



# Artificial & Computational Intelligence

## DSE CLZG557

### M1 Introduction

Indumathi V  
Guest Faculty,  
BITS - CSIS

**BITS** Pilani  
Pilani Campus

# Agenda

---

- Course Administration
- Course Overview with example
- Getting Started (Module 1)



# Course Administration

— s/w AI agent → chat bot  
 It/w AI agent → self driving car

## About the course

- Focus on

- Principles of artificial intelligence

- Concepts, algorithms involved in building rational agents

- Topics covered like

- M2 (informed, uninformed & local) search & optimizations & applications

- (logical & probabilistic) knowledge representation

- (logical & probabilistic) Reasoning & applications

- a bit of learning (reinforcement learning)

- Topics not-covered like

- Hardware aspect of the design

- Formal machine learning & deep learning algorithms, neural networks etc., are covered under next semester courses like : Applied Machine Learning, Deep Learning

- Deeper applications & algorithms of application of AI in Computer Vision, Natural Language processing etc., are included under next semester courses like Natural Language Processing, Information Retrieval.

AI computing s/w sys

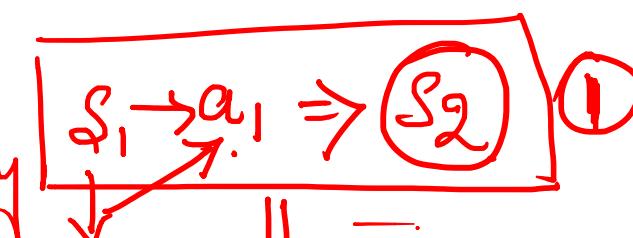
AI

$$P(e_1) = 0.7$$

$$P(e_2) = 0.3$$

Uncertainty

Q-learning



$$s_1 \rightarrow a_1 \Rightarrow s_3$$

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

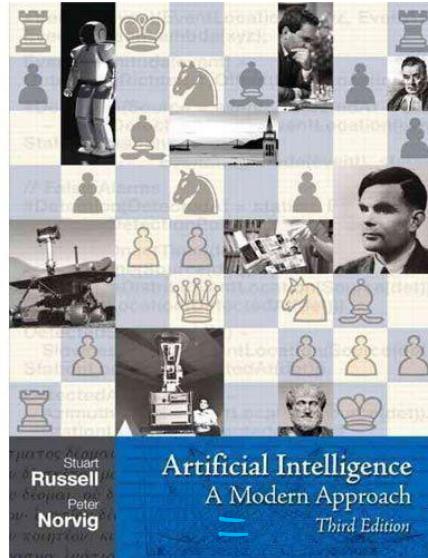
M5 Probabilistic Representation and Reasoning

M6 Reasoning over time, Reinforcement Learning

M7 Ethics in AI

# About the course

Text Book



**Exercises :** In Python & its libraries

**Evaluation :** 25% Assignment + 5% Quiz + 30% Mid Semester Written Exam  
+ 40% Comprehensive Written Exam

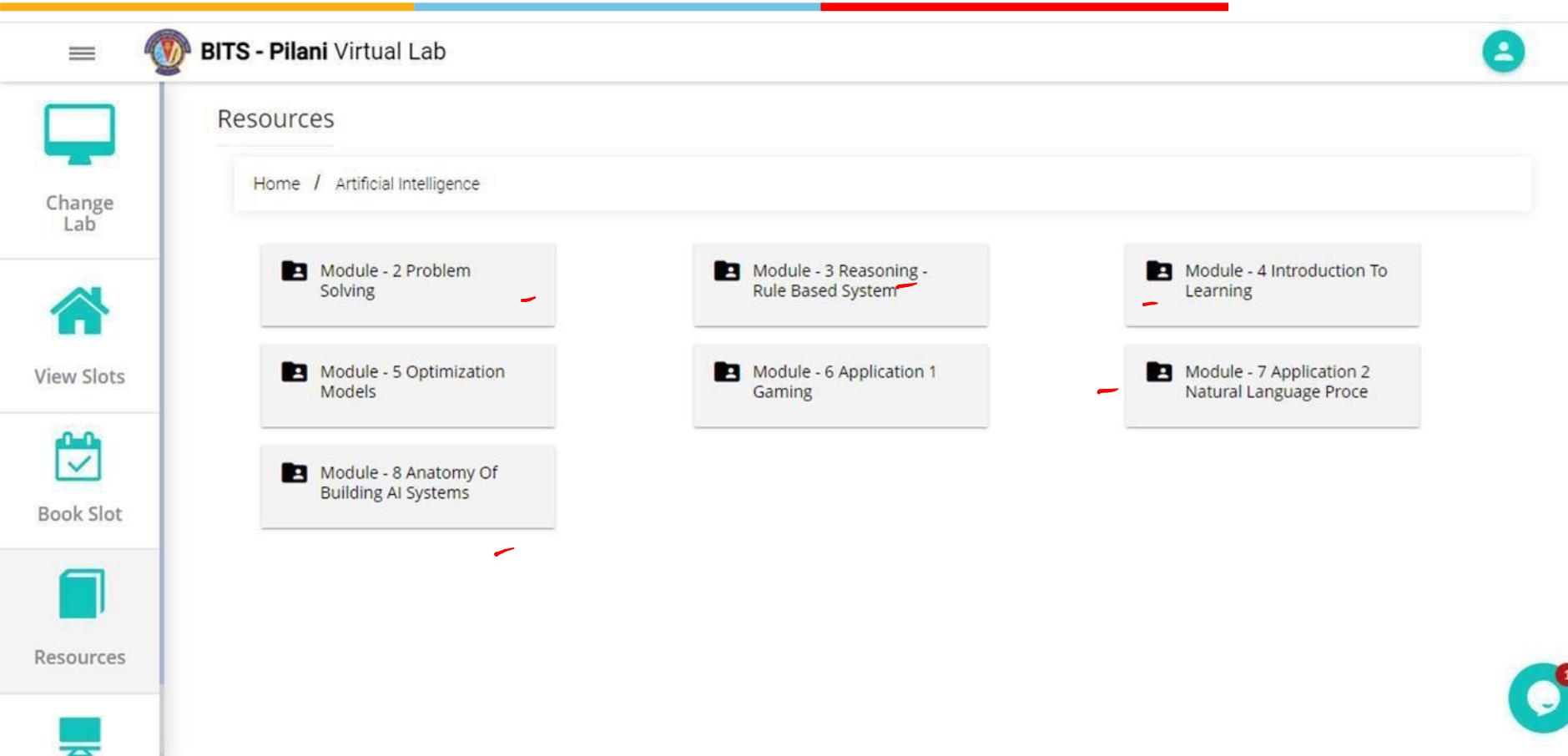
13      12

=

↓      ↓      ↓

(2)

# About Labs



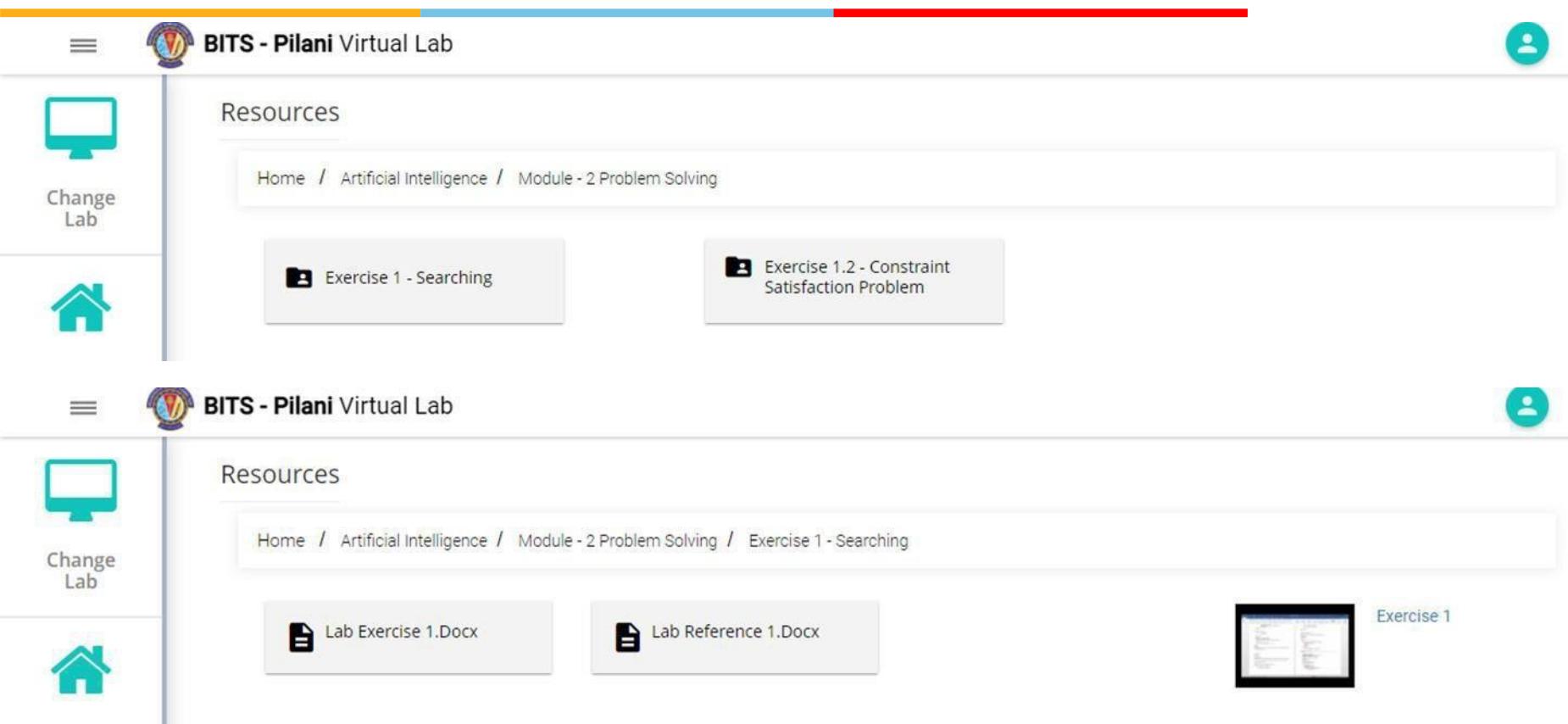
The screenshot shows the 'BITS - Pilani Virtual Lab' interface. On the left, a vertical sidebar lists navigation options: 'Change Lab' (with a computer monitor icon), 'View Slots' (with a house icon), 'Book Slot' (with a calendar icon), 'Resources' (with a book icon), and a 'Logout' button (with a person icon). The main content area is titled 'Resources' and shows the path 'Home / Artificial Intelligence'. Below this, there are eight module cards arranged in two rows of four:

- Module - 2 Problem Solving
- Module - 3 Reasoning - Rule Based System
- Module - 4 Introduction To Learning
- Module - 7 Application 2 Natural Language Proce
- Module - 5 Optimization Models
- Module - 6 Application 1 Gaming
- Module - 8 Anatomy Of Building AI Systems

A red horizontal bar spans across the top and bottom of the slide.

## Exercises : In Python & its libraries

# About Labs



The screenshot displays the BITS-Pilani Virtual Lab interface. It features a sidebar with icons for 'Change Lab' (monitor), 'Home' (house), and user profile. The main area shows the title 'BITS - Pilani Virtual Lab' and a 'Resources' section. A breadcrumb navigation path is visible: Home / Artificial Intelligence / Module - 2 Problem Solving. Below this, there are two items: 'Exercise 1 - Searching' and 'Exercise 1.2 - Constraint Satisfaction Problem'. In the second instance of the interface, the path is expanded to include 'Exercise 1 - Searching'. The resources listed are 'Lab Exercise 1.Docx', 'Lab Reference 1.Docx', and a thumbnail for 'Exercise 1'.

## Exercises : In Python & its libraries

# About Labs



BITS - Pilani Virtual Lab



## Resources

[Home](#) / [Artificial Intelligence](#) / [Module - 2 Problem Solving](#) / [Exercise 1 - Searching](#)



Lab Exercise 1.Docx



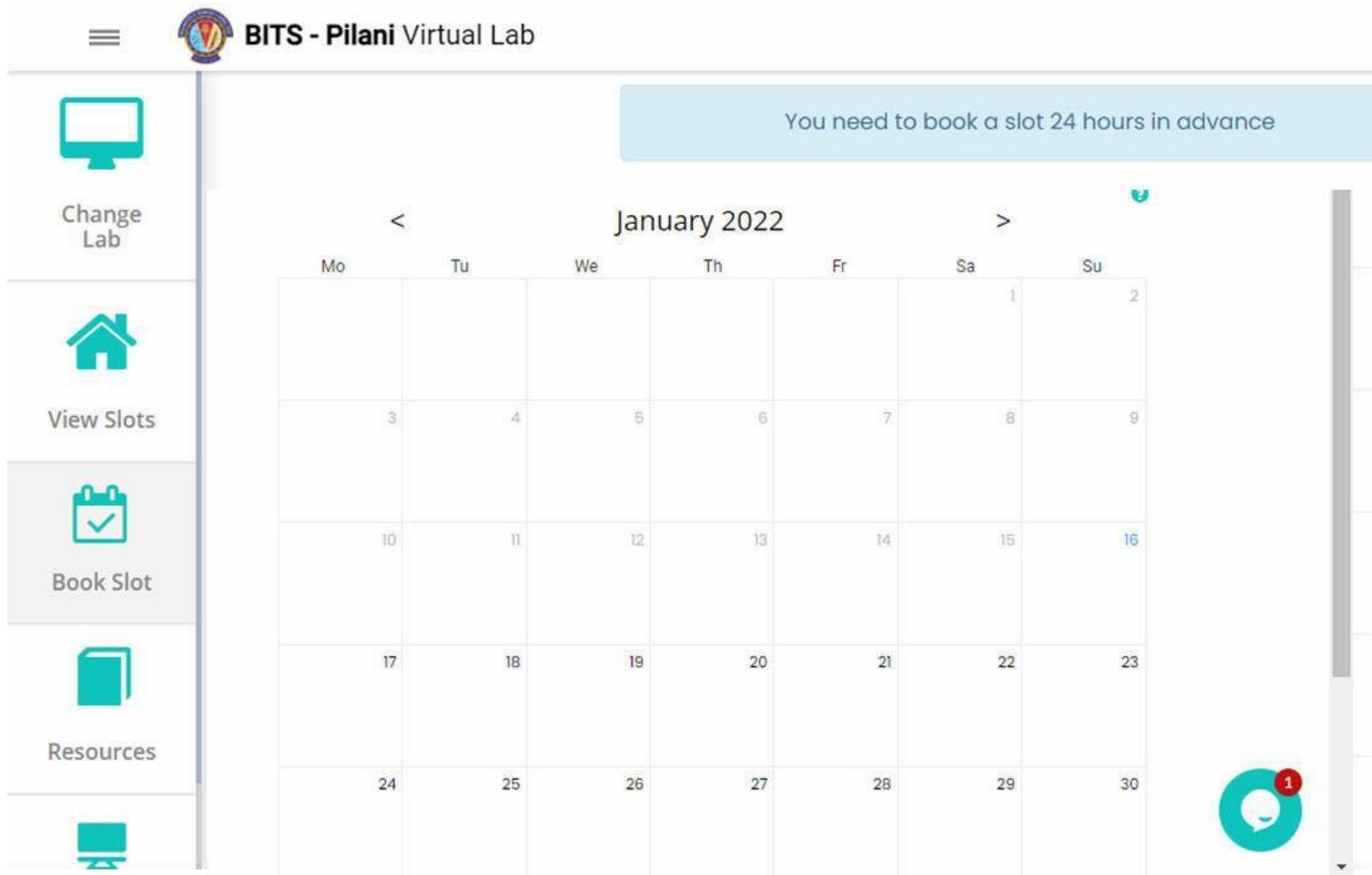
Lab Reference 1.Docx



Exercise 1

## Exercises : In Python & its libraries

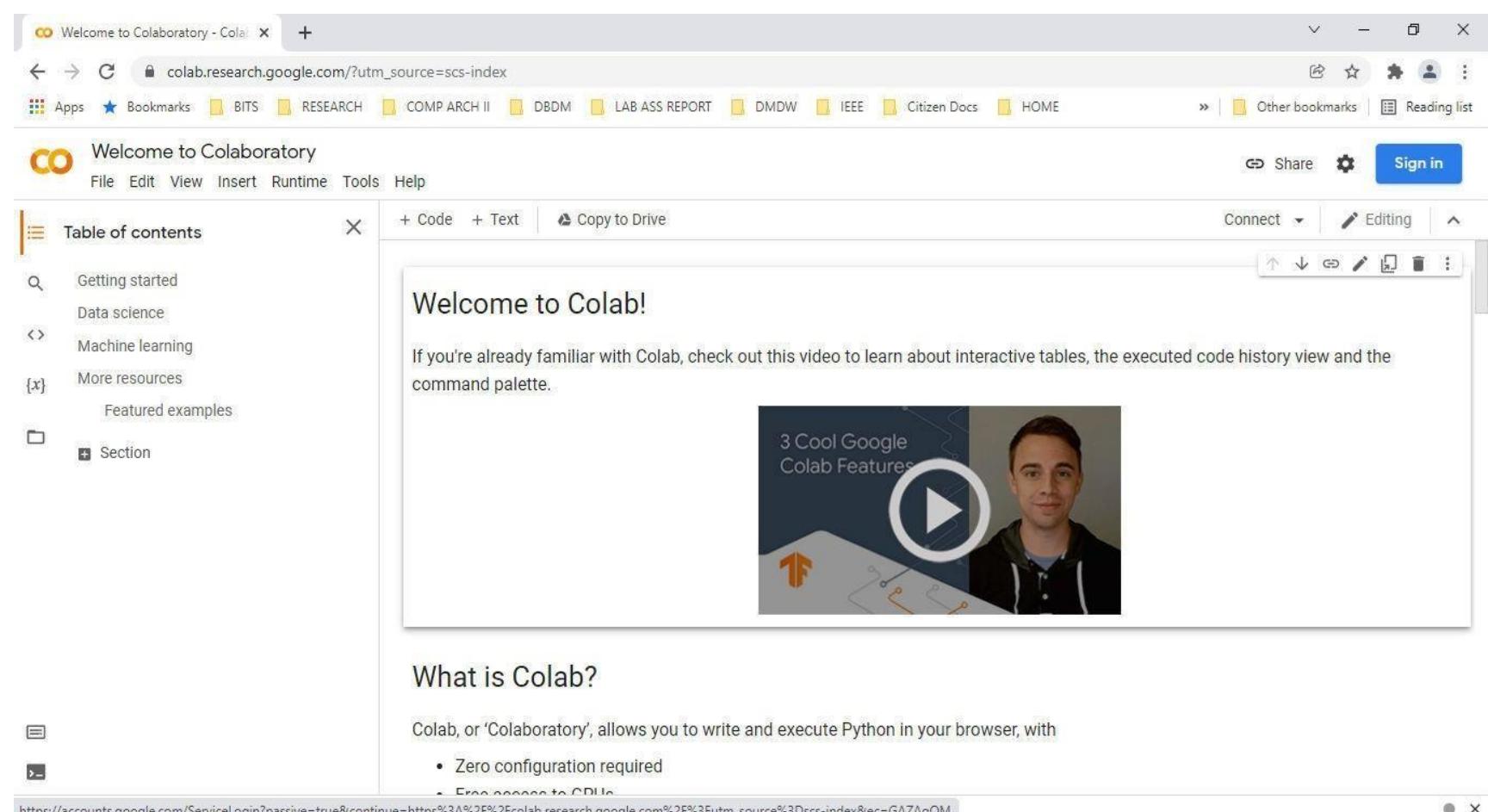
# About Labs



The screenshot shows a virtual lab booking interface for January 2022. On the left, a sidebar menu includes 'Change Lab' (monitor icon), 'View Slots' (house icon), 'Book Slot' (calendar icon with checkmark), 'Resources' (book icon), and an unlabelled icon (square with horizontal lines). The main area displays a calendar grid for January 2022. A blue banner at the top right of the calendar area says 'You need to book a slot 24 hours in advance'. The days of the week are labeled Mo, Tu, We, Th, Fr, Sa, Su. The dates from 1 to 31 are listed sequentially. A red notification bubble in the bottom right corner indicates '1' new message.

## Exercises : In Python & its libraries

# About Labs



Welcome to Colaboratory - Colab

colab.research.google.com/?utm\_source=scs-index

Welcome to Colaboratory

File Edit View Insert Runtime Tools Help

Share Sign in

Table of contents

- Getting started
- Data science
- Machine learning
- More resources
- Featured examples
- Section

+ Code + Text Copy to Drive Connect Editing

## Welcome to Colab!

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.



What is Colab?

Colab, or 'Colaboratory', allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs

[https://accounts.google.com/ServiceLogin?passive=true&continue=https%3A%2F%2Fcolab.research.google.com%2F3Futm\\_source%3Dscs-index&ec=GAZaQm](https://accounts.google.com/ServiceLogin?passive=true&continue=https%3A%2F%2Fcolab.research.google.com%2F3Futm_source%3Dscs-index&ec=GAZaQm)

## Exercises : In Python & its libraries



# Course Overview

1950 - 1956  
=  
**Artificial Intelligence**  
=

if  $a \Rightarrow b$  (True)  
 $b \Rightarrow c$  (True)  
 $\therefore a \Rightarrow c$  (True)

- Term coined by, John McCarthy (1955) & Dartmouth Summer Research Project on Artificial Intelligence (1956)

16<sup>th</sup>

Artificial Neuron

40

Bayes  
Prob

On September 2, 1955, the project was formally proposed by McCarthy, Marvin Minsky, Nathaniel Rochester and Claude Shannon. The proposal is credited with introducing the term 'artificial intelligence'.

The Proposal states<sup>[7]</sup>

“ We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer. ”

$$P(e_1) = 0.2$$
$$\downarrow$$
$$P(n|e_1) = 0.8$$

$$P($$
  
$$1 - P(e_1)$$
  
$$1 - 0.2 = 0.8$$

- Term coined by, *John McCarthy* (1955) & *Dartmouth Summer Research Project on Artificial Intelligence* (1956)

On September 2, 1955, the project was formally proposed by McCarthy, Marvin Minsky, Nathaniel Rochester and Claude Shannon. The proposal is credited with introducing the term 'artificial intelligence'.

The Proposal states<sup>[7]</sup>

“ We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves.

We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

[https://en.wikipedia.org/wiki/Dartmouth\\_workshop](https://en.wikipedia.org/wiki/Dartmouth_workshop) [01 June, 2019]

Natural Lang  
=

Larger Intent, Dream,  
Overconfidence ...

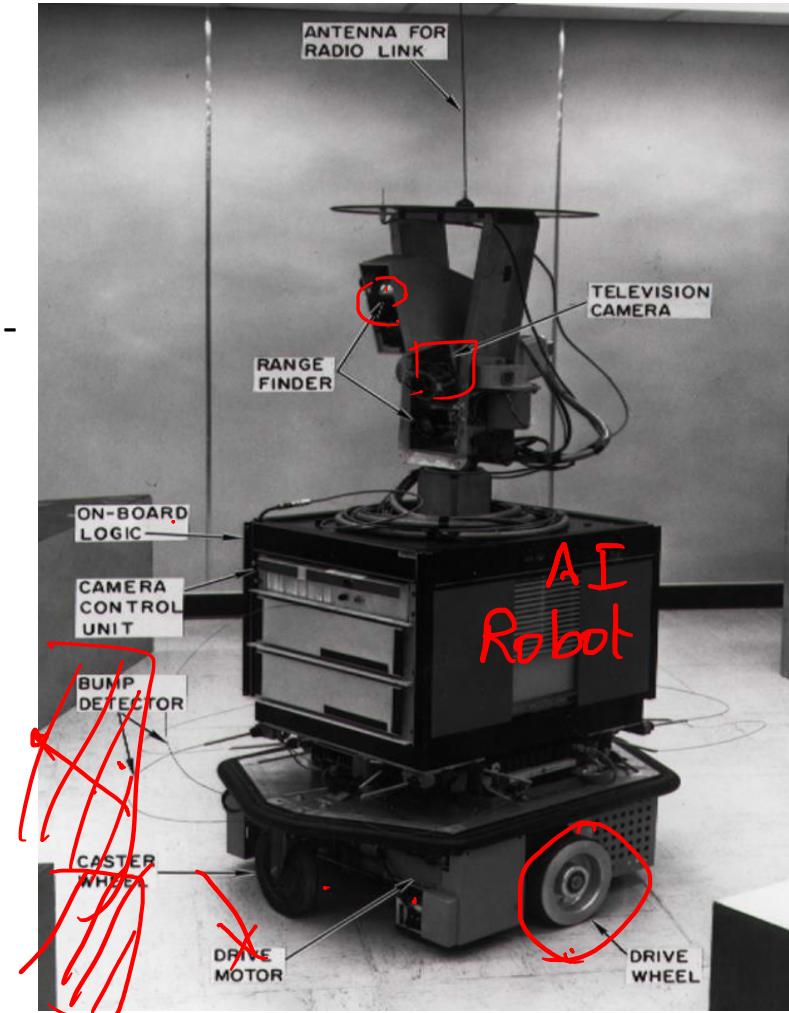
”

## Some Early successes of Dartmouth

Many key projects were initiated after Dartmouth summer project.

**Shakey robot** - First mobile robot to perceive environment & reason about surroundings, actions -

m2 Introduced A\* algorithm to find paths - Hough Transform for image analysis - Used Lisp for programming - visibility graph used for finding shortest paths in the presence of obstacles...



## Some Early successes of Dartmouth

### 1<sup>st</sup> Expert system

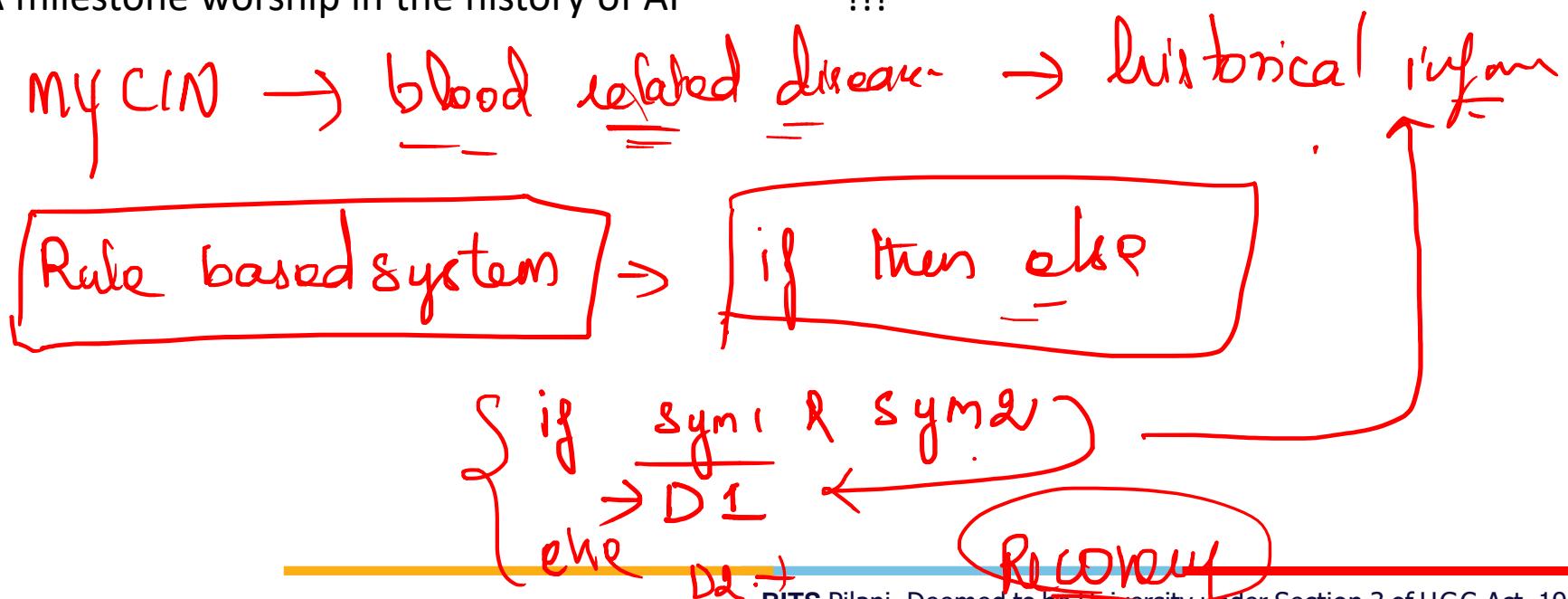
DENDRAL -

Attempted to encode the domain expertise in molecular biology as an expert system

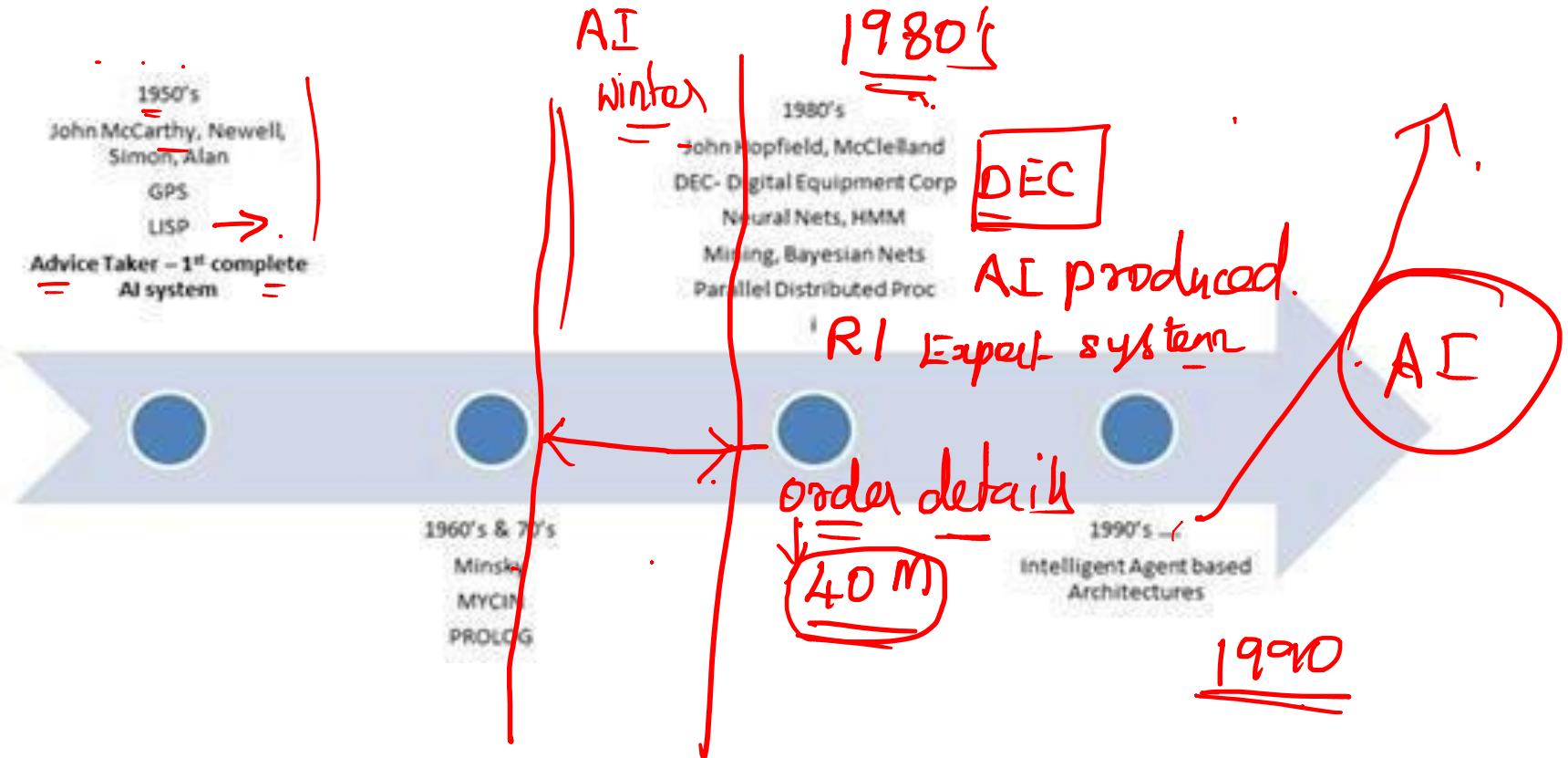
Led to the creation of expert systems for various other domain, including medical.

A milestone work in the history of AI

!!!



# A brief history of AI

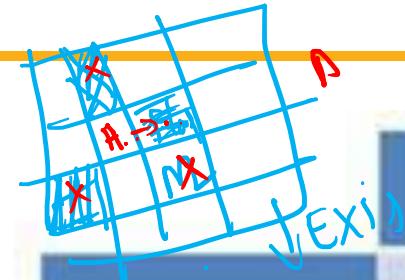




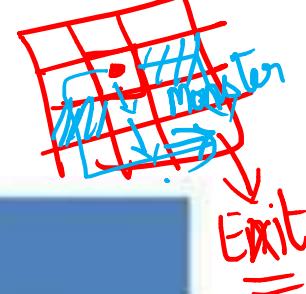
# Perspectives of AI

# Definitions

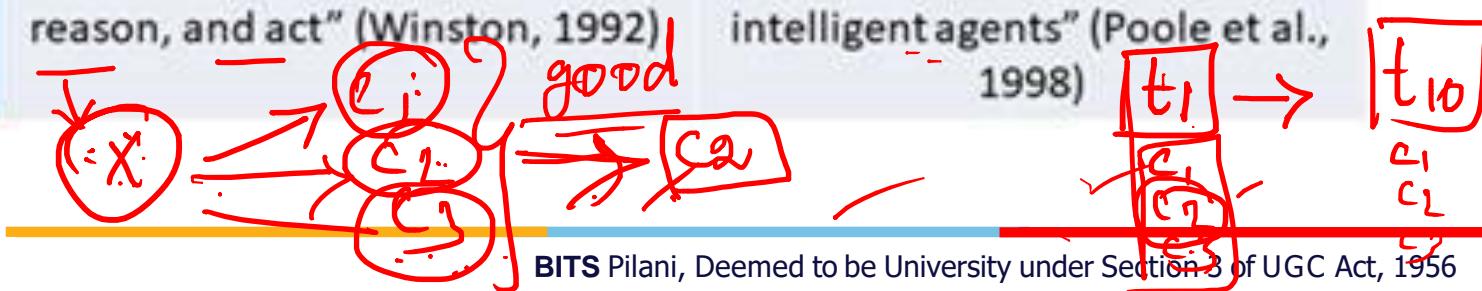
## Perspective of AI



ML  
MR  
MU  
MD



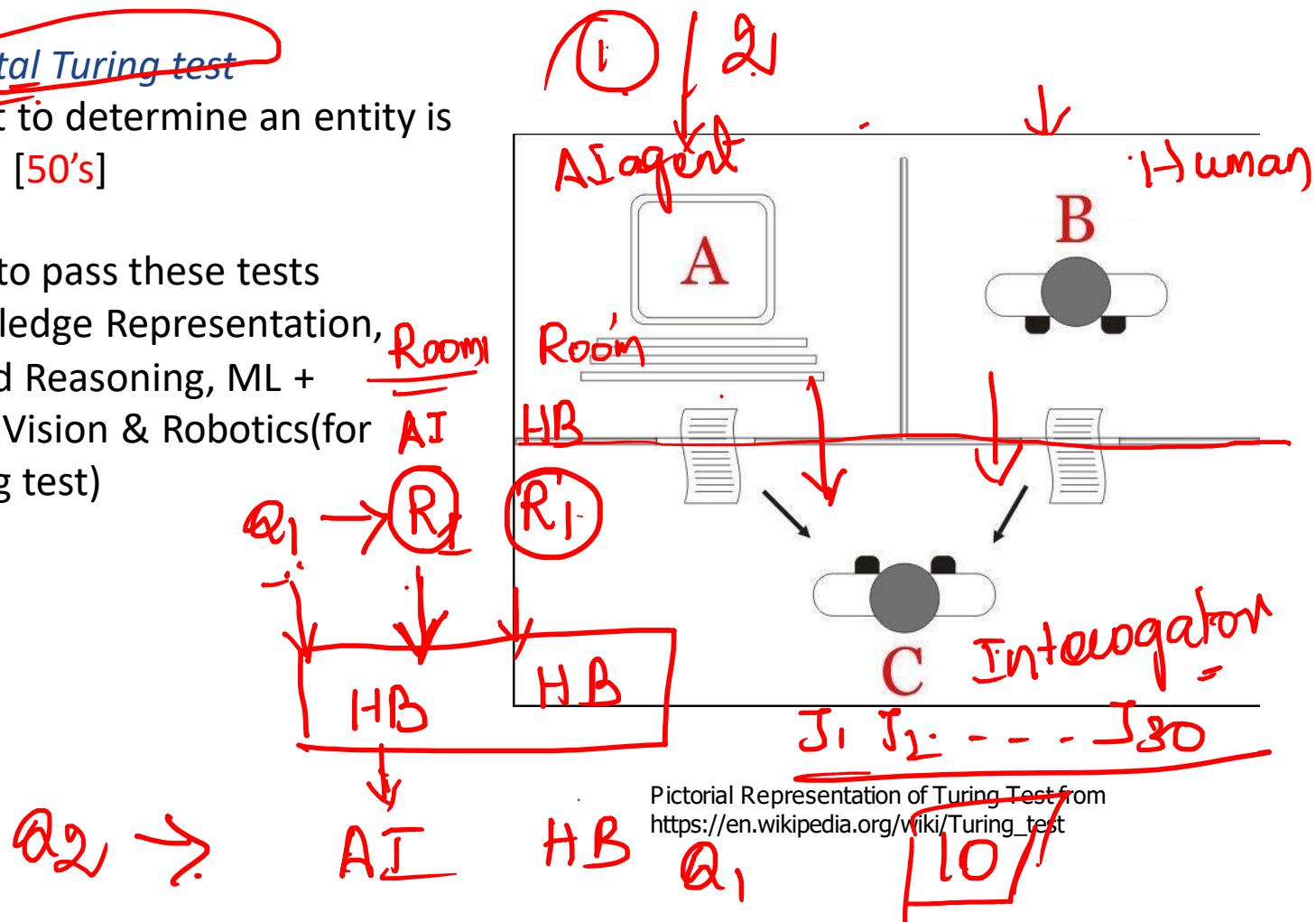
	Thought / Reasoning	Acting
Human Performance	<p>① THINKING HUMANLY</p> <p>"[The automation of] activities that we associate with <u>human thinking</u>, activities such as decision-making, problem solving, learning, ... " (Bellman, 1978)</p>	<p>② ACTING HUMANLY</p> <p>"The art of creating <u>machines</u> that perform <u>functions</u> that require <u>intelligence</u> when performed by people" (Kurzweil, 1990)</p>
Rational Performance	<p>③ THINKING RATIONALLY</p> <p>"The study of <u>computations</u> that make it possible to perceive, reason, and act" (Winston, 1992)</p>	<p>④ ACTING RATIONALLY</p> <p>"Computational intelligence is the study of the design of intelligent agents" (Poole et al., 1998)</p>



# Acting Humanly → Sophia

## Turing Test Approach

- Turing Test & Total Turing test  
(operational test to determine an entity is intelligent / not) [50's]
- Skills necessary to pass these tests
  - NLP, Knowledge Representation, Automated Reasoning, ML + Computer Vision & Robotics(for total turing test)



# Passing the Turing Test

- 2014 - Royal Society ( London ) - Sixteenth Anniversary of Alan Turing -
- Chabot - Eugene Goostman - Pretended to be a thirteen-year-old Ukrainian boy
  - Passed the turing test for the first time
  - 10/30 Judges believed the response is from human
- *Turing predicted in 50 years time, computers can be programmed to play imitation game in which an average interrogator fails to identify the machine 70% time in a 5 mins questioning*

not AI agent

10 / 30

# Passing the Turing Test



## Transcript of a chat

EUGINE - a thirteen-year-old Ukrainian boy, chats

JUDGE: Hello.

EUGENE: Hello, I'm really glad to have the chance to chat with you! My guinea pig Bill sends his regards too!

JUDGE: Is Bill a male or a female?

EUGENE: Ask Bill personally, please.

JUDGE: Well I'd rather talk to you. What is your name?

EUGENE: Call me Eugene. I am glad to talk to you!

JUDGE: My name is Jane and I am female. How about you? What's your gender?

EUGENE: I'm a male. A "guy" I'd say.

JUDGE: Pleased to meet you Eugene. What's the weather like where you are?

EUGENE: Let's get on with our conversation!

JUDGE: Don't you like talking about the weather?

EUGENE: All these talks about weather is a waste of time.

JUDGE: What would you like to discuss?

EUGENE: I don't know. Better tell me more about yourself!

# Acting Humanly

## Turing Test Approach

---

Some Definitions of AI:

*“The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil, 1990)*

*“The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight, 1991)*

# Thinking Humanly

## Cognitive Modelling Approach

---

- How do we capture human thinking to implement?
  - Introspection
  - Psychological Experiments
  - Brain Imaging
- System : “*General Problem Solver*” (*Newell and Simon, 1961*)
  - Designed to work as a universal problem solver
  - Problems represented by horn clauses
  - First AI Machine which has KB + Inference separation
  - Authors focus on this is on comparing the trace of its reasoning steps to traces of human subjects solving the same problems
- Growth of Cognitive science and AI supports each other

## Cognitive Modelling Approach

---

Some Definitions of AI:

*“The exciting new effort to make computers think . . . machines with minds, in the full and literal sense.” (Haugeland, 1985)*

*“[The automation of] activities that we associate with human thinking, activities such*

*as decision-making, problem solving, learning . . .” (Bellman, 1978)*

# Thinking Rationally

## “Laws of Thought” Approach

- Invention of Formal Logic, Greek Philosopher **Aristotle**, Third century BC.
- Introduced syllogisms, providing argument structures

*In all boring classes, students sleep It  
is a boring class*

*Students sleep in this class [ Are you ?  
]*

- Field of Logics gave rise to codifying rational thinking
  - When elements are ‘**things**’, we reason about things

### Hurdles to the idea :

- (1) Not everything can be logically coded
- (2) no provably correct action at a moment
- (3) Exhaustive computational resources

# Acting Rationally

## The Rational Agent Approach

---

- An agent is an entity that perceives and acts  
*This course is about designing rational agents*
- Abstractly, an agent is a function from percept histories to actions:  $[f: P^* \rightarrow A]$
- For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance
- Computational limitations make perfect rationality unachievable
- Design best program for given machine resources

# Acting Rationally

## The Rational Agent Approach

---

- Rational behaviour: doing the *right thing*
- The *right thing*: that which is expected to maximize goal achievement, given the available information
- Rational behaviour is not just about correct inference / thinking, skills needed to pass turing test etc.

(adv) : More General - Correct inference is just a thing

(adv) : More amenable for scientific developments, as the rational behaviour is better defined than human thinking and behaviour

**Required Reading:** AIMA - Chapter #1

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



**BITS** Pilani  
Pilani Campus



# **Artificial & Computational Intelligence**

**DSE CLZG557**

**M1 : Introduction  
&**

**M2 : Problem Solving Agent using Search**

Indumathi V  
Guest Faculty,  
BITS - WILP

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

# Traveller's Problem

No data  $\Rightarrow$  ① Tries all possible paths



achieve

lead

② Randomization

③ Right  $\Rightarrow$  DE, WP, RP



Exploration Phase

Trip 1000

④

Utility

$$\underline{P_1} > \underline{P_2}$$

$\downarrow$   
(Numerical  
mean)

Trip 1  
Trip 2

## Traveller's Problem



Reasoning Under Uncertainty  $\rightarrow$  Probabilities

$P_{0.001}$

$1001$

$P_1, P_2, P_3, P_4$

$P_5 \rightarrow 0$

$P_2 \rightarrow 0.2$      $P_1 \rightarrow 0.1$

S2: Least info mn

$S \rightarrow D : (P_1, P_2, P_3, P_4) \rightarrow$

$D \stackrel{1001}{=} \rightarrow$  best utility  $\rightarrow$

Rationality  
Reasoning

Partial info mn

$P_2, P_3, P_4$

$P_3$

$P_1 \rightarrow BP$

$P_2$

$P_3$

$P_4$

invalid  
too

$P_1 \rightarrow less weight$

$P_2 \rightarrow 0.2$      $P_1 \rightarrow 0.1$

## Traveller's Problem

S<sub>B</sub> :- Weather condition flip  
 info → change car speed fraction  
topic



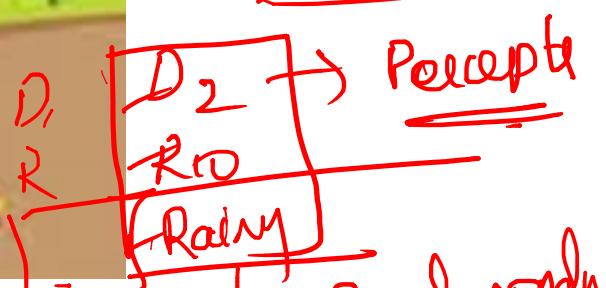
Weather

→ Observed event  
 [HMM]

Weather → feature  
time oriented

D <sub>1</sub>	D <sub>1</sub> . . . . .	D <sub>2</sub>
Rider	R <sub>2</sub> . . . . .	R <sub>4</sub>
Sunny	S - - - - -	Rainy

road condition?

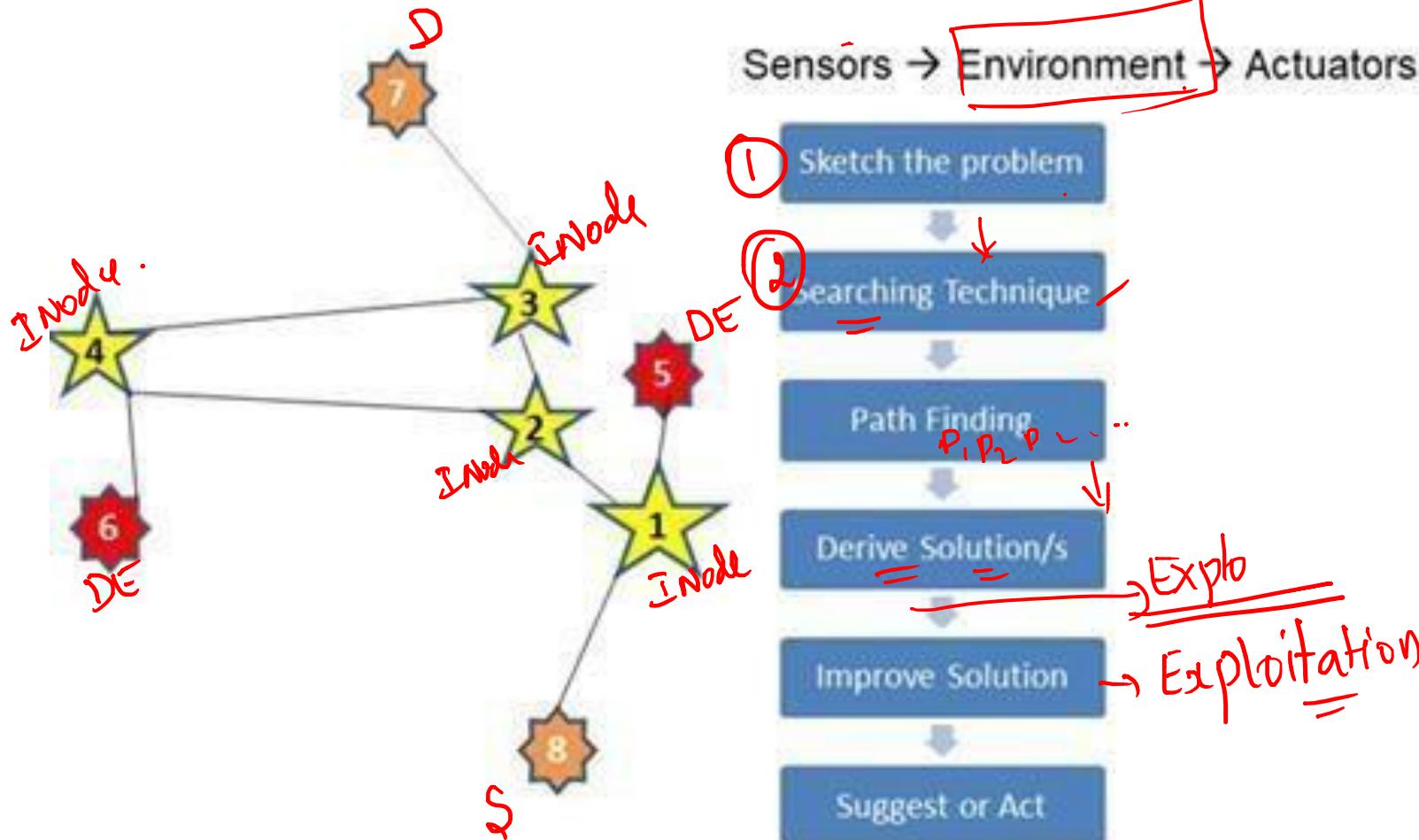


→ Road condn

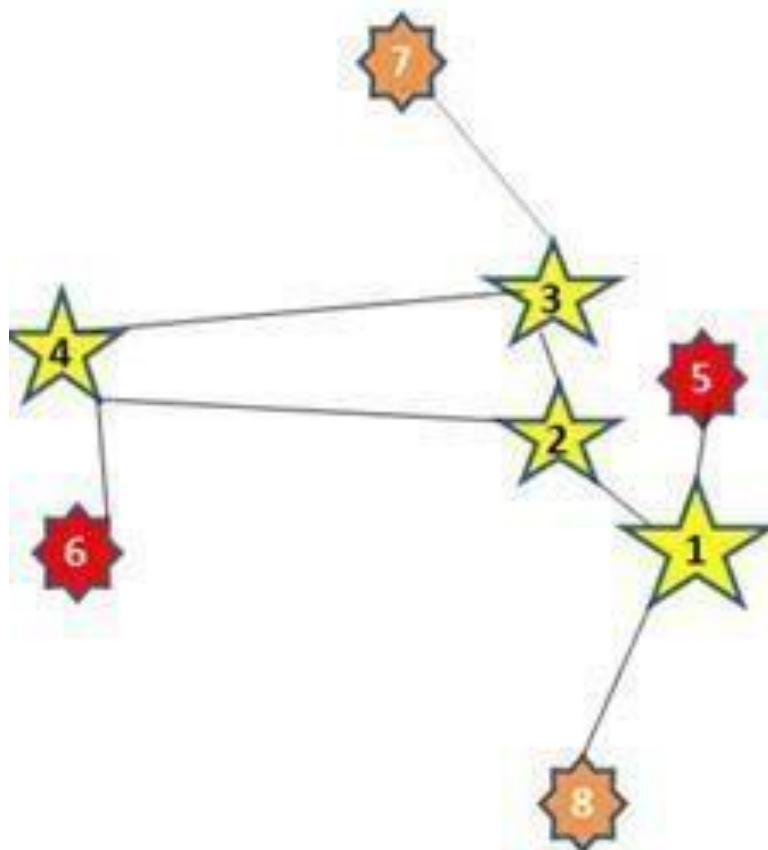
Explo → learning from Unknown Env  
Traveller's Problem

Exploitation

→ optimize the  
Path

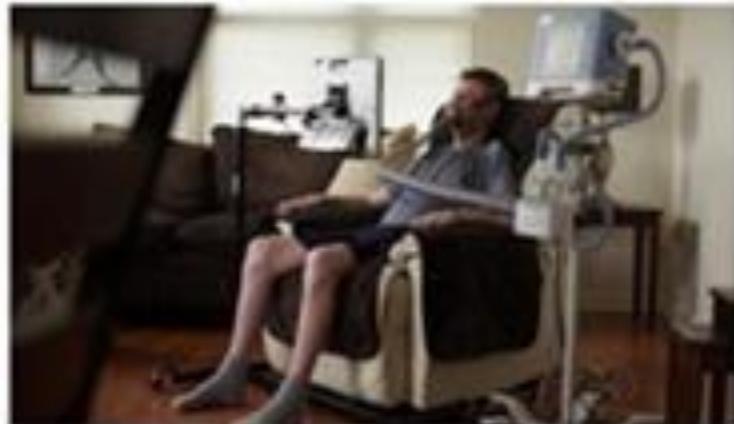


# Traveller's Problem



Sensors → Environment → Actuators

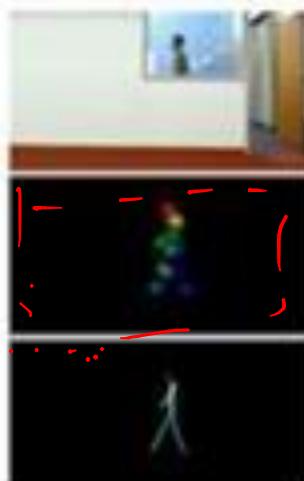
- Sketch the problem
- Searching Technique
- Path Finding
- Derive Solution/s
- Improve Solution
- Suggest or Act



Voice cloning

ALS

Lyrebird's Project Re-Voice



## AI in Culinary Field

---



Spyce

---

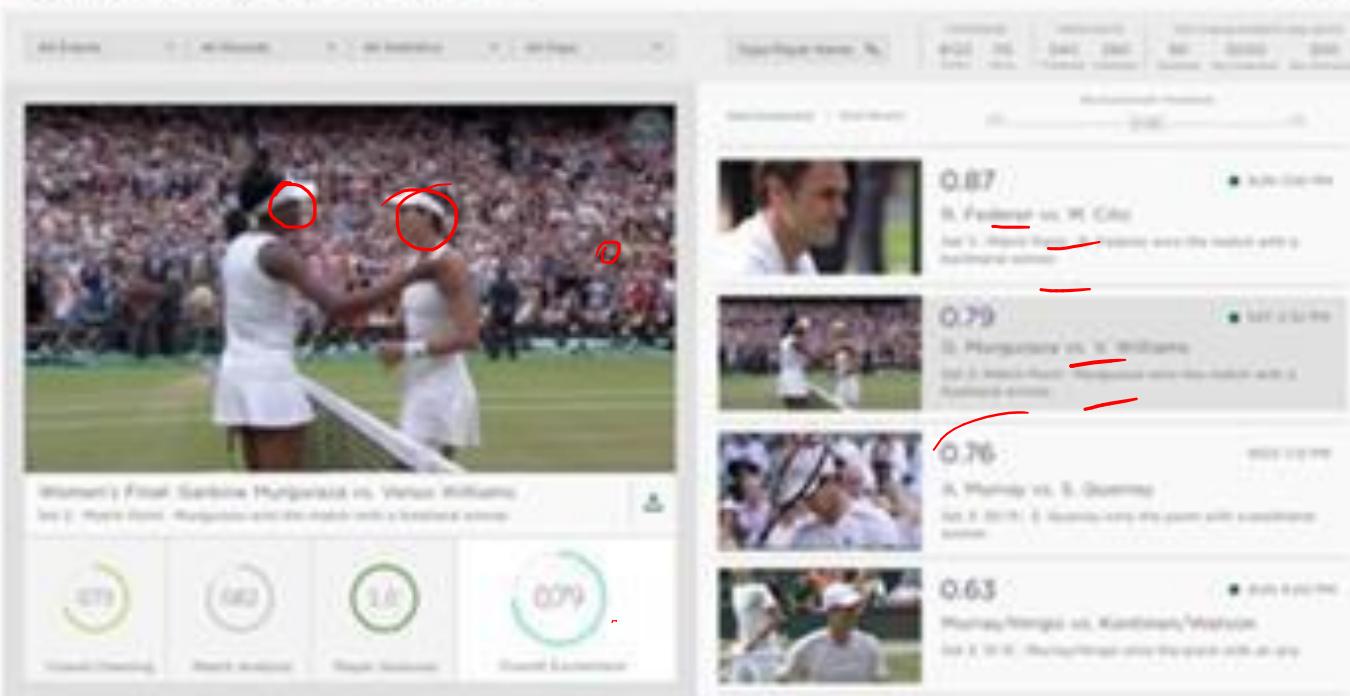
## AI in Transportation



# Natural Lang Processing

## Wimbledon AI Highlights

IBM



The screenshot shows a grid of five tennis highlights from the 2017 Wimbledon final. Each highlight includes a thumbnail image, a confidence score, the match name, and a brief description.

Highlight	Confidence Score	Match Name	Description
1	0.87	R. Federer vs. W. Cai	Set 2: Federer wins the opening game with the most aces in tournament history.
2	0.79	S. Halepova vs. S. Williams	Set 2: Williams wins the opening game with the most aces in tournament history.
3	0.76	A. Kerber vs. S. Halepova	Set 2: Kerber wins the opening game with a maximum serve.
4	0.63	Murray/Francis vs. Kontinen/Mashurov	Set 2: Murray/Mashurov wins the point with an aces.

Computer Vision  
NLP  
ML  
Speech Recognition  
Automation



# Rational Agents

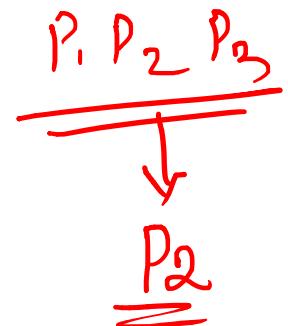
## Design Principles & Techniques

	Thought / Reasoning	Acting
Human Performance	<b>THINKING HUMANLY</b> [The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning, ... " (Bellman, 1978)	<b>ACTING HUMANLY</b> "The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)
Rational Performance	<b>THINKING RATIONALLY</b> The study of computations that make it possible to perceive, reason, and act" (Winston, 1992)	<b>ACTING RATIONALLY</b> "Computational intelligence is the study of the design of intelligent agents" (Poole et al., 1998)

# Acting Rationally

## The Rational Agent Approach

- An agent is an entity that perceives and acts  
*This course is about designing rational agents* → doing right thing
- Abstractly, an agent is a function from percept histories to actions:  $[f: P^* \rightarrow A]$
- For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance
- Computational limitations make perfect rationality unachievable
- Design best program for given machine resources



## Properties of Rational Agent

- Omniscience : Expected Vs Actual Performance
- Learning Capability : Apriori Knowledge
- Autonomous in decision making: An agent is autonomous if its behaviour  
is determined by its own experience (with ability to learn and adapt)

KB

→ Maximum achievable objective

## Intelligent Agent

## Vacuum cleaner Robot



controlled environment



Exactly 2 Room → Room A = Room B

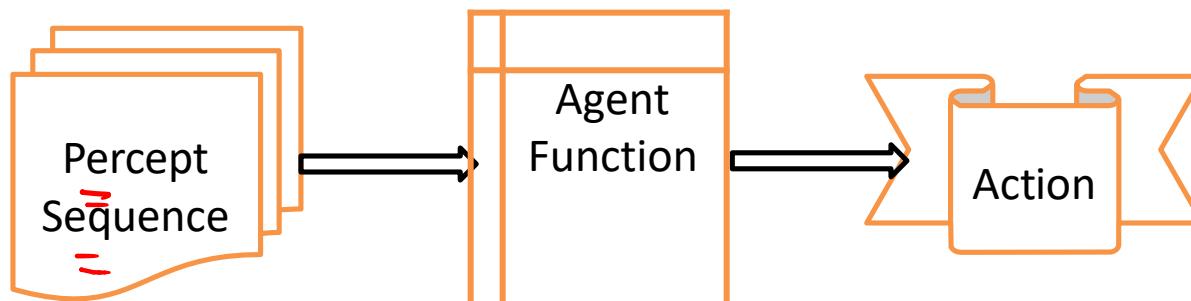
Rational Agent is one that acts to achieve the best outcome or the best expected outcome even under uncertainty

\* 3 actions

↳ M<sub>L</sub>, M<sub>R</sub>, Suck

obj: → clean both the  
Room → Room A + Room B

Maps / Tabulated / Programmed



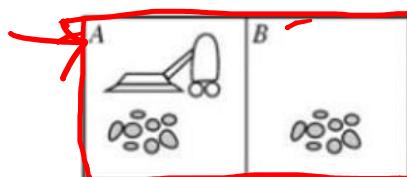
$$R_0 \rightarrow A$$

$$R_A \rightarrow \text{Dirty}$$

$$R_B \rightarrow \text{Dirty}$$

$$R_{in} \rightarrow B$$

$\left[ A, \text{clean} \right], \left[$



$$D \rightarrow 1$$

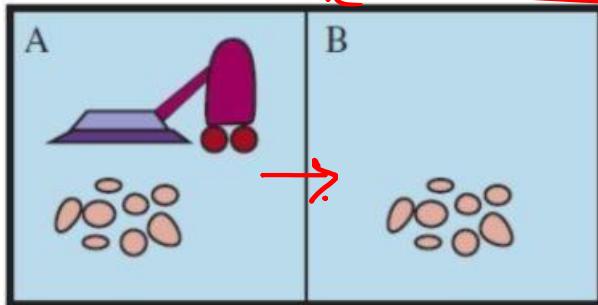
$$ND \rightarrow 0$$

$$\begin{array}{c} A \rightarrow 1 \\ B \rightarrow 0 \end{array}$$

Percept sequence	Action
$t_1$ : [A, Clean]	Right
$t_2$ : [A, Dirty]	Suck
$t_3$ : [B, Clean]	Left
$t_4$ : [B, Dirty]	Suck
$t_5$ : [A, Clean], [A, Clean]	Right
$t_6$ : [A, Clean], [A, Dirty]	Suck

	A	R <sub>B</sub>	R <sub>loc</sub>
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

## Intelligent Agent



[A, clean] → MR  
[A, Dirty] → Suck

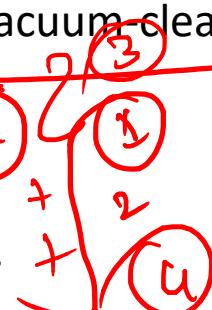
- Percepts: location and contents, e.g., [A, Dirty] → Suck
- Actions: Left, Right, Suck, NoOp → R

$$f(PM_1) + f(PM_2)$$
$$\downarrow$$
$$0.40 \times + 0.6(PM)$$

Performance measure: An objective criterion for success of an agent's behaviour

E.g., performance measure of a vacuum cleaner agent

- » amount of dirt cleaned up
- » amount of time taken
- » amount of electricity consumed
- » amount of noise generated, etc.



$$PM_1 + PM_2 + PM_3 + PM_4$$

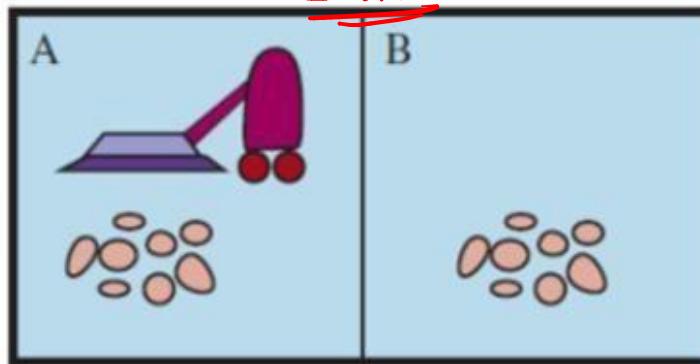
TIV

VIAV

PEAS Design

# Intelligent Agent

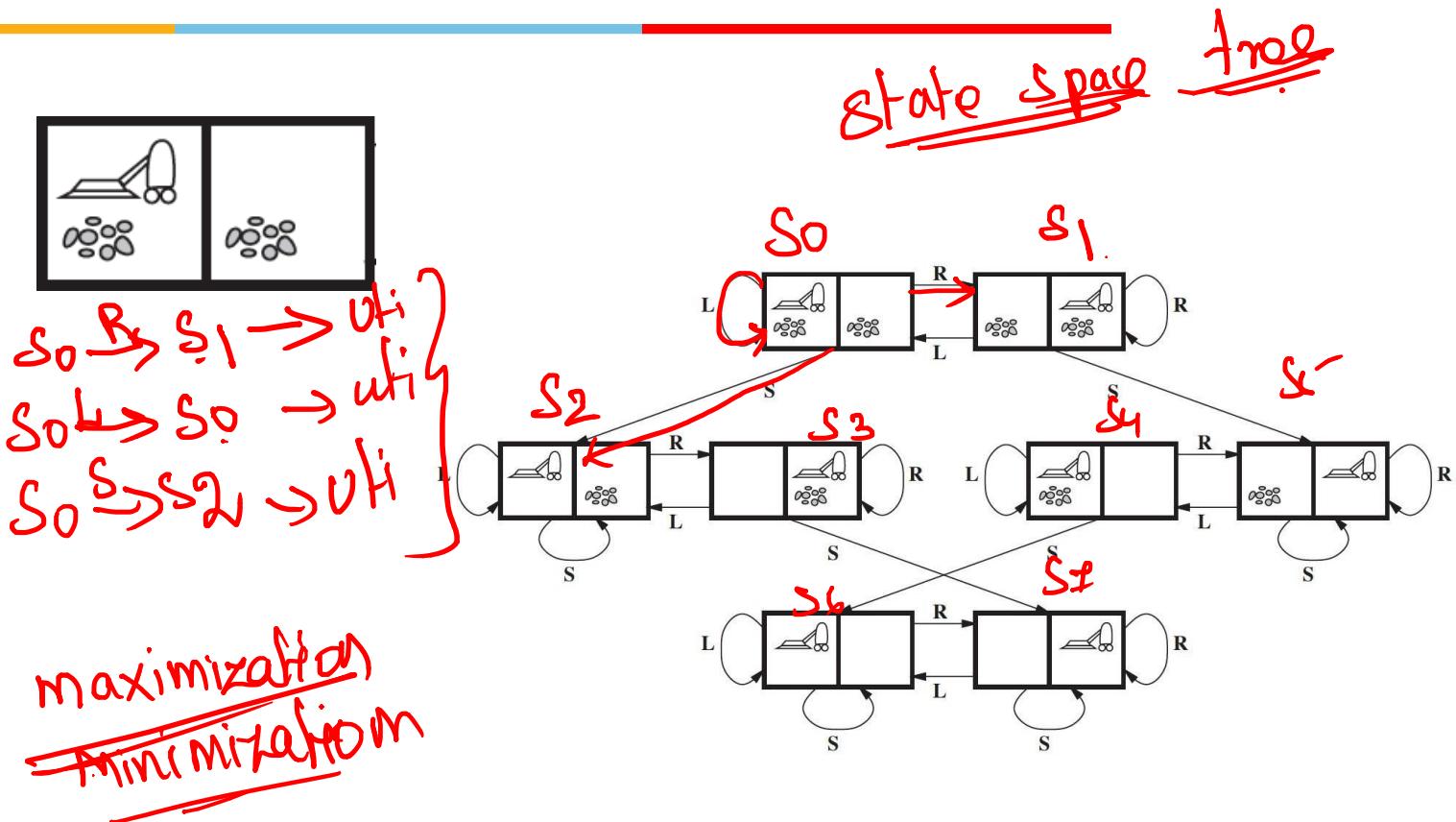
Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
<u>P</u> [A, Clean], <u>P</u> [A, Clean]	Right
<u>P</u> [A, Clean], <u>P</u> [A, Dirty]	Suck
:	:
[A, Clean], [A, Clean], [A, Clean]	Right
<u>P</u> [A, Clean], [A, Clean], [A, Dirty]	Suck
:	:



$t_1$  [A,dirt]  $\rightarrow$  Suck  
 $t_2$  [A,clean]  $\rightarrow$  Right  
 $[B,dirt]$   $\rightarrow$  Suck.

