

↳ [nothing but BFS]

→ " " " " " 2. " " "

[all vertices not put in stack, dl = 2, only till 2 its put]

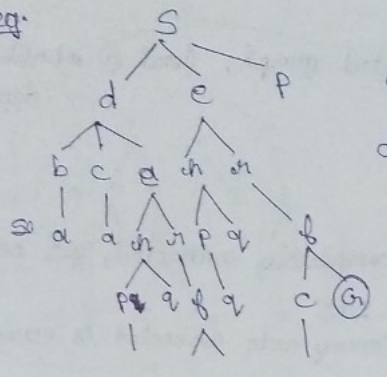
→ " " 3. . . .

. . . . . .

* Redundant wastefully? not costly, complete & optimal soln.

• most work in lowest dpl search ⇒ not so bad!

eg:



dl=1

dl=2 search till b is come back,
c " "
e " "
. . . . .
. . . . .

If dl=4, G1 is reached!

• limiting depth : BFS incorporated
• called iterative as d.l is not pre fixed

COST SENSITIVE Search: [Uninformed]

• BFS opti. only when costs r same
finds shortest path in terms of no. of actions

↳ To find least cost path
Uniform Cost Search

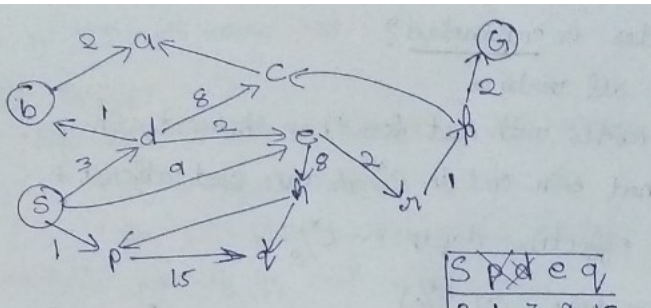
Uniform Cost Search [UCS]:

• Variation of BFS
• gets least possible cost
• Not queue strgy., greedy expansion of fringe vertices & choose path
• Sort vertices in queue, based on cost

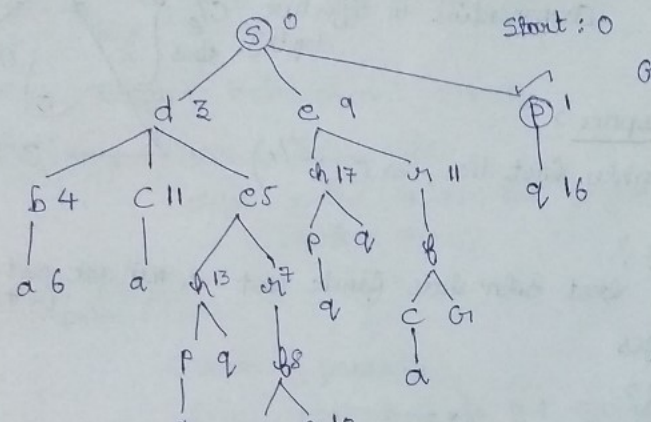
Strgy: expand cheapest node 1st

Fringe is priority [cumulative cost] queue.

Eg:



S	p	d	e	q
0	1	3	9	16

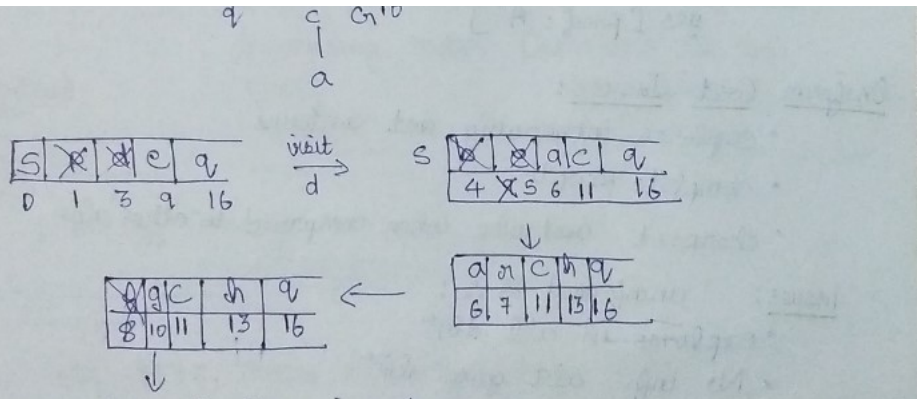


Start : 0

Go with lowest cost

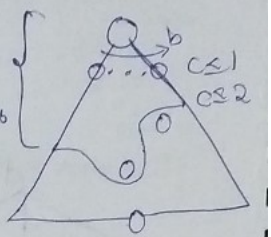
BFS with first node visited being lowest cost node

↓ cumulative



↳ What nodes r expanded?

- not all vertices
- all nodes with cost less than cheapest soln.
- If that soln cost is C^* & arcs cost atleast ϵ,
  ⇒ effective depth ~ C^*/ϵ
- Takes time $O(b^{C^*/\epsilon})$
  [exponential in effective depth] C^*/ϵ tiers



↳ fringe space?

- roughly last tier, ⇒ $O(b^{C^*/\epsilon})$

↳ complete?

- if best soln has finite cost & min arc cost is +ve,
  yes

↳ Optimal?

yes [proof: A^*]

Uniform Cost Issues:

- explores increasing cost contours
- complete & opti
- cheapest cost wise when compared to other algo

Issues: uninformed & ∴ :

- explores in all dir^
- No info abt goal loc^

to be covered with informed strgy.

Demo: 1: UCS with some knowledge about goal
2: UCS
3: BFS as the path goes via first deepest node

- BFS is UCS with same cost

The One Queue:

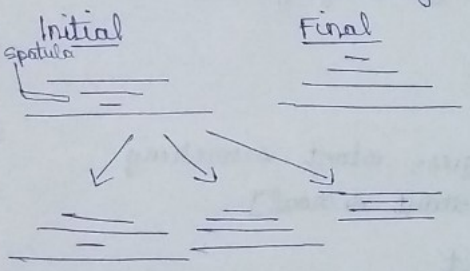
- All algos r same except fringe strgies:
  - all fringes r priority queues
  - for DFS & BFS, can avoid $\log(n)$ overhead from actual priority q; with stacks & q
  - Can code an implementn with variable queuing objs.

- One more strgy: Bidirectional search

eg: Google map: ocean b/w 2 pts.
says go thr. ocean bt 6 yrs.
[walk in ocean]

Pancake Problem:
[chapathi / dosa]

- Stack of pancake
- flip pancakes to get arrangemnt in increasing order [smallest on top]
- How many flips: cost



- In 1978, Gates & Papadimitriou did:

for permu. σ of 1 to n, • $f(\sigma)$: smallest no. of prefix reversal for $\sigma \rightarrow$ identity permu.

- $g(n)$: longest $f(\sigma)$ for all σ in (symm. grp.) S_n

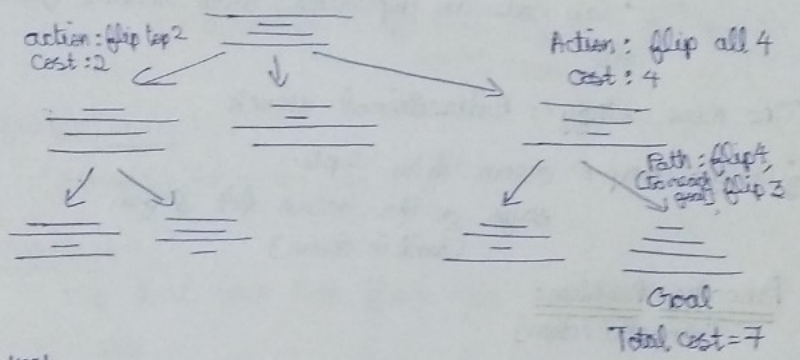
we show, $f(n) \leq (5n+5)/3$ & $f(n) \geq 17n/16$
n: multiple of 16

If each int. is required to participate in even no. of reversed prefixes, corresponding $g(n)$ obeys:

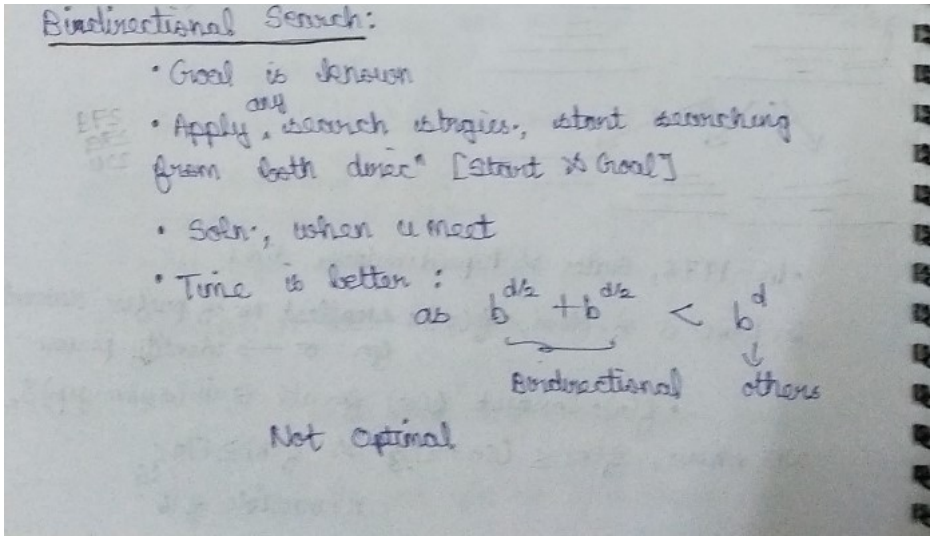
$$3n/2 - 1 \leq g(n) \leq 2n+3$$

III^er to:
$2^n - 1$ moves for Tower of hanoi

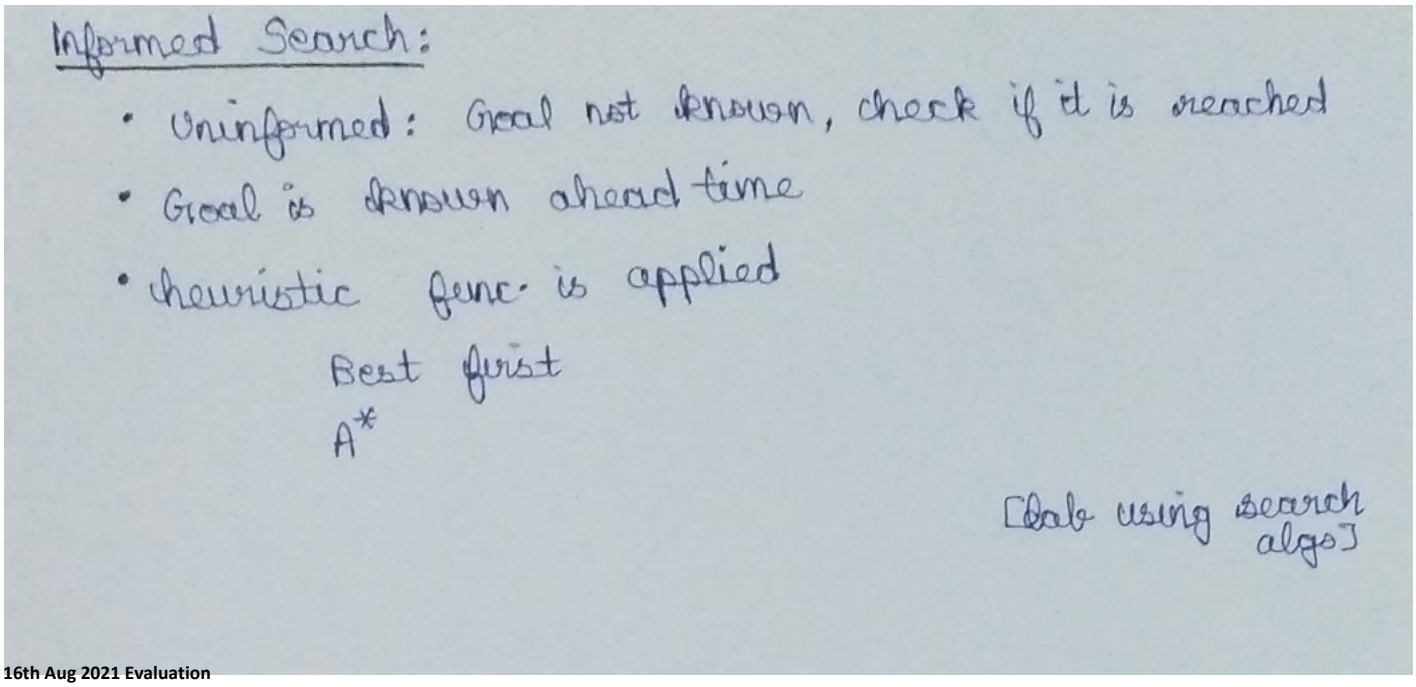
→ State space tree is based on gnl. tree search



- UCS was applied on Pancake problem
  later proved that Pancake prob was shown can get only feasible soln



AiML-Notes-17-



16th Aug 2021 Evaluation
Group no: 22
Members: Shashwat.T (106119138), Hari Hara Sudhan (106119046)

1) Missing point: The one queue:
   a. only for DFS and BFS it can be used
   b. for UCS we have to only use priority queue
2) Apart from this the notes are perfect and covered every single line said in the lecture and PPT shown

Notes for 17-08-

FILE FOR EASIER VIEWING

Date: 17/08/2021

Notes by : Group 25 Suseender (106119122) & Suraj (106119128)

Topics covered in class:

- Informed Search Strategies
  - Introduction to Heuristic functions
  - Greedy Best First Search
  - A* search algorithm

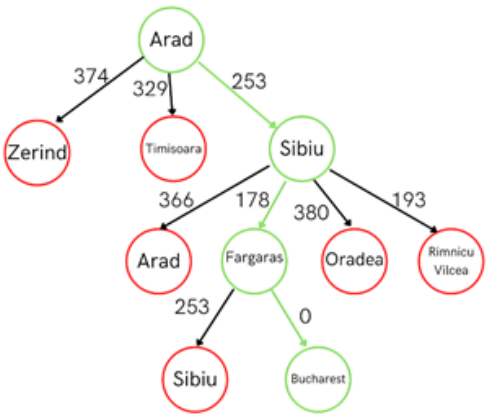
Informed Search Strategies:

- These are strategies that are based on the concept of Heuristic function:
  - Functions that decide on the efficiency of the algorithm where the goal is already known. It is a function that estimates how close a state is to a goal state. And designed for each problem individually.
  - The general approach is based on Best First Search (expand the best node first).
    - 2 types of approaches are the tree search or the graph search.

