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

AIML Notes
03-08-21

AI/ML Notes 03/08/2021

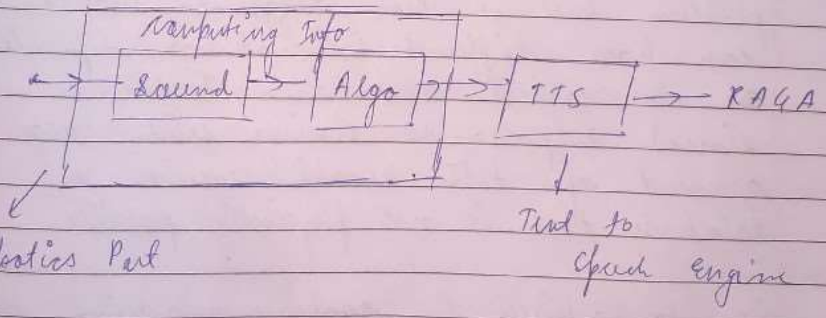
By Group 5

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 science and Engineering of making intelligent machine and intelligent computer program. 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

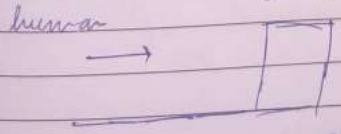
Intelligence is a fuzzy term
It is computational part of the ability to achieve goals in the world.
Example

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
  am.



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.

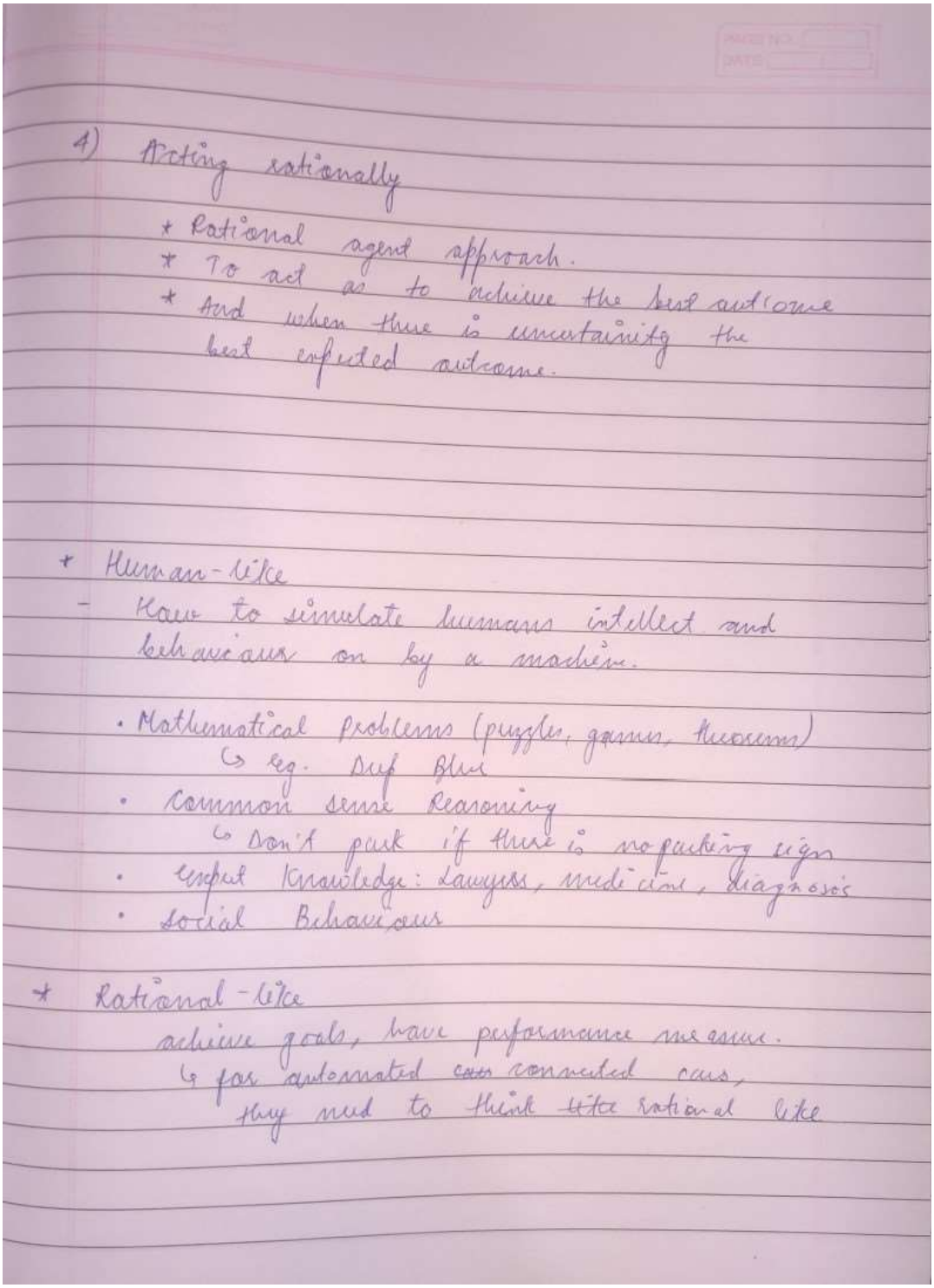
3) Thinking rationally

* 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.



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

4/8/21

Date: □□/□□/□□□□ ASB-31

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

Premise - sequence of argumentative structure.

Computers - Created to do repetative human work. → Evolved to compete with humans. doing human work.

Thinking humanly Acting humanly

Thinking rationally → Acting rationally

Best outcome ← uncertainty.

(Gunikula, Indian philosophy. Thinking based on argumentation

Thought process & Behaviour.

Turing test

Interrogater —ask Question→ 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 reasoning, 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: □□/□□/□□□□ ASB-33

Q.No. Exercise No. □□ 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) → Alive (3).
Shot

Designing games.

Search problem.

Target :- Alive (0).

Update vs revise:

A or B → C

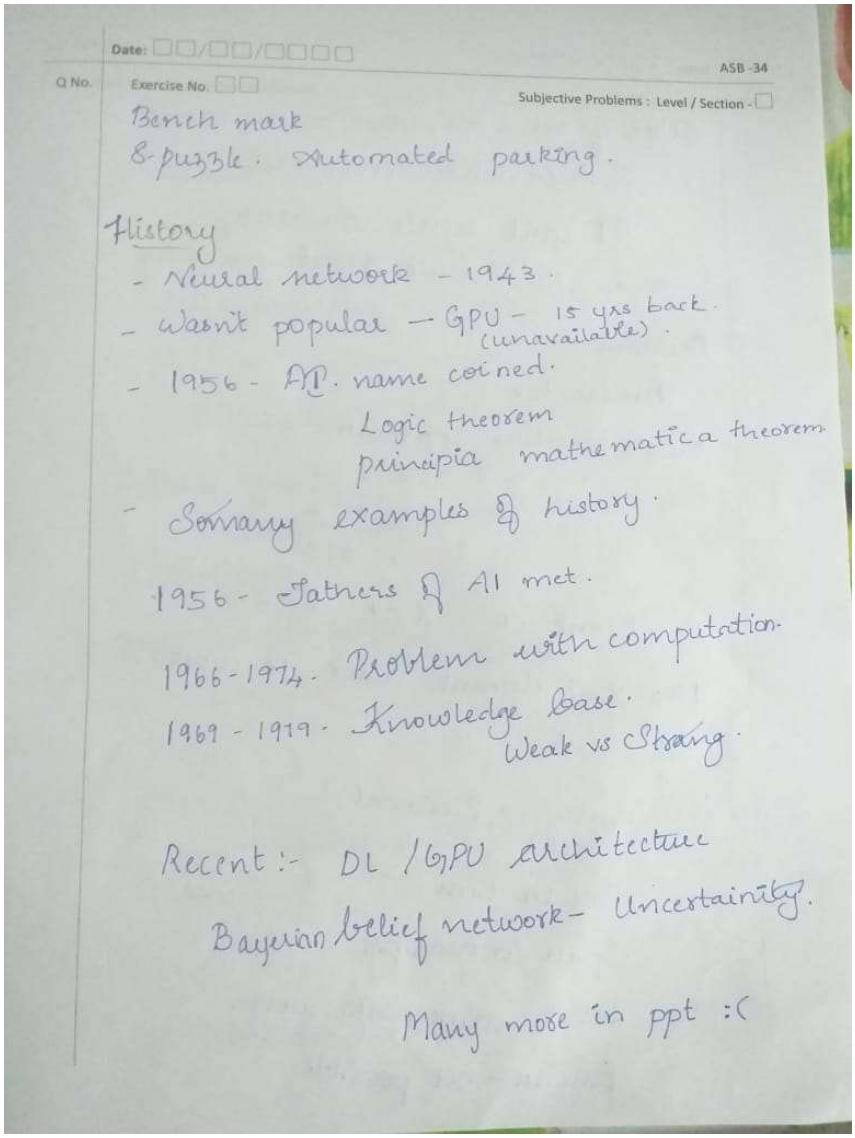
Find C rather than A & B.

Training Theories of actions

Looks. like (P) → is (P)

Make-looks like (P) → Looks-like (P)

. . . . make-looks-like (P) → is (P)?