(10)

# Vacuum World Problem



**Required Reading:** AIMA - Chapter #2

Note : Some of the slides are adopted from AIMA TB materials

Thank You for all your Attention



# Artificial & Computational Intelligence

**AIML CLZG557**

**M1 : Introduction  
&**

**M2 : Problem Solving Agent using Search**

Indumathi V  
Guest Faculty,  
BITS - WILP

**BITS Pilani**

Pilani Campus



# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

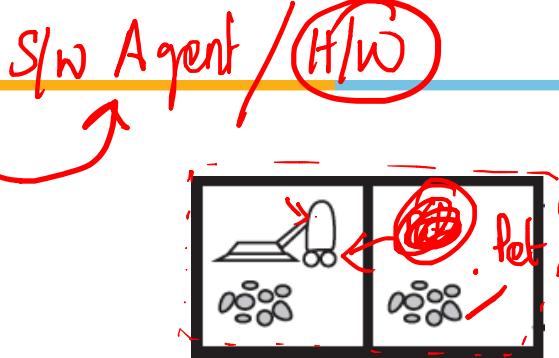
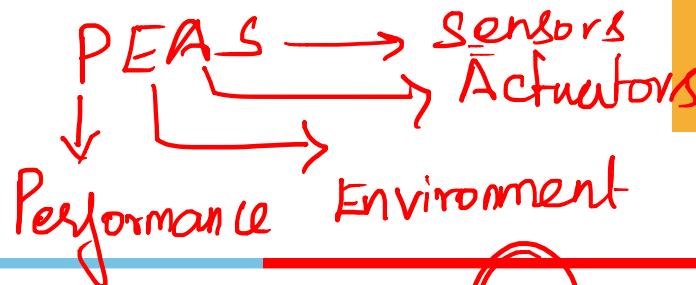
# Learning Objective

---

At the end of this class , students Should be able to:

1. Design problem solving agents
2. Create search tree for given problem
3. Apply uninformed search algorithms to the given problem
4. Compare performance of given algorithms in terms of completeness, optimality, time and space complexity
5. Differentiate for which scenario appropriate uninformed search technique is suitable and justify

## Vacuum World Problem

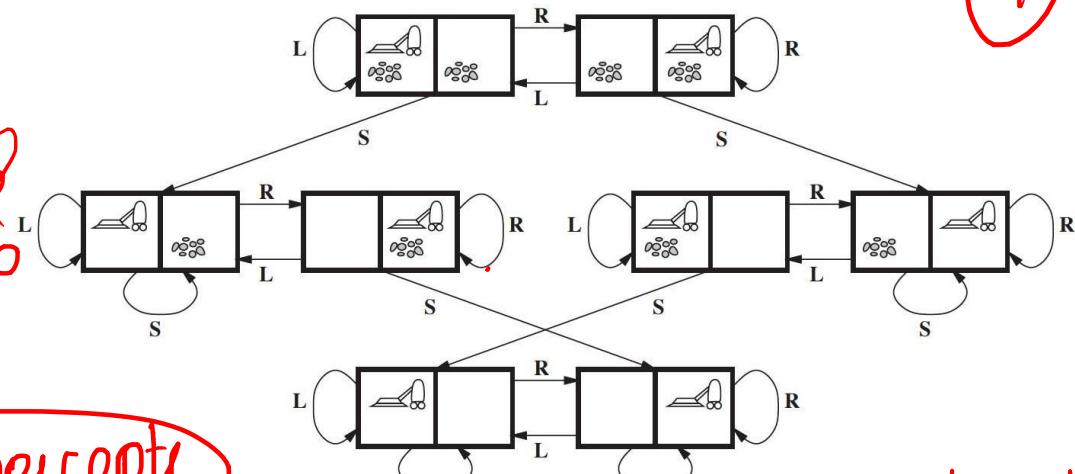


① P ✓

② E → Presence & absence  
— Dirt  
→ Presence & absence

③ Sensors → ③ Var percepts

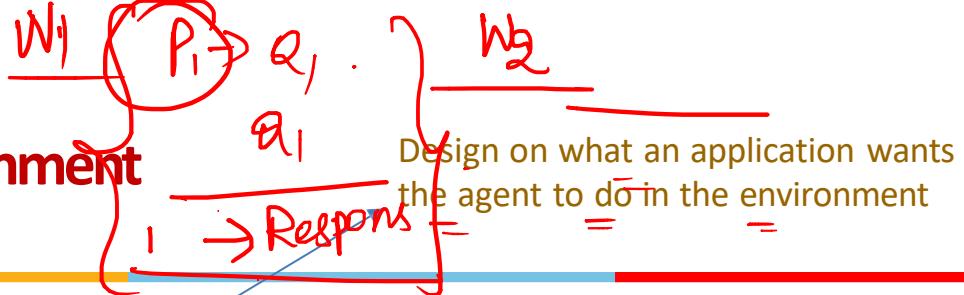
④ Actuator → Movement  
ML, MR, Suck



A table titled "Status of Room A" with columns for "location", "status", and "movement".

location	S-B	A → 1
Clear	0/1	B → 0
Dirt		

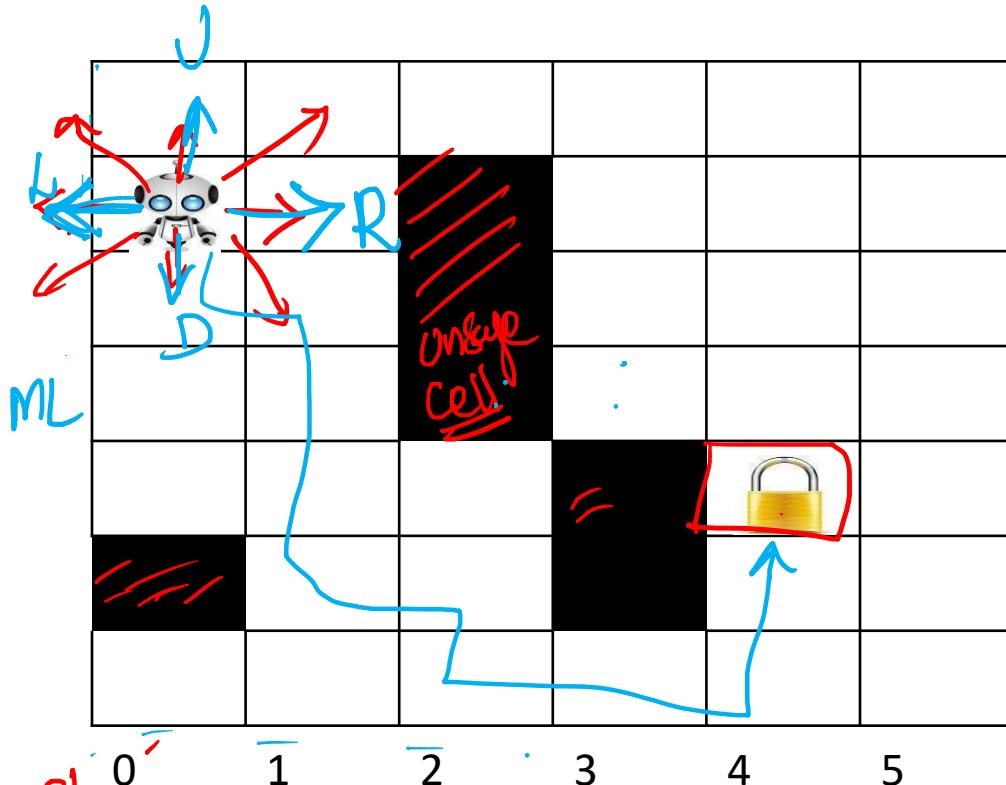
## PEAS Environment



Agent	Performance	Environment	Sensors	Actuators
① Medical diagnosis system chat bot. Re	Healthy patient, reduced costs	Patient, hospital, staff	Keyboard entry of symptoms, findings, patient's answers	Display of questions, tests, ② diagnosis, treatments, referrals → Slow Agent
② Satellite Image analysis system S/I	Correct image categorization Img 1 → Mountain Img 2 → River Img 3 → Land	Downlink from orbiting satellite H/W	Color pixel analysis I/P	Display of scene categorization
③ Interactive English tutor MS Team CANVAS	Student's score on test	Set of students, testing agency	Keyboard entry	Display of exercises, suggestions, corrections

## Path finding Robot - Lab Example

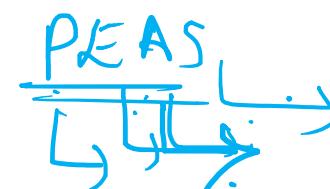
~~4° degree go~~



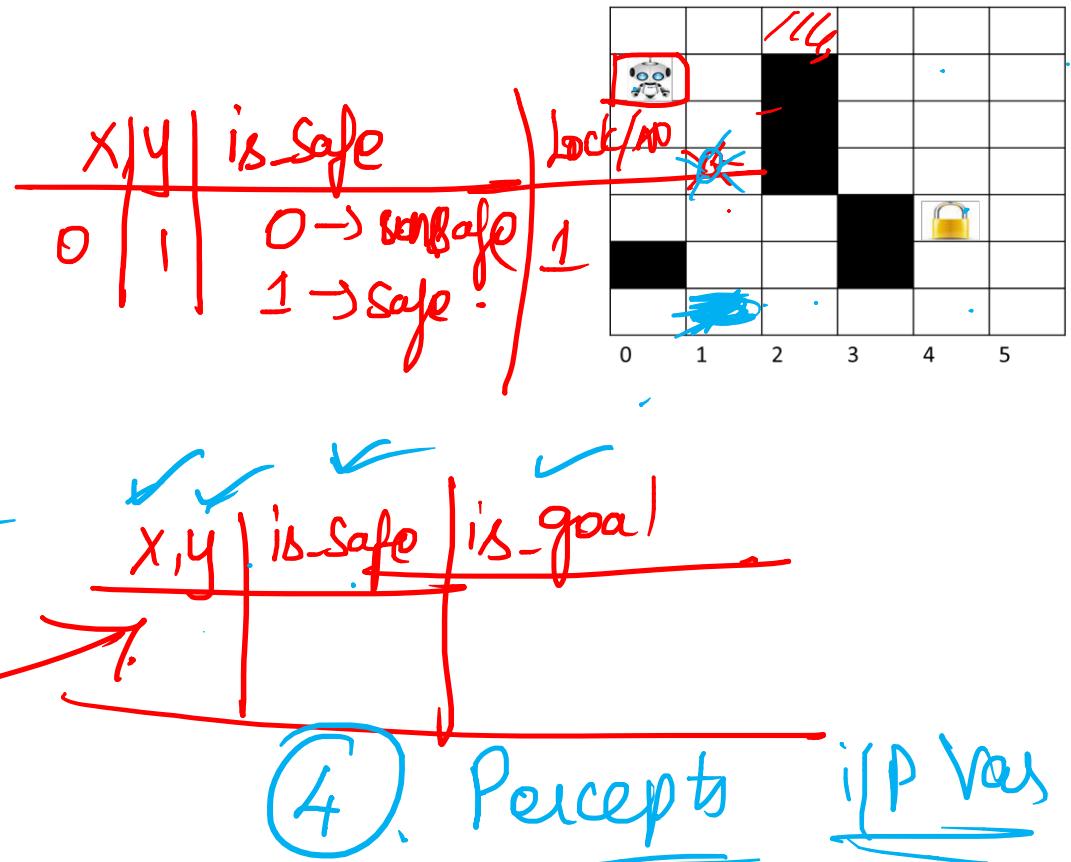
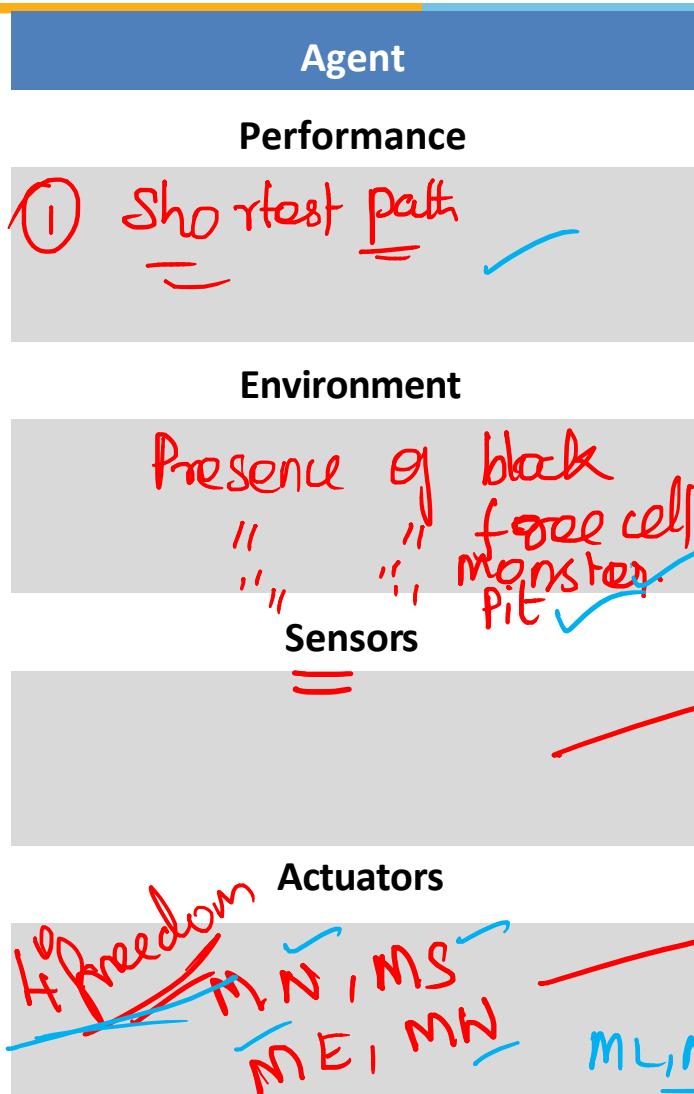
P EAS

0	↓	↓	↓	↓
1	Shortest Path			
2	+ min hit on wall			
3	<u>Environment</u>			
4	↳ Presence of black cell			
5	↳ " " Free			
6	↳ <u>Wall</u>			

A S



# PEAS Environment



# Dimensions of Task Environment



## ① Sensor Based:

- Observability : Full Vs Partial

## ② Action Based:

- Dependency : Episodic Vs Sequential

## ③ State Based:

- No.of.State : Discrete Vs Continuous

## ④ Agent Based:

- Cardinality : Single Vs MultiAgent

## ⑤ Action & State Based:

- State Determinism : Deterministic Vs Stochastic | Strategic
- Change in Time : Static Vs Dynamic

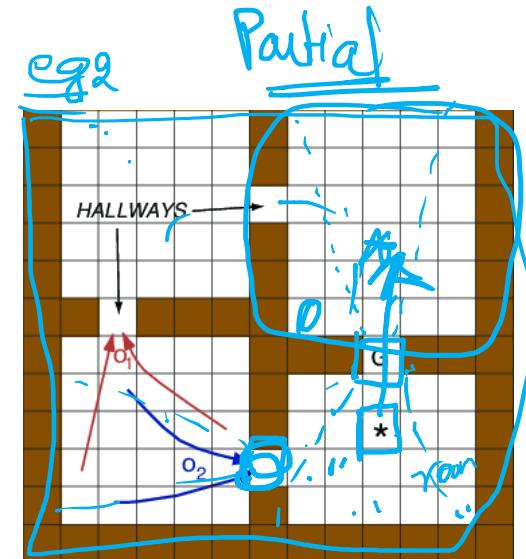
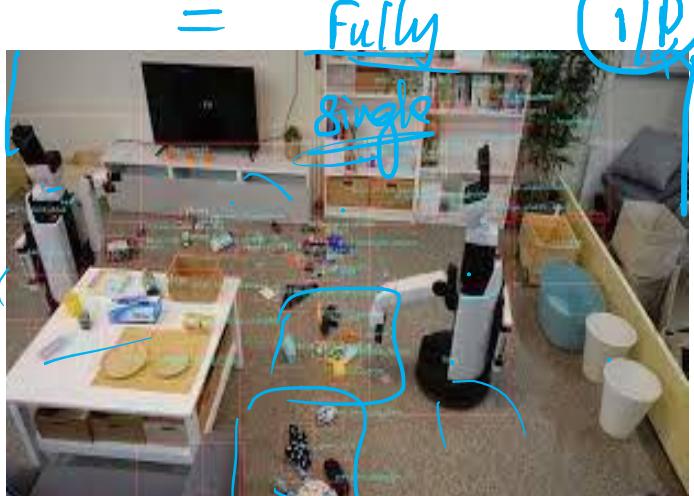
# Task Environment

A rational agent is built to solve a specific task. Each such task would then have a different environment which we refer to as **Task Environment**

Based on the applicability of each technique for agent implementation its task environment design is determined by multiple dimension

## ① Sensor Based:

➤ Observability : Full Vs Partial



eg 1 // Controlled

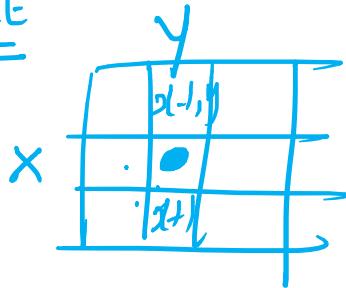
# Task Environment

## Action Based:

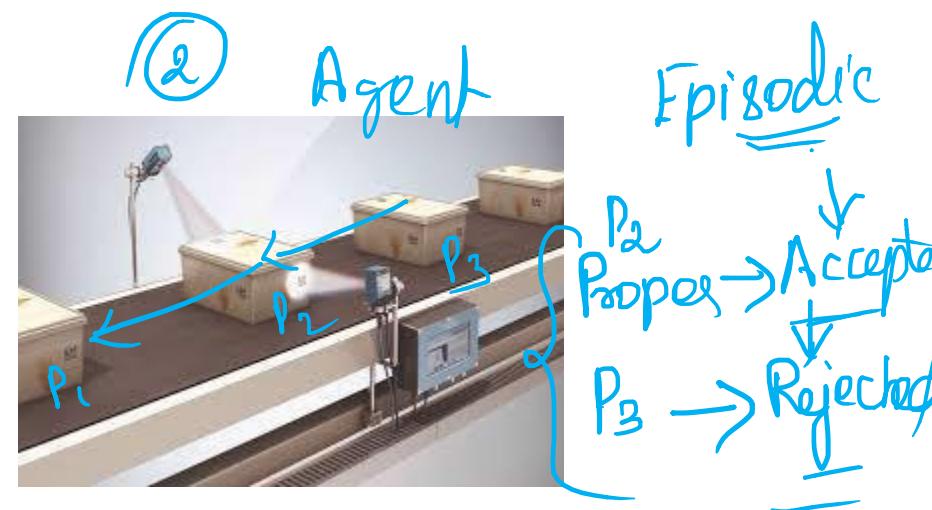
- Dependency : Episodic Vs Sequential



MAZE



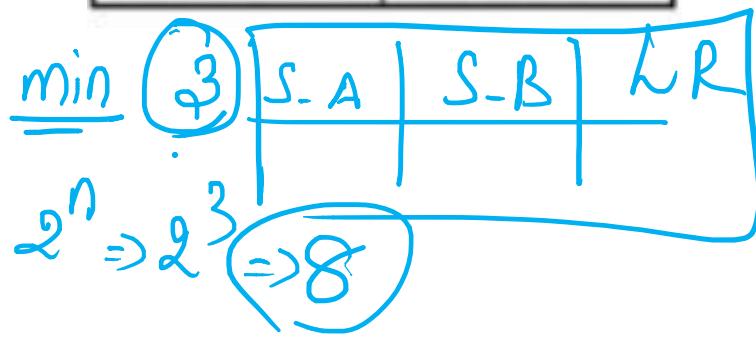
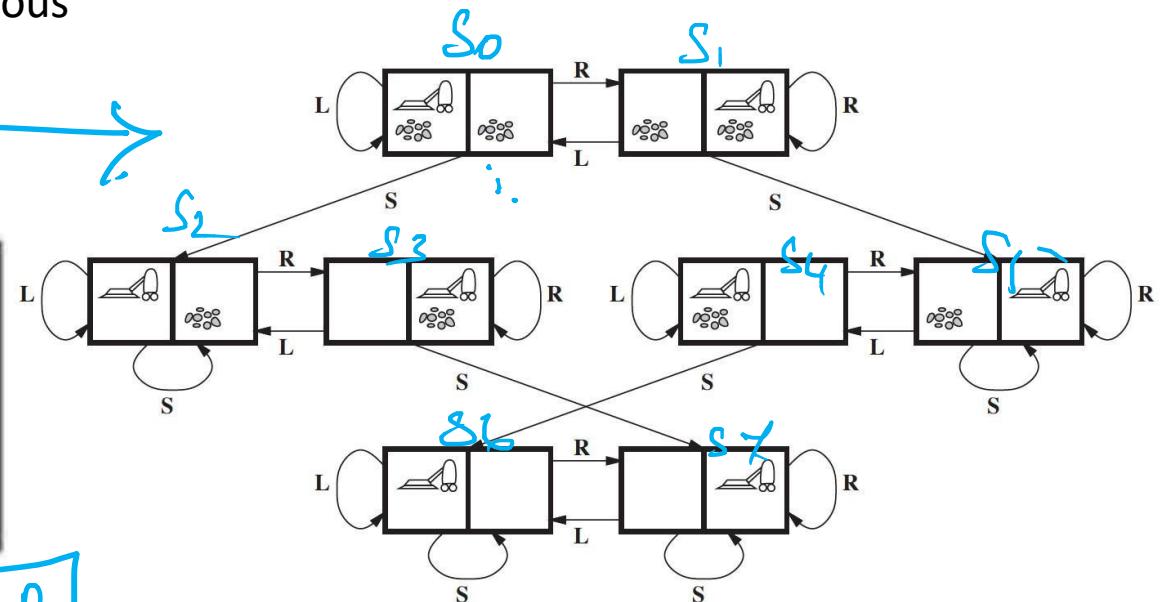
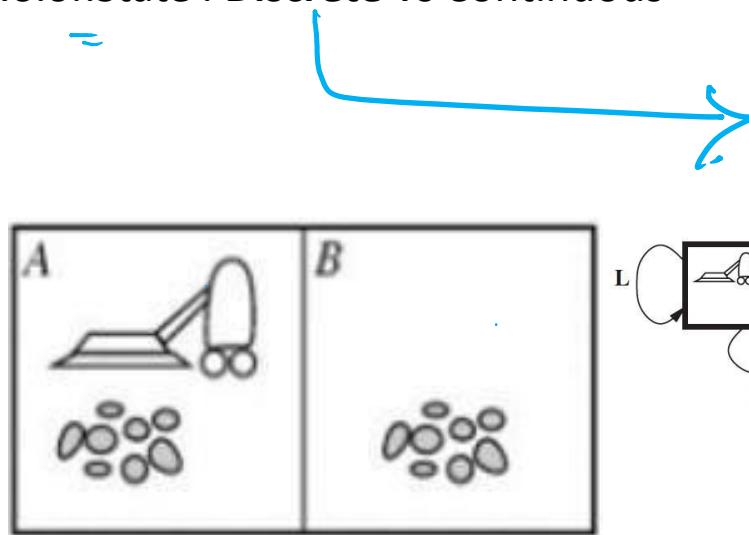
$x+1, y$   
 $x-1, y$



# Task Environment

**State Based:**

- No. of State : Discrete Vs Continuous



(8)

## Task Environment

### State Based:

- No.of.State : Discrete Vs Continuous



VS.



n of  
num.  
2^n

## Task Environment

$$T(s_0, a_1) = s_1$$

Deterministic

$$T(s_0, a_1) = s_1 \quad 0.6$$

$$T(s_0, a_1) = s_2 \quad 0.4$$

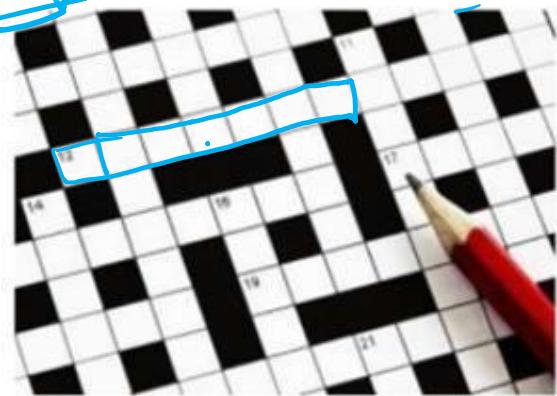
Probability

Action & State Based:

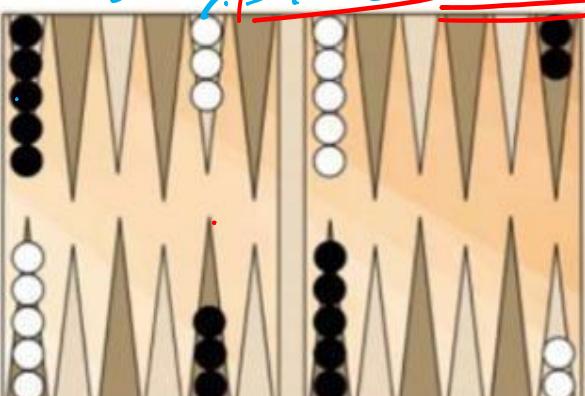
➤ State Determinism : Deterministic Vs Stochastic | Strategic

(If the environment is deterministic, except for the actions of other agents, then the environment is strategic)

eg  $T(s_0, m_1) \rightarrow$  Board  $\xrightarrow{S_1}$  Game + Rolling dice  $\xrightarrow{S_2}$  Game + No Dice  $\xrightarrow{2}$  eg - stochastic amount of luck



Puzzles  $D \leftarrow +$  eq 3



~~s1, s2~~  $S_1 \xrightarrow{v} \text{stochastic}$   
 $S_2 \xrightarrow{v} \text{strategic}$



Ro-Jai  
 Randomness  
 Uncertainty Prob

# Task Environment

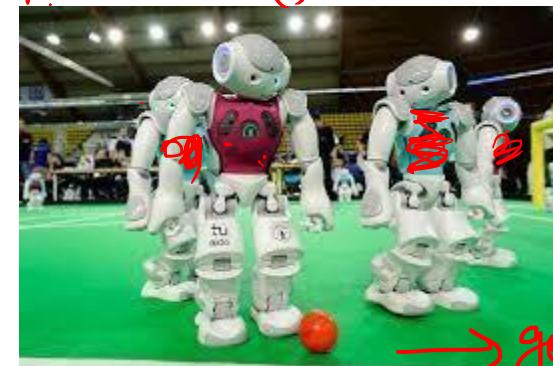
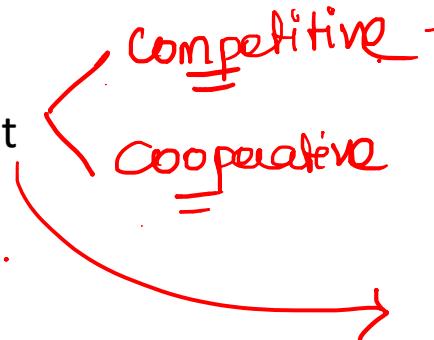
Agent Based:

> Cardinality : Single Vs MultiAgent

~~no of~~



eg.1



eg.2

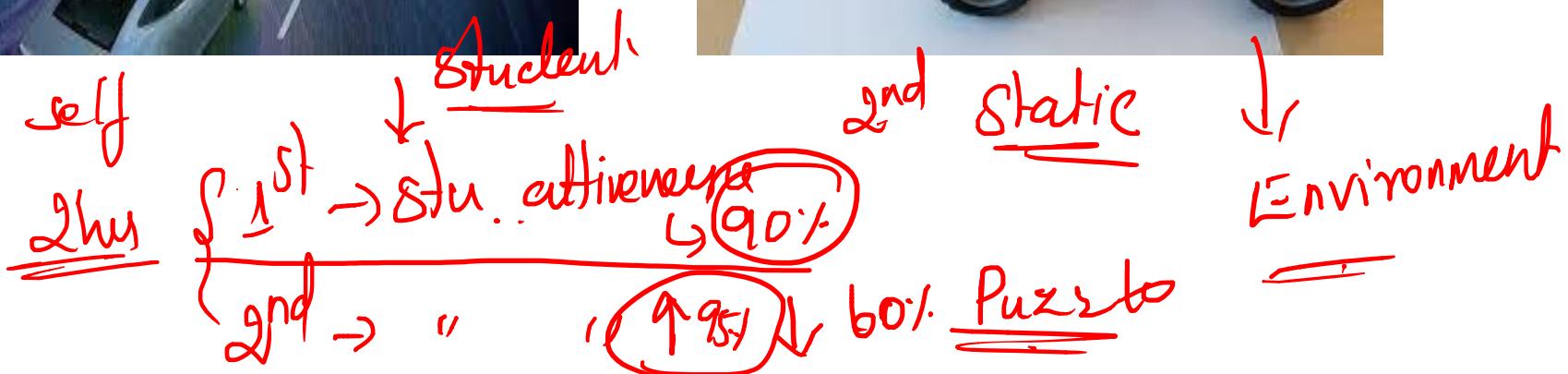
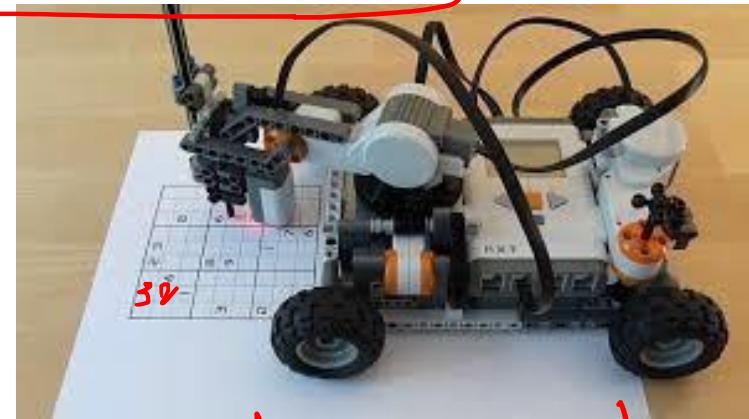
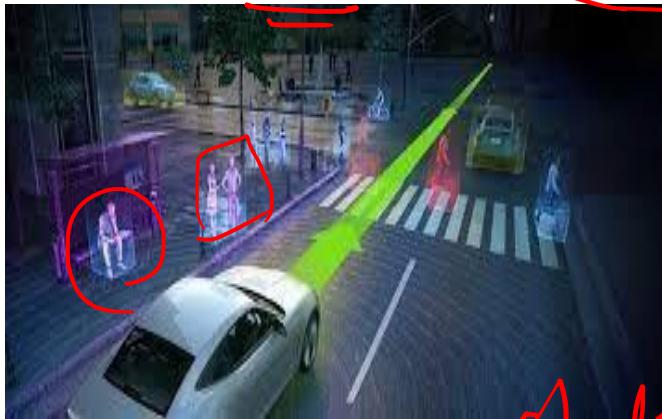
→ goal

# Task Environment

$$a_1 \rightarrow S_1$$

## Action & State Based:

- Change in Time : Static Vs Dynamic
- (The environment is semi dynamic if the environment itself does not change with the passage of time but the agent's performance score does)



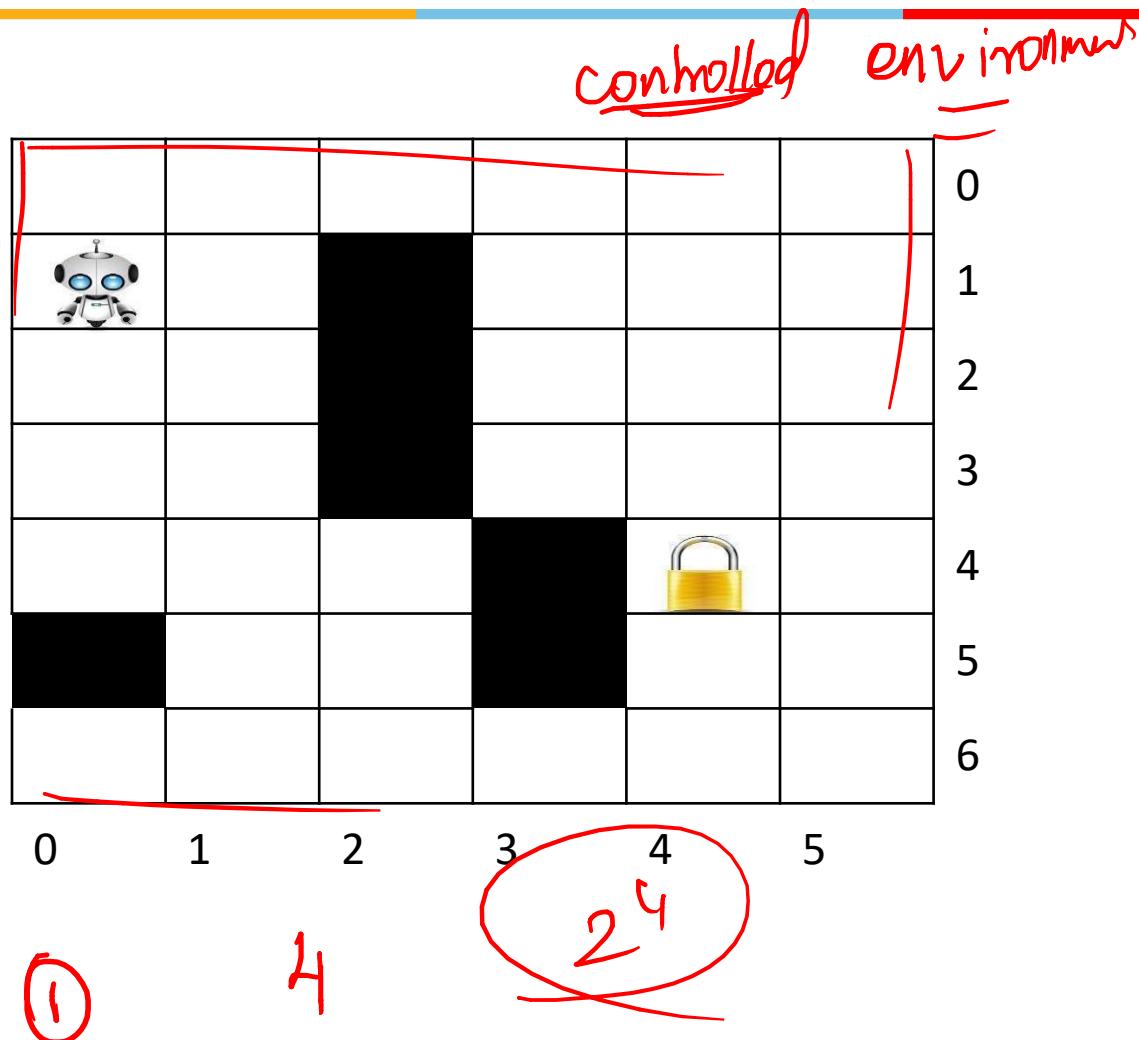
## Task Environment

$P_1, S_1$   
 $B \rightarrow T$  |  $P_2, S_2$   
 $D$

Task Environment	Fully vs Partially Observable	Single vs Multi-Agent	Deterministic vs Stochastic	Episodic vs Sequential	Static vs Dynamic	Discrete vs Continuous
Medical diagnosis system	✓ Partially	✓ Single	Stochastic	Sequential	Dynamic	Continuous
Satellite Image Analysis System	Fully	Single	Deterministic	Episodic	Static	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

MS  
 teams  
 CANVAS  
 stu p

## Path finding Robot - Lab Example

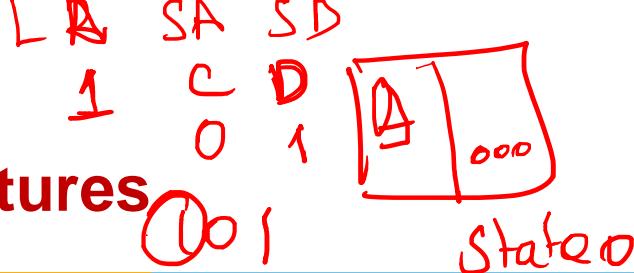


Agent	
Observability	fully
No.of.Actors	single
No.of.States	Disecrete
Determinism	Definite
Dynamicity	static
Output Dependency	seq

Agents  
Architectures  
= 

$\left. \begin{array}{l} S_1 \rightarrow \text{one} \\ S_2 \rightarrow A_1 + A_2 \end{array} \right\}$   
= 

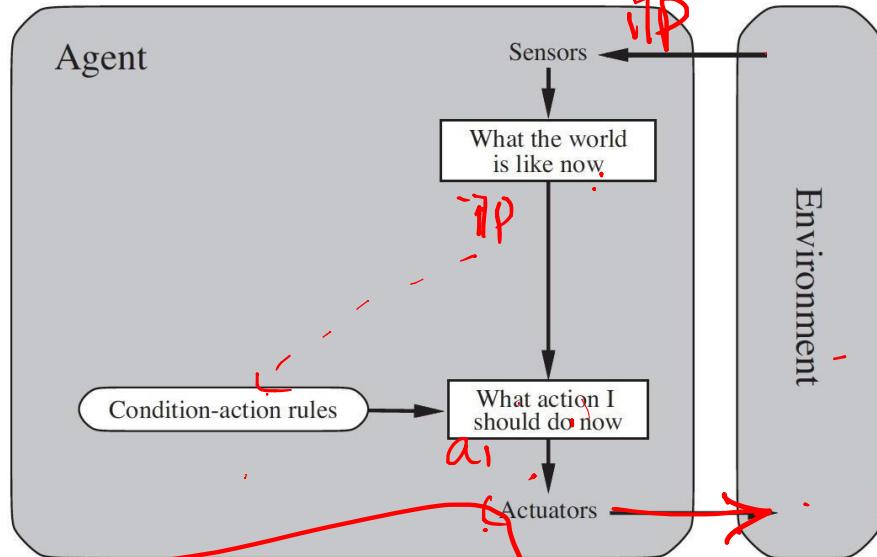
# Agent Architectures



## Reflex Agent

function SIMPLE-REFLEX-AGENT(*percept*) returns an action  
persistent: *rules*, a set of condition-action rules  
*state*  $\leftarrow$  INTERPRET-INPUT(*percept*)  
*rule*  $\leftarrow$  RULE-MATCH(*state*, *rules*)  
*action*  $\leftarrow$  *rule.ACTION* *MR*  
return *action*

function REFLEX-VACUUM-AGENT([*location*, *status*]) returns an action  
if *status* = Dirty then return Suck  
else if *location* = A then return Right  
else if *location* = B then return Left



Simple  
Reflex

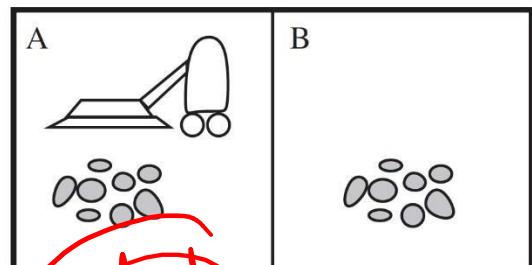
Rationally

=  
↳ Critical  $\rightarrow$  default action

Apply break

accident

→ fully observable

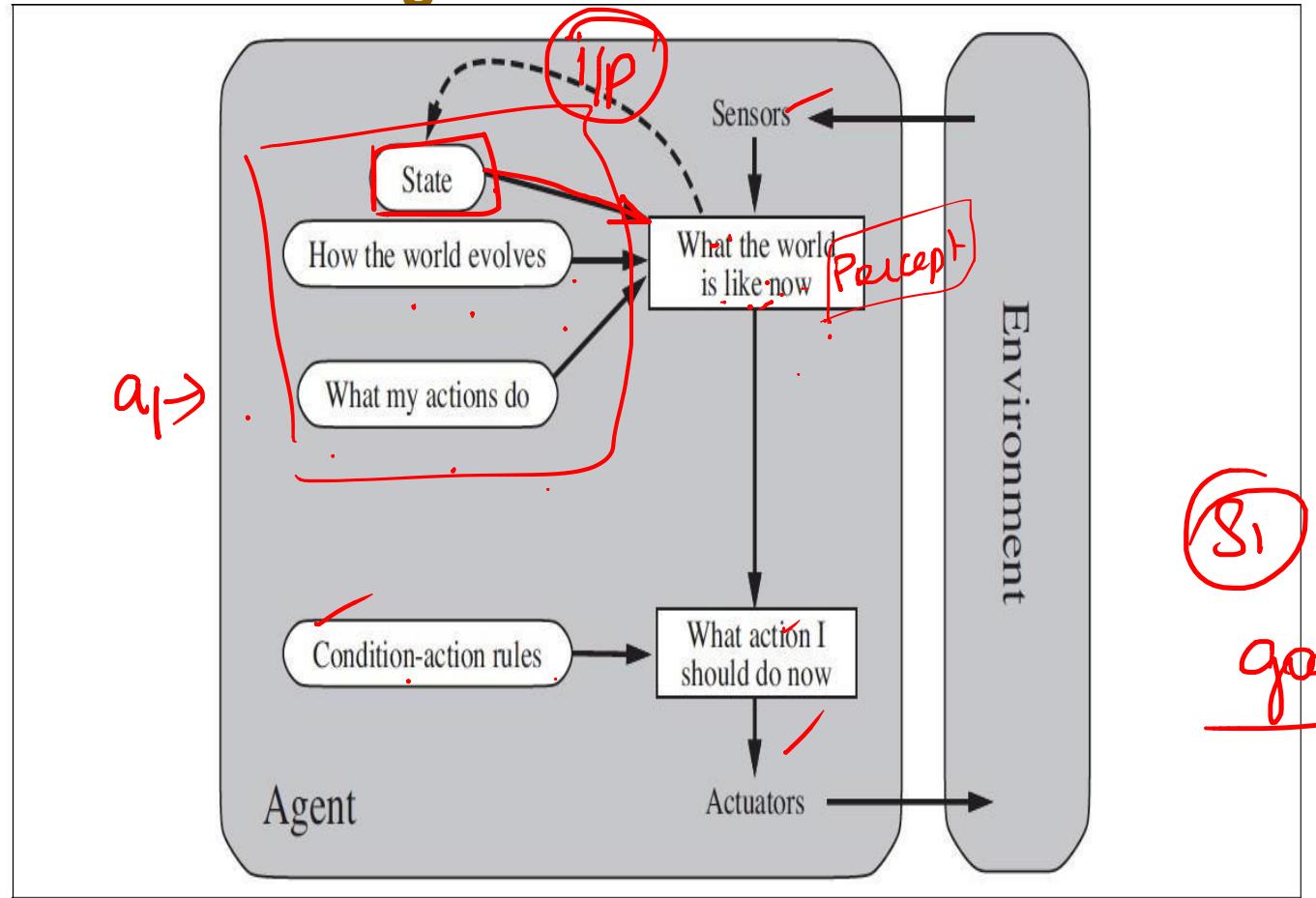


# Agent Architectures

## Model based Agent

Simple  
Reflex

✓  
Model Based  
Agents



# Agent Architectures

## Model based Agent

**function** MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action

**persistent:** *state*, the agent's current conception of the world state

*transition model*, a description of how the next state depends on the current state and action

*sensor model*, a description of how the current world state is reflected in the agent's percepts

*rules*, a set of condition-action rules

*action*, the most recent action, initially none

*state*  $\leftarrow$  UPDATE-STATE(*state*, *action*, *percept*, *transition model*, *sensor model* )

*rule*  $\leftarrow$  RULE-MATCH(*state*, *rules*)

*action*  $\leftarrow$  *rule.ACTION*

**return** *action*

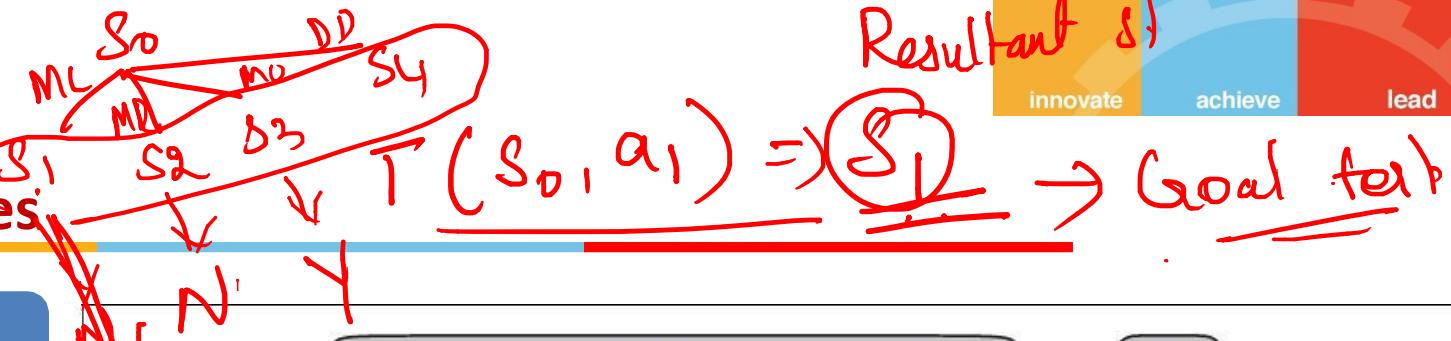
$a_1 \rightarrow \text{update}$

Non goal oriented approach

SD

probabilistic

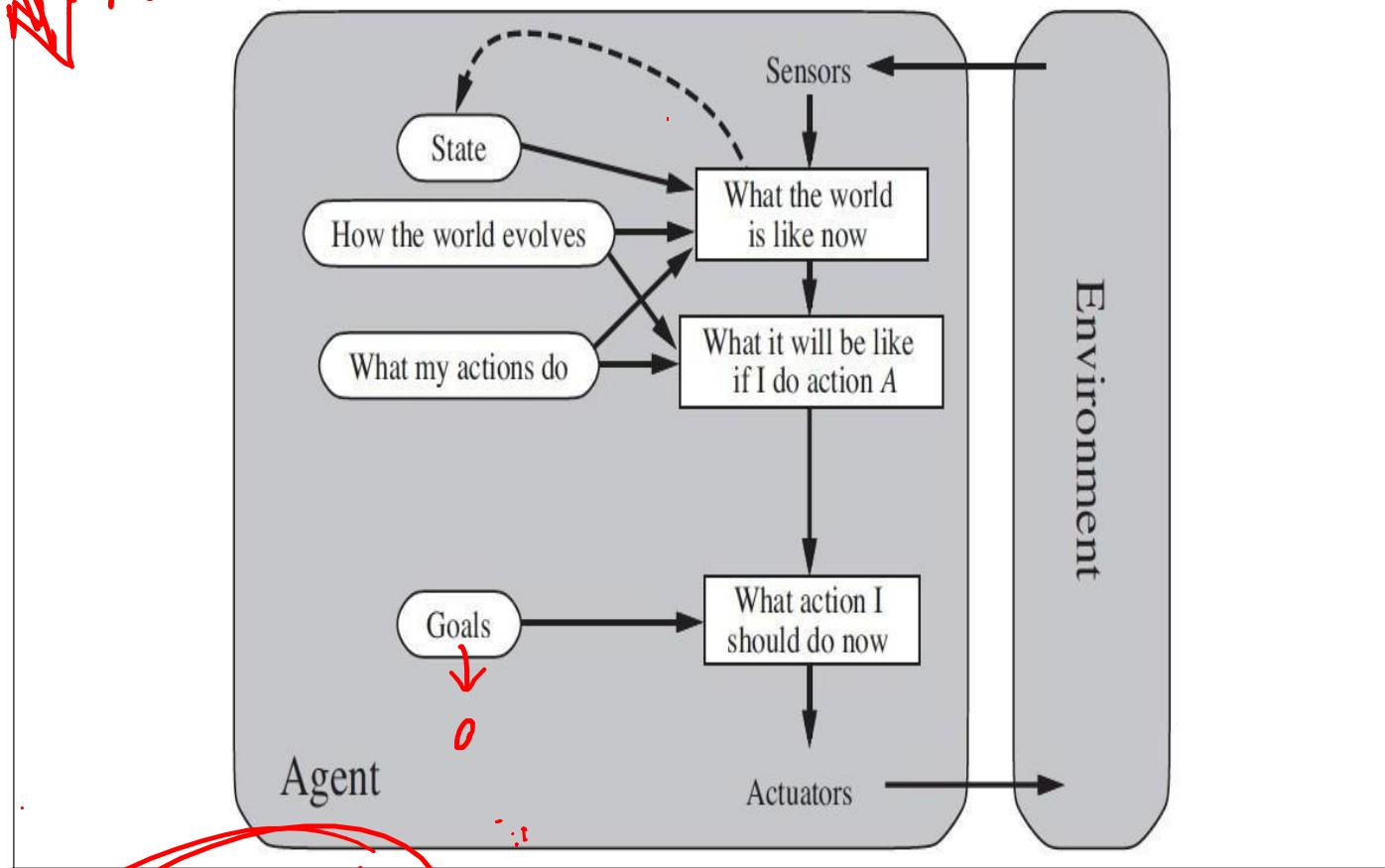
# Agent Architectures



Simple Reflex Agents

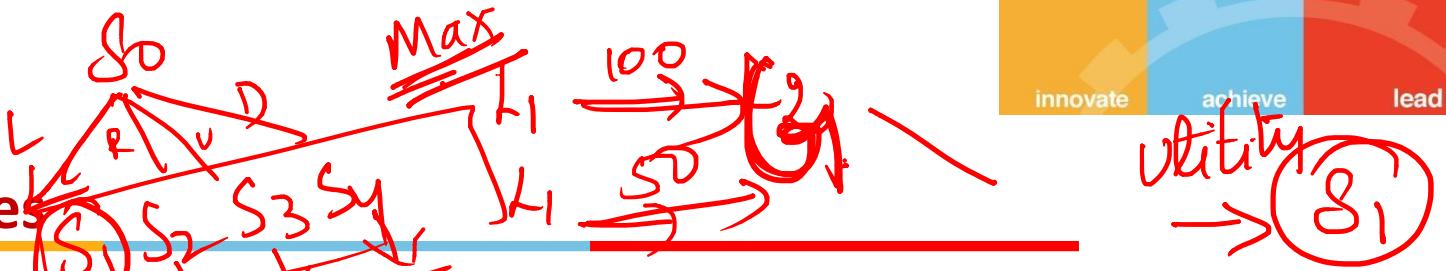
Model Based Agents

Goal Based Agents



$G + d$

# Agent Architectures

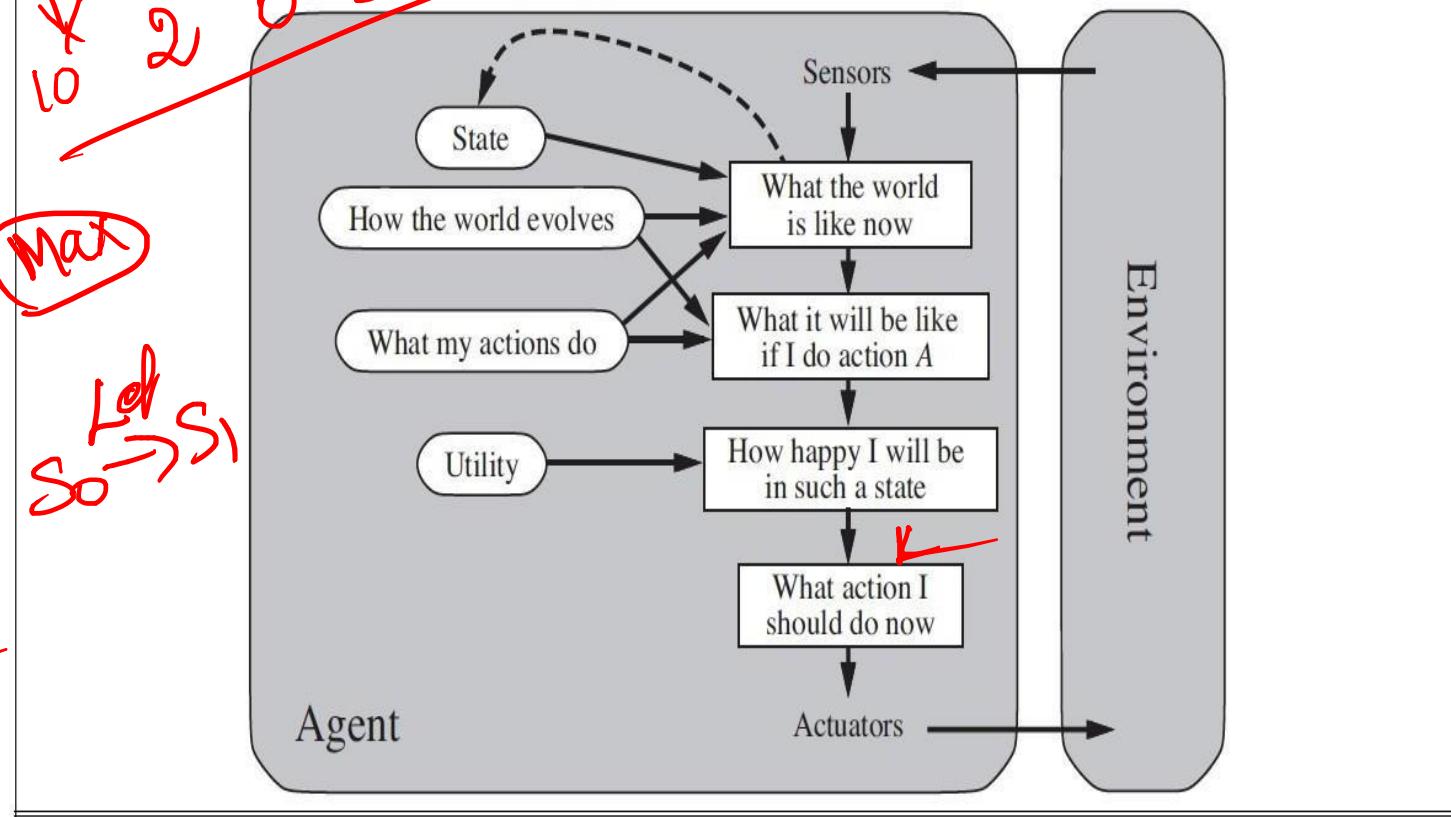


Simple Reflex Agents

Model Based Agents

Goal Based

Utility Based



# Agent Architectures

Simple Reflex Agents



Model Based Agents



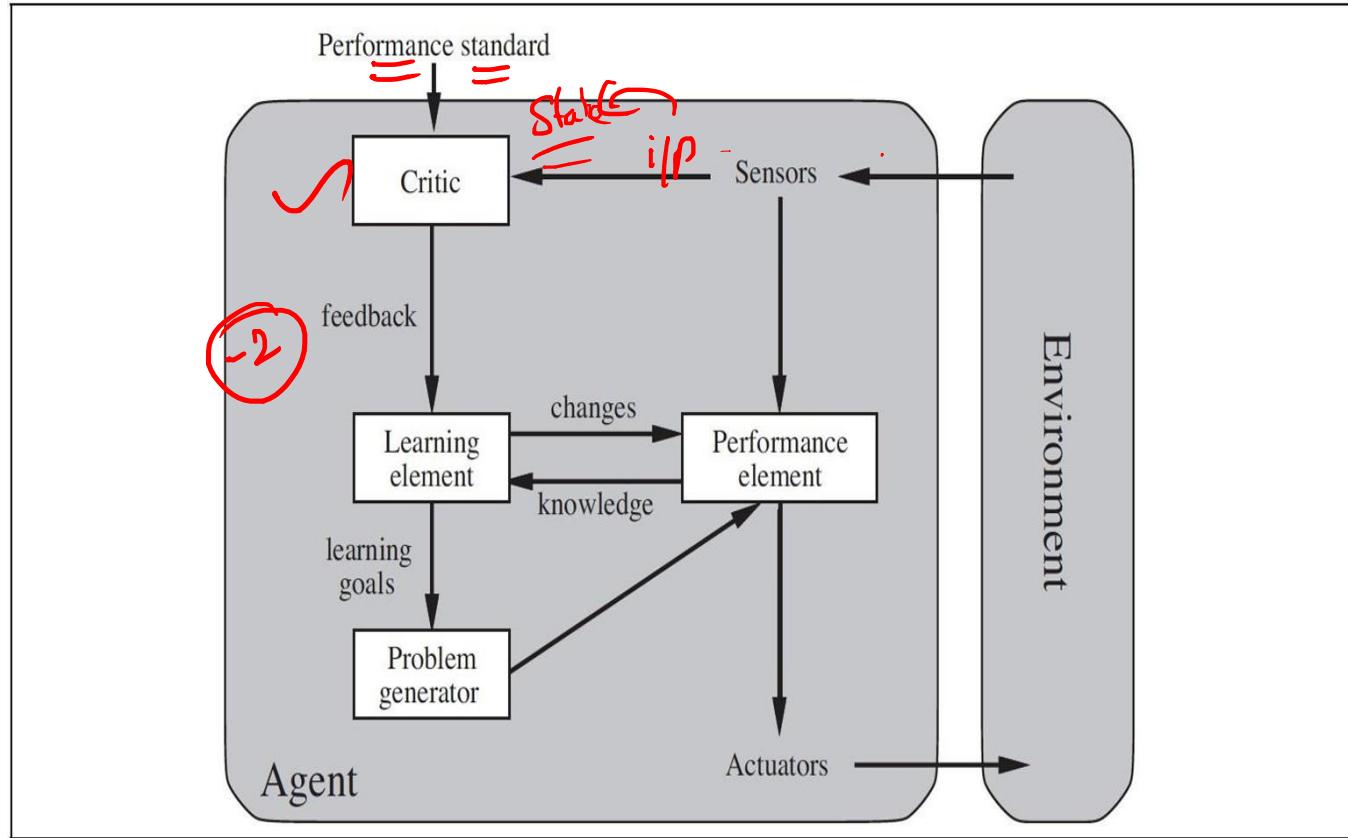
Goal Based Agents



Utility Based Agents

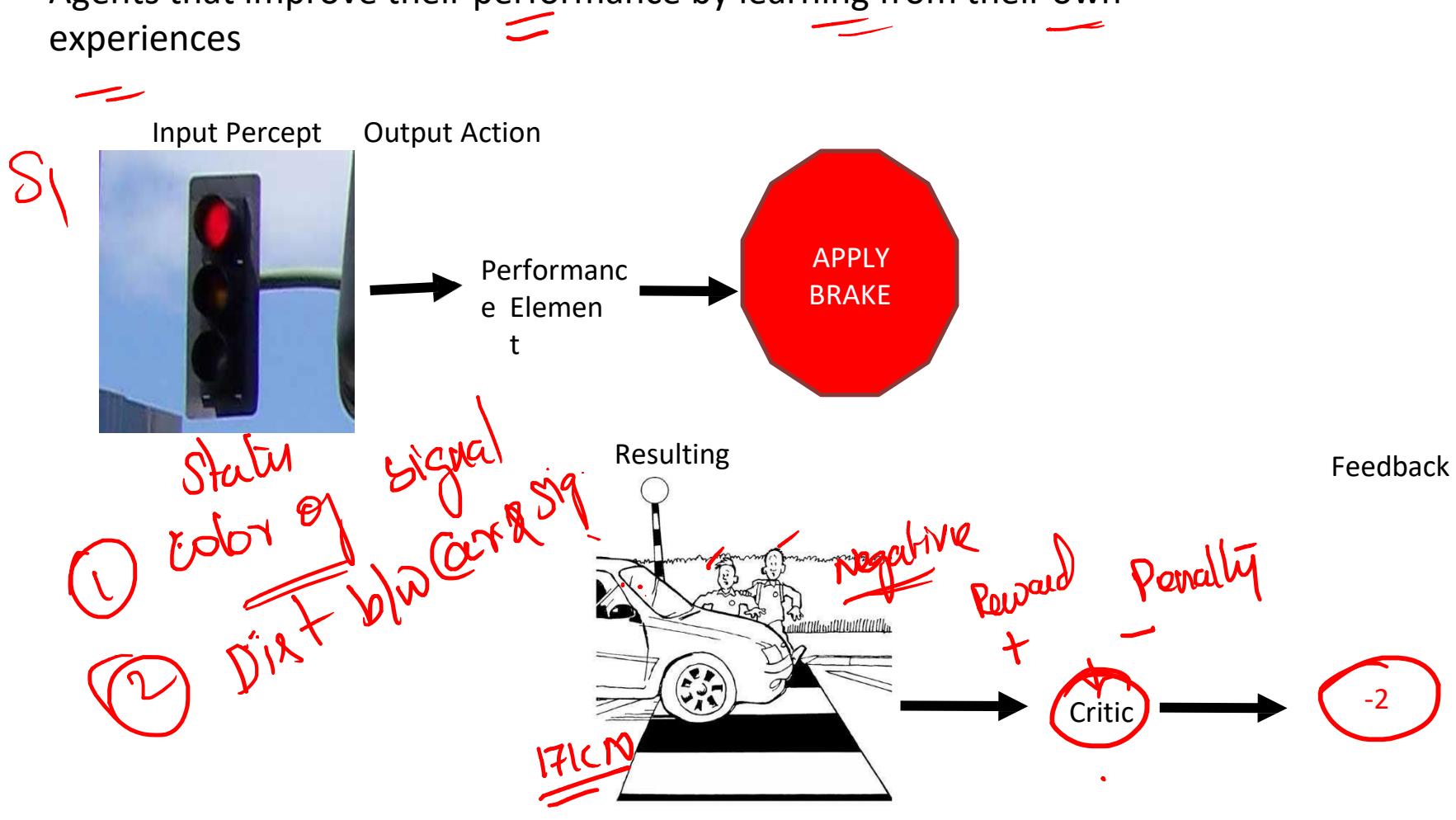


✓ Learning Agents

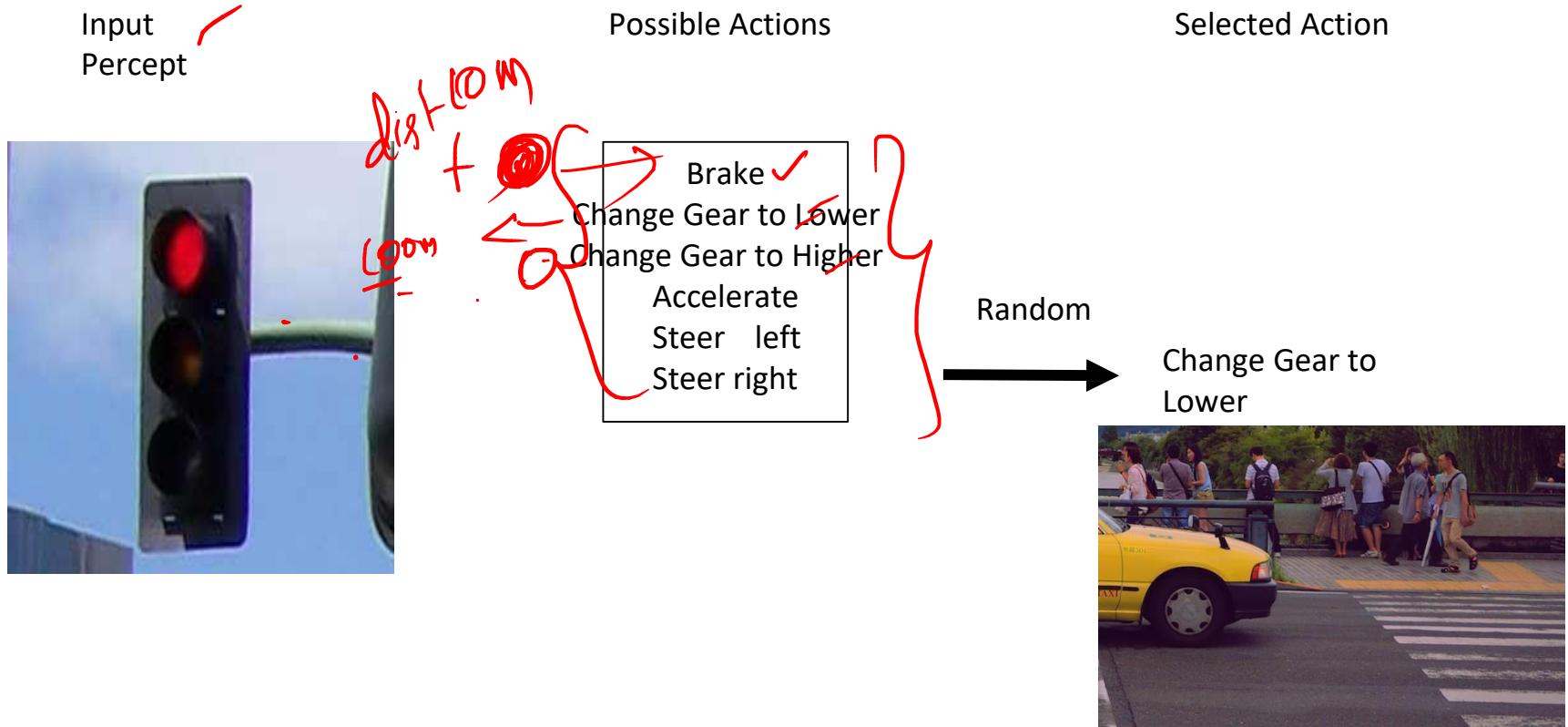


# Role of Learning

Agents that improve their performance by learning from their own experiences



# Role of Learning

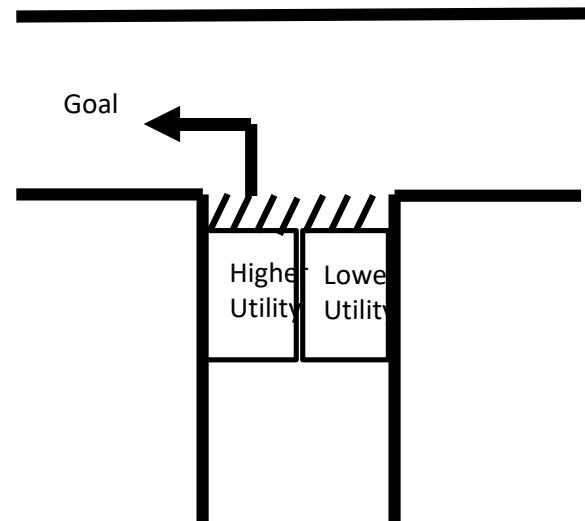


# Role of Learning

Performance Element – Takes decision on action based on percept

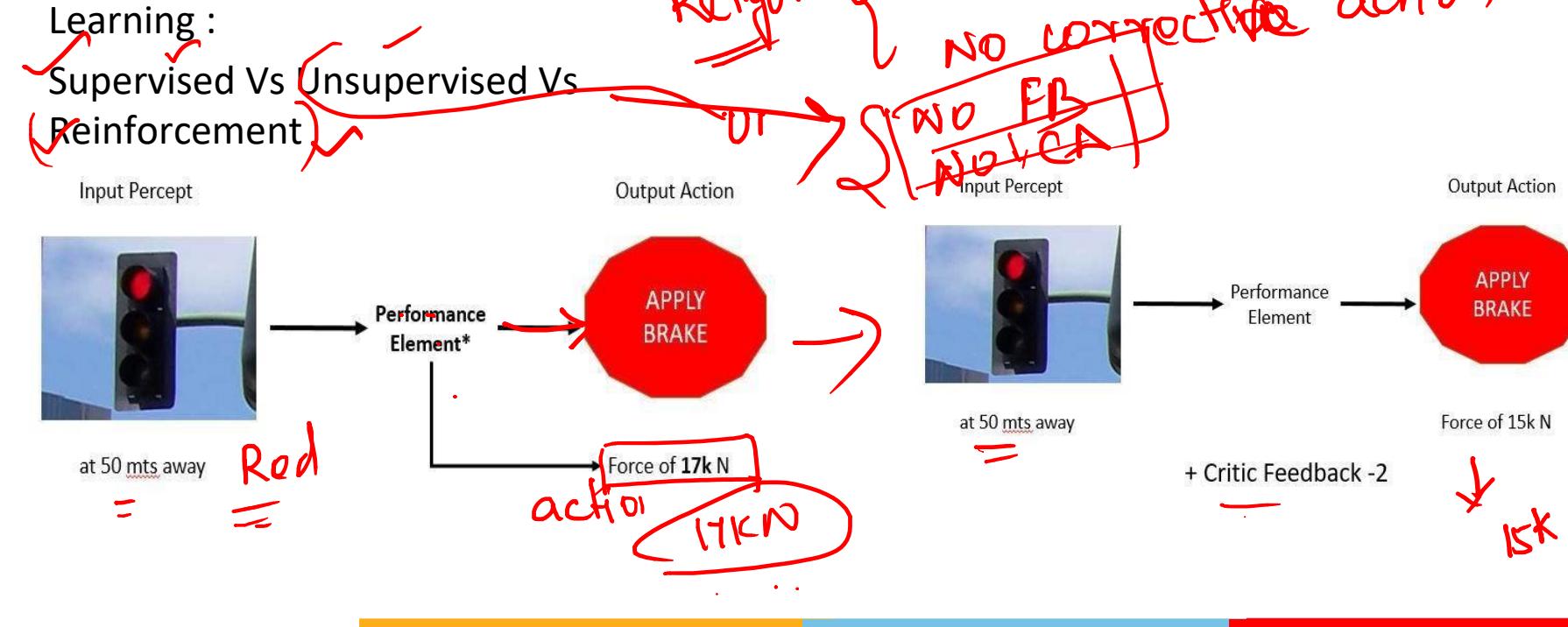
*f red signal, distance = 15k N brake*  
*distance = f'(percept sequence )*  
*f(percepts, distance, raining )*

- (  $f state_0, actionA$  = 0.83, )
  - $f state_0, actionB$  = 0.45



## Role of Learning

- Critic – Provides feedback on the actions taken

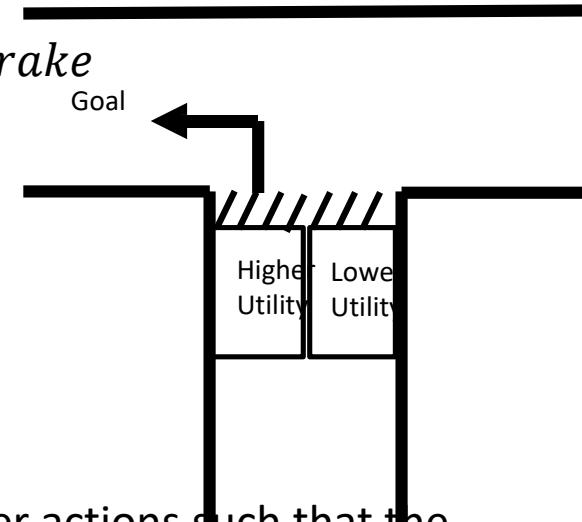


# Role of Learning

Performance Element – Takes decision on action based on percept

$f^{red\ signal, distance = 15k\ N\ brake}$   
 $distance = (f' percept\ sequence)$   
 $(f\ perceps, distance, raining)$

- (  $f state_0, actionA = 0.83,$
- (  $f state_0, actionB = 0.45$



Learning Element – Make the performance element select better actions such that the utility function is optimized

Critic – Provides feedback on the actions taken

Problem Generator – Make the Performance Element select sub-optimal actions such that you would learn from unseen actions

**Required Reading:** AIMA - Chapter #1, 2, 3.1, 3.2, 3.3

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

DSE CLZG557

M1 : Introduction  
&

M2 : Problem Solving Agent using Search

Indumathi V  
Guest Faculty,  
BITS - WILP

**BITS** Pilani

Pilani Campus



# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

# Learning Objective

---

At the end of this class , students Should be able to:

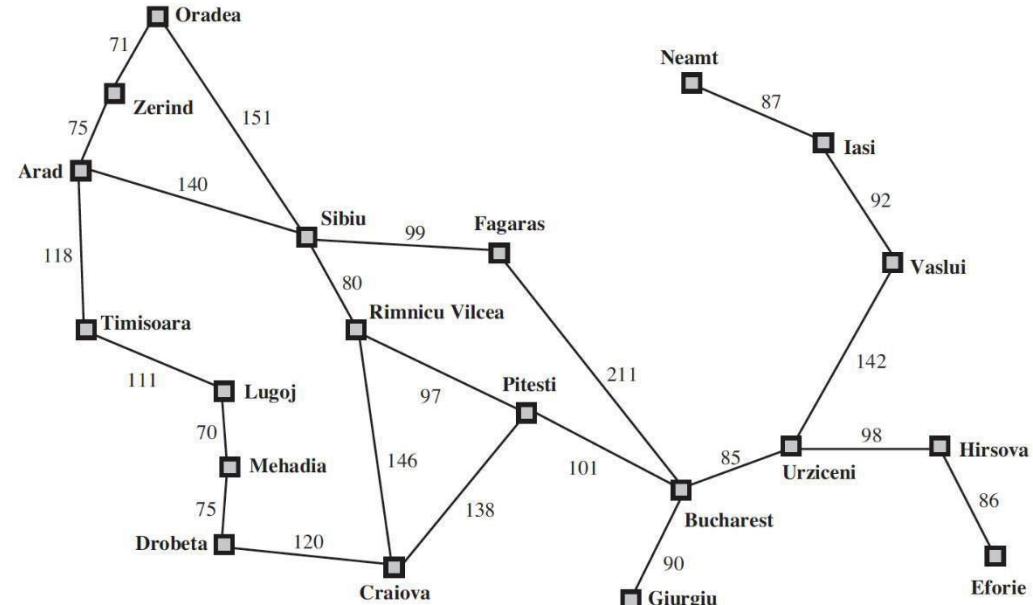
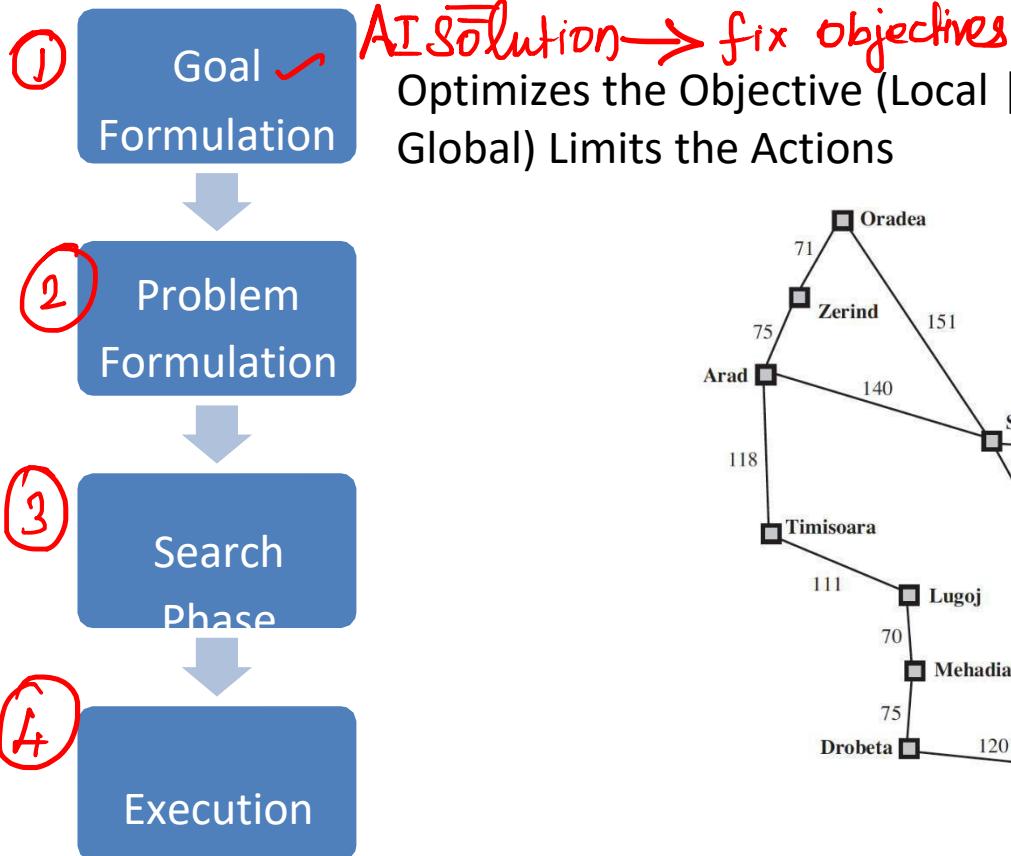
1. Design problem solving agents
2. Create search tree for given problem
3. Apply uninformed search algorithms to the given problem
4. Compare performance of given algorithms in terms of completeness, optimality, time and space complexity
5. Differentiate for which scenario appropriate uninformed search technique is suitable and justify

# Problem Formulation

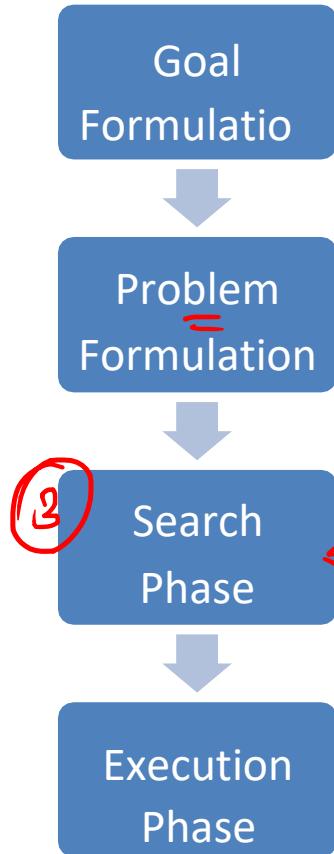
# Problem Solving Agents

Goal based decision making agents finds sequence of actions that leads to the desirable state.

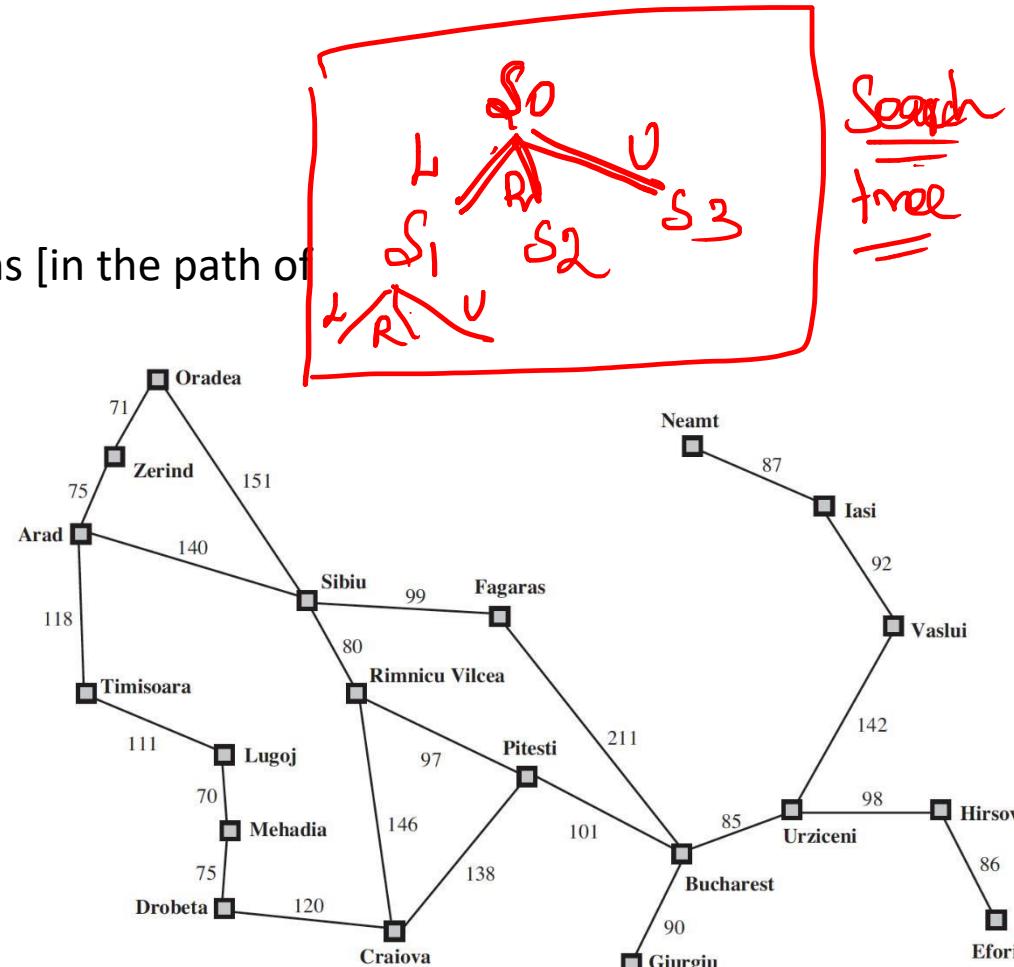
## Phases of Solution Search by PSA



# Problem Solving Agents

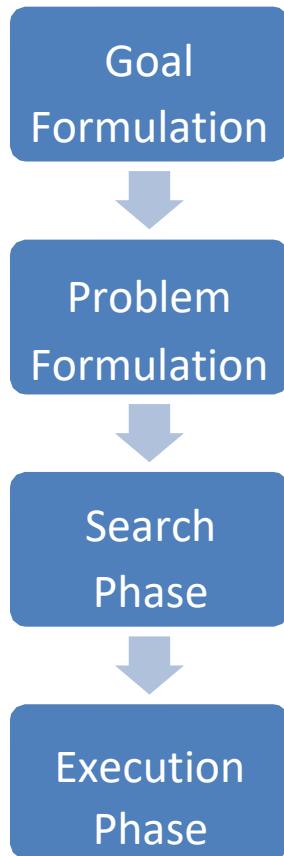


## Phases of Solution Search by PSA



# Problem Solving Agents

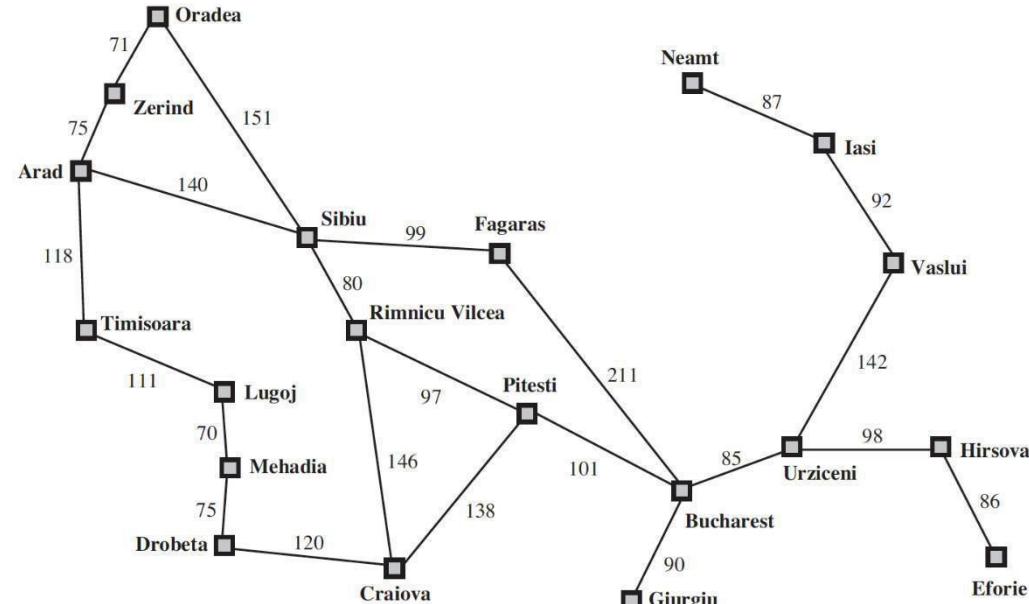
Module 2



## Phases of Solution Search by PSA

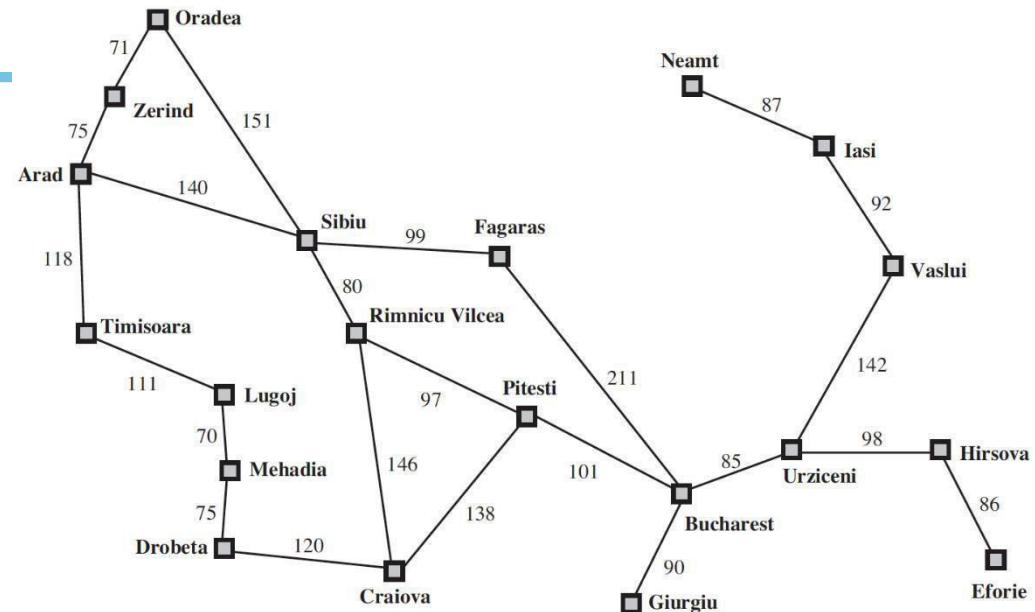
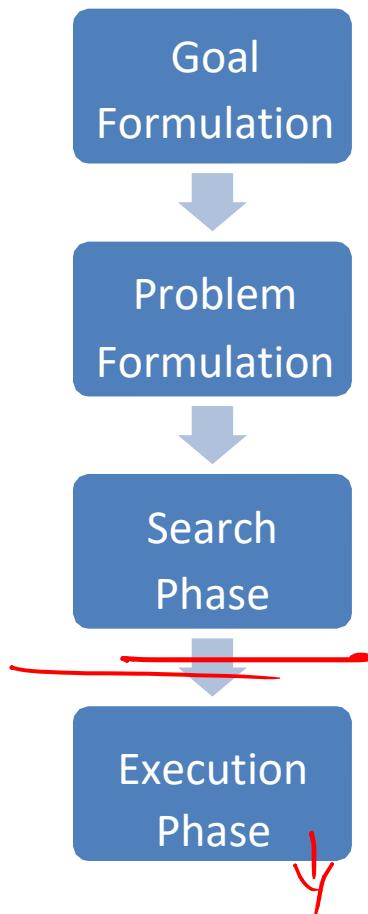
Assumptions – Environment :

- ① Static
- ② Observable Discrete
- ③ Deterministic



# Problem Solving Agents

## Phases of Solution Search



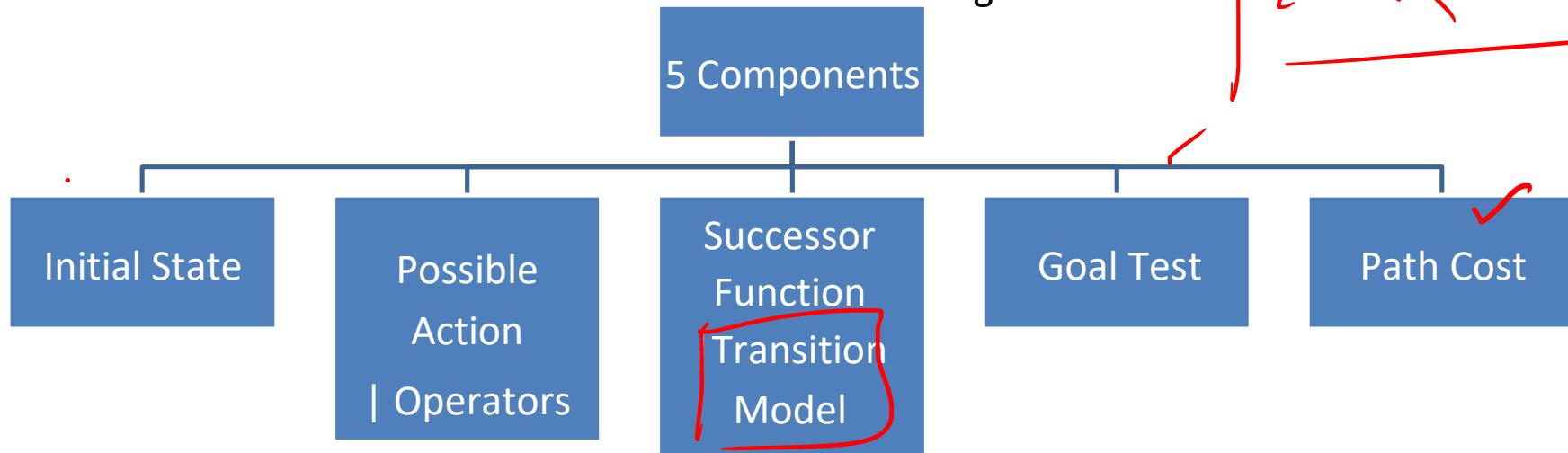
Examine all sequence  
Choose best |  
Optimal

# Problem Solving Agents – Problem Formulation

8ta'

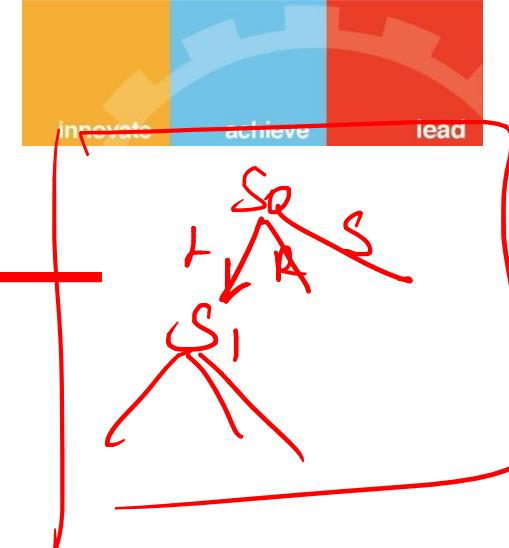
Abstraction Representation

Decide what actions under states to take to achieve a goal



A function that assigns a numeric cost to each path. A path is a series of actions. Each action is given a cost depending on the problem.