    - We go to the next node from the current node based on the evaluation function, f(n).
  - The Heuristic Function, h(n) is the cost of the lowest path from the start node 's' to the end goal 'g'.
  - One of the uses of these search strategies is in games where paths need to be found.
  - We use different ways of measuring distance such as Manhattan distance (*The Manhattan Distance between two points (X1, Y1) and (X2, Y2) is given by |X1 – X2| + |Y1 – Y2|),* Euclidean Distance for paths.
  - Heuristic function can be used to solve the Pancake Flipping problem.
  - Romania Map Example

- For this example, let the start city be Arad and the end city be Bucharest. We have been given the distances between the cities and the straight-line distance between each city and the goal, Bucharest. (We can use haversine distance to estimate straight-line distance between 2 points using curvature. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances.html).

- **Using Greedy Best First Search**
  - Best First Search is based on the straight-line distance and the greedy approach uses a heuristic on the straight line distance to solve it.
  - In Greedy Best First Search you will look at the neighbouring nodes of the current node and then choose the one with the least straight-line distance to the goal city.
  - The heuristic is to estimate the distance to the nearest goal for each state.
  - Below is the Greedy Best First Search implementation for the Romanian map example:

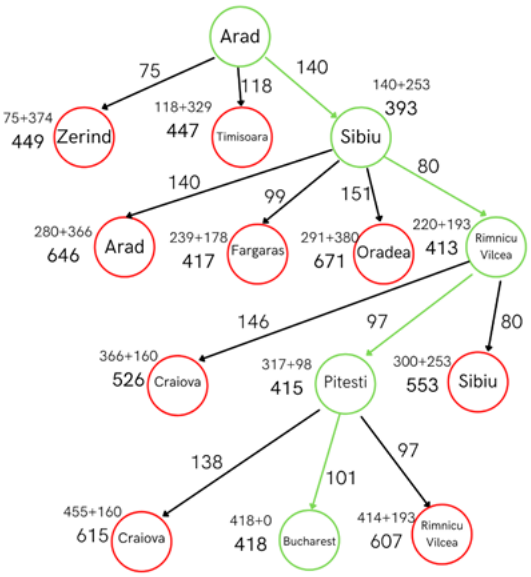


    - In the 1st step Sibiu is chosen as in has the least straight-line path to Bucharest and its nodes are expanded
    - In the 2nd step Fargaras has the least straight-line path and it is expanded. This is done until Bucharest is reached.
    - The final path is Arad->Sibiu->Fargaras->Bucharest, and the final cost comes to 450.
  - Greedy is not very effective as we need straight line distance to all the nodes at the worst-case time complexity is (b^m) if all nodes are expanded.
  - If goal is not properly defined, then it might lead to the wrong answer. The worst case is that it will become like a badly guided Depth First search.
- **A* Searching Algorithm**
  - Was used for a long time in games
  - Variations of A* are memory bounded A* (MA*) and simplified memory bounded A* (SMA*).
  - A* is similar to greedy but it is based on 2 functions rather than 1.
  - In greedy straight-line distance from current state to the goal state was considered but the path from source to the current state was not considered.
  - In A* we will consider 2 costs g(n) and h(n).
    - g(n) is the cost to reach a particular node (path cost from start till current node).
    - h(n) is the cost from the current node to the goal (this could be straight-line distance).
    - Therefore f(n), the heuristic function can be stated as f(n) = g(n) + h(n).
  - If UCS was a tortoise and greedy was a rabbit, then A* is both combined. UCS considers the cost of leaving a particular node and greedy considers the cost to reach a particular node. So, we can see how A* is a combination of these 2.

  - The conditions for optimality for A* are:
    - Admissibility
      - The Admissible heuristic tries not to overestimate the cost to reach the goal.
      - A heuristic h is admissible if 0 <= h(n) <= h*(n), where h*(n) is the true cost to a nearest goal.
    - Consistency
      - Also known as monotonicity.

- A heuristic function h(n) is called consistent if for all n and n' (*which is successor of n)* if the estimated cost for reaching the goal from n is not greater than the step cost of getting to n' + the cost of reaching the goal from n'.
- h(n) <= c(n,a,n') + h(n') {*h(n) is cost to reach goal from n*}
- It is a stronger condition.
- It is trying to prove the correctness of you're A* algorithm.
- It is used in graph search algorithms

- Below is the A* algorithm when used on the Romanian map example. At each node we will be calculating g(n) + h(n).



  - At each arrow the cost of going between the cities is written. The sum at each node is g(n)+h(n) and the number below is the final sum f(n){g(n)+h(n)}.
  - In the 1st step initial g(n) = 0 for starting city. For Zerind g(n) = 75 and h(n) = 374 and hence f(n) = 75 + 374 = 449. Similarly, for Timirosa and Sibiu f(n) is calculated and least f(n) from all the nodes is chosen.
  - We can see the final traversal route is Arad->Sibiu->Rimnicu Vilcea->Pitesti->Bucharest, and the final cost comes to 418.
  - We found that in the greedy method the final cost came to 450. Hence, we can see how the A* search algorithm can give a better result.
- When should A* terminate?
  - We normally stop when we come to a goal state
- Is A* Optimal?
  - A* works well if it is a tree but works less well in a graph where there maybe cycles in which it might get caught in travelling back and forth.
  - For A* Tree Search, it is optimal as it will reach the optimal fringe vertex first before a suboptimal one. Because all ancestors of optimal node will be reached before a suboptimal one and hence optimal solution will also be reached before the suboptimal solution.
- USC vs A*
  - While UCS expands in all directions equally, A* expands mainly toward the goal while hedging its bets to ensure optimality.
  - USC does not keep in mind the goal. A* moves such that it reaches the goal each time.
- Applications of A*
  - A lot of video games
  - Paths and routes problems
  - Language analysis
  - Machine translation, etc
- 8 puzzle game can be solved using A* algorithm.

Evaluation for 17th August 2021 (Monday) Class

Done by Group 1 : 106119104-Rishi Sathish and 106119118-SreeGaneshTN

- Definition of Search Heuristics is missing .
- Proof of Optimality of A* Search is missing.
- Graph Example for Termination of A* is missing.
- UCS vs A* Contour is missing.
- Other the above mentioned points, everything is covered.

Informed search → Based upon heuristic function.

will decide on the efficiency of the algorithm, where goal is reach.

Search Heuristics:

General approach:

Best-first search: (Which node will we expand first)
→ We will expand the best node first which is based upon the goal
→ Depends NOT ONLY upon the source.

In order to do that:

We can go for → Tree search
or → Graph search

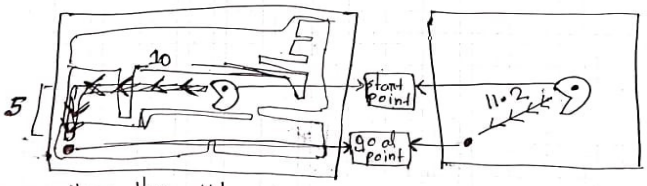
Choosing the approach is based upon:-

Heuristic function:-

h(n) = cost of lowest path from the start(s) Node 'n' to a goal state 'g'.

From the starting Node we'll use an evaluation function → which will be used for expanding the next Node and selecting the next Node. f(n)

A heuristic is
- A function that estimates how close a state is to a goal.
- designed for a particular search problem.
- example: Manhattan distance, Euclidean distance for pathing (gaming).



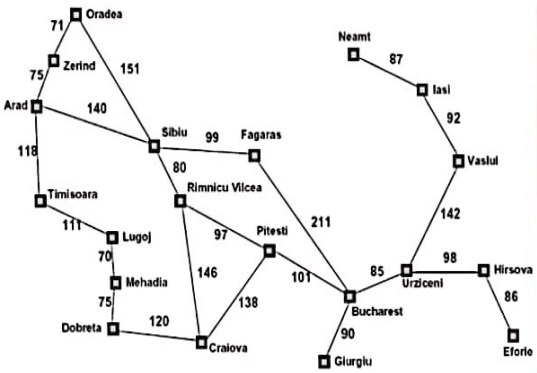
Here the distance is 15 (actual cost)

11.2 is SLD

SLD (straight line distance)

(Only if we're able to go)

Example: Heuristic Function



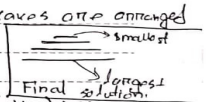
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

EVERWAY

DATE:

- So we have to get our ~~cost~~ Heuristic function such that we the cost will going to be almost close to the ~~SLD~~ SLD.
- if we have landlord scenario we can use haversine distance in order to estimate the straightline distance between 2 points
- Haversine distance is basically ~~fato~~ to estimate the distance b/w 2 points in a Geographical situation.
- We can use this function to solve pancake problem.
  → We're going to flip the pancakes such that the pancakes are arranged from smaller to longer from top to bottom.

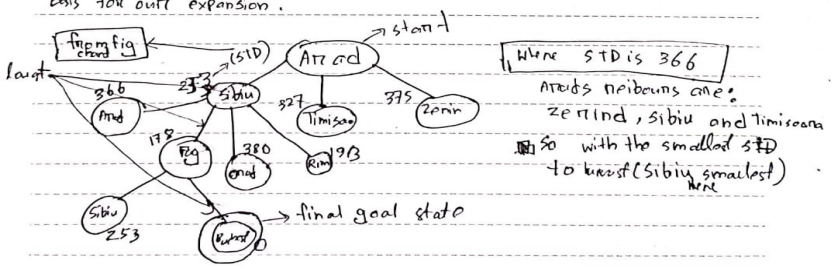


Greedy Search / Greedy Best first search / Pathway of

→ based upon SLD as the heuristic function

time/s complexity: O(b^m)

When we want to expand a particular Node the SLD is used as the basis for our expansion.

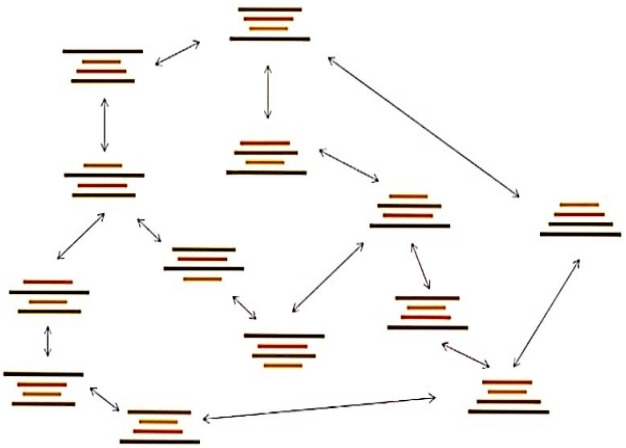


Where STD is 366

Arads neibours are: zerind, sibiu and Timisoara

So with the smallest STD to furthest (Sibiu smallest)

Example: Heuristic Function



EVERWAY

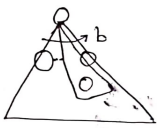
Greedy search

Strategy: expand a node that you think is closest to goal state.

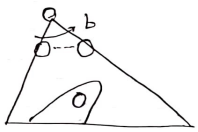
- Heuristic: estimate of distance to nearest goal for each state

A common case:

- Best-first takes you straight to the (WRONG) goal. (sometimes)



Worst-case: like a badly-guided DFS



A* Search (greedy)

→ Memory bounded A*/(MA*)

→ Simplified " A* / (SMA*)

Based upon 2 functions as opind a single heuristic function.

The problem that discussed in Best-first search was that we always looked at the SLD from the the new state to the goal state.