Date: □□/□□/□□□□

ASB -34

Q No. Exercise No. □□

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 1:

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

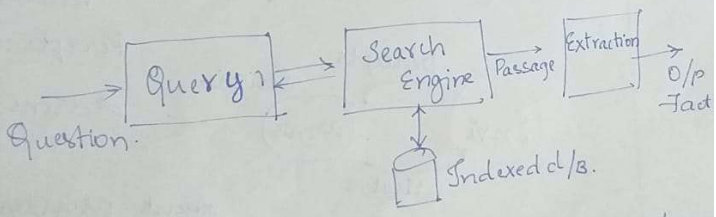
Q No. Exercise No. □□ Subjective Problems : Level / Section - □

- 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
Question — Factoid.



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

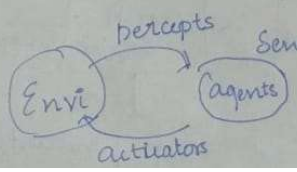
Date: □□/□□/□□□□ ASB -37

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

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

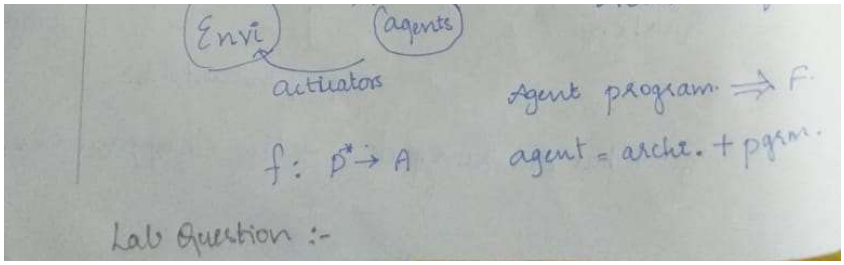
Agent
Perceive info (sensors)
↓
from environment
↓
act upon (actuators).

HumanAgents :- Eyes, Ears . . . .
Actuators :- legs, mouth.
Robotic Agent :- Sensors, motors



Vaccum cleaner
Percepts :- Location, clean
Actions :- Left, Right

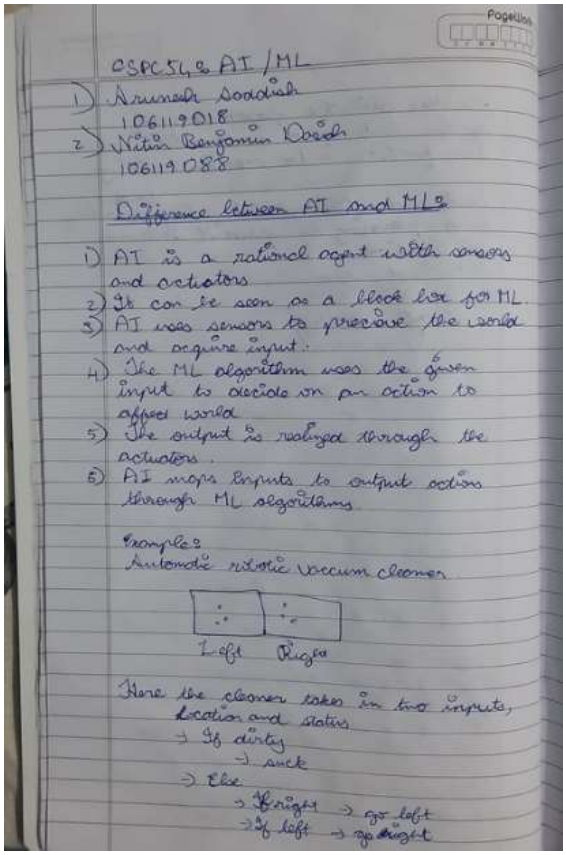
Agent program ⇒ F.



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.


CSPC54 & AI / ML
1) Arunesh Sooddiah
106119018
2) Nitin Benjamin Daniel
106119088

Difference between AI and ML?

1) AI is a rational agent with sensors and actuators
2) It can be seen as a black box for ML
3) AI uses sensors to perceive the world and acquire input.
4) The ML algorithm uses the given input to decide on an action to affect world
5) The output is realized through the actuators.
6) AI maps inputs to output actions through ML algorithms

Examples:
Automatic robotic vacuum cleaner

Left Right

Here the cleaner takes in two inputs, location and status
→ If dirty
→ suck
→ Else
→ If right → go left
→ If left → go right

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

→ Here the robot senses or perceives two inputs, location and status, through its sensors.
→ Based on the input, it produces an output using ML algorithms.
→ The output is carried out through wheels of 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 digits.

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

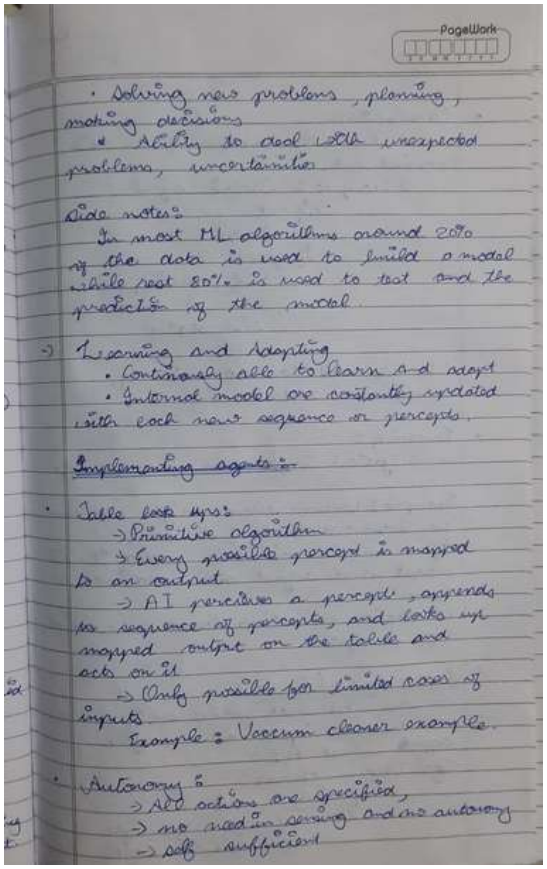
Examples
→ The robot vacuum will not take diagonal paths usually as it will reduce increase overall distance travelled to cover entire coordinate.
→ Higher distance travelled to cover a given area will be lowering performance.

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

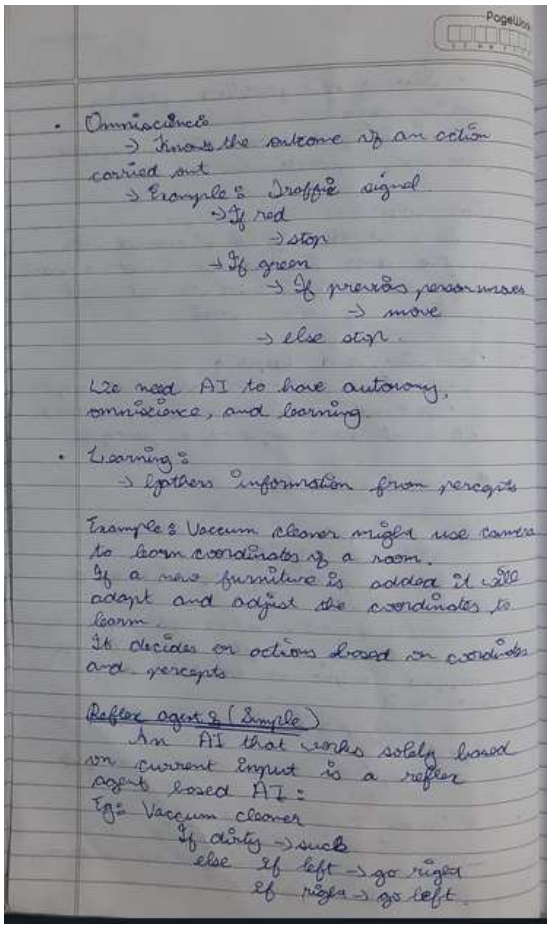
Side notes:
In most ML algorithms around 20% of the data is used to build a model while rest 80% is used to test and the predicts of the model.

→ Learning and adapting
- Continously 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 to 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 sensing 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.

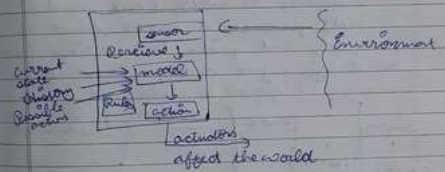
- Learning:
  → gathers information from percepts

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 agents 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:



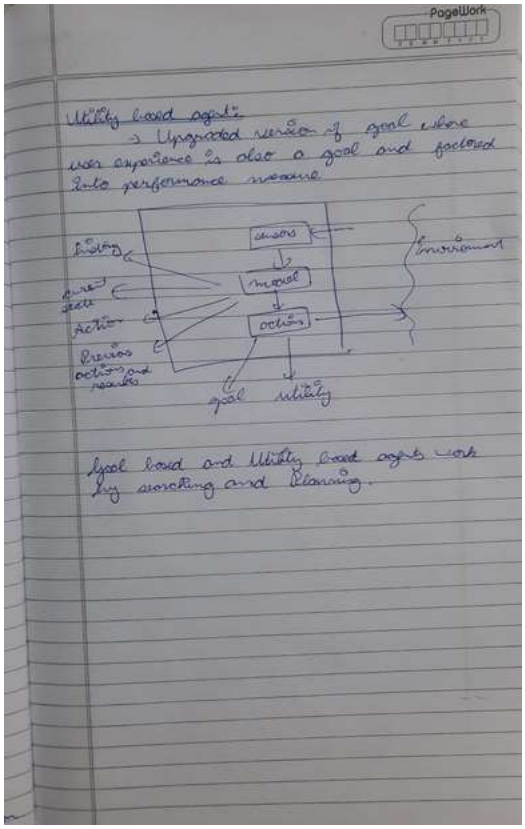
The AI uses its senses to perceive the world and build a model and take actions based on the model and the percepts.