**Solution = Path Cost Function + Optimal Solution**



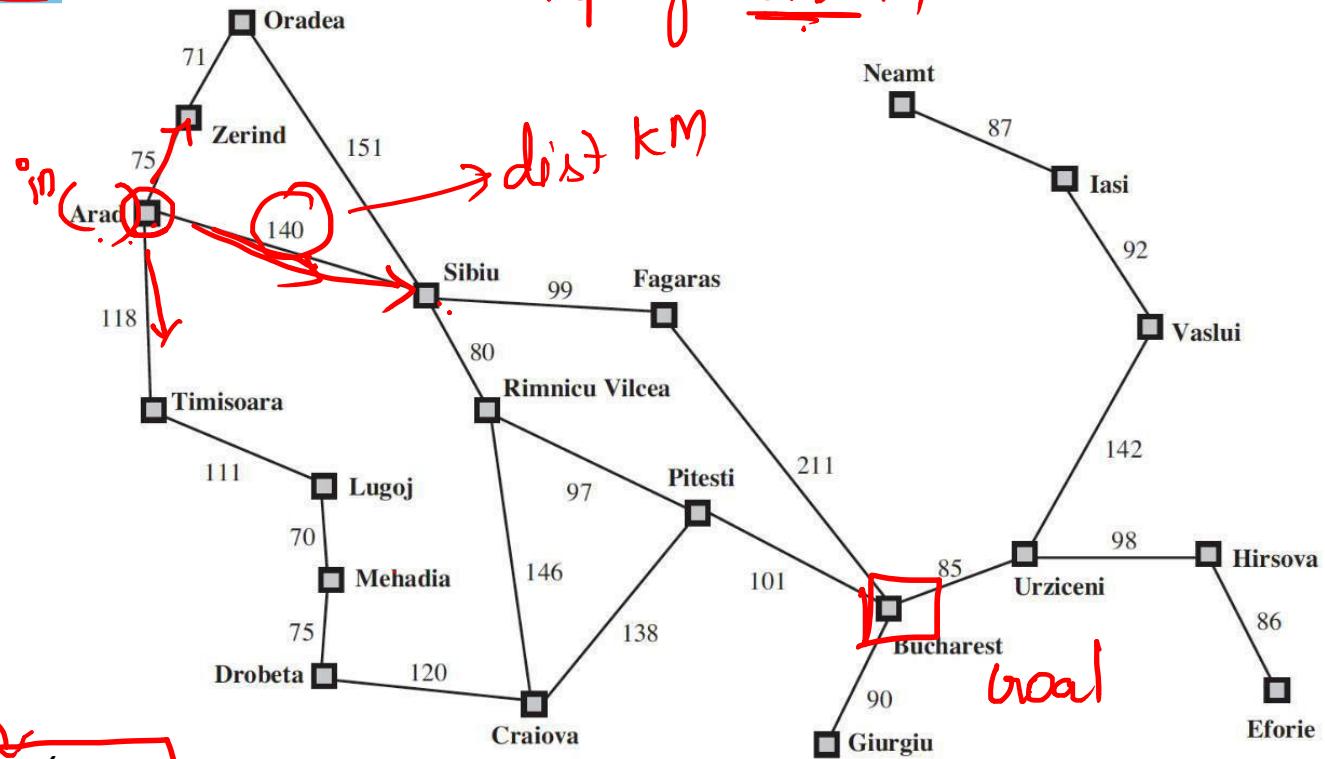
# Problem Solving Agents – Problem

## Formulation: BookExample

Map of Romania

(1,0)

HP ac Res u  
T ( $s_0, a_1 \Rightarrow s_1$ )



function

Initial State – E.g.,  $In(Arad)$

Possible Actions – ACTIONS(s)  $\Rightarrow \{Go(Sibiu), Go(Timisoara), Go(Zerind)\}$

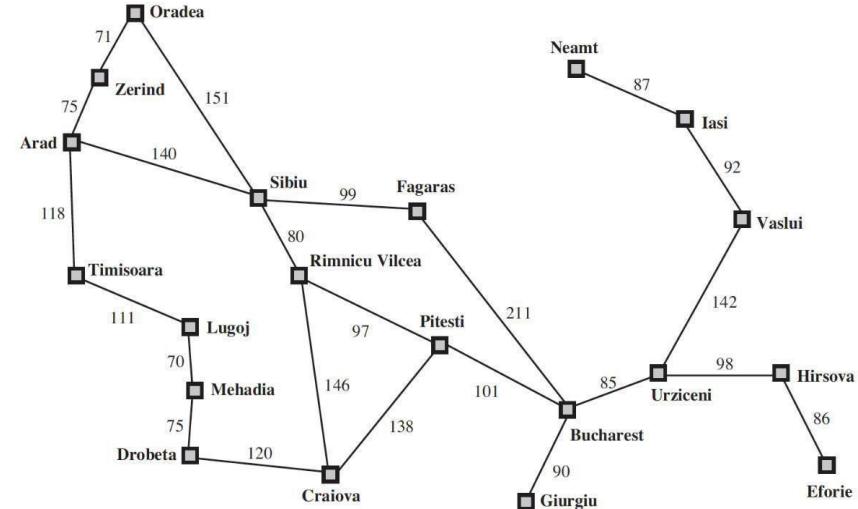
Transition Model – RESULT(  $In(Arad), Go(Sibiu)$  ) =  $In(Sibiu)$

Goal Test – IsGoal(  $In(Bucharest)$  ) = Yes

Path Cost – cost(  $In(Arad), go(Sibiu)) = 140 \text{ kms}$  ✓ + PMS + PMS  $\Rightarrow$  single number val.

# Example Problem Formulation

	Travelling Problem
Initial State	Based on the problem
Possible Actions	Take a flight   Train   Shop
Transition Model/ Successor Function	$[A, \text{Go}(A \rightarrow S)] = [S]$
Goal Test	Is current = B (destination)
Path Cost	<u>Cost</u> + <u>Time</u> + <u>Quality</u>



## Example Problem Formulation

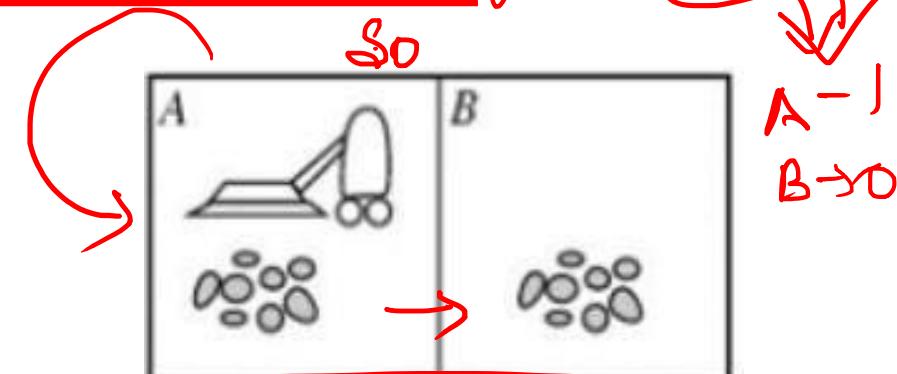
$s_A \rightarrow$  status Room A  
 $s_B \rightarrow$  " " Room B  
 $l_R \rightarrow$  Location of Robot

in private

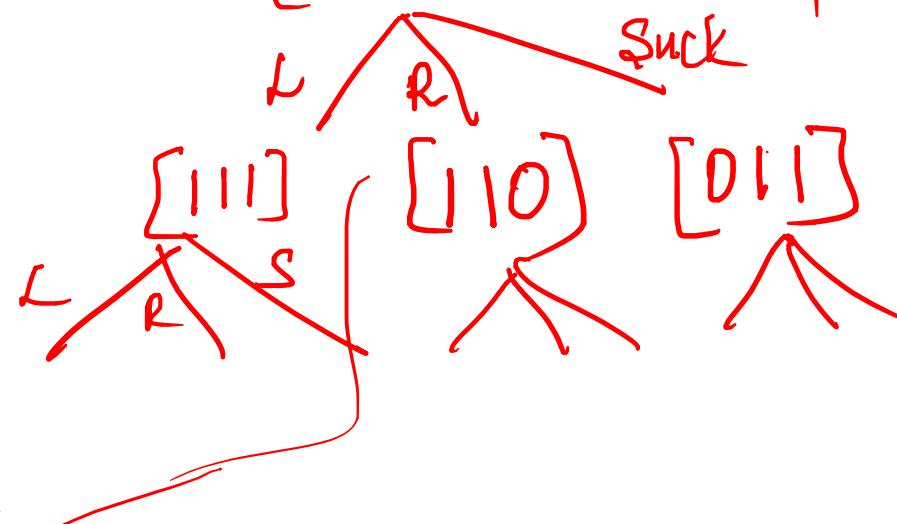
achieve

lead

Vacuum World	
Initial State	Any $[1, 1, 1]$
Possible Actions	[Move Left, Move Right, Suck, No Ops]
Transition Model/ Successor Function	$[A, ML] = [B, Dirty]$ $[A, ML] = [B, Clean]$
Goal Test	Is all room clean? $[A, Clean]$ $[B, Clean]$
Path Cost	No of steps in path

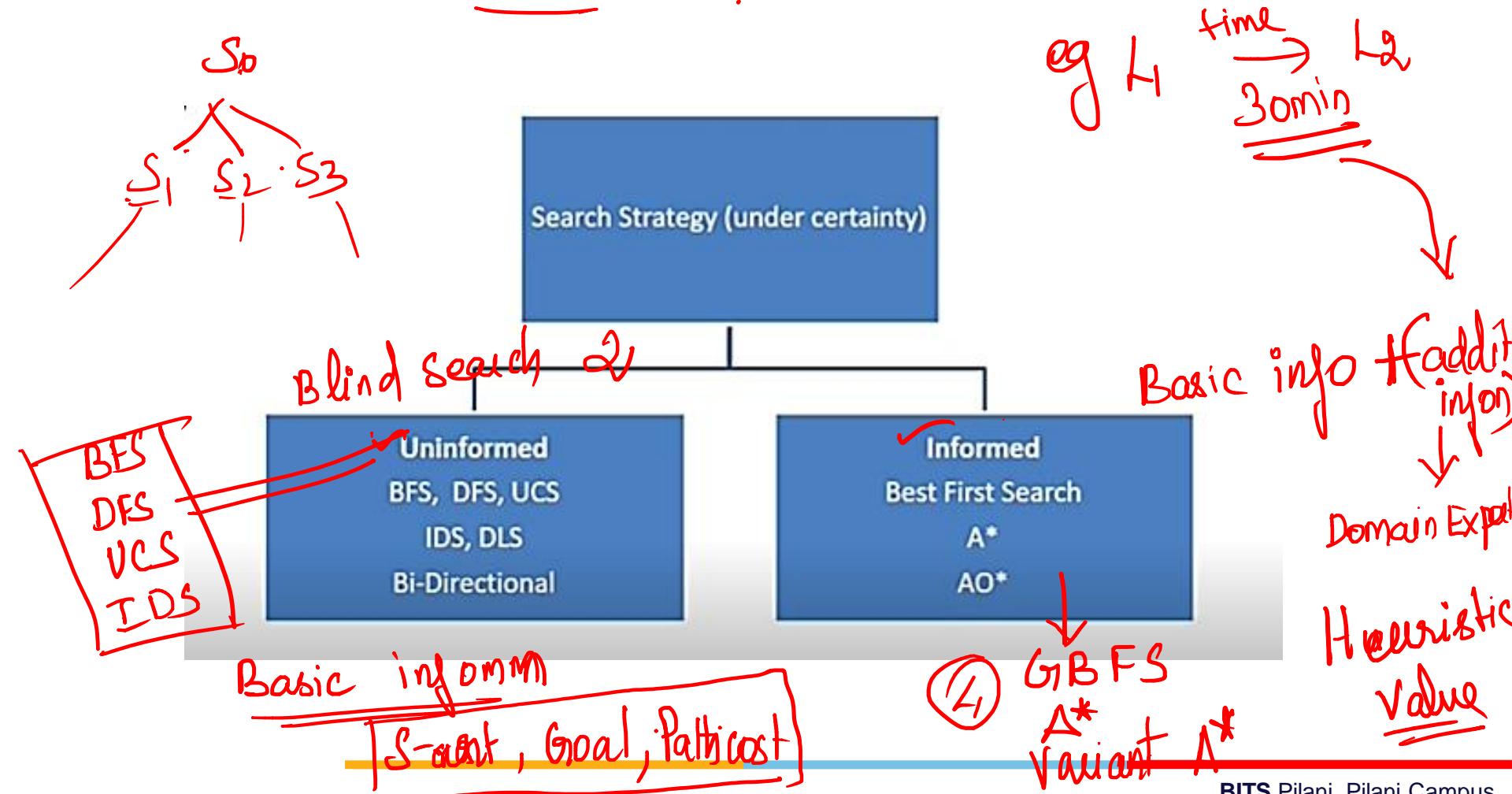


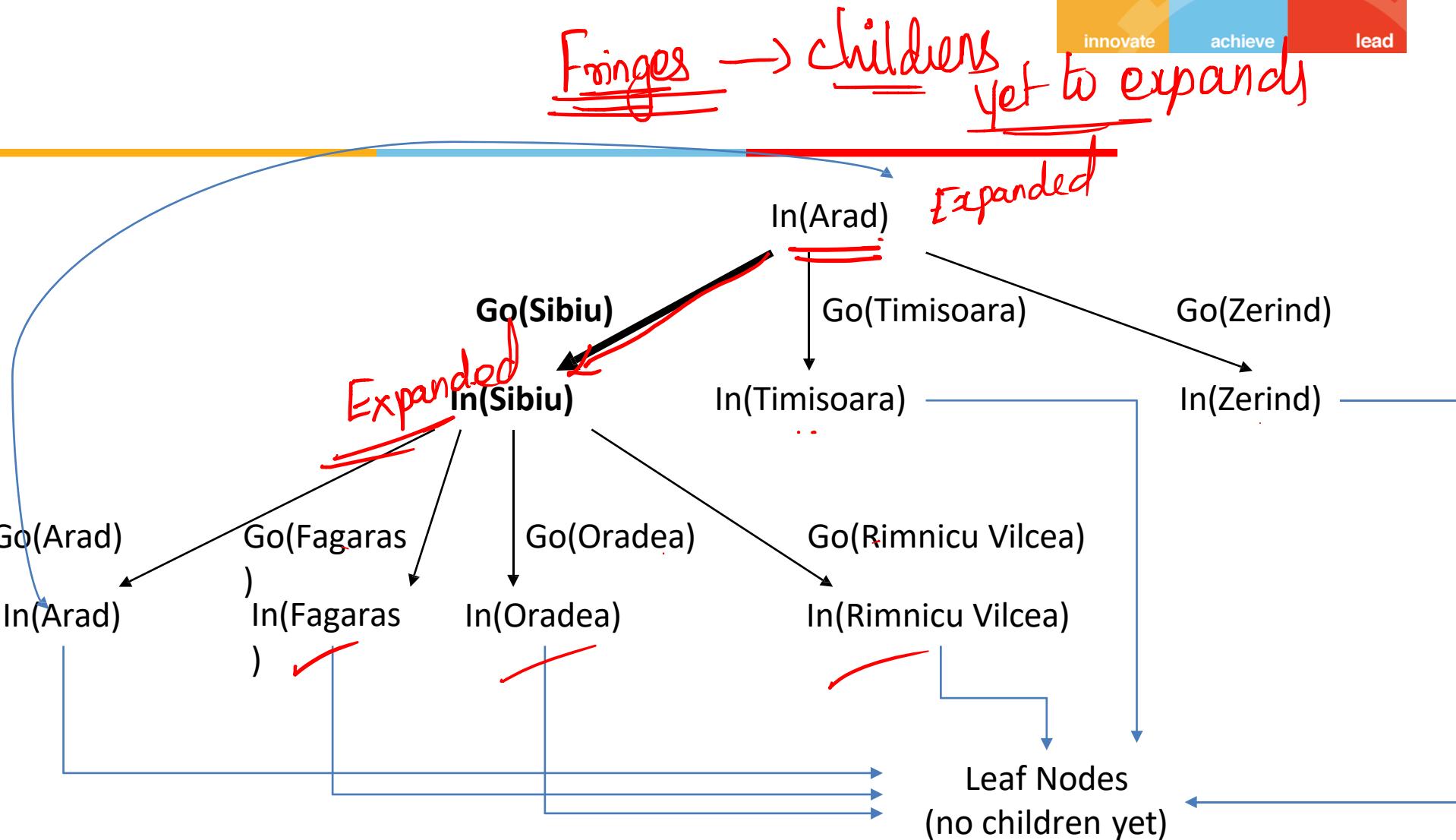
3 val  $s_A$   $s_B$   $l_R$   $s_0$



# Searching for Solutions

Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy





## Breadth First Search

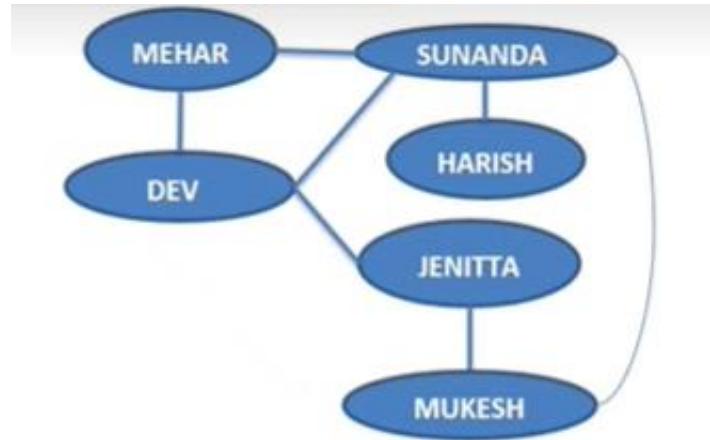
- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph

## Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

## Breadth First Search

- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph



## Depth First Search

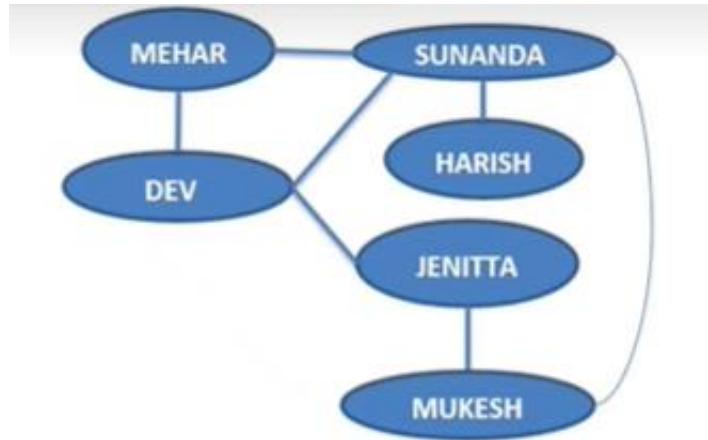
- Find the Connectedness in a graph
- Topological Sorting

# Application



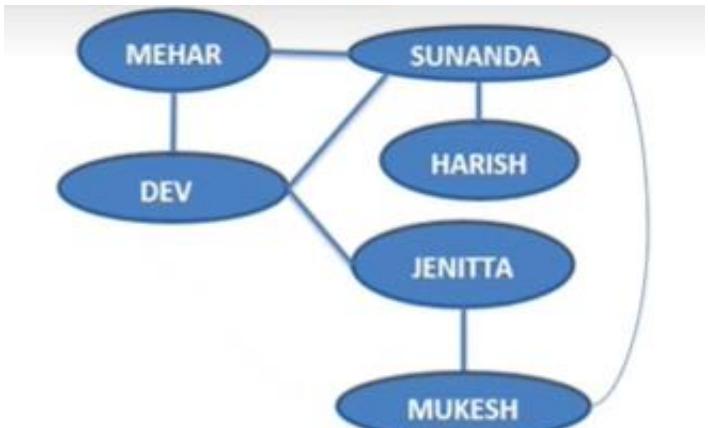
## Breadth First Search

- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph



## Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

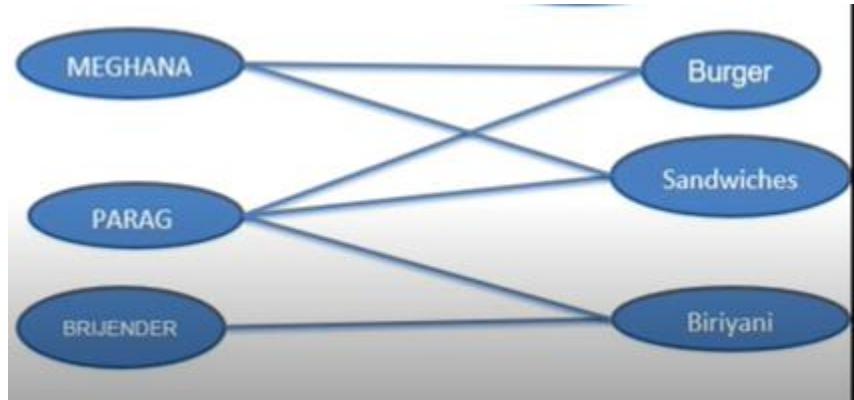


# Application



## Breadth First Search

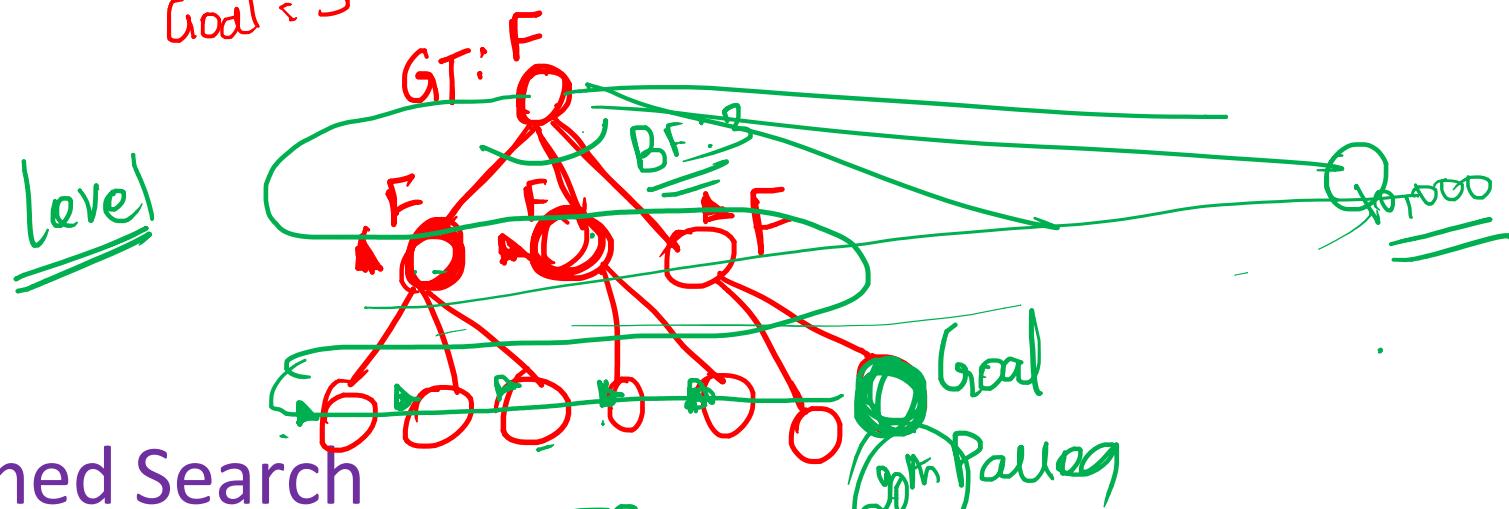
- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph



## Depth First Search

- Find the Connectedness in a graph
- Topological Sorting

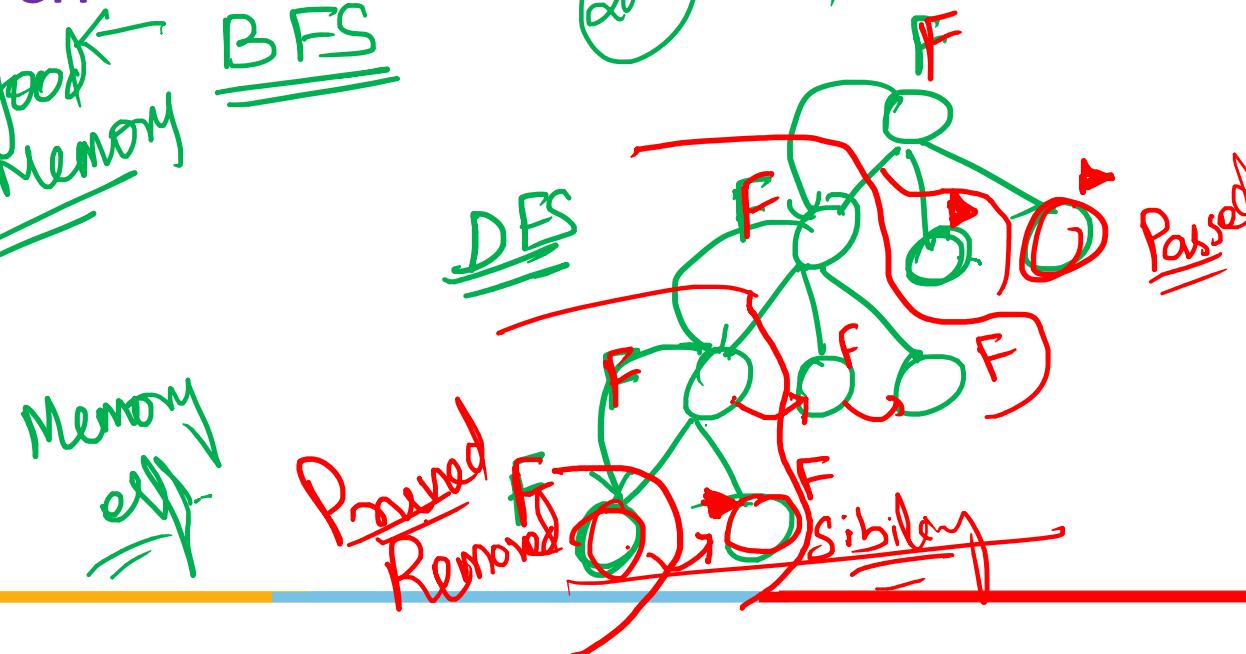
$$\text{Star: } \underline{1} \quad \underline{1=5?}$$



# Uninformed Search Overview

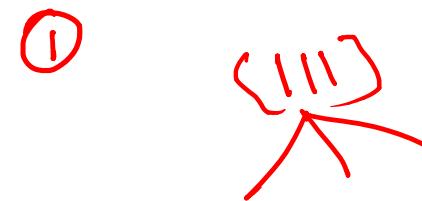
BFS

Not good  
in memory

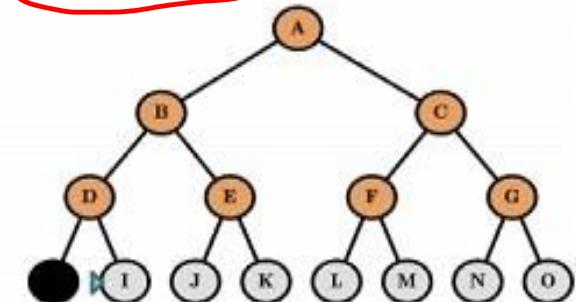
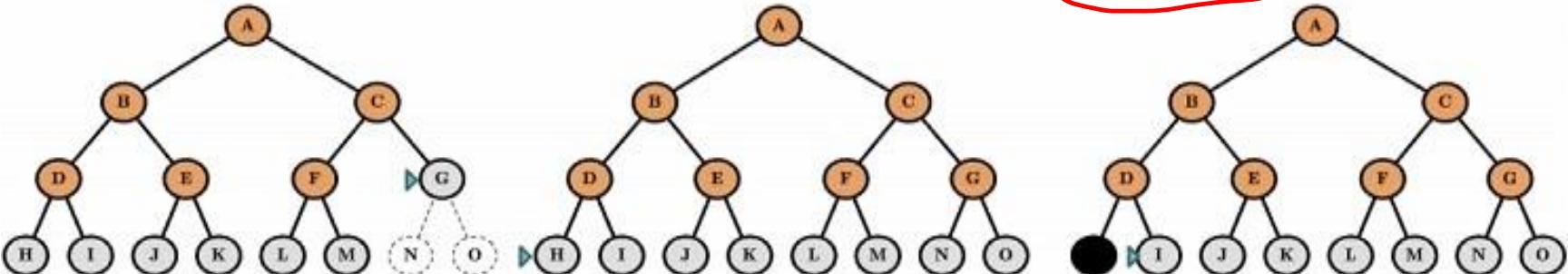
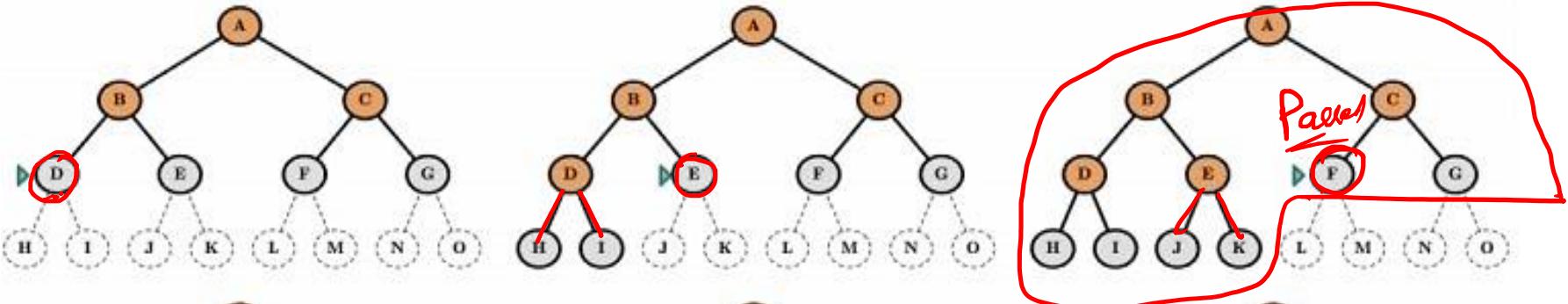
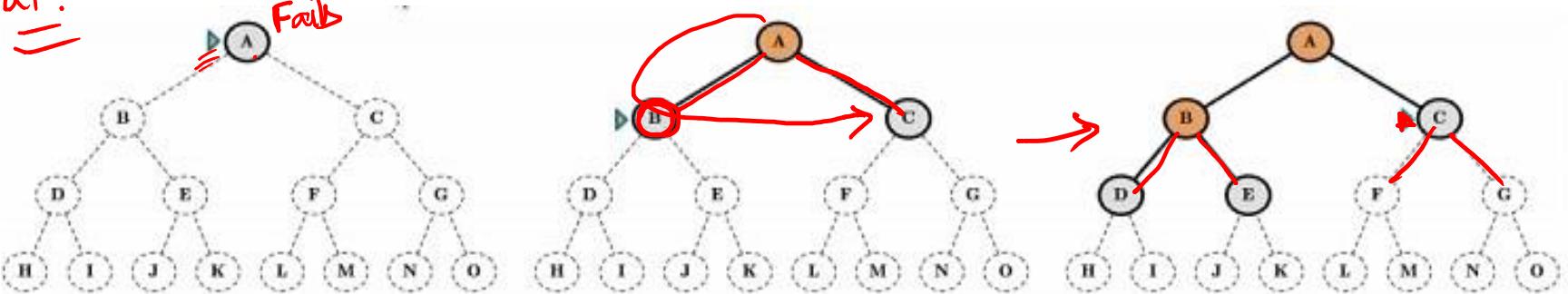


# Breadth First Search (BFS)

Start : A = :



Goal : F

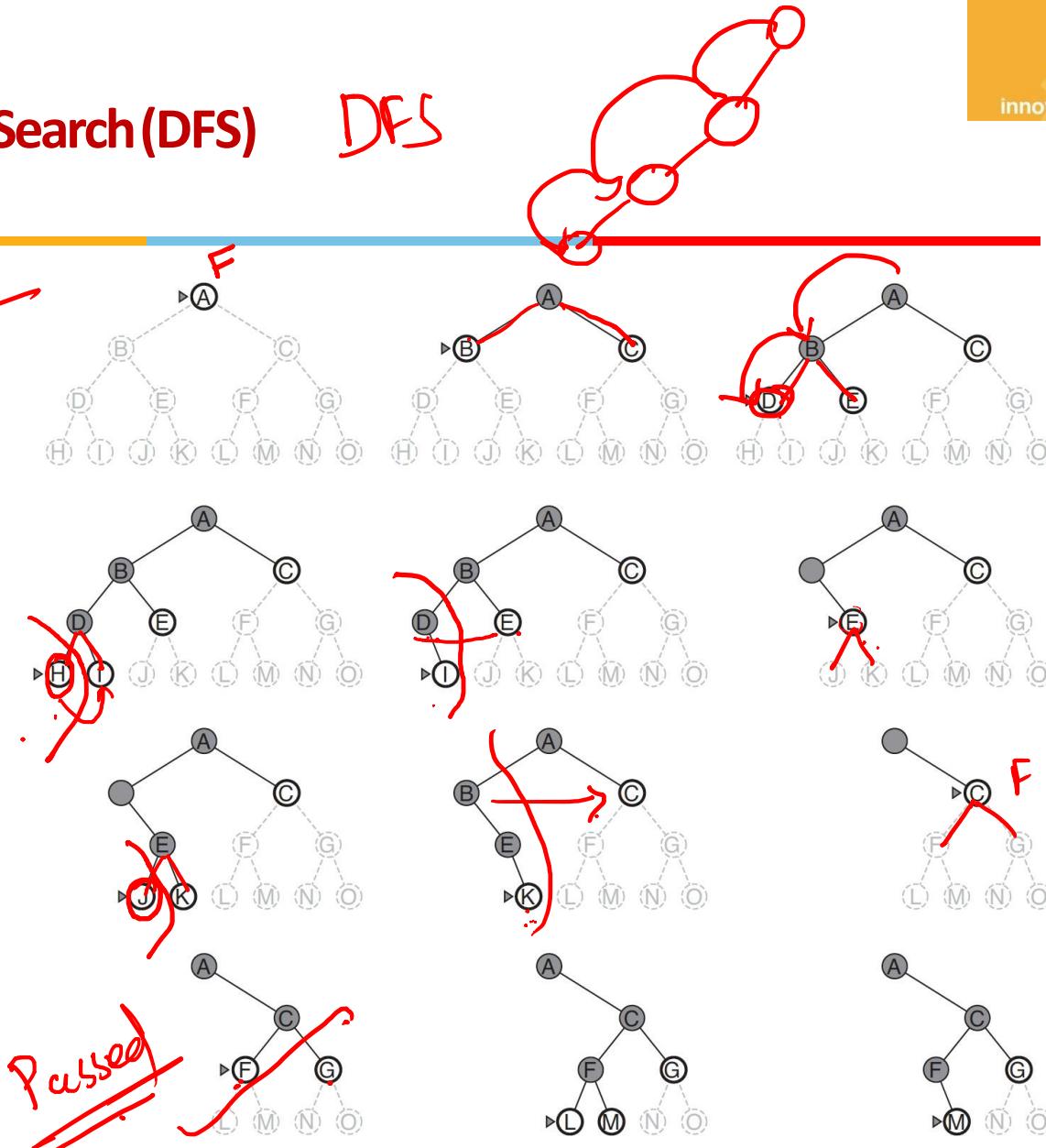


## Depth First Search (DFS)

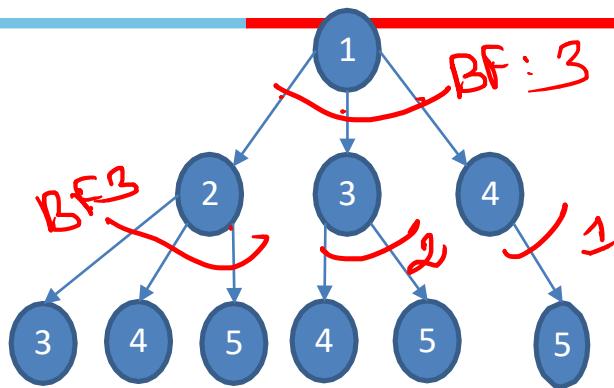
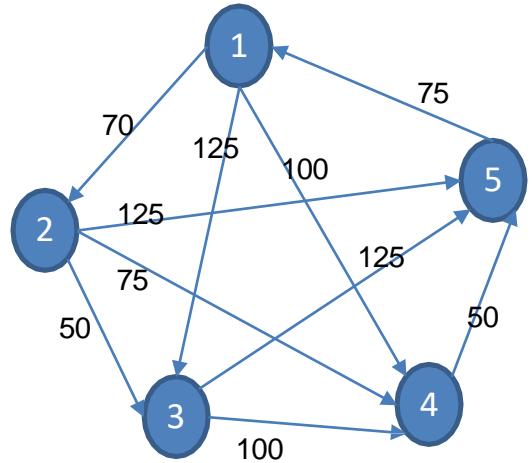
~~Start : A~~

~~Goal : F~~

DFS



# Search Tree – Sample Generation



**Each NODE in the search tree denotes an entire PATH through the state space graph.**

# Search Algorithm – Uninformed Example - 1

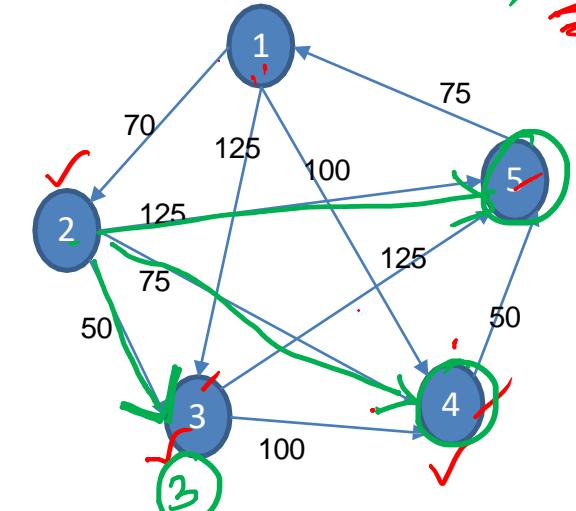


Start : 1  
Goal : 5

Stack →  
LIFO

DFS

Random



(1) → ①  
(1, 2) (1, 3) (1, 4) → ③  
(1, 2, 3) (1, 2, 4) (1, 2, 5) (1, 3) (1, 4) → ⑤  
(1, 2, 3, 4) (1, 2, 3, 5) (1, 2, 4) (1, 2, 5) (1, 3) (1, 4)  
**(1, 2, 3, 4, 5)** (1, 2, 3, 5) (1, 2, 4) (1, 2, 5) (1, 3) (1, 4)

$$C(1-2-3-4-5) = 70 + 50 + 100 + 50 = 270$$

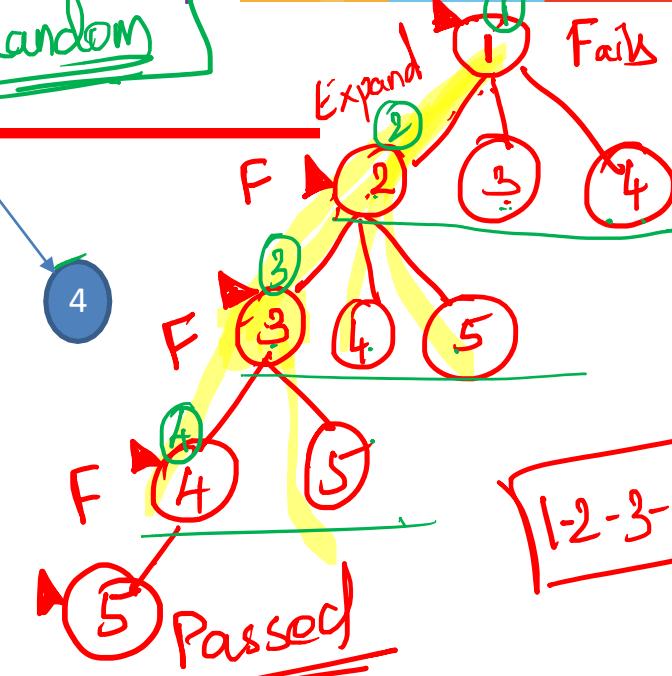
Expanded : 4

Generated : 10

Max Queue Length : 6

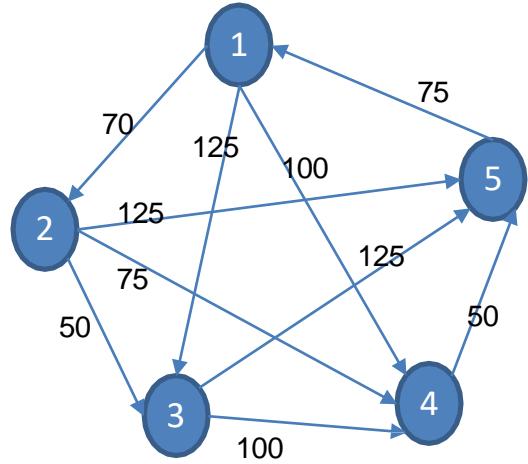
Stack Table - X

Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)	.	Fail on (1)
2.	(2)	(1)	Fails
3.	(3)	(1, 2)	F(123)
4.	(4)	(1, 2, 3)	F(1234)
5.	(5)	(1, 2, 3, 4)	P



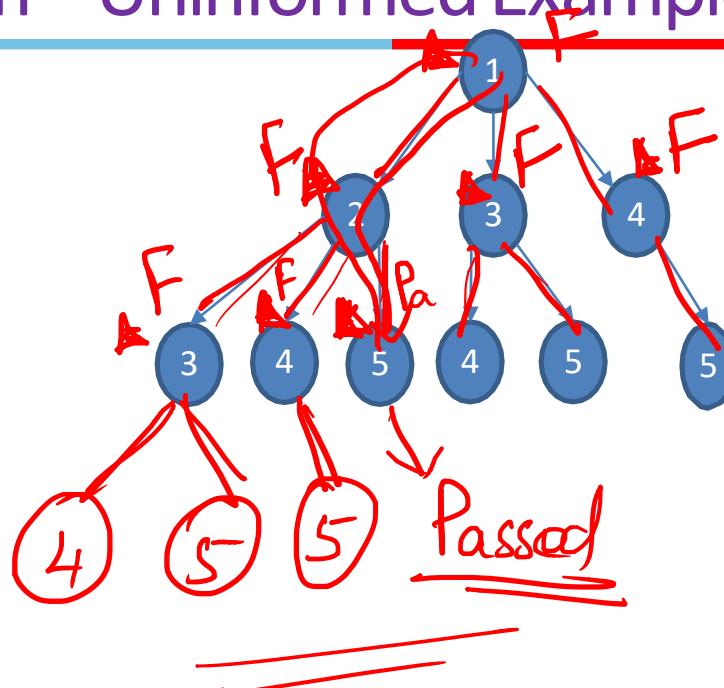
BFS

## Search Algorithm – Uninformed Example - 2



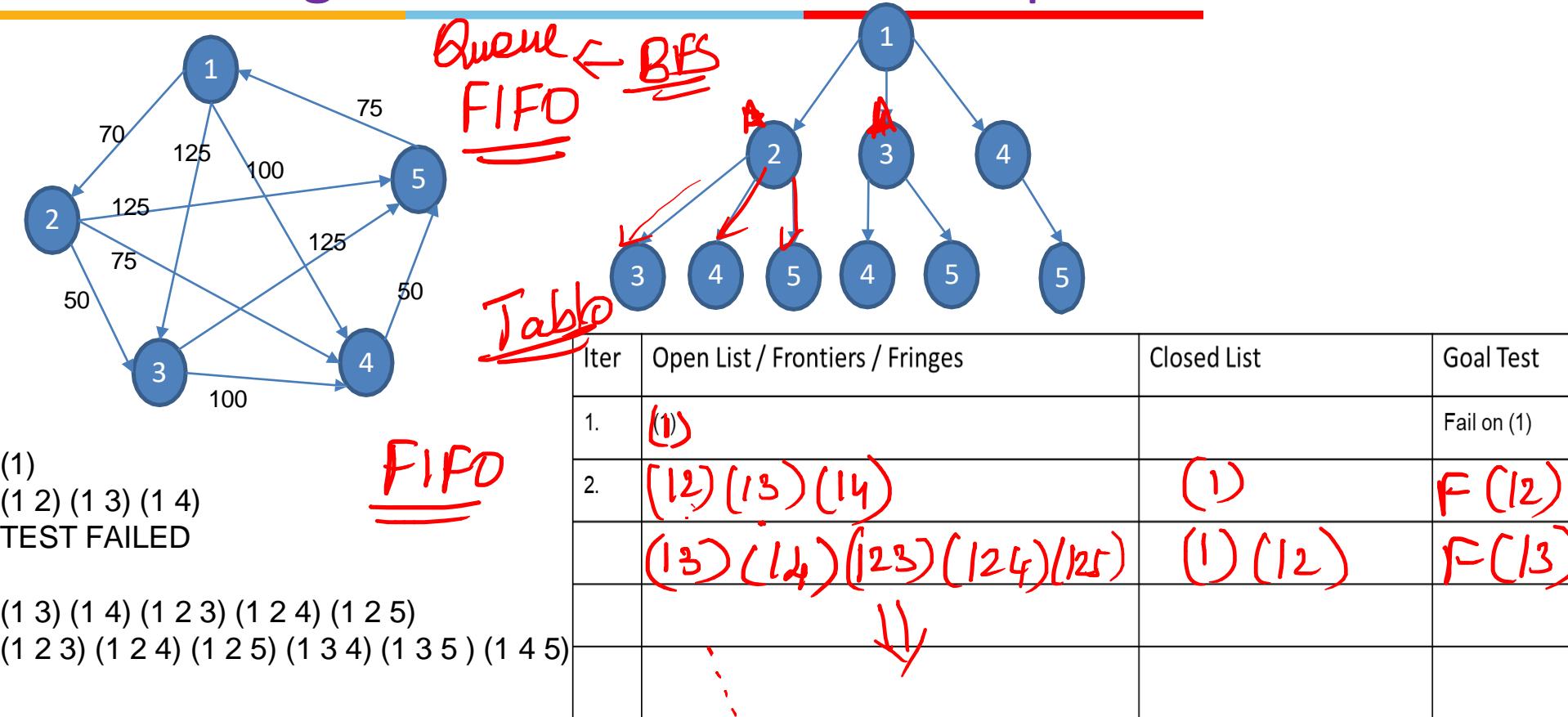
(1)  
 (1 2) (1 3) (1 4)  
 TEST FAILED

(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)  
 (1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)



~~C(1-2-5) = 70 + 125 = 195~~  
 Expanded : 4  
 Generated : 10  
 Max Queue Length : 6

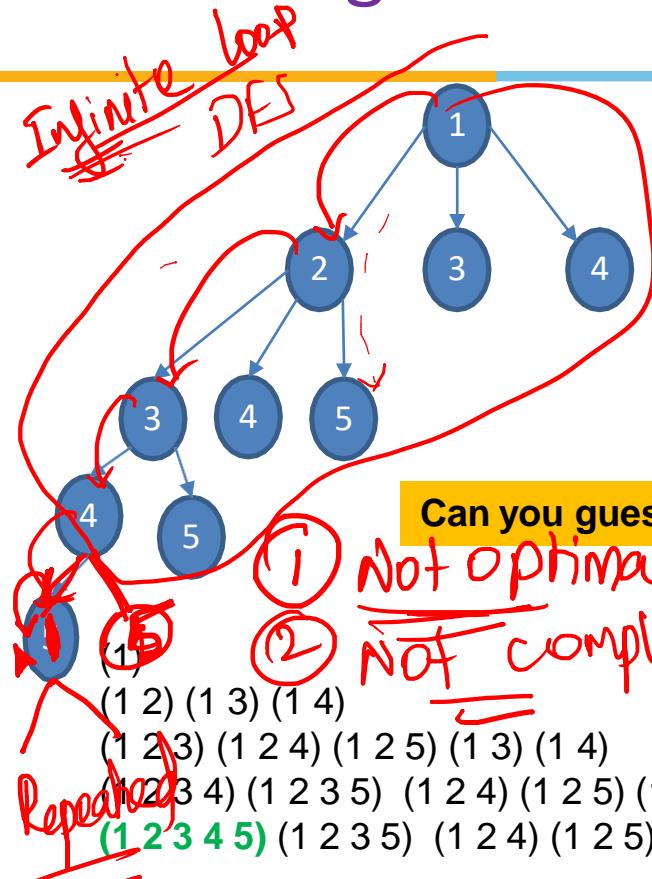
## Search Algorithm – Uninformed Example - 2



$C(1-2-5) = 70 + 125 = 195$   
 Expanded : 4  
 Generated : 10  
 Max Queue Length : 6

125

# Search Algorithm – Uninformed Example - 2

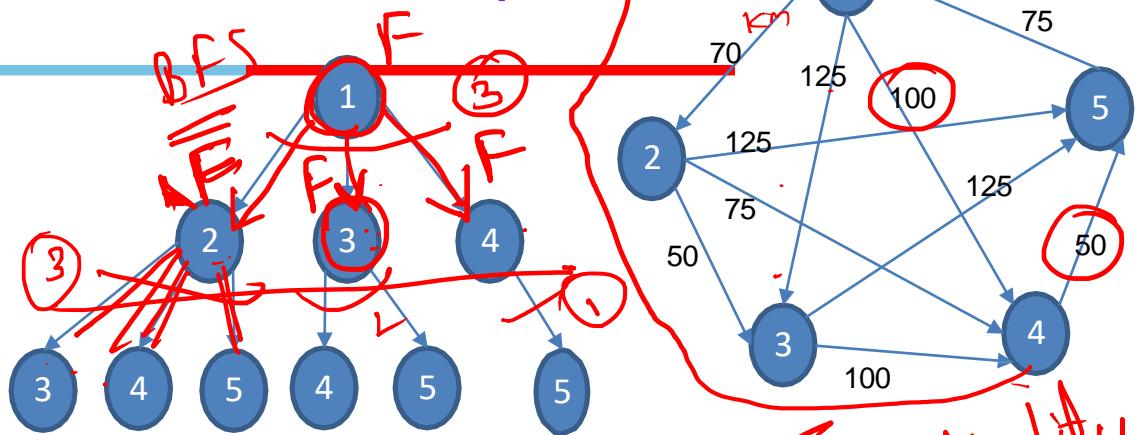


$$C(1-2-3-4-5) = 70 + 50 + 100 + 50 = 270$$

Expanded : 4

Generated : 10

Max Queue Length : 6



① can be optimal  
② Complete

(1) (1 2) (1 3) (1 4)  
TEST FAILED

(1 3) (1 4) (1 2 3) (1 2 4) (1 2 5)

(1 2 3) (1 2 4) (1 2 5) (1 3 4) (1 3 5) (1 4 5)

TEST PASSED

C(1-2-5) = 70 + 125 = 195

Expanded : 4

Generated : 10

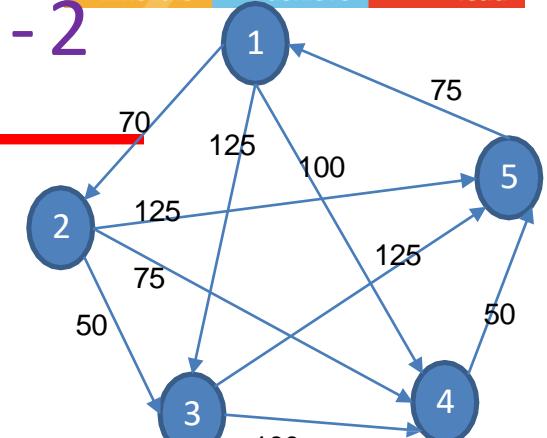
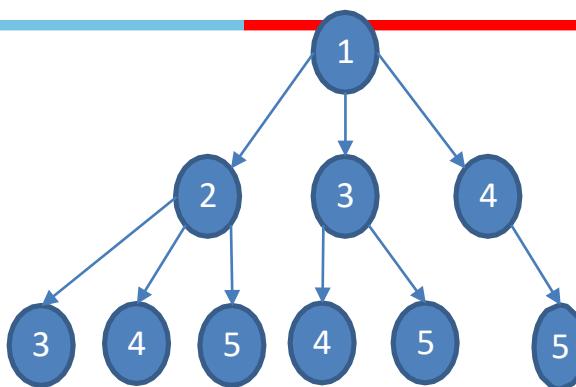
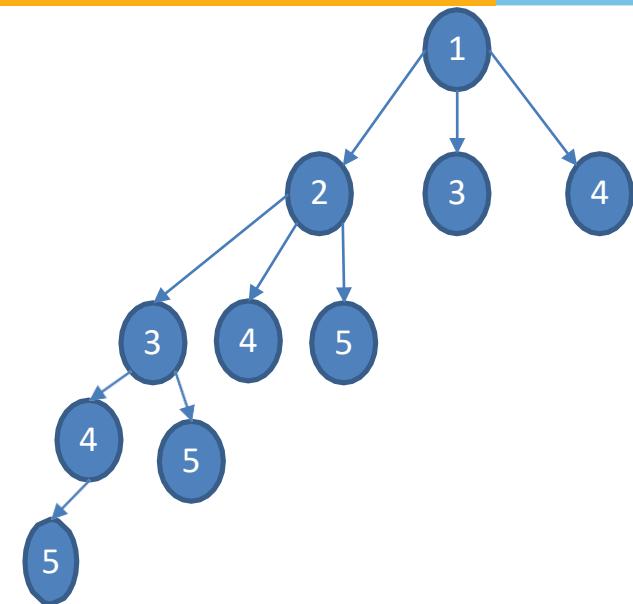
Max Queue Length : 6

1-4-5

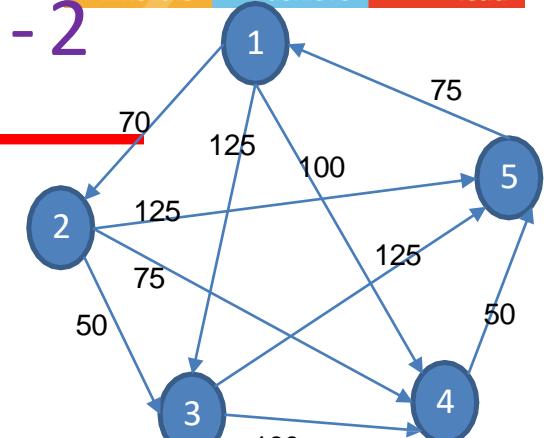
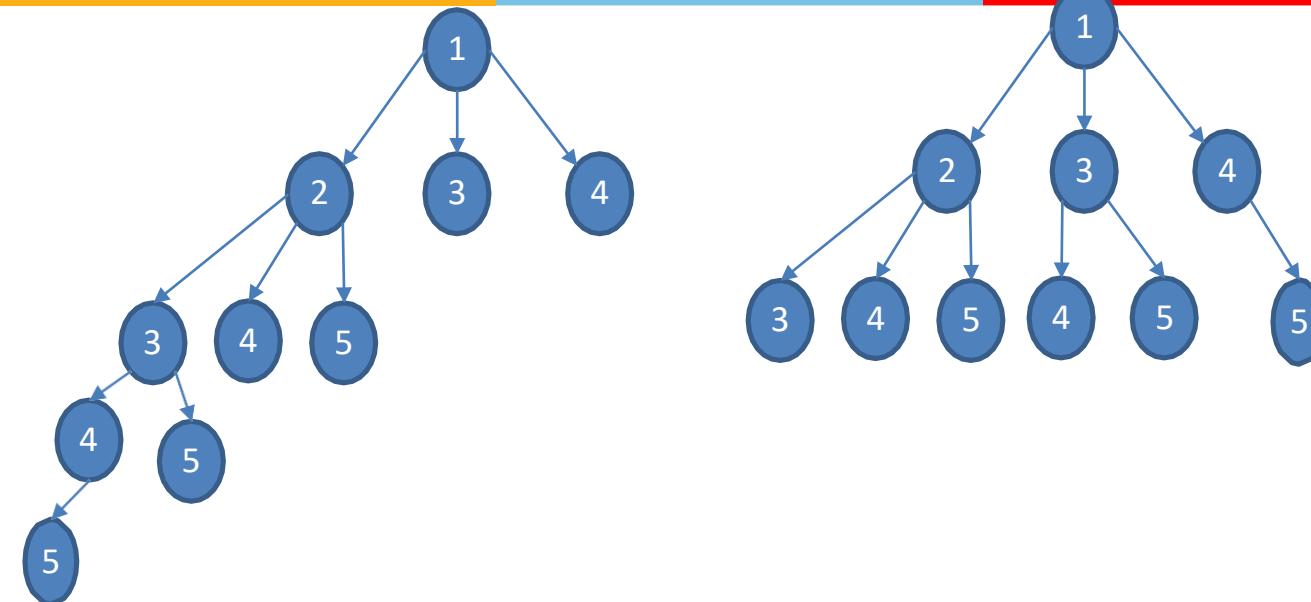
IFO



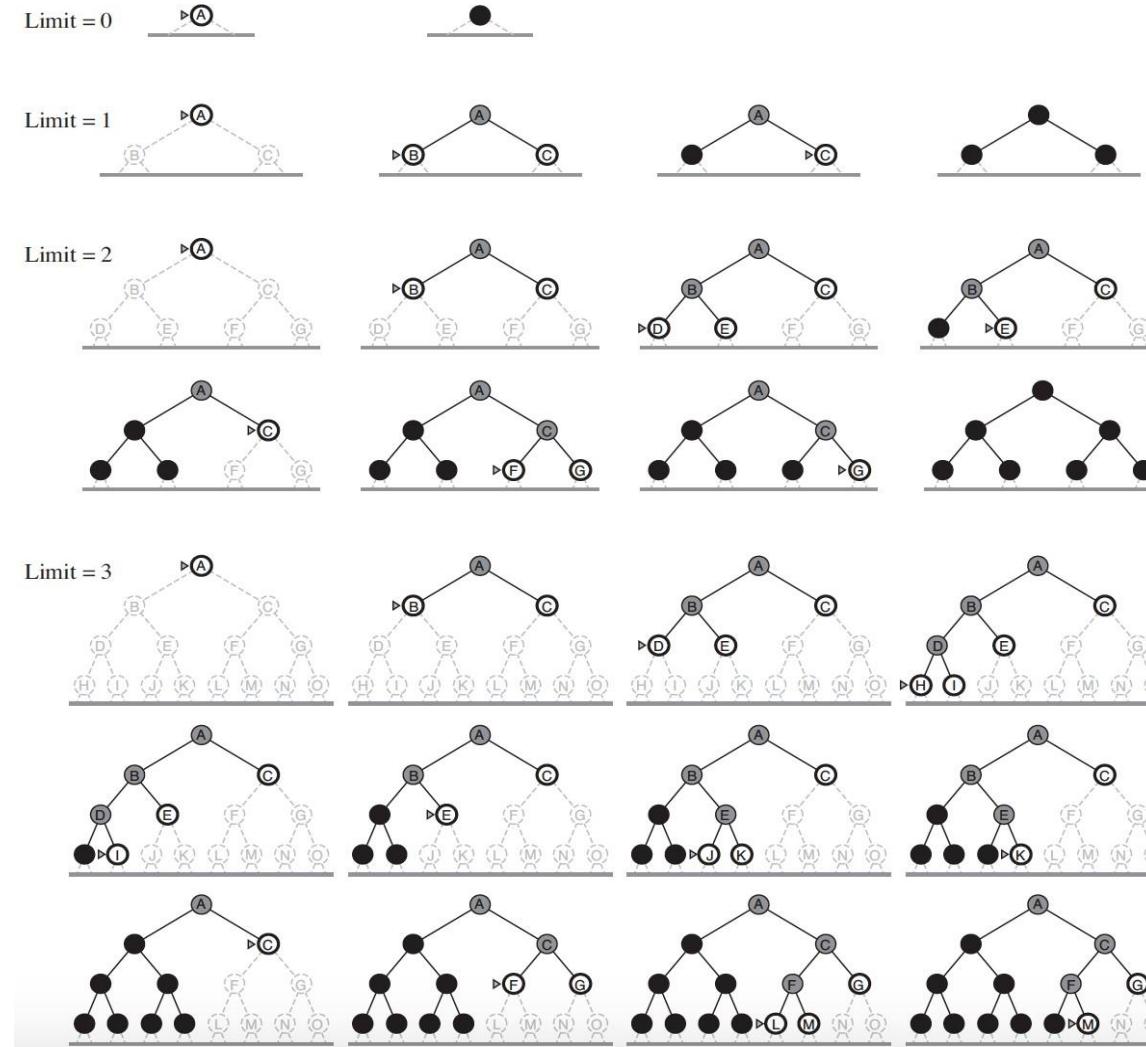
# Search Algorithm – Uninformed Example - 2



# Search Algorithm – Uninformed Example - 2



# Iterative Deepening Depth First Search (IDS)



# Algorithm Tracing

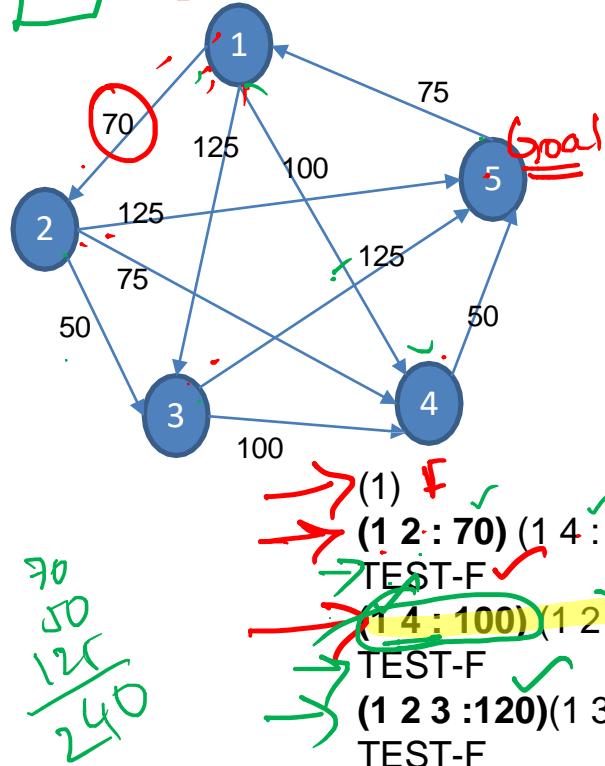
Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)		Fail on (1)
2.			

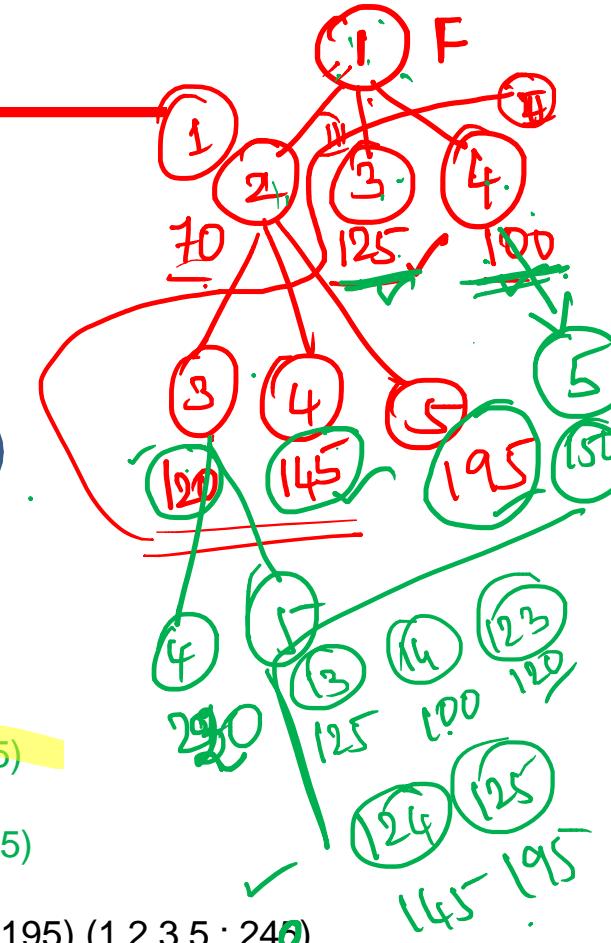
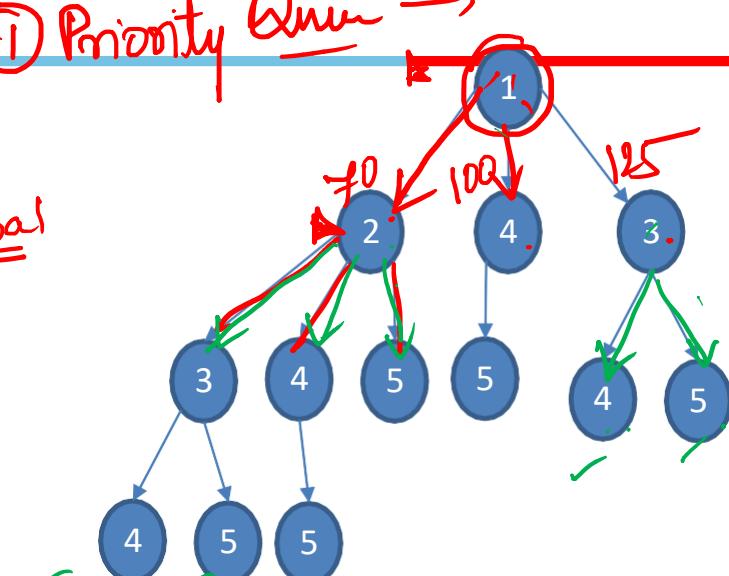
# UCS → Uniform Cost Search

$$\frac{U_{CS}}{T} = \underline{\underline{BFS}} + \text{Path cost}$$

A photograph of a child's handwriting on lined paper. The letter 'H' is written in green ink, and the word 'straat' is written in red ink below it.