But the path cost from the source to the new state was not considered.

so we might have an alternate path which probably doesn't have good SLD But we might get has path cost. (It didn't consider that aspect of the algo)

So, in A* we'll consider 2 cost.

g(n) → cost to reach a particular node

h(n) → " " get from the node to the goal (2nd distance cost)

so total function, f(n) = g(n) + h(n)

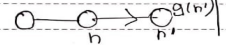
EVERWAY

DATE :

$g(n)$ → path cost from the stand state to the Node n } (can be)
$h(n)$ ↔ SLD

A^* → UCS → (g+h) (instead of g) UCS → lontoix
Greedy → rabbit
A^* → UCS ∪ Greedy

Optimality of A^*

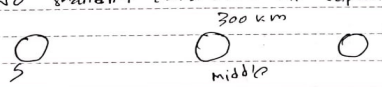
Admisibility of heuristic	consistency → stronger
→ tries to not overestimate the cost to reach the goal.	monotonicity
$g(n) \to n$, $f(n) = g(n) + h(n)$	$A^* \to$ graph search algorithm.
	$h(n)$ is consistent if $\forall n$ & n' successor of n the estimated cost of reaching the goal from n isn't greater than the step cost of getting to n' + the cost of reaching the goal from n'.

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

$h(n) \to$ cost to reach a goal state from 'n'

$h(n') \to$ " " " " " " " n',

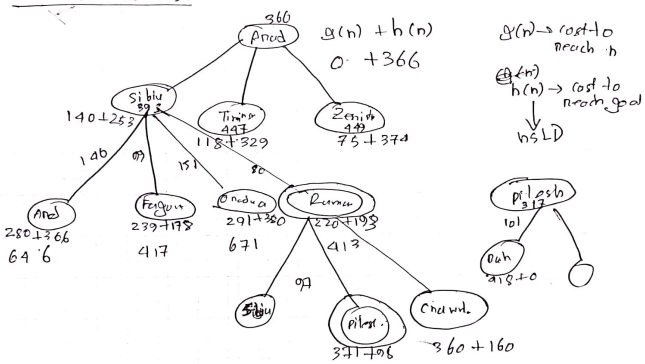
* At every particular point of time when we are trying to go from one node to next node we will consider the path cost to reach the particular node + the cost to reach the goal state, so we have to cumulatively do that, to do that we should'nt overestimate or we shouldn't ~~exist~~ exit the capacity - for example.



EVERWAY

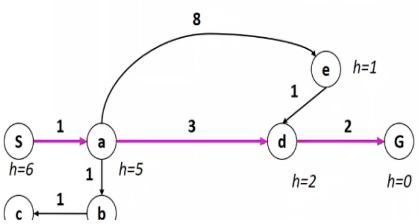
Every consistent heuristic is called as consistent heuristic and admissible (no overestimate)

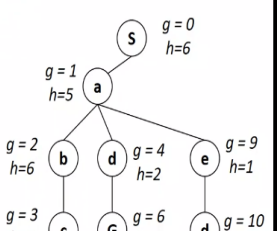
How A* behaves:

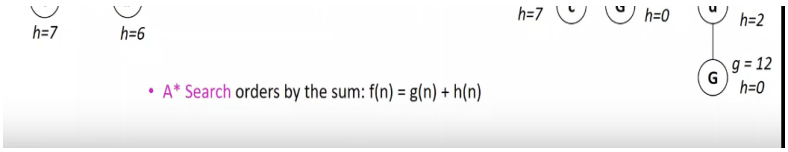


Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



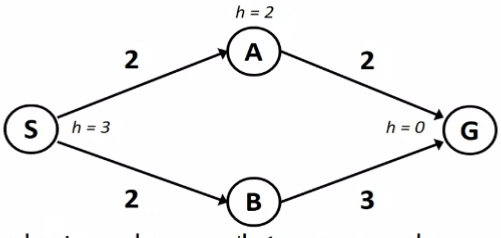




- A* Search orders by the sum: f(n) = g(n) + h(n)

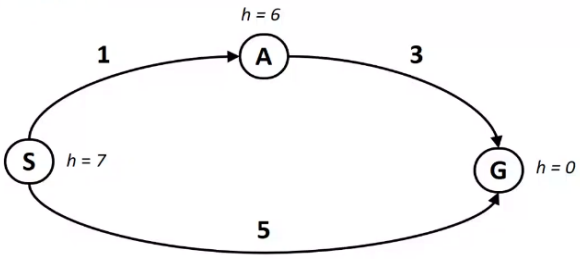
When should A* terminate?

- Should we stop when we enqueue a goal?



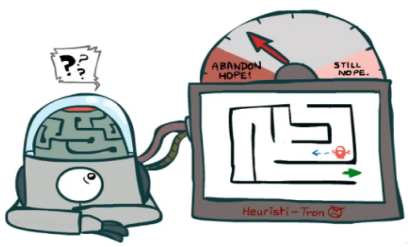
- No: only stop when we dequeue a goal

Is A* Optimal?

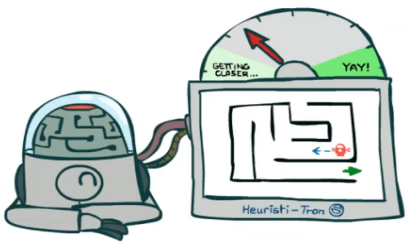


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

Idea: Admissibility



Inadmissible (pessimistic) heuristics break optimality by trapping good plans on the fringe



Admissible (optimistic) heuristics slow down bad plans but never outweigh true costs

Admissible Heuristics

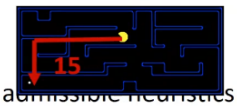
- A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

- Examples:

- Examples:

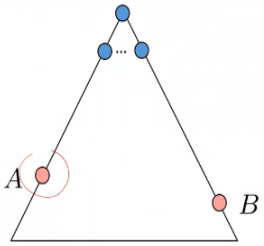


- Coming up with admissible heuristics is most of what's involved in using A* in practice.

Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible



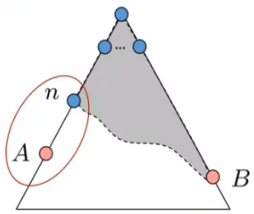
Claim:

- A will exit the fringe before B

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)

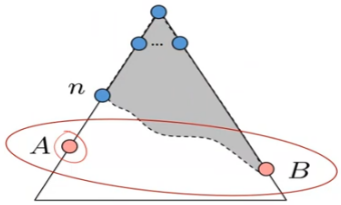


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)

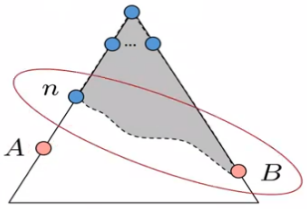


$g(A) < g(B)$	B is suboptimal
$f(A) < f(B)$	h = 0 at a goal

Optimality of A* Tree Search: Blocking

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B

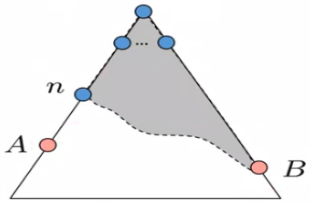


$$f(n) \leq f(A) < f(B)$$

Optimality of A* Tree Search: Blocking

Proof:

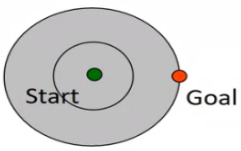
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



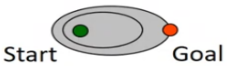
Properties of A*

UCS vs A* Contours

- Uniform-cost expands equally in all "directions"

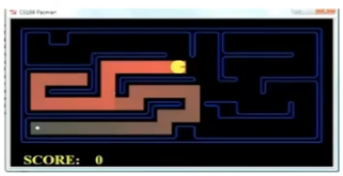


- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



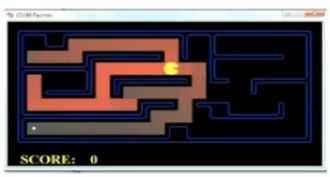
[Demo: contours UCS / greedy / A* empty (L3D1)]
[Demo: contours A* pacman small maze (L3D5)]

Comparison



Greedy

Uniform Cost



A*

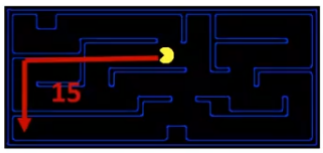
A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

[Demo: UCS / A* pacman tiny maze (L3D6,L3D7)]
[Demo: guess algorithm Empty Shallow/Deep (L3D8)]

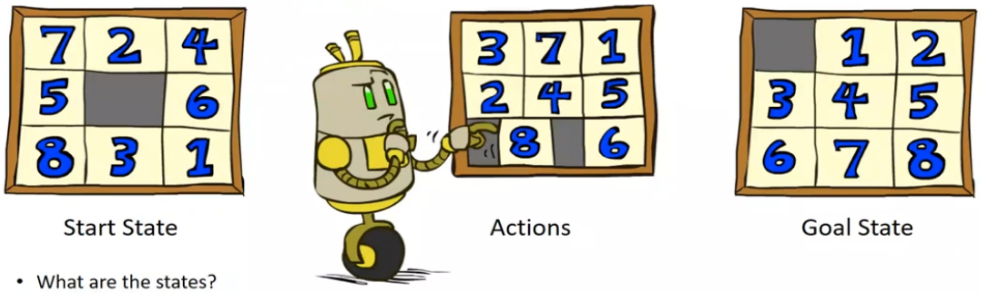
Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics
- Often, admissible heuristics are solutions to *relaxed problems,* where new actions are available



- Inadmissible heuristics are often useful too

Example: 8 Puzzle

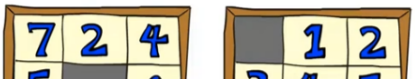


Start State | Actions | Goal State

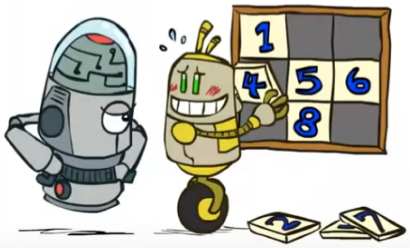
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

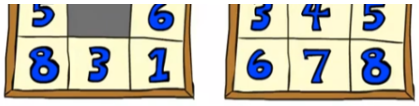
8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?



- h(start) = 8
- This is a *relaxed-problem* heuristic





Start State Goal State

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES	13	39	227

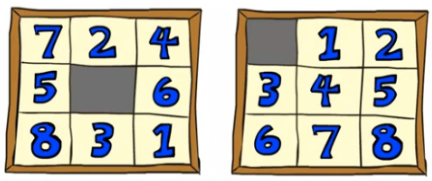
8 Puzzle II

- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
- Total *Manhattan* distance
- Why is it admissible?

City block distance

$3 + 1 + 2 + \ldots = 18$

- h(start) =

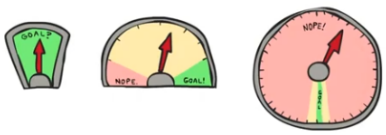


Start State Goal State

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
TILES	13	39	227
MANHATTAN	12	25	73

8 Puzzle III

- How about using the *actual cost* as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?



- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

23/08/2021 CLASS NOTES BY GROUP 8

Asif Mohaideen : 106119020
Divakar PS : 106119030

AI_ML_N...

PDF File for easy viewing

23/8/21

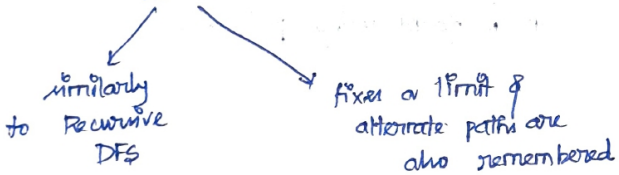
will A* always lead to a complete solution?
↳ Yes, but the problem is, memory, if the duration is too far.

so, the solution is:
→ memory bounded heuristic search
it is similar to iterative deepening. (IDA*)

$h(n) = f(n) + g(n)$ [with depth fixed]

→ thus the amount of memory req to store intermediate states is reduced

Another solution: Recursive best first search
RBFS → using linear space



pseudocode:

f_limit

return RBFS (problem, make_node (prob, initialstate, ∞))

RBFS (—, —, —)
if problem goal test (node.state) return s/n;
successor []; // assigns successory
for each action in problem.action (node.state)

if (successor is empty) return false

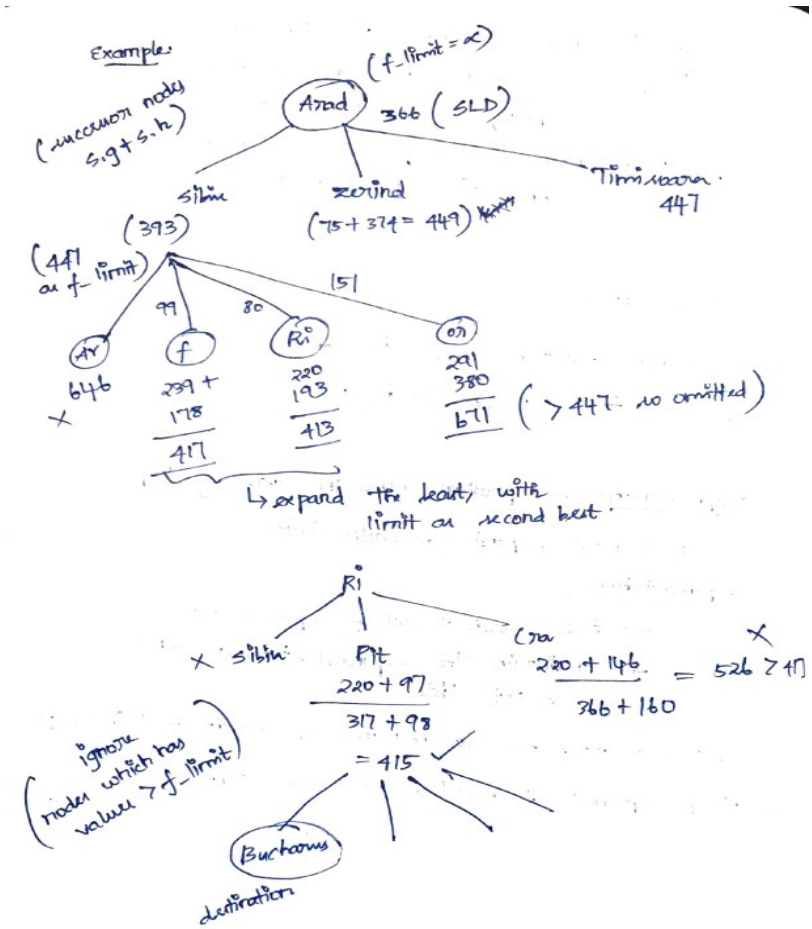
else (// if successor is available)
∀ s in successor
$s.f = \max(s.g + s.h, node.f)$
do
best = lowest f.value node in successor
if best.f > f_limit return false, best.f;

alternative node = record lowest f.value of successor

result, best.f = RBFS (problem, best, min (f_limit, alternate node))
if result ≠ failure

return result as solution.

successors can be found using priority-queue.