Example:
→ Self 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

→ Google maps:
Uses previous data to predict areas of high traffic and correct itself based on current input
Example: Might change direction during course of action travel to pick path with least traffic.

Goal based agent:
→ similar to model based agent
→ AI produces a model based on percepts, but picks best action based on a given goal and performance measure (Upgrade to model based)

Ex Example:
Convert google map to goal based:
If you see a red light, google map checks for alternate routes for faster or quicker time, as goal is fastest possible time or reaching on time, rather than simple travel
Performance measure involves the specific goal rather than a simple solution



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

TNPL பக்கம்: நாள் : 11

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

* Fully observable environment vs Partially
  - Complete environment is available
  - Partial env.
* The other one is ~~Non~~ 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 if start~~ ~~senses~~ may not ~~know~~ sense the full env. So it ~~may~~ is partially at some times.

→ Single agent env vs Multi agent env

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

→ Deterministic vs stochastic

- Deterministic: If the next state depends upon the current state and the action
- Stochastic: 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 ~~know~~ called as stochastic environment.

TNPL பக்கம்: நாள் : 11

* Episodic vs Sequential
  - Episodic: 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).
  - Sequential: Current action depends on previous action. (Ex: gaming, vacuum cleaner)

* Static vs Dynamic
  - Static: * Ex: Crossword puzzle, garden sprinkler
  - Dynamic: Change in * Ex: games

* Discrete vs Continuous

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

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

* Changing env: with time (continuously)
* Taxi driving

* 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

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

P - Performance measure
E - Environment
S - Sensors

Environment Types

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

	Medical Diagnosis	Interactive English Tutor
Observable	Partial	No
Deterministic	No	Stochastic
Episodic	No	sequential
static	Dynamic	Dynamic
Discrete	Continous	Discrete
Single Agent	Single	Multi agent

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

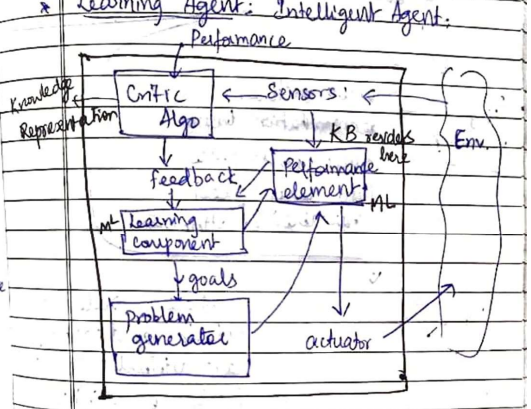
PEAS for English Tutor

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

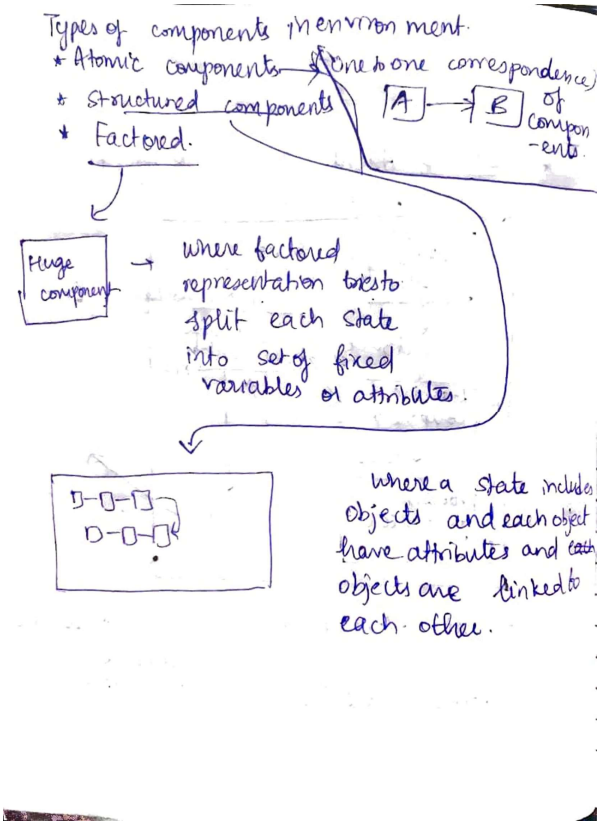
PEAS for Medical Diagnostics

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

* Learning Agent: Intelligent Agent.



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



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'.

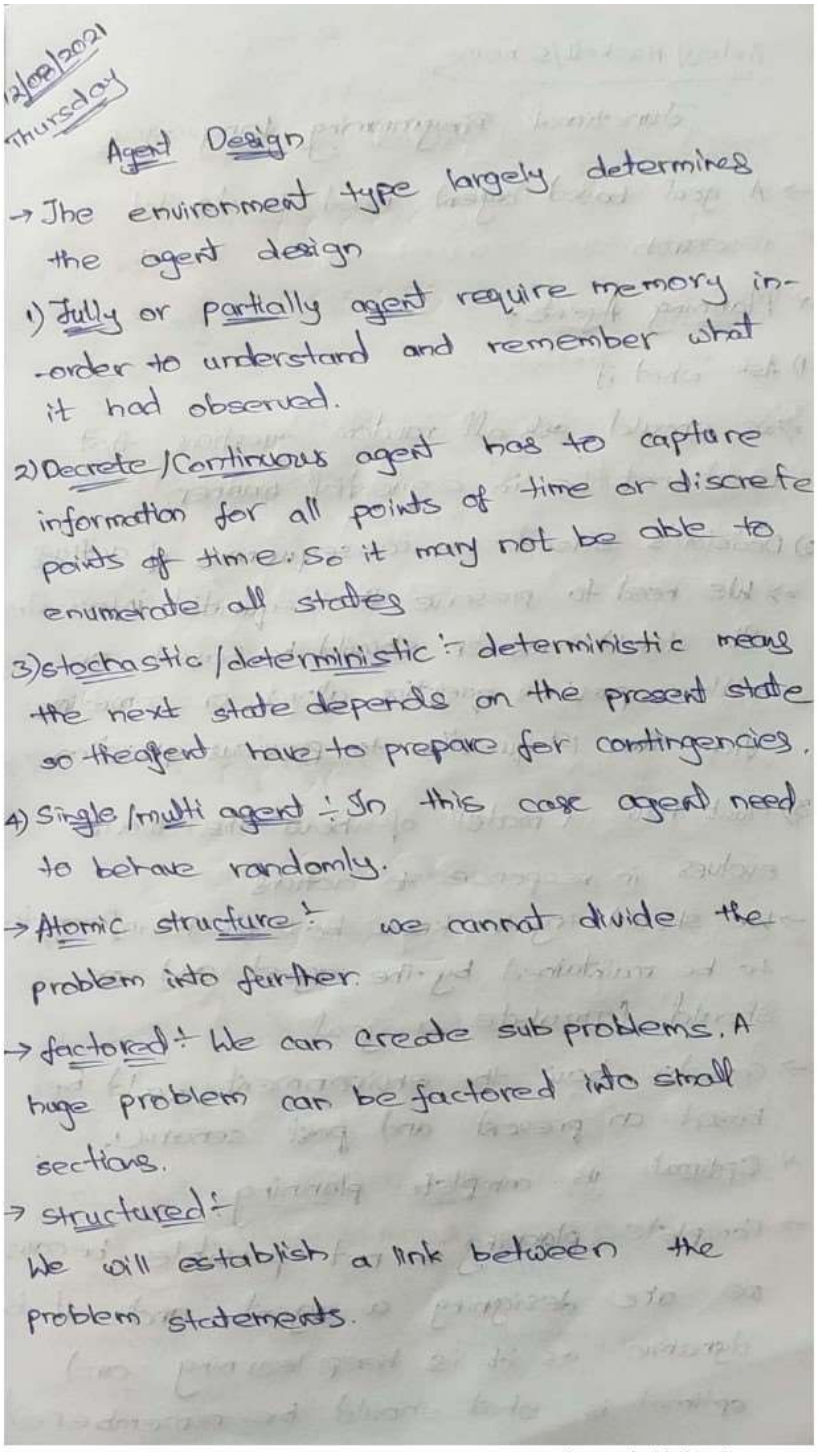
Ontology is the connection between entities and relationships.

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) stochastic/deterministic: deterministic means the next state depends on the present state so the agent have to prepare for contingencies.