~~Optimal Solution~~



TEST - P

# Uniform Cost Search

---

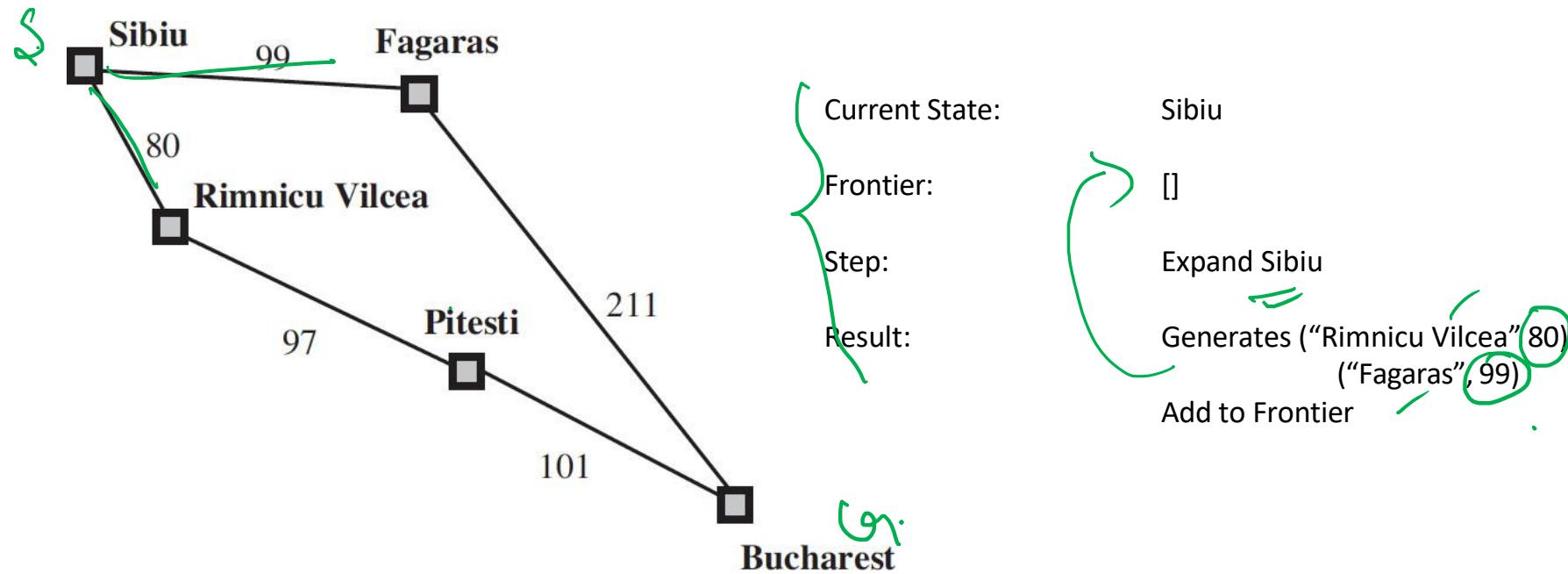
Instead of expanding the shallowest node, Uniform-Cost search expands the node  $n$  with the lowest path cost  $g(n)$

Sorting the Frontier as a priority queue ordered by  $g(n)$

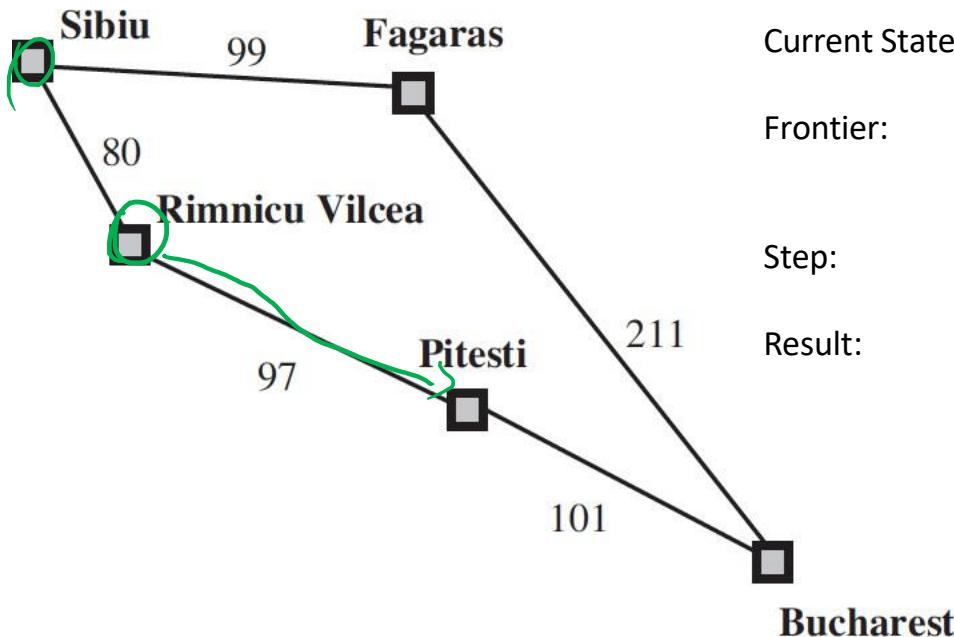
Goal test is applied during expansion

- The goal node if generated may not be on the optimal path
- Find a better path to a node on the Frontier

# Uniform Cost Search



# Uniform Cost Search



Current State:

Sibiu

Frontier:

[("Rimnicu Vilcea" 80)  
("Fagaras", 99)]

Step:

Expand "Rimnicu Vilcea" (least cost)

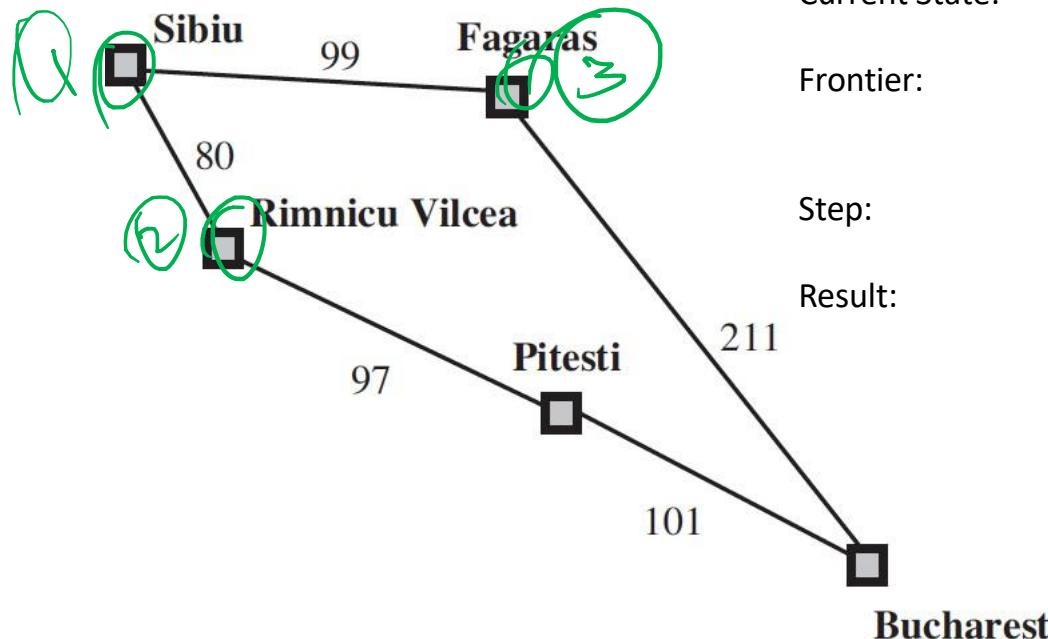
Result:

Generates ("Pitesti", 177)  
Add to Frontier

Initial State:  
Goal State:

Sibiu  
Bucharest

# Uniform Cost Search



Initial State:  
Sibiu

Goal State:  
Bucharest

Current State:

Rimnicu Vilcea (not a Goal state)

Frontier:

[ ("Fagaras", 99)  
("Pitesti", 177)]

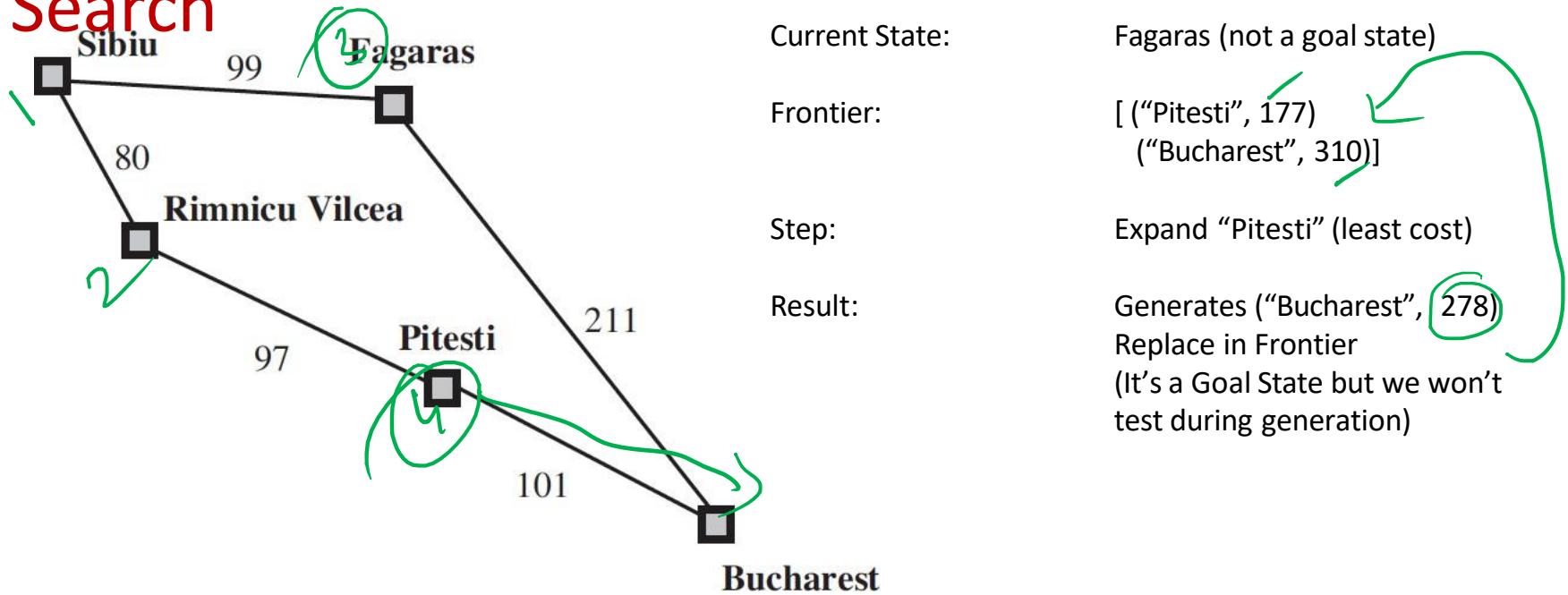
Step:

Expand "Fagaras" (least cost)

Result:

Generates ("Bucharest", 310)  
Add to Frontier  
(It's a Goal State but we won't  
test during generation)

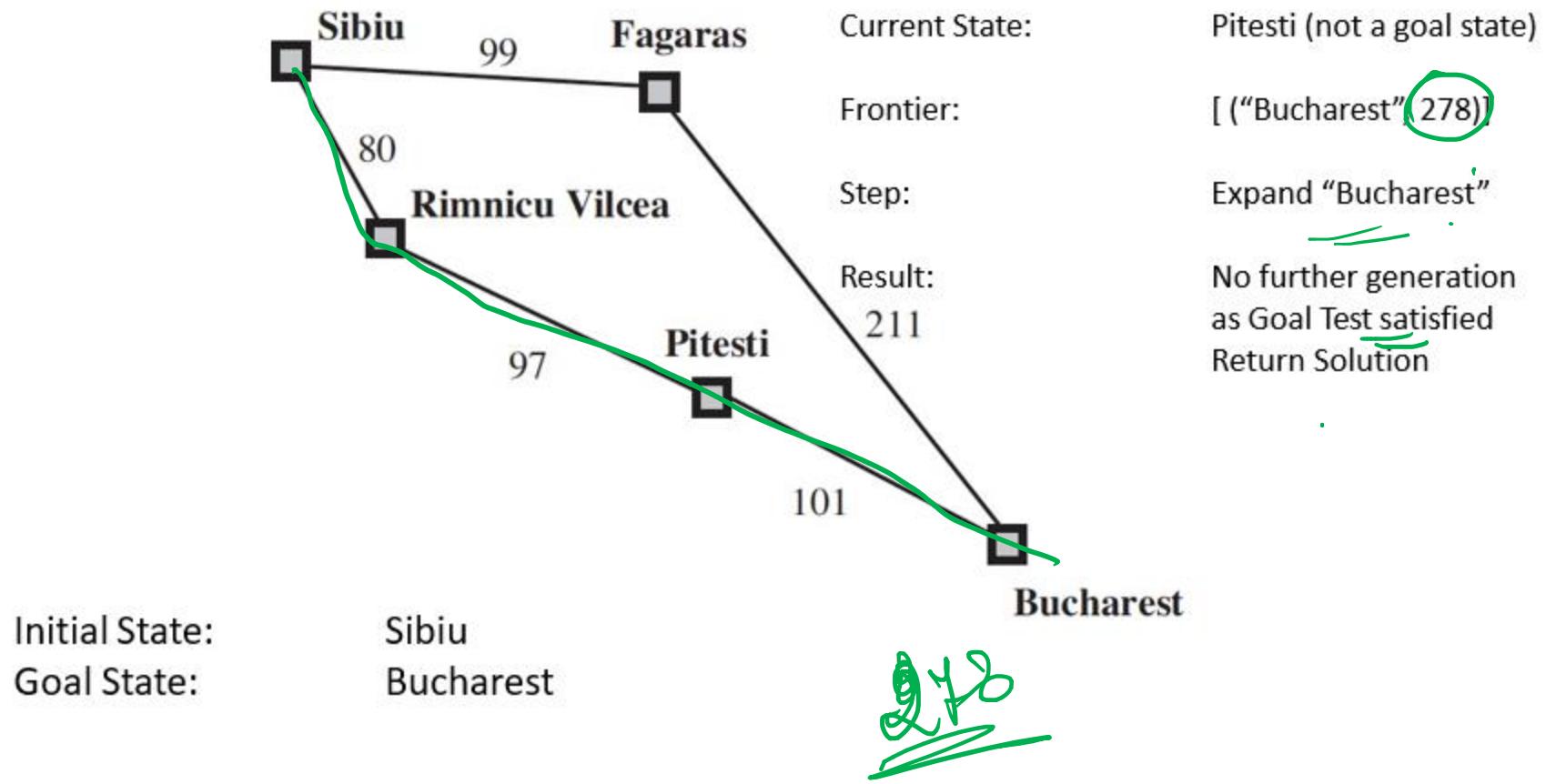
# Uniform Cost Search



Initial State:  
Sibiu

Goal State:  
Bucharest

# Uniform Cost Search



# Sample Evaluation of the Algorithm

**Complete** – If the shallowest goal node is at a depth  $d$ , BFS will eventually find it by generating all shallower nodes

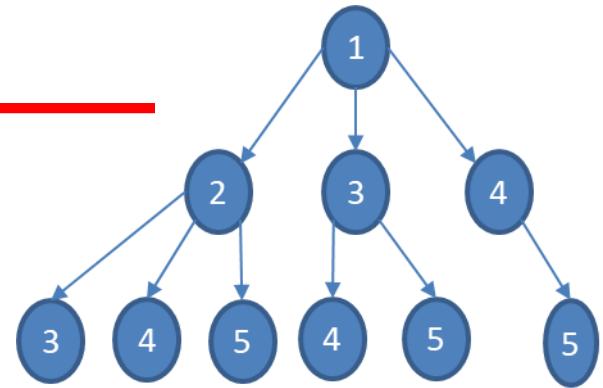
**Optimal** – Not necessarily. Optimal if path cost is non-decreasing function of depth of node.  
E.g., all actions have same cost

**Time Complexity** –  $G(b^d)$  b - branching factor, d – depth

- Nodes expanded at depth 1 =  $b$
- Nodes expanded at depth 2 =  $b^2$
- Nodes expanded at depth  $d$  =  $b^d$
- Goal test is applied during generation, time complexity would be  $G(b^{d+1})$

**Space Complexity** –  $G(b^d)$

- $G(b^{d-1})$  in explored set
- $G(b^d)$  in frontier set



# Uniform Cost Search – Evaluation

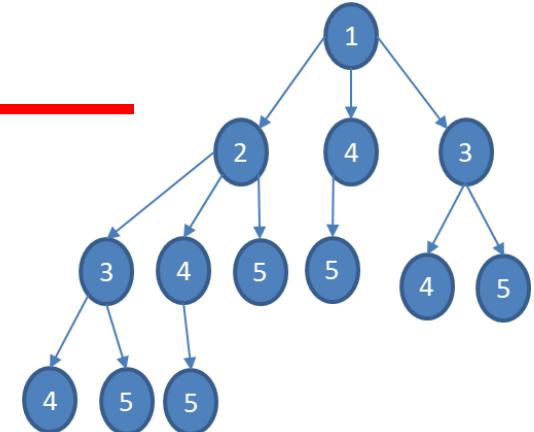
**Completeness** – It is complete if the cost of every step > small +ve constant  $\epsilon$

- It will stuck in infinite loop if there is a path with infinite sequence of zero cost actions

**Optimal** – It is Optimal. Whenever it selects a node, it is an optimal path to that node.

**Time and Space complexity** – Uniform cost search is guided by path costs not depth or branching factor.

- If  $C^*$  is the cost of optimal solution and  $\epsilon$  is the min. action cost
- Worst case complexity =  $G(b^{1+\frac{C^*}{\epsilon}})$ ,
- When all action costs are equal  $\square$   $G(b^{d+1})$ , the BFS would perform better
  - As Goal test is applied during expansion, Uniform Cost search would do extra work



# Terminologies – Learnt Today

---

- Nodes
- States
- Frontier | Fringes
- Search Strategy : LIFO | FIFO | Priority Queue
- Performance Metrics
  - Completeness
  - Optimality
  - Time Complexity
  - Space Complexity
- Algorithm Terminology
  - d Depth of a node
  - b Branching factor
  - n – nodes
  - l – level of a node
  - m – maximum
  - $C^*$  - Optimal Cost
  - E – least Cost
  - N – total node generated

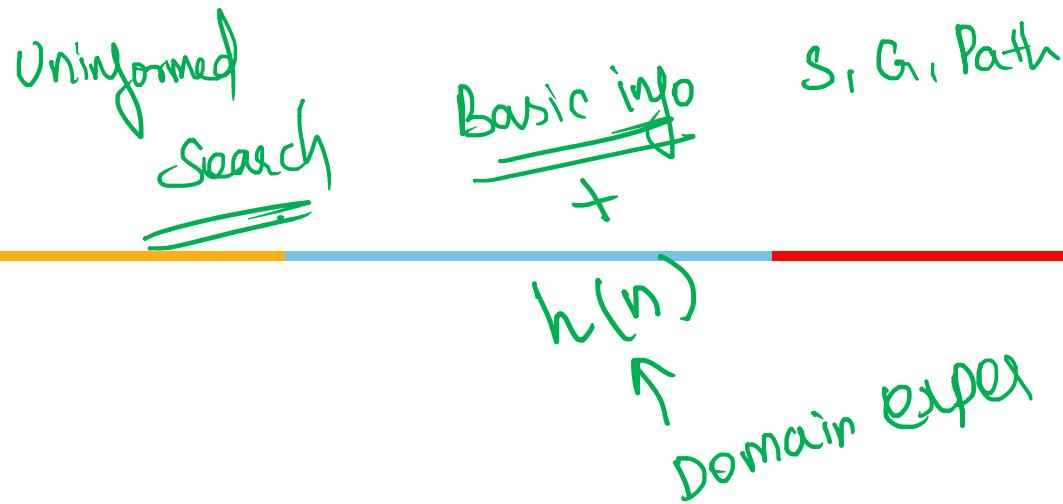
## Module 2 : Problem Solving Agent using Search

- A. Uninformed Search
- B. Informed Search
- C. Heuristic Functions
- D. Local Search Algorithms & Optimization Problems

## Learning Objective

At the end of this class , students Should be able to:

1. Differentiate between uninformed and informed search requirements
2. Apply UCS, GBFS & A\* algorithms to the given problem
3. Prove if the given heuristics are admissible and consistent
4. Design and compare heuristics apt for given problem
5. Apply A\* variations algorithms to the given problem



(Informed Search)

Greedy Best First ✓

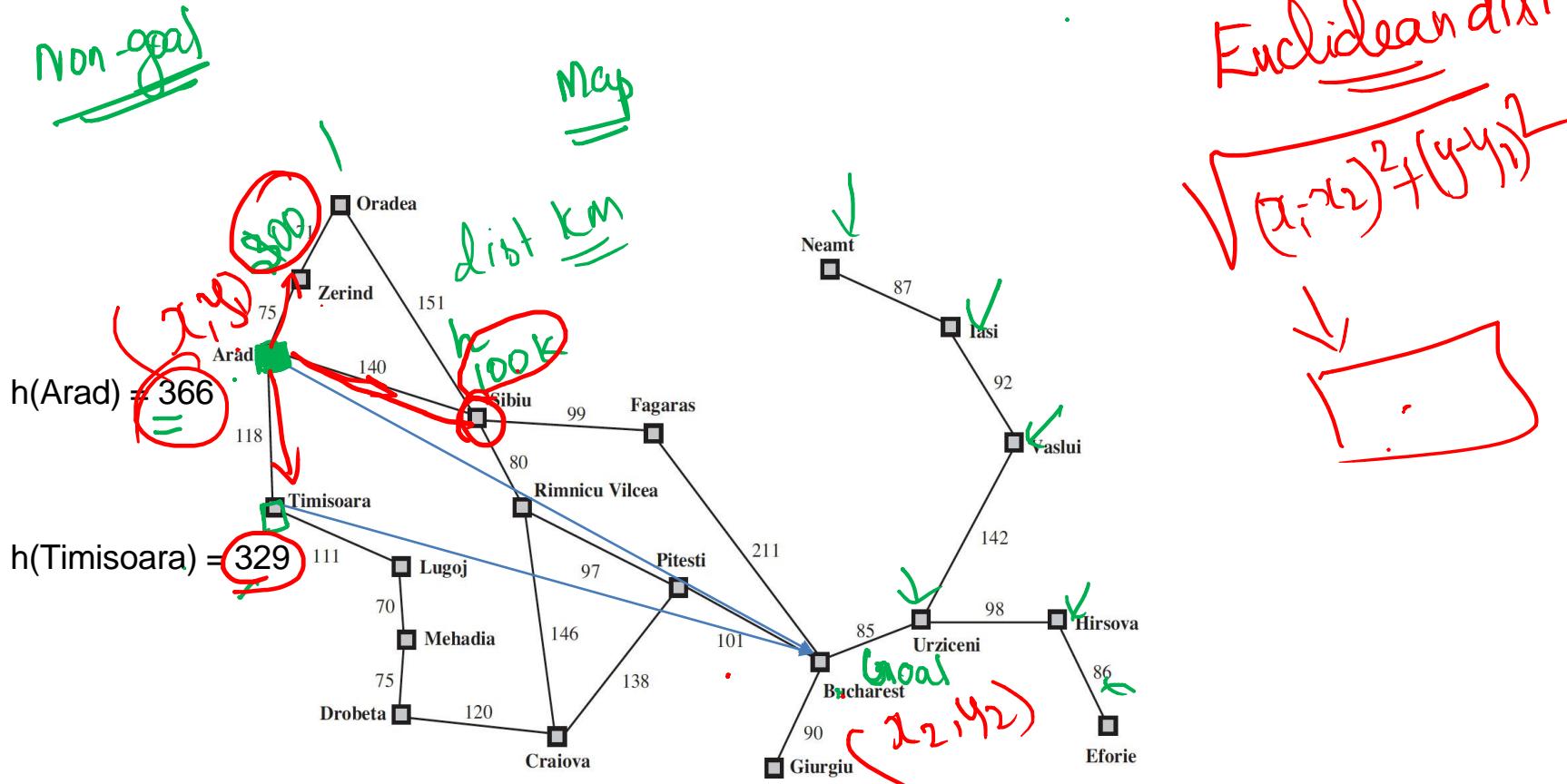
A\*

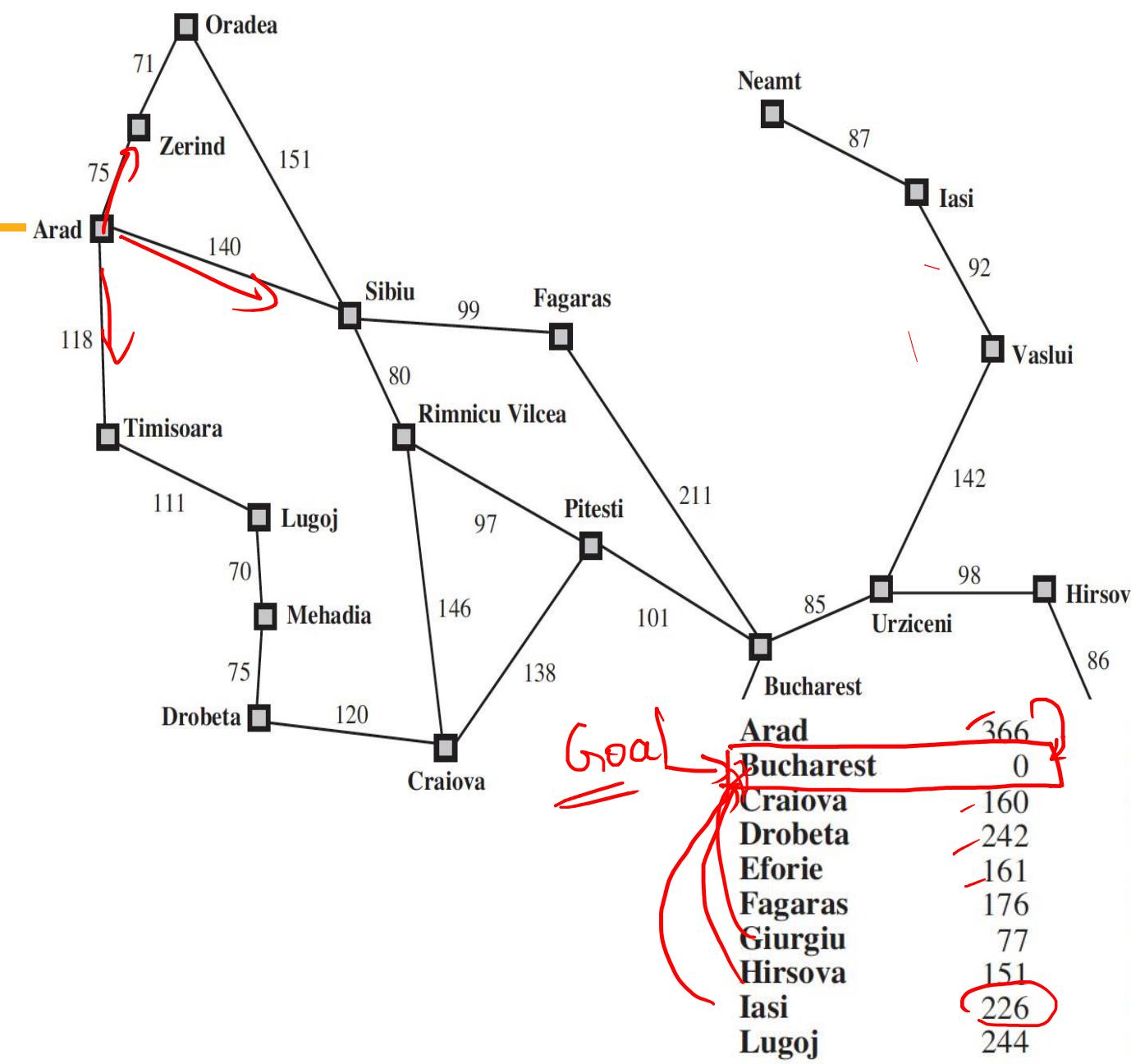
✗

✗  
2var

## Informed / Heuristic Search

Strategies that know if one non-goal state is more promising than another non-goal state





<b>Arad</b>	366
<b>Bucharest</b>	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244

## Greedy Best First Search

Expands the node that is closest to the goal

Thus,  $f(n) = h(n)$

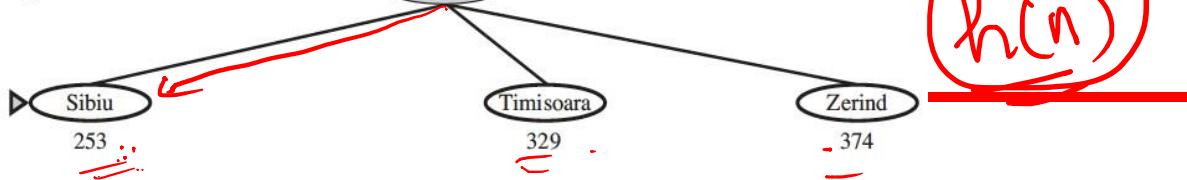
$h(n)$

<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

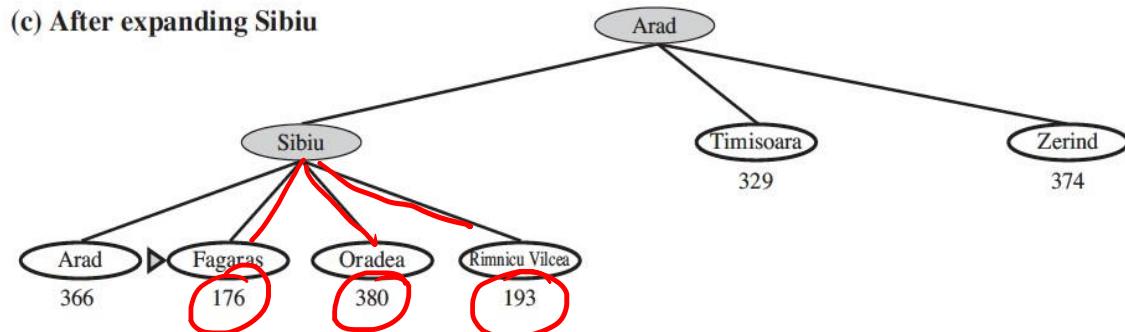
(a) The initial state



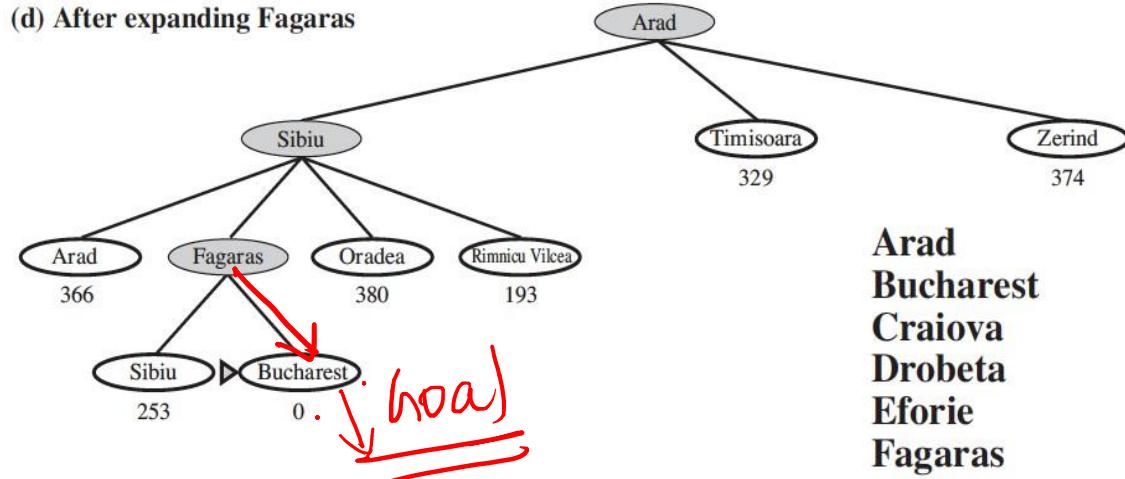
(b) After expanding Arad



(c) After expanding Sibiu

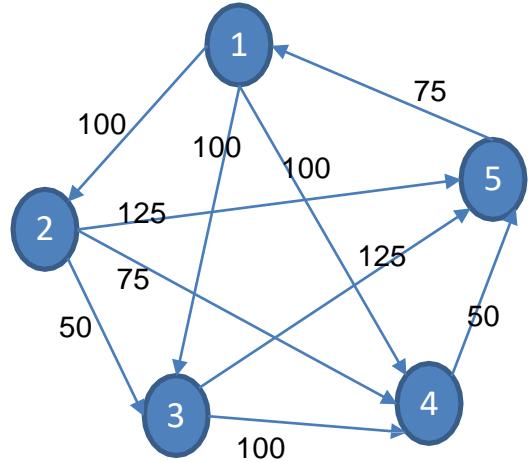


(d) After expanding Fagaras



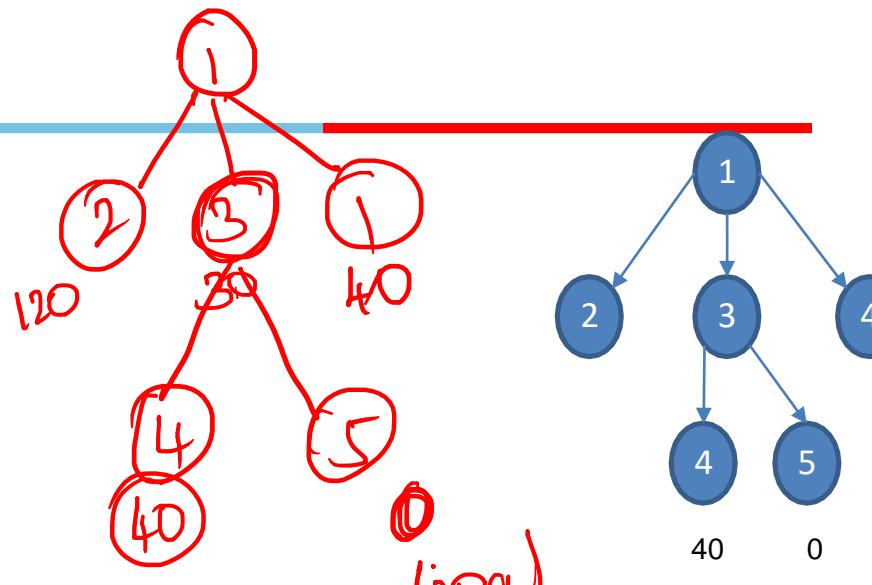
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

# Greedy Best First Search



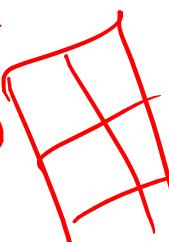
Given

n	$h(n)$
1	60
2	120 ✓
3	30 ✓
4	40 ✓
5	0



$C(1-3-5) = 100 + 125 = 225$   
 Expanded : 2  
 Generated : 6  
 Max Queue Length : 3

$H(n) \rightarrow$  Priority  
 $h(n)$

Hint  
 $h(n)$ 


## A\* Search

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

=

Paths worth

$f(n) \Rightarrow h(n) + g(n)$

Expands the node which lies in the closest path (estimated cheapest path) to the goal

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

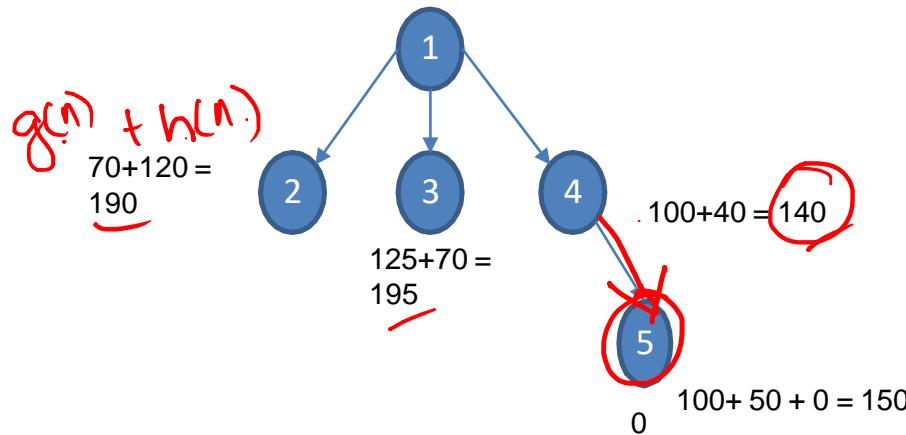
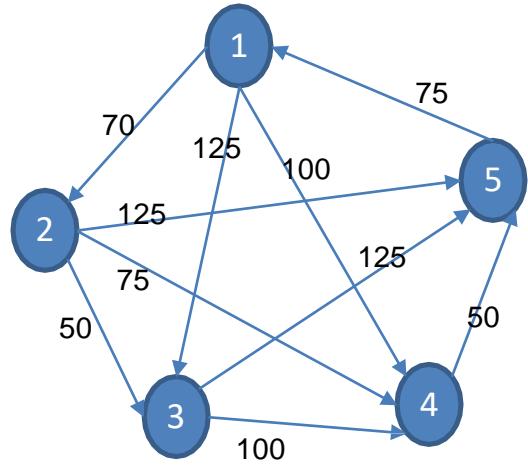
$g(n)$  – the cost to reach the node

$h(n)$  – the expected cost to go from node to goal

$f(n)$  – estimated cost of cheapest path through node n

<b>Arad</b>	366	<b>Mehadia</b>	241
<b>Bucharest</b>	0	<b>Neamt</b>	234
<b>Craiova</b>	160	<b>Oradea</b>	380
<b>Drobeta</b>	242	<b>Pitesti</b>	100
<b>Eforie</b>	161	<b>Rimnicu Vilcea</b>	193
<b>Fagaras</b>	176	<b>Sibiu</b>	253
<b>Giurgiu</b>	77	<b>Timisoara</b>	329
<b>Hirsova</b>	151	<b>Urziceni</b>	80
<b>Iasi</b>	226	<b>Vaslui</b>	199
<b>Lugoj</b>	244	<b>Zerind</b>	374

A\* Search → Variant of BFS



n	$h(n)$
1	60
2	120
3	70
4	40
5	0

(1)  
 (1 4) (1 2) (1 3)  
 (1 4 5) (1 2) (1 3)

$C(1-4-5) = 100 + 150 = 150$   
 Expanded : 2  
 Generated : 5  
 Max Queue Length : 3

1-4-5

**Required Reading:** AIMA - Chapter #1, 2, 3.1, 3.2, 3.3

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



**BITS** Pilani  
Pilani Campus



# **Artificial & Computational Intelligence**

**DSE CLZG557**

**M1 : Introduction  
&**

**M2 : Problem Solving Agent using Search**

Indumathi V  
Guest Faculty,  
BITS - WILP

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

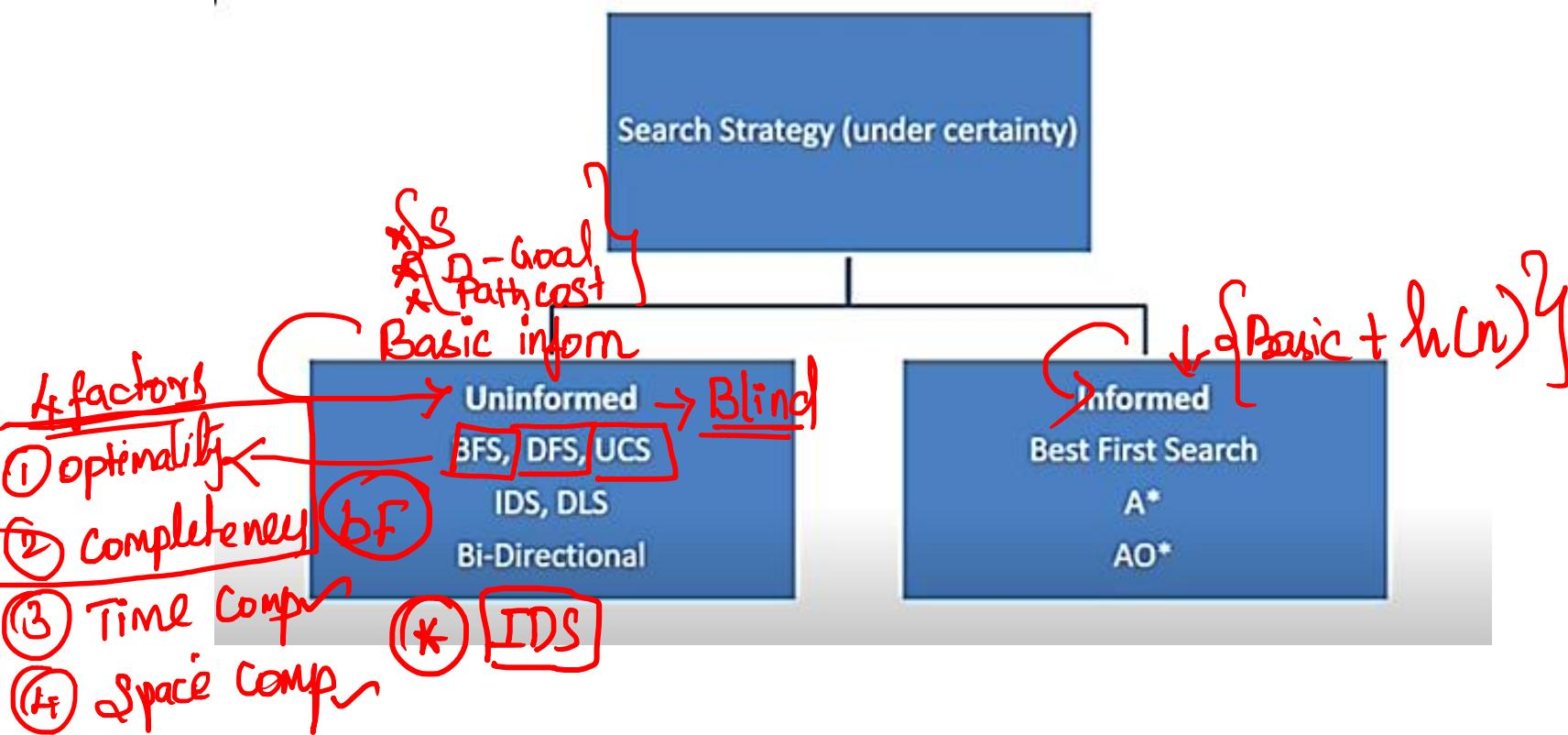
M5 Probabilistic Representation and Reasoning

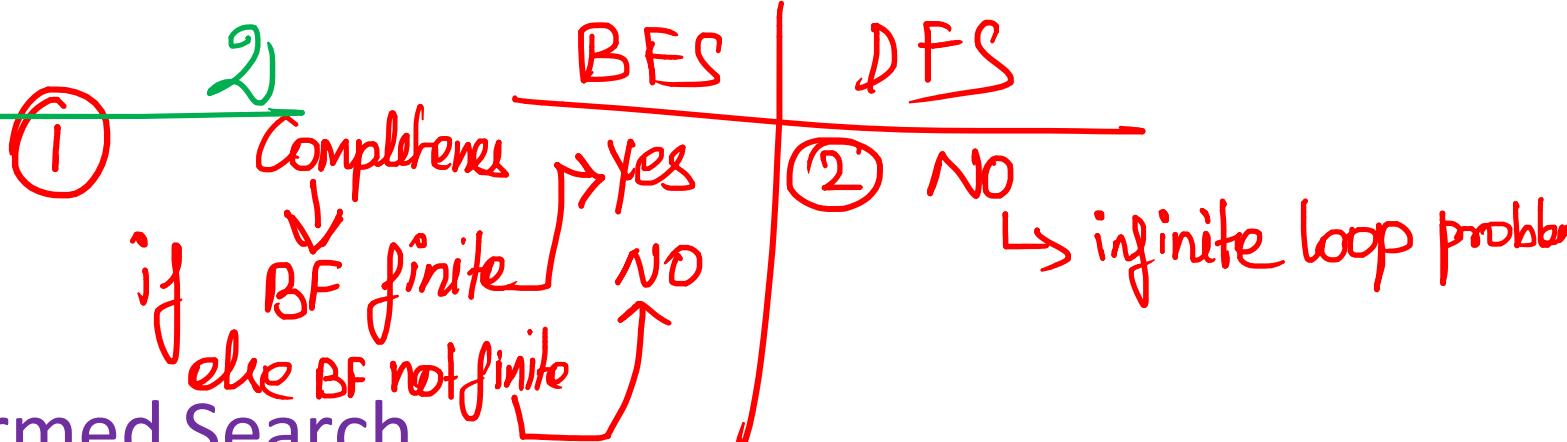
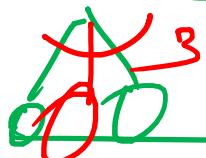
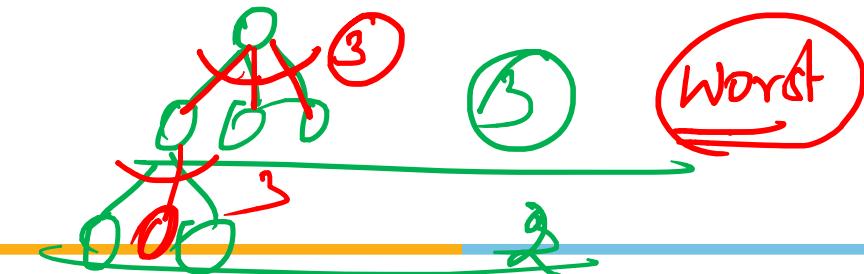
M6 Reasoning over time, Reinforcement Learning

M7 Ethics in AI

# Searching for Solutions

Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy



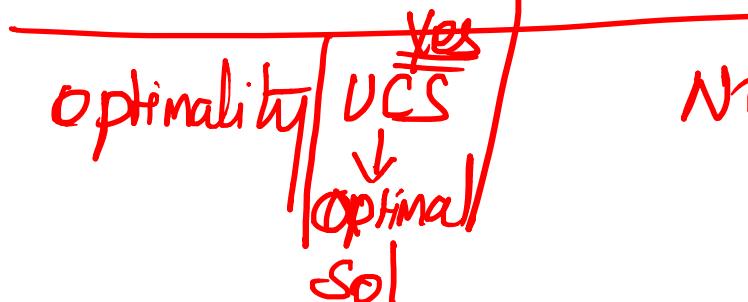


## Uninformed Search

### Overview

0

②



# Application

find the path from my home loc  
to Chinese Restaurant



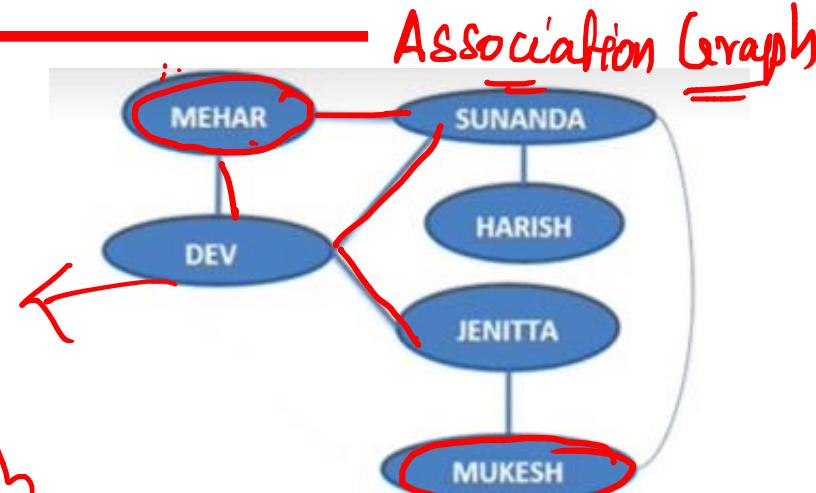
## Depth First Search

- Find the Connectedness in a graph
- Topological Sort

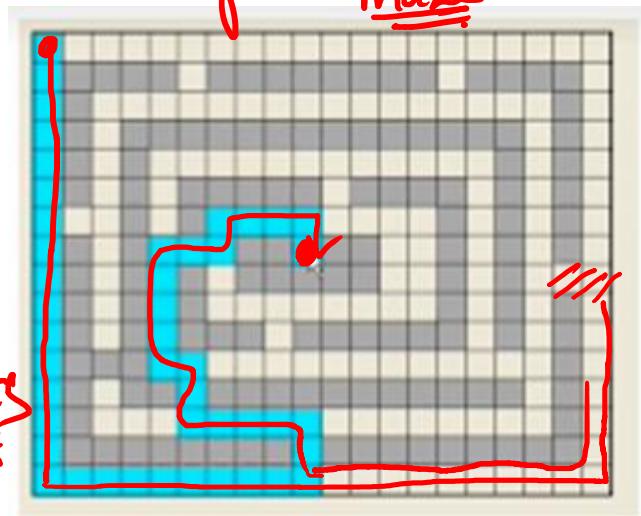
2 ways

① Retain explored  
↳ Tree based search approach

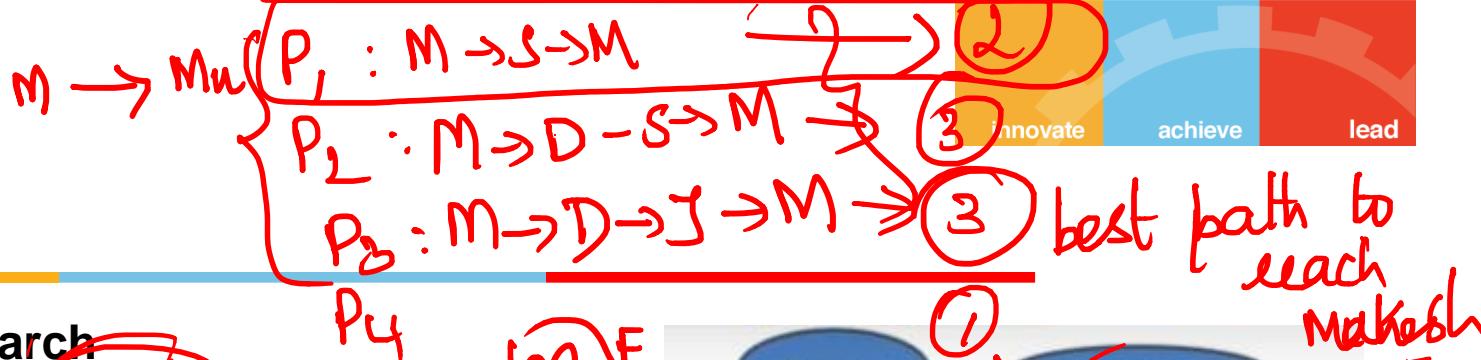
② Remember explored  
↳ Graph Search



eg Maze  
Sequence  
DFS



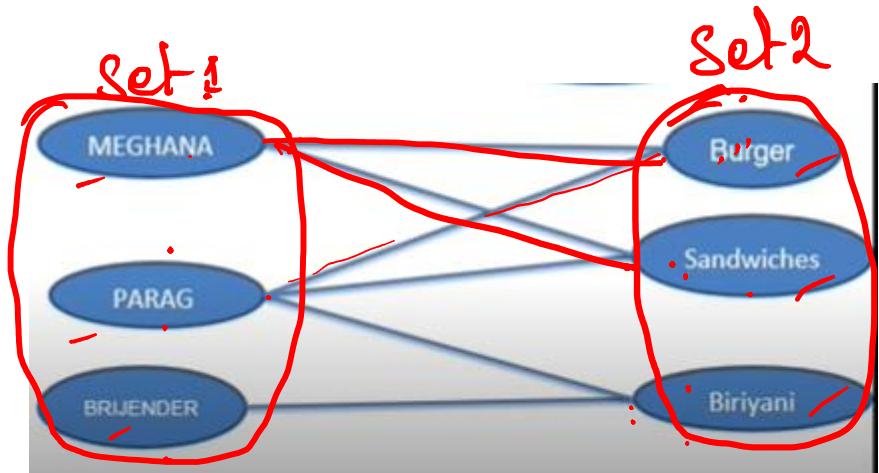
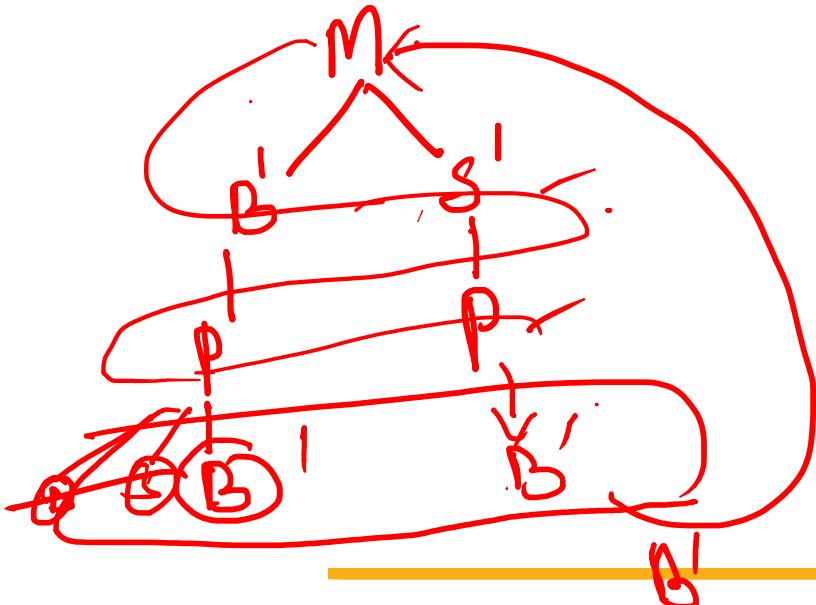
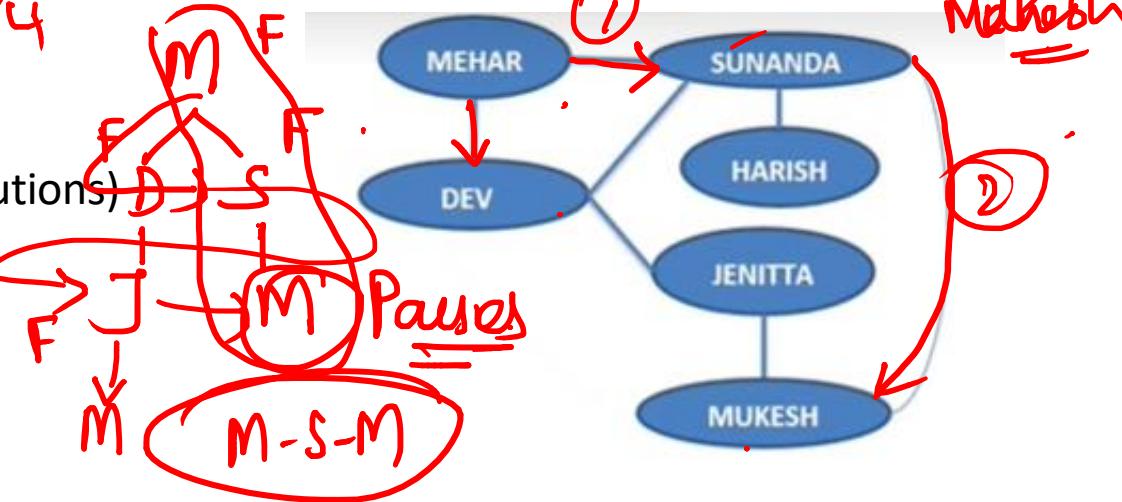
# Application



## Breadth First Search

- Finding path in a graph (many solutions)
- Finding the Bipartitions in a graph

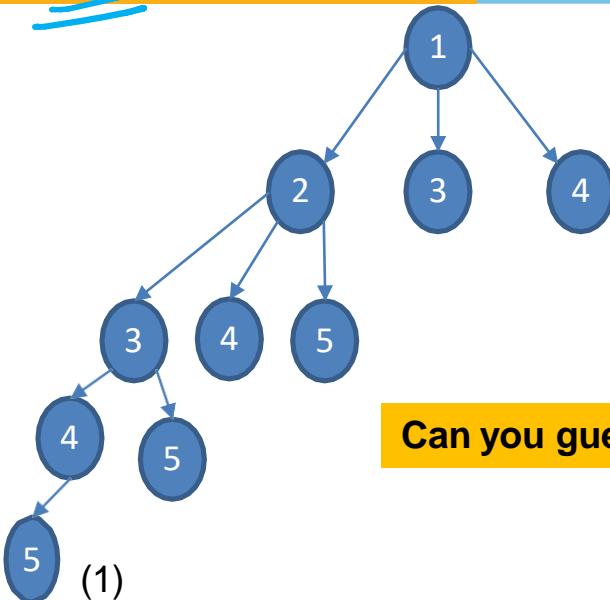
=  
2 different set



Food Preference graph

## Search Algorithm – Uninformed Example - 2

DPS



Can you guess which algorithm are these ?

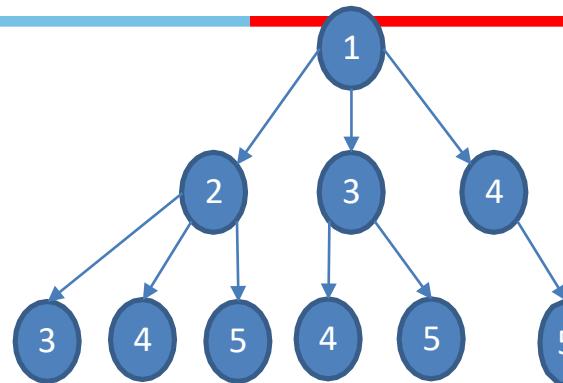
$(1)$   
 $(1\ 2)\ (1\ 3)\ (1\ 4)$   
 $(1\ 2\ 3)\ (1\ 2\ 4)\ (1\ 2\ 5)\ (1\ 3)\ (1\ 4)$   
 $(1\ 2\ 3\ 4)\ (1\ 2\ 3\ 5)\ (1\ 2\ 4)\ (1\ 2\ 5)\ (1\ 3)\ (1\ 4)$   
 $\textcolor{red}{(1\ 2\ 3\ 4\ 5)}$   $(1\ 2\ 3\ 5)\ (1\ 2\ 4)\ (1\ 2\ 5)\ (1\ 3)\ (1\ 4)$

$$C(1-2-3-4-5) = 70 + 50 + 100 + 50 = 270$$

Expanded : 4

Generated : 10 DFS  
Max Queue Length : 6

Generated : 10 BFS  
Max Queue Length : 6



$(1)$   
 $(1\ 2)\ (1\ 3)\ (1\ 4)$   
**TEST FAILED**

$(1\ 3)\ (1\ 4)\ (1\ 2\ 3)\ (1\ 2\ 4)\ (1\ 2\ 5)$   
 $(1\ 2\ 3)\ (1\ 2\ 4)\ (1\ 2\ 5)\ (1\ 3\ 4)\ (1\ 3\ 5)\ (1\ 4\ 5)$   
**TEST PASSED**

$$C(1-2-5) = 70 + 125 = 195$$

Expanded : 4

Generated : 10 BFS  
Max Queue Length : 6

## Search Algorithm – Uninformed Example - 2

Eg

bF, depth

Max Avg

bF: 3

bF: 2

bF: 1

bF: 0

bF: 1

bF: 2

bF: 3

depth Max

$O(b \cdot d)$

$O(b \cdot d)$

$3 \times 4 = 12$

DFS

# nodes generated

d

time complexity

bF: 3

1

0

$3^0 \rightarrow b$

3

1

$3^1 \rightarrow b$

3

2

$3^2 \rightarrow b$

3

3

$3^3 \rightarrow b$

3

3

$3^4 \rightarrow b$

3

3

$3^5 \rightarrow b$

3

3

$3^6 \rightarrow b$

3

3

$3^7 \rightarrow b$

3

3

$3^8 \rightarrow b$

3

3

$3^9 \rightarrow b$

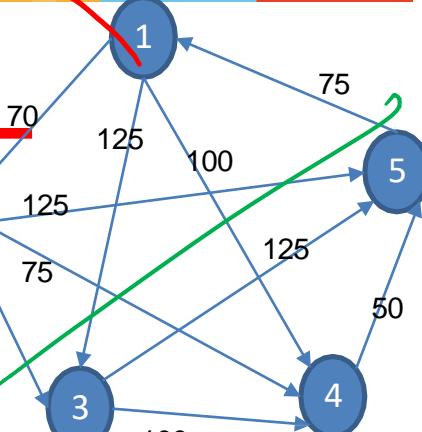
3

3

$3^{10} \rightarrow b$

n <

n = 10



$O(n)$

$i=0$  to  $9 \Rightarrow O(k)$  time  
(10 times)

for ( $i=0$ ;  $i < n$ ;  $i++$ )  
{  
 imp open  
}

Tot. no nodes generated

$1 + 4$

$B \cdot d$

$3 \times 4$

$= 12$

$O(b \cdot d)$

$3 \times 4$

$= 12$

$d \rightarrow$  depth @ which goal node is founded

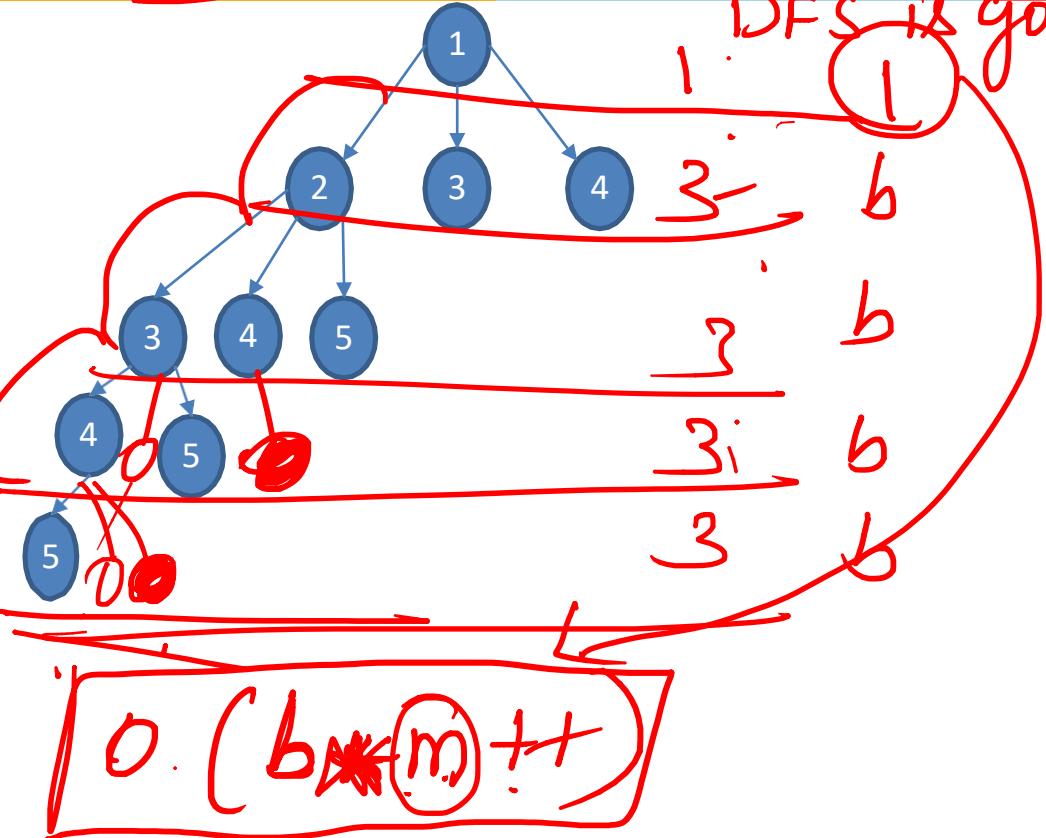
## Search Algorithm - Uninformed Example - 2

b, d

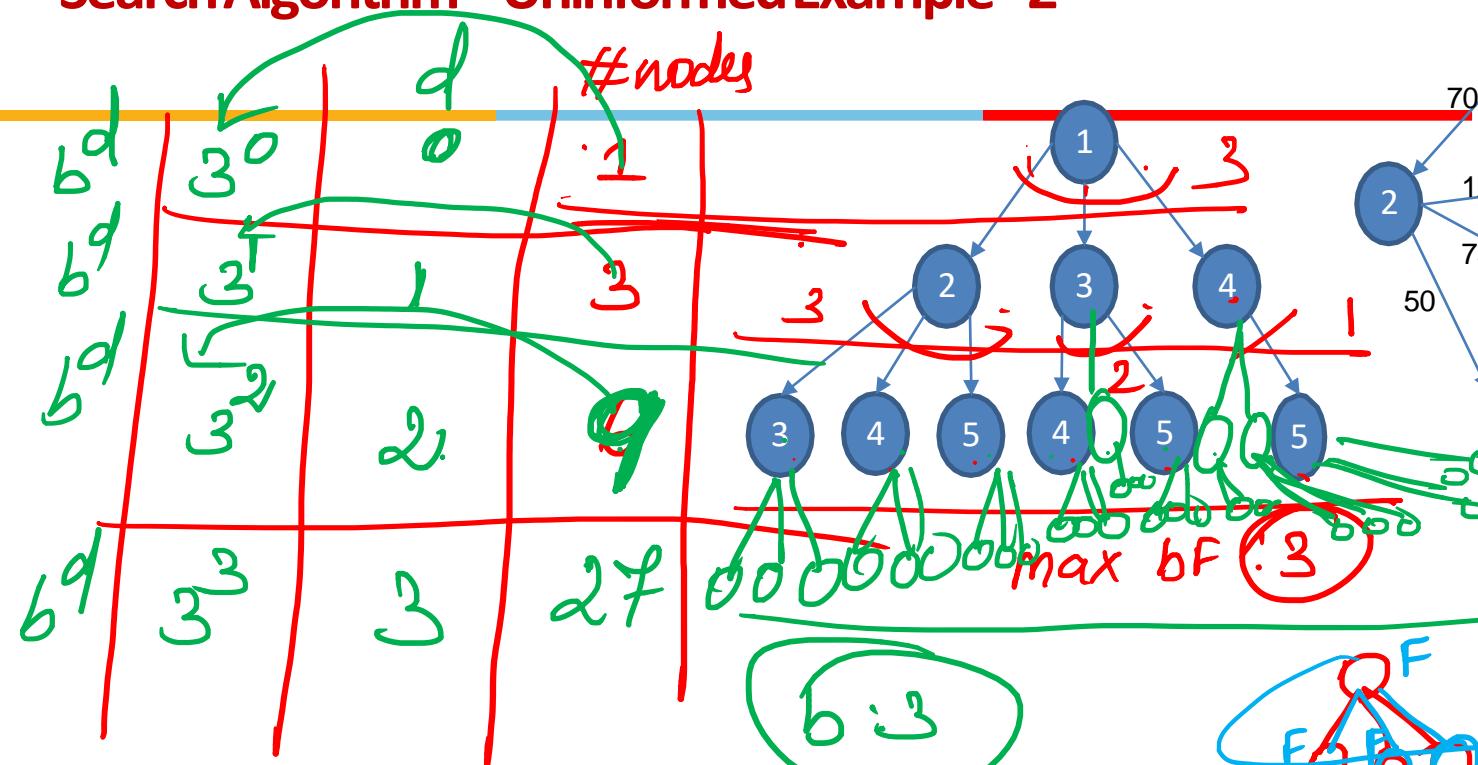
(DFS)

$m \rightarrow$  max depth at which

DFS is going to expand



## Search Algorithm – Uninformed Example - 2

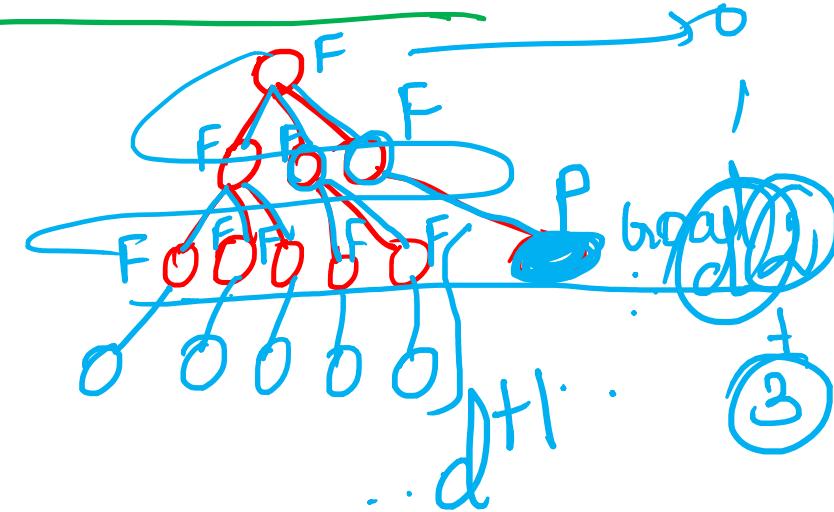


# nodes generated

$$O(b^d)$$

$$O(b^{d+1})$$

$$O(b^{d+1})$$



DFS → best memory  
=

IDS

## Iterative Deepening Depth First Search (IDS)

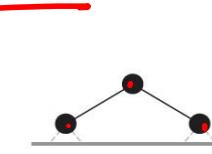
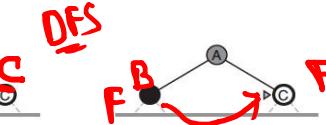
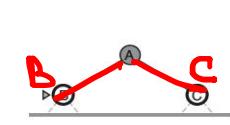
BFS → Expans.



DFS → bad  
Restart

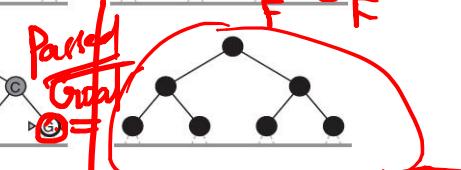
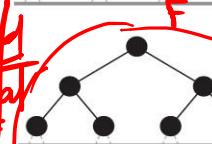
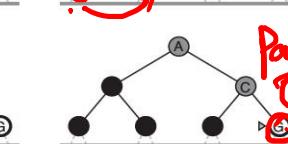
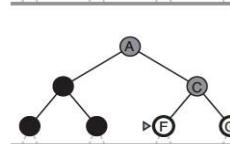
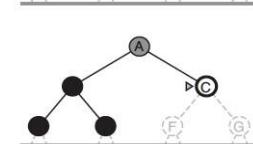
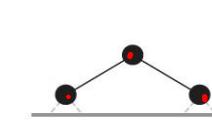
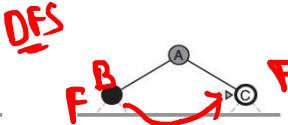
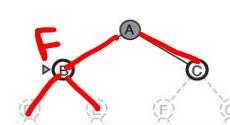
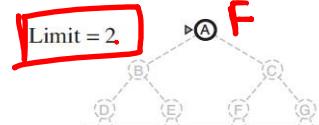
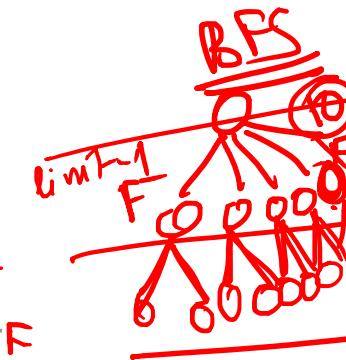


IDS



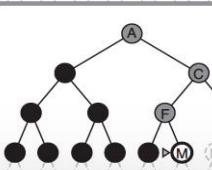
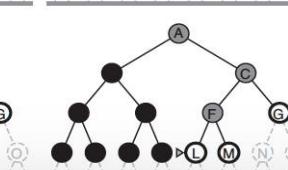
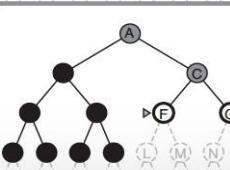
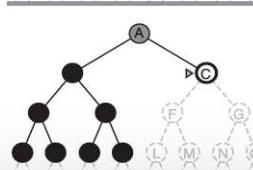
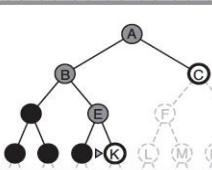
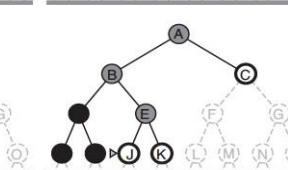
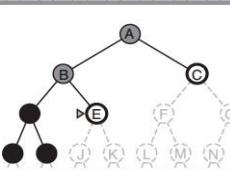
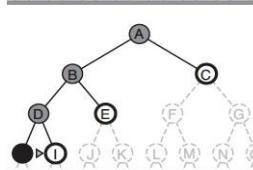
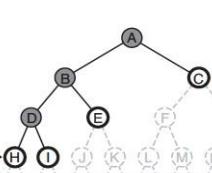
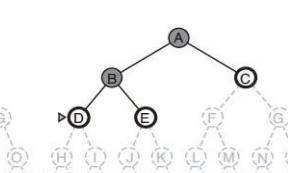
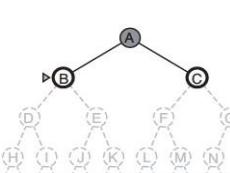
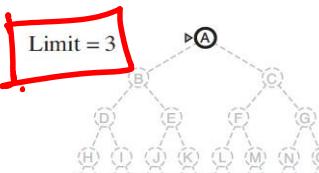
S: A  
Goal : G

BFS



Parity  
Error

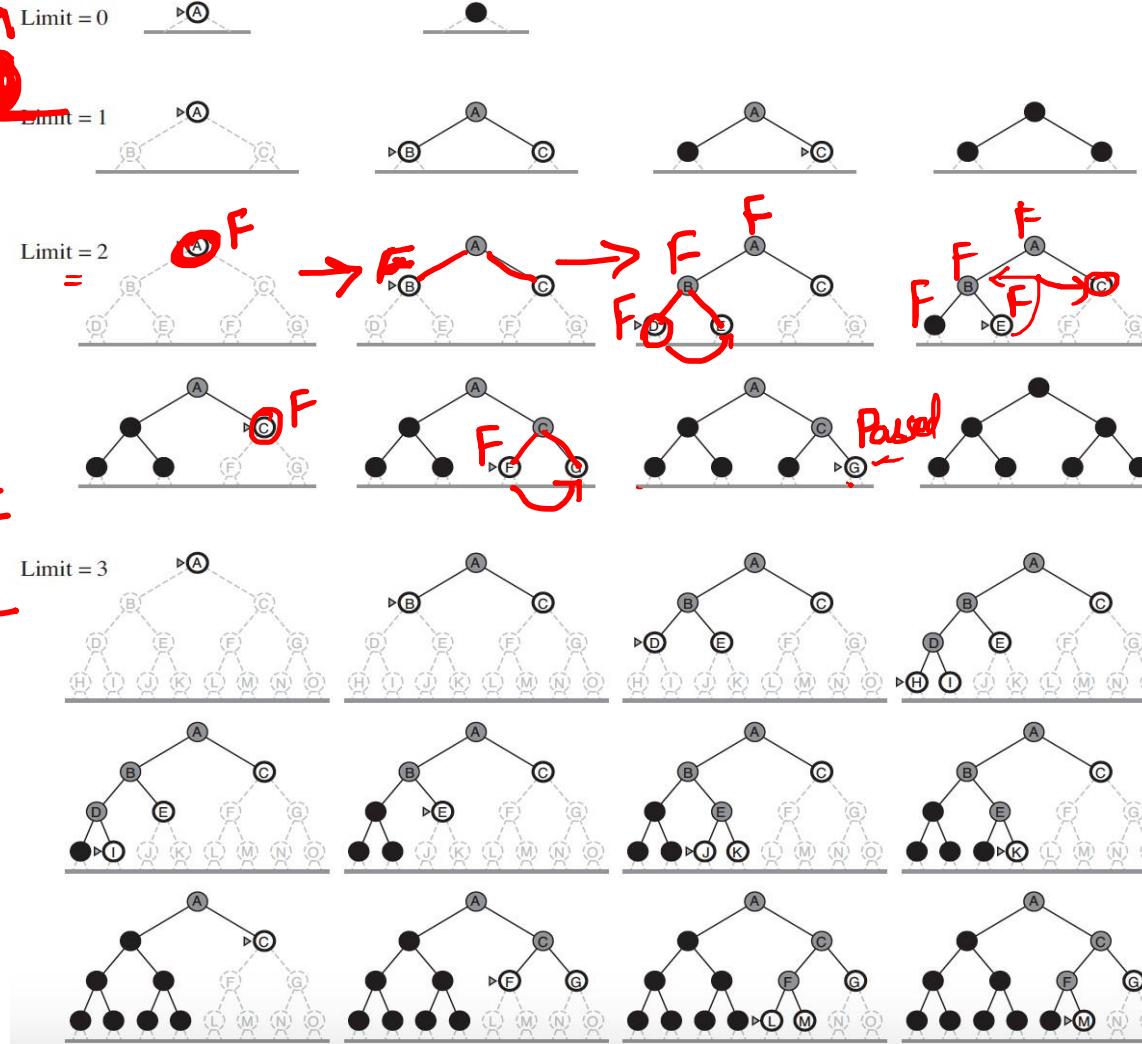
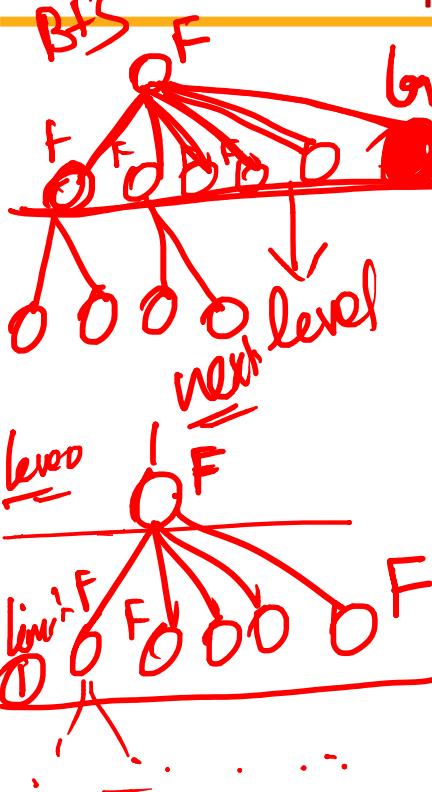
Too much regeneration



IDS

cost of redo ~  
complexity of  
BFS larger

## Iterative Deepening Depth First Search (IDS)



# Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)		Fail on (1)
2.			