↳ expand the least with limit as second best.

simplified memory bounded A* (SMA*)

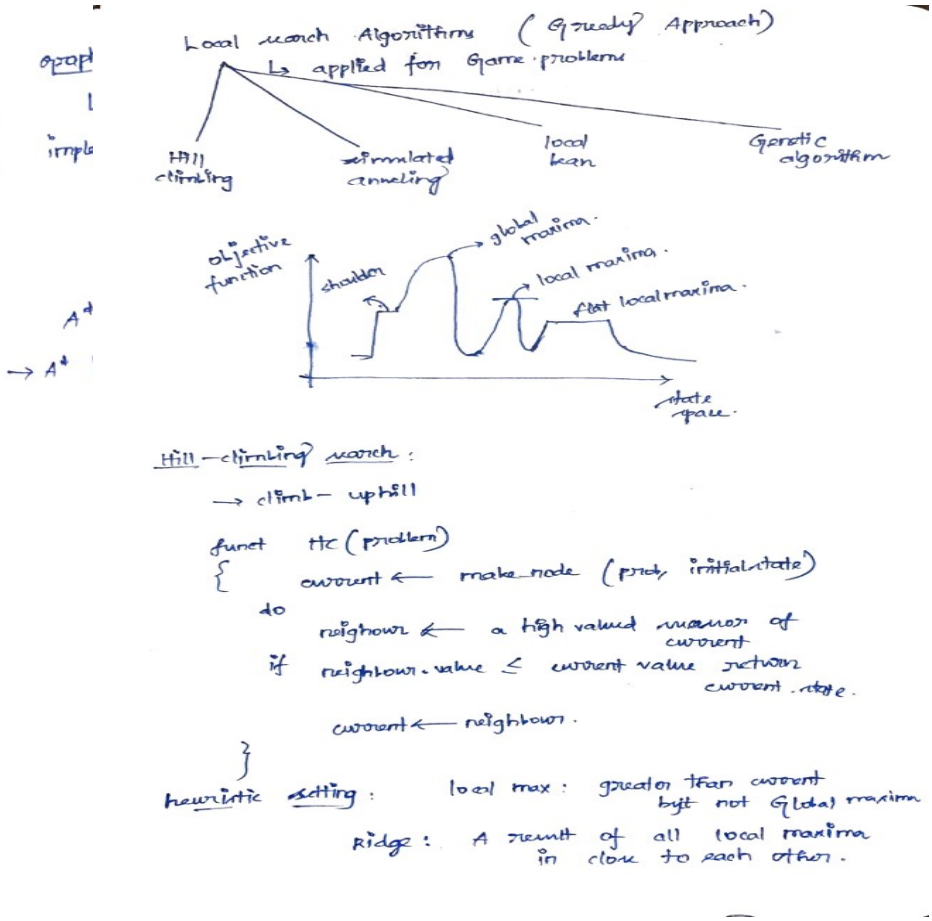
↳ new node can't be added without dropping an existing node.

→ backtracking is not possible as we have dropped few nodes from trees.

Trivial Heuristics

dominance function: $h_a \geq h_c$ if

$\forall n : h_a(n) \geq h_c(n)$



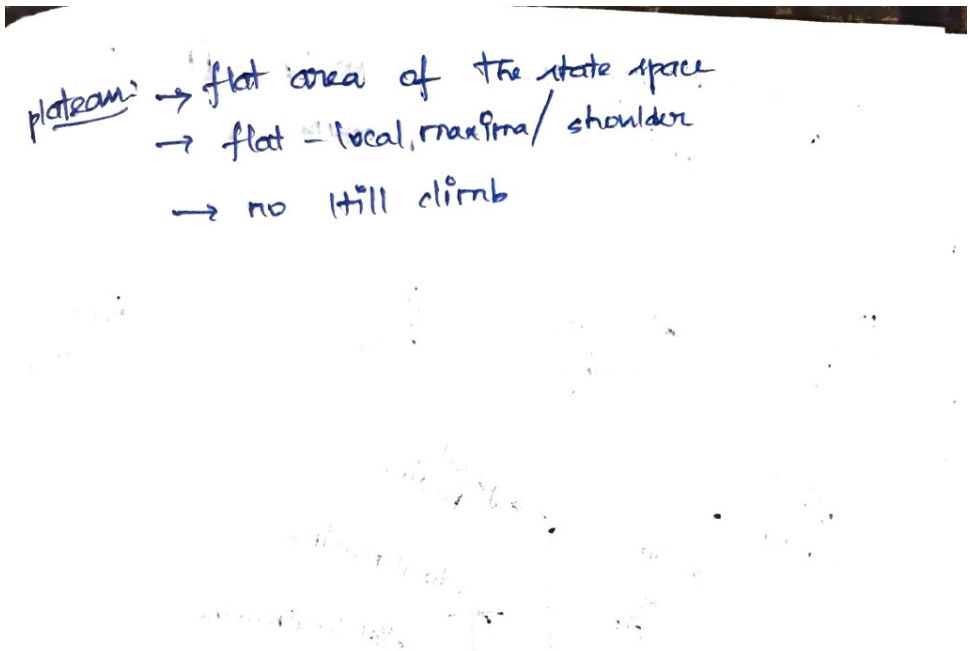
Hill-climbing search:

→ climb - uphill

```
funct HC (problem)
{
    current ← make_node (prob, initial_state)
    do
        neighbour ← a high valued successor of current
        if neighbour.value ≤ current value return current.state.
        current ← neighbour.
}
```

heuristic setting: local max: greater than current but not Global maxima

Ridge: A result of all local maxima in close to each other.



23rd August 2021 – Notes Review by Group 10

Bokam Pavan- 106119024
Poduri Sri Sai Kesava- 106119092

- Memory Bounded herustic search (explanation missing)
- Even memory bounded herustic search is also not a conducive solution since depth has been fixed.
- RBFS approach-Greedy approach.
- For herustic function example (explanation needed).
- Herustics form a semi-lattice(missing).
- A* graph search example (state space graph -> search tree) (missing).

- Optimality of A* Graph search(explanation missing).
- In SMA*-new node can't be added because of limitation of memory.
- Difference b/w RBFS,SMA*,A* search –i.e,whether expanding all nodes,some nodes depends upon these algorithms.

25th August 2021 – Notes by Group 13

Anirudh V S – 106119012
Rinish Sam I – 106119102

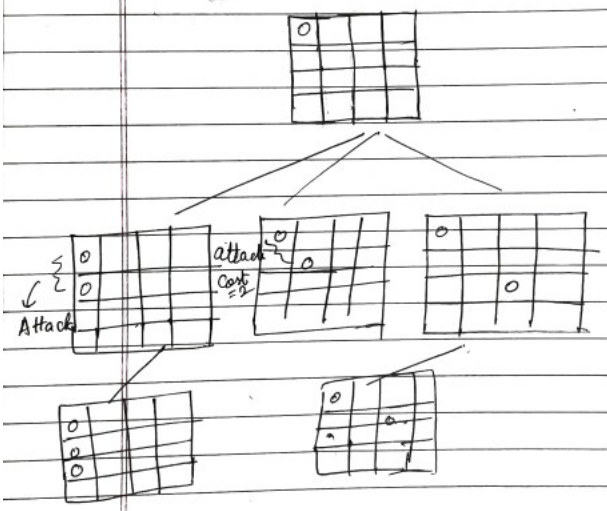
Notes 1061190

ANIRUDH.V.S 106119012
RINISH SAM I 106119102

25th Aug 2021

Hill climbing Problem with Greedy Approach is we may get trapped in local maxima

4 Queen Problem



Heuristic function = No of pair of queens that can attack each other if you start from particular point

Leftmost / rightmost subtree will have maximum heuristic as queens are in last column Intermediate states will have lesser heuristic since ~~not~~ ~~of~~ less queens attack each other

We need to start from some point and move in such a way that objective function will be less.

Minimize (Heuristic)

Draw Backs:

→ Certain possibilities have to be ruled out and expansion is not needed.
But to find heuristic value we have to expand the entire state space graph.

→ Might end up with local minima.

How many pairs of queen attack each other in complete state space graph

18	12	14	13
14	16	13	15
14	12	18	13

Sideways or ridges may be present so this is not a good local search algorithm.

Useful for problems that do not have complete state space graph.

More Search Methods

→ Local Search
  i) Hill Climbing
  ii) Simulated Annealing
  iii) Beam Search
  iv) Genetic Search.

→ Local Search in Continuous Spaces
→ Searching with non deterministic Actions
→ Online search

Decision Tree

Local Search Algorithms.

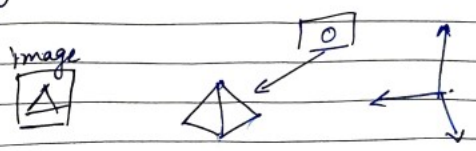
Useful when State formulation can be expanded

Start somewhere and go to soln from there. Perform local search wrt current state to decide next state.

Pose Estimation Example

Given: Geometric Model 3D and 2D

Geometric Model 3D → 2D image
To determine position and orientation of object wrt the camera



Page

Choose a random point and keep altering angles and position to get to goal state
(x, y) coordinate

Successor fuction → intelligence aspect of Hill climbing search
→ be conservative enough to preserve good portions of soln
→ liberal enough to allow state space to be preserved without degenerating into random walk.

Hill Climbing Algo
Works on single Hill
Need to restart if there are multiple hills

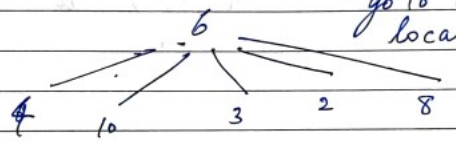
Problems
→ local maximum
→ plateaus
→ ridges

Not complete/optimal
No memory probs.

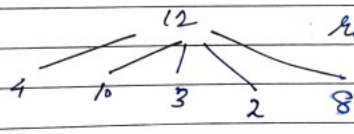
AI Hill Climbing

Steepest Ascent Hill Climbing
current ← node
loop do
neighbour ← highest valued successor
if neighbour <= current.val then
ret cur.state
current ← neighbour
end loop

end loop



go to 10 since it is local max.



redefine heuristic since we cannot move to next higher state

Variations In Hill Climbing

→ Stochastic Hill Climbing → Choose random uphill

→ First choise hill climbing – Create successor at random

→ Random restart hill Climbing – Random initial state and keep climbing high

Simulated Annealing

In metallurgy → heating to high temp and suddenly cooling

General Hill Climbing doesn't come ↓

Σ local (maxima/minima) != Global maxima/minima

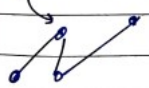
But this may not always be correct

Simulate Anneling is variation of HC [Hill Climbing]

. up is good.

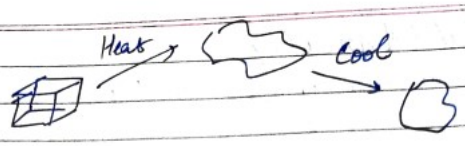
We travel downhill as well as uphill.

If there is ridge/plateau try to come down and go up.



→ Tries to explore enough of state space early on ⇒ final soln is less sensitive to start.

→ downhill moves ⇒ find better uphill move



Probability of moving to higher energy state over lower energy

$$P = e^{\frac{-\Delta E}{kT}}$$

$\Delta E \Rightarrow$ the change in energy level

$T \Rightarrow$ temperature

$k \Rightarrow$ Boltzmann's Constant

Moving down is probabilistically controlled by current temp & bad moves

$t[1]$, $t[2]$ → temperature schedule
usually $t[k] = 0.9 * t[k-1]$
and $t[1]$ is high

$E \rightarrow$ quality measure of state

Goal ⇒ maximize E

Sim Annealing Algo:

1. Current = random state, $k = 1$
2. if $T[k] = 0$; stop
3. Next = random next state
4. if next is better move there.
   if next is worse
   Delta $= -\Delta E = E(\text{next}) - E(\text{current})$

$$P = e^{\left(\frac{\text{Delta}}{T[k]}\right)}$$

5. $k = k + 1$

Discussion

① no guarantee of good soln.