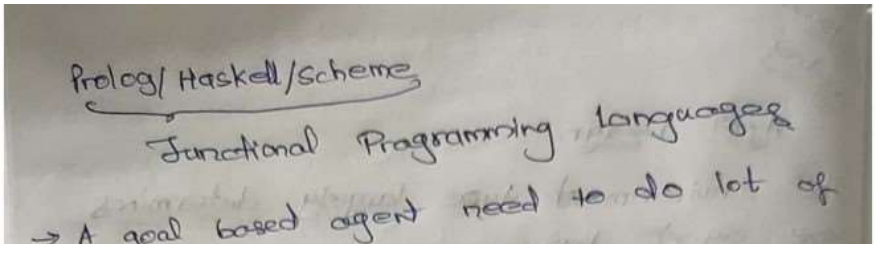
4) Single/multi agent: In this case agent need to behave randomly.

→ 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

→ 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

Scanned with CamScanner

* 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

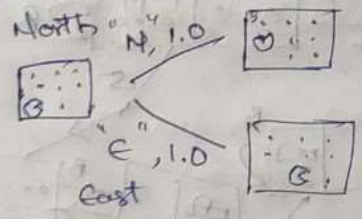
• A search problem consist of:

- 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.

Scanned with CamScanner

and what should not be

* planning vs. replanning:-

→ Plan ahead and replanning is working on 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:-

→ 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) Successor :- 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.

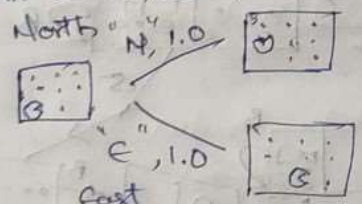
* Goal test:
→ Test to verify the given state is final state or not
* Path cost:
→ Function to assign a numeric cost to each path
* Solution:
→ The one that leads to the final state from the goal state.

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 as
and edges as cost for travelling b
them.
→ state space:
· cities

→ Start state:

- cities

→ Successor function:

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

→ start state:

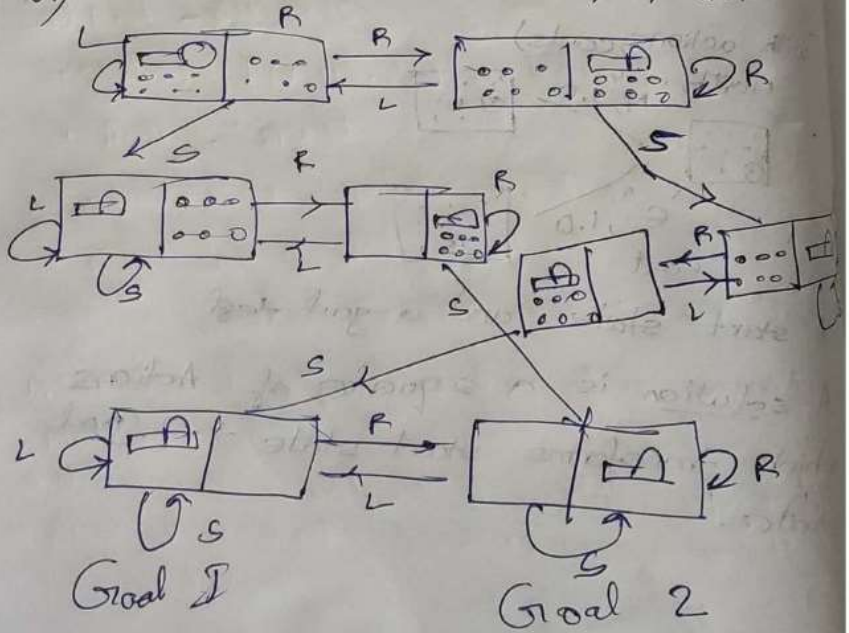
- Arad.

→ Goal state:

- Is state == Bucharest?

Q) Vacuum world:

L, R, Suck



Scanned with CamScanner

States: 2×2^2 Agent → 2

$\Rightarrow 2 \times 2^2$ possible states

(n) $\Rightarrow n \times 2^n$

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

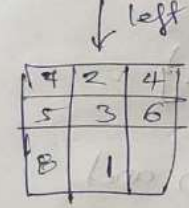
↓ up

7	2	4
5	3	6
8		1

No of actions = 9×2^9 a lot

we should think of Applicable

No of actions = 9×2^1 a lot we should think of Applicable actions.



Q) 8-queens problem

Initial state: No queens

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

transition state:

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 :- Petral (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

→ 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 :

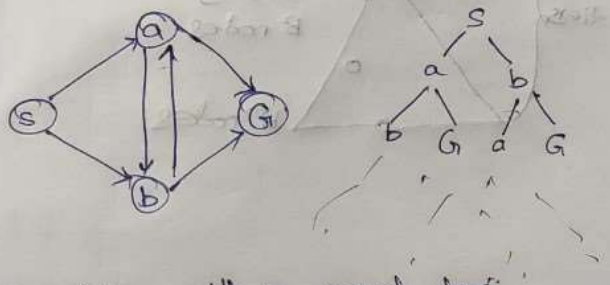
→ A "what if" tree of prans and their outcomes

→ 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

Scanned with CamScanner

Depth-First Search

Strategy : Expand a deepest node first.

Implementation :

Fringe is a LIFO stack.

Properties of Search Algorithm

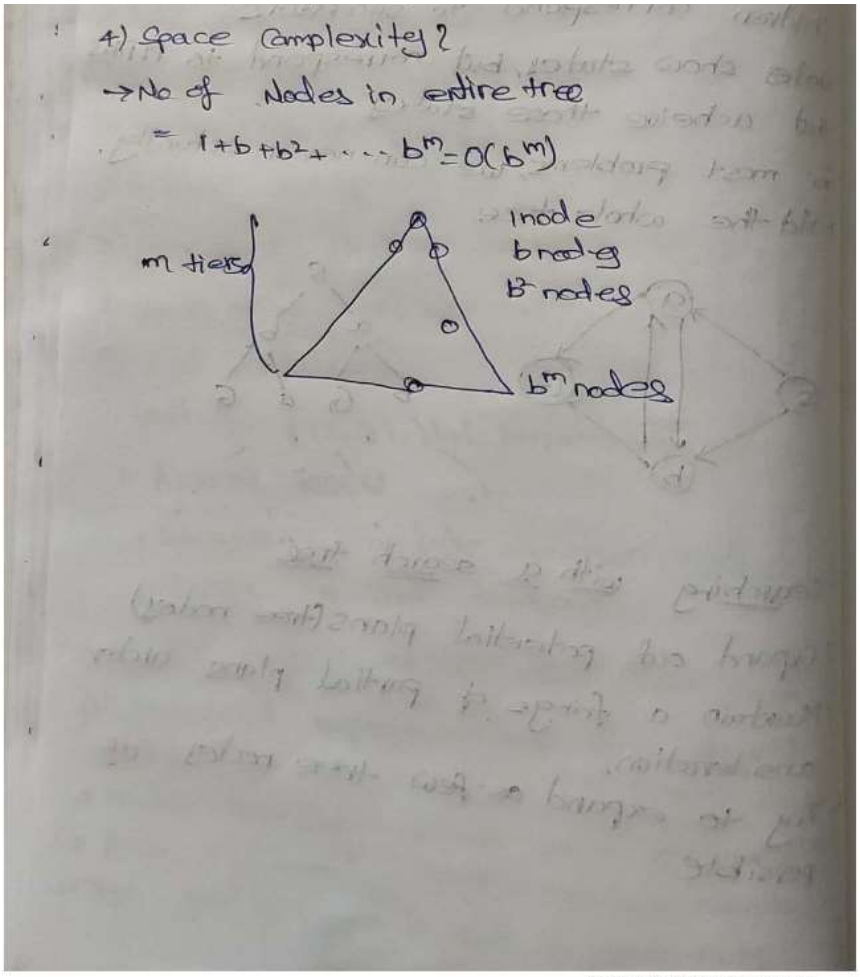
1) Complete :

2) Optinal :

3) Time Complexity?

4) Space Complexity?

→ No of Nodes in entire tree



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.

AIMLnotes16_08

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

16/8/21

Recap : Properties of Search Algorithm:

- Complete
- Optimal
- Time Complexity } Should be minimal
- Space Complexity

General Tree Search:

Fringe : Vertices which are part of tree, but not visited yet.

function TREE-SEARCH (problem, strategy) returns a solution, or failure (problem → given)

initialize the search tree using the initial state of problem

loop do

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

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:

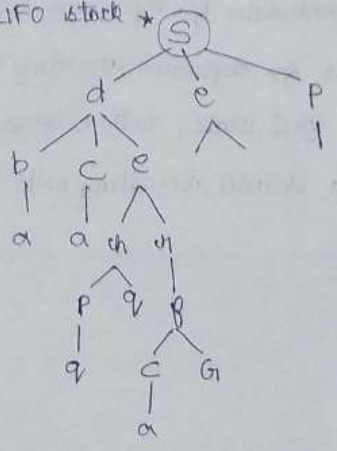
Last class eg.*

- start from S

DFS:

Last class eg.*

- expand deepest node first
- Fringe: LIFO stack



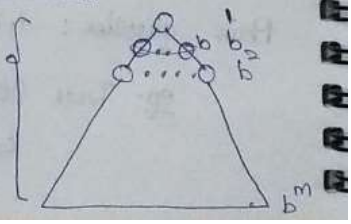
- Start from S [d, e, p are fringe vertex]
- Pick b, go till a
  - ↓ not goal
  - ⇓ backtrack
- continue so on
  . . . . . .
- reach G: Goal state
  - ⇓ Soln.