## Sample Evaluation of the Algorithm -BFS

**Complete** – If the shallowest goal node is at a depth  $d$ , BFS will eventually find it by generating all shallower nodes

**Optimal** – Not necessarily. Optimal if path cost is non-decreasing function of depth of node.  
E.g., all actions have same cost

**Time Complexity** –  $G(b^d)$  b - branching factor, d – depth

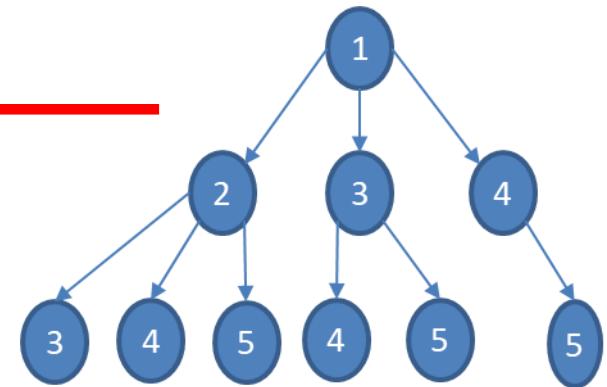
- Nodes expanded at depth 1 =  $b$
- Nodes expanded at depth 2 =  $b^2$
- Nodes expanded at depth  $d$  =  $b^d$
- Goal test is applied during generation, time complexity would be  $G(b^{d+1})$

**Space Complexity** –  $G(b^d)$

$\nearrow G(b^{d-1})$  in explored set

$\nearrow G(b^d)$  in frontier set

*closed list*  
*open list*



# Variation BFS



## Uniform Cost Search – Evaluation

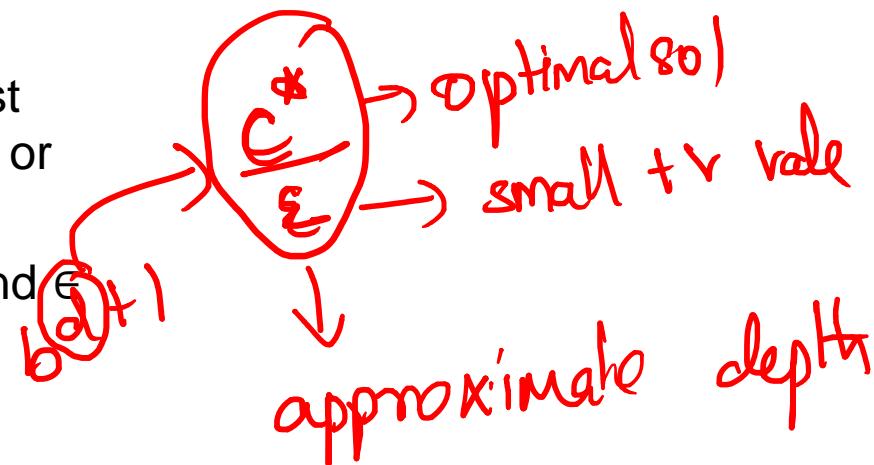
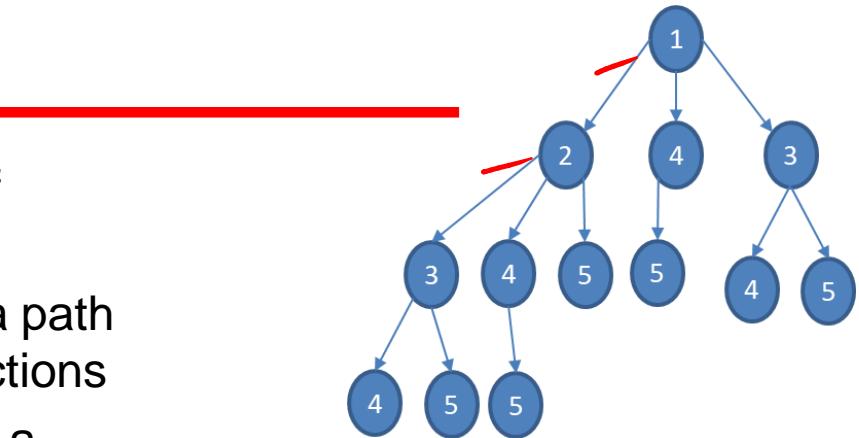
**Completeness** – It is complete if the cost of every step > small +ve constant  $\epsilon$

- It will stuck in infinite loop if there is a path with infinite sequence of zero cost actions

**Optimal** – It is Optimal. Whenever it selects a node, it is an optimal path to that node.

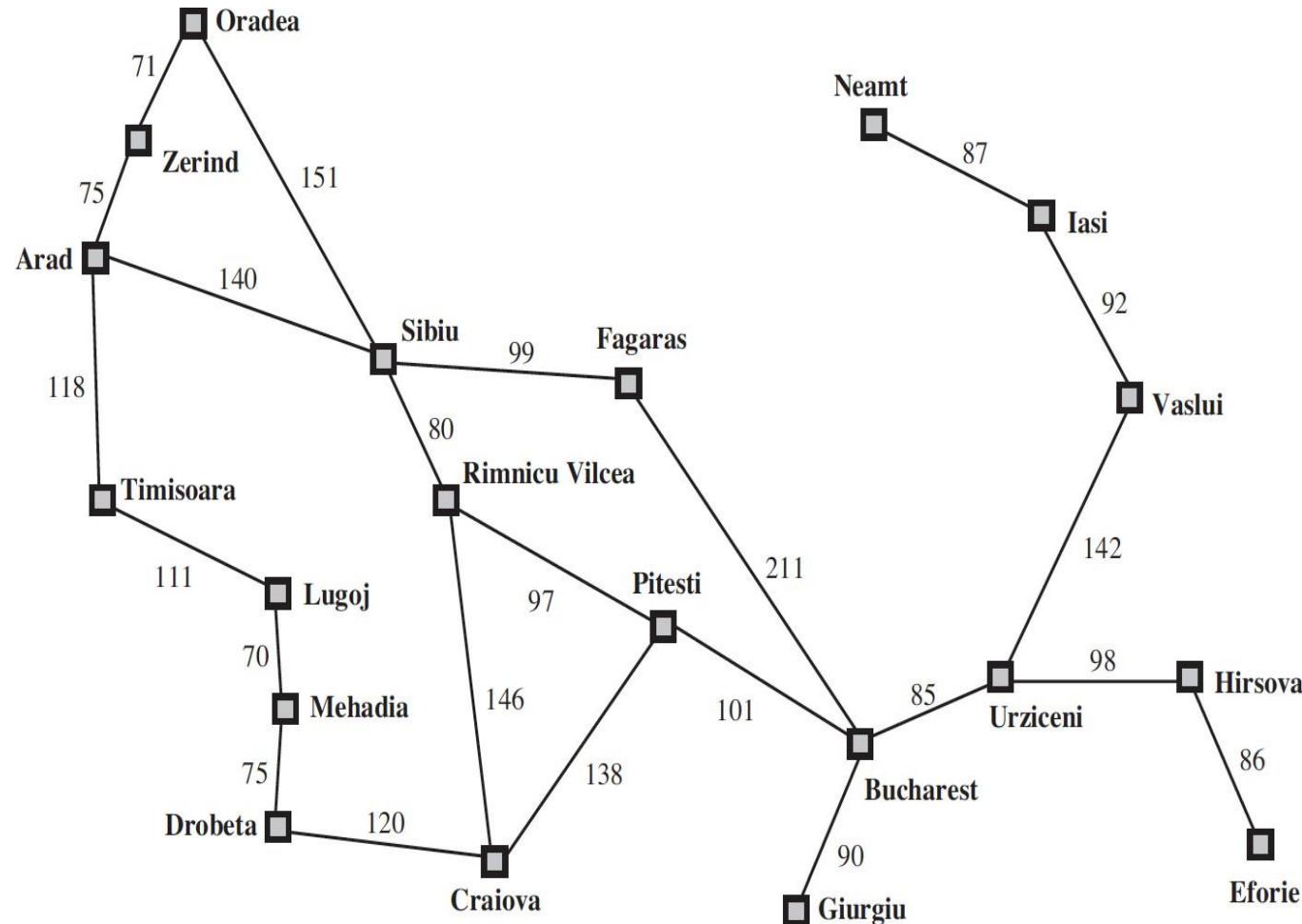
**Time and Space complexity** – Uniform cost search is guided by path costs not depth or branching factor.

- If  $C^*$  is the cost of optimal solution and  $\epsilon$  is the min. action cost
- Worst case complexity =  $G(b^{1+\frac{C^*}{\epsilon}})$ ,
- When all action costs are equal  $\rightarrow G(b^{d+1})$ , the BFS would perform better
  - As Goal test is applied during expansion, Uniform Cost search would do extra work



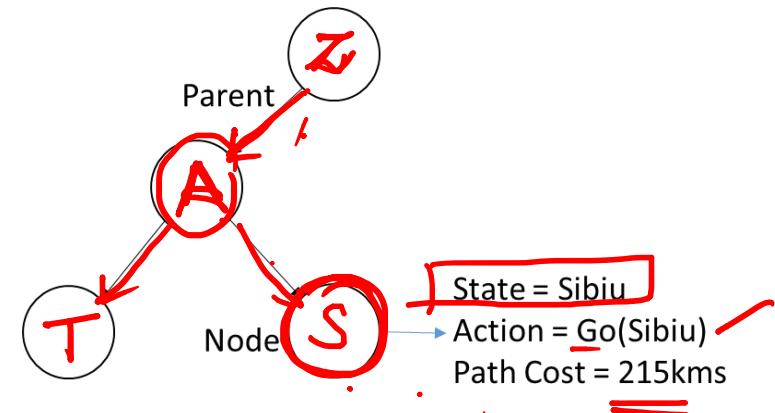
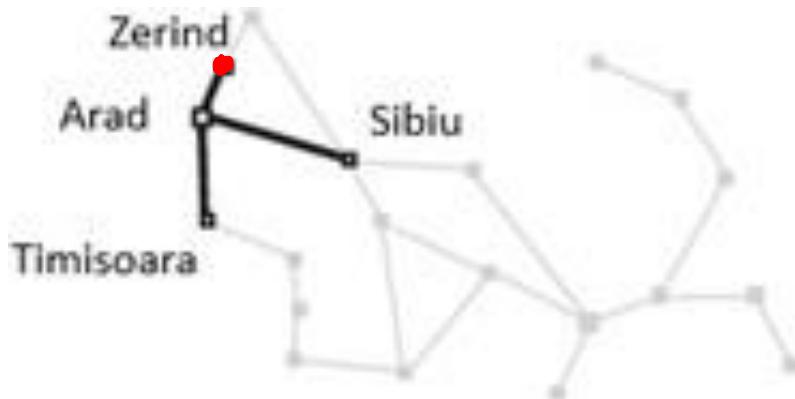


# Tree Search Vs Graph Search



For each node n of the tree,

- { **n.STATE** : the state in the state space to which node corresponds
- n.PARENT** : the node in the search tree that generated this node
- n.ACTION** : the action that was applied to parent to generate the node
- n.PATH-COST** : the cost, denoted by  $g(n)$ , of the path from initial state to node



# Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

Iter	Open List / Frontiers / Fringes	Goal Test
1.	(1)	Fail on (1)
2.	(1 3), (1 4), (1 2)	Fail on (1 3)

NO closed List

# Tree Search Algorithms

```
function Tree-Search (problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidate for expansion
            then return failure
        choose: leaf node for expansion according to strategy
        if the node contains a goal state
            then return the corresponding solution
        else
            Expand the node
            Add the resulting nodes to the search tree
    end
```

*front / open*

# Tree Search Vs Graph Search Algorithms

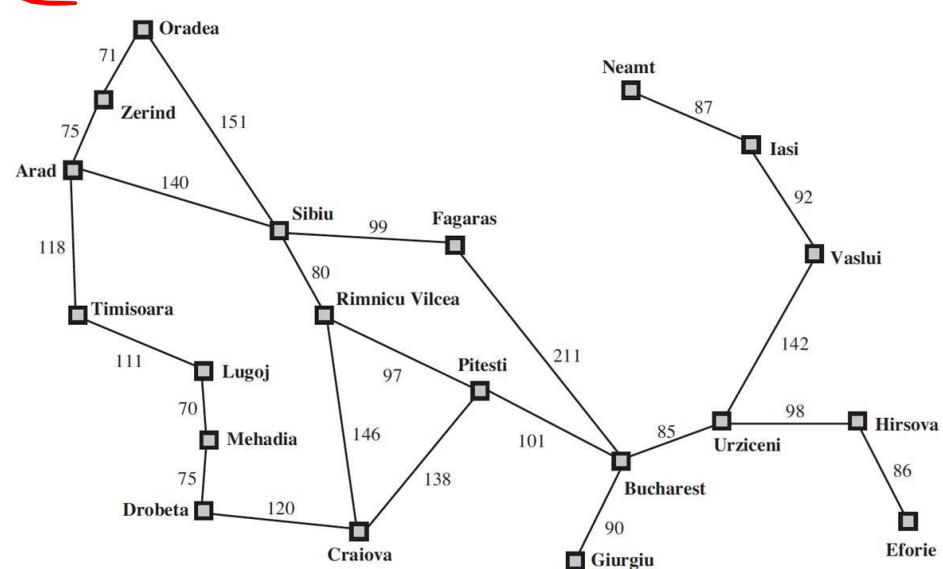
The logo consists of three interlocking gears in orange, blue, and red. The word "innovate" is written in white on the orange gear, "achieve" in white on the blue gear, and "lead" in white on the red gear.

# Coding Aspects

## Need:

**Redundant Path Problem** :More than one way to reach a state from another.

# Infinite Loop Path Problem



Start : Arad

## Goal : Craiova

# Tree Search Vs Graph Search Algorithms

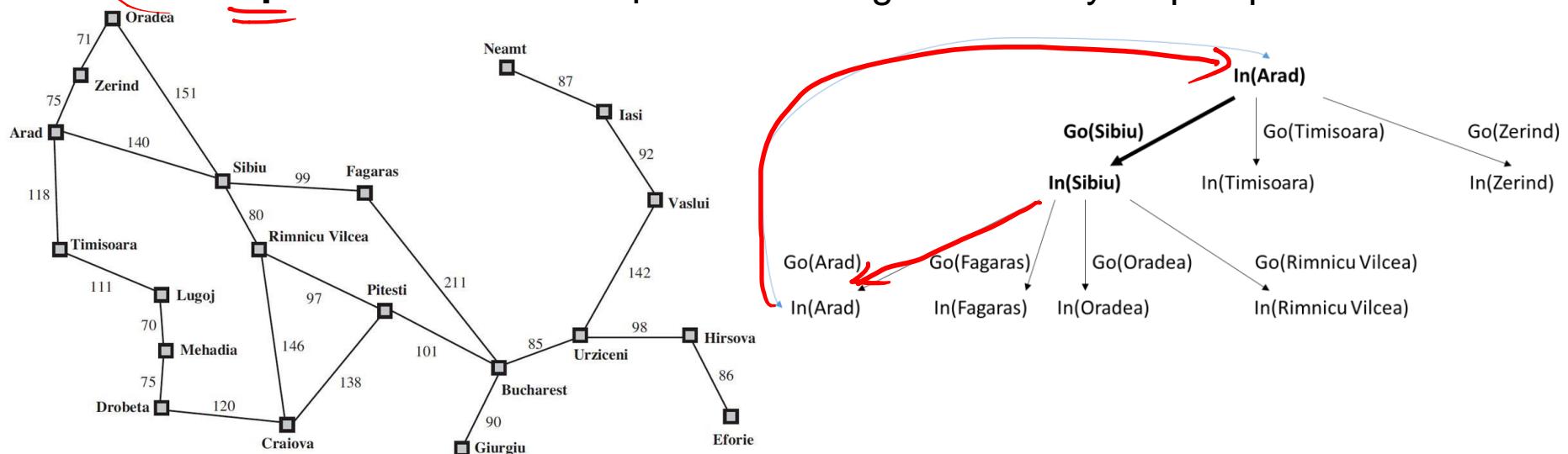


## Coding Aspects

Need:

Redundant Path Problem

**Infinite Loop Path Problem:** Repeated State generated by looped path existence.



Start : Arad

Goal : Craiova

# Algorithm Tracing

Students must follow this in the exams for all the search algorithms in addition to the search tree constructions. The ordering of the Open Lists must be in consistent with the algorithm with a note on the justification of the order expected!

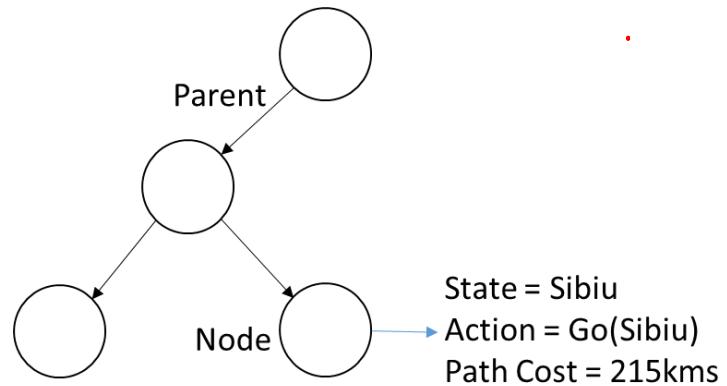
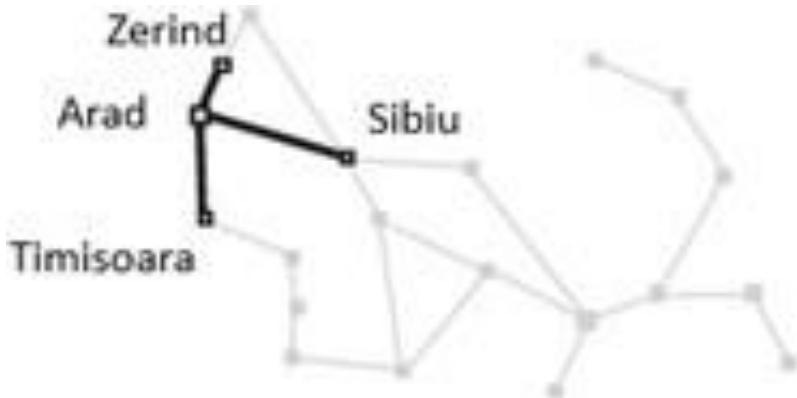


Iter	Open List / Frontiers / Fringes	Closed List	Goal Test
1.	(1)		Fail on (1)
2.	(1 3), (1 4), (1 2)	(1)	Fail on (1 3)

## Coding Aspects

For each node n of the tree,

- ~~n.STATE~~ : the state in the state space to which node corresponds
- ~~n.PARENT~~ : the node in the search tree that generated this node
- ~~n.ACTION~~ : the action that was applied to parent to generate the node
- ~~n.PATH-COST~~ : the cost, denoted by  $g(n)$ , of the path from initial state to node
- n.VISITED** : the boolean indicating if the node is already visited and tested (**or** global SET of visited nodes)  
a



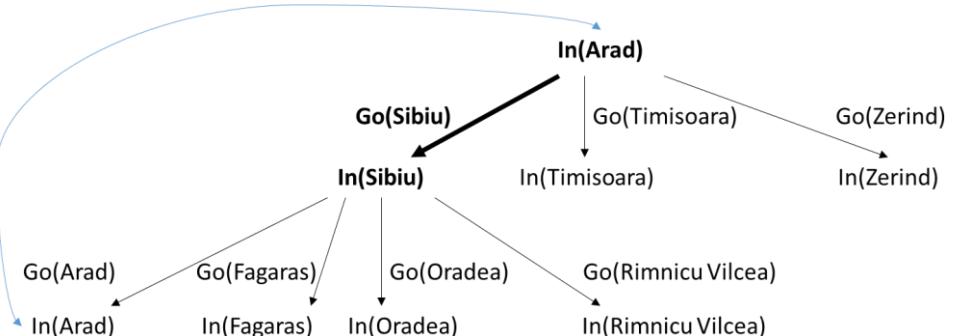
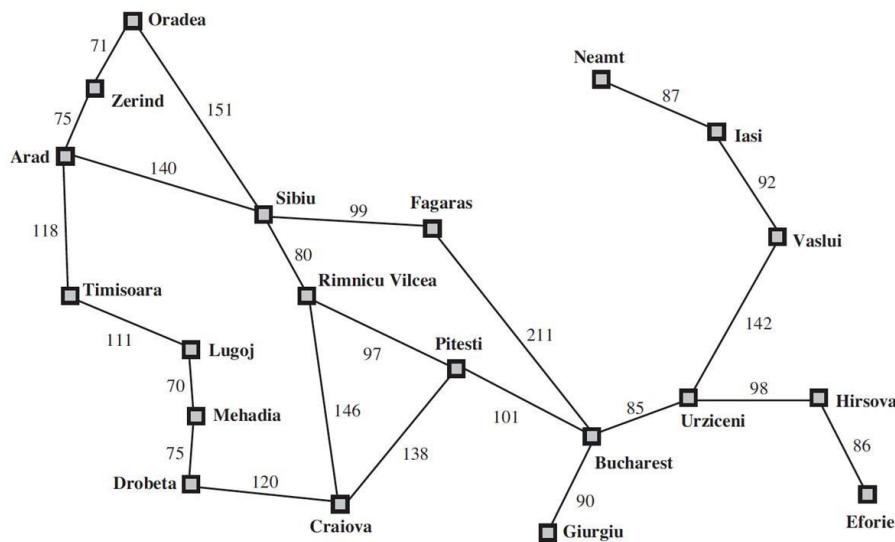
# Tree Search Vs Graph Search Algorithms



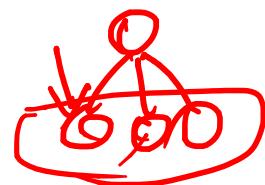
## Coding Aspects

### Graph-Search Algorithm

Augments the Tree-Search algorithm to solve redundancy by keeping track of states that are already visited called as **Explored Set**. Only one copy of each state is maintained/stored.



# Graph Search Algorithms



## Informed Search

~~Greedy Best First~~

~~A\*~~

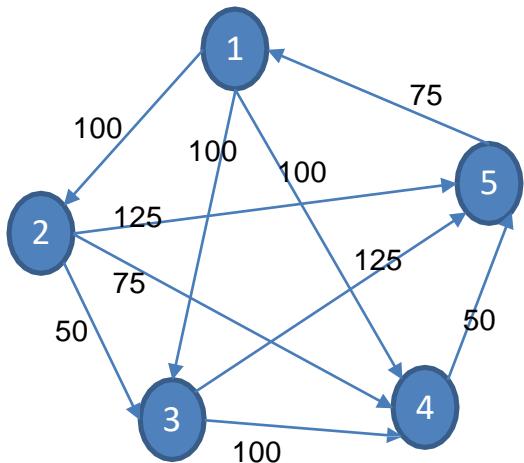


## Greedy Best First Search

① randomly

②

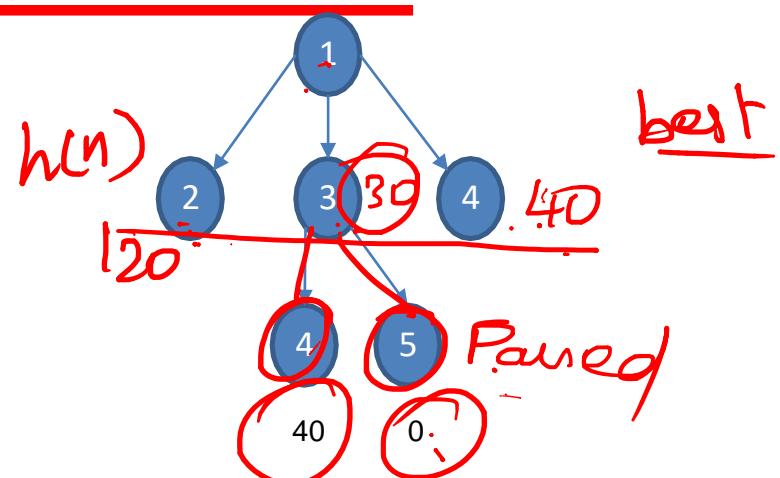
A B C D E F



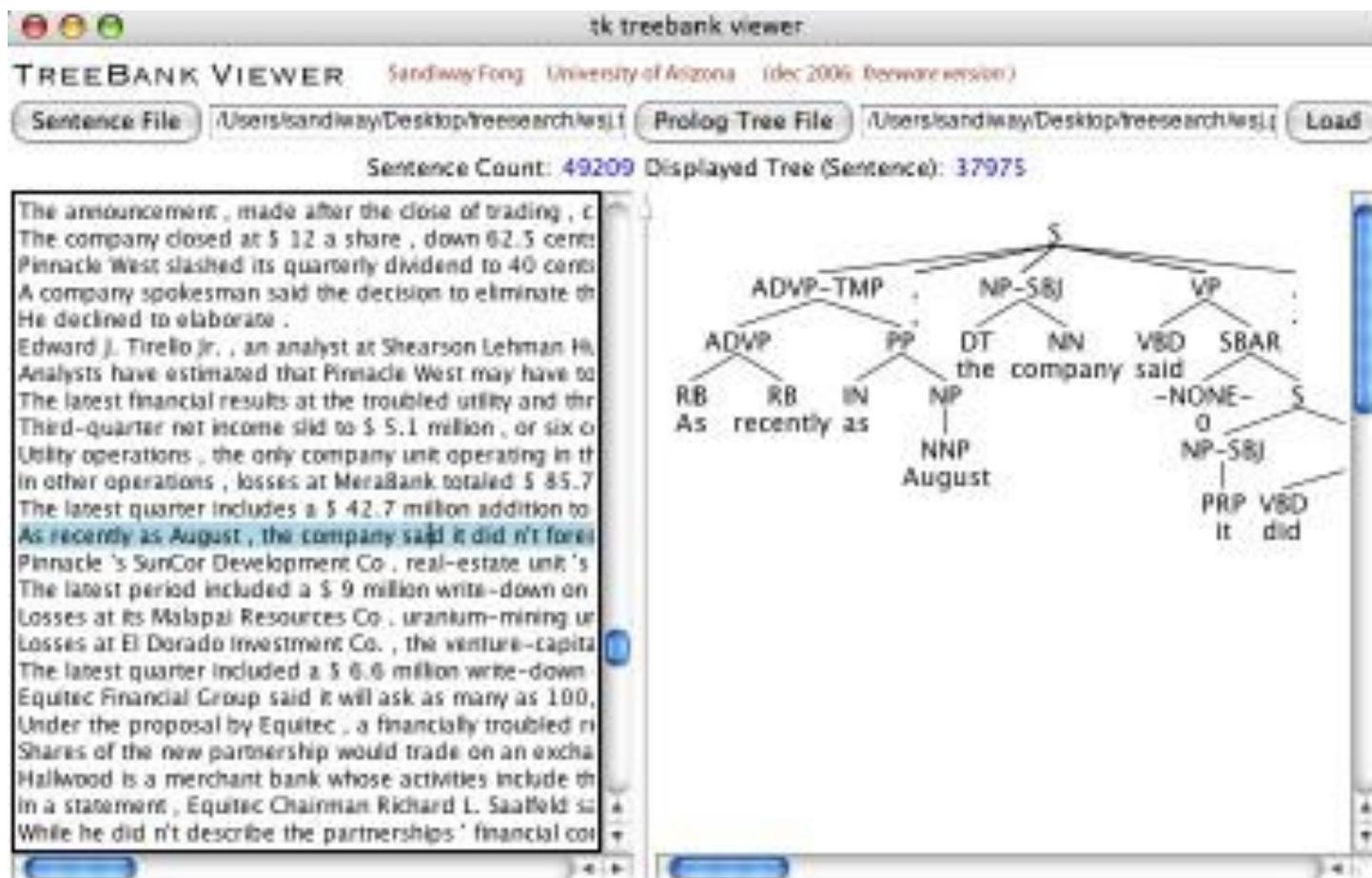
n	h(n)
1	60
2	120
3	30
4	40
5	0

(1)  
 (1 3) (1 4) (1 2)  
**(1 3 5)** (1 3 4)

$C(1-3-5) = 100 + 125 = 225$   
 Expanded : 2  
 Generated : 6  
 Max Queue Length : 3



# Case Study – 1 Search in Treebanks

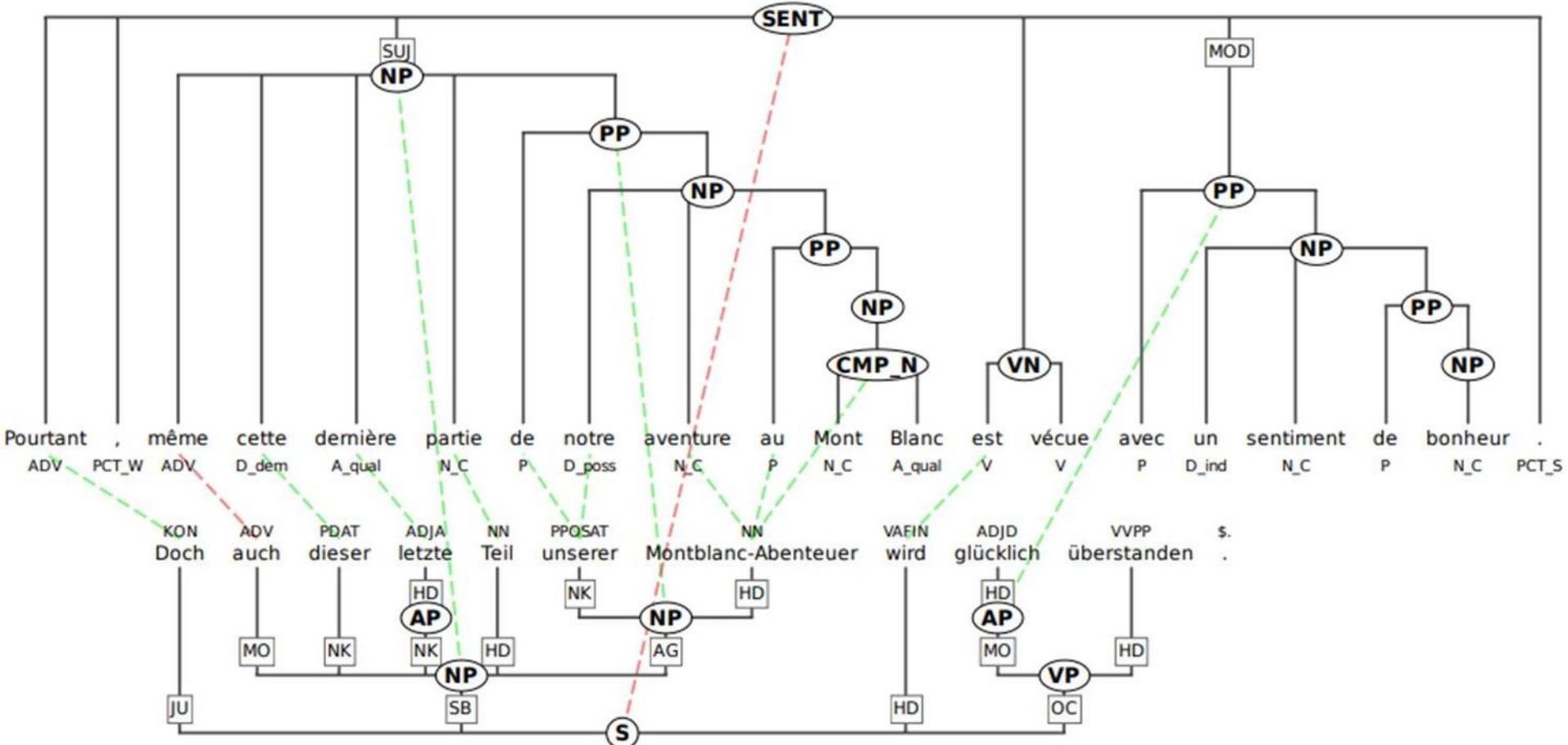


Source Credit :

<https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html>

<https://ufal.mff.cuni.cz/pdt3.5>

# Case Study – 1 Search in Treebanks

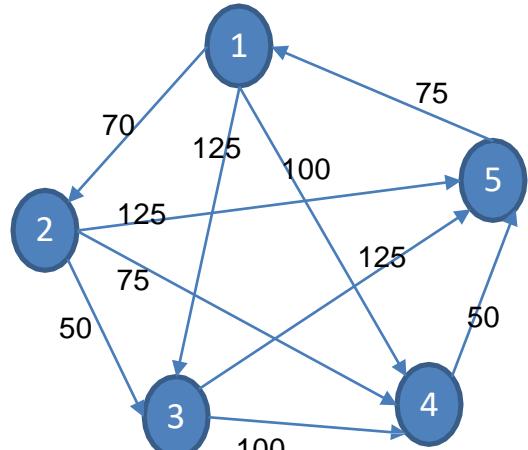


Source Credit :

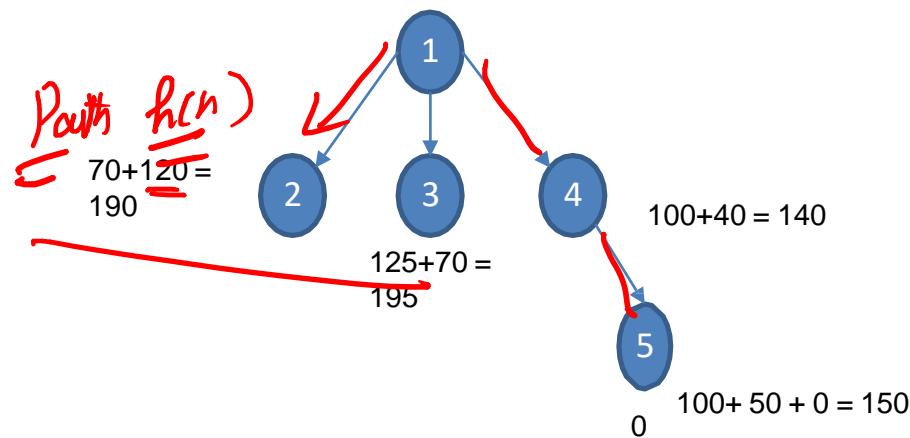
<https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html>

<https://ufal.mff.cuni.cz/pdt3.5>

# A\* Search



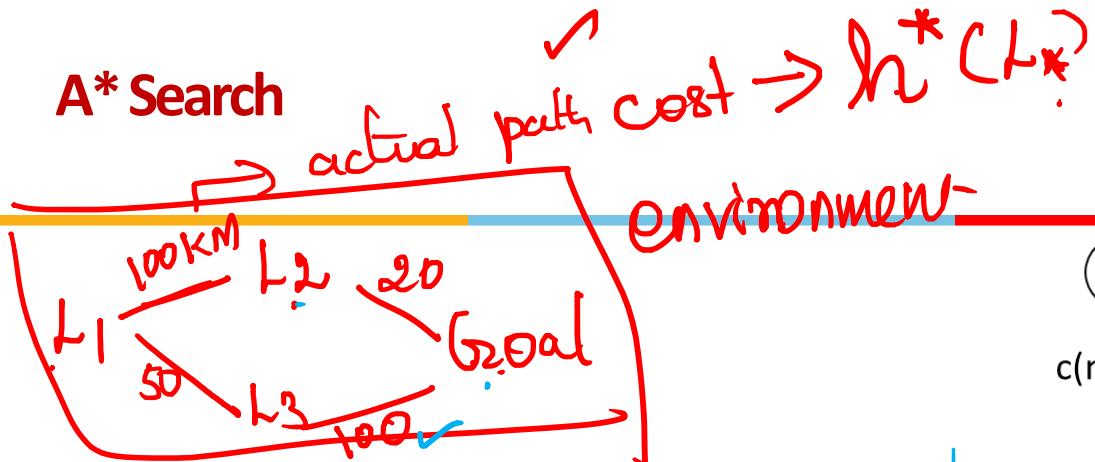
n	$h(n)$
1	60
2	120
3	70
4	40
5	0



(1)  
 (1 4) (1 2) (1 3)  
 (1 4 5) (1 2) (1 3)

$C(1-4-5) = 100 + 150 = 150$   
 Expanded : 2  
 Generated : 5  
 Max Queue Length : 3

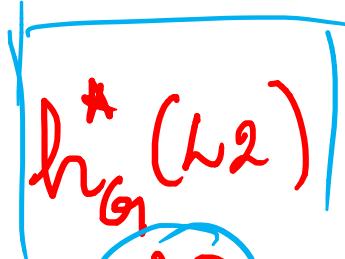
## A\* Search



D<sub>i</sub>:

<del>h(n)</del>	<del>L1</del>
50	L2
30	L3

more deviated from my environment  
less informed w.r.t goal



20

overestimate

More informed w.r.t goal

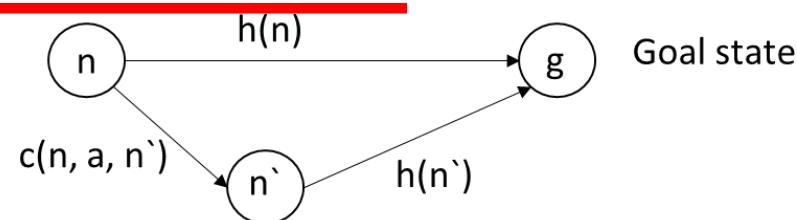
$h_{G_1}(l_3)$

30

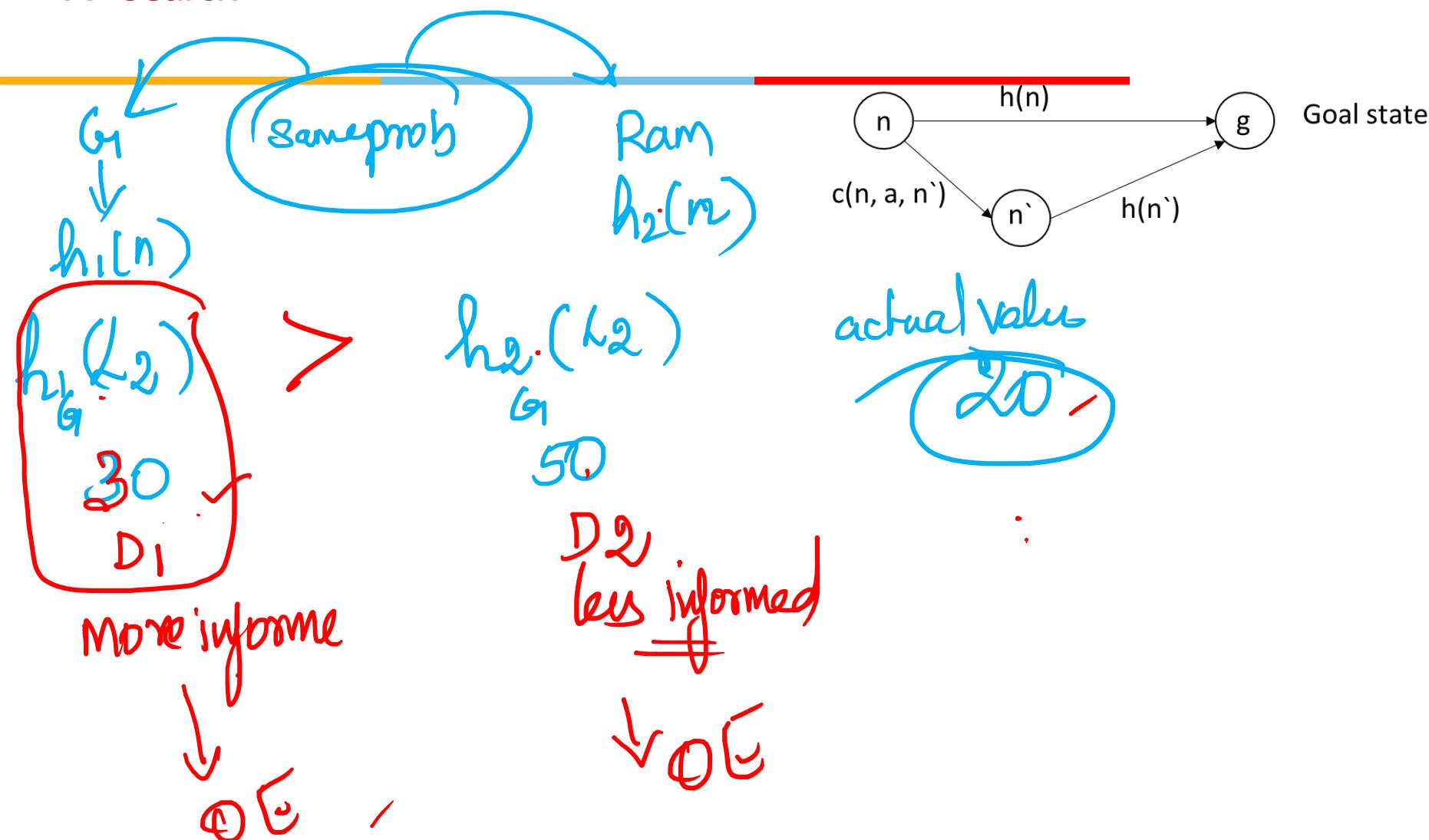
$h_{G_1}^*(l_3)$

100

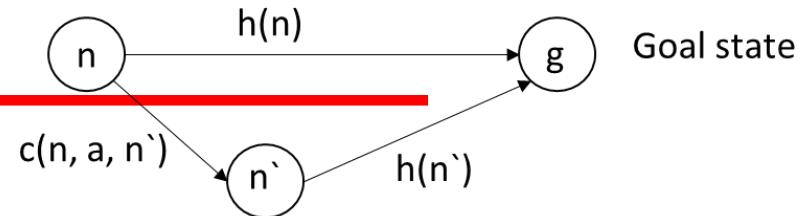
underestimate



# A\* Search



## A\* Search



### Optimal on condition

$h(n)$  must satisfy two conditions:

- Admissible Heuristic – one that never overestimates the cost to reach the goal
- Consistency – A heuristic is consistent if for every node  $n$  and every successor node  $n'$  of  $n$  generated by action  $a$ ,  $h(n) \leq c(n, a, n') + h(n')$

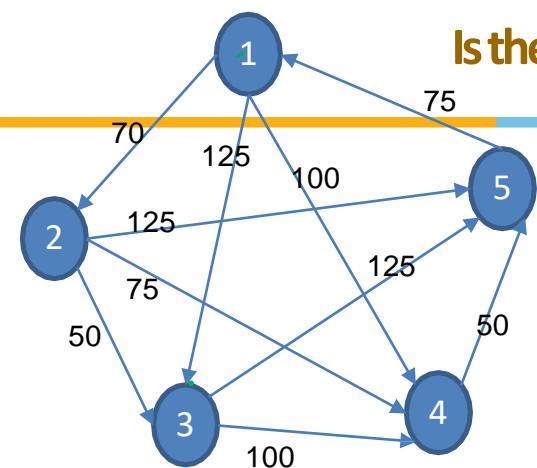
### Complete

- If the number of nodes with cost  $\leq C^*$  is finite
- If the branching factor is finite
- A\* expands no nodes with  $f(n) > C^*$ , known as pruning

Time Complexity -  $G(b^\Delta)$  where the absolute error  $\Delta = h^* - h$

# A\* Search

Is the heuristic designed leads to optimal solution? Not overestimate



Goal: 3

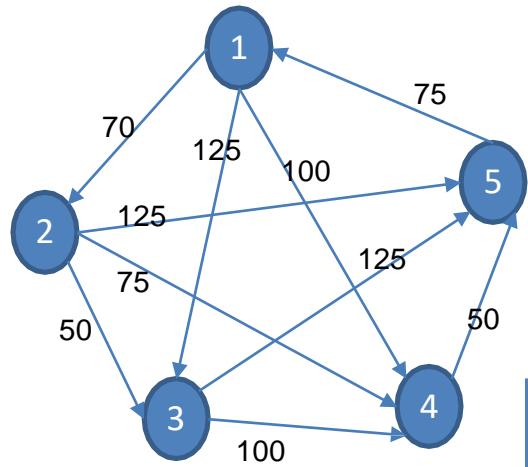
Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

$$(1-3) \Rightarrow P_1 \rightarrow 1-2-3 \Rightarrow 120 \\ P_2 \rightarrow 1-3 \Rightarrow 125$$

Node  
↓  
2-N

Cost	Best Path	n	$h(n)$	Is Admissible? $h(n) \leq h^*(n)$	Is Consistent? For every arc $(i,j)$ : $h(i) \leq g(i,j) + h(j)$
0	1-2-3	1	80	$80 \leq 120$ Yes	
50	2-3	2	60	$60 \leq 50$ ND	
120	0	3	0		
195	4-5-1-2-3	4	200	$200 \leq 245$ Yes	
245	5-1-2-3	5	190	$190 \leq 195$ Yes	

Is the heuristic designed leads to optimal solution?



$h(n)$

Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

n	$h(n)$	Is Admissible? $h(n) \leq h^*(n)$	Is Consistent? For every arc $(i,j)$ : $h(i) \leq g(i,j) + h(j)$
1	80	Y	N $(5 \rightarrow 1) : 190 \leq 155$
2	60 55 48	N	Y $(1 \rightarrow 2) : 80 \leq 130$
3	0	Y	
4	200	Y	Y $(1 \rightarrow 4) : 80 \leq 300$ Y $(2 \rightarrow 4) : 60 \leq 275$
5	190	Y	Y $(2 \rightarrow 5) : 60 \leq 315$ Y $(4 \rightarrow 5) : 200 \leq 240$

# Path finding Robot – Sample Planning Agent Design

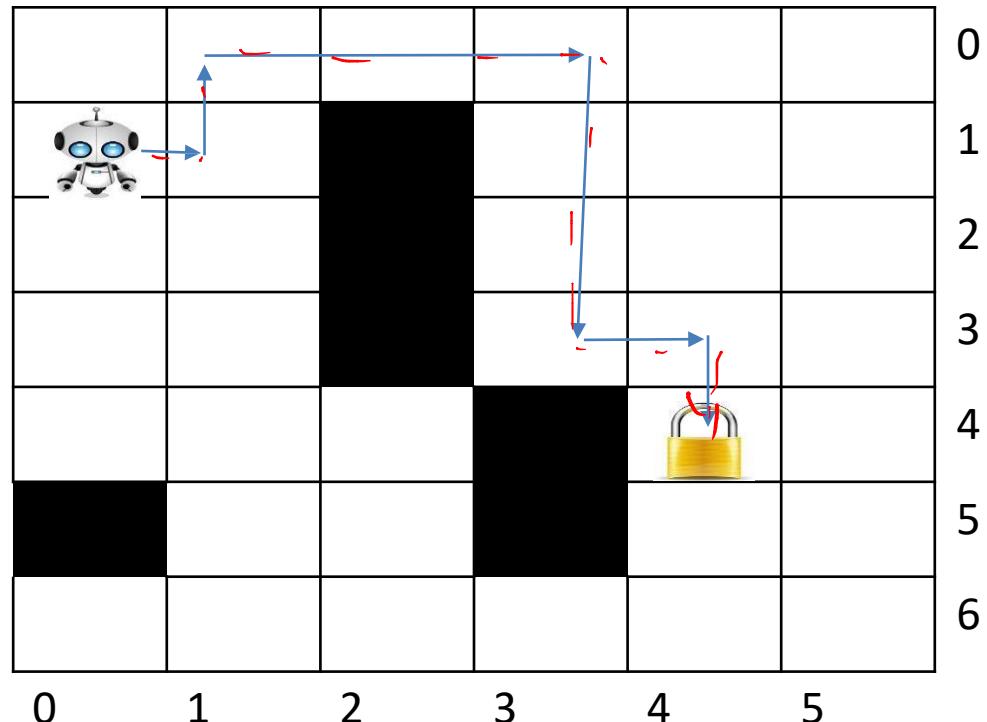
## Successor Function Design

1	2	3	4	5	6	
8			10	11	12	0
13	14		16	17	18	1
19	20		22	23	24	2
25	26	27		30		3
	32	33		35	36	4
37	38	39	40	41	42	5
0	1	2	3	4	5	6

N-W-E-S

A\*  
—

## Demo



---

**Required Reading:** AIMA - Chapter #1, 2, 3.1

Note : Some of the slides are adopted from AIMA TB materials

Thank You for all your Attention



# Artificial & Computational Intelligence

**DSECLZG557**

**M2 : Problem Solving Agent using Search**

**BITS** Pilani  
Pilani Campus

Indumathi V  
Guest Faculty  
BITS -WILP

# Course Plan

- M1 Introduction to AI
- M2 Problem Solving Agent using Search
- M3 Game Playing
- M4 Knowledge Representation using Logics
- M5 Probabilistic Representation and Reasoning
- M6 Reasoning over time
- M7 Ethics in AI

## Module 2 : Problem Solving Agent using Search

- A. Uninformed Search
- B. Informed Search
- C. Heuristic Functions
- D. Local Search Algorithms & Optimization Problems

## Learning Objective

---

At the end of this class , students Should be able to:

1. Differentiate between uninformed and informed search requirements
2. Apply UCS, GBFS & A\* algorithms to the given problem
3. Prove if the given heuristics are admissible and consistent
4. Design and compare heuristics apt for given problem
5. Apply A\* variations algorithms to the given problem



Optimal on condition

$h(n)$  must satisfies two conditions:

↳ Admissible Heuristic – one that never overestimates the cost to reach the goal

↳ Consistency – A heuristic is consistent if for every node  $n$  and every successor node  $n'$  of  $n$  generated by action  $a$ ,  $h(n) \leq c(n, a, n') + h(n')$

*good design*

Complete

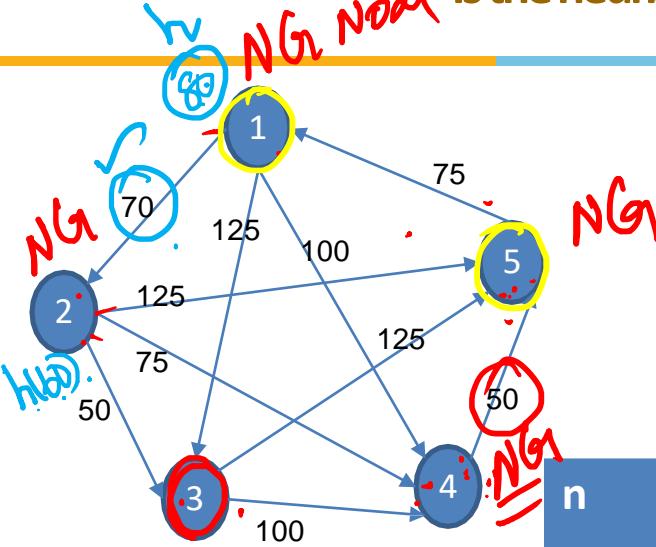
- If the number of nodes with cost  $\leq C^*$  is finite
- If the branching factor is finite
- A\* expands no nodes with  $f(n) > C^*$ , known as pruning

Time Complexity -  $G(b^\Delta)$  where the absolute error  $\Delta = h^* - h$

A\* Search  
 $80 - 60 \Rightarrow 20$   
*accept*

NG Node

Is the heuristic designed leads to optimal solution?



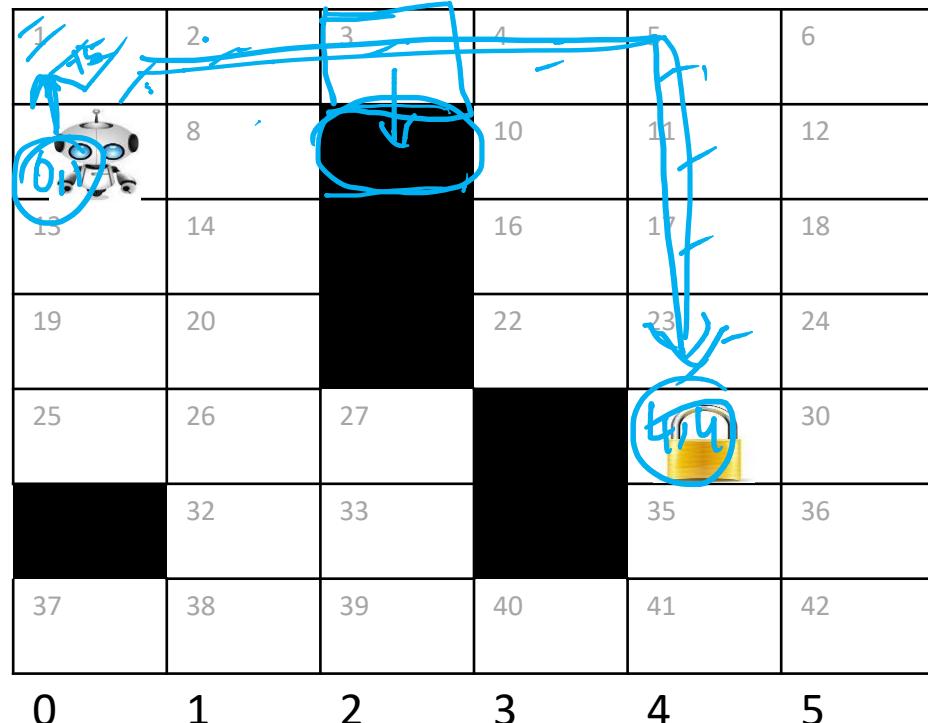
Assuming node 3 as goal, taking only sample edges per node below is checked for consistency

optimal path cost = best Path

n	$h(n)$	Is Admissible? $h(n) \leq h^*(n)$	Is Consistent? For every arc $(i,j)$ : $h(i) \leq g(i,j) + h(j)$
1	80	Y	$80 \leq 120$
2	60	N	$Y$ $(1 \rightarrow 2) : 80 \leq 130$
3	0	Y	
4	200	Y	$Y$ $(1 \rightarrow 4) : 80 \leq 300$ $Y$ $(2 \rightarrow 4) : 60 \leq 275$
5	190	Y	$Y$ $(2 \rightarrow 5) : 60 \leq 315$ $Y$ $(4 \rightarrow 5) : 200 \leq 240$

# Path finding Robot – Sample Planning Agent Design

## Successor Function Design



N-W-E-S

$g(n)$ ) every transition +5

(1) Near by unsafe cell -3

(2) +5.

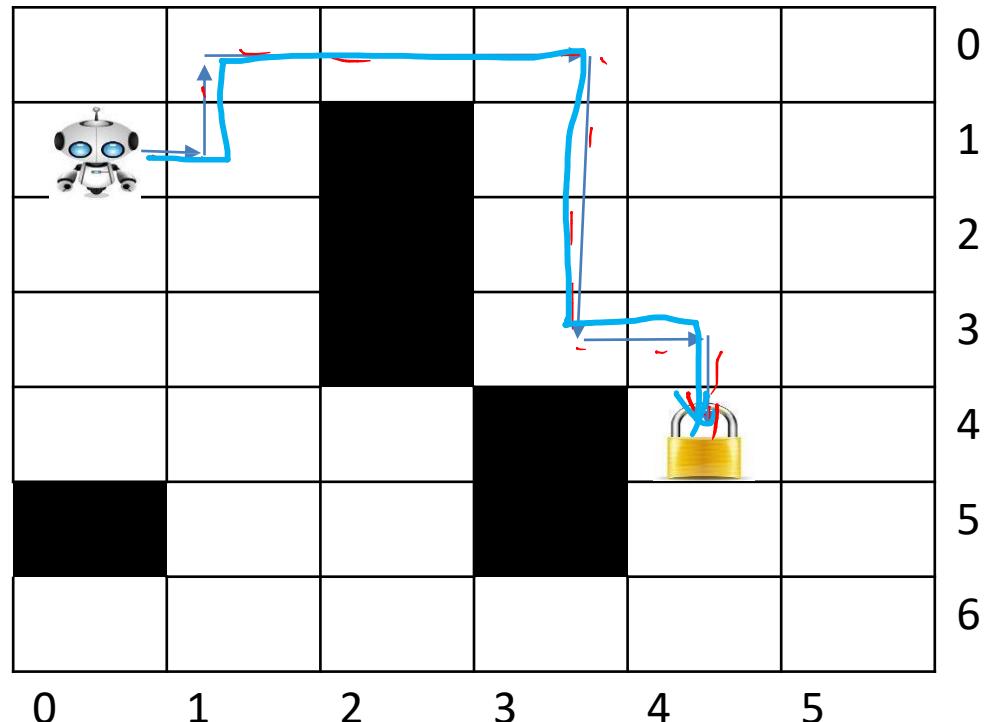
$h(n)$ ) → Euclidean distance  
→ Manhattan dist

$8 \times 5 \Rightarrow 40 - 3 = 37$

+ h

A\*  
—

## Demo



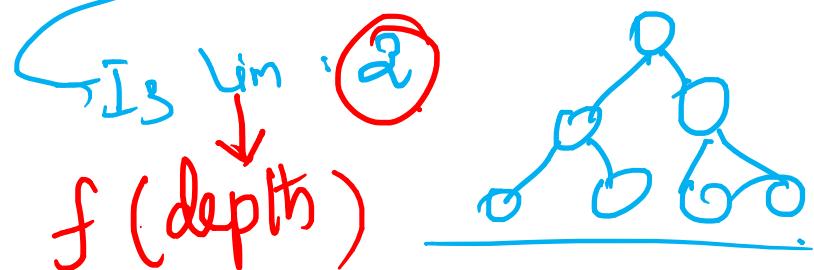
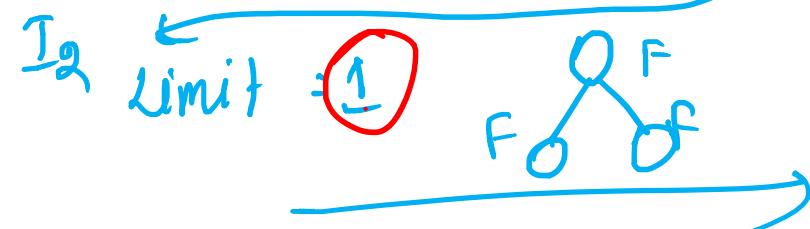
## Variations of A\*

Memory Bounded Heuristics

A\*

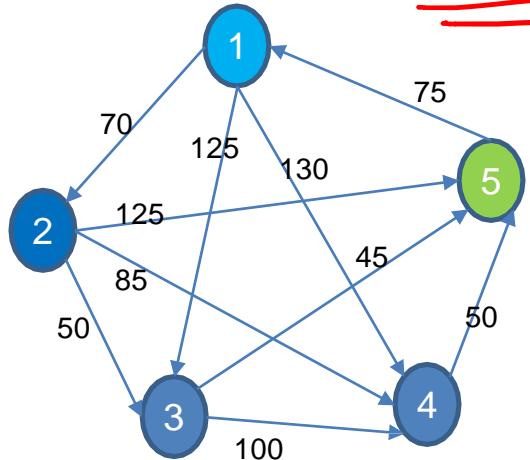
IDA\*  
RBFS

BFS + DFS  $\Rightarrow$  IDS



# Iterative Deepening A\*

$\text{DFS} + \text{A}^* \Rightarrow \text{IDA}^*$

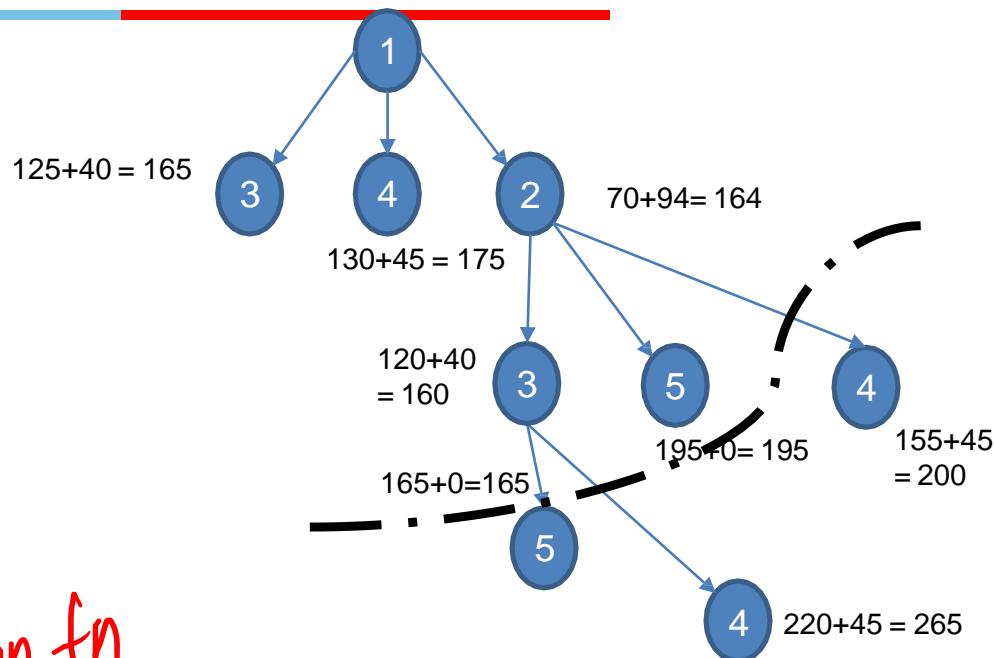


n	$h(n)$
1	60
2	94
3	40
4	45
5	0

limit :  $f(f(n))$   
 ↓  
 evaluation fn  
 ↓  
 $f(n) = h(n) + g(n)$

$B = \boxed{\quad}$

Set limit for  $f(n)$



Cut off value is the smallest of f-cost of any node that exceeds the cutoff on previous iterations

Iterative Limit : Eg

$$f(n) = 180$$

$$f(n) = 195$$

$$f(n) = 200$$

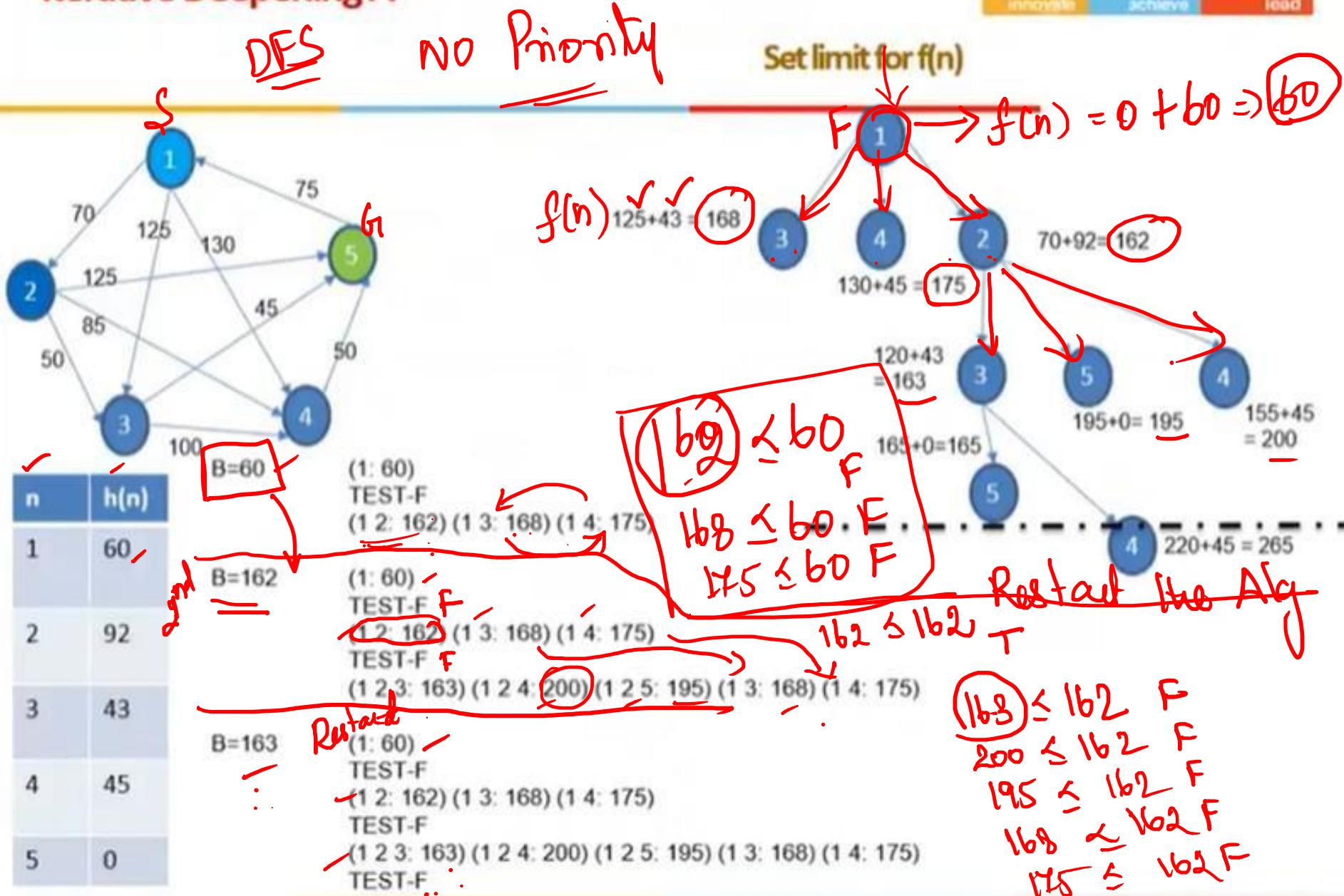
.

.

.

.

# Iterative Deepening A\*



## Iterative Deepening A\*

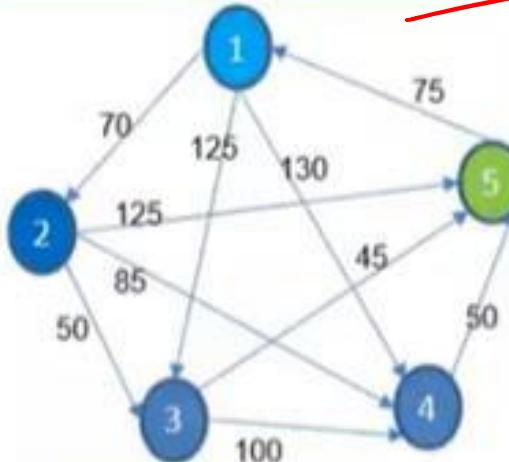
Innovate

Achieve

Lead

$A^*$  + DFS

Last gone



n	$h(n)$
1	60
2	92
3	43
4	45
5	0

B=163

(1: 60)  
TEST-F  
(1 2: 162) (1 3: 168) (1 4: 175)  
TEST-F  
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)  
TEST-F  
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

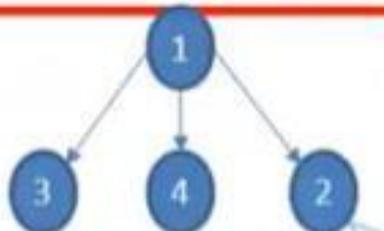
B=165

(1: 60)  
TEST-F  
(1 2: 162) (1 3: 168) (1 4: 175)  
TEST-F  
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)  
TEST-F  
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

Passed  
(1 2 3 5: 165)

Set limit for  $f(n)$

$$125+43 = 168$$



$$130+45 = 175$$

$$120+43 = 163$$

$$165+0=165$$

$$70+92=162$$

$$195+0=195$$

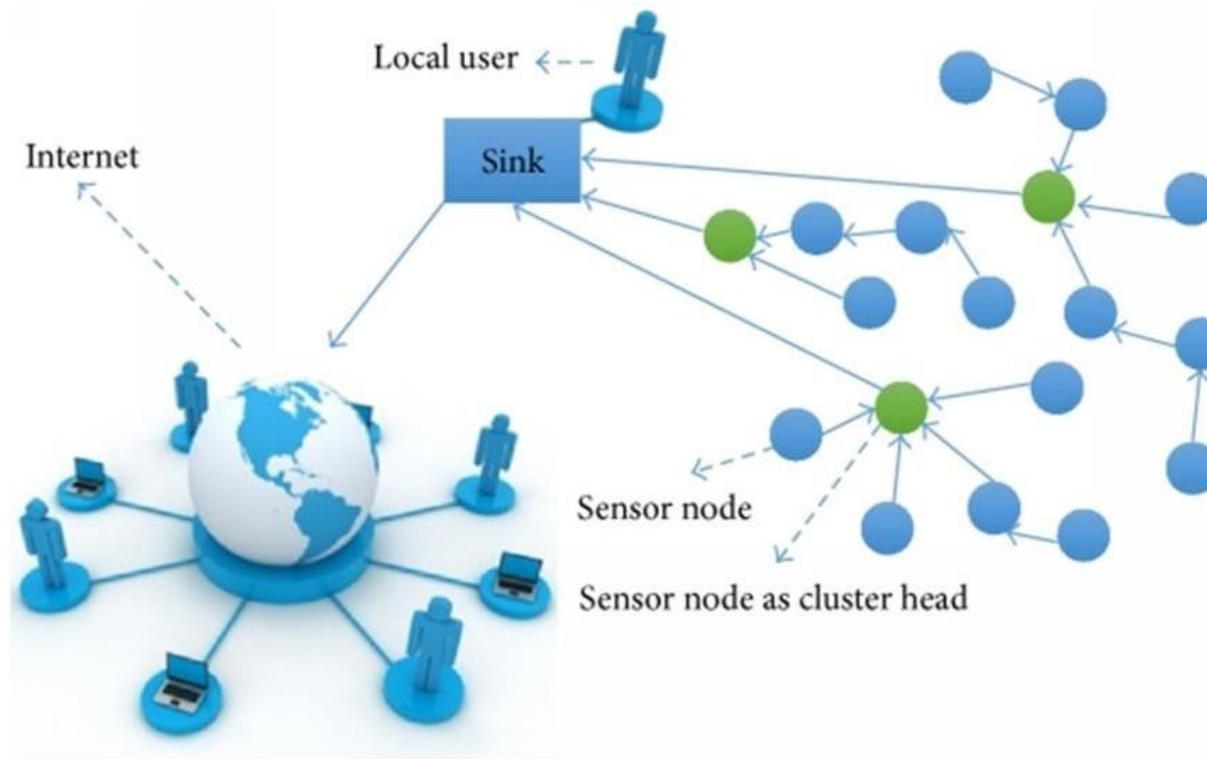
$$155+45 = 200$$

$165 \leq 165$

1235 : 165

# Case Study – Search in Network Routing

---



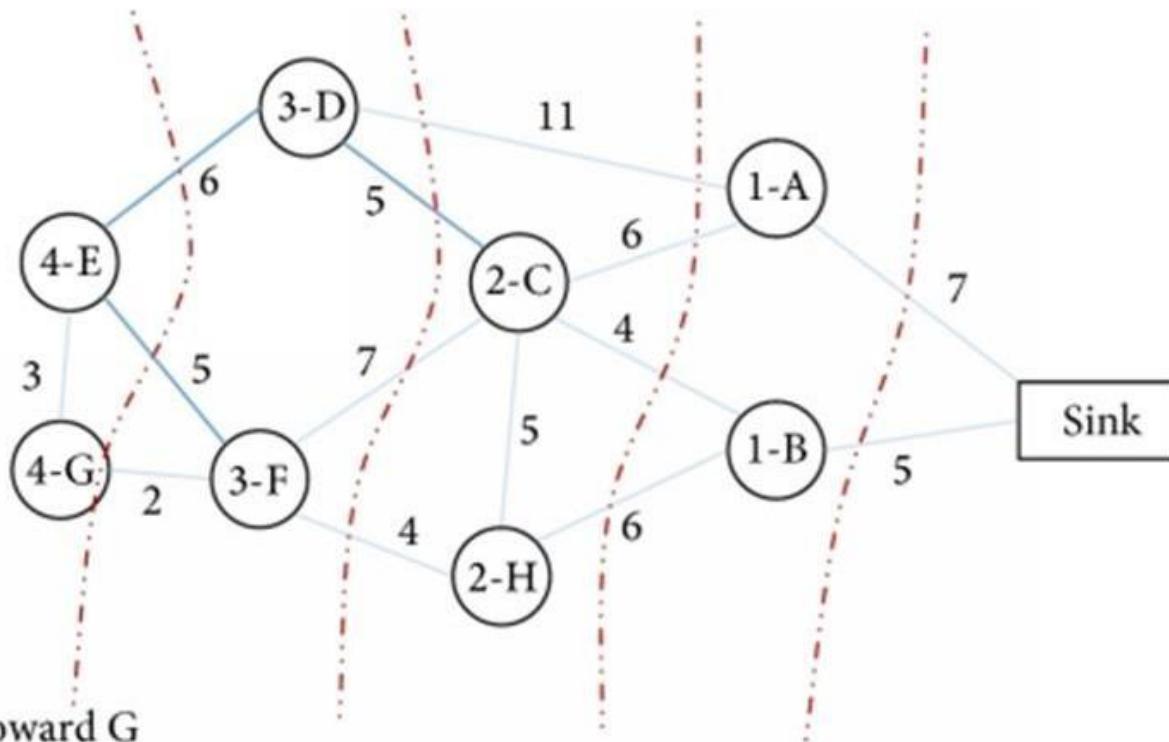
Source Credit :

AR-RBFS: Aware-Routing Protocol Based on Recursive Best-First Search Algorithm for Wireless Sensor Networks

<https://doi.org/10.1155/2016/8743927>

# Case Study – Search in Network Routing

A	14.1
B	11.3
C	8.2
H	6.6
F	2
E	3
D	4.8



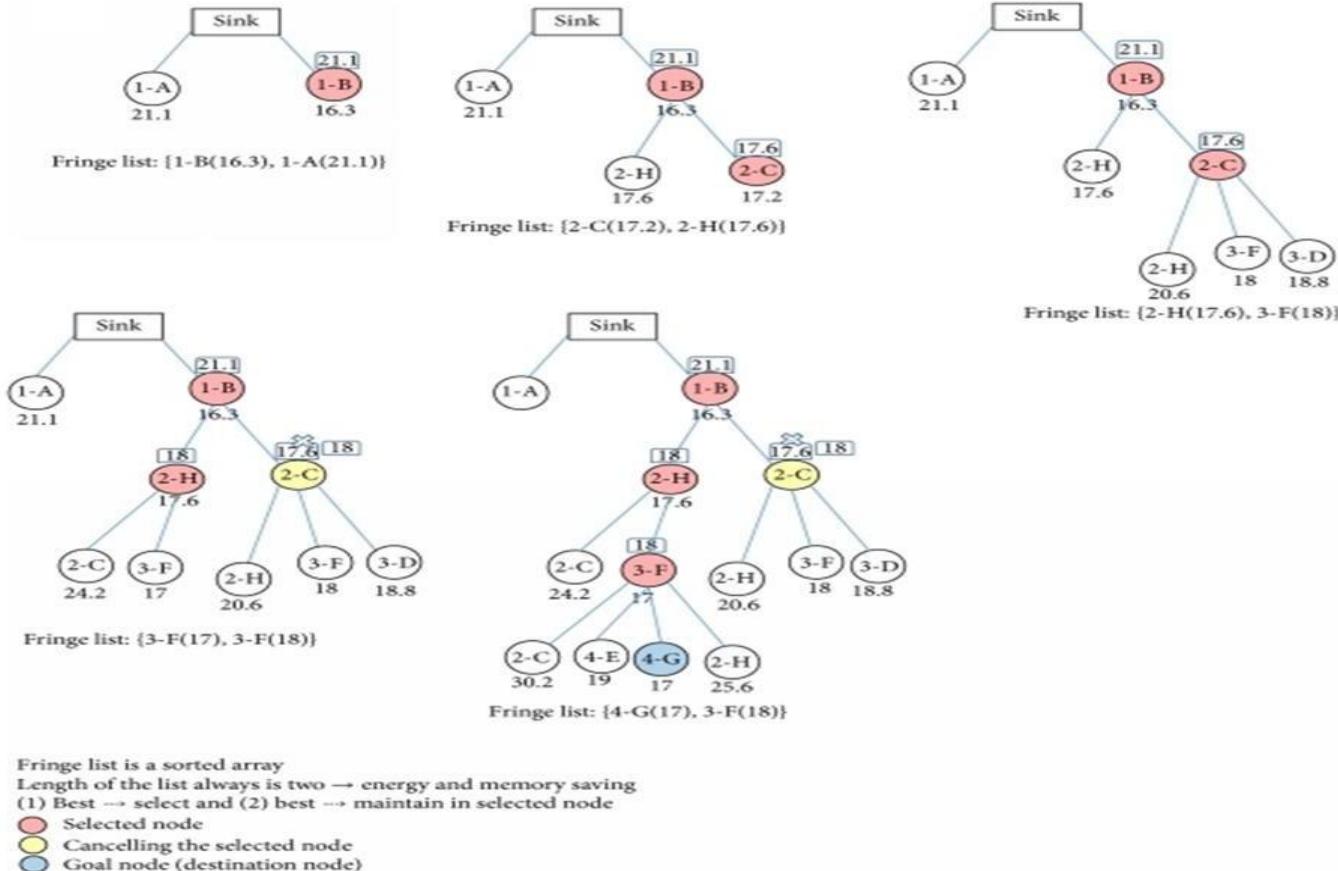
Heuristic values toward G

Source Credit :