② if T is large $e^{\frac{-\Delta E}{T}} = e^{0} = 1$
so any option is fine

③ if T is small then $e^{-\text{int}}$ so avoid most bad moves

④ if $T=0$ ⇒ do simple hill climbing

③ Execution time depends on schedule and mem use is trivial.

Initially temp is high

Over time temp ↓ $\left(t[k] = 0.9\, t[k-1]\right)$

So, KT ↓

So, $\frac{\Delta E}{KT}$ ↑

So, $\left(\frac{-E}{KT}\right)$ ↓

$e^{-E/KT}$ ↓

So probability of downhill

decreases

ΔE → change in value of objective fn.

$p = e^{-\frac{\Delta E}{T}}$ k is dropped since no physical relationships appear.

Annealing schedule:
Sequence of values of T, ex, T_0, T_1, T_2
We need a schedule for the algo to work

Pseudo Code

current ← start node

for each T on schedule

next ← random successor

evaluate next, if goal, return

if $\Delta E > 0$

then current ← next

else current ← next with probability $p = e^{-\frac{\Delta E}{T}}$

Properties.

If T decreases slowly then Sim Annealing will find global optimum with probability approaching 1.

Applications: VLSI, Airline scheduling.

Probabilistic Selection.

select next with prob P.
Generate random no.
if <= p then select next

T is set of nodes
Can be obtained by heuristic

In TSP we can have heuristic as
min(edge cost (n,), path cost (n,)).
which can be used to get schedule

In airplane problem

heuristics = no of flights on runway
- no of waiting flights

T is time schedule.

Random move is chosen to get global optimum. Best move is not always chosen

Applications of Sim Annealing

Basic Problems

→ TSP
→ Graph Partitioning
→ Matching problems
→ Graph Coloring
→ Scheduling

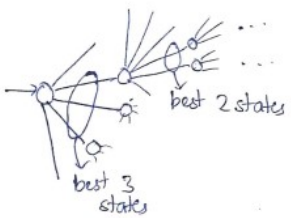
Engineering

VLSI design
→ Placement
→ Routing
→ Array logic imp
→ Layout

~~Facility~~
Facilities layout
Image processing
Code design in Information Tech

Beam search : (local beam search)

* Combination of hill climbing and best first search
* Goes in k directions → remembers k states (cached)
* Starts with randomly generating k states.
  At each step, successors of all the states are generated
  If anything is a goal ⇒ halt
  else, selects tries to select best scenario from each of the successors and moves towards that direction.
* Carried out in parallel ⇒ achieves goal faster.



Stochastic local beam search
⇓
next k-states randomly ~~generated~~ selected instead of best

Local (iterative improving):

* Initial state = full candidate solution
* Greedy hill-climbing
  → if up, do it
  → if flat, probabilistically decide to accept move
  → if down, don't do it
* We're gradually expanding the possible moves

Performance :

- Solves 1000000 queen problem quickly
- Useful for scheduling
- Useful for BSAT
- More time, better answer
- No memory problems
- No guarantees of anything

Genetic Algorithm :

- Analogous to evolution.
- No theoretic guarantees
- Applies to nearly any problem

Terms :

Population : set of individuals
Fitness function on individuals
Mutation operator : new individual from old one
Cross-over : new individuals from parents

Algorithm :

* Population = random set of n individuals
* Probabilistically choose n pairs of individuals to mate
* Probabilistically choose n descendants for next generation (may include parents or not)
* Probabilistically depends on fitness function as in simulated annealing
* How well does it work? - works to some extent and works better with mutation

Scores to probabilities :

Suppose scores of n individuals are
a[1], a[2], ... a[n]
The probability of choosing j^{th} individual

$$= \frac{a[j]}{a[1] + a[2] + \ldots + a[n]}$$

GA Example :

N-queens problem :

→ Use 8 digits : 1, 2, 3, 4, 5, 6, 7, 8
to specify position of queen in

corresponding column

(eg) (3,2,5,6,1,4,7,8)

individual:

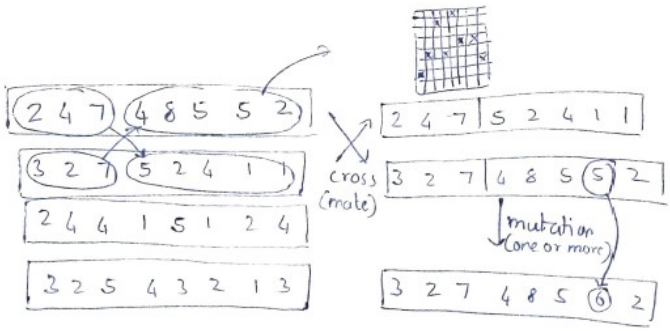
→ array indicating column where ith queen is assigned.

mating: cross-over

fitness (minimize): no. of constraint violations (no. of queen pairs that can attack each other)

GA function optimization:

* Let $f(x,y)$ be function to optimize (fitness function)
* Domain for x & y is real number b/w 0 and 10
* Say the hidden function is
  - → $f(x,y)=2$ if $x>9$, $y>9$
  - → $f(x,y)=1$ if $x>9$ or $y>9$
  - → $f(x,y)=0$ otherwise



GA Works Well here:

→ Individual: point = (x,y)

→ Mating: something from each so: mate($\{x,y\}$, $\{x',y'\}$) is $\{x,y'\}$ and $\{x',y\}$

→ No mutation

→ Hill climbing does poorly, GA does well in trying to get the offspring

→ This example generalises functions with large arity (no. of arguments)

GA discussion:

- Reported to work well on some problems
- Typically not compared with other approaches, (eg) hill climbing with restarts
- Works if "mating" operator captures good substructures → capture offspring properly ⇒ good result

Algorithm process (sequence):

* start with random population of states
  → representation serialized
  → states are ranked with "fitness function"
* Produce new generation
  → Select random pair(s) using probability
    probability ∝ fitness
  → Random choose "crossover point"
    offspring mix halves
  → Randomly mutate bits

Pseudocode:

Given: population P & fitness function f

→ repeat
  * newP ← empty set
  * for i = 1 to size(P)
    x ← RandomSelection(P, f)
    y ← RandomSelection(P, f)
    child ← Reproduce(x, y) } → might lead to unnecessary state as well
    if (small random probability) then
      child ← Mutate(child)
    add child to newP
  * P ← newP
→ until some individual is fit enough or enough time has elapsed
→ return best individual in P according to f

Using Genetic Algorithms

2 important aspects:

1. How to encode your real-life problem
2. Choice of fitness function

Example:
in case 8 queens problem, how to encode states of 8 queens?
binary? → more no. of bits

Research example:

→ We want to generate new operator for finding interesting points in images

→ Operator will be some function of a set of values $v_1, v_2, \ldots v_k$

→ idea: weighted sums, weighted mins, etc.
$a_1, \ldots a_k$ $b_1 \ldots b_k$
to generate next sample set

Searching with Non deterministic Actions:
(not in syllabus)
↓
result of actions can vary
(erratic vacuum world)

25th Aug 2021 Evaluation

Group no: 27

Members: Deekshika Tomar (106119026), E.Vidhyadharan (106119034)

1) Variations in hill climbing could have been elaborated further.

2) Apart from this the notes are perfect and covered every single line said in the lecture and PPT shown.

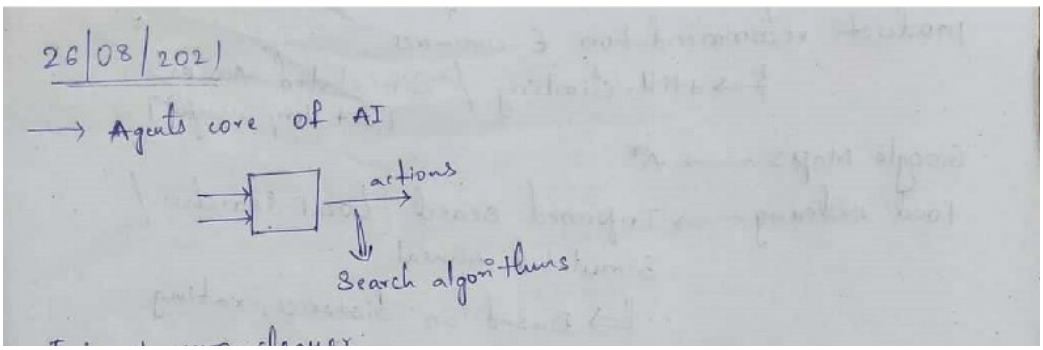
26th August 2021 – Notes Group -26

106119082 – Ushasree

106119086 – Niranjana D P

Link: https://drive.google.com/file/d/15Dqzfw7FAAubbJNLFzK5jCh5d2qHKSXE/view?usp=sharing

AI Notes
27.08.2021



Ex. Vaccum cleaner

Ex. Automatic Google news

preferences → sports
→ AI, ML
→ central govt
→ JEE Main

It will try to push these articles first

If we stop looking at articles for some period, it will push that down.

There will be search algorithm for this.

1. Depth limit → time will be the limit
2. local Beam Search.

No A*, because no optimization.

Ex. News feed FB → depth limit, local Beam.

Advertisements in FB/youtube → Random Search, Iterative deepening + BFS

recommendations in youtube → ~~BFS~~, ~~A*~~ Iterative deepening

Linkedin – Employers looking for good resume
↳ BFS, A* → heuristic func ⇒ Experience
⇒ skill set.

Scanned with CamScanner

product recommendation E-commerce
↳ Hill-climbing / Simulated Anneal [Not very useful]

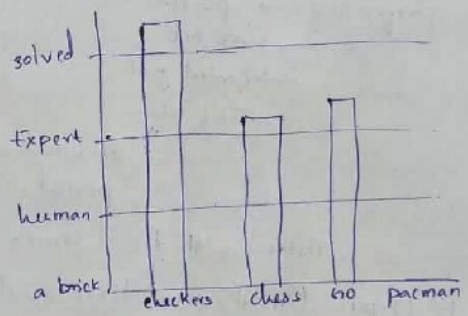
Google Maps → A*

Food ordering → Informed search / Local Searches / Simulated Anneal.
↳ Based on distance, rating.

restuarants → Similar to newsfeed.

→ Adversial Search:

* checkers → 2007 solved
1950 1st comp ~~game~~ player
* chess → Deep blue — 1997
* GO → 2016 → Alpha GO defeats human champion.
Monte carlo tree search.
* pac Man → not found yet
* Think opposite player as expert



* Games
deterministic

- stochastic.

* Want algorithm for calculating a strategy which recommends a move from each state.

Search algorith [start ---> goal]

* Deterministic Games.
  - No random move is possible
  - Next move is based on others move.

States: S [start S_0]

players: P = {1, --, N) ⟶ which player can move.

Actions: A (depend on player / state)

⇓ return set of legal moves

~~Term~~ Transition function: S x A ⟶ S

⇓ defines the ~~the~~ result of a move.

Terminal Test : S ⟶ {t, f}

⇓ True → game is over.

False → game is not over.

Terminal utilities: S x P ⟶ R

called as objective / pay off function.

Defines numeric values for a game.

- In chess ⟶ utility func ⟶ 0, +1, +1/2
- Zero-sum game ⟶ utility func ⟶ 0, +1, −1

Total pay off = 0.

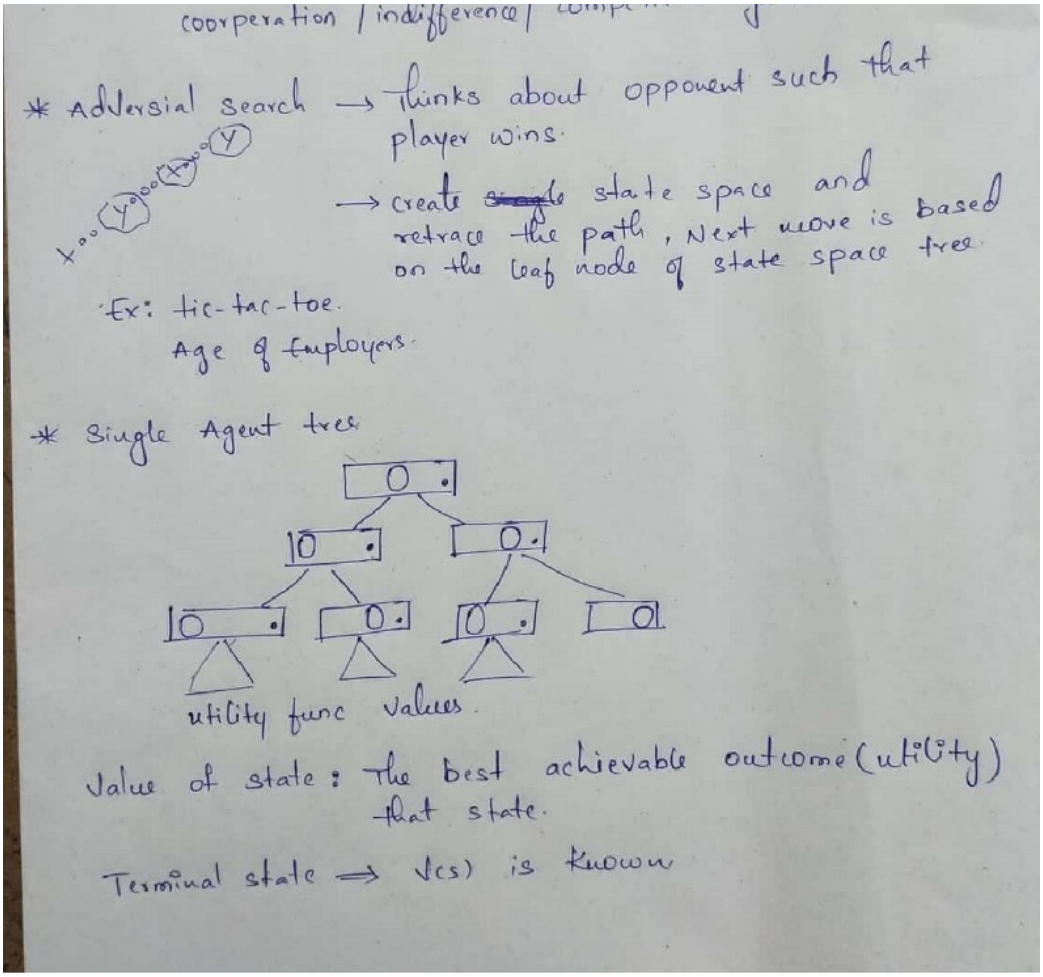
When one utility ↑ses, other ↓ses.

competition oriented game.