Go as deeper as want

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

discussed last class:

- TC: order of b^m
- SC:



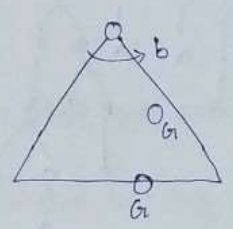
• No. of nodes: $1 + b + b^2 \dots + b^m = O(b^m)$ — b: branching fac. m: max. depth

↓ upper bound [including repeating nodes]

• 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] DFS directed graph ↓ sometimes can't reach

- Not always
- 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.

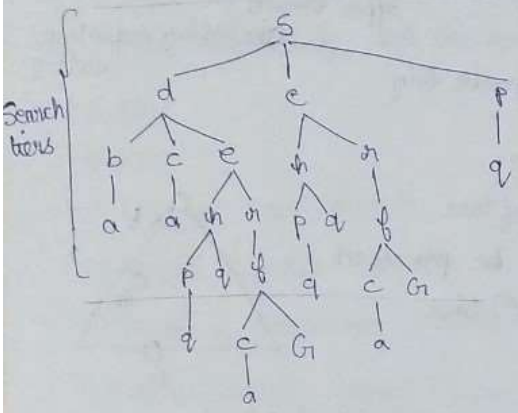
↳ limits ~~branching factor~~ depth

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

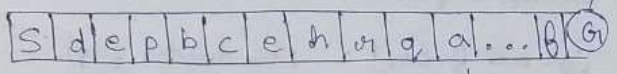
BFS:

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

Eg.



- Start from S → fringe
- expand to d, e, p
- 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.

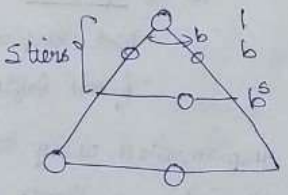


Goal, trace back for route

↓ already put in queue, not inserted again

$S \rightarrow e \rightarrow r \rightarrow f \rightarrow G$

level by level search



→ What nodes expanded?
- all nodes above shallowest sol^n
- 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:

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

Increases limit of algo : from 0, 1, ...
[uses depth limit concept] fix limit & do DFS

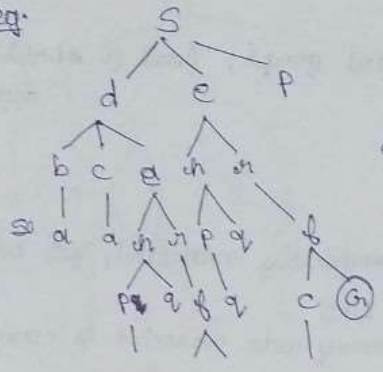
→ Run DFS with depth limit 1. If no soln. . . . .
[nothing but BFS]
→ " " " " " 2. " " "
[all vertices not put in stack, dl=2, only till 2 its put]
→ " " 3. . . . .
. . . . . .

* Redundant wastefully? not costly, complete & optisoln.

as it combines DFS & BFS

- most work in lowest lvl search ⇒ not so bad!

eg.



dl=1

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

If dl=4, G is reached!

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

- 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

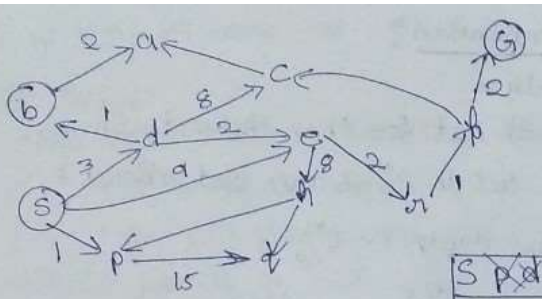
Uniform Cost Search [UCS]:

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

Strgy: expand cheapest node 1st

Fringe is priority [cumulative cost] queue.

Eg:

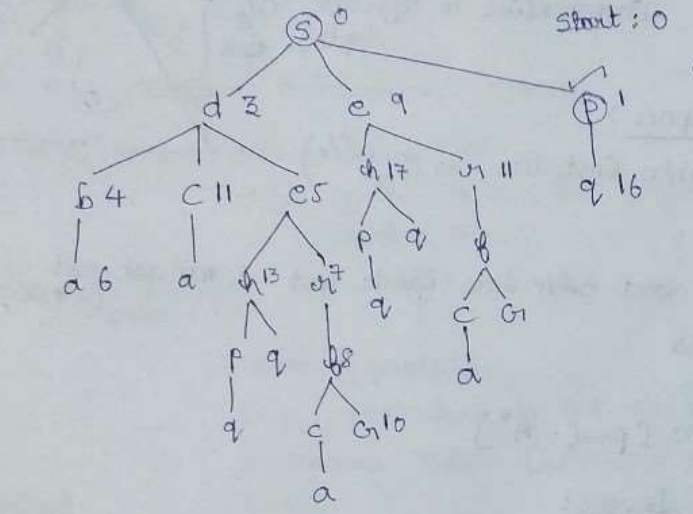


S	p	d	e	q
0	1	3	9	16

Start : 0

Go with lowest cost

BFS with first node visited being lowest cost node cumulative



S			e	q
0	1	3	9	16

visit d →

S			a	c	q
4	X	5	6	11	16

a	r	c	h	q
6	7	11	13	16

	g	c	h	q
8	10	11	13	16

8	9	C	h	9
8	10	11	13	16

←

G	7	11	13	16

↓

reached Goal with min cost

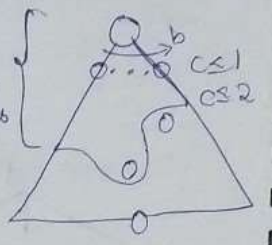
~ to ~~Prim's~~ Dykstra's algo to find shortest route from S – G

[only edge cost]

↳ 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: DFS 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

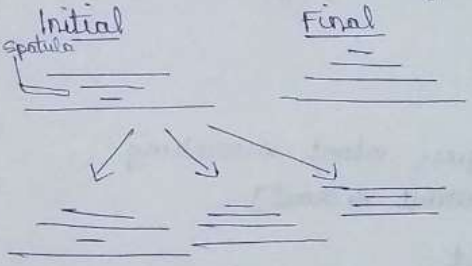
- all fringes r priority queues
- for DFS & BFS, can avoid $\log(n)$ overhead from actual priority q; with stacks & q
- Can code an implementn with variable queuing obj.

*One more strgy: Bidirectional search

eg: Google map: ocean b/w 2 pts.
says go thr. ocean b4 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 permn. σ of 1 to n, $f(\sigma)$: smallest no. of prefix reversal for $\sigma \rightarrow$ identity permn.

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

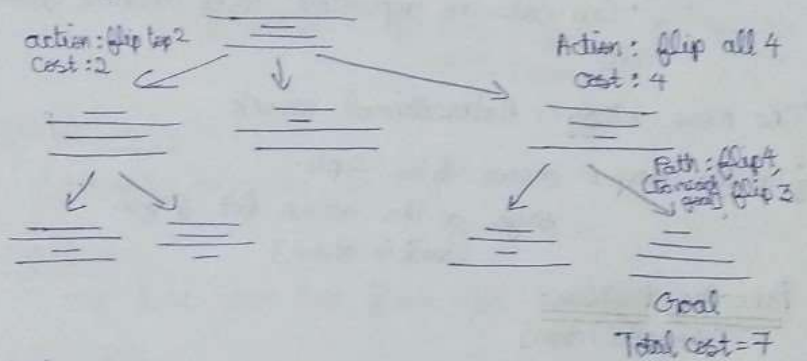
we show, $f(n) \leq (5n+5)/3$ & $f(n) \geq 17n/16$
n: multiple of 16