AR-RBFS: Aware-Routing Protocol Based on Recursive Best-First Search Algorithm for Wireless Sensor Networks

<https://doi.org/10.1155/2016/8743927>

# Case Study – Search in Network Routing



Source Credit :

AR-RBFS: Aware-Routing Protocol Based on Recursive Best-First Search Algorithm for Wireless Sensor Networks

<https://doi.org/10.1155/2016/8743927>

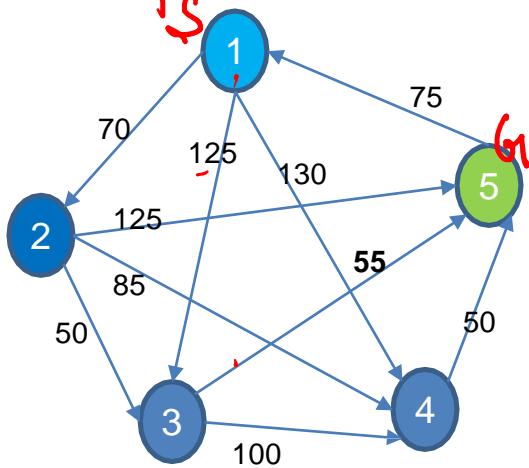
# Recursive Best First Search A\*

innovate

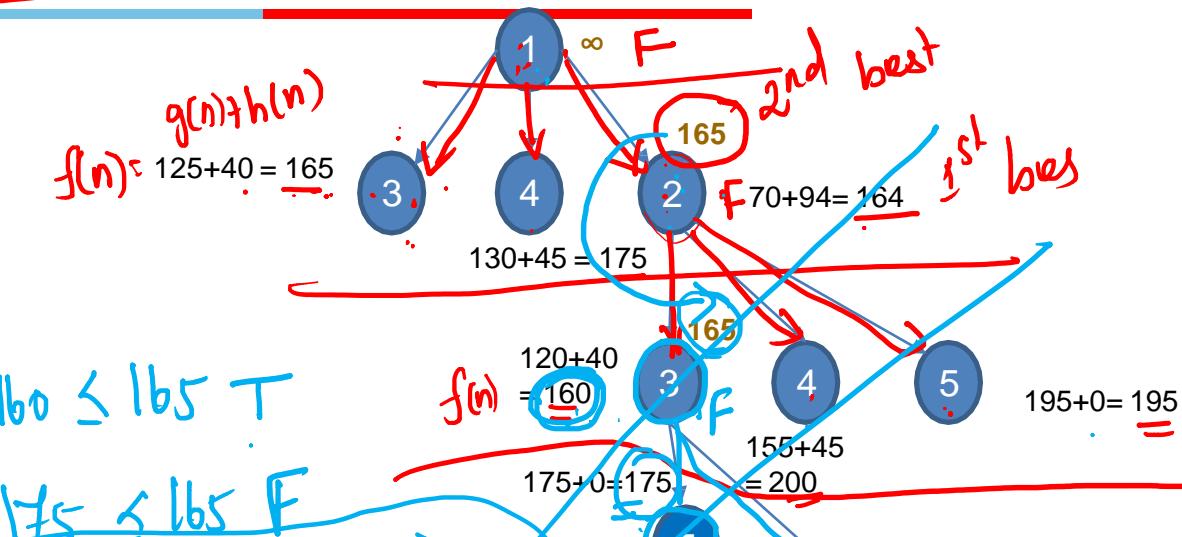
achieve

lead

Remember the next best alternative f-Cost to regenerate  
Priority Queue



n	h(n)
1	60
2	94
3	40
4	45
5	0



$(123 : 160) (124 : 200) (125 : 195) (13 : 165)$   
 $(14 : 175)$

$(123 | 160) | 13 | 165$

$F \checkmark (12 | 164) (13 | 165) (14 | 175)$

$(12 | 175) (14 | 175) (13 | 180)$

$(123 | 175) (14 | 175) (13 | 180) (125 | 195) (124 | 200)$

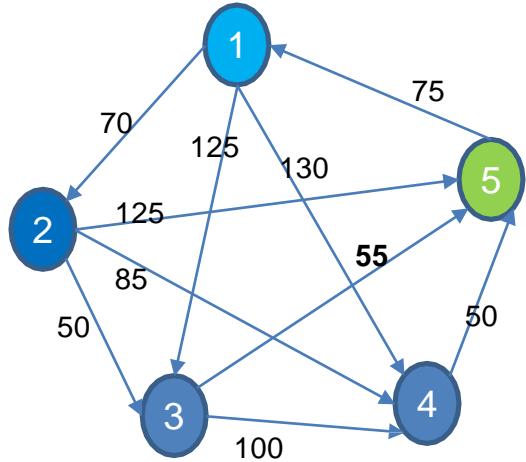
$(1235 | 175) (14 | 175) (13 | 180) (125 | 195) (124 | 200) (1234 | 267)$

PASS

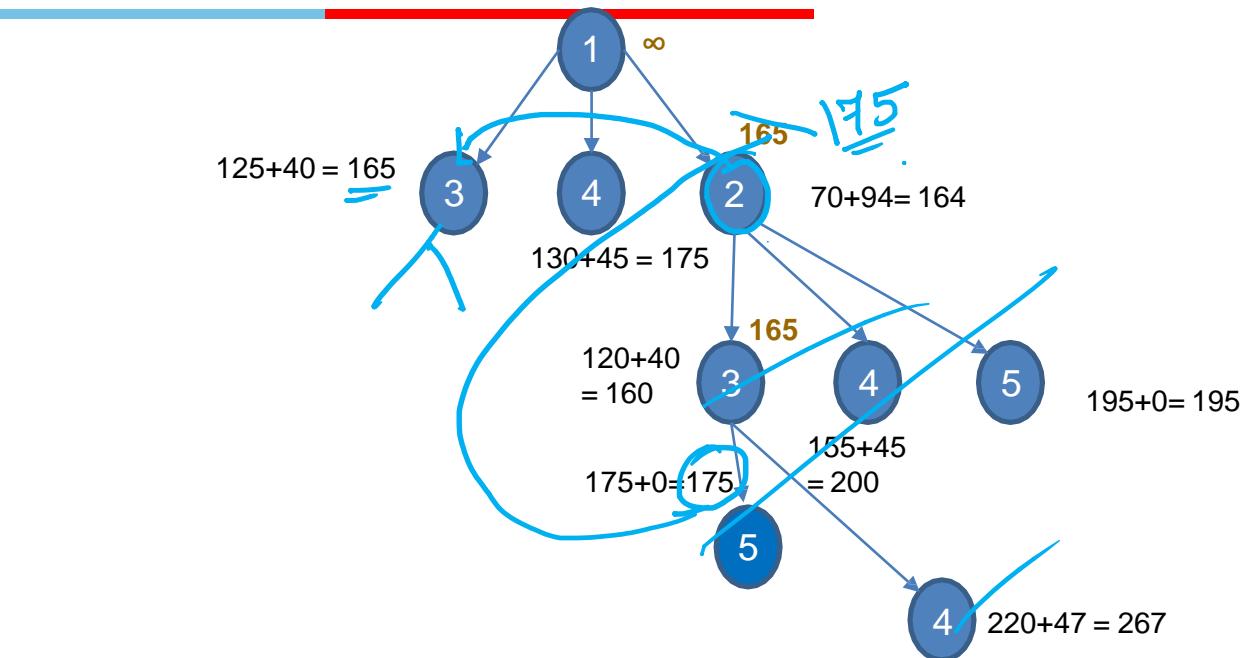
# Recursive Best First Search A\*



Remember the next best alternative f-Cost to regenerate



n	$h(n)$
1	60
2	94
3	40
4	45
5	0



(1, 60)  
 $(1 \ 2 \mid 164) \ (1 \ 3 \mid 165) \ (1 \ 4 \mid 175)$

$(1 \ 2 \mid 175) \ (1 \ 4 \mid 175) \ (1 \ 3 \mid 180)$

$(1 \ 2 \ 3 \mid 175) \ (1 \ 4 \mid 175) \ (1 \ 3 \mid 180) \ (1 \ 2 \ 5 \mid 195) \ (1 \ 2 \ 4 \mid 200)$

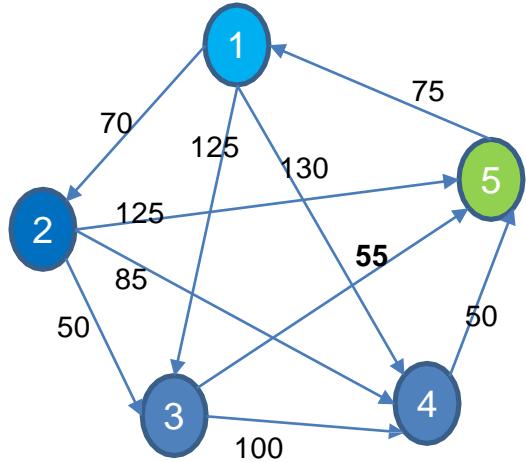
$(1 \ 2 \ 3 \ 5 \mid 175) \ (1 \ 4 \mid 175) \ (1 \ 3 \mid 180) \ (1 \ 2 \ 5 \mid 195) \ (1 \ 2 \ 4 \mid 200) \ (1 \ 2 \ 3 \ 4 \mid 267)$

PASS

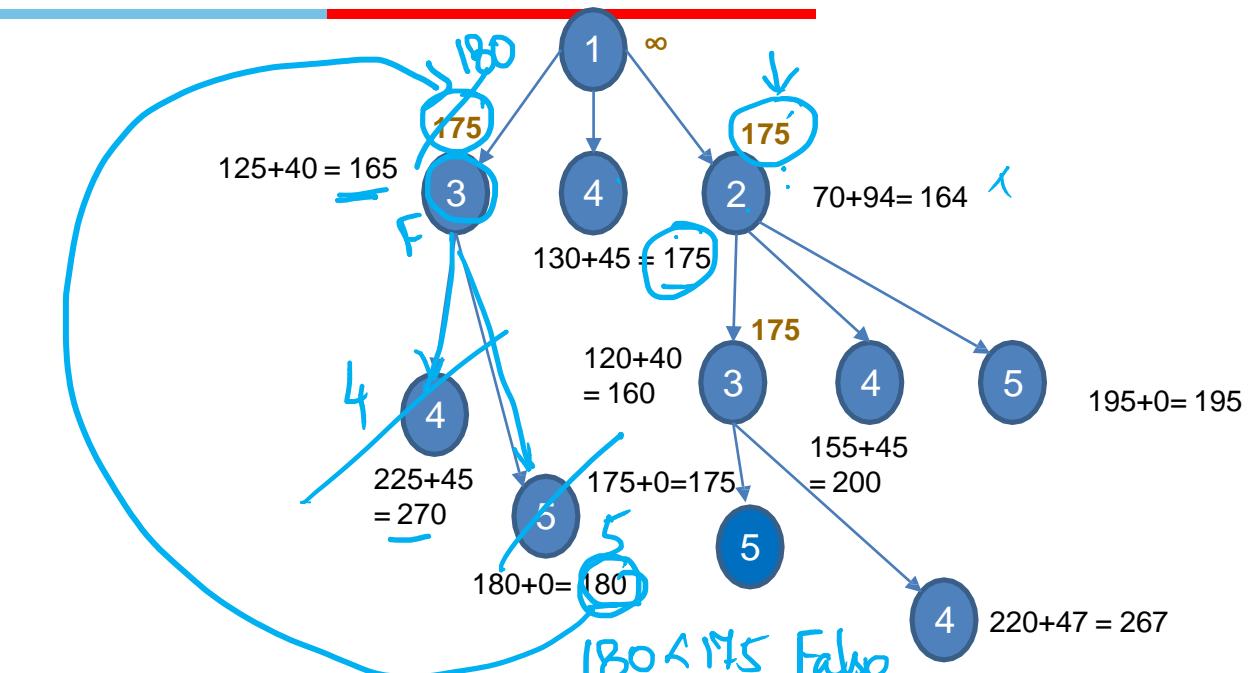
# Recursive Best First Search A\*



Remember the next best alternative f-Cost to regenerate



n	$h(n)$
1	60
2	94
3	40
4	45
5	0



(1, 60)

(1 2 | 164) (1 3 | 165) (1 4 | 175)

(1 2 | 175) (1 4 | 175) (1 3 | 180)

(1 2 3 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200)

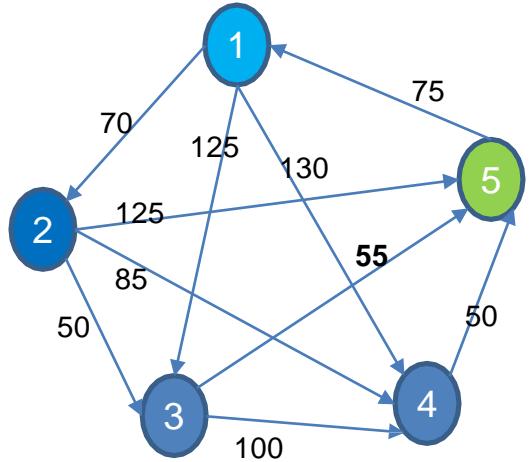
(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

PASS

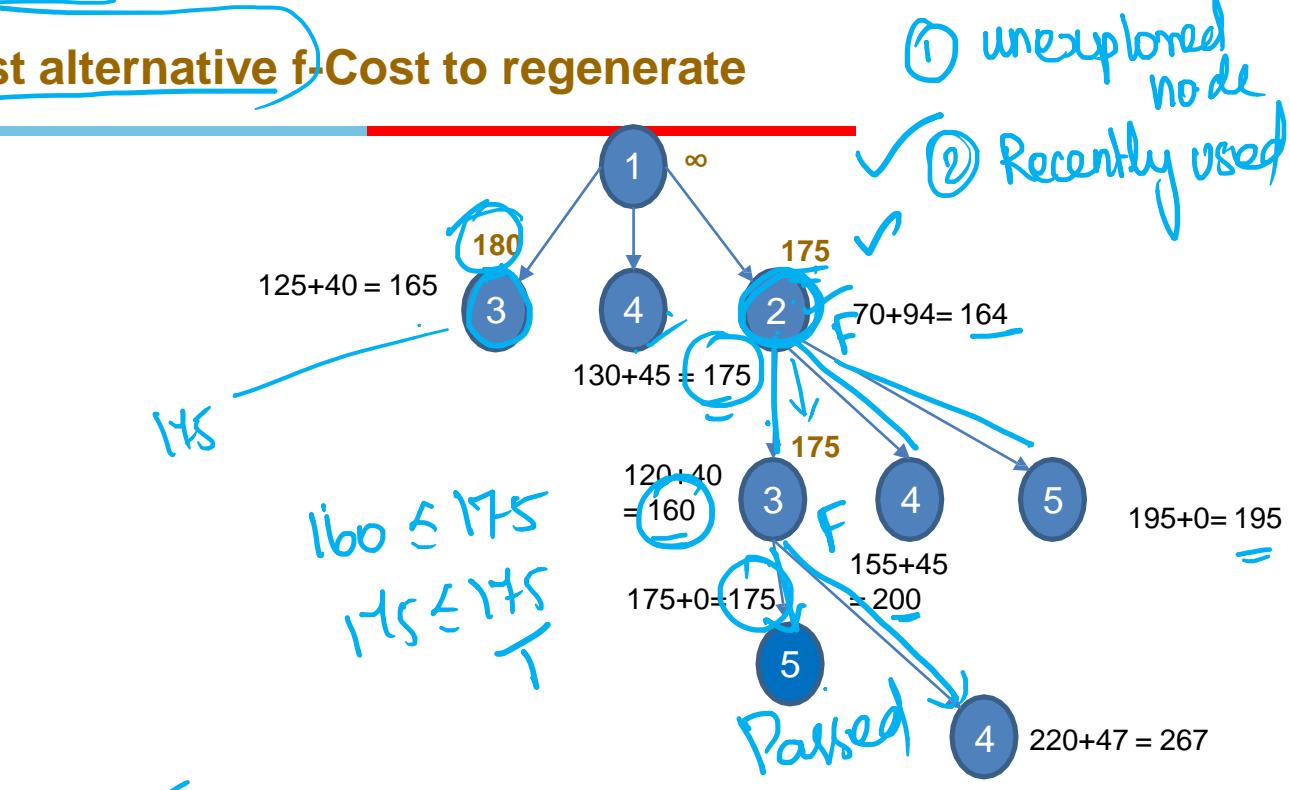
# Recursive Best First Search A\*



Remember the next best alternative f-Cost to regenerate



n	$h(n)$
1	60
2	94
3	40
4	45
5	0



(1, 60)  
 (1 2 | 164) (1 3 | 165) (1 4 | 175)

(1 2 | 175) (1 4 | 175) (1 3 | 180)

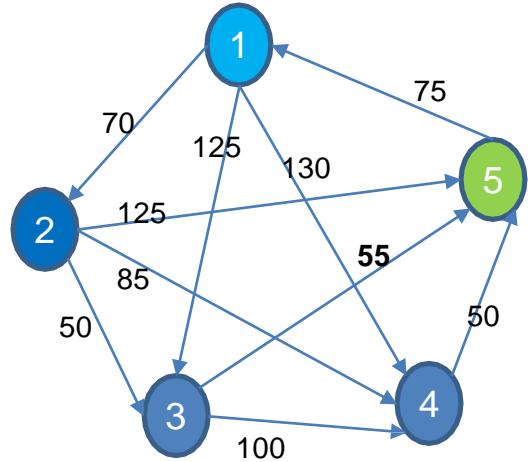
(1 2 3 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200)

(1 2 3 5 | 175) (1 4 | 175) (1 3 | 180) (1 2 5 | 195) (1 2 4 | 200) (1 2 3 4 | 267)

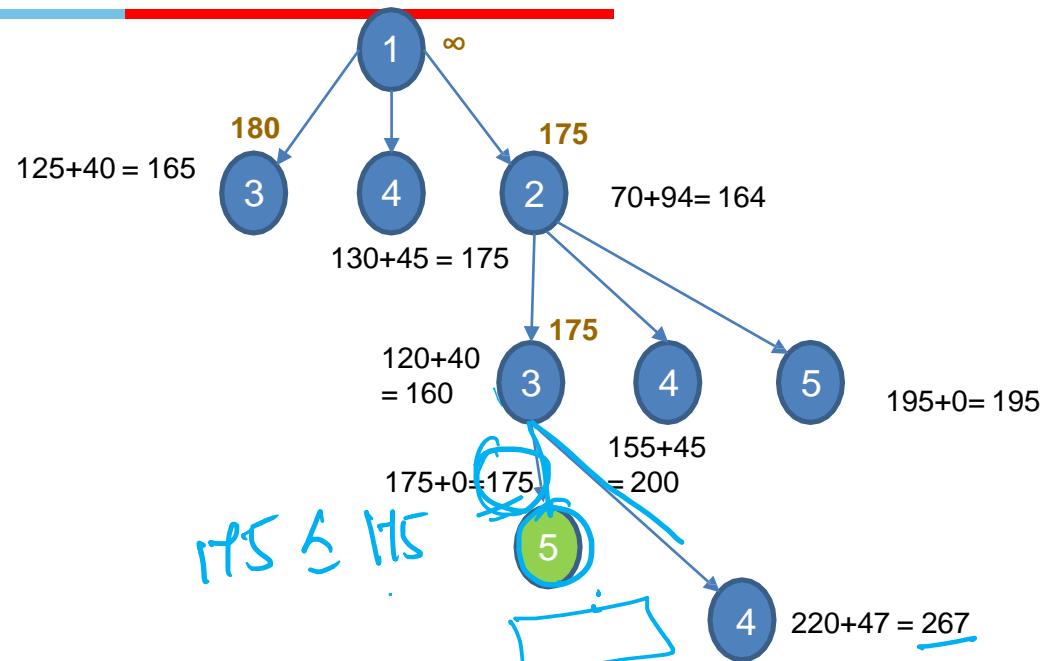
PASS

# Recursive Best First Search A\*

Remember the next best alternative f-Cost to regenerate



n	$h(n)$
1	60
2	94
3	40
4	45
5	0



If the current best leaf value > best alternative path  
Best leaf value of the forgotten subtree is backed up to the  
ancestors

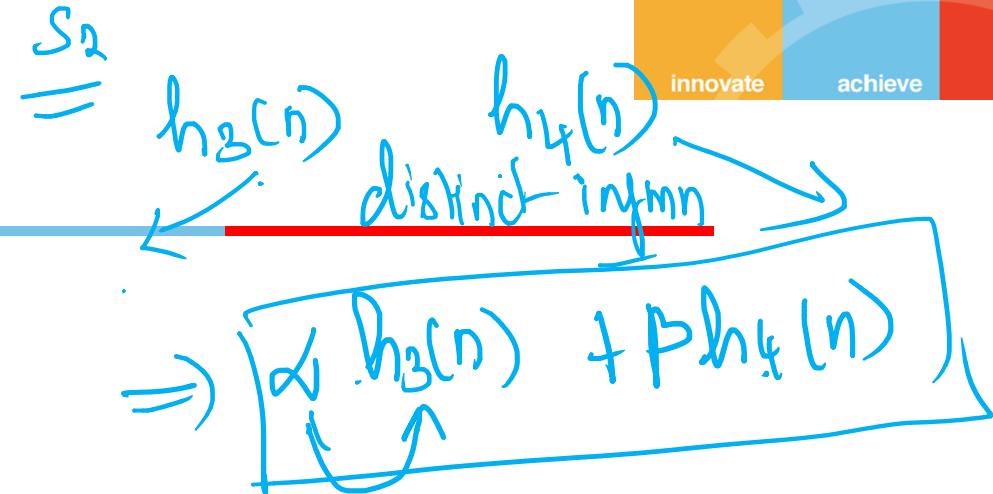
Else  
Recursion unwinds  
Continue expansion

Space Usage =  $O(bd)$  very less



## Heuristic Design

- Effective Branching Factor
- Good Heuristics (CT)
- Notion of Relaxed Problems
- Generating Admissible Heuristics



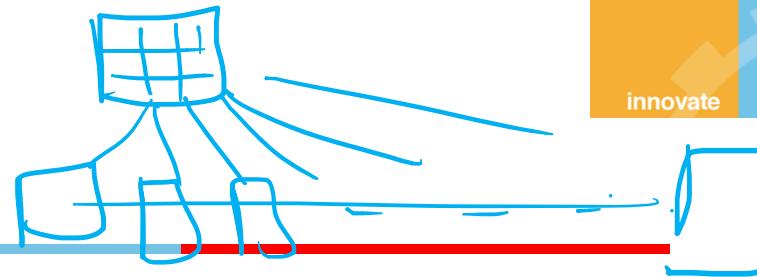
Effective branching factor ( $b^*$ ):

If the algorithm generates  $N$  number of nodes and the solution is found at depth  $d$ , then

$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + \dots + (b^*)^d$$

$S_3$   $\sqrt{h_x(n)} > h_y(n)$ . } Similar data-  
addn info

# Heuristic Design



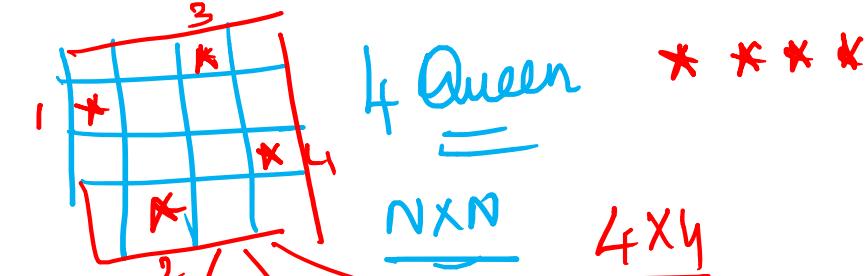
- Effective Branching Factor
- Good Heuristics
- **Notion of Relaxed Problems**
- Generating Admissible Heuristics

Simplify the problem

~~Original~~

Assume no constraints

Cost of optimal solution to relaxed problem  $\leq$  Cost of optimal solution for real problem



- ① ~~NOT  $\rightarrow$  same row~~
- ② "  $\rightarrow$  " col
- ③ " diagonal"

~~Original~~

~~1~~

~~2~~

~~3~~

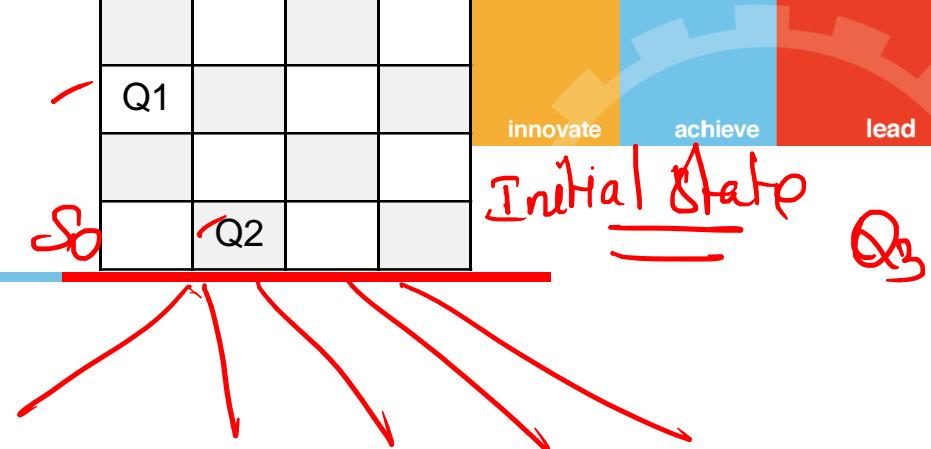
$4 \times 4$



# Design of Heuristics

## N-Queen

	Q		
			Q
Q			
		Q	

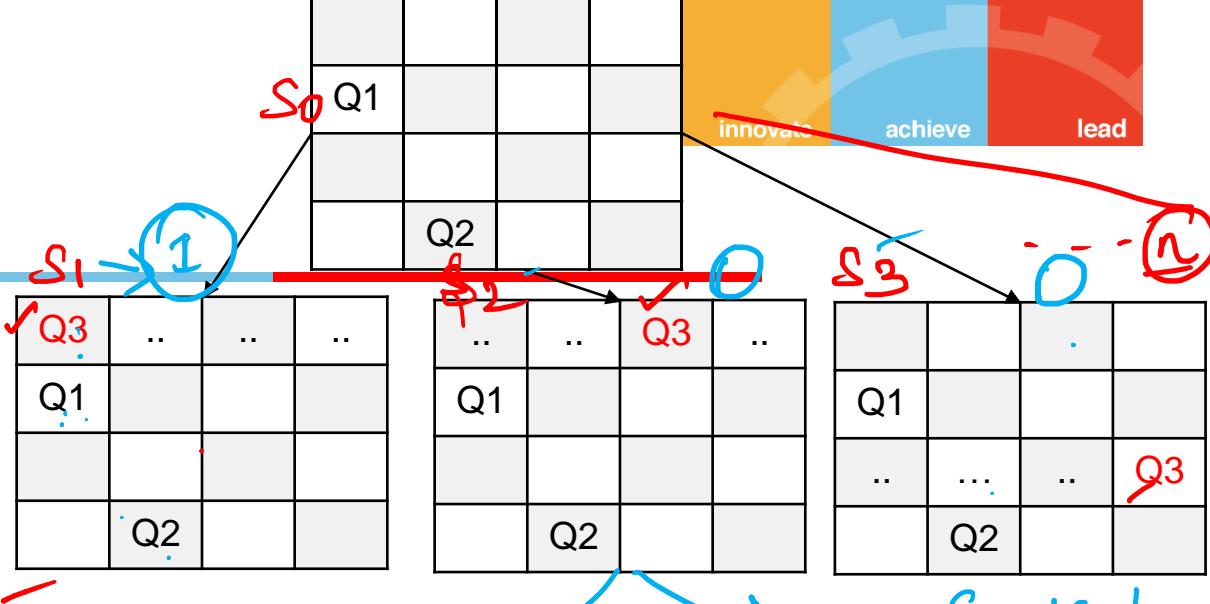


- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
$< X_i , Y_i >$	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	$n!$

# N-Queen

	Q		
			Q
Q			
		Q	



- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

Design 1 :  $h_1(n)$  : # of conflicting pairs of Queen

Design 2  $h_2(n)$  : # of non-conflicting pairs

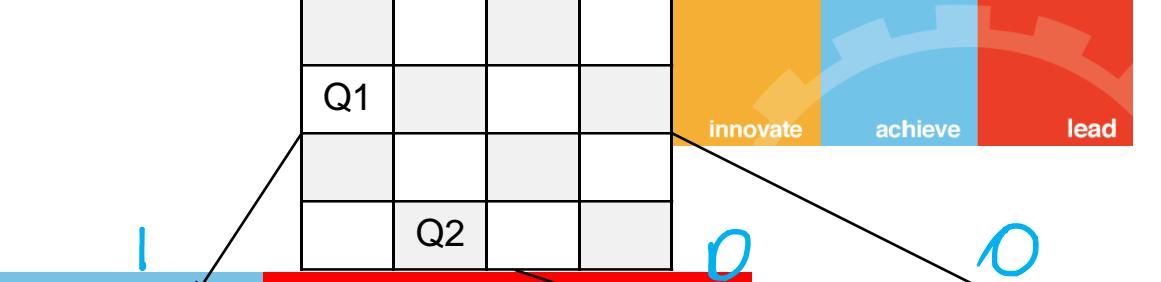
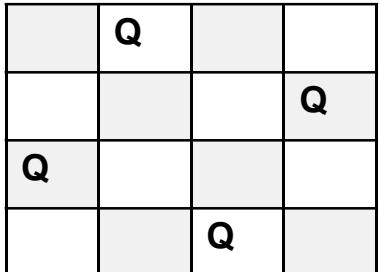
Design 3  $h_3(n)$  : # of safest Queen

$S_1$	$S_2$	$S_3$
$Q_1 Q_2$	AVC	NC NC
$Q_1 Q_3$	C	NC NC
$Q_2 Q_3$	NC	NC NC

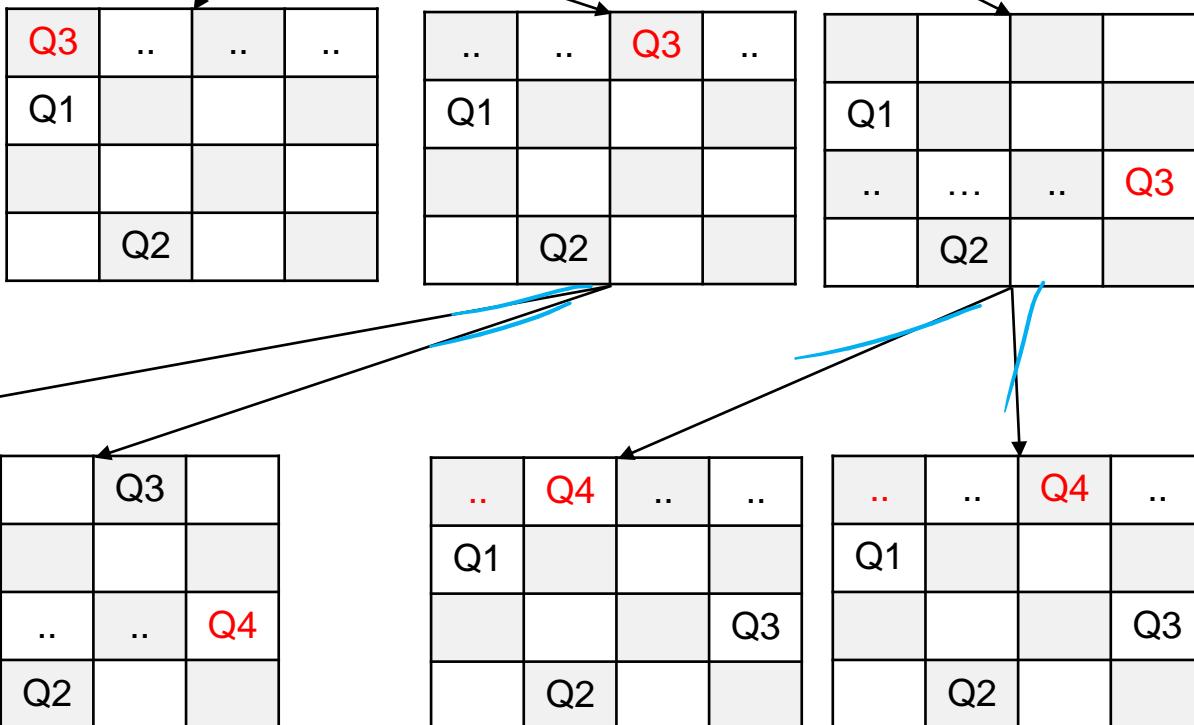
Min	D1	1	0	0
NC	D2	2	3	3
Safe	D3	1	3	3

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of. States
$< X_i, Y_i >$	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	$n!$

# N-Queen



- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions



Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
$< X_i , Y_i >$	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	$n!$

Goal

N-Tile  
=

-	1	2
3	4	5
6	7	8

S0

1	7	6
4	8	5
-	2	3

Initial

innovate

achieve

lead

Swapping → adjacent position

$h_1(n) \Rightarrow$  Manhattan distance  
↓ of empty tile

$h_2(n) \Rightarrow$  m. distance  
of all labelled  
tile.

$h_3(n) \Rightarrow$  no. of misplaced tile w.r.t goal

S1

1	7	6
4	8	5
2	-	3

1	7	6
-	8	5
4	2	3

S2

S3

1	7	6
4	-	5
2	8	3

S4

1	7	6
4	8	-
2	3	-

S5

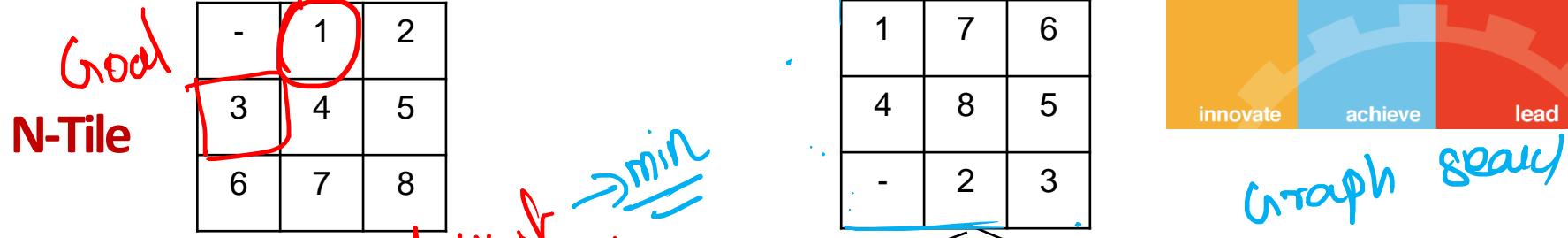
1	7	6
8	-	5
4	2	3

S6

-	7	6
1	8	5
4	2	-3

7+

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of STATES
<LOC, ID>	Move Empty to near by Tile		ID=LOC+1	Transition + Positional + Distance+ Other approaches	9!



$h_3(n) \Rightarrow$  no. of misplaced tile wrt goal

$h_2(n) =$  dis of all labelled tile

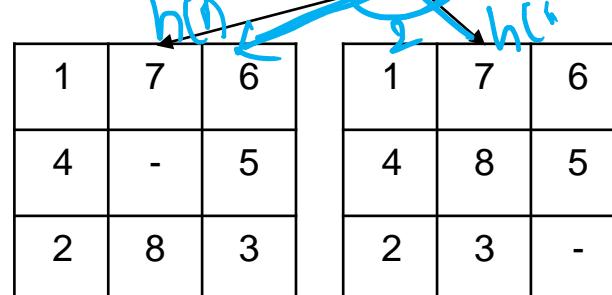
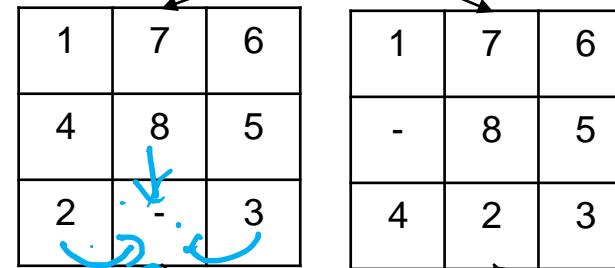
cells away

1 2  
2 3  
3 2  
4 3  
5 0  
6 4  
7 2  
8 1

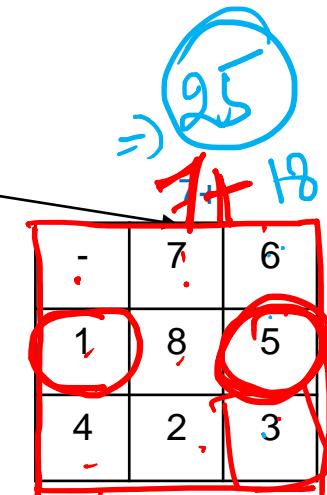
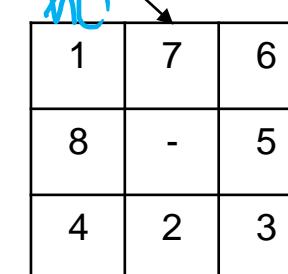
18



graph search



Hint

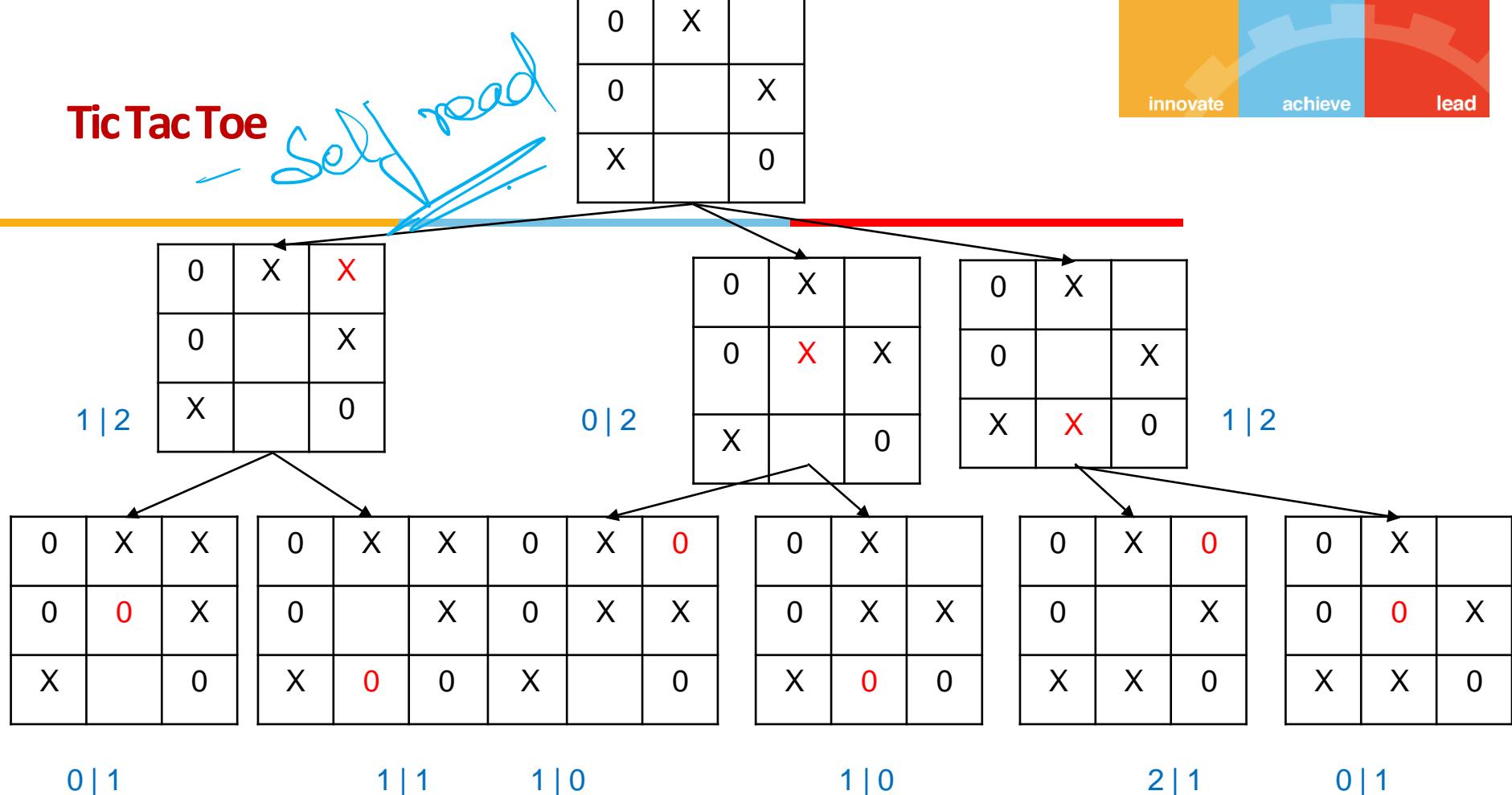


1 - NO  
2 - NO  
3 - NO

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
<LOC, ID>	Move Empty to near by Tile		ID=LOC+1	Transition + Positional + Distance+ Other approaches	9!

# Tic Tac Toe

*Solve road*



Opposite Win | Player Win

Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
([X <sub>ij</sub> ], [Y <sub>ij</sub> ])	Place a coin in unoccupied (i,j)		N : i's N : j's N : i=j	No.of.Steps + Opp.Win + (N-1-Curr.Win)	19,683=3 <sup>9</sup>

## Learn from experience

Trail / Puzzle	X1(n) : No.of.Misplaced Tiles	X2(n): Pair of adjacent tiles that are not in goal	X3(n): Position of the empty tile	.....h'(n)
Example 1	7	10	7	.....
Example 2	5	6	6	.....
.....	..	..	..	.....

-	1	2
3	4	5
6	7	8

1	7	6
4	8	5
-	2	3

Create a suitable model:

$$h(n) = c_1 \cdot X_1(n) + c_2 \cdot X_2(n) + \dots$$

## Module 2 : Problem Solving Agent using Search

- A. Uninformed Search
- B. Informed Search
- C. Heuristic Functions
- D. Local Search Algorithms & Optimization Problems

## Learning Objective

At the end of this class , students Should be able to:

1. Differentiate which local search is best suitable for given problem
2. Design fitness function for a problem
3. Construct a search tree
4. Apply appropriate local search and show the working of algorithm at least for first 2 iterations with atleast four next level successor generation(if search tree is large)
5. Design and show local search Algorithm steps for a given problem

## Greedy choice ————— Local Search & Optimization

Memory limit



① Partial  
Obs

Sensor ✓ —  
② computational  
capability

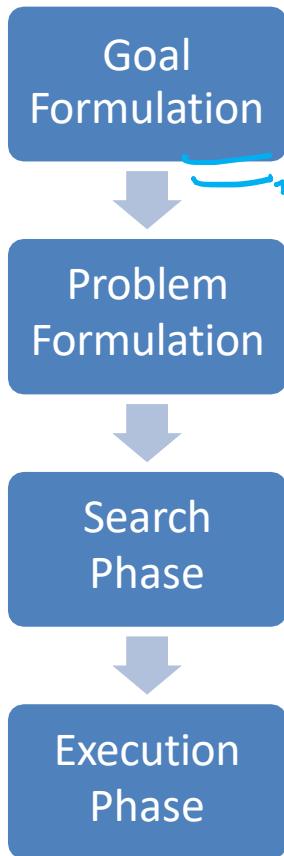
③ Irrelevant path  
④ Expected goal



## Task Environment

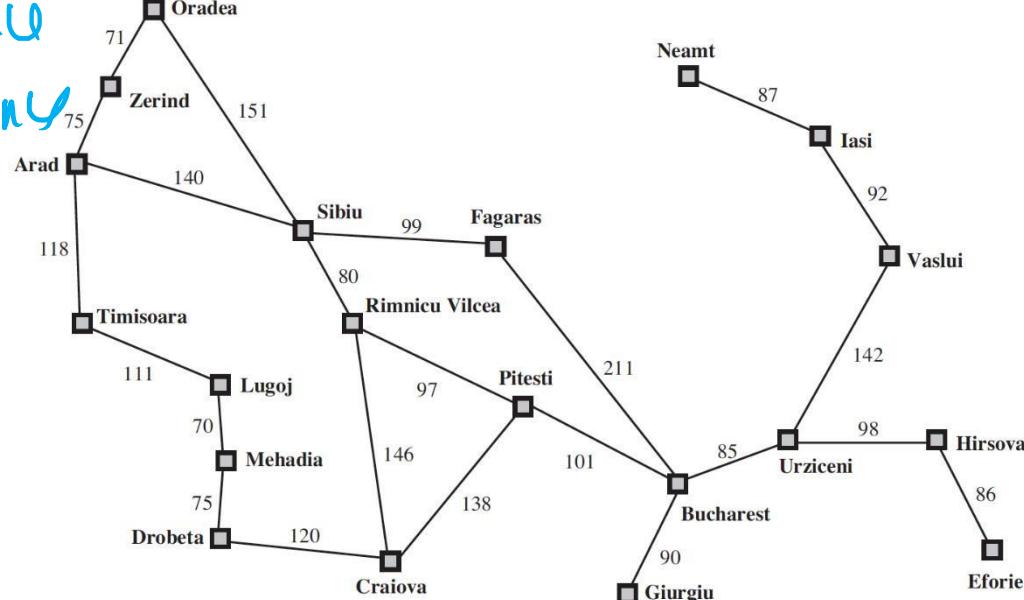
Optimal Sol → near to the goal

### Phases of Solution Search by PSA

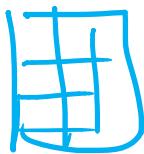


**Assumptions – Environment :**  
**Static (4.5)**  
**Observable**  
**Discrete (4.4)**  
**Deterministic (MDP)**

Single instance  
Multiple instance

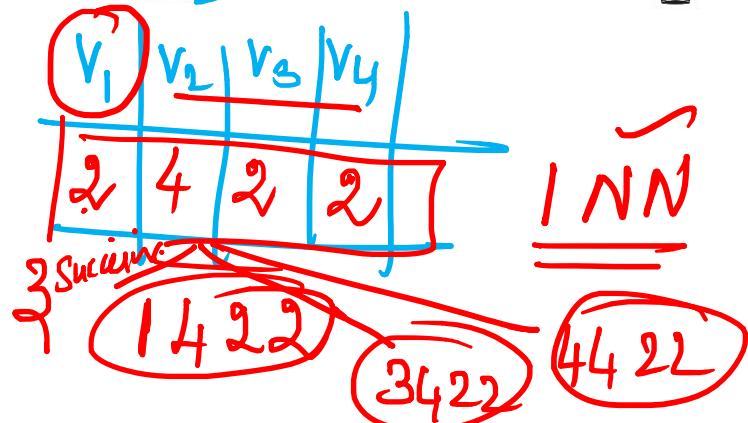
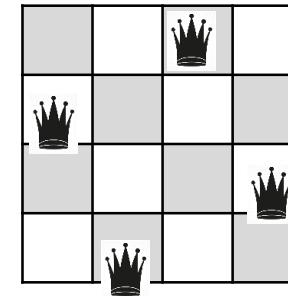
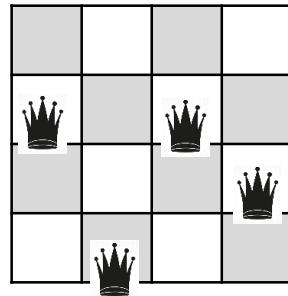
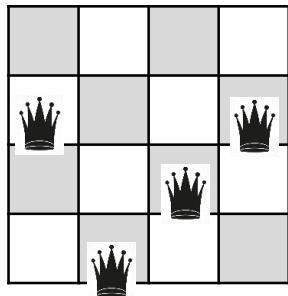
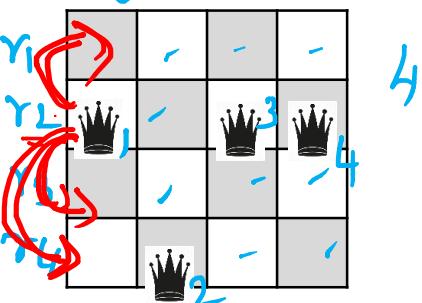


## Local Search

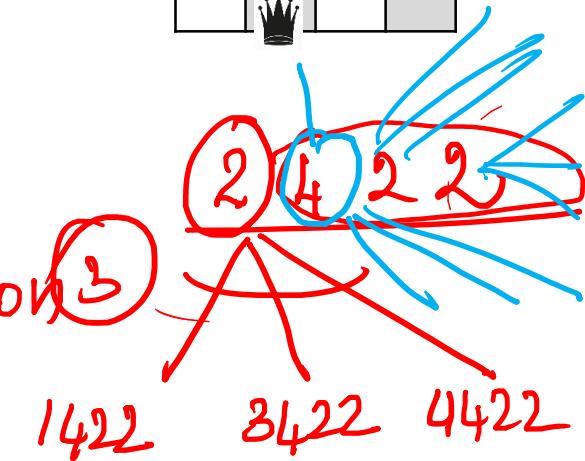


## Terminology

**Local Search** : Search in the state-space in the neighbourhood of current position until an optimal solution is found



Neighborhood  
region expansion



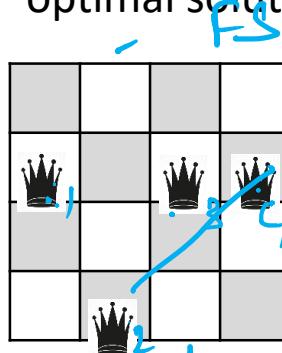
12 successors

Local Search

$Q_1 Q_2$	NC
$Q_1 Q_3$	C
$Q_1 Q_4$	C
$Q_2 Q_3$	NC

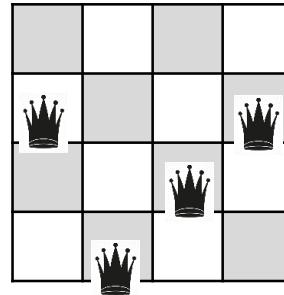
Terminology

**Local Search** : Search in the state-space in the neighbourhood of current position until an optimal solution is found

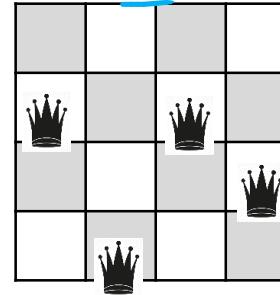


13  
14  
24

FS



Neighboring States



FS

Feasible State/Solution

Fitness Value:

$$h(n) = 4$$

$$h(n) = 4$$

$$h(n) = 2$$

Optimal Solution

$$h(n) = 0$$

min

D<sub>1</sub> h(n) = No.of.Conflicting pairs of queens

$$h(n) = 2$$

$$h(n) = 2$$

$$h(n) = 4$$

D<sub>2</sub> h(n) = No.of.Non-Conflicting pairs of queens.

$$h(n) = 6$$

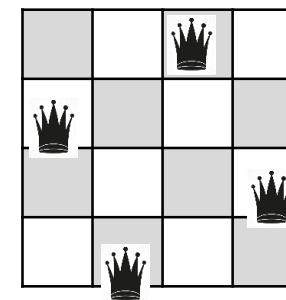
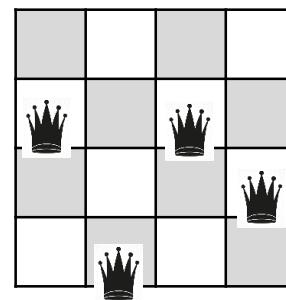
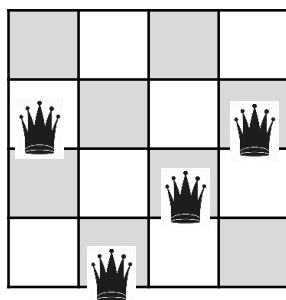
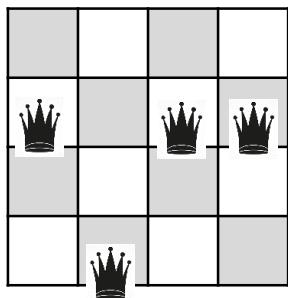
max

Local Search

Not Same row  
S " col  
,, diaognal

## Terminology

**Local Search** : Search in the state-space in the neighbourhood of current position until an optimal solution is found



### Feasible State/Solution

Fitness Value:

$$h(n) = 4$$

$h(n)$  = No.of.Conflicting **pairs** of queens

### Neighboring States

$$h(n) = 4$$

$h(n)$  = No.of.Conflicting **pairs** of queens.

$$h(n) = 2$$

$h(n)$  = No.of.**Non**-Conflicting **pairs** of queens.

$$h(n) = 2$$

$$h(n) = 4$$

### Optimal Solution

$$h(n) = 0$$

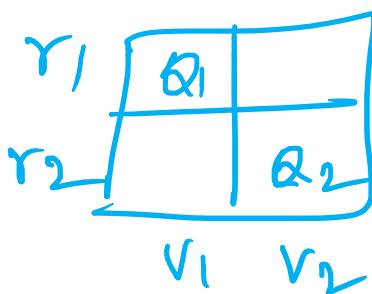
$$h(n) = 6$$

## Terminology

**Local Search** : Search in the state-space in the neighbourhood of current position until an optimal solution is found

### Algorithms:

- Choice of Neighbour
- Looping Condition
- Termination Condition



$\{1, 2\}$

$\{2, 1\}$

1NN



1NN

2NN

NNN



2NN

(1, 2)

(2, 1)



4 - all vau

$1 \rightarrow 2$   
 $2 \rightarrow 1$

## Optimization Problem

**Goal** : Navigate through a state space for a given problem such that an optimal solution can be found

**Objective** : Minimize or Maximize the objective evaluation function value

**Scope** : Local

**Objective Function** : Fitness Value evaluates the goodness of current solution

**Local Search** : Search in the state-space in the neighbourhood of current position until an optimal solution is found

### Single Instance Based

Hill Climbing

Simulated Annealing

Local Beam Search

Tabu Search

### Multiple Instance Based

Genetic Algorithm

Particle Swarm Optimization

Ant Colony Optimization



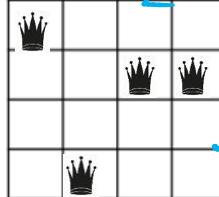
# Hill Climbing

## Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

$h(n)$  = No.of non-conflicting pairs of queens in the board. *max*

Q1-Q2



Q1-Q3

Q1-Q4

Q2-Q3

Q2-Q4

Q3-Q4

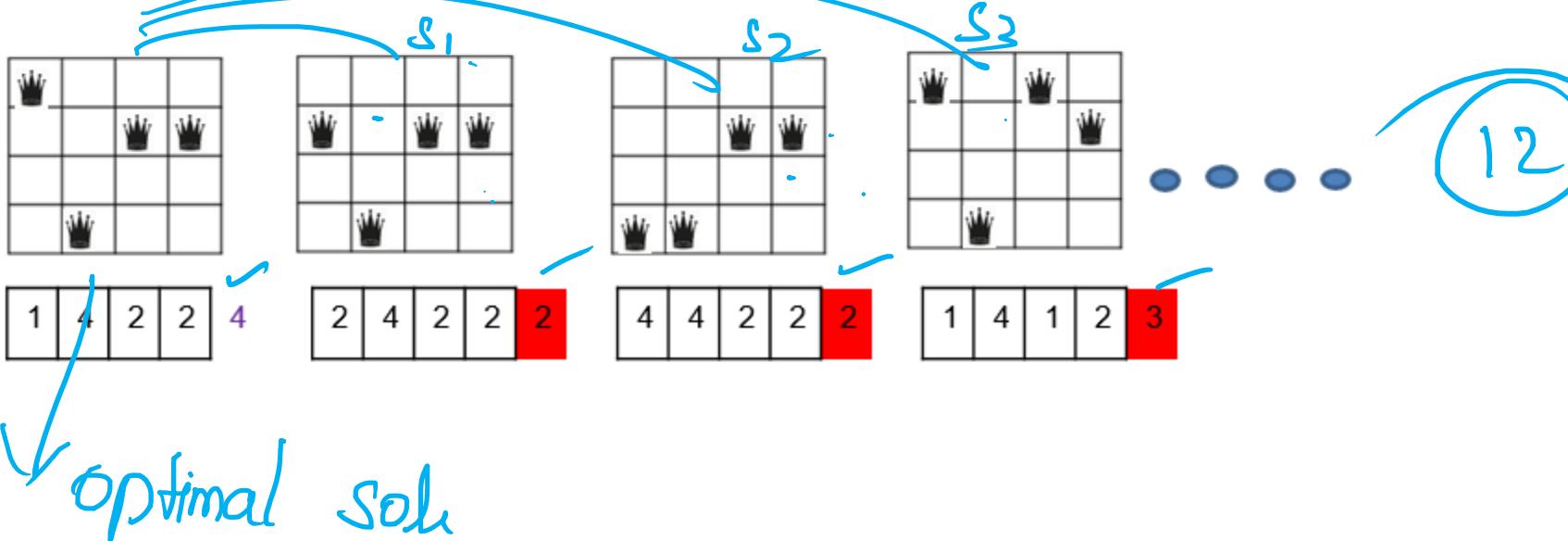


Note : Steps 3 & 4 in the above algorithm will be a part of variation of Hill climbing

## Hill Climbing

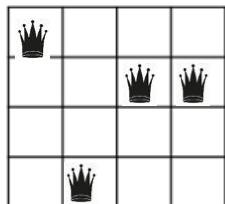


1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

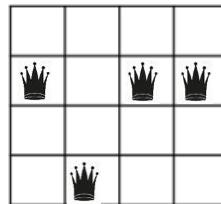


# Hill Climbing

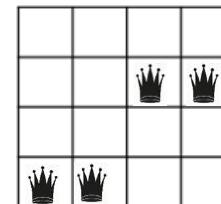
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



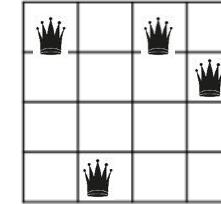
1	4	2	2
---	---	---	---



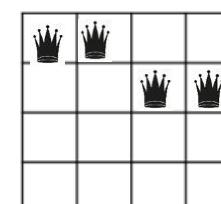
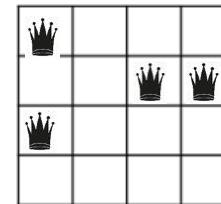
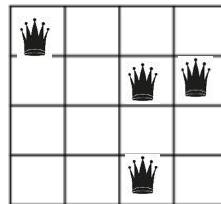
2	4	2	2
---	---	---	---



4	4	2	2
---	---	---	---

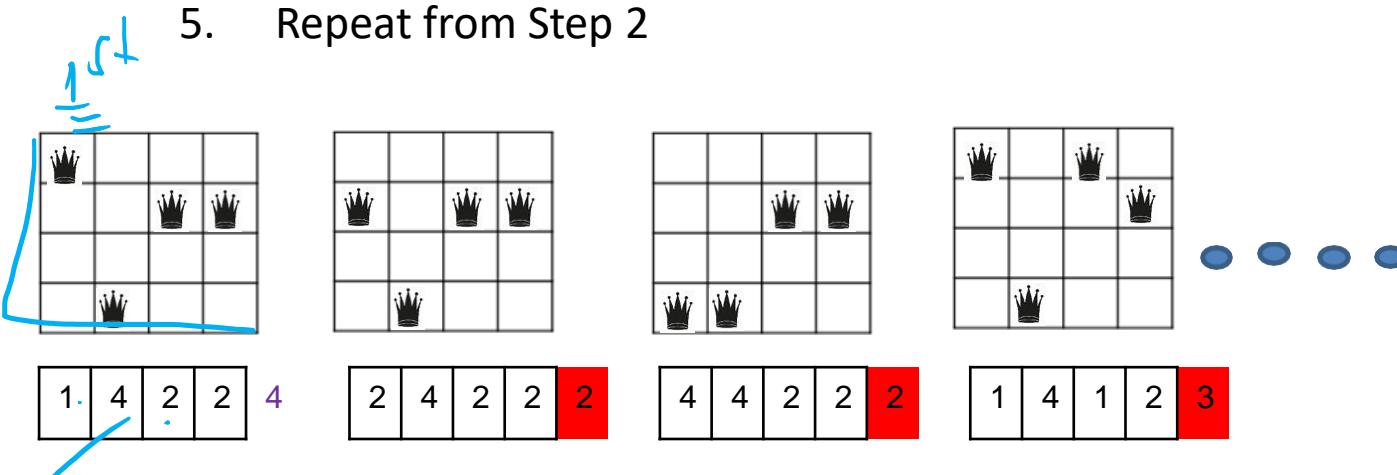


1	4	1	2
---	---	---	---



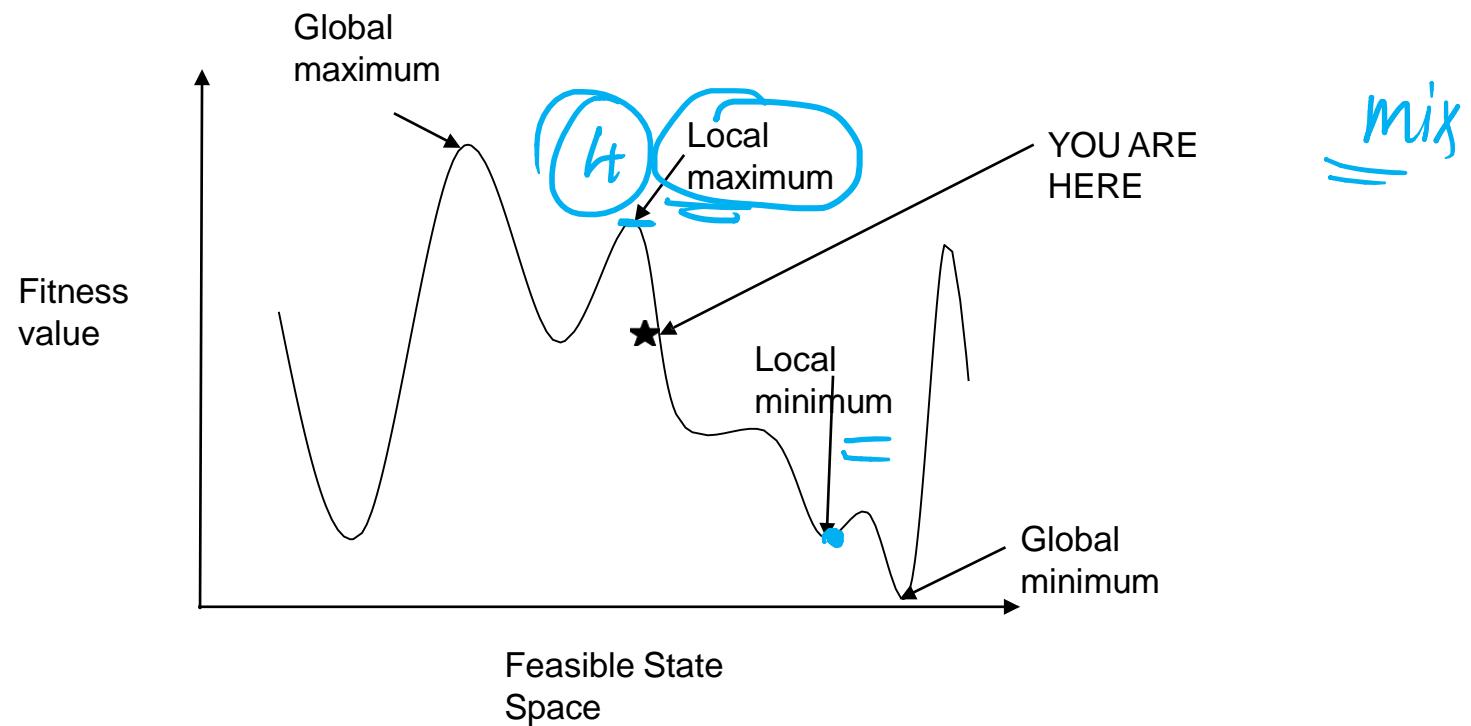
## Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



## Hill Climbing

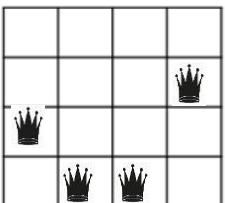
$h(n)$  max no of non- b



## Random Restart

1 iteration

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

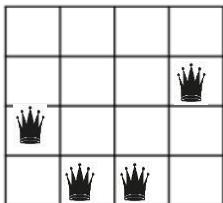


3	4	4	2	3
---	---	---	---	---

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor  $\leftarrow$  a highest-valued successor of current
        if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE
        current  $\leftarrow$  neighbor
```

## Random Restart

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

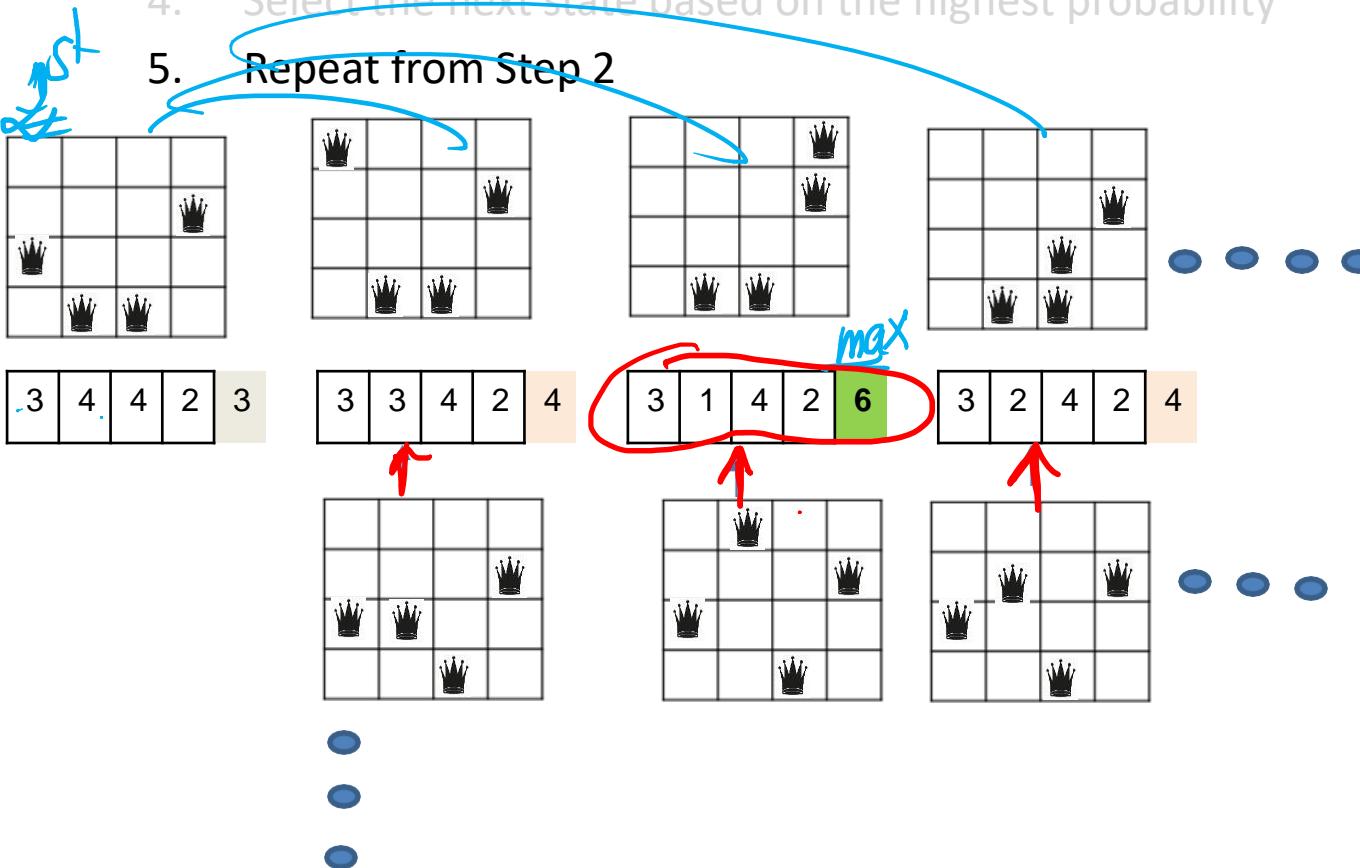


3	4	4	2	3
---	---	---	---	---

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor  $\leftarrow$  a highest-valued successor of current
        if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE
        current  $\leftarrow$  neighbor
```

## Random Restart

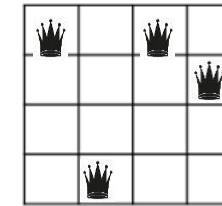
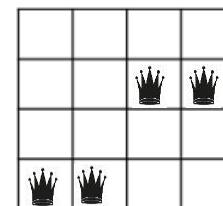
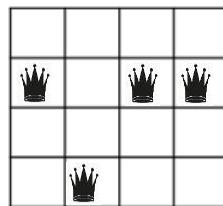
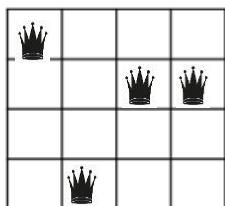
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



# Stochastic Hill Climbing



1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



1	4	2	2	4
---	---	---	---	---

2	4	2	2	2
---	---	---	---	---

4	4	3	3	2
---	---	---	---	---

1	4	1	2	3
---	---	---	---	---

0.08

0.42

0.42

$$12 \text{ N} = \{4, 2, 2, 3, 3, 2, 2, 0, 2, 1, 3, 0\}$$



# Artificial & Computational Intelligence

DSECLZG557

## M2 : Problem Solving Agent using Search

Indumathi V  
Guest Faculty,  
BITS - WILP

**BITS** Pilani  
Pilani Campus

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

## Module 2 : Problem Solving Agent using Search

- A. Uninformed Search
- B. Informed Search
- C. Heuristic Functions
- D. Local Search Algorithms & Optimization Problems

## Learning Objective

At the end of this class , students Should be able to:

1. Differentiate which local search is best suitable for given problem
2. Design fitness function for a problem
3. Construct a search tree
4. Apply appropriate local search and show the working of algorithm at least for first 2 iterations with atleast four next level successor generation(if search tree is large)
5. Design and show local search Algorithm steps for a given problem



# Local Search & Optimization

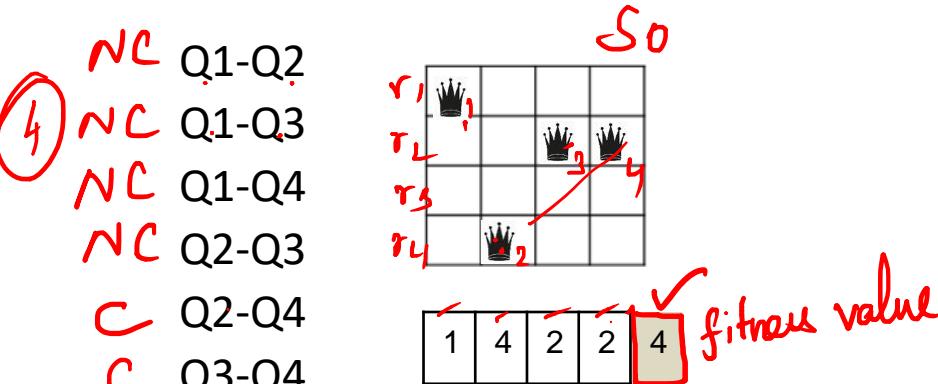


# Hill Climbing

# Hill Climbing

1. ✓Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

$h(n)$  = No.of non-conflicting pairs of queens in the board.

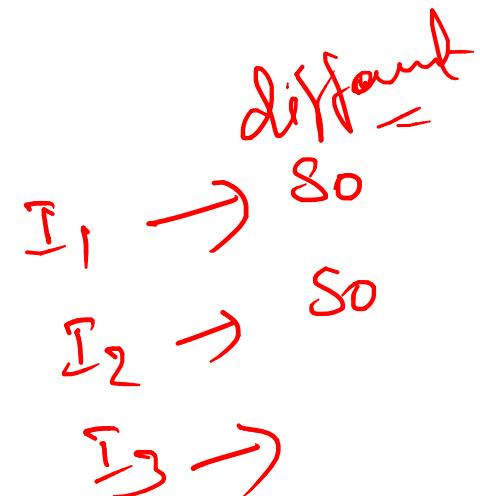
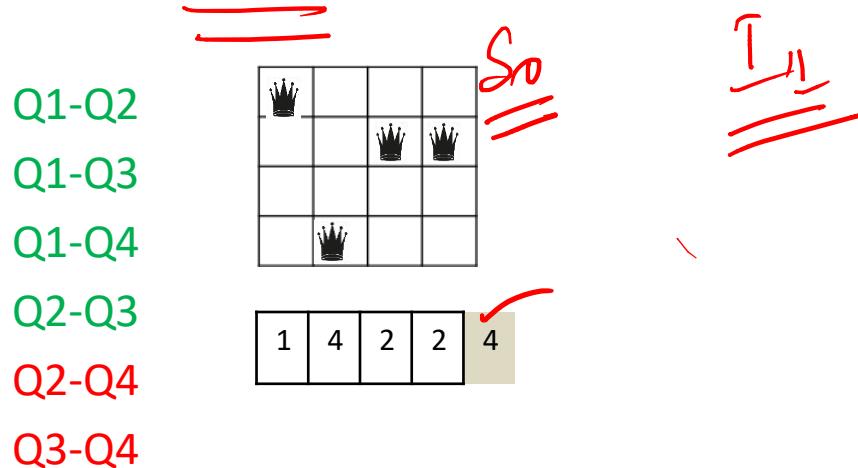


Note : Steps 3 & 4 in the above algorithm will be a part of variation of Hill climbing

# Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

**$h(n)$  = No.of non-conflicting pairs of queens in the board.**

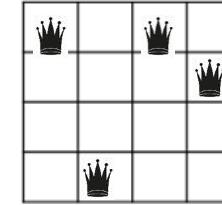
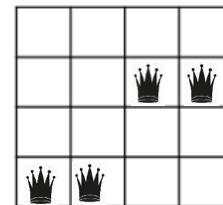
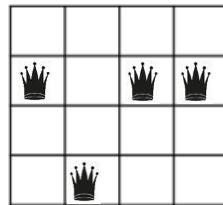
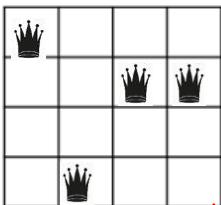


# Hill Climbing

INN

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

*s<sub>0</sub>*



...