- constant sum / General game.

Total pay off ≠ 0., but maximum values

Independent utility.



Evaluation of 26th August 2021
By Group No: 33
Vishal Jaiswal : 106119144
Ritu Gain : 106119106

Good sides:
Algorithm used in social media's written precisely.
Everything has been written properly except few missing things that I've written bellow.

Missing / Bad sides:
Definition and types of Agents Missing.
Reason behind using search algorithm is missing.
Explanation and search strategy about Vaccum Cleaner example of search algorithm (Collection of dirt) is missing.
Reason behind using depth limited search in Google News is missing.
Didn't mention about combination of depth limited search and local beam search which is an effective search algorithm for Google News.
Important information about Game playing state of the Arts graph are missing.
For example: positions in graph, reason for keeping PacMan's place as blank in the graph.
Didn't mention about Adversarial Game and it's definition and search strategy of adversarial search or Game search.
Didn't write properly about Zero Sum games.
For example: Objective values for zero sum games, what opponents does when value maximizes.
Characteristic of Zero Sum Game is Missing. (Adversarial, pure competition)
Didn't mention anything about Single agent Tree except the picture!
Algorithm applied in Value of state is Missing. (Minimax Algorithm).
Information about terminal and non-terminal states are not given properly.

30th Aug 2021

Group no: 35

Members: Harsh Khandelwal (106119048), Aryan Wadhavekar (106119148)

Notes

Evaluation of 30th August 2021
By Group no:30
Aravind Tammireddy - 106119016
Korada Srikanth - 106119062

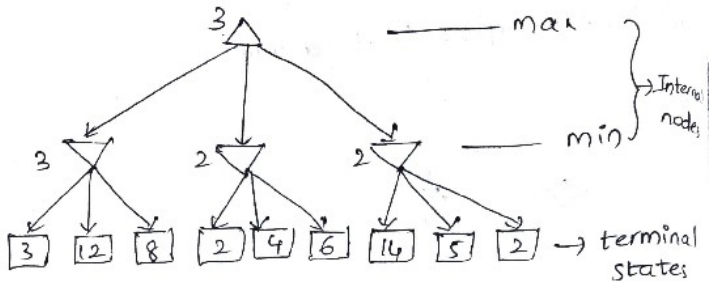
Didn't explain about utility function i.e. utility(S,P)
Didn't explain about utility values of terminal states in tic tac toe game
#Minmax gives optimal only against perfect players not cheaters this point is missing
#Didn't explain about other examples of minmax algorithm included only tictactoe example
Explanation about Time complexity of minmax is missing
#The explanation of tic tac toe example in minmax also incomplete.
Didn't explain the need for pruning algorithm(to decrease the unnecessary states in minmax
#Not a clear explanation about alpha-beta, not explained all the examples told in class for alpha beta pruning
#Apart from this the notes are perfect and covered every single line said in the lecture and PPT shown

31st Aug 2021
Group No: 29
Members: Y Bhargavi(106119150), Y Pushpanjali(106119152)

Adobe
Scan 31

Minimax Example :

* It is a 2-player game and typically represented as 'ply'.

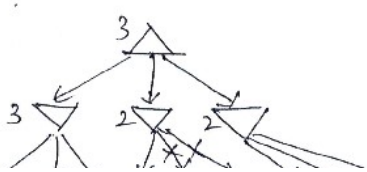


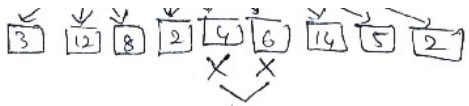
* This is a zero sum game.
* +1, -1, 0 are the utility values for win, lose, and draw respectively.
* It is not necessary to expand every node instead we use pruning known as α-β pruning.

α-β pruning :

* It tries to expand from the left most node and depending upon the min and max level, it decides whether the other child nodes to be expanded or not.

example :





These two nodes will be pruned.

Starting from the left most node '3', nodes 3, 12, 8 gets expanded to find min value at min level and the corresponding max value at max level.

Starting from node 2, as 2 is less than the value at max level (i.e., 3), the siblings '4' and '6' won't be expanded.

Starting from 14, since $14 > 3$, next node 5 will be expanded and since $5 > 3$, next node 2 will be expanded.

∴ Only 4 and 6 will be pruned in the above example.

Minimax Properties:

* These games are optimal against a perfect player.
* These are known as Perfect decision games.
  Because, both the players get to know the moves of other player.
  Examples: Chess, checkers, monopoly
* Imperfect decision games:
  In this type of games, opponent will not get to know the complete state of the game.
  Examples: Card games,

Minimax Efficiency:

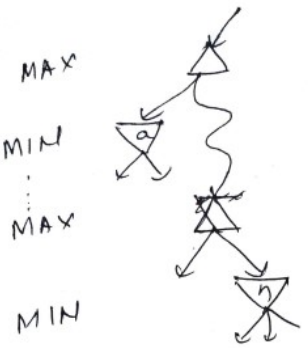
* Efficiency of minimax is just like (exhaustive) DFS.
* Time Complexity: $O(b^m)$
* Space Complexity: $O(bm)$ where b is branching factor and m is depth of ~~tree~~ situation.
* Example: for chess, $b \approx 35$, $m \approx 100$.
  Since m is 100, it is very difficult to expand. So

we go for the pruning.

α, β pruning doesn't work in all cases. It depends upon which node is getting expanded first and at what level.

α-β Pruning: → minimizing / → maximizing

- General configuration (MIN version)
  - We're computing the MIN-value at some node n
  - We're looping over n's children
  - n's estimate of the children's min is dropping.
  - Who cares about n's value? MAX
  - Let α be the best value that MAX can get at any choice point along the current path from the root
  - If n becomes worse than α, MAX will be avoid it, so we can stop considering n's other children (it's already bad enough that it won't be played)
- MAX version is symmetric.



α-β implementation

α: MAX's best option on path to root

β: MIN's best option on path to root

```
def max-value(state,α,β):
    initialize v=-∞
    for each successor of state:
        v=max(v,value(successor,α,β))
        if v≥β return v
        α=max(α,v)
    return v
```

```
def min-value(state,α,β):
    initialize v=+∞
    for each successor of state:
        v=min(v,value(successor,
                      α,β))
        if v≤α return v
        β=min(β,v)
    return v
```

Alpha-Beta Pruning Properties:

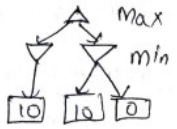
- The pruning has no effect on minimax value computed for the root!
- values of intermediates nodes might be wrong

- Good child ordering improves effectiveness of pruning

function α-β(state) returns an action

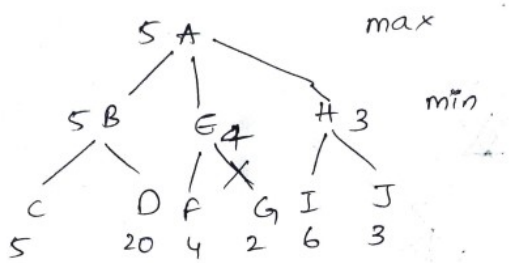
v → max-value(state, -∞, ∞)

return the actions in action(state) value v



Examples:

①



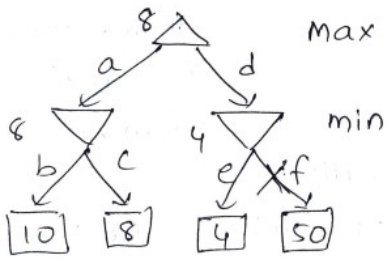
Starting from the left most node 5, as it is the first node both 5 and 20 gets expanded. And we get value of 'A' and 'B' as 5.

Starting from '4', since 4<5, we no need to expand G (value 2).

Starting from node '6', Since 6>5, we need to expand node J (3) also.

So only 'G' will be pruned.

②

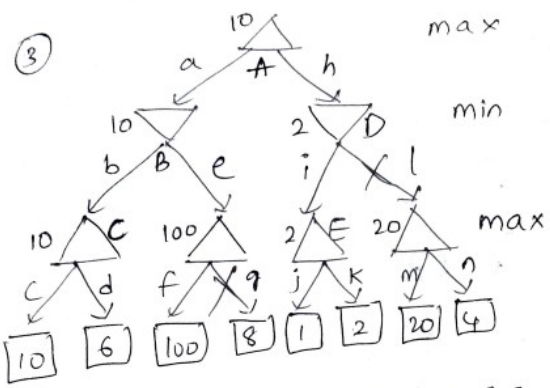


Starting from 10, as it is the first node both 10 and 8 gets expanded. And we get 8 at both min and max node.

Starting from 4, since 4<8, 50 will not be expanded.

So only f will be pruned. (node 50)

③



from 10 nodes A, B and C get

Starting from
10 & 6 gets expanded

Starting from 10, both 10 & 6 get expanded and we get value at C as 10. Values at A and B will also be 10. and starting from 100, since 100 > 10, 8 doesn't get expanded.

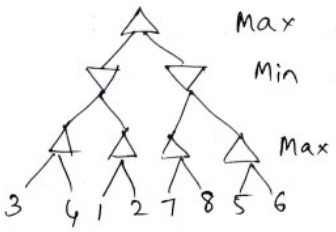
Starting from 1, both 1 & 2 get expanded and we get values at D and E as 2.

Since 2 < 10, right subtree of 'D' won't be expanded.

i.e., path 'l' won't be expanded.

So, g and L(m,n) will be pruned.

④



In this example, none will be pruned. Because the most favourable nodes for both are explored last (i.e, in the diagram, are on the right-hand side).

winning state, we will reaching non-winning states

eg: Tic-Tac-Toe:

Games:

	Deterministic	Stochastic
Perfect information	Chess, checkers, go	backgammon, monopoly
Imperfect information	blind tictactoe, battleships	bridge, poker, scrabble, nuclear war

All card games are imperfect information and stochastic.

Resource Limits:

In realistic games, we will not search to leaves.

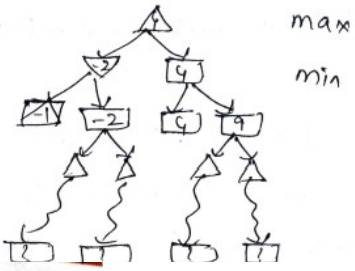
Solution: Use Depth-limited search rather than a depth first search (dfs).

i.e, we search only to a limited depth.

Or we can use iterative deepening search which reduces space and time complexity.

⇒ We will replace terminal utilities with an evaluation function for non-terminal positions.

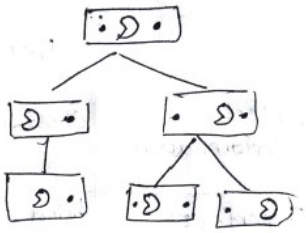
* Guarantee of optimal play is gone.



* We can increase depth using iterative deepening ~~search~~ algorithm.

Pacman : (A danger of replanning agents)

In this game, score is obtained by eating dot.



Once it eats the dot, there is no point-scoring opportunity.

So, waiting seems just as good as eating : he may go east, west in the next round of replanning.

⇒ It is a game of replanning.

In imperfect decision algorithms and variations to α-β pruning, we use evaluation functions.

Evaluation functions :

- These are typical imperfect-(real-time) decision oriented games.
- Can be used for perfect decision games ~~also~~.
- Minimax algorithm generates the entire game space.
- Alpha-Beta pruning still has to search all the way to the terminal states.
- Algorithms should cutoff the search earlier and apply heuristic evaluation function. Turns non-terminal nodes to terminal leaves.
- Replace utility function by a heuristic function which estimates the position utility and replace the terminal test by a cutoff test.
- Returns an estimate of the expected utility of the game from any given position.
- Order the terminal states in the same way as the true utility function
- Computation must not take too long

Try to apply a cut-off test rather than a terminal test.

for example, in α-β search:

```
α-β Search
{
  V → max-value(state, -∞, +∞)
  {
  max-value (state, α, β)
  if terminal-test(state) return utility(state)
V = -∞
  max (v, min value(successor)).
```

In modified α-β, we use cut-off test instead of terminal test.

Cut-off can be depth or heuristic or terminal value based on our assumptions.