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

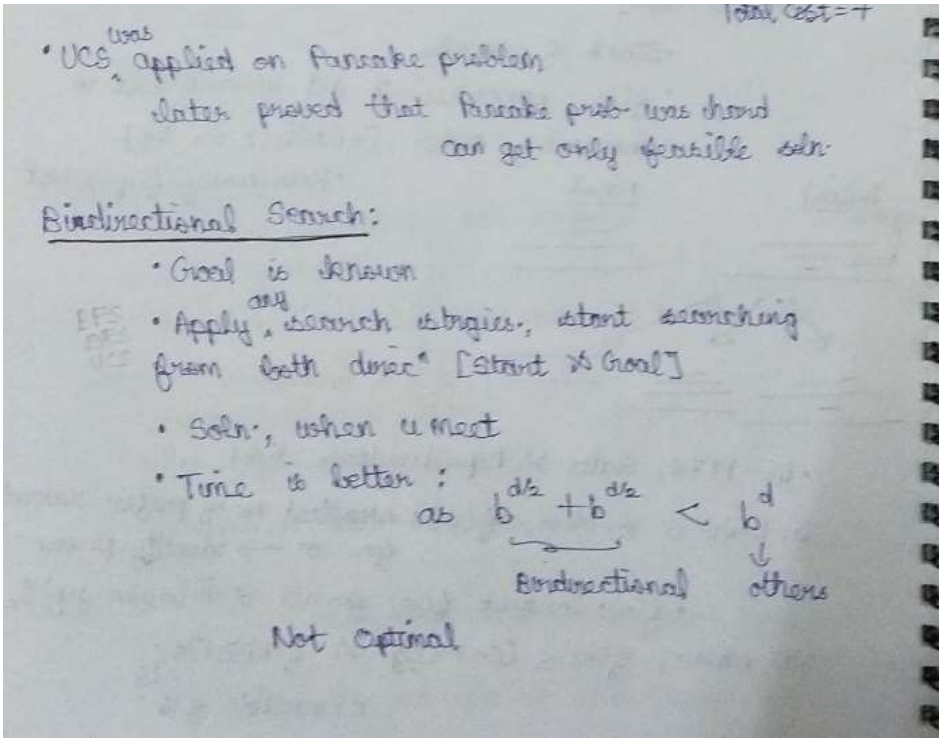
$$3n/2 - 1 \leq g(n) \leq 2n+3$$

III^er te:
$2^n - 1$ moves for Tower of Hanoi

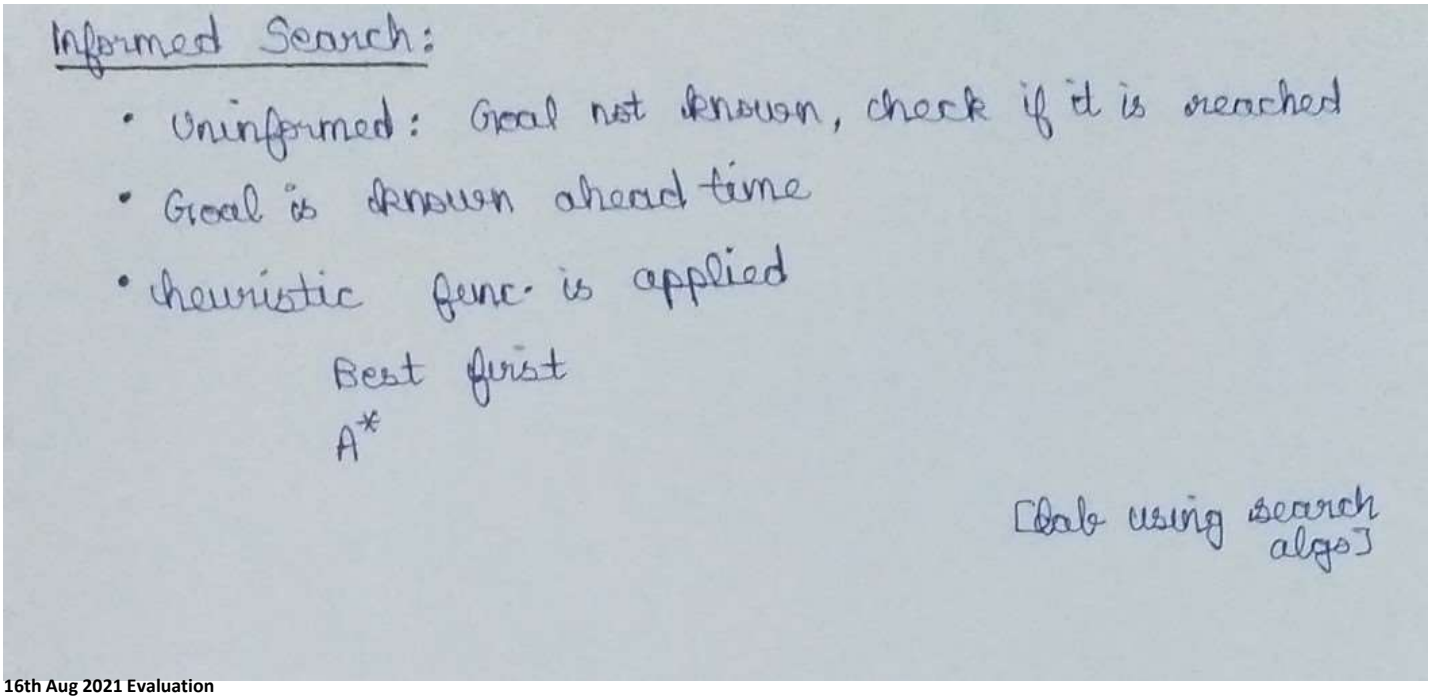
→State space tree is based on gnl. tree search



- UCS was applied on pancake problem



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 17-08-2021

FILE FOR EASIER VIEWING

Date: 17/08/2021

Notes for
17-08-2021

FILE FOR EASIER VIEWING

Date: 17/08/2021

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

Topics covered in class:

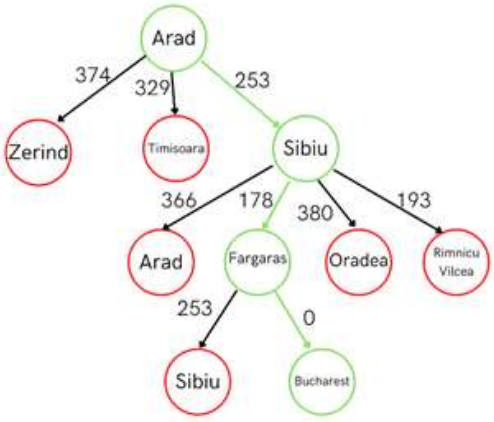
- Informed Search Strategies
  - Introduction to Heuristic functions
  - Greedy Best First Search
  - A* search algorithm

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:



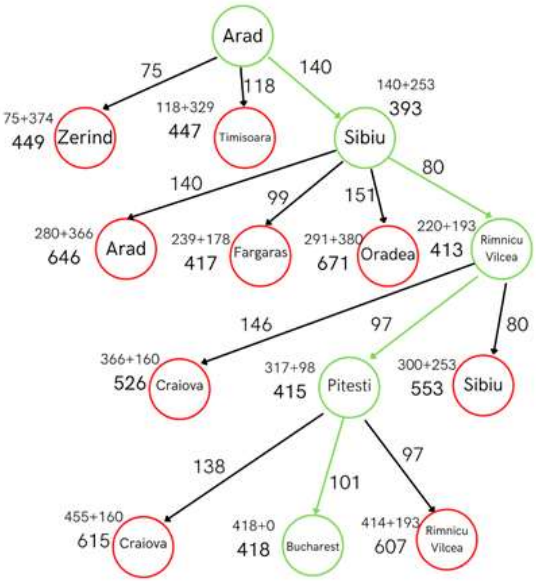
  - In the 1st step Sibiu is chosen as in has the least straight-line path to Bucharest and its nodes are expanded
  - In the 2nd step Fargaras has the least straight-line path and it is expanded. This is done until Bucharest is reached.
  - The final path is Arad->Sibiu->Fargaras->Bucharest, and the final cost comes to 450.

- 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*

- 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.

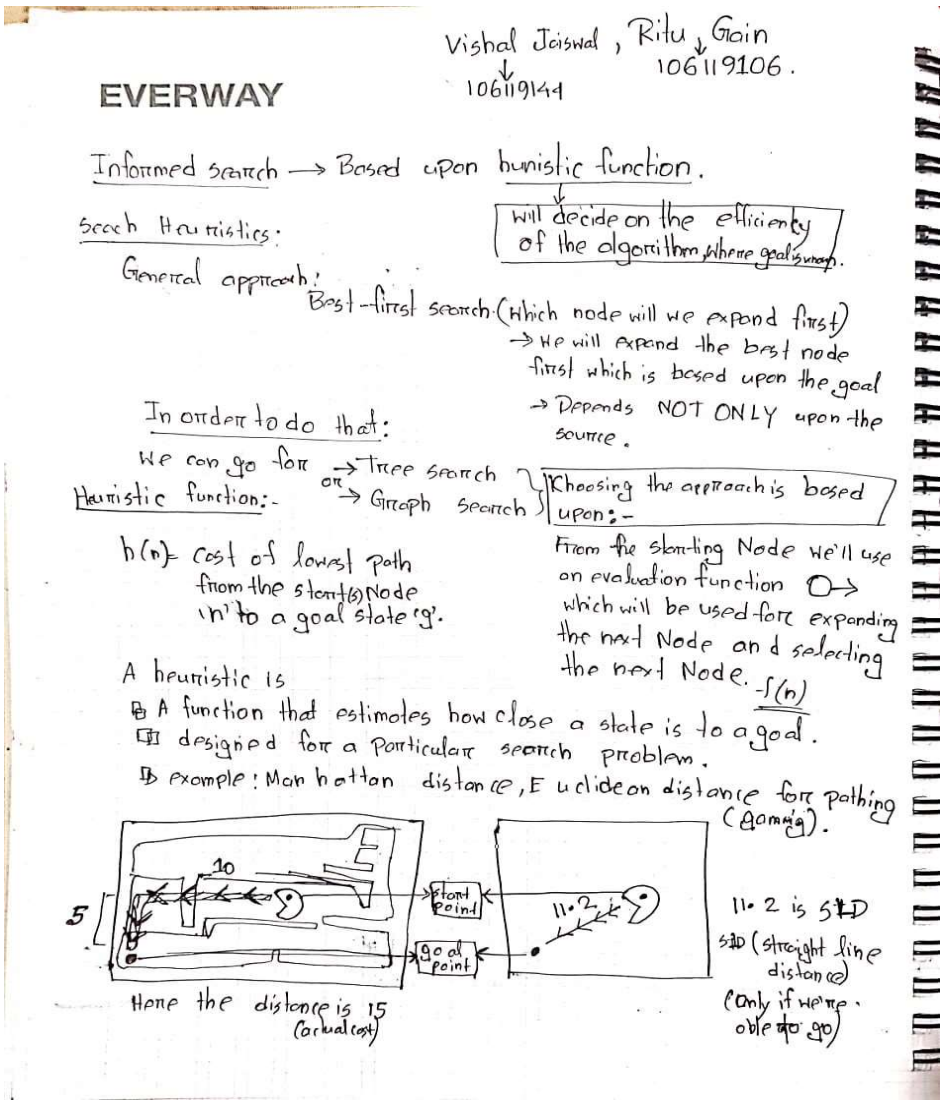
- Applications of A*
  - A lot of video games
  - Paths and routes problems
  - Language analysis
  - Machine translation, etc
- 8 puzzle game can be solved using A* algorithm.