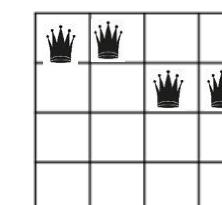
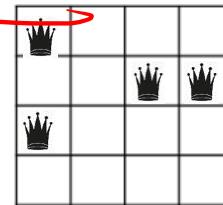
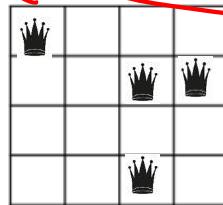
12 Successor

1	4	2	2
---	---	---	---

2	4	2	2
---	---	---	---

4	4	2	2
---	---	---	---

1	4	1	2
---	---	---	---



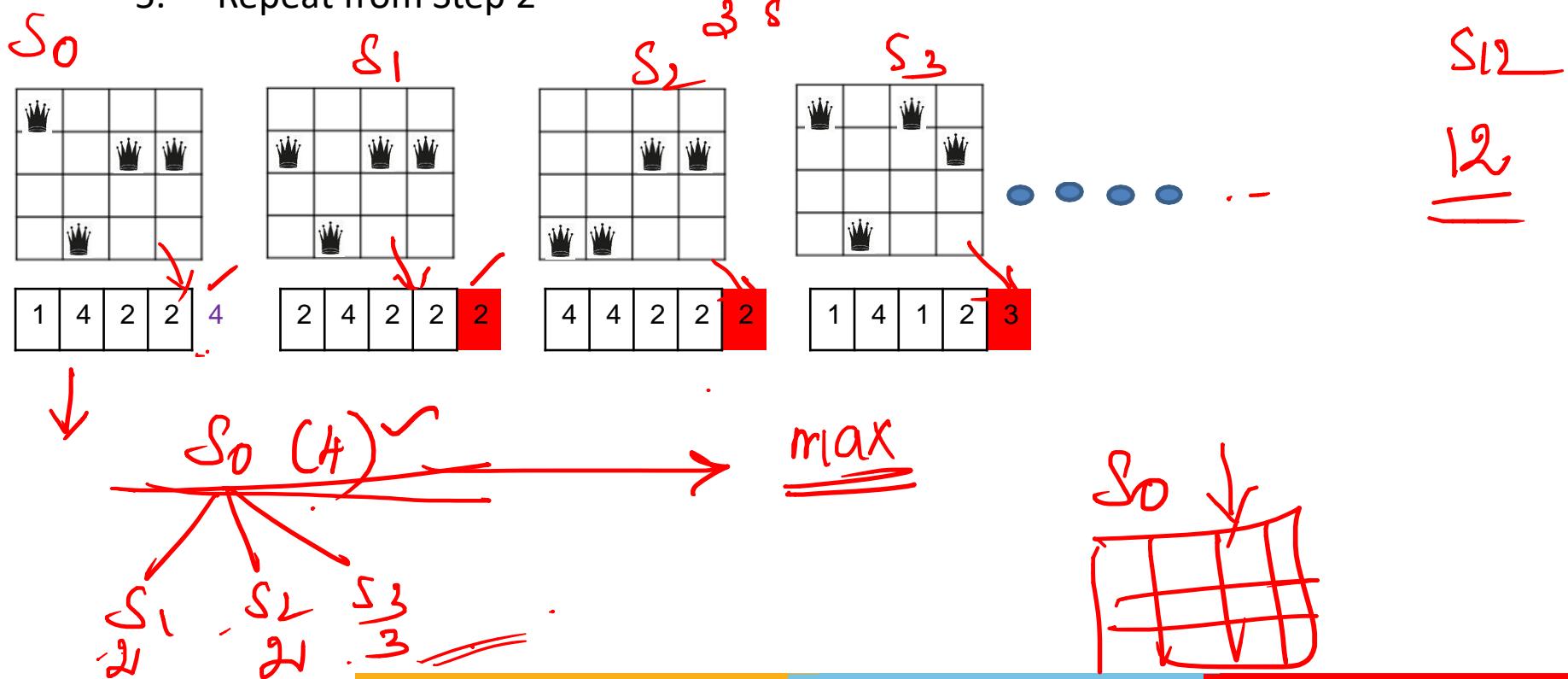
...

12

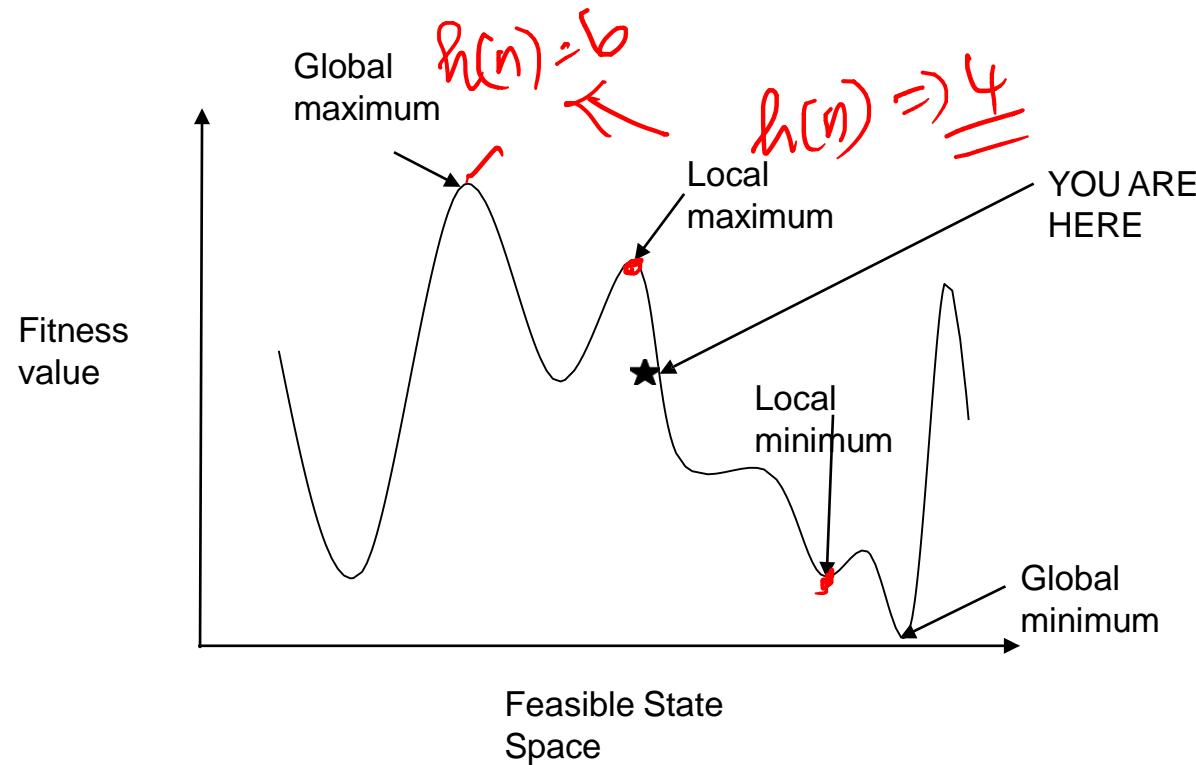


# Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2

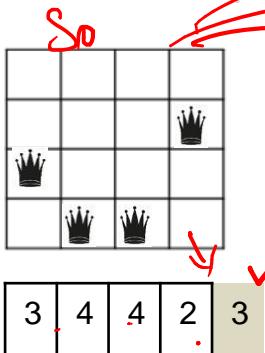


# Hill Climbing



## Random Restart

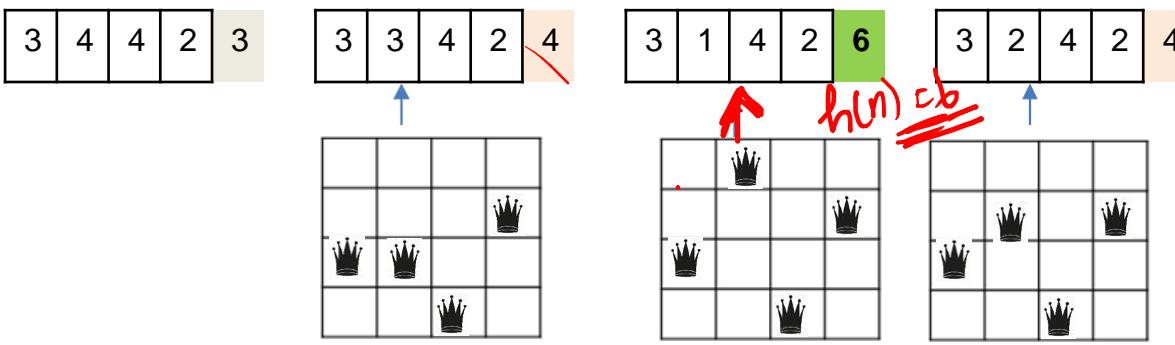
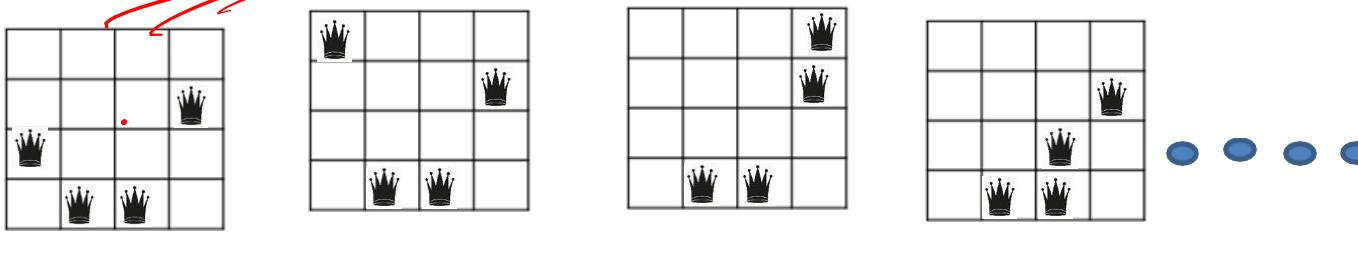
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2
6. Random Restart



```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor  $\leftarrow$  a highest-valued successor of current
    if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE
    current  $\leftarrow$  neighbor
```

# Hill Climbing

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



optimal  
80%  
~~Cal~~

100/100

$K = \frac{100}{1000}$   
 $\frac{1000}{8(n) + 5}$

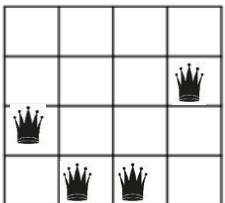
$TC_1 \rightarrow$  optimal soln

$TC_2 = \frac{K=100}{(100 \text{ firms})}$   
~~Shop~~

~~$TC_2 h(n) = 4 \text{ or } 5$~~

## Random Restart

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2 6.



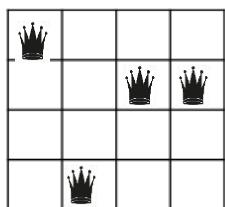
3	4	4	2	3			
---	---	---	---	---	--	--	--

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor  $\leftarrow$  a highest-valued successor of current
        if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE
        current  $\leftarrow$  neighbor
```

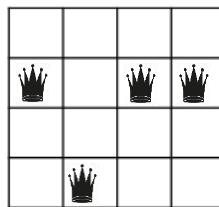
## Stochastic Hill Climbing

*↳ Probability*

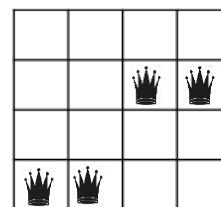
1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



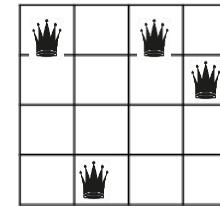
1	4	2	2
4			



2	4	2	2	2



4	4	2	2	2



1	4	1	2	3



## Stochastic Hill Climbing

$h(n) \rightarrow T_r \rightarrow \text{frequency}$

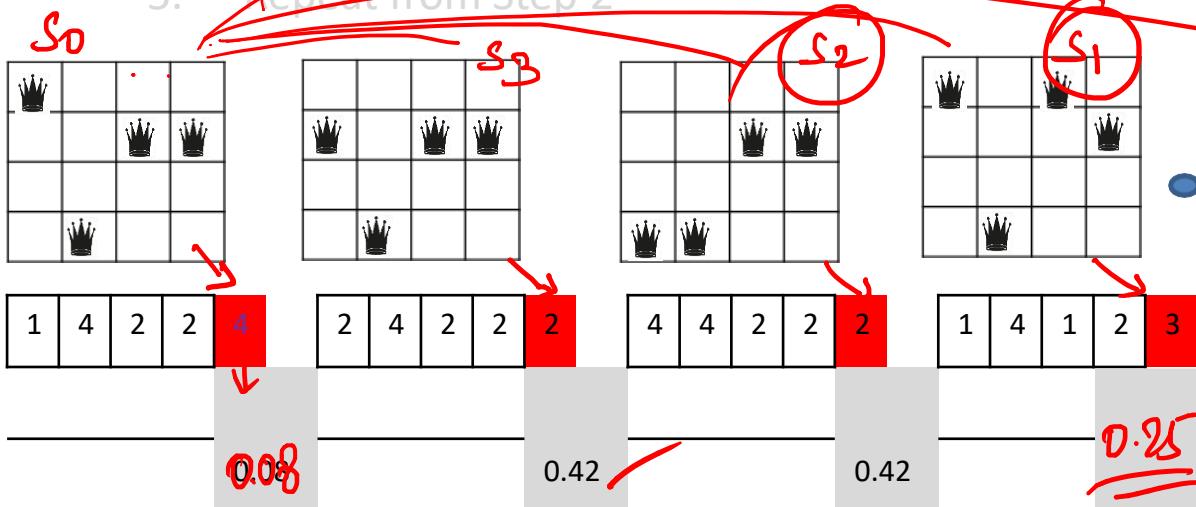


Random Probability  $\Rightarrow 0.2$

$$0.2 < 0.25$$

F

1. Select a random state
2. Evaluate the fitness scores for all the successors of the state
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. Repeat from Step 2



	fit-score	Prob
0	2	$2/12 \ 0.16$
1	1	$1/12 \ 0.08$
2	5	$5/12 \ 0.41$
3	3	$3/12 \ 0.25$
4	1	$1/12 \ 0.08$

~~$12 N = \{4, 2, 2, 3, 3, 2, 2, 0, 2, 1, 3, 0\}$~~

$12 N$

$s_1, s_2, s_3, s_4, s_{12} \rightarrow h(n)$