Heuristic-Minimax (S, d): S - state, d - depth

EVAL(S) if Cut-off test(s,d)

max H-Minimax (Result (S,a), d+1) if player = max

min H-Minimax (Result (S,a), d+1) ∀ player = min

evaluation function depends on features.

for example, in Tic-Tac-Toe game characteristics (features) are

no. of X/O in a column/row/diagonal.

With these features, we give a weightage for each of the particular feature.

Evaluation function:

$$e.Eval(S) = \underbrace{w_1 f_1(s)}_{\text{feature } i \text{ of a particular state } s} + w_2 f_2(s) + \dots + w_n f_n(s)$$

$w_i \Rightarrow$ weightage associated with feature i,

if evaluation function is greater than a threshold value, it is a winning scenario or else a loosing scenario. So it can be used as a cut-off point.

Depth matters:

- Evaluation functions are always imperfect. Because it depends on the weightage given to it.
- The deeper in the tree the evaluation function is buried, the less the quality of the evaluation function

matters.

- An important example of the tradeoff between Complexity of features and Complexity of Computation
- These are known as imperfect decisions.

Problems of using evaluation functions:

① Quiscent :

→ Variation to it is known as quiscent search.

→ Quiscent : states are unlikely to exhibt wild swings.

We can apply quiscent search to restrict to only certain set of moves.

If we know that we are going to win, it is known as quiscent scenario.

→ Using quiscent search, we can apply certain possible set of moves inorder to go to a winning state.

→ This is also a part of evaluation function and this can also be used as a cut-off test scenario.

Eval function : Cut-off test(s).

② Horizon Effect :

→ Damaging scenario

→ Horizon is the sky's bottom most point.

→ This is very difficult to get rid off

→ It typically arises when program is facing opponent's move that causes lot of damage to the player.

→ In this scenario, we try to defer the move in order to escape from the horizon effect. But we cannot completely escape from horizon effect.

→ This is also one of the parameter for determining the eval function.

eg: online chess.

⇒ Quiscent and Horizon effect are opposites of each other.

Solutions to avoid horizon effect:

① Singular extension : It is a move that is clearly better than all other moves.

So we defer the move by assuming that the opponent might actually do a mistake.

forward Pruning :

α-β do backward pruning.

We try to prune a move at a given node immediately without even considering that.

Beam Search :

Search in K-path and choose best among the 'K' paths.

In forward pruning, we try to do beam search.

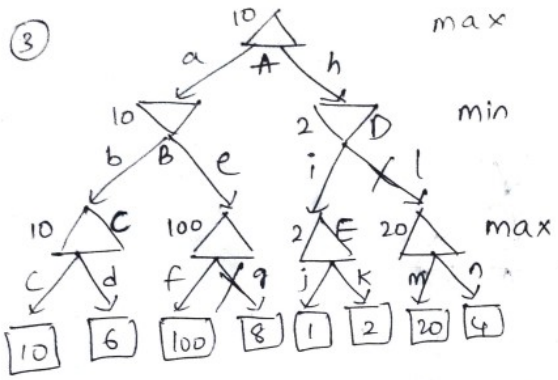
Probabilistic Cut → approach to forward pruning.

↳ It is a forward pruning version of α-β pruning.

It depends upon previous experience and uses Statistics and apply probability of a winning move and probability of a loosing move and it tries to use a maximum probability value in order to prune other branches.

⇒ It cannot get perfect game design, it needs to learn from

③



Starting from ~~10~~ nodes A, B and C get 10 & 6 gets expanded

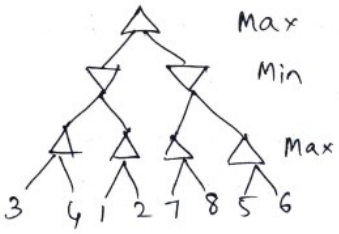
Starting from 10, both 10 & 6 get expanded and we get value at C as 10. Values at A and B will also be 10. and starting from 100, since 100 > 8, 8 doesn't get expanded.

Starting from 1, both 1 & 2 get expanded and we get values at D and E as 2.

Since 2 < 10, right subtree of 'D' won't be expanded i.e., path 'l' won't be expanded.

So, q and L(m,n) will be pruned.

④

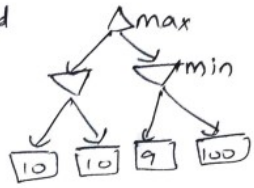


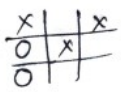
In this example, none will be pruned. Because the most favourable nodes for both are explored last (i.e., in the diagram, are on the right-hand side).

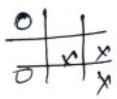
previous experiences.

Worst case Vs. Average case:

Idea: Uncertain outcomes controlled by chance, not an adversary







Expectimax Search:

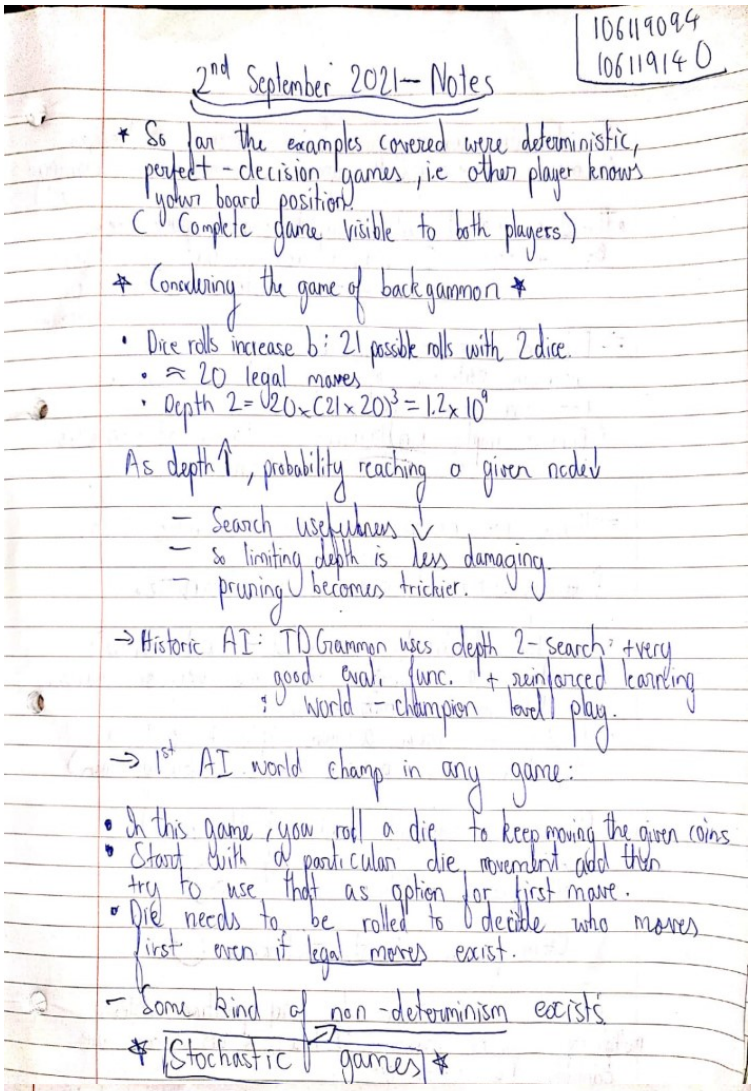
→ Probability based search

→ typically applied for imperfect decision games and partially observable games.

Evaluation of 31st August 2021 by Group No: 34
Vamsi Sunkavalli : 106119124
Pranay Lanka : 106119066

Remarks :- Covered each and every point correctly in detail with respective examples.

2nd September 2021
Group No: 18
Pranav Somaiah- 106119094
Vaasu Gambhir- 106119140



106119094
106119140

2nd September 2021 – Notes

* So far the examples covered were deterministic, perfect-decision games, i.e other player knows your board position.
(Complete game visible to both players.)

* Considering the game of backgammon *

- Dice rolls increase b: 21 possible rolls with 2 dice.
  - ≈ 20 legal moves
  - Depth 2 = $20 \times (21 \times 20)^3 = 1.2 \times 10^9$

As depth ↑, probability reaching a given node ↓
- Search usefulness ↓
- so limiting depth is less damaging.
- pruning becomes trickier.

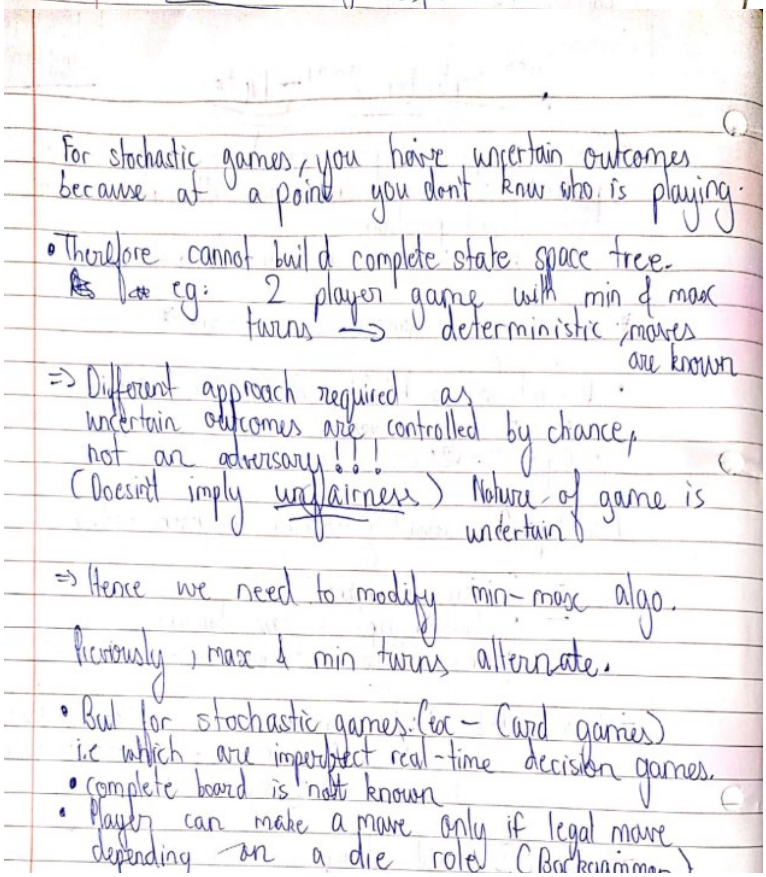
→ Historic AI: TD Gammon uses depth 2-search + very good eval. func. + reinforced learning : world-champion level play.

→ 1st AI world champ in any game:

- In this game, you roll a die to keep moving the given coins
- Start with a particular die movement and then try to use that as option for first move.
- Die needs to be rolled to decide who moves first even if legal moves exist.

– Some kind of non-determinism exists

* Stochastic games *



For stochastic games, you have uncertain outcomes because at a point you don't know who is playing.

- Therefore cannot build complete state space tree.
  eg: 2 player game with min & max turns → deterministic, moves are known

⇒ Different approach required as uncertain outcomes are controlled by chance, not an adversary!!!
(Doesn't imply unfairness) Nature of game is uncertain

⇒ Hence we need to modify min-max algo.

Previously, max & min turns alternate.

- But for stochastic games (ex – Card games) i.e which are imperfect real-time decision games.
- Complete board is not known
- Player can make a move only if legal move depending on a die role (Backgammon)

Main goal - move all coins off the board.

One person - Clock wise ↻

Other person - Counter-Clockwise ↺

And we can move to any position unless multiple opp. piece exists. If there is an opponent we capture & start over.

* We need a die roll to start, hence very different from deterministic games (chess, checkers, tic-tac toe etc)
* Nature of game itself is stochastic.

Expectimax (or expectiminimax) Search

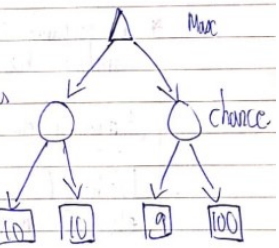
Why wouldn't we know result of an action will be?

→ Explicit randomness : Rolling) dice.

→ Unpredictable opps. : ghosts respond randomly (keep moving till opp found).

→ Actions can fail : when moving a robo, wheels may slip which calls for a restart.
(Some uncertainity exists)

* Values should now reflect avg-case (expectimax) outcomes, not worst case (minimax) outcomes



* Expectimax Search - Compute Avg score under optimal play.

→ Max nodes as in minimax search

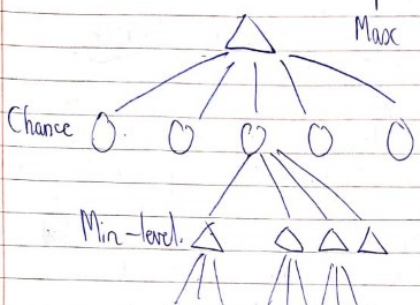
→ Chance nodes (probabilistic nodes) are like min nodes but the outcome is uncertain (We associate a probability value)

→ Calculate their expected utilities, i.e take weighted average (expectation) of children

Σ wifi

* We calculate expected value as some probability is associated with it.

Hence to construct state space tree.



* We expand chance giving us the least

value here x

Expectiminimax(S) Algo

Utility(S) is typically determine if Terminal_Test(S)
else if max_node ( ) Expectiminimax(Result(S,a))

Suppose if Player(S) = Max_Node

min_ (") ; if Player(S) = Min_

If player is chance,

$\sum$ P(particular dice roll(r)) x Expetimax(Result(S,r))

Pseud Code: def value(state):
if state is terminal : return state utility
if next agent is MAX: return max-value(state)
if next agent is EXP: return exp-value(state)

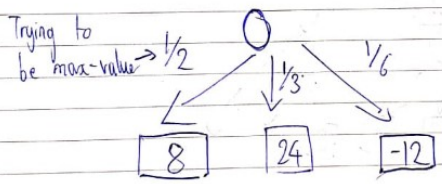
def max-value(state):
initialise v = −∞

for each successor of state:
v = max(v, value(successor))

return v.

def exp-value(state):
Initialise v = 0

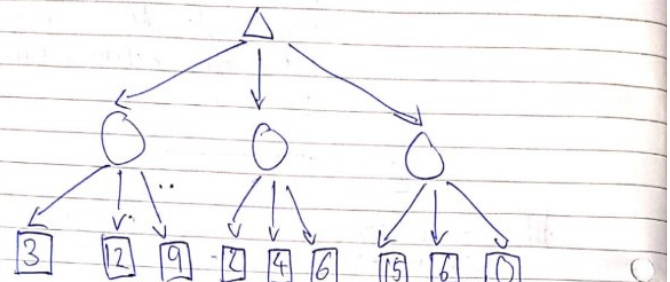
for each successor:
{ p = probability(successor)
v+ = p * value(successor)
return v



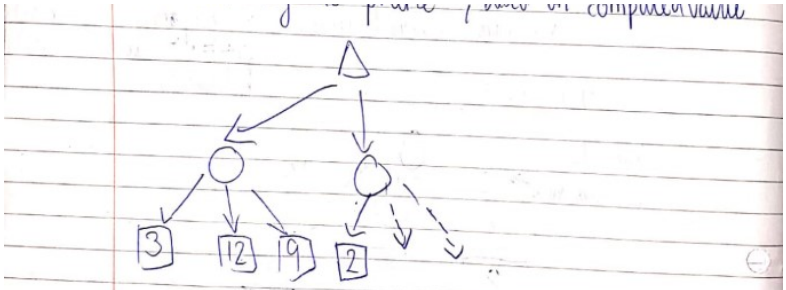
V = (1/2) 8 + (1/3) 24 + (1/6) -12

= 10 (Value associated with particular node)

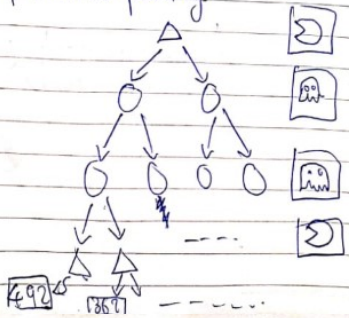
Expectimax Example:



We can try to prune based on computed value

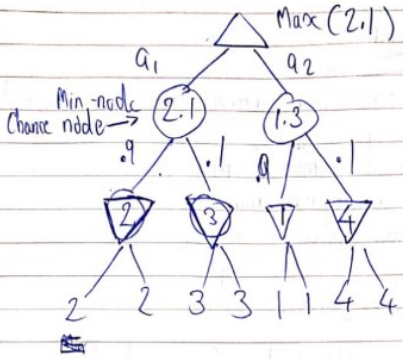


Depth-limited pruning



Determine limit of depth or iterative deepening also possible

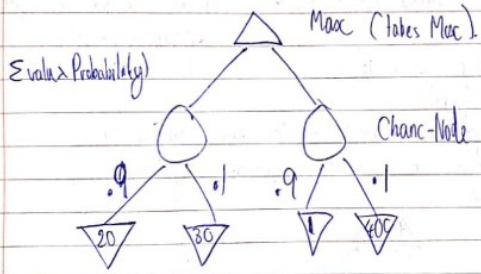
Another Example:



Since we know terminal value, we can use previously discussed computation.

* $2 \times 0.9 + 3 \times 0.1 = 2.1$
* $1 \times 0.9 + 4 \times 0.1 = 1.3$

← Only 1 option



Probabilities

Random Variable – Event whose outcome unknown
Probability distribution – assignment of weights to outcome
Probability density funct.

Ex: Traffic on freeway
Random V: T = whether there is traffic
Outcomes: T in {none, light, heavy}
Distribution, P(T=none) = 0.25, P(T=light) = 0.50
P(T=heavy) = 0.25

* P. over all possible outcomes sums to 1.
* P. are always non negative.

* Try to design Backgammon.

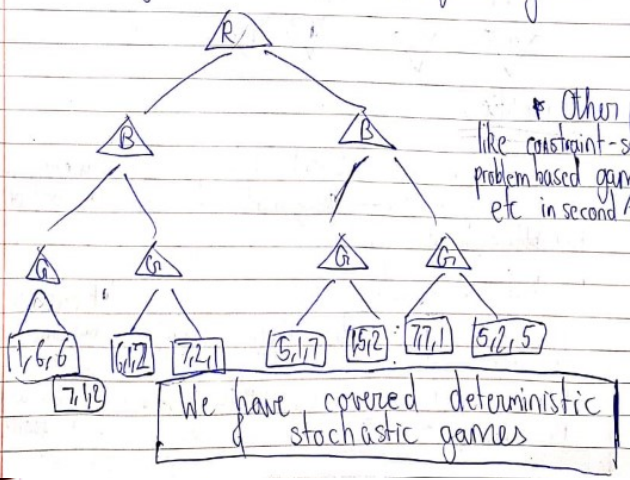
Multi-agent utilities

* What if not zero-sum game or has multiple dem?

Not alternating b/w max & min.
Each tries to max or minimize...
(3 or more values instead of 2)

Generalization of min-max.

* Terminals have utility tuples (unlike tic tac toe with only 2 players, +1, -1, 0)
* Node values are also utility tuples.
* Each player maximises its own component.
* Can give rise to cooperation, dynamically.



* Other problems like constraint-satisfaction problem based games etc in second AI course

We have covered deterministic & stochastic games

---

①

M T W T F S S
Page No.:
Date:
YOUVA

Knowledge base agents

Knowledge base (KB)	Inference engine (IE)
→ has domain specific content	→ domain-independent algorithms :- They are applied to domain specific content & hence are modified to a small extent.
→ contains all possible information about the domain content at focus.	
→ known as "set of sentences in formal language"	

Eg :- KB :- a KB that answers questions from data structures.
(contains all possible DS topics)

→ so inference engine can 'infer' multiple strings like

"binary heaps are binary trees"

or "vector can be a dynamic implementation of an array"

these seem domain-specific one can apply the IE by tweak it for this

domain, but most algorithms always follow a more generic approach.

(2)

=> Knowledge base :-

→ 'set of sentences in a formal language'
→ sentence :- corresponds to particular representation.

eg: a) A = B + C;
b) The book is on the table.

↳ Thus, we can use '...' :: a declarative / procedural approach for building an agent :-

→ declarative approach :-
- constructing a KB by telling it or encoding the rules to or what it needs to know,
- feeding it facts.

domain: predicate / propositional logic

→ procedural approach :-
- encodes desired behaviours (or expected behaviours) directly as program code

=> Quoting other previous example :-
→ we can add things like :-

"The leaf nodes of a KB tree are typically black nodes"
↳ it will try to give it's own inference.

(3)

=> Hence it will 'ASK' itself what to do ↳ what it knows.
answers are taken from KB

=> It's like we provide it a repository of documents & we 'ask' certain things about it & it will give us respective results.

→ it will tell the KB which action was chosen.

- Knowledge level
→ need to specify, what the agent knows & the it's what can it's plans regarding the same (goals).
↳ we do not worry about
- Implementation level implementation level.

→ actual action & sequence of steps required to achieve the goals.
→ Basically it makes use of the provided

content it knows.

→ combination of both the levels are preferred.

Eg:- Knowledge level:- contents of DS as given as a sequence of steps./ statements goal is to answer all questions related to this.

→ This can be done at one level, wait for the question to be asked, & then give respective answers.

or else:-

the agent can learn from the KB & construct intermediate information.

↓

this is referred when a question is asked.

Google search eg:- crawls

crawler ~~scatter~~:- ~~scatter~~ through all websites to gather information & create index to refer to that information

↳ this is intermediate information.

⇒ ∴ this is combination of implementation + knowledge level

as there is an algorithm that manipulates information & then gives us result.

→ combine & make sense of facts to create intermediate ~~info~~ representations,

which is used by inferencing systems at a later point of time.

↓

this is faster

→ this is called as reasoning:-

as we are reasoning from known facts to get to the intermediate representation.

reasoned info is stored, rather than KB.

→ Simple knowledge Base agent:-

function KB-Agent (percept) ← input given returns an action

static: KB, a knowledge base

t, a counter (initially 0), to indicating time.

→ converts input to logical sentence

we provide the input to the KB ← TELL (KB, MAKE-PERCEPT-SENTENCE (percept, t))

& ← action ← ASK (KB, MAKE-ACTION-QUERY(t))

converts action to logical sentence

TELL (KB, MAKE-ACTION-SENTENCE (action, t))

t ← t+1

return action

each KB & it will give an action

& we update the KB based on the action (as told before).

⇒ ~~The agent must be able~~ to :-

⇒ KB agent returns answer as an action. based on the input.

Page No.: Date: YOUVA

⇒ The agents must be able to:-

→ represent states & actions (by using DS as intermediate information)

→ incorporate new percepts / inputs

→ update internal representation (KB

→ deduce hidden properties of world / environment, ~~& deduce actions~~

→ deduce appropriate actions.

⇒ WUMPUS WORLD (example)

→ a cave with partitioned rooms.

→ it has a monster (called wumpus) & is present in one partition, existing that, you can get eaten

→ pit :- absorbs you & you can't come out.

→ . adjacent rooms of pit surrounded by breeze

" " " wumpus " " stench

↳ not at diagonals.

(no: mentions).

→ purpose is to go to the room with gold & safely.

→ he has one arrow & can shoot the wumpus.

→ wumpus can move between rooms.

Page No.: Date: YOUVA

4	Stench		Breeze	PIT
3		Stench Breeze	PIT	Breeze
2	Stench		Breeze	
1		Breeze	PIT	Breeze

1 2 3 4

↳ can be converted to PEAS description :-

- performance measure
  - → gold, ~~score += 1000~~
  - → death, score ~~-=~~ -= 1000
  - → ~~-=~~ step, score -= 1
  - → using arrow score -= 10

- Environment : 4×4 grid of rooms
  - → squares adjacent to wumpus are really smelly
  - → squares adjacent to pit are breezy
  - → glitter is observed if only if gold is in same square.
  - → shooting kills wumpus if you are facing it.
  - → shooting uses arrow only once
  - → grabbing picks up gold if in same square
  - → climb drops the gold in same square.

(8)

- ⇒ Sensors is required :- (to hitting walls)
  - → Stench, Breeze, Glitter, Bump, Scream (when wumpus it will scream)

- Actuators :- (change dir. by 90°)
  - → left turn, right turn, forward, grab, climb shoot

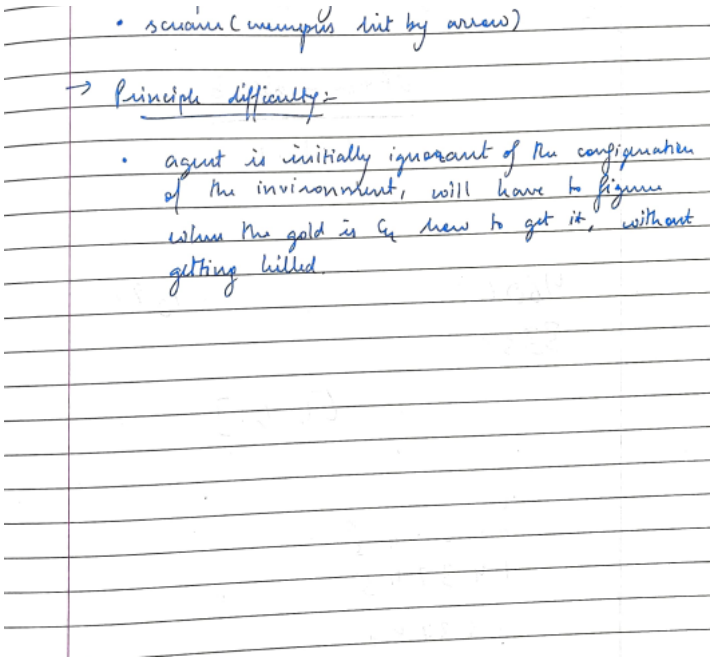
⇒ Wumpus world characterization :-

- → Fully observable? → No only local perception (partially observable)
- → Deterministic? → Yes - outcome exactly specified
- → Episodic? → No - current actions do not depend on previous actions.
- → Static? → yes wumpus & pits do not move.
- → discrete? → Yes, actions are not continuous
- → Single-agent? → Yes - we have only one adjacent to move around & grab the gold.

(9)

→ Percepts given to the agent :-

- stench
- breeze
- glitter
- bump (running into wall)



Evaluation of 2nd September 2021 by Group No: 14

Rajneesh Pandey : 106119100
Satyarth Pandey : 106119112

Remarks :-

1) Depth-Limited Pruning could have been explained through an example with Values.
2) A small Comparison (with differences) between deterministic games and Stochastic games could have been done.
3) Expectimax Search Algorithm could have been stated more clearly. However, Pseudo Code was mentioned.
4) Apart from above, They covered each and every point correctly in detail.