Evaluation for 17th August 2021 (Monday) Class

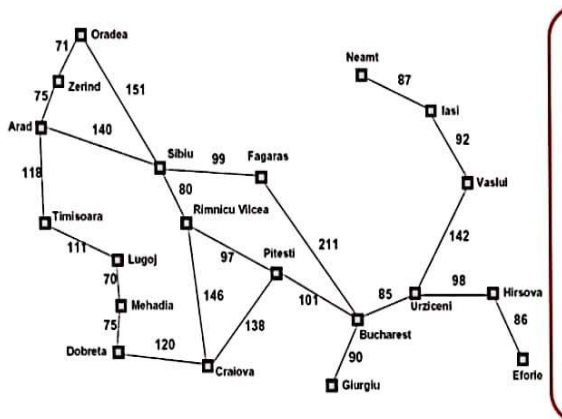
Done by Group 1 : 106119104-Rishi Sathish and 106119118-SreeGaneshTN

- 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.

Class Notes by Group no: 33
Vishal Jaiswal : 106119144
Ritu Gain : 106119106



Example: Heuristic Function



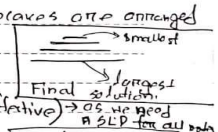
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

EVERWAY

DATE:

- So we have to get our ~~euclic~~ Heuristic function such that ~~we~~ the cost will going to be almost close to the ~~STD~~ 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 ~~fata~~ 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.

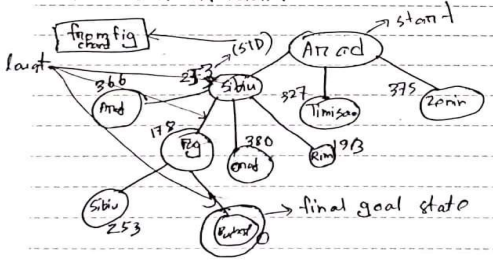


Greedy search / Greedy Best first search (Not very effective) → as we need a SLD for all nodes

→ 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.

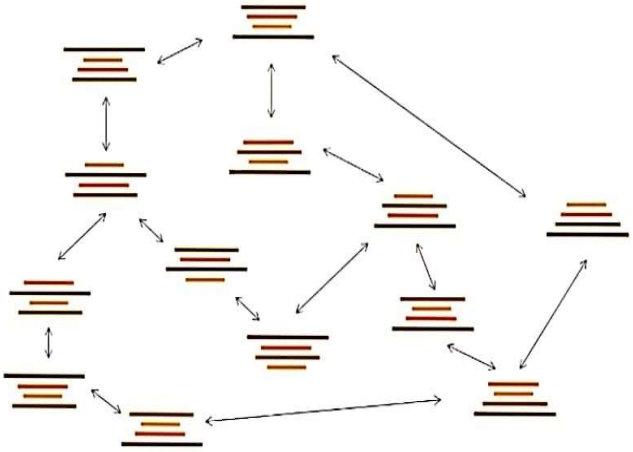


Where STD is 366

Arads neibours are:
zerind, sibiu and Timisoara

- So with the smallest STD to least (Sibiu smallest)

Example: Heuristic Function



EVERWAY

Greedy search

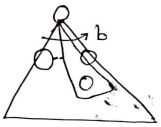
Strategy: expand a node that you think is closest to goal state.

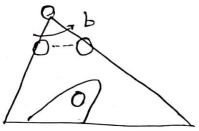
- Heuristic: estimate of distance to nearest goal for each state

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 against 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. (I didn't consider that aspect of the algo)

So, in A* we will consider 2 cost.

g(n) → cost to reach a particular node

h(n) → " " goal from the node to the goal (and distance cost)

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

EVERWAY

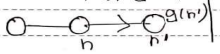
DATE :

$g(n)$ → path cost from the start state to the Node n } (combo)
$h(n)$ ↔ SLD

A^* → UCS → (g+h) (instead of g)

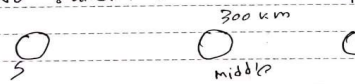
UCS → lontoix
Greedy → rabbit
A^* → UCS ∪ Greedy

Optimality of A^*

Admisibility heuristic	consistency → stronger
→ tries to not over estimate 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'

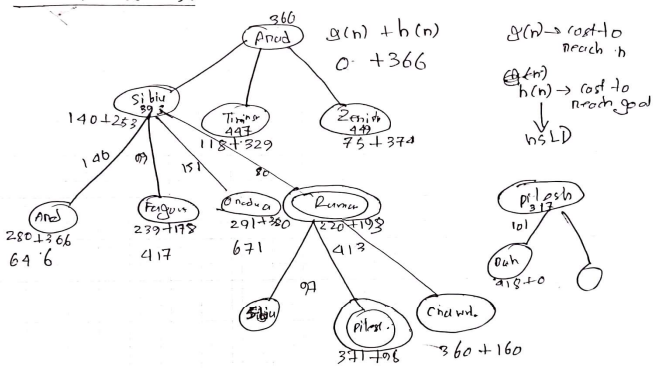
At every particular point of time when ~~over~~ we're trying to go from one node to next node we will consider the path cost to reach the particular node + the cost to reach the goal state, so we have to cumulatively do that, to do that we should'nt over estimate or we shouldn't ~~exist~~ exit the capacity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



EVERWAY

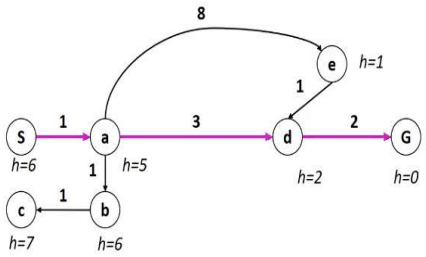
Every consistent heuristic is called as consistent heuristic and admisable (no overestimate)

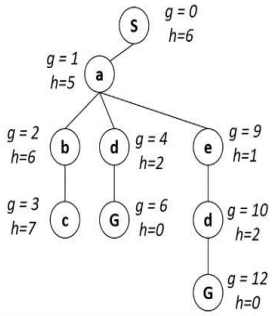
How A* behaves:



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)

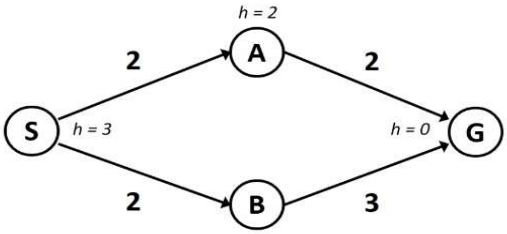




- A* Search orders by the sum: f(n) = g(n) + h(n)

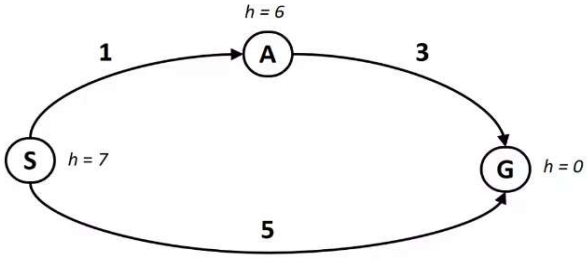
When should A* terminate?

- Should we stop when we enqueue a goal?



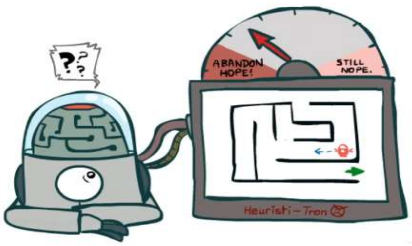
- No: only stop when we dequeue a goal

Is A* Optimal?