```

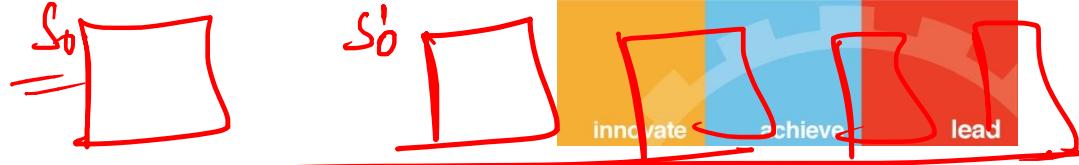
next ← a randomly selected successor of current
ΔE ← next.VALUE - current.VALUE
if ΔE > 0 then current ← next
else current ← next only with probability  $e^{\Delta E/T}$ 

```



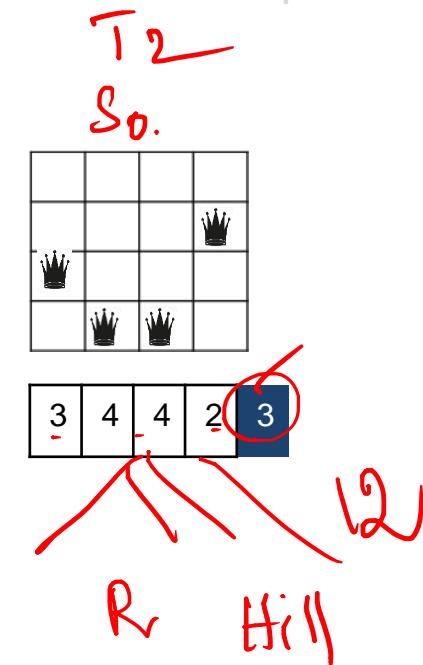
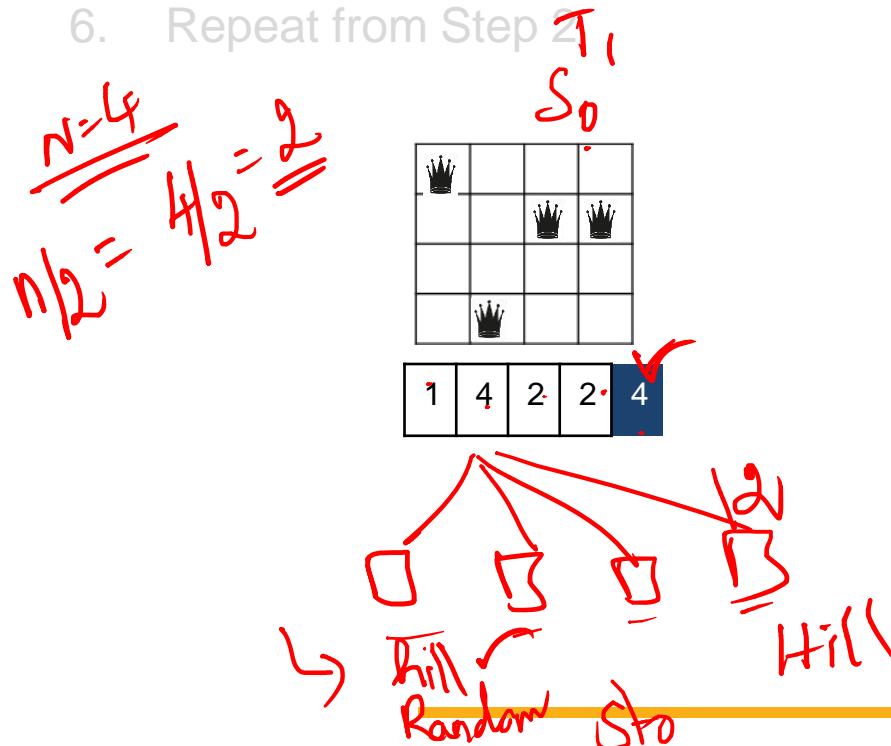
# Local Beam Search

## Beam Search



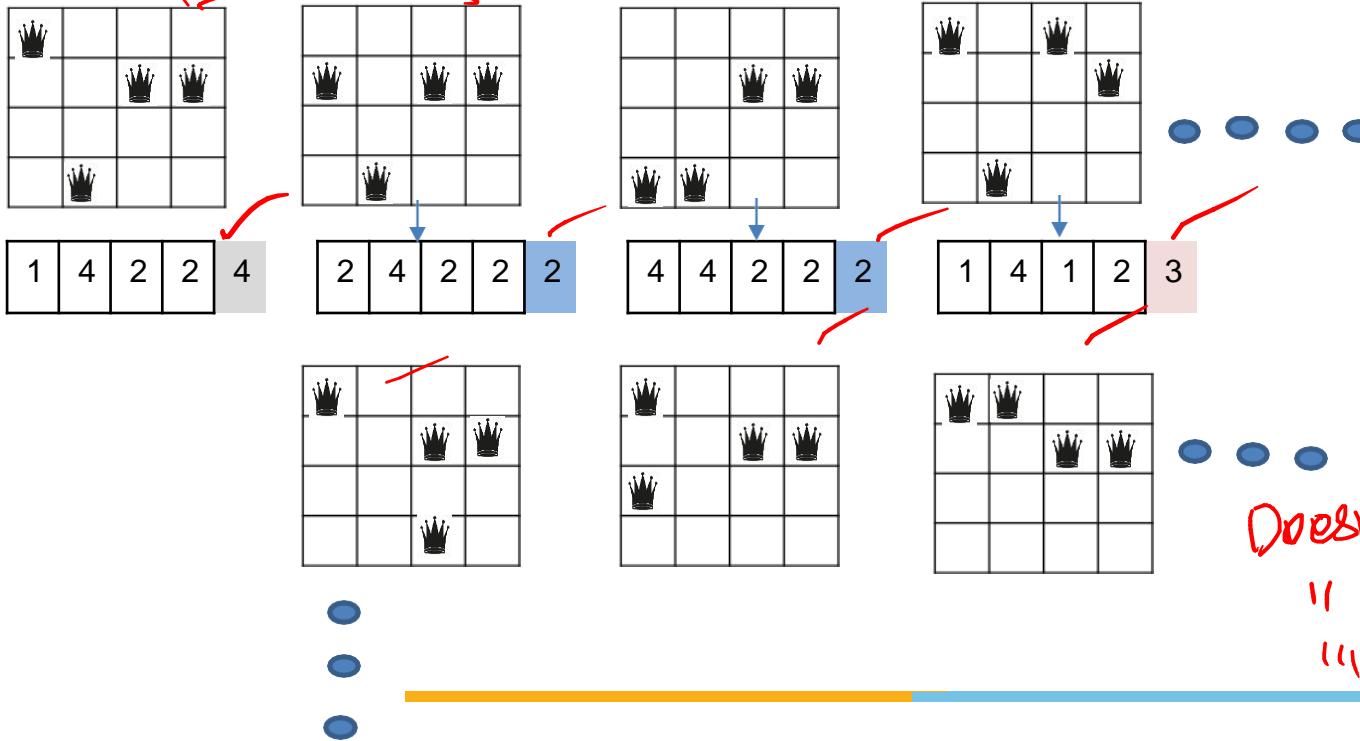
$$K = \underline{2}, \underline{1}, \underline{5}$$

1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next 'k' states randomly based on the probability
6. Repeat from Step 2



# Beam Search

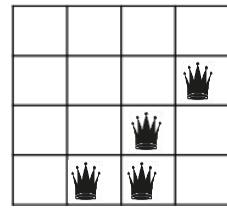
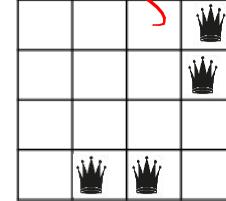
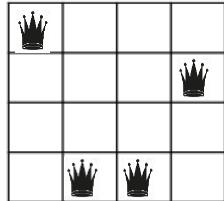
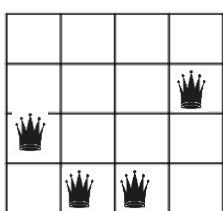
- ~~K =~~
1. Initialize K random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next 'K' states randomly based on the probability
6. Repeat from Step 2



# Beam Search

## 2<sup>nd</sup> State

1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next 'k' states randomly based on the probability
6. Repeat from Step 2.



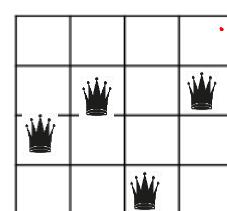
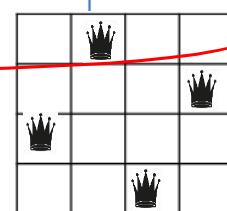
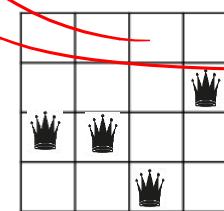
...

3	4	4	2	3
---	---	---	---	---

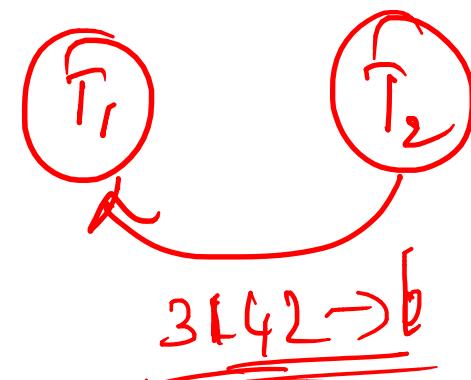
3	3	4	2	4
---	---	---	---	---

3	1	4	2	6
---	---	---	---	---

3	2	4	2	4
---	---	---	---	---



...

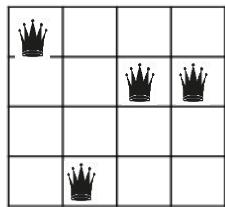


...

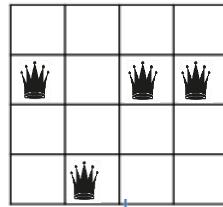
# Stochastic Beam Search

## Sample from 1<sup>st</sup> State

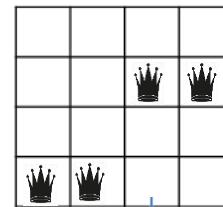
1. Initialize k random state
2. Evaluate the fitness scores for all the successors of the k states
3. Calculate the probability of selecting a successor based on fitness score
4. Select the next state based on the highest probability
5. If the goal is not found, Select the next 'k' states randomly based on the probability
6. Repeat from Step 2



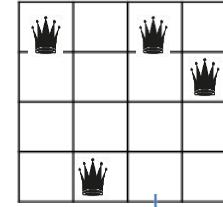
1	4	2	2	2	4		
---	---	---	---	---	---	--	--



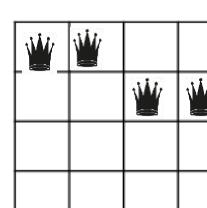
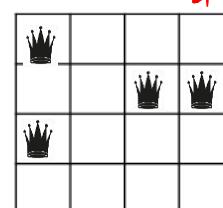
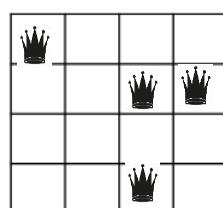
2	4	2	2	2	2		
---	---	---	---	---	---	--	--

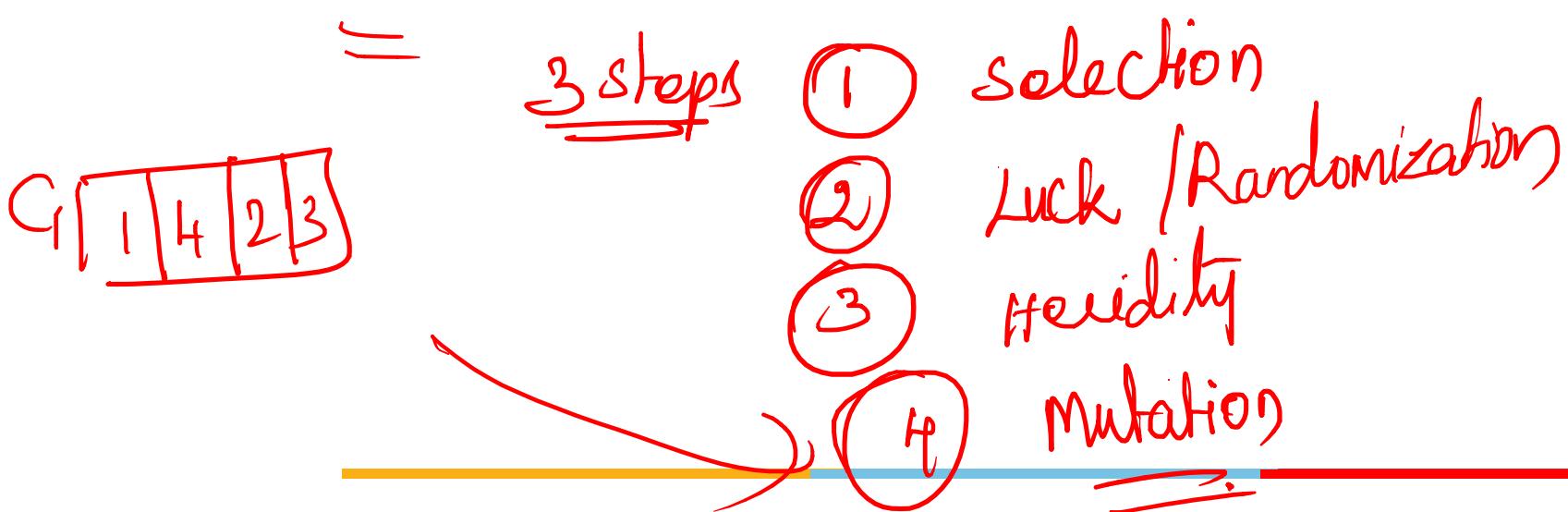
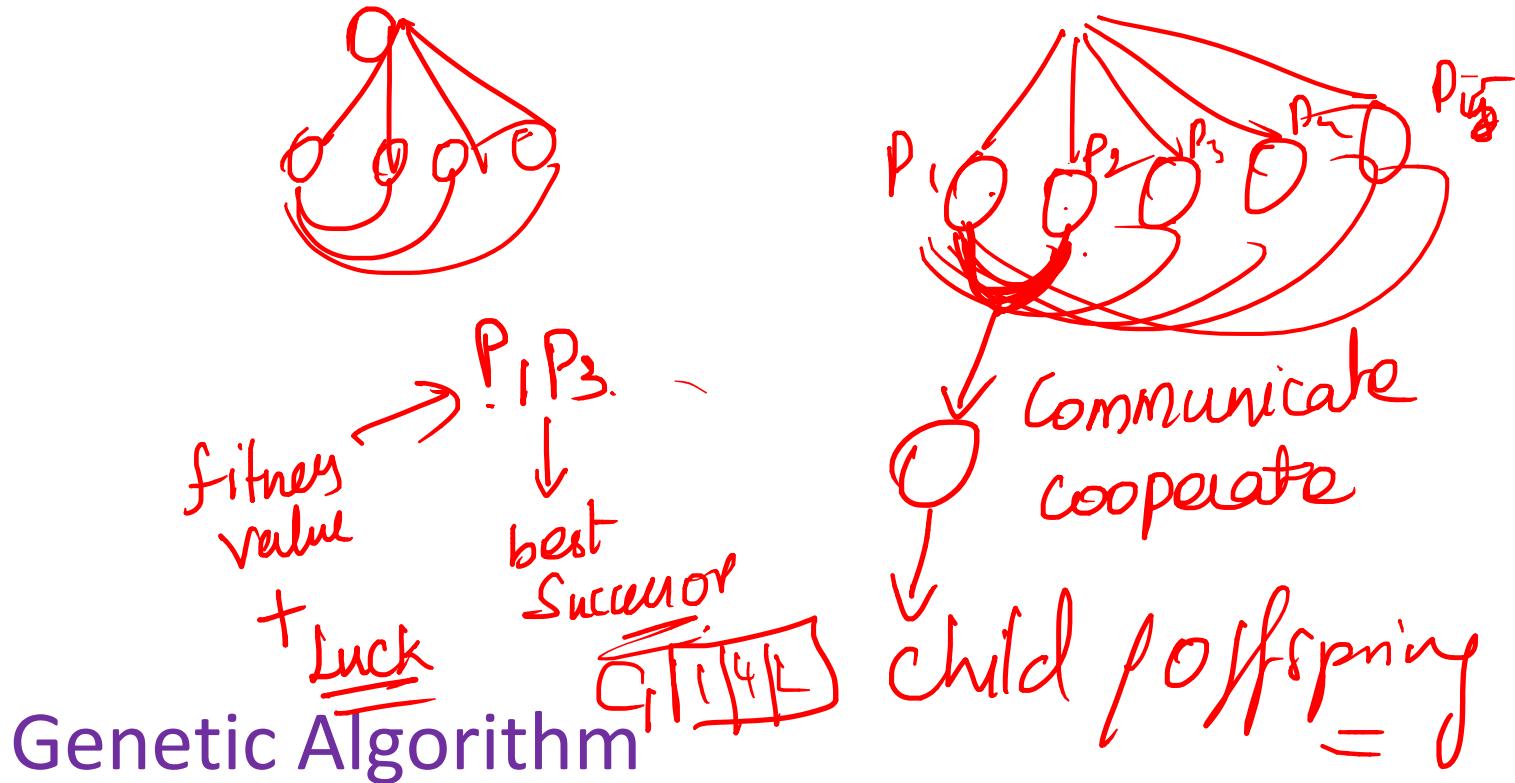


4	4	2	2	2	2		
---	---	---	---	---	---	--	--



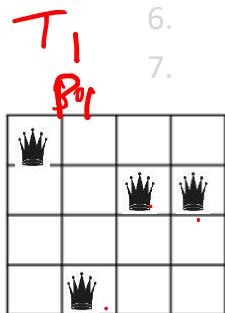
1	4	1	2	2	3		
---	---	---	---	---	---	--	--





# Genetic Algorithm

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
7. **Select 'k' random states – Initialization : k=4**  
 Evaluate the fitness value all states  
 If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops  
 Else, use roulette wheel mechanism to select pair/s

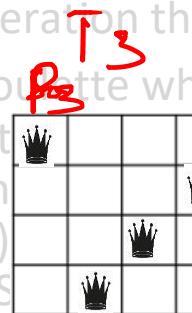


1 4 2 2

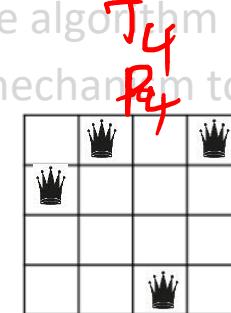
↓  
chromosome



2 1 3 4



1 4 3 2

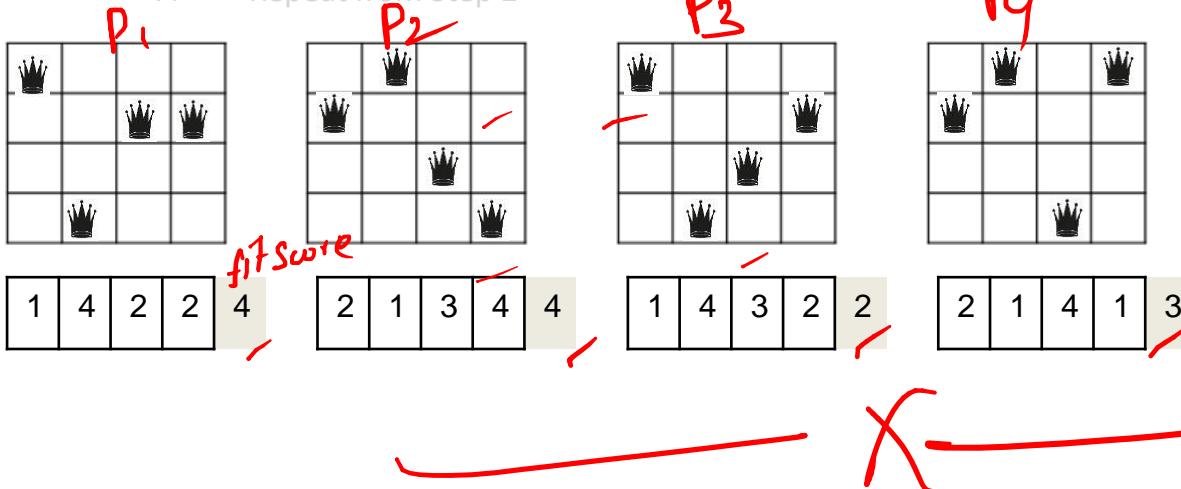


2 1 4 1

allowed to mutate  
Repeat from Step 2

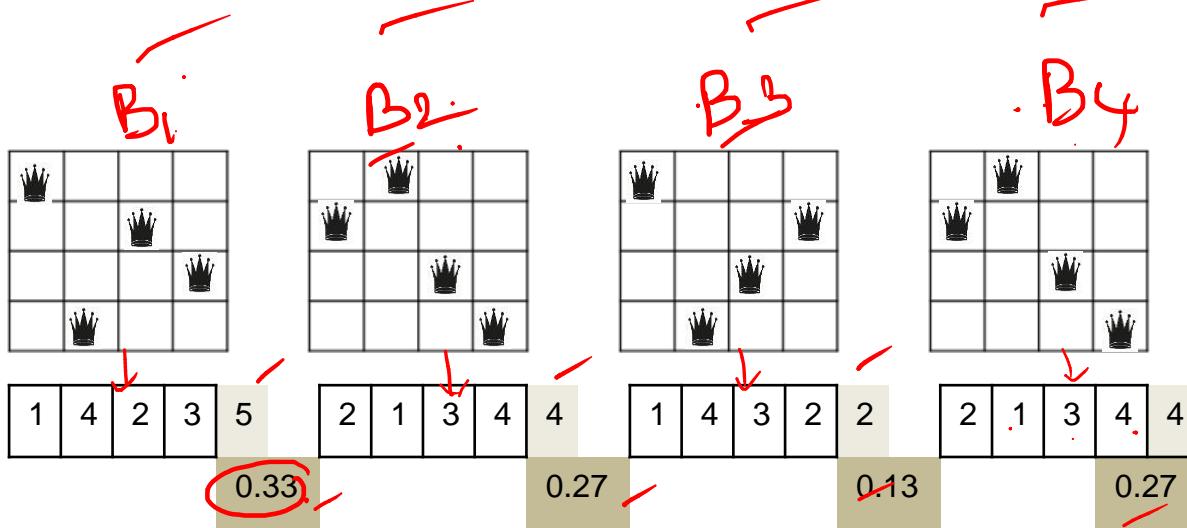
# Genetic Algorithm

1. Select 'k' random states – Initialization :  $k=4$
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens Threshold = 6
3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2

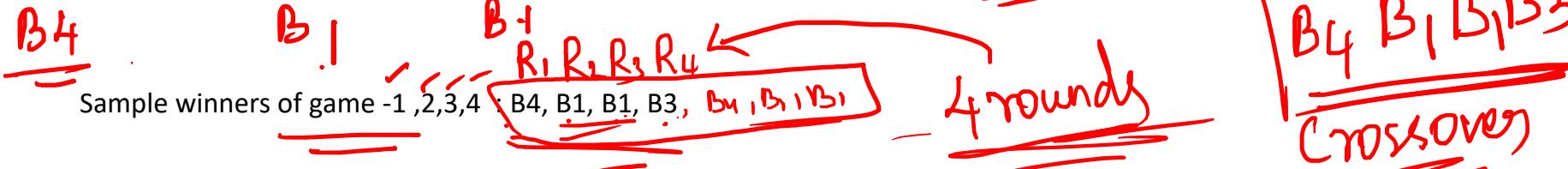


# Genetic Algorithm

Eg., use roulette wheel mechanism to select pair/s



2   1   3   4	1   4   2   3	1   4   2   3	1   4   3   2
---------------	---------------	---------------	---------------



# Genetic Algorithm

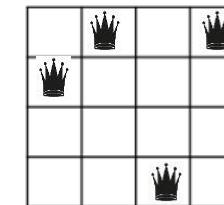
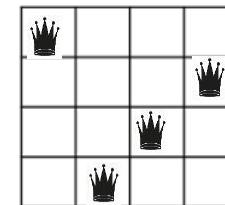
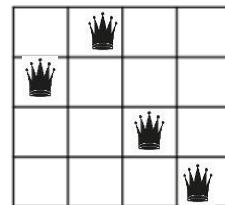
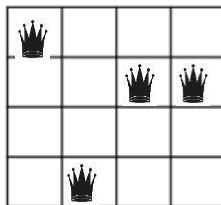
1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens  Threshold =
- ~~3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops~~
- ~~4. Else, use roulette wheel mechanism to select pair/s~~
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2

2	1	3	4
---	---	---	---

1	4	2	3
---	---	---	---

1	4	2	3
---	---	---	---

1	4	3
---	---	---



1	4	2	2	4
---	---	---	---	---

2	1	3	4	4
---	---	---	---	---

1	4	3	2	2
---	---	---	---	---

2	1	4	1	3
---	---	---	---	---

0.31

0.31

0.15

0.23

2	1	4	1
---	---	---	---

1	4	2	2
---	---	---	---

1	4	2	2
---	---	---	---

1	4	3	2
---	---	---	---

Sample winners of game -1 ,2,3,4 : B4, B1, B1, B3

= = = =

# Genetic Algorithm



Select 'k' random states – Initialization : k=4

Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens ⚙ Threshold = 6

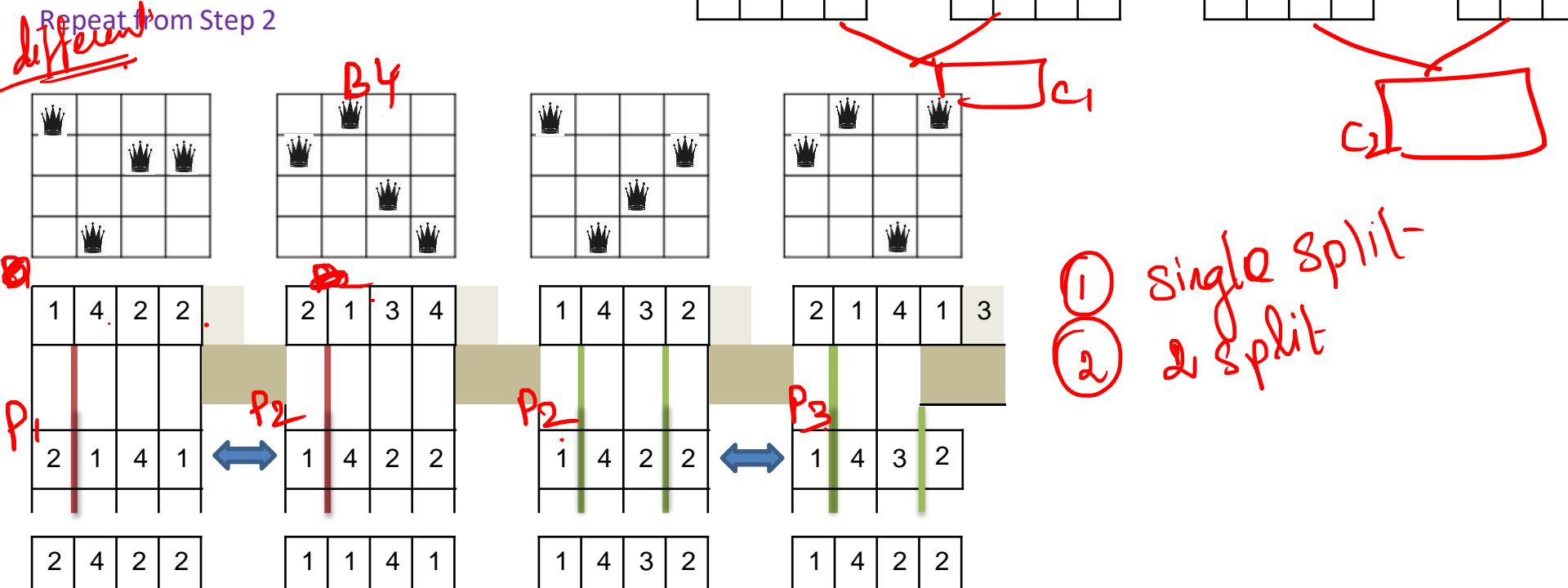
If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops

Else, use roulette wheel mechanism to select pair/s

Pairs selected produces new state (successor) by crossover

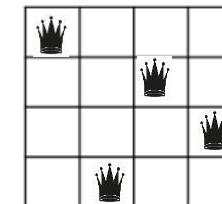
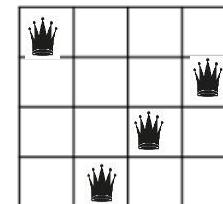
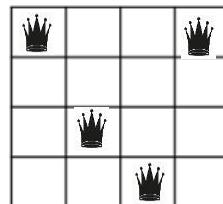
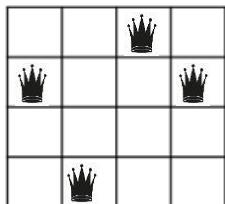
Successor is allowed to mutate

Repeat from Step 2



# Genetic Algorithm

1. Select 'k' random states – **Initialization : k=4**
2. Evaluate the fitness value all states : Maximizing function : No.of.Non-attacking pairs Queens  Threshold =
- ~~3. If anyone of the state's has achieved the threshold fitness value or threshold new states or no change is seen than previous iteration then the algorithm stops~~
4. Else, use roulette wheel mechanism to select pair/s
5. Pairs selected produces new state (successor) by crossover
6. Successor is allowed to mutate
7. Repeat from Step 2



2	4	2	2
---	---	---	---

1	1	4	1
---	---	---	---

1	4	3	2
---	---	---	---

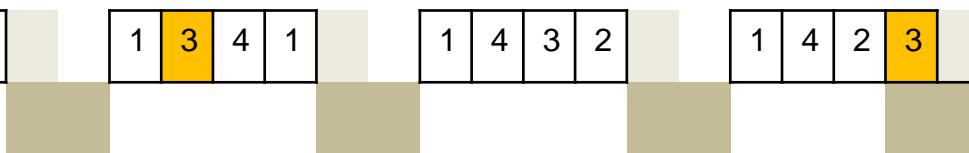
1	4	2	2
---	---	---	---

2	4	1	2
---	---	---	---

1	3	4	1
---	---	---	---

1	4	3	2
---	---	---	---

1	4	2	3
---	---	---	---



# Genetic Algorithm

## Techniques:

1. Design of the fitness function
2. Diversity in the population to be accounted
3. Randomization

## Application:

- Creative tasks
- Exploratory in nature
- Planning problem
- Static Applications

# Genetic Algorithm - Application in Games



## Source Credit:

<https://ai.googleblog.com/2018/03/using-evolutionary-automl-to-discover.html>

<https://eng.uber.com/deep-neuroevolution/>

# Crossover Operators in Genetic Algorithms: A Review

A.J. Umbarkar<sup>1</sup> and P.D. Sheth<sup>2</sup>

<sup>1</sup>Department of Information Technology, Walchand College of Engineering, India

E-mail: anantumbarkar@rediffmail.com

<sup>2</sup>Department of Master of Computer Applications, Government College of Engineering, Karad, India

E-mail: pranalisheth@gmail.com

## Abstract

The performance of Genetic Algorithm (GA) depends on various operators. Crossover operator is one of them. Crossover operators are mainly classified as application dependent crossover operators and application independent crossover operators. Effect of crossover operators in GA is application as well as encoding dependent. This paper will help researchers in selecting appropriate crossover operator for better results. The paper contains description about classical standard crossover operators, binary crossover operators, and application dependant crossover operators. Each crossover operator has its own advantages and disadvantages under various circumstances. This paper reviews the crossover operators proposed and experimented by various researchers.

## Keywords:

Evolutionary Algorithm, Genetic Algorithm, Crossover, Genetic Operators

## 1. INTRODUCTION

Genetic algorithm is a method of searching. It searches a result equal to or close to the answer of a given problem. New generation of solutions is created from solutions in previous generation. Basic strategy used in GA to create the best solutions/offspring is to crossover the parent genes. Various crossover techniques are built to get the optimum solution as early as possible in minimum generations. The selection of crossover operator has more impact on the performance of GA. The premature convergence [38] in GA can be avoided by selecting appropriate breeding operators. In this paper, the crossover operators are classified in three categories such as standard crossovers, binary crossovers and real/tree crossovers which are application dependant.

The Section 2 explains standard crossovers, which are application independent. Section 3 explains the binary crossovers with some modified crossovers to improve performance of GA. Section 4 explains the application dependant crossovers (real/tress). Section 5 discusses the findings of this review work.

## 2. STANDARD CROSSOVERS

### 2.1 1-POINT CROSSOVER

It is one of the simple crossover technique used for random GA applications. This crossover uses the single point fragmentation of the parents and then combines the parents at the crossover point to create the offspring or child.

1-Point crossover first selects two parents used for crossover and then randomly selects any crossover point  $p_i$  ( $i = 0$  to  $n-1$ ).

Two offspring are created by combining the parents at crossover point. A simple example is shown below which performs one point crossover and creates two parents.

Parent 1: 1 0 1 0 | 1 0 0 1 0

Parent 2: 1 0 1 1 | 1 0 1 1 0

Offspring 1: 1 0 1 0 | 1 0 1 1 0

Offspring 2: 1 0 1 1 | 1 0 0 1 0

In above example, point between 4<sup>th</sup> and 5<sup>th</sup> gene is selected as crossover point.

### 2.2 K-POINT CROSSOVER

It uses the random crossover point to combine the parents same as per 1-Point crossover. To provide the great combination of parents it selects more than one crossover points to create the offspring or child [1].

K-Point Crossover first selects the two parents used for crossover and then randomly select  $K$  crossover points  $P_{1i}$  to  $P_{k-1i}$  ( $i = 0$  to  $n - 1$ ). Two offspring are created by combining the parents at crossover point. A simple example is shown below which performs one point crossover and creates two parents.

Parent 1: 1 0 | 1 0 | 1 0 0 | 1 0

Parent 2: 1 1 | 0 0 | 1 0 1 | 1 0

Offspring 1: 1 0 | 0 0 | 1 0 0 | 1 0

Offspring 2: 1 1 | 1 0 | 1 0 1 | 1 0

In above example, the points between 2<sup>nd</sup> and 3<sup>rd</sup>, 4<sup>th</sup> and 5<sup>th</sup> and 7<sup>th</sup> and 8<sup>th</sup> gene are selected as crossover points.

### 2.3 SHUFFLE CROSSOVER

Shuffle Crossover helps in creation of offspring which have independent of crossover point in their parents. It uses the same 1-Point Crossover technique in addition to shuffle.

Shuffle Crossover selects the two parents for crossover. It firstly randomly shuffles the genes in the both parents but in the same way. Then it applies the 1-Point crossover technique by randomly selecting a point as crossover point and then combines both parents to create two offspring. After performing 1-point crossover the genes in offspring are then unshuffled in same way as they have been shuffled.

#### Select Shuffle Points

Parent 1: 1 1 1 0 1 0 0 1 0

Parent 2: 1 0 0 0 1 0 1 1 0

#### Shuffle genes as Shuffle Points

Parent 1: 0 1 0 1 1 0 1 1 0

Parent 2: 0 0 1 1 1 0 0 1 0

Select 1- Point Crossover Point

Parent 1: 0 1 0 1 | 1 0 1 1 0

Parent 2: 0 0 1 1 | 1 0 0 1 0

Perform 1-Point Crossover Point

Offspring 1: 0 1 0 1 | 1 0 0 1 0

Offspring 2: 0 0 1 1 | 1 0 1 1 0

Select unshuffled points same as shuffled points

Offspring 1: 0 1 0 1 1 0 0 1 0

Offspring 2: 0 0 1 1 1 0 1 1 0

Unshuffled the genes in Offspring

Offspring 1: 1 1 0 0 1 0 0 1 0

Offspring 2: 1 0 1 0 1 0 1 1 0

## 2.4 REDUCED SURROGATE CROSSOVER

Reduced Surrogate Crossover minimizes the unwanted crossover operations in case of the parents having same genes. In these cases the Reduced Surrogate Crossover first checks for the individual genes in the parents. It creates list of all possible crossover points where the genes of the both parents are different.

After performing this check, if no crossover point is there then no action is taken. But in case, if parents are differing in more than 1 gene then it keeps the list of all crossover points. It then randomly selects one crossover point from the list and performs 1-point crossover to create the offspring.

## 2.5 UNIFORM CROSSOVER

Uniform crossover provides the uniformity in combining the bits of both parents. It performs this operation of swapping bits in the parents to be included in the offspring by choosing a uniform random real number  $u$  (between 0 to 1).

Uniform crossover selects the two parents for crossover. It creates two child offspring of  $n$  genes selected from both of the parents uniformly. The random real number decides whether the first child select the  $i^{\text{th}}$  genes from first or second parent [1].

Parent 1: 1 1 1 0 1 0 0 1 0

Parent 2: 1 0 0 0 1 0 1 1 0

Offspring 1: 1 1 0 0 1 0 1 1 0

Offspring 2: 1 0 1 0 1 0 0 1 0

## 2.6 AVERAGE CROSSOVER (AX)

Average Crossover is the value based crossover technique. It uses two parents to perform crossover and creates only one offspring [1]. Average Crossover creates one offspring from taking average of the two parents. It selects two parents as X and Y and generate the child Z as follows: each gene in a child is taken by averaging genes from both parents.

Parent 1: 5 3 3 2 3 9 7 6 5

Parent 2: 5 4 7 6 5 2 6 1 3

Offspring 1: 5 3 5 4 4 5 6 3 4

## 2.7 DISCRETE CROSSOVER (DC)

Discrete Crossover uses the random real number to create one child from two parents.

Unlike the uniform crossover only one child is generated in Discrete Crossover. It selects two parents as X and Y and generate the child Z such that it select genes of both the parents uniformly. The random real number decides from which parent to take the genes for child. [1]

Parent 1: 1 1 1 0 1 0 0 1 0

Parent 2: 1 0 0 0 1 0 1 1 0

Offspring 1: 1 1 0 0 1 0 1 1 0

## 2.8 FLAT CROSSOVER (FC)

Flat Crossover uses the random real number to create one child from two parents.

Same as of Discrete Crossover it selects the genes from parent based on uniform random real number. But the selected random real number should be a subset of set having the minimum and maximum of the genes of the both parents. It selects two parents as X and Y and generate the child Z such that it selects random real number which is either min or max from genes in both parents and then assign this real number in child gene.

## 2.9 HEURISTIC CROSSOVER/INTERMEDIATE CROSSOVER (HC/IC)

Heuristic Crossover creates one child offspring from two parents. It uses the  $\alpha \in <0, 1>$  assuming  $x_i \leq y_i$ . For each gene in the child it select the uniform random real number  $\alpha$ . And the child gene is calculated from above equation.

$$x_i(t+1) = x_i(t) + \alpha (y_i(t) - x_i(t))$$

Parameter  $\alpha$  may be of constant value equal to 0.5 or may be selected by a draw from interval  $<0, 1>$  (row: 5).

## 2.10 STATISTICS-BASED ADAPTIVE NON-UNIFORM CROSSOVER (SANUX)

It is based on the concepts of intrinsic attribute and extrinsic tendency of valuing allele for a gene locus. In optimal solution (encoded in binary string) of a given problem, for a gene locus if its allele is 1 it is called 1-intrinsic, if its allele is 0 it is called 0-intrinsic, otherwise if its allele either 0 or 1 it is called neutral. During the running of a GA, for a gene locus, if the frequency of 1's in its alleles over, the population tends to increase with time (generation), it called 1-inclined; if the frequency of 1's tends to decrease, it is called 0-inclined; otherwise it is called non-inclined. [13]

Usually and hopefully as the GA progresses, the gene loci which are 1-intrinsic will appear to be 1-inclined. SANUX makes use of this convergence information as feedback information to direct the crossover by adjusting the swapping probability of each locus.

Now during the evolution of the GA, after generation of new population the distribution of 1's  $f_1(i, t)$  from each locus over the population is calculated. Then SANUX operation is performed. After applying SANUX, the mask is generated bit by bit by

flipping a coin biased to generate “1” with probability  $p_s(i, t)$ . Finally, the generated mask is used to guide the crossover in the same way it guides the traditional uniform crossover.

1's Frq. in loci:	0.4 0.2 0.6 0.9 0.9 0.2
Calculating:	
Swapping prob:	0.4 0.2 0.4 0.1 0.1 0.2
biased flipping:	
Created mask:	1 0 1 0 0 0
Applying mask:	
Parent P1:	0 1 0 1 1 1
Parent P2:	1 1 1 1 1 0
Swapping:	
Child C1:	1 1 1 1 1 1
Child C2:	0 1 0 1 1 0
	SANUX

### 3. BINARY CROSSOVERS

#### 3.1 RANDOM RESPECTFUL CROSSOVER (RRC)

It selects two parents for crossover and offspring is generated based on the similarity vector of the parents. It first creates similarity vector  $S_{ab} = (S_{1ab}, \dots, S_{nab})$  such that if both genes of parents have the same values then the similarity vector contains the values of the parent else similarity vector contain null value for that gene [1].

After creation of similarity vector  $S$ , two children are created according to the values of the similarity vector. If similarity vector contains 1 then gene of both children is set to 1 and if it contains 0 then genes of both children are set to 0. Apart from this if similarity vector contains null value for any gene then the child gene is selected by taking a uniform random real number. If this number is  $< 0.5$  then 1 is stored otherwise 0 is stored.

Parent 1:	1 1 1 0 1 0 0 1 0
Parent 2:	1 0 0 0 1 0 1 1 0
Offspring 1:	1 1 0 0 1 0 1 1 0
Offspring 1:	1 0 0 0 1 0 0 1 0

This algorithm duplicates genes of parents in an offspring at every position wherever they are identical.

#### 3.2 MASKED CROSSOVER (MX)

The MX operator uses a mask vector to determine which bits of which parent are inherited by the offspring. The first step is the duplication of the bits of the parents. The bits of the first parent are copied to the first offspring and, accordingly, of the second parent to the second offspring. In the second step, the offspring exchange bits among each other at those positions where the mask vectors of the parent were equal to 1, indicated domination of that parent at that position and the mask vectors of the other parent were equal to 0.

The mask vectors are initiated in  $P(0)$  randomly. During every iteration of GA, the mask vectors are inherited by each offspring from its parent. Then the mask vectors of the offspring as well as the parents undergo modification. The modification

process is based on comparison of fitness of offspring and the parents. If good offspring are created, the masks of the parents do not need to be modified and the masks of the offspring may be very similar to those of the parents. In a situation where bad offspring were created the masks of the parents as well as of the offspring need to be modified.

#### 3.3 1- BIT ADAPTATION CROSSOVER (1BX)

In the 1 BX method, the last bit of the solution vector is reserved for the code of one of the two of the applied crossover operators. Assuming “0” corresponds to Uniform Crossover (UX) operator and “1” corresponds to 2-Point Crossover (2-PX) operator, the choice of one of them is made according to the rule: if the last bit of the parents is off the same value then choose the operator indicated by this bit. Otherwise chooses the operator through the selection by a draw i.e. select the uniform random real number from 0 to 1. If this value is  $< 0.5$ , then Uniform Crossover is performed otherwise 2-Point Crossover is used.

Application of the described crossover scheme combines the choice of the operator with the solution vector. Moreover, this choice is carried out separately for each parent pair; hence this scheme is called local adaptation. Global adaptation version has also presented, but as it was emphasized by the author, significantly worse results were obtained by its application.

#### 3.4 MULTIVARIATE CROSSOVER (MC)

MC divides the whole parent string into the q substrings. The crossover is performed based on random value selected for each substring. If this value is  $< P_c$  then crossover is performed other only parent genes are copied into child offspring. It performs the standard 1-Point Crossover for the substring when the condition is satisfied [1].

The most fundamental difference between the MC operator and other operators using variable-to-variable recombination is that the answer to the question “whether to crossover” is checked in the MC method separately for each substring. As for other operators, the answer to that question refers to parent vector as a whole.

#### 3.5 HOMOLOGOUS CROSSOVER (HX)

The HX operator is based on the standard K-Point Crossover operator. Introduced modification relies on the fact that only strings of bits which are at least of a certain length or of an admissible degree of similarity are allowed to participate in crossover. Determination of the degree of similarity is based on the XOR operator.

This strategy is aimed at transferring strings with specified parameters to the next generation. In HX, the value of  $o$  and  $r$  is determined a priori as constant or dynamically changed in the GA run.

#### 3.6 COUNT PRESERVING CROSSOVER (CPC)

The CPC operator carries out its task by assuming that the number of bits equal to “1” in every chromosome in the initial population  $P(0)$  is the same.

CPC may guarantee the preservation of the constant number of bits equal to “1” due to application of two lists noting the differences between the parents. List  $L_{up}$  includes positions of those bits, on which there are differences between the parents, but the first parent at a given position holds a bit equal to “1” and the second equal to “0”. List  $L_{down}$  similarly notes the positions of differences, but the first parent at a given position holds a bit equal to “0” and the second equal to “1”. The offspring creation process which makes use of those lists is based on the exchange of bits between the offspring at those positions which, are indicated by subsequent element pairs from lists  $L_{up}$  and  $L_{down}$ .

Number of elements in  $L_{up}$  and in  $L_{down}$  is the same, which is a direct result of the assumption, that the number of bits equal to “1” is constant for all chromosomes in  $P(0)$ .

### 3.7 ELITIST CROSSOVER (EX)

In the standard genetic algorithm, the selection process is always preceded by the crossover process. In the EX method, both processes are integrated. During the first step of the entire population is randomly shuffled. Then from each successive pair of parental vectors, two new vectors are created by crossover. From a ‘family’ created, two best vectors are singled out and implemented as offspring to the next population.

Application of elitist selection in the traditional way that is on the level of the entire population may often be the reason for the premature convergence of the algorithm. An EX elitist selection applied on the “family” level eliminates this danger according to the authors.

### 3.8 SCANNING CROSSOVER (SCAN), UNIFORM SCANNING CROSSOVER (U-SCAN), OCCURRENCE BASED SCANNING CROSSOVER (OB-SCAN) FITNESS BASED SCANNING CROSSOVER (FB-SCAN)

Depending on the heuristics applied to scanning crossover, three variations are: uniform scanning crossover (U-Scan), occurrence based scanning crossover (OB-Scan), and fitness based scanning crossover (FB-Scan).

Increasing the number of parents in OB-Scan leads to a higher probability for the major alleles in the population to exist in every offspring, which will cause the population to concentrate on a certain region and thus lose the diversity rapidly [18], [19]. This situation is called premature convergence, a result of unbalanced exploitation and exploration.

### 3.9 SELF-ADAPTIVE SIMULATED BINARY CROSSOVER (SBX)

A self-adaptive procedure for updating the distribution index used in the simulated binary crossover or SBX operator which is a commonly-used real-parameter recombination operator. This crossover is also good for multi-objective optimization problem.

### 3.10 OTHER BINARY CROSSOVER

Some binary crossover is proposed by researchers are - Circle-ring crossover, Sufficient Exchanging crossover [12],

Adaptive crossovers [12], Diagonal crossover [21, 22], Best combinatorial crossover (BCX) and Hybridization crossover (HX) [14].

## 4. APPLICATION DEPENDANT CROSSOVERS (REAL/TREE)

### 4.1 CROSSOVER FOR TSP PROBLEMS

#### 4.1.1 Order Based Crossover (OBX):

The order based crossover operator selects at random several positions in one of the parent tours, and the order of the cities in the selected positions of this parent is imposed on the other parent to produce one child. The other child is generated in an analogous manner for the other parent [1].

#### 4.1.2 Modified Order Crossover (MOC):

A randomly chosen crossover point divides the parent strings in left and right substrings. The right substrings of the parent s1 and s2 are selected. After selection of cities the process is the same as the order crossover. Only difference is that instead of selecting random several positions in a parent tour all the positions to the right of the randomly chosen crossover point are selected [1].

For example with the following parents and crossover point

$$s1 = (1 \ 2 \ 3 \ 4 \ | \ 6 \ 9 \ 8 \ 5 \ 7) \text{ and}$$

$$s2 = (2 \ 1 \ 9 \ 8 \ | \ 5 \ 6 \ 3 \ 7 \ 4)$$

After position selection

$$s1 = (1 \ 2 \ * \ * \ * \ 9 \ 8 \ * \ *) \text{ and}$$

$$s2 = (2 \ 1 \ * \ * \ * \ * \ 3 \ * \ 4)$$

Now obtain the generated pair of children as

$$b1 = (1 \ 2 \ 5 \ 6 \ 3 \ 9 \ 8 \ 7 \ 4) \text{ and}$$

$$b2 = (2 \ 1 \ 6 \ 9 \ 8 \ 5 \ 3 \ 7 \ 4)$$

Clearly this method allows only the generation of valid strings.

#### 4.1.3 Partially-Mapped Crossover (PMX):

It transmits ordering and values information from the parent strings to the offspring. A portion of one parent string is mapped onto a portion of the other parent string and the remaining information is exchanged. Consider, for example, the following two parents: (1 2 3 4 5 6 7 8) and (3 7 5 1 6 8 2 4). The PMX operator creates an offspring in the following way. It begins by selecting uniformly at random two cut points along the strings, which represent the parents. Suppose, for example, that the first cut point is selected between the third and the fourth string element, and the second one between the sixth and the seventh string element. Hence, (1 2 3 | 4 5 6 | 7 8) and (3 7 5 | 1 6 8 | 2 4). The substrings between the cut points are called the mapping sections. In our example, they define the mappings 4  $\leftrightarrow$  1, 5  $\leftrightarrow$  6, and 6  $\leftrightarrow$  8. Now the mapping section of the first parent is copied into the second offspring, and the mapping section of the second parent is copied into the first offspring: offspring 1: (x x | 1 6 8 | x x) and offspring 2: (x x | 4 5 6 | x x). Then offspring  $i$  ( $i = 1, 2$ ) is filled up by copying the elements of the  $i^{\text{th}}$  parent. In case, a number is already present in the offspring it is replaced according to the mappings [2].

For example, the first element of offspring 1 would be a 1, like the first element of the first parent. However, there is already a 1 present in offspring 1. Hence, because of the mapping  $1 \leftrightarrow 4$  choose the first element of offspring 1 to be a 4. The second, third and seventh elements of offspring 1 can be taken from the first parent. However, the last element of offspring 1 would be an 8, which is already present. Because of the mappings  $8 \leftrightarrow 6$ , and  $6 \leftrightarrow 5$ , it is chosen to be a 5. Hence, offspring 1:  $(4\ 2\ 3\ | 1\ 6\ 8\ | 7\ 5)$ . Analogously, we find offspring 2:  $(3\ 7\ 8\ | 4\ 5\ 6\ | 2\ 1)$ . The absolute positions of some elements of both parents are preserved.

#### **4.1.4 Modified Partially-Mapped Crossover (MPMX):**

Modified PMX (MPMX) crossover operator was proposed (independently) by Brown [39] in the late 80's. The MPMX operator initially partitions the parents' solution strings and the offspring strings into three sections (left, middle and right). These sections are randomly created through the selection of two random crossover points that will be used for both the parents and offspring for this instance of the crossover in the GA. Stage two provides the offspring with the middle section of its solution string. This is the donated middle section of parent 1. The third stage, is the insertion of elements into the left and right sections of the offspring. This is accomplished using parent 2 as the donator. Corresponding positions in the parents donate elements to the offspring, provided they have not already been donated by parent 1. The final stage is to complete the offspring using a random permutation of the elements not yet allocated to the offspring over the previous stages. The following example illustrates the:

Parent 1 -  $(0\ 8\ | 4\ 5\ 6\ 7\ | 1\ 2\ 3\ 9)$

Parent 2 -  $(6\ 7\ | 1\ 2\ 4\ 8\ | 3\ 5\ 9\ 0)$

Offspring stage1-  $(- - | - - - | - - -)$

Offspring stage2-  $(- - | 4\ 5\ 6\ 7\ | - - -)$

Offspring stage3-  $(- - | 4\ 5\ 6\ 7\ | 3\ - 9\ 0)$

Offspring stage4-  $(8\ 1\ 4\ 5\ 6\ 7\ 3\ 2\ 9\ 0)$

#### **4.1.5 Cycle Crossover (CX) [3]:**

It attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents. For example, consider again the parents  $(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$  and  $(2\ 4\ 6\ 8\ 7\ 5\ 3\ 1)$ . Now choose the first element of the offspring equal to either the first element of the first parent string or the first element of the second parent string. Hence, the first element of the offspring has to be a 1 or a 2. Suppose choose it to be 1,  $(1\ * * * * * * *)$ . Now consider the last element of the offspring. Since this element has to be chosen from one of the parents, it can only be an 8 or a 1. If a 1 were selected, the offspring would not represent a legal individual. Therefore, an 8 is chosen,  $(1\ * * * * * * 8)$ . It finds that the fourth and the second element of the offspring also have to be selected from the first parent, which results in  $(1\ 2\ * 4\ * * * 8)$ . The positions of the elements chosen up to now are said to be a cycle. Now consider the third element of the offspring. This element it may choose from any of the parents. Suppose that we select it to be from parent 2. This implies that the fifth, sixth and seventh elements of the offspring also have to be chosen from the second parent, as they form another cycle. Thus, we find the following

offspring:  $(1\ 2\ 6\ 4\ 7\ 5\ 3\ 8)$ . The absolute positions, of on average half the elements of both parents are preserved.

#### **4.1.6 Order Crossover Operator (OX1):**

It constructs an offspring by choosing a substring of one parent and preserving the relative order of the elements of the other parent. For example, consider the following two parent strings:  $(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$  and  $(2\ 4\ 6\ 8\ 7\ 5\ 3\ 1)$ , and suppose that we select a first cut point between the second and the third bit and a second one between the fifth and the sixth bit. Hence,  $(1\ 2\ | 3\ 4\ 5\ | 6\ 7\ 8)$  and  $(2\ 4\ | 6\ 8\ 7\ | 5\ 3\ 1)$ . The offspring are created in the following way. Firstly, the string segments between the cut point are copied into the offspring, which give  $(* * | 3\ 4\ 5\ | * * *)$  and  $(* * | 6\ 8\ 7\ | * * *)$ . Next, starting from the second cut point of one parent, the rest of the elements are copied in the order in which they appear in the other parent, also starting from the second cut point and omitting the elements that are already present. When the end of the parent string is reached, we continue from its first position. In our example, this gives the following children:  $(8\ 7\ | 3\ 4\ 5\ | 1\ 2\ 6)$  and  $(4\ 5\ | 6\ 8\ 7\ | 1\ 2\ 3)$  [4].

#### **4.1.7 Order-based Crossover (OX2):**

OX2 was suggested in connection with schedule problems, is a modification of the OX1 operator. The OX2 operator selects at random several positions in a parent string, and the order of the elements in the selected positions of this parent is imposed on the other parent. For example, consider again the parents  $(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$  and  $(2\ 4\ 6\ 8\ 7\ 5\ 3\ 1)$ , and suppose that in the second parent in the second, third and sixth positions are selected. The elements in these positions are 4, 6 and 5 respectively. In the first parent, these elements are present at the fourth, fifth and sixth positions. Now the offspring are equal to parent 1 except in the fourth, fifth and sixth positions:  $(1\ 2\ 3\ * * * 7\ 8)$ . We add the missing elements to the offspring in the same order in which they appear in the second parent. This results in  $(1\ 2\ 3\ 4\ 6\ 5\ 7\ 8)$ . Exchanging the role of the first parent and the second parent gives, using the same selected positions,  $(2\ 4\ 3\ 8\ 7\ 5\ 6\ 1)$  [5].

The position-based crossover operator (POS), which was also suggested in connection with schedule problems, is a second modification of the OX1 operator. It also starts with selecting a random set of positions in the parent strings. However, this operator imposes the position of the selected elements on the corresponding elements of the other parent. For example, consider the parents  $(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)$  and  $(2\ 4\ 6\ 8\ 7\ 5\ 3\ 1)$ , and suppose that the second, third and sixth positions are selected. This leads to the following offspring:  $(1\ 4\ 6\ 2\ 3\ 5\ 7\ 8)$  and  $(4\ 2\ 3\ 8\ 7\ 6\ 5\ 1)$ .

#### **4.1.8 Voting Recombination Crossover operator (VR):**

It can be seen as a P-sexual crossover operator, where  $p$  is a natural number greater than, or equal to, 2. It starts by defining a threshold, which is a natural number smaller than, or equal to  $p$ . Next, for every;  $i \in \{1, 2, \dots, N\}$  the set of  $i^{\text{th}}$  elements of all the parents is considered. If in this set an element occurs at least the threshold number of times, it is copied into the offspring. For example, if we consider the parents ( $p = 4$ )  $(1\ 4\ 3\ 5\ 2\ 6)$ ,  $(1\ 2\ 4\ 3\ 5\ 6)$ ,  $(3\ 2\ 1\ 5\ 4\ 6)$ ,  $(1\ 2\ 3\ 4\ 5\ 6)$  and we define the threshold to be equal to 3 we find  $(1\ 2\ x\ x\ x\ 6)$ . The remaining positions of the offspring are filled with mutations. Hence, our example might result in  $(1\ 2\ 4\ 5\ 3\ 6)$  [6].

#### 4.1.9 Maximal Preservation crossover (MPX):

The MPX operator was developed by Gorges-Schleuter and Mülhelenbein [42] in 1988 specifically for the TSP. It is closely related to the PMX crossover operator [48]. MPX operates by initially selecting a random substring (the TSP this is a subtler) from the first parent (called the donor). This subtour is usually defined as being a tour with string length less than or equal to the TSP problem size  $n$  divided by 2. A minimum subtour length is also set, typically at 10 elements (unless the TSP problem size is very small), as substrings that are very short are ineffective and substrings that are too large do not allow for meaningful variation. Selecting appropriate sized substrings provides a suitable means for parents to transmit significant loci information to the offspring. The second stage of MPX is to remove the elements currently in the offspring from the second parent. Then the remaining elements are inserted into the offspring, the first parent's substring having been placed at the start of the offspring and the remaining free elements of the offspring being filled by the clean parent 2 strings. This three stage operation of MPX is illustrated in following example:

Parent 1 - (1 4 3 5 2 6)  
 Parent 2 - (1 2 4 3 5 6)  
 Offspring (1 4 3 x x x)  
 Cleaned Parent 2 - (- 2 - - 5 6)  
 Offspring (1 4 3 2 5 6)

With regard to the MPX and its application to the TSP, although the MPX prevents invalid tour generation in the offspring, they are liable to be produced with few building blocks being inherited from both parents due to the cleaning of the second parent's string prior to completing the offspring strings.

#### 4.1.10 Masked crossover (MkX):

The Masked Crossover (MkX) technique was first proposed by Louis and Rawlins in 1991 [43] as a crossover operator which would efficiently operate in the combinatorial logic design problem area rather than as a combinatorial optimization technique. MkX [48] attempts to impart loci information from parent to offspring in a more effective manner than previous crossover methods. Louis and Rawlins state that MkX tries to preserve schemas identified by the masks and they identify this as one of their key goals [43]. The MkX operator assigns each parent a mask that biases crossover. Once these masks have been positioned then the operation is as following:

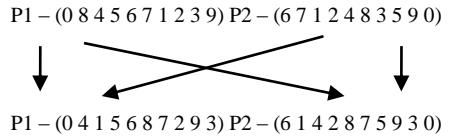
1. Copy Parent1 to Offspring1 and Parent2 to Offspring2
2. For ( $i$  from 1 to string-length)
  - if  $\text{Mask2}_i = 1$  and  $\text{Mask1}_i = 0$
3. Copy the  $i^{\text{th}}$  bit from Parent2 to Offspring1
  - if  $\text{Mask1}_i = 1$  and  $\text{Mask2}_i = 0$
4. Copy the  $i^{\text{th}}$  bit from Parent1 to Offspring2

The offspring of MkX also require masks, should they be selected to be parents in another generation. The masks are normally provided to the offspring by the parents. Typically the parent that is designated the dominant parent is called Parent1 the dominant parent with respect to Offspring1 as Offspring1 inherits Parent1's bits unless Parent2 feels strongly ( $\text{Mask2}_i = 1$ ) and Parent1 does not ( $\text{Mask1}_i = 0$ ). A number of mask rules are

also defined by Louis and Rawlins. Two of which are used when the simple rule of assigning masks from dominant parent to offspring don't apply. Chan [47] notes that the MkX is an ineffective crossover operator for the TSP as it fails to preserve the ordering of the solutions. Validity of solution is problematic and (in conjunction with the selected mutation operator) typically involves a repair or penalty function.

#### 4.1.11 Position crossover (PX):

The Position Crossover (PX) operator was developed by Syswerda in 1991 [5], [48]. PX was later evaluated by Barbulescu [44] where she examined and compared PX's operation to similar operators for scheduling problems. This crossover technique is closely related to OX and OX2 crossover techniques. PX operates by selecting several random locations along the parent strings. The elements are then inherited by the offspring in the order that they occur in the first parent (P). The remaining elements required to complete the offspring (O) are donated by the second parent (with the elements donated by the first parent omitted) in the order that they appear in the second parent. Each step of the operation is illustrated as follows:



#### 4.1.12 Complete Subtour Exchange Crossover:

The complete subtour exchange crossover (CSEX) operator is designed to operate with the path representation. CSEX was proposed by Katayama and Narihisa [45] to be used specifically for permutation problems (such as the TSP). The philosophy behind CSEX [48] is to encourage the offspring to inherit as many good traits (substance) from the parents as possible. CSEX enumerates substance that have the same direction (or reversed direction) on two permutations as common substance.

Parent 1 - (0 1 2 3 4 5 6 7 8 9)  
 Parent 2 - (4 9 7 6 5 0 8 2 1 3)  
 Offspring 1- (0 2 1 3 4 5 6 7 8 9)  
 Offspring 2- (0 1 2 3 4 7 6 5 8 9)  
 Offspring 3- (0 2 1 3 4 7 6 5 8 9)  
 Offspring 4- (4 9 5 6 7 0 8 2 1 3)  
 Offspring 5- (4 9 7 6 5 0 8 1 2 3)  
 Offspring 6- (4 9 5 6 7 0 8 1 2 3)

In above see an example of CSEX in operation. The common subtours are 12 and 5 6 7 in parent 1, and 7 6 5 and 2 1 in parent 2. It should be noted that CSEX does not include the subtour 1 2 3 from parent 1 and 2 1 3 from parent 2 as they are not the same or symmetrical. CSEX by allowing only the same (or symmetrical) subtours can enumerate all the common subtours with  $O(n)$  time. Having selected the common subtours, the offspring are produced by inverting the common subtours from the parent. In the example, parent 1 produces offspring 1, 2 and 3 by inverting a common subtour for each offspring. This is then repeated for parent 2 which produces offspring 4, 5 and 6. Once all the offspring are produced they are evaluated for fitness and the two fittest offspring survive to the next generation.

#### **4.1.13 Heuristic crossover (HX):**

It is worth noting that the previous crossover operators did not exploit the distances between the cities (i.e. the length of the edges). In fact, it is a characteristic of the genetic approach to avoid any heuristic information about a specific application domain, apart from the overall evaluation or fitness of each chromosome. This characteristic explains the robustness of the genetic search and its wide applicability [9] [10] [48].

However, some researchers departed from this line of thinking and introduced domain-dependent heuristics into the genetic search, to create “hybrid” genetic algorithms. They have sacrificed robustness over a wide class of problems, for better performance on a specific problem. The heuristic crossover HX is an example of this approach and can be described as follows:

1. Pick a random starting city at one of the two parents.
2. Compare the edges leaving the current city in both parents and select the shorter edge.
3. If the shorter parental edge introduces a cycle in the partial tour, then extend the tour with a random edge that does not introduce a cycle.
4. Repeat Steps 2 and 3 until all cities are included in the tour.

#### **4.1.14 Edge Recombination crossover (ER):**

Quite often, the alternate edge operator introduces many random edges in the offspring, particularly the last edges, when the choices for extending the tour are limited. Since the offspring must inherit as many edges as possible from the parents, the introduction of random edges should be minimized. The edge recombination operator reduces the myopic behavior of the alternate edge approach with a special data structure called the “edge map”.

Basically, the edge map maintains the list of edges that are incident to each city in the parent tours, and lead to cities not yet included in the offspring. Hence, these edges are still available for extending the tour, and are said to be active. The strategy is to extend the tour by selecting the edge that leads to the city with the minimum number of active edges. In case of equality between two or more cities, one of these cities is selected at random. With this strategy, the approach is less likely to get trapped in a “dead end”, namely, a city with no remaining active edges that require the selection of a random edge [8] [48].

For tours of 13564287 and 14236578 (path representation), the initial edge map is shown below:

City 1 has edges to: 3 4 7 8  
 City 2 has edges to: 3 4 8  
 City 3 has edges to: 1 2 5 6  
 City 4 has edges to: 1 2 6  
 City 5 has edges to: 3 6 7  
 City 6 has edges to: 3 4 5  
 City 7 has edges to: 1 5 8  
 City 8 has edges to: 1 2 7

Fig.1. Edge Map

Let us assume that city 1 is selected as the starting city. Accordingly, all edges incident to city 1 must be deleted from the initial edge map. From city 1, we can go to city 3, 4, 7 or 8.

City 3 has three active edges, while cities 4, 7 and 8 have two active edges, as shown by the edge map (a) in Fig.1. Hence, a random choice is made between cities 4, 7 and 8. We assume that city 8 is selected. At 8, we can go to cities 2 and 7. As indicated in the edge map (b), city 2 has two active edges and city 7 only one, so the latter is selected. From 7, there is no choice, but to go to city 5. From this point, edge map (d) offers a choice between cities 3 and 6 with two active edges. Let us assume that city 6 is randomly selected. From city 6, we can go to cities 3 and 4, and edge map (e) indicates that both cities have one active edge. We assume that city 4 is randomly selected. Finally, from city 4 we can only go to city 2, and from city 2 we must go to the city 3.

Some modified edge recombination crossover are Edge Exchange Crossover (EXX) and Edge Assembly Crossover (EAX) [31].

#### **4.1.15 Alternate Edges Crossover:**

It is a good introduction to other edge-preserving operators. Here, a starting edge  $(i, j)$  is selected at random in one parent. Then, the tour is extended by selecting the edge  $(j, k)$  in the other parent. The tour is progressively extended in this way by alternatively selecting edges from the two parents. When an edge introduces a cycle, the new edge is selected at random (and is not inherited from the parents) [9] [48].

In following example, an offspring is generated from two parent chromosomes that encode the tours 13564287 and 14236578, respectively, using the adjacency representation. Here, edge (1, 4) is first selected in parent 2, and city 4 in position 1 of parent 2 is copied at the same position in the offspring. Then, the edges (4, 2) in parent 1, (2, 3) in parent 2, (3, 5) in parent 1 and (5, 7) in parent 2 are selected and inserted in the offspring. Then, edge (7, 1) is selected in parent 1. However, this edge introduces a cycle and a new edge incident to 7 and to a city not yet visited is selected at random. Let us assume that (7, 6) is chosen. Then, edge (6, 5) is selected in parent 2, but it also introduces a cycle. At this point, (6, 8) is the only selection that does not introduce a cycle. Finally, the tour is completed with edge (8, 1).

parent 1: 3 8 5 2 6 4 1 7

parent 2: 4 3 6 2 7 5 8 1

offspring: 4 3 5 2 7 8 6 1

The Alternate Edges Crossover

The final offspring encodes the tour 14235768, and all edges in the offspring are inherited from the parents, apart from the edges (7, 6) and (6, 8). In the above description, an implicit orientation of the parent tours is assumed. For symmetric problems, the two edges that are incident to a given city can be considered. In the above example, when we get to city 7 and select the next edge in parent 1, edges

(7, 1) and (7, 8) can both be considered. Since (7, 1) introduces a cycle, edge (7, 8) is selected. Finally, edges (8, 6) and (6, 1) complete the tour.

parent 1: 3 8 5 2 6 4 1 7

parent 2: 4 3 6 2 7 5 8 1

offspring: 4 3 5 2 7 1 8 6

It is alternate edges crossover.

#### 4.1.16 Greedy Subtour Crossover:

New crossover operator named ‘Greedy Subtour Crossover (GSX)’ that acquires the longest possible sequence of parents’ subtours. Using GSX the solution can pop up from local minima more effectively than by using simulated annealing (SA) methods.

In the GSX, we use the path representation for a genetic coding. For example, chromosome  $g = (D, H, B, A, C, F, G, E)$  means that the salesperson visits towns D, H, B, A,.., E, successively, and returns to town D.

Suppose that chromosomes of parents are  $ga = (D, H, B, A, C, F, G, E)$  and  $gb = (B, C, D, G, H, F, E, A)$ . First, choose one town at random. In this example, town C is chosen. Then,  $x = 4$  and  $y = 1$  because  $a_4 = C$  and  $b_1 = C$  respectively. Now the child  $g$  is  $(C)$ .

Next, pick up towns from the parents alternately. Begin with  $a_3$  (town A) because  $x < 4 < 1 = 3$ , and next is  $b_2$  (town D) because  $y < 1 < 1 = 2$ . The child becomes  $g = (A, C, D)$ .

In the same way, add  $a_2$  (town B),  $b_3$  (town G),  $a_1$  (town H), and the child becomes  $g = (H, B, A, C, D, G)$ . Now the next town is  $b_4 = H$  and town H has already appeared in the child (remember the salesperson may not visit the same town twice), so we can't add any more towns from parent  $gb$ . Therefore we add towns from parent  $ga$ . The next town is  $a_0 = D$ , but D is already used. Thus we can't add towns from parent  $ga$ , either. Then, we add the rest of the towns, i.e., E and F, to the child in the random order. Finally the child is  $g = (H, B, A, C, D, G, F, E)$  [11] [48].

#### 4.1.17 Edge Assembly Crossover:

EAX [48] has two important features—preserving parents’ edges using a novel technique and adding new edges by a greedy method, analogous to a minimal spanning tree. Several issues, including the selection mechanism and heuristic methods, which affect the performance of EAX have been considered.

### 4.2 CROSSOVER FOR OBJECT CLASSIFICATION PROBLEMS

- **Object classification problems:** Looseness control crossover (LCC) and Headless chicken crossover (HCC) [32]
- **Crossover for Sudoku problem:** Product Geometric Crossover (PMX) [33]
- **Crossover for grouping, graph, sequence and glude space applications:** Quotient Geometric Crossovers [34] and Merge Crossover (MX) [46].
- **Crossover for Graph Partitioning Problems:** Geometric Crossover (GX) & Geometric Crossover Labeling-independent (LI) GX Crossover and Landscape of Labeling-independent Crossover [36].
- **Crossover for Graph Colouring Problem (for Parallel GA):** Conflict elimination crossover (CEX), Greedy partition crossover (GPX), Union Independent set crossover (UISX) and Sum product crossover (SPPX) [37].
- **Multi-Parent Crossover/ Multi-Parent feature Crossover (MFX), Multicut crossover (MX) and Seed crossover (SX) [14],[15]and[16]:** The increase of parents

brings about a more comprehensive survey for determining the offspring genes and leads to a stronger tendency towards exploitation or exploration or both [18], [19], [20].

- **Center of mass crossover (CMX):** These multi-parent crossovers can lead to better performance although the performance is problem-dependent [14, 15].

- **Unimodal normal distribution crossover (UNDX):** Multiple parents into unimodal normal distribution crossover (UNDX) to enhance the diversity of offspring. This multi-parent extension of UNDX exhibits its improvement in search ability on highly epistatic problems [24].

- **Simplex crossover (SPX):** SPX performs well with three or four parents for multimodal and epistatic problems [17].

### 4.3 CROSSOVER FOR SUDOKU PROBLEM

Knowledge-Based Nonuniform Crossover [26], Strong Context Preserving Crossover (SCPC) Weak Context Preserving Crossover (SCPC) [27], Hierarchical Crossover [28], Selective crossover [29], Rank & proximity Based Crossover (RPC) [30], Depth-Dependent Crossover (DDX) [35], Alternating-Position Crossover (AP) [7], Circle Ring Crossover [12] and Sufficient Exchanging [12].

### 5. CONCLUSION

Many crossover operators are present in GA that are used in applications. The encoding type used in GA is the major criteria for selecting the crossover. The global convergence and search space must be considered while selecting the crossover operators. Effect of crossover operators in GA is application as well as encoding dependent. Many researchers say that the value of crossover probability is in between 0.6 and 1.0 and it also depends on the type of crossover used. Increasing crossover probability increases the opportunity for recombination but also may disrupt in good combination. Depending on encoding, standard crossover can have high chance to produce illegal offspring (it is application dependent e.g. TSP). The application to be solved must consider for all the crossover operators available along with possible encoding methods to get good results.

Many new applications are solved by existing crossover operators by considering their effectiveness in related applications. Most of time new crossover operators use old crossover operators along with additional changes. To get the proper crossover operators for a new problem it is recommended that, to see similar types of problems solved using GA along with various crossover operators used. It is recommended that for solving any problem it is important to overview the search space with modality extremes. Continuity (nature of search space), study existing crossover operators and then select or create a new crossover (by mixing).

### ACKNOWLEDGMENTS

Our special thanks to Dr. Tomasz Dominik Gwiazda for his e-book “Genetic Reference Volume-I Crossover for Single objective numerical optimization” and Dr. George G. Mitchell for his Ph.D Thesis “Evolutionary Computation Applied to

Combinatorial Optimization Problems" for inspiring to write review paper on crossover operator.

## REFERENCES

- [1] Tomasz Dominik Gwiazda, "Genetic Algorithms Reference", Volume –I, Poland: Tomasz Gwiazda, 2006
- [2] David E. Goldberg and Robert Lingle Jr, "Alleles, loci and the traveling salesman problem", *Proceedings of the 1<sup>st</sup> International Conference on Genetic Algorithms*, pp. 154-159, 1985.
- [3] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the TSP", *Proceedings of the 2<sup>nd</sup> International Conference on Genetic Algorithms on Genetic Algorithms and their Application*, pp. 224-230, 1987.
- [4] Lawrence Davis, "Applying adaptive algorithms to epistatic domains", *Proceedings of the 9<sup>th</sup> international joint conference on Artificial Intelligence*, Vol. 1, pp. 162-164, 1985.
- [5] Gilbert Syswerda, "Schedule optimization using genetic algorithms", *Handbook of Genetic Algorithms*, pp. 332-349, New York: Van Nostrand Reinhold, 1991.
- [6] H. Muhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization", *Proceedings of Workshop on Parallel Processing: Logic, Organization and Technology*, pp. 398-406, 1991.
- [7] P. Larranaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga, "Optimal Decomposition of Bayesian Networks by Genetic Algorithms", Internal Report, EHU-KZAA-IKT-3-94, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1994.
- [8] L. Darrell Whitley, Timothy Starkweather and D'Ann Fuquay, "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator", *Proceedings of the 3<sup>rd</sup> International Conference on Genetic Algorithms*, pp. 133-140, 1989.
- [9] John J. Grefenstette, Rajeev Gopal, Brian J. Rosmaita and Dirk Van Gucht, "Genetic Algorithms for the Traveling Salesman Problem", *Proceedings of the 1<sup>st</sup> International Conference on Genetic Algorithms*, pp. 160-168, 1985.
- [10] J. Grefenstette, "Incorporating Problem Specific Knowledge into Genetic Algorithms", In L. Davis (ed.) "Genetic Algorithms and Simulated Annealing", Morgan Kaufmann, pp. 42-60, 1987.
- [11] Sengoku, H., Yoshihara, I., "A Fast TSP Solution using Genetic Algorithm (Japanese)", *Proceedings of 46<sup>th</sup> National Convention of Information Processing Society of Japan*, 1993.
- [12] Liang-Jie Zhang, Mao Zhi-Hong and Li Yan-Da, "Mathematical Analysis of Crossover Operator in Genetic Algorithms and its Improved Strategy", *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 412-417, 1995.
- [13] Shengxiang Yang, "Adaptive Non-Uniform Crossover Based on Statistics for Genetic Algorithms", *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 650-657, 2002.
- [14] Yoshida Junichi, Miki Mitsunori, Hiryasu Tomoyuki and Sakata Yoshinobu, "New Crossover Scheme for Parallel Distributed Genetic Algorithms " *Proceedings of the IASTED International Conference on Parallel and Distributed Computing And Systems*, Vol. 1, No. 6-9, pp. 145-150, 2000.
- [15] S. Shigeyoshi Tsutsui, "Multi-parent recombination in genetic algorithms with search space boundary extension by mirroring", *Parallel Problem Solving from Nature – PPSN V, Lecture Notes in Computer Science*, Vol. 1498, pp. 428-437, 2006.
- [16] S. Tsutsui and A. Ghosh, "A study on the effect of multi-parent recombination in real coded genetic algorithms", *Proceedings of IEEE World Congress on Computational Intelligence*, pp. 828-833, 1998.
- [17] Chuan-Kang Ting and Chun-Cheng Chen, "The Effects of Supermajority on Multi-Parent Crossover", *IEEE Congress on Evolutionary Computation*, pp. 4524-4530, 2007.
- [18] S. Tsutsui, M. Yamamura and T. Higuchi. "Multi-parent recombination with simplex crossover in real coded genetic algorithms", *Proceedings of the Genetic and Evolutionary Computation Conference*, Vol. 1, pp. 657-664, 1999.
- [19] Chuan-Kang Ting, "On the convergence of multi-parent genetic algorithms", *IEEE Congress on Evolutionary Computation*, Vol. 1, pp. 396-403, 2005.
- [20] Chuan-Kang Ting, "On the mean convergence time of multi-parent genetic algorithms without selection", *Proceedings of the 8<sup>th</sup> European Conference on Advances in Artificial Life*, Vol. 3630, pp. 403-412, 2005.
- [21] A. E. Eiben, "Multiparent Recombination", *Handbook of Evolutionary Computation*, Institute of Physics Publishing and Oxford University Press, 1997.
- [22] A. E. Eiben and C.H.M. van Kemenade, "Diagonal crossover in genetic algorithms for numerical optimization", *Journal of Control and Cybernetics*, Vol. 26, No. 3, pp. 447-465, 1997.
- [23] A.E. Eiben, C.H.M. van Kemenade and J.N. Kok. "Orgy in the Computer: Multi-parent reproduction in genetic algorithms", *Proceedings of the 3<sup>rd</sup> European Conference on Artificial Life*, pp. 934-945, 1995.
- [24] Kalyan Deb, S. Karthik and Tatsuya Okabe, "Self-Adaptive Simulated Binary Crossover for Real-Parameter Optimization", *Genetic and Evolutionary Computation Conference*, pp. 1187-1194, 2007.
- [25] H. Kita, I. Ono and S. Kobayashi, "Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms", *Proceedings of the Congress on Evolutionary Computation*, Vol. 2, pp. 1581-1588, 1999.
- [26] Inki Hong, Andrew B. Kahng and Byung Ro Moon, "Exploiting Synergies of Multiple Crossovers: Initial Studies", *Proceedings of IEEE International Conference on Evolutionary Computation*, Vol. 1, 1995.
- [27] Harpal Maini, Kishan Mehrotra, Chilukuri Mohan and Sanjay Ranka, "Knowledge-Based Nonuniform

- Crossover”, *Proceedings of the 1<sup>st</sup> IEEE Conference on World Congress on Computational Intelligence Evolutionary Computation*, pp. 22-271, 1994.
- [28] Patrik D’haeseleer, “Context Preserving Crossover in Genetic Programming”, *Proceedings of the 1<sup>st</sup> IEEE Conference on World Congress on Computational Intelligence Evolutionary Computation*, Vol. 1, pp. 256-261, 1994.
- [29] P. J. Bentley and J. P. Wakefield, “Hierarchical Crossover in Genetic Algorithms”, *Proceedings of the 1<sup>st</sup> On-line Workshop on Soft Computing*, 1996.
- [30] Chilkuri K. Mohan, “Selective crossover: towards fitter offspring”, *Proceedings of the ACM symposium on Applied Computing*, pp. 374-378, 1998.
- [31] Goutam Chakraborty and Basabi Chakraborty, “Rank and Proximity Based Crossover (RPC) to Improve Convergence in Genetic Search”, *IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1311-1316, 2005.
- [32] Yuichi Nagata, “Criteria for designing crossovers for TSP”, *IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1465-1472, 2004.
- [33] Mengjie Zhang, Xiaoying Gao and Weijun Lou, “Looseness Controlled Crossover in GP for Object Recognition”, *IEEE Congress on Evolutionary Computation*, pp.1285-1292, 2006.
- [34] Alberto Moraglio, Julian Togelius and Simon Lucas, “Product Geometric Crossover for the Sudoku Puzzle”, *IEEE Congress on Evolutionary Computation*, pp. 470-476, 2006.
- [35] Yourim Yoon, YongHyuk Kim, Alberto Moraglio and Byung Ro Moon, “Quotient Geometric Crossovers”, Technical Report, CSM-467, Department of Computer Science, University of Essex, 2007.
- [36] Takuya Ito, Hitoshi Iba and Satoshi Sato, “Depth-Dependent Crossover for Genetic Programming”, *Proceedings IEEE World Congress on Computational Intelligence Evolutionary Computation*, pp. 775-780, 1995.
- [37] Alberto Moraglio, Yong-Hyuk Kim, Yourim Yoon and Byung-Ro Moon, “Geometric Crossovers for Multiway Graph Partitioning”, *Evolutionary Computation*, Vol. 15, No. 4, pp. 445-474, 2007.
- [38] Zbigniew Kokosiński, Marcin Kołodziej and Krzysztof Kwarciany, “Parallel Genetic Algorithm for Graph Coloring Problem”, *Computational Science - ICCS 2004*, Vol. 3036, pp. 215-222, 2004.
- [39] Jong De Jong and Kenneth Alan, “An Analysis of the Behavior of a class of Genetic Adaptive Systems”, PhD Thesis, University of Michigan, 1975.
- [40] Donald E. Brown, Christopher L. Huntley and Andrew R. Spillane, “A Parallel Genetic Heuristic for the Quadratic Assignment Problem”, *Proceedings of the 3<sup>rd</sup> International Conference on Genetic Algorithms*, pp. 406-415, 1989.
- [41] Andrew L. Tuson, “The Implementation of a Genetic Algorithm for the Scheduling and Topology Optimisation of Chemical Flowshops”, Technical Report TRGA94-01, Physical Chemistry Laboratory, Oxford University, 1994.
- [42] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, “Evolution Algorithms in Combinatorial Optimization”, *Parallel Computing*, Vol. 7, No. 1, pp. 65-85, 1988.
- [43] Sushil. J. Louis and Gregory J. E. Rawlins, “Designer Genetic Algorithms: Genetic Algorithms in Structure Design”, *Proceedings of the 4<sup>th</sup> International Conference on Genetic Algorithms*, pp. 53-60, 1991.
- [44] Laura Barbulescu, Adele E. Howe, L. Darrell Whitley and Mark Roberts, “Understanding Algorithm Performance on an Oversubscribed Scheduling Application”, *Journal of Artificial Intelligence Research*, Vol. 27, pp. 577-615, 2006.
- [45] Kengo Katayama and Hiroyuki Narihisa, “An Efficient Hybrid Genetic Algorithm for the Traveling Salesman Problem”, *Electronics and Communications in Japan*, Vol. 84, No. 2, pp. 76-83, 2001.
- [46] Sushil J. Louis, Xiangying Yin and Zhen Ya Yuan, “Multiple Vehicle Routing With Time Windows Using Genetic Algorithms”, *Proceedings of the Congress on Evolutionary Computation*, 1999.
- [47] Yam Ling Chan, “A Genetic Algorithm Shell for Iterative Timetabling”, M.S. Thesis, Department of Computer Science, RMIT University, 1994.
- [48] George G. Mitchell, “Evolutionary Computation Applied to Combinatorial Optimisation Problems,” Ph.D. Thesis, School of Electronic Engineering, Dublin City University, 2007.

# Swarm Intelligence

David Corne, Alan Reynolds and Eric Bonabeau

Increasing numbers of books, websites and articles are devoted to the concept of ‘swarm intelligence’. Meanwhile, a perhaps confusing variety of computational techniques are seen to be associated with this term, such as ‘agents’, ‘emergence’, ‘boids’, ‘ant colony optimisation’, and so forth. In this chapter we attempt to clarify the concept of swarm intelligence and its associations, and we attempt to provide a perspective on its inspirations, history, and current state. We focus on the most popular and successful algorithms that are associated with swarm intelligence, namely ant colony optimisation, particle swarm optimisation, and (more recently) foraging algorithms, and we cover the sources of natural inspiration with these foci in mind. We then round off the chapter with a brief review of current trends.

## 1. Introduction

Nature provides inspiration to computer scientists in many ways. One source of such inspiration is the way in which natural organisms behave when they are in groups. Consider a swarm of ants, a swarm of bees, a colony of bacteria, or a flock of starlings. In these cases and in many more, biologists have told us (and we have often seen for ourselves) that the group of individuals itself exhibits behaviour that the individual members do not, or cannot. In other words, if we consider the group itself as an individual – the *swarm* – in some ways, at least, the swarm seems to be more intelligent than any of the individuals within it.

This observation is the seed for a cloud of concepts and algorithms, some of which have become associated with swarm intelligence. Indeed, it turns out that swarm intelligence is only closely associated with a small portion of this cloud. If we search nature for scenarios in which a collection of agents exhibits behaviour that the individuals do not (or cannot), it is easy to find entire and vast sub-areas of science, especially in the bio-sciences. For example, any biological organism seems to exemplify this concept, when we consider the individual organism as the ‘swarm’, and its cellular components as the agents.

We might consider brains, and nervous systems in general, as a supreme exemplar of this concept, when individual neurons are considered as the agents. Or we might zoom in on certain inhomogeneous sets of bio-molecules as our ‘agents’, and herald gene transcription, say, as an example of swarm behaviour. Fortunately, for the sake of this chapter’s brevity and depth, it turns out that the

swarm intelligence literature has come to refer to a small and rather specific set of observations and associated algorithms. This is not to say that computer scientists are uninspired by the totality of nature's wonders that exhibit such 'more than the sum of the parts' behaviour – much of this volume makes it clear that this is not so at all. However, if we focus on the specific concept of swarm intelligence and attempt to define it intensionally, the result might be thus:

1. Useful behaviour that emerges from the cooperative efforts of a group of individual agents;
2. ... in which the individual agents are largely homogeneous;
3. ... in which the individual agents act asynchronously in parallel;
4. ... in which there is little or no centralised control;
5. ... in which communication between agents is largely effected by some form of stigmergy;
6. ... in which the 'useful behaviour' is relatively simple (finding a good place for food, or building a nest -- not writing a symphony, or surviving for many years in a dynamic environment).

So, swarm intelligence is not about how collections of cells yield brains (which falls foul of at least items 2, 5, and 6), and it is not about how individuals form civilizations (violating mainly items 3, 5 and 6), and it is not about such things as the lifecycle of the slime mould (item 6). However, it is about individuals co-operating (knowingly or not) to achieve a definite goal. Such as, ants finding the shortest path between their nest and a good source of food, or bees finding the best sources of nectar within the range of their hive. These and similar natural processes have led directly to families of algorithms that have proved to be very substantial contributions to the sciences of computational optimisation and machine learning.

So, originally inspired, respectively, by certain natural behaviours of swarms of ants, and flocks of birds, the backbone of swarm intelligence research is built mainly upon two families of algorithms: ant colony optimisation, and particle swarm optimisation. Seminal works on ant colony optimisation were Dorigo et al (1991) and Colorni et al (1992a; 1992b), and particle swarm optimisation harks back to Kennedy & Eberhart (1995). More recently, alternative inspirations have led to new algorithms that are becoming accepted under the swarm intelligence umbrella; among these are search strategies inspired by bee swarm behaviour, bacterial foraging, and the way that ants manage to cluster and sort items. Notably, this latter behaviour is explored algorithmically in a subfield known as swarm robotics. Meanwhile, the way in which insect colonies collectively build complex and functional constructions is a very intriguing study that continues to be carried out in the swarm intelligence arena. Finally, another field that is often considered in the swarm intelligence community is the synchronised movement of swarms; in particular, the problem of defining simple rules for individual behaviour that led to realistic and natural behaviour in a simulated swarm. 'Reynolds' rules' provided a

general solution to this problem in 1987, and this can be considered an early triumph for swarm intelligence, which has been exploited much in the film and entertainment industries.

In the remainder we expand on each of these matters. We start in section 2 with an account of the natural behaviours that have inspired the main swarm intelligence algorithms. Section 3 then discusses the more prominent algorithms that have been inspired by the techniques in section 2, and section 4 notes some current trends and developments and offers some concluding remarks.

## 2. Inspiration from Nature

### 2.1 Social Insects and Stigmergy

Ants, termites and bees, among many other insect species, are known to have a complex social structure. Swarm behaviour is one of several emergent properties of colonies of such so-called *social insects*. A ubiquitous characteristic that we see again and again in such scenarios is *stigmergy*. Stigmergy is the name for the *indirect* communication that seems to underpin cooperation among social insects (as well as between cells, or between arbitrary entities, so long as the communication is *indirect*).

The term was introduced by Pierre-Paul Grassé in the late 1950s. Quite simply, stigmergy means communication via signs or cues placed in the environment by one entity, which affect the behaviour of other entities who encounter them. Stigmergy was originally defined by Grassé in his research on the construction of termite nests. Figure 1 shows a simplified schematic of a termite nest. We will say more about termite nests in section 2.1.3, but for now it suffices to point out that these can be huge structures, several metres high, constructed largely from mud and from the saliva of termite workers. Naturally, the complexity and functionality of the structure is quite astounding, given what we understand to be the cognitive capabilities of a single termite.

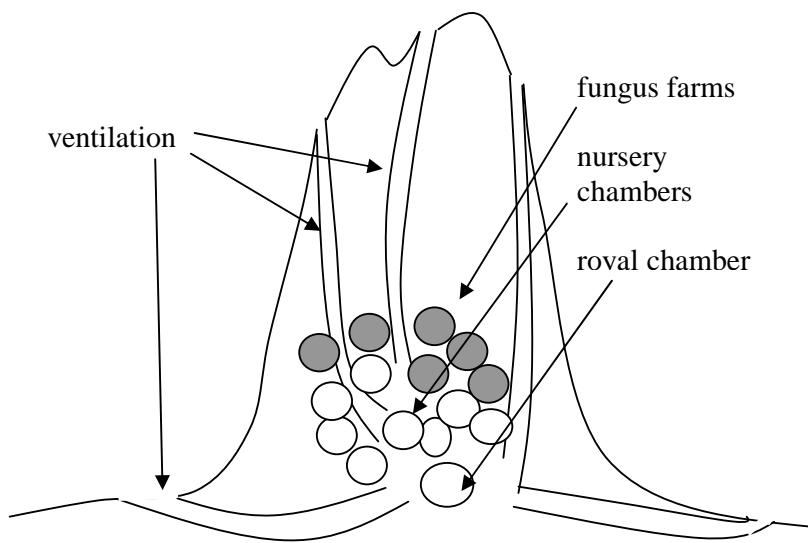


Figure 1. A highly simplified schematic of a termite nest

Following several field trips to Africa in the late 1930s and 1940s studying termites and their nests, among other things, Grasse showed that the regulation and the co-ordination of the nest-building activity did not depend on the termite workers themselves, but was instead achieved by the nest itself. That is, some kind of stimulating configuration of materials triggers a response in a termite worker, where that response transforms the configuration into another configuration that may, in turn, trigger yet another, possibly different, action performed by the same termite or by any other termite worker in the colony. This concept of stigmergy was attractive and stimulating, but at the time, and often today, it was and is often overlooked by students of social insects because it leaves open the important operational issue of how the specific trigger-response configurations and stimuli must be organized in time and space to allow appropriate co-ordination. But despite the general vagueness of Grassé's formulation, stigmergy is recognised as a very profound concept, the consequences of which are still to be fully explored. Stigmergy is not only of potential importance for our understanding of the evolution and maintenance of social behaviour in animals, from communally breeding species to highly social insects, it is also turning out to be a crucial concept in other fields, such as artificial intelligence, robotics, or the social, political and economic sciences. Meanwhile, in the arena of natural computing,

stigmergy is the fundamental concept behind one of the main swarm intelligence algorithms, as well as several others.

Apart from termite nests, another exemplary case of stigmergy in nature is that of pheromone deposition. Ants deposit pheromone along their paths as they travel; an ant striking out on her own will detect pheromone trails, and prefer to follow such trails already travelled. In general, the concept of stigmergy captures underlying commonalities in (usually) insect behaviours that are underpinned by indirect communication. This covers more emergent behaviours than trail-following, and (the original inspiration for the term) the construction of structures such as termite mounds and bee hives. Stigmergy also seems key to behaviours such as brood sorting and cemetery clustering -- some ant species are known to spatially cluster their young into age-groups within the nest, and they keep their nests tidy by removing dead nest mates and piling them into clusters outside.

The phenomenon of stigmergy has much earlier evolutionary roots; it is now used to explain the morphologies of multicellular organisms, sea-shells, and so forth. Essentially, individual cells position themselves in a way influenced by deposits left behind by their colleagues or precursors. A useful way to think of it is that stigmergic communication involves a ‘stigmergy structure’ which is like a notepad, or an actual structure, built from cues left by individuals.

The structure itself may be a spatially distributed accumulation of pheromone, or a partially built hive, or a partially constructed extracellular matrix. The structure itself influences the behaviours of the individuals that ‘read’ it, and these individuals usually also add to the structure. Army ants find their directions of travel influenced by pheromone trails, and they add to the trails themselves. Termites are triggered by particular patterns that they see locally in the partially built mound, and act in simple and specific ways as a result, resulting in additions to the structure itself. An authoritative overview of stigmergy associated behaviours in nature is Bonabeau et al (1997), while Theraulaz (1994) provides a comprehensive survey of self-organisation processes in insect colonies. As hinted above, when we consider the stigmergic processes often observed in nature, the most prominent sources of inspiration from the swarm intelligence viewpoint are those of navigation to food sources, sorting/clustering, and the collective building of structures. We briefly consider each of these next.

### 2.1.1 Natural Navigation

Navigation to food sources seems to depend on the deposition of pheromone by individual ants. In the natural environment, the initial behaviours of a colony of ants in seeking a new food source is for individual ants to wander randomly. When an ant happens to find a suitable food source they will return to their colony; throughout, the individual ants have been laying pheromone trails. Subsequent ants setting out to seek food will sense the pheromone laid down by their precursors, and this will influence the path they take up. Over time, of course, pheromone trails evaporate. However, consider what happens in the case

of a particularly close food source (or, alternatively, a faster or safer route to a food source). The first ant to find this source will return relatively quickly. Other ants that take this route will also return relatively quickly, so that the best routes will enjoy a greater frequency of pheromone laying over time, becoming strongly fancied by other ants. The overall collective behaviour amounts to finding the best path to a nearby food source, and there is enough stochasticity in the process to avoid convergence to poor local optima – trail evaporation ensures that suboptimal paths discovered early are not converged upon too quickly, while individual ants maintain stochasticity in their choices, being influenced by but not enslaved by the strongest pheromone trail they sense. Figure 2 shows a simple illustration, indicating how ants will converge via stigmergy towards a safer and faster way to cross a flow of water between their nest and a food source.

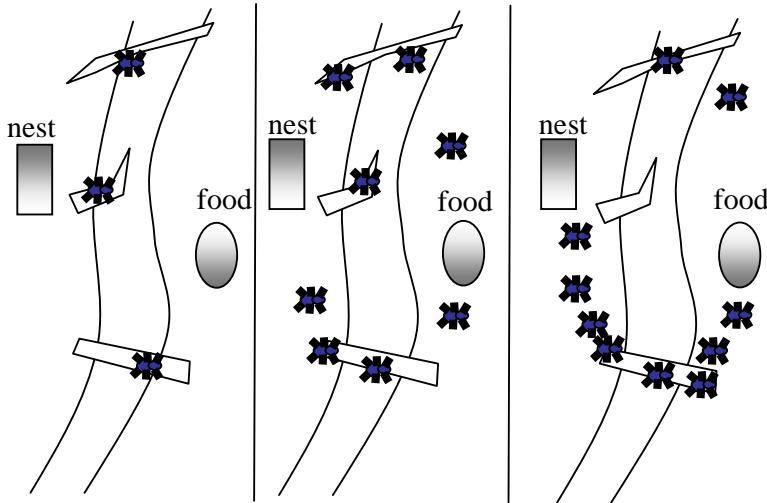


Figure 2: Convergence to a safer crossing over time.

In figure 2, we see three contrived snapshots of a simple scenario over time. On the left, ants need to cross a narrow stream of flowing water towards a tempting food source, and three crossings – fallen twigs – present themselves. Initially ants are equally likely to try each one. Each of these ants lays its trail of pheromone as it makes its journey towards the right. Over time, however, the path

towards the middle twig becomes less laid with pheromone, simply because, unlike the situation with the upper and lower twigs, there are no ants laying pheromone *on their way back* from that particular path. Eventually, on the right, we see several ants are following the path defined by the lower twig, both attracted by, and further multiplying, the steady build-up of pheromone on this path, which is faster than that on the path defined by the upper twig. Although the upper twig provides a fairly short journey, it is more perilous, since this twig is quite narrow, and several ants fall off before being able to strengthen the path.

Actually, this is one of the simplest and most straightforward activities in social insects related to pheromones. The term ‘pheromone’ itself was promoted for this context in 1959 (the late 1950s was clearly a fruitful period for swarm intelligence vocabulary) (Karlson & Lüscher, 1959), to encompass the broad range of biologically active chemicals used by insects for varieties of communicative purpose. The context in which we describe it above is known in biology as ‘recruitment’, referring broadly to tasks in which individuals discover an opportunity (usually a food source) and need to recruit others to help exploit it. However there are many other behaviours that are associated with pheromones, such as indicating alarms or warnings, interactions between queens and workers, and mating. An excellent source of further information is Vander Meer et al (1998b), and in particular Vander Meer et al (1998a).

### 2.1.2 Natural Clustering

Turning now to a different style of swarm behaviour, it is well known that ant species (as well as other insect species) exhibit emergent sorting and clustering of objects. Two of the most well known examples are the clustering of corpses of dead ants into cemeteries (achieved by the species *Lasius Niger*, *Pheidole pallidula*, and *Messor sancta* (Depickere et al, 2004), and the arrangement of larvae into similar-age groups (so called brood-sorting, achieved by *Leptothorax unifasciatus* (Franks & Sendova-Franks, 1992)).

For example, in *Leptothorax unifasciatus* ant colonies, the ants' young are organised into concentric rings called annuli. Each ring comprises young of a different type. The youngest (eggs, and micro-larvae) are clustered together in the central cluster. As we move outwards from the centre, progressively older groups of larvae are encountered. Also, the spaces between these rings increase as we move outwards.

In cemetery formation, certain ant species are known to cluster their dead into piles, with individual piles maintained at least a minimal distance from each other. In this way, corpses are removed from the living surroundings, and cease to be a hindrance to the colony. One aspect of this behaviour in particular is that it is arguably not exemplary of swarm behaviour; i.e. it is perhaps not *collective* intelligence. The explanatory model that seems intuitively correct and confirmed by observation (Theraulaz et al, 2002) is one in which an individual operates

according to quite simple rules while otherwise generally wandering around randomly:

1. If not carrying a corpse, and a single corpse (or quite small cluster of corpses) is encountered, pick it up.
2. If carrying a corpse, and a relatively large cluster of corpses is encountered, put it down.

A single ant could achieve the clustering observed in nature, that seems to operate according to these rules (Theraulaz et al, 2002). However, the emergent clusters are produced faster when a collection of ants are involved. Gaubert et al (2007) is a useful reference for discussion of mathematical and other models of these behaviours. Meanwhile, a steady line of recent research is investigating computational clustering methods that are directly inspired by these natural phenomena (Handl & Meyer, 2007). The first such inspired clustering algorithm was proposed by Lumer & Faieta (1994), closely based on the Deneubourg et al (1991) model of the natural process. In recent work (Handl et al, 2006), an ant-based clustering method called ATTA is tested, and the case is made convincingly that ant-based clustering algorithms certainly have a niche in data mining, performing particularly well on problems where the number of clusters are not known in advance, and where the clusters themselves are highly variable in size and shape. In this article our focus stays with optimisation and we will say little more about clustering; we refer the reader again to Handl and Meyer's recent review (Handl & Meyer, 2007) for further study on this topic.

### 2.1.3 Natural construction

We now consider the extraordinary collective behaviour that leads to the construction of achievements such as wasp nests, termite mounds and bee hives. Brood-sorting, considered above, exemplifies a simple structure that arises from collective behaviour. However the more visible and impressive structures such as termite mounds have always impressed observers, and often confounded us when we try to imagine how such simple minds can lead to such creations. As will be clear from context, stigmergy seems to be the key to understanding these buildings; patterns inherent in partial elements of structures are thought to trigger simple rule-based behaviour in the insect, which in turn changes the perceived patterns, and so on ..., until a complete hive or nest is built. Much computation-based study has been made of this by Bonabeau, Theraulaz and co-workers.

In nature, the sizes of such social insect structures can reach an astounding 30 metres in diameter (Grassé, 1984). An impressive example of the complexity of these structures comes from the African termites *Macrotermitinae*, 'the fungus

growers'. In a mature nest of this species, there are typically six distinct elements of structure (we adapt this description from Bonabeau et al, 1998):

1. The protective and ribbed cone-shaped outer walls (also featuring ventilation ducts).
2. Brood chambers within the central 'hive' area. They have a laminar structure and contain the nurseries where the young termites are raised.
3. The hive consists of thin horizontal lamellae supported by pillars.
4. A flat floor structure, sometimes exhibiting cooling vents in a spiral formation.
5. The royal chamber: a thick walled enclosure for the queen, with small holes in the walls to allow workers in and out. This is usually well protected underneath the hive structure, and is where the queen lays her eggs.
6. Garden areas dedicated to cultivating fungi. These are arranged around the hive, and have a comb-like structure, arranged between the central hive and the outer walls.
7. Tunnels and galleries constructed both above and below ground which provide pathways from the termite mound to the colony's known foraging sites.

So, how does a collection of termites make such a structure? Perhaps this is the most astonishing example of natural swarm intelligence at work. Observations and descriptions of such structures from the biology literature have tended to focus on *description*, elucidating further and finer detail from a variety of species, but have done little to clarify the mechanism. However, computational simulation work such as Theraulaz & Bonabeau (1995), has indicated how such behaviour can emerge from collections of 'micro-rules', where patterns of the growing structure perceived by an individual termite (or ant, wasp, etc ...) act as a stigmergic trigger, perhaps in tandem with other environmental influences, leading to a specific response that adds a little new structure. Theraulaz and Bonabeau (1995) and Bonabeau et al (2000) have shown how specific collections of such micro-rules can lead to, in simulation, a variety of emergent structures, each of which seems convincingly similar to wasp nests from specific wasp species. Figure 3 (reprinted detail with permission) shows examples of three artificial nests, constructed in this fashion, that closely resemble the nest structures of three specific wasp species;

several more such are presented in Theraulaz & Bonabeau (1995) and Bonabeau et al (2000).

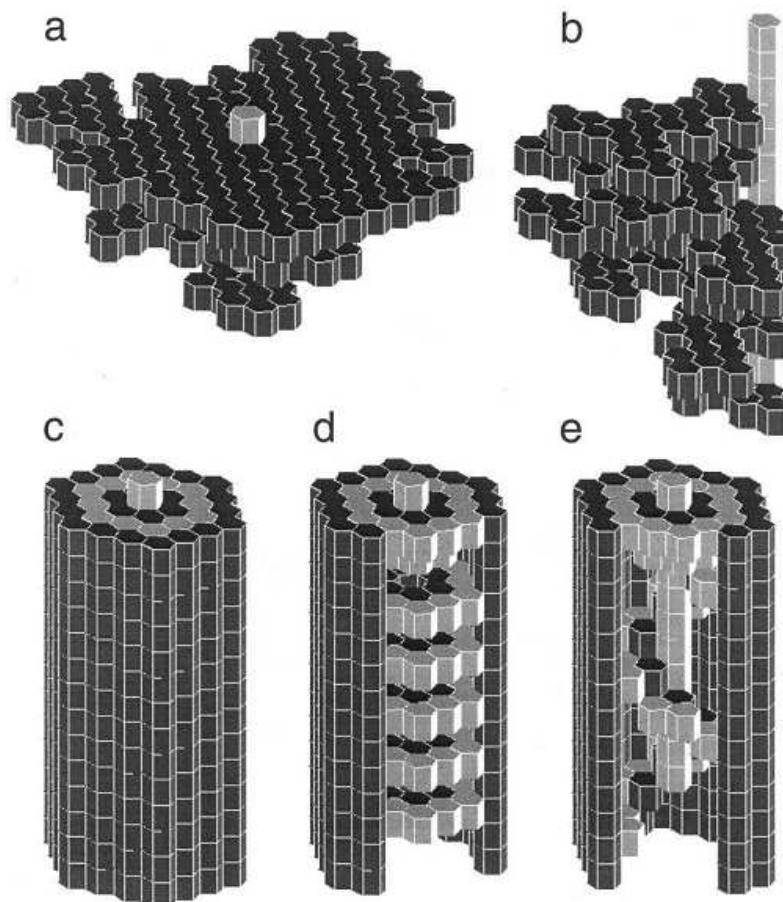


Figure 3. Fig. 2 from Bonabeau et al (2000), reproduced with permission. These show results from artificial colonies of ten wasps, operating under the influence of stimulus-response micro-rules based on patterns in a 3D hexagonal brick lattice. (a) A nest-like architecture resembling the nests of *Vespa* wasps, obtained after 20,000 building steps; (b) An architecture resembling the nests of *Parachartergus* wasps, obtained after 20 000 building steps; (c) resembling *Chatergus* nests, obtained after 100 000 building steps; (d, e) Showing internal structure of (c).

In Figure 4 we see some examples of micro-rules of the kind that can lead to the types of structures shown in Figure 3. A micro-rule simply describes a three-dimensional pattern of ‘bricks’; in the case of the experiments that led to Figure 3, a brick has a hexagonal horizontal cross section, and there are two types of brick (it was observed that at least two different brick types seem necessary for interesting results). The three dimensional pattern describes the immediate neighbourhood of a single central brick, including the seven hexagonal cells above it (the upper patches of hexagons in figure 4) the six that surround it, and the seven below it. In the figure, a ‘white’ hexagonal cell is empty – meaning no brick here; otherwise there are two kinds of bricks, distinguished by different shadings in the figure. A micro-rule expresses the following building instruction: ‘if the substructure defined by this pattern is found, with the central cell empty, then add the indicated type of brick in the central position’.

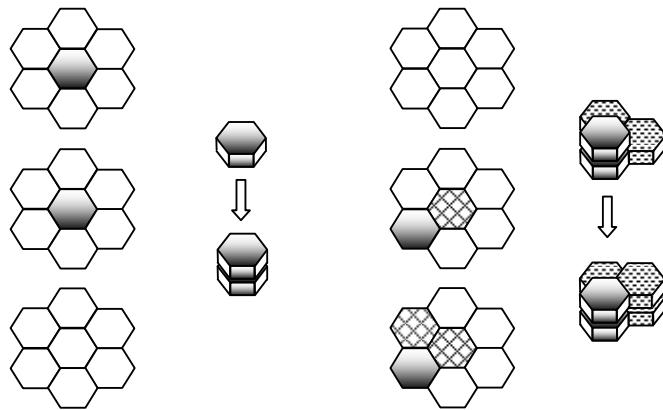


Figure 4. An illustration of two ‘micro-rules’ from the space of such rules that can lead to structures such as those in Figure 3. A single micro-rule defines a building instruction based on matching a pattern of existing bricks. A single column of three groups of 7 hexagonal cells is a micro-rule, by describing a structure around the neighbourhood of the central cell (which is empty in the matching pattern). The building instruction is to fill the central cell with the indicated type of ‘brick’. In this figure, two example micro-rules are shown, each further illustrated by ‘before’ and ‘after’ building patterns on the right.

Intuition suggests how the construction by a collection of agents such as wasps of artefacts such as those in Figure 3, or even more complex artefacts, may be facilitated by specific collections of micro-rules, however that does not make it easy to design a set of micro-rules for a specific target construction. Sets of micro-rules achieving the illustrated results were obtained by using a carefully designed

evolutionary algorithm. Interested readers should consult Bonabeau et al (2000) for further detail, including analyses of the operation of the emerging rulesets, revealing the requirement for various types of co-ordination implicitly built in to the micro-rule collection.

Meanwhile, it is a long way from wasp nests to termite mounds, especially mounds of the complexity hinted at above. However, by considering and extending partial models for elements of termite mounds, in Bonabeau et al (1998) some basis is provided for the suspicion that such complexity may be explained by the interaction of stimulus-response ('micro-rule') based processes and pheromone-based triggers that modify the stimulus-response behaviour, unfolding over time as a controlled morphogenetic process.

Finally we note that much of what we have discussed in this subsection forms part of the basis for the new field of 'swarm robotics'. This area of research (Sahin, 2005; Mondada et al, 2005) focuses on what may be achievable by collections of small, simple robots furnished with relatively non-sophisticated ways to communicate. Chief among the motivating elements of such research are the qualities of robustness, flexibility and scalability that swarm robots could bring to a number of tasks, ranging through agriculture, construction, exploration and other fields. Imagine we wish to build a factory on Mars, for example. We might imagine this could be performed, in some possible future, by a relatively small collection of very sophisticated and intelligent robots. However, in the harsh conditions of Mars, we can expect that loss or destruction of one or more of these is quite likely. Swarms of simple robots, instead, are far more robust to loss and damage, and may present an altogether more manageable, shorter-term and more adaptable approach than the 'super-robot' style.

As yet, swarm robotics has not risen to such heights, although there is published discussion along these lines, drawing from the sources of natural inspiration we have already discussed (Cicirello & Smith, 2001). Meanwhile, interesting and useful behaviours have been demonstrated in swarm robotics projects, after, in almost all cases, considerable work in design, engineering and construction of the individual 'bots'. Supporting these studies, a large part of swarm robotics research is into how to design the individual robots' behaviours in such a way that the swarm (or team) achieves an overall goal or behaviour. Unsurprisingly to many, it turns out that the judicious use of evolutionary computation proves effective for this difficult design problem (Waibell et al, 2009). However, hand-design or alternative principled methods for designing behaviours remains a backbone of this research, especially when the desired overall behaviour is complex, involving many tasks (e.g. Ducatelle et al, 2009), and in general there are several emerging issues in swarm robotics that have sparked active current lines of research, such as the problem of 'interference' – swarm robots, whether co-operating on the same task or not, often physically interfere with each others' operation (Pini et al, 2009) – and the problem of achieving tasks with minimal energy requirements (Roberts et al, 2008).

Relatively impressive behaviours from swarm robotics research has so far included co-operative transport of one or more objects, and co-operation towards moving up a vertical step (as large as the bots involved); readers may visit the European Project 'Swarmbots' and 'Swarmanoids' websites for explanation and many other resources, at <http://www.swarm-bots.org/>, and at <http://www.swarmanoid.org/> as well as similar resources such as <http://www.pherobot.com/>. Hardware and related technology issues remain a bottleneck that still inhibits a full exploration of social insect swarm intelligence in robotics, however this work continues toward that end and will be observed with great interest.

## 2.2 Foraging

There are broadly two types of natural process that go by the term "foraging", and in turn provide sources of inspiration for optimisation (or resource allocation) methods. In both cases, the overall emergent behaviour is that the swarm finds and exploits good food sources, adaptively moves to good new sources as current ones become depleted, and does all of this with efficient expenditure of energy (as opposed to, for example, brute force search of their environment). The means by which this behaviour is achieved is rather different in these two sources of inspiration.

In one case, that of 'bacterial foraging' (Passino, 2002), individual bacteria are (essentially) directed towards rich areas via *chemotaxis*; that is, they exist in an environment in which their food source diffuses, so they can detect and respond to its presence. In particular, chemotaxis refers to movement along a chemical gradient. An individual e-coli bacterium has helical appendages called *flagellae* which spin either clockwise or anticlockwise (we can think of them as analogous to propellers). When they spin in one direction, the bacterium will 'tumble'; this is an operation which ends up moving the bacterium a short distance, and leaving it with an essentially random new orientation. When the flagellae spin in the other direction, the bacterium's movement will be a 'run' - this is a straight-line movement in the direction the bacterium was facing at the beginning, and continues as long as the flagellae continue to spin in the same direction.

In a nutrient-free and toxin-free environment, an individual bacterium will alternate between clockwise and anti-clockwise movement of its flagellae. So, it tumbles, runs, tumbles, runs, and so forth. The effect of this behaviour is random search for nutrients. However, when the bacterium encounters an increasing nutrient gradient, that is, a higher concentration of nutrient in its direction of movement, its internal chemistry operates in a way that causes the runs to be of longer duration. It still alternates between tumbles and runs, but maintains longer run lengths so long as the gradient continues to increase. The effect of this is to

explore and exploit the food source, moving upwards along the nutrient gradient, while maintaining an element of stochastic exploration.

In addition, under certain conditions we know that bacteria secrete chemicals that attract each other. There is speculation that this can happen in response to nutrient rich environments, so that additional bacteria are recruited to exploit the food source, where the attractive secretions build further on the attraction provided by the chemical gradient. Also, there is evidence that bacteria release such an attractant under stressful conditions too, which in turn may be a protective response; as they swarm into a sizeable cluster, many individuals are protected from the stressful agent. These self-attractant and chemotactic behaviours are known to lead to pattern formation under certain conditions (Budrene & Berg, 1991). These and many other details have been elucidated for *e. coli* and similar bacteria via careful experimentation, for example: Berg & Brown (1972), Segall et al (1986), and deRosier (1998).

The other broad style of efficient collective foraging behaviour is that exhibited by the honeybee (among other insects). When a bee discovers a food source some distance from the hive, it returns to the hive and *communicates the direction, distance and quality* of the food source to other bees. The details of this communication, achieved by specialised ‘dances’, are quite remarkable, and have emerged from a series of ingenious experiments and observations, largely by Karl von Frisch (1967). The essential details are these: in context, the dance is performed in alignment with a particular aspect of the hive structure, which provides an absolute reference against which the bee audience can perceive specific angles. The main dance is the ‘waggle’ dance, which consists of a straight line movement, during which the bee wiggles from side to side along the way. This straight line movement is done upwards at a particular angle from the vertical. At the end of the straight line part, the bee loops round to the starting point and repeats (actually, it alternates the direction of this loop, drawing a figure “8”). The angle of this dance from vertical indicates to the bee audience the direction they need to take with respect to the current position of the Sun. Among various extraordinary aspects of this, it is known that the bee automatically corrects for movement of the Sun during the day, and communicates the correct direction. Also, at times, the bee will pause its dance, and allow watching bees to sample the nectar it is carrying, giving an indication of the quality of the food source.

More interesting from the algorithmic viewpoint, however, is that the abundance of the food source is communicated by the duration of the dance (essentially, the number of times the figure “8” is repeated). An individual enjoying this performance may or may not decide to follow these directions and be ‘recruited’ to this particular source. Such an individual may also be exposed to rival performances. However, the longer the duration, the more bees will see this dance, and the more will be recruited to this dance rather than others. In this way, the bee colony sports the emergent behaviour of smart resource allocation, with

more bees assigned to better sources, and adaptation over time as returning bees gradually provide shorter and shorter dances as the source becomes depleted.

As we will see later, both bee and bacterial foraging have been taken as the inspiration for general optimisation methods, as well as for approaches to the specific problem domain of optimal resource allocation.

### 2.3 Flocking

Perhaps the most visible phenomenon that brings to mind swarm intelligence is the travelling behaviour of groups (flocks, swarms, herds, etc...) of individuals that we are all familiar with. The mesmerising behaviour of large flocks of starlings is a common morning sight over river estuaries. Swarms of billions of monarch butterflies, herds of wildebeeste, schools of tunafish, swarms of bees, all share common emergent behaviours, chiefly being:

1. the individuals stay close to each other, but not too close, and there seem to be no collisions;
2. swarms change direction smoothly, as if the swarm was a single organism;
3. *unlike* a single organism, yet still smoothly and cleanly, swarms sometimes pass directly through narrow obstacles (in the way that a stream of water passes around a vertical stick placed centrally in the stream's path).

In some ways, such swarm behaviour is arguably less mysterious than other emergent behaviours; it seems clear that we might be able to explain this behaviour via a built-in predisposition for individuals to stay with their colleagues, and we can readily imagine how evolution will have favoured such behaviour: there is safety in numbers. However, the devil is in the detail, and it took seminal work by Reynolds (1987) to outline and demonstrate convincing mechanisms that can explain these behaviours. Reynolds' work was within the computer graphics community, and has had a volcanic impact there. Now known as 'Reynold's rules', the recipe that achieves realistic swarm behaviour (with some, but not obtrusively much, parameter investigation needed depending on the species simulated) is this triplet of steering behaviours to be followed by each individual in a swarm:

**Separation:** steer to avoid coming too close to others.

**Alignment:** steer towards the mean heading of others.

**Cohesion:** steer towards the mean *position* of others

A basic illustration of each rule is given in Figure 4. In the figure, we take the common terminology of 'boid' to refer to an individual in a flock. The figure shows examples of the adjustments that might be made under the guidance of the

rules. To understand how realistic swarm simulation works, it is important to note that each boid has its own perceptual field – i.e. it could only ‘see’ a certain distance, and had a specific field of view (boids cannot see behind them, for example). The adjustments it makes to its velocity at any time are therefore a function of the positions and velocities of the boids in its perceptual field, rather than a function of the flock as a whole.

The rules are key ingredients to a realistic appearance in simulated flocks, but there are several other details, particularly regarding obstacle avoidance and goal-seeking behaviour. Interested readers may consult Reynolds (1987) and the many papers that cite it. It is important to note that these rules are not strictly nature-inspired, in the sense that Reynolds was not attempting to explain natural swarming behaviour, he was simply attempting to emulate it. However, the resulting behaviour was found to agree well with observations of natural flocking behaviour (e.g. Partidge (1982) and Potts (1984)), and Reynolds (1987) reported that “many people who view these animated flocks immediately recognize them as a representation of a natural flock, and find them similarly delightful to watch”. These techniques are now common in the film industry; among the earliest uses were in the film Batman Returns (1992, director Tim Burton), in which Reynolds’ rules lay behind the simulated bat swarms and flocks of penguins.

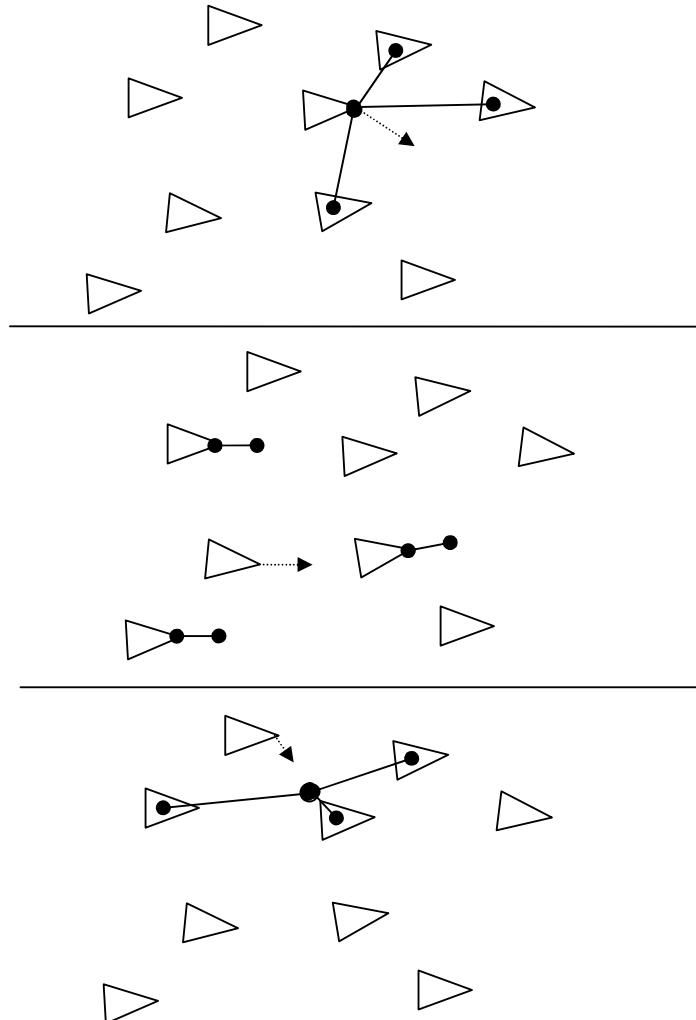


Figure 5. Illustrating Reynolds' rules, which lead to natural-looking behaviour in simulated swarms. Upper: Separation: each boid makes an adjustment to velocity which prevents it coming too close to the flockmates in its perceptual field. Middle: Alignment: a boid adjusts its heading towards the average of those in its perceptual field; lower: a boid makes an adjustment to velocity that moves it towards the mean position of the flockmates in its perceptual field.

Meanwhile, natural flocking behaviour also turns out to be one of the sources of inspiration for the highly popular and successful particle swarm optimisation

algorithm, which appears in the next section as one of the prominent flagships for swarm intelligence. It is not obvious why flocking behaviour might lead to an optimisation algorithm, however it soon becomes clear when we consider the dynamics of flocking, and the tendency of optimisation landscapes to be locally smooth. In the case of bacterial foraging, the dynamics of the natural behaviour are such that individuals will tend to congregate around good areas. With the bacterial example, nature provides mechanisms for suggesting appropriate directions of movement, while there is a clear goal for the bacterial colony to achieve - find nutrient rich (and toxin free) areas. In particular for the current context, the secretion of attractant chemicals is a mechanism that promotes bacteria swarming together, while an individual's position in its environment directly provides it with a level of 'fitness' that it can sense in terms of nutrient concentration.

When we consider flocking behaviour in birds, however, Reynolds' work provides clues about appropriate ways to move together as a swarm, but there is no clear mirror of a 'fitness' in the environment. Often birds will migrate from A to B, knowing where they are going, rather than seeking new environments. However, if flocks *did* have a goal to move towards 'fitter' positions in the landscape they travel, then it becomes intuitively reasonable to consider the cohesive swarm behaviour as a sensible way to achieve local exploration around fit areas, perhaps enabling the sensing of even fitter areas that may then sway the overall movement of the flock. In this way, flocking behaviour combines with a little algorithm engineering to achieve a very successful optimisation mechanism.

### 3. Two Main Concepts for Swarm Intelligence Algorithms

When we consider the impact of swarm intelligence so far on computer science, two families of algorithms clearly stand out in terms of the amount of work published, degree of current activity, and the overall impact on industry. One such family is inspired directly by the pheromone-trail following behaviour of ant species, and this field is known as Ant Colony Optimization (ACO). The other such family is inspired by flocking and swarming behaviour, and the main exemplar algorithm family is known as Particle Swarm Optimization (PSO). Also in this family are algorithms based on bacterial foraging, and some of the algorithms that are based on bee foraging; these share with PSO the broad way in which the natural phenomenon is mapped onto the concept of search within a landscape. In this section we discuss these two main families in turn.

#### 3.1 Ant Colony Optimization

Ant Colony Optimization (ACO) was introduced in 1996 via an algorithm called 'Ant System' (AS) (Dorigo et al, 1996). The basic approach used in AS remains highly characteristic of most ACO methods in current use, and we will describe it next.

Recall that, in the natural case, an ant finds a path from its nest to a food source by following the influences of pheromone trials laid down by previous ants who have previously sought food (and usually returned). AS, and ACO algorithms in general, mirror aspects of this behaviour quite faithfully. In short: an artificial ant builds a solution to the optimisation problem at hand, and lays down simple 'artificial pheromone' along the route it took towards that solution. Following artificial ants then build solutions of their own, but are influenced by the pheromone trails left behind by their precursors. This is the essential idea, and starts to indicate the mapping from the natural to the artificial case. However, there are various further issues necessary to consider to make this an effective optimization algorithm. We discuss this further in the next section, focussing on the mapping from the basic ideas of ACO to applications in optimisation.

### 3.1.1 Applying ACO to optimisation problems

In order to apply AS to an optimisation problem, the problem needs to be represented in such a way that any candidate solution to it is a specific path along a network. This network can be conceived as having a single start node, from which (usually) every ant starts, and a single finish node, reaching which indicates that the path taken encodes a complete solution. A clear example might be the network of roads in a city, where each junction of roads is a node in the network, and each road is an arc or edge between nodes. Consider the problem of finding the shortest path between a specific junction A and another specific junction B. In this case, A and B are clearly the start and finish nodes, and we can imagine an ACO approach which maps very closely indeed to the natural case. However, with a little thought, it is clear that we should constrain each individual ants' path construction so that it does not return to a junction it has already visited (unless this is a valid move for the problem under consideration). Also, though we might choose to simulate the preferential recruitment of new ants to shorter paths by closely following the natural case, it seems more sensible and straightforward to make the pheromone trail strength directly a function of the solution quality. That is, when an ant has completed its path, we evaluate the quality of its solution, and render things such that better solutions lead to stronger pheromone deposits along its arcs. These pheromone deposits will decay over time, however, just as in the natural case - we can see that this will prevent premature convergence to poor solutions that happen to be popular in the early stages.

Finally, since such information is often available to us, and would seem useful in cases where ants have large numbers of choices, we might bias the paths available at each junction with the aid of a simple heuristic evaluation of the potential of that arc. For a shortest path problem, for example, this could be based on how much closer to B each arc would leave the ant.

With such considerations in mind, we can envisage artificial ants travelling the road network from A to B via distinct but sensible routes. At each junction, the ant senses the pheromone levels that await it at each of the arcs it can feasibly take. These levels are made from many components; arcs that are highly attractive will probably enjoy the remnants of trails from prior ants that have reported good

solutions, and/or may have a good heuristic component. Arcs with low pheromone levels will probably be losers in the heuristic stakes, and have seen little activity that has led to good solutions; however, the ant may still choose such an arc, since our algorithm is stochastic.

Finally, as an individual artificial ant arrives at B, it retrospectively lays pheromone on the path it took, where the strength of that pheromone trail will reflect the quality of its solution. The next artificial ant starts from A and sees a slightly updated pheromone trail (stigmergic) pattern, and so it continues.

To apply this method to other problems, we simply need (implicitly, at least) a network-based representation of the problem as described. If we are solving the traveling salesperson problem (TSP), for example, the network is the complete graph of the cities, each arc between cities indicates the distance or cost of that arc, and in this case an individual ant can start anywhere. As we follow an individual ant's route, we sensibly constrain its potential next-hops to avoid cycles, and along the way we may bias its choices simply by using the distance of the next arc, and it retrospectively lays pheromone once it has completed a tour of all cities. In general, an optimisation problem can always be approached in this way, by suitable choices of semantics for nodes and arcs, and well designed routines for generating and constraining an ant's available choices at each junction.

We can now finish this explication by clarifying the AS algorithm, which in fact has already been covered verbosely in the above. Once the transformation of the problem has been designed, so that an ant's path through a network provides a candidate solution, the algorithm cycles through the following two steps until a suitable termination condition is reached:

**Solution Construction:** a number of ants individually construct solutions based on the current pheromone trail strengths (initially, pheromone is randomly distributed). Each ant steps through the network choosing among feasible paths. At each choice point, the ant chooses among available arcs according to a function of the pheromone strength on each arc, and of the heuristic values of each arc. In the original, and still commonly used version of this function (see Dorigo et al (1996) and many more), the pheromone and heuristic components for each arc are exponentiated to parameters  $\alpha$  and  $\beta$  respectively, allowing for tuning of the level to which the algorithm relies on exploration and heuristic support. Also, the overall attractiveness value of each arc is scaled so that the ant can treat these values as a probability distribution over its available choices.

**Pheromone Update:** When the ants' paths are complete, for each ant, the corresponding solution is evaluated, and pheromone is laid on each arc travelled in proportion to the overall solution quality. Also, the component of pheromone strength that arises from earlier ants is reduced. Quite simply, for any particular arc, its updated pheromone strength  $p_{\text{new}}$  is  $(1-\rho)p_{\text{old}} + \rho f$ , where  $\rho$  controls the speed of pheromone decay, and  $f$

accumulates the overall quality of solutions found which involved that arc in the current iteration.

ACO has now been applied to very many problems, and (clearly, or we would probably not have devoted so much time to it) has been very successful, especially when hybridized with local search or some other meta-heuristic (in such hybridized algorithms, an ant will typically use the additional heuristic for a short time to find an improvement locally to its solution). Initially demonstrated for the TSP (Dorigo et al, 1996), there are an enormous number of applications of ACO now published. We mention vehicle routing (Gambardella et al, 1999), rule discovery (Parpinelli et al, 2002), and protein/ligand docking (Korb et al, 2007), just to give some initial idea of the range of applications. To discover more, recent surveys include Blum (2005) and Gutjahr (2007), the latter concentrating on theoretical analyses. Meanwhile, Socha & Dorigo (2008) show how to apply ACO to continuous domains (essentially, ants select parameters via a probability density function, rather than a discrete distribution over a fixed set of arcs).

### 3.1.2 Ant-Based Routing in Telecommunications

The basic ACO idea of exploiting pheromone-trail based recruitment is also the inspiration for a healthy sub-area of research in communications networks; therein, ant-inspired algorithms are designed to assist with network routing and other network tasks, leading to systems that combine high performance with a high level of robustness, able to adapt with current network traffic and robust to network damage. Early and prominent studies in this line were by Schoonderwoerd et al (1996; 1997), which were soon built upon by di Caro and Dorigo's AntNet (1998). To explain this application area, and the way that ACO ideas are applied therein, it will be helpful to first explore the problem and the associated solution that was studied in Schoonderwoerd et al's seminal work.

Schoonderwoerd et al were concerned with load balancing in telecommunication networks. The task of a telecommunication network is to connect calls that can arise at any node, and which may need to be routed to any other node. The networks themselves, as a function of the capacities of the constituent equipment at each node, cannot guarantee successful call connections in all cases, but they aim to maintain overall acceptable performance under standard conditions. At very busy times, and/or if a particular node is overwhelmingly flooded with calls, then typically many calls will be 'dropped'. It is worth noting that there are problems with central control of such systems (landline telecommunication networks, mobile networks, traffic networks, and so forth ...). To achieve centralised control in a way that manages load-balancing or any other such target, several disadvantages are apparent. The controller usually needs current knowledge about many aspects of the entire system, which in turn necessitates using communication links from every part of the system to the controller itself. Centrally managed control mechanisms therefore scale badly as a result of the rapidly increasing overheads of this communication, interfering with

the performance of the system itself. Also, failure in the controller will often lead to complete failure of the complete system.

Schoonderwoerd et al's ant-inspired approach, which remains a central part of the majority of more recent ant-based approaches, was to replace the routing tables in such networks with so-called 'pheromone tables'. Networks of the type of interest invariably have a routing table at each node, specifying which 'next-hop' neighbouring node to pass an incoming call to, given the ultimate destination of that call. In Schoonderwoerd et al's 'Ant-Based Control' (ABC) method, the routing table at a network node instead provided  $n$  probability distributions over its neighbouring nodes, one for each of the  $n$  possible destinations in the network. When a routing decision is to be made, it is made stochastically according to these probabilities – i.e. it is most likely that the next-hop with the highest probability will be taken, but there is a chance that the next-hop with the lowest probability will be taken instead. The entries in the pheromone table were considered analogous to pheromone trail strengths, and changed adaptively during the operation of the network. Updating of these trails in ABC is very simple – whenever a call is routed from node  $A$  to  $B$ , the entries for  $A$  in node  $B$ 's routing table are all increased, with corresponding reductions to other entries. However this simple idea has obvious intuitive benefits; first, by testing various routing decisions over time (rather than deterministic decisions), the process effectively monitors the current health of a wide variety of different routing strategies; when a link is over-used, or down, this naturally leads to diminution in its probability of use, since the associated entries in routing tables will not be updated, and hence will naturally reduce as alternatives are updated. Also, as it turns out, the pheromone levels can adapt quite quickly to changes in call patterns and loads. The ABC strategy turned out to be surprisingly effective, despite its simplicity, when Schoonderwoerd et al compared it with a contemporary agent-based strategy developed by Appleby and Steward (1994), and found it superior over a wide range of different situations.

Research in ant-based approaches for decentralised management is increasingly very active (e.g. Hossein and Saadawi, 2003; Rosati et al, 2008; Di Caro et al, 2008). The essential idea, to replace static built-in routing strategies with stochastic 'pheromone tables' or similar, is applicable in almost all modern communication scenarios, ranging through ad-hoc computer networks, mobile telephone networks, and various layers of the internet. Ongoing research continues to explore alternative strategies for making the routing decisions, controlling the updates to pheromone trails, and so forth, while investigating various distinct application domains, and continuing to find competitive or better performance than alternative state of the art methods used in network engineering.

### 3.2 Particle Swarm Optimization and Foraging

Particle swarm optimization (PSO) was established in 1995 with Kennedy and Eberhart's paper in IJCNN (Kennedy & Eberhart, 1995). The paper described a rather simple algorithm (and time has seen no need to alter its straightforward fundamentals), citing Craig Reynolds' work as inspiration (Reynolds, 1987), along with slightly later work in the modelling of bird flocks (Heppner & Grenander, 1990). The basic idea is to unite the following two notions: (i) the behaviour of a flock of birds moving in 3D space towards some goal; (ii) a swarm of solutions to an optimisation problem, moving through the multidimensional search space towards good solutions.

Thus, we equate a 'particle' with a candidate solution to an optimization problem. Such a particle has both a *position* and a *velocity*. Its position is, in fact, precisely the candidate solution it currently represents. Its velocity is a displacement vector in the search space, which (it hopes) will contribute towards a fruitful change in its position at the next iteration.

The heart of the classic PSO algorithm is in the step which calculates a new position for the particle based on three influences. The inspiration from Reynolds (1987) is clear, but the details are quite different, and, of course, exploit the fact that the particle is moving in a search space and can measure the 'fitness' of any position. The influences - the components that lead to the updated position - are:

**Current velocity:** the particle's current velocity (obviously);

**Personal Best:** the particle remembers the fittest position it has yet encountered, called the personal best. A component of its updated velocity is the direction from its current position to the personal best;

**Global Best:** every particle in the swarm is aware of the best position that any particle has yet discovered (i.e. the best of the personal bests). The final component of velocity update, shared by all particles, is a vector in the direction from its current position to this globally best known position.

Following a random initialisation of positions and velocities, evaluation of the fitness of the particles' current positions, and consequent initialisation of the personal bests and global best, PSO proceeds in a remarkably straightforward manner. First, each particle updates its velocity by adding a vector in each of the above three component directions. To provide these vectors, in the classic algorithm, the current velocity component is left undisturbed, while the personal and global best components are each scaled by a random scalar drawn uniformly from [0,2]. The resulting vector is used to update the current velocity, and the new velocity vector is used to update the current position. The new position is evaluated, bookkeeping is done to update personal and global bests, and then we repeat.

Kennedy and Eberhart initially reported that PSO appeared to do very well over a wide range of test problems, including its use as an alternative to backpropagation for training an artificial neural network (Kennedy & Eberhart, 1995). Perhaps helped by the ease of implementation of this algorithm (remarkably few lines of code are needed for the classic algorithm), an avalanche of papers began to follow, almost invariably adding to the evidence that this algorithm provides a very substantial contribution to optimization practice. Naturally, this field is now rich in variants and extensions to the original design -- a number of recent surveys are available (e.g. Reyes-Sierra & Coello, 2006; Yang et al, 2007) -- while the published applications are as varied as one might expect from such a generally applicable algorithm.

### 3.2.1 Bacteria and Bees

Newer to the ranks of swarm-intelligence based optimisation, and yet to prove quite as widely successful, are techniques inspired by bacterial and bee foraging. For the most part, these algorithms follow the broad direction of PSO, in that individuals in a swarm represent solutions moving through a landscape, with the fitness of their current solution easily obtained by evaluating their position. Meanwhile, just as with PSO, an individual's movement through this landscape is influenced by the movements and discoveries of other individuals. The fine details of a Bacterial Foraging Algorithm (BFA), however, are quite distinct, and in one of the more popular methods draw quite closely from what is known (and briefly touched upon above) about bacterial swarming in nature. Passino (2002) presents a fine and detailed explication of both the natural case and the BFA. It turns out that BFA-style algorithms are enjoying quite some success in recent application to a range of engineering problems (e.g. Niu et al, 2006; Tripathy & Mishra, 2007; Guney & Basbug, 2008).

Also inspiring, so far, a small following are algorithms that are inspired by bee foraging behaviour. The authoritative sources for this are Quijano and Passino's papers respectively outlining the design and theory (Quijano & Passino, 2007a) and application (Quijano & Passino, 2007b) of a bee foraging algorithm. In Quijano & Passino (2007a) the design of a bee foraging algorithm is presented in intimate connection with an elaboration of the mechanisms of natural bee foraging (such as we briefly described earlier). The algorithm is as much a model of the natural process as it is a routine applicable to certain kinds of problem. Considering individual bees as resources, the concept here is to use bee foraging behaviour as a way to ideally distribute those resources in the environment, and maintain an ideal distribution over time as it adapts to changing patterns of supply. Just as natural bees maintain an efficient distribution of individuals among the available sources of nectar, the idea is that this can be mapped to control problems which aim to maintain a distribution of resources (such as power or voltage) in such a way that some goal is maintained (such as even temperature or maximal efficiency). In Quijano & Passino, 2007b), we see the algorithm tested successfully on a dynamic voltage allocation problem, in which the task is to

maintain a uniform and maximal temperature across an interconnected grid of temperature zones.

Finally, we note that bee foraging behaviour has also directly inspired techniques for internet search, again, based on the notion of maintaining a maximally effective use of server resources, adapting appropriately and effectively to the relative richness of new discoveries (Walker, 2000; Nakrani & Tovey, 2003).

#### 4. Current Trends and Concluding Notes

We have pointed to a number of survey papers and other works from which the reader can attain a full grasp of the current activity in swarm intelligence algorithms, but in this brief section we attempt a few notes that outline major current trends, and then we wrap up.

A notable trend in recent work on particle swarm optimisation, and indeed on metaheuristics in general, is towards the creation of hybrid algorithms. While themes from evolutionary computation continue to be incorporated in PSO (Shi et al, 2005), others have explored the idea of hybridization with less frequently used techniques such as scatter search and path relinking (Yin et al, 2007), immune system methods (Zhang & Wu, 2008) and, indeed, ant colony optimization (Holden & Freitas, 2007). Meanwhile, the range of problems to which PSO may be applied has been greatly increased with the development of multi-objective forms of