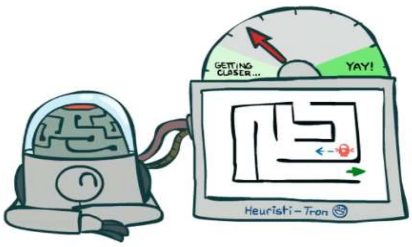


- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

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

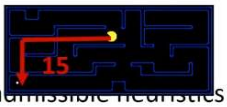
- A heuristic h is *admissible* (optimistic) if:

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

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

- Examples:

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



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

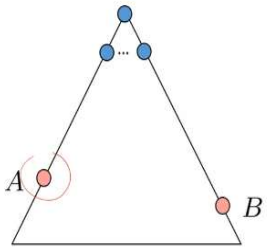
Optimality of A* Tree Search

Assume:

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

Claim:

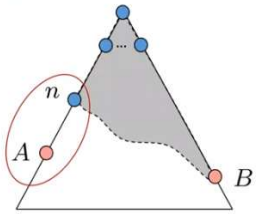
- A will exit the fringe before B



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)

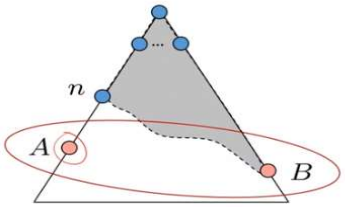


$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

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)

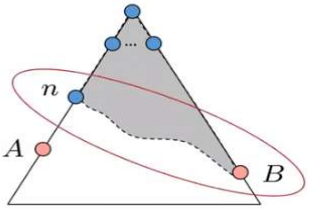


$g(A) < g(B)$	B is suboptimal
$f(A) < f(B)$	h = 0 at a goal

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

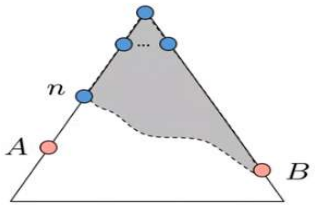


$$f(n) \leq f(A) < f(B)$$

Optimality of A* Tree Search: Blocking

Proof:

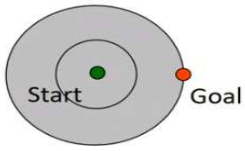
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



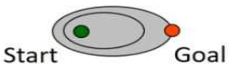
Properties of A*

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*

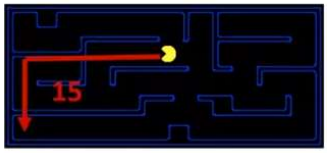
A* Applications

- 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)]

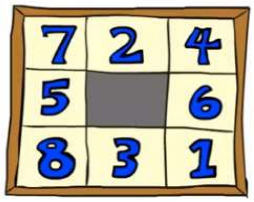
Creating Admissible Heuristics

- 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

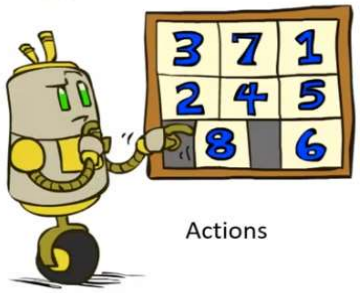


- Inadmissible heuristics are often useful too

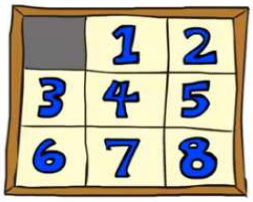
Example: 8 Puzzle



Start State



Actions

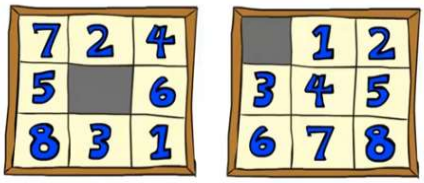


Goal State

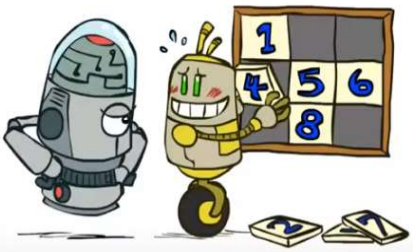
- What are the states?
- How many states?
- What are the actions?
- How many successors from the start state?
- What should the costs be?

8 Puzzle I

- Heuristic: Number of tiles misplaced
- Why is it admissible?
- h(start) = 8
- This is a *relaxed-problem* heuristic



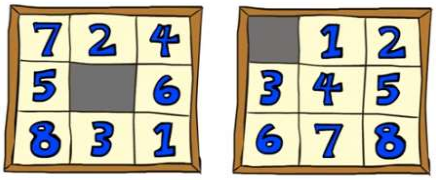
Start State Goal State



	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
- h(start) = 3 + 1 + 2 + ... = 18

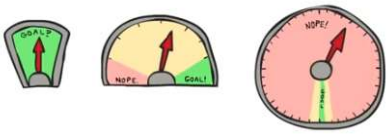


Start State Goal State

	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_NOTES
PDF File for easy viewing

23/8/21

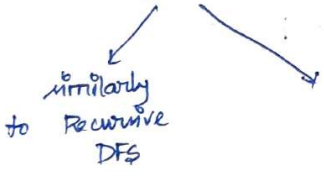
will A* always lead to a complete solution?
↳ Yes, but the problem is, memory, if the destination is too far.

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;

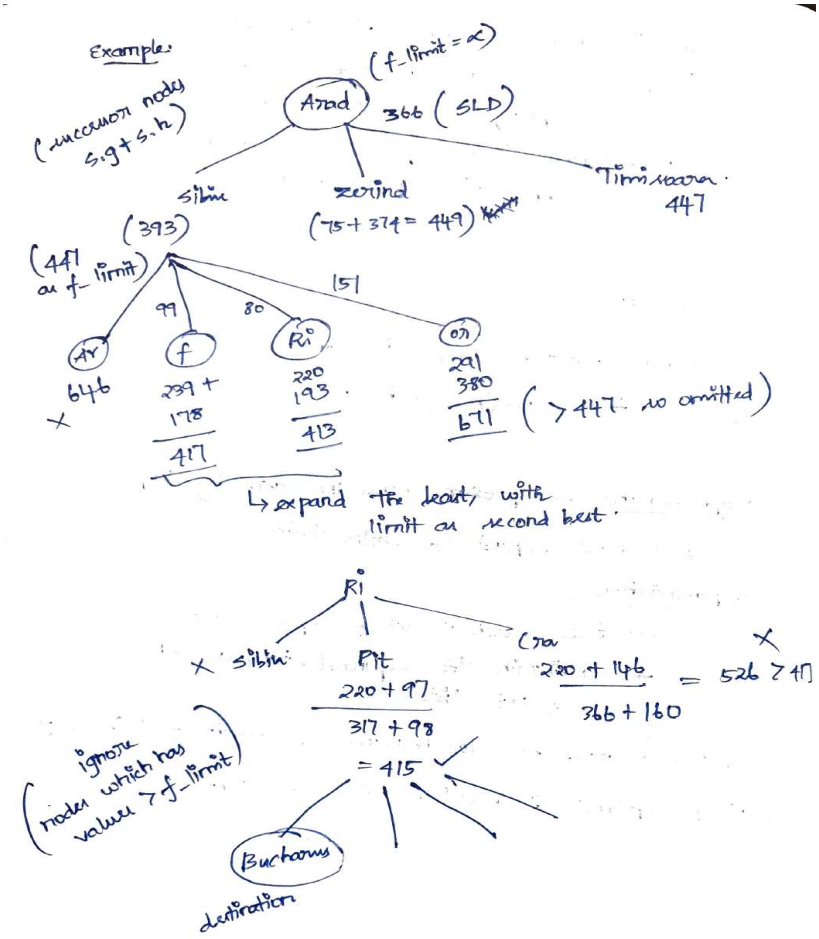
alternative node = second lowest f.value of successor

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.


Example:
(succession node
s.g + s.h)
(f_limit = ∞)
Arad
366 (SLD)
Sibiu
Zerind
(75 + 374 = 449)
Timisoara
447
(393)
(441
as f_limit)
151
99
80
Ar
f
Ri
Or
646
239 +
178
417
220
193
413
291
380
671
(> 447 no omitted)
↳ expand the least with
limit as second best.
Ri
Sibiu
Rit
220 + 97
317 + 98
= 415
Cra
220 + 146
366 + 160
= 526 > 417
ignore
(nodes which has
values > f_limit)
Bucharest
destination

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

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

graph search: → problem comes in detecting loops

↳ so never expand a vertex twice

implementation:

→ tree search + set of expanded states

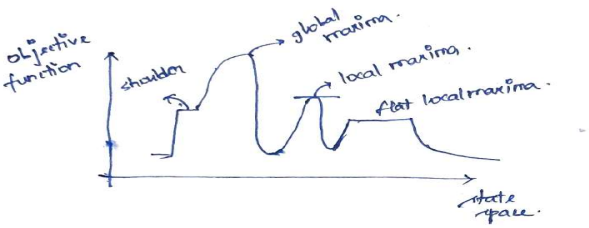
→ maintain the expanded states as SET & not lists

A^* → looks for loc minima (so might fail in Graphs.

→ A^* is optimal if heuristic is consistent.

Local search Algorithms (Greedy Approach)
↳ applied for Game problems

Hill climbing | simulated annealing | local beam | Genetic algorithm



Hill-climbing search:

→ climb – uphill

```
funct hc (problem)
{
    current ← make_node (prob, initialstate)
    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.

plateau → flat area of the state space
→ flat – local maxima / shoulder
→ no Hill climb

23rd August 2021 – Notes Review by Group 10

Bokam Pavan- 106119024
Poduri Sri Sai Kesava- 106119092
Memory Bounded herustic search (explanation missing)
IDA*-informed search,ID-uninformed search
Even memory bounded herustic search is also not a conducive solution since depth has been fixed
RBFS approach-Greedy approach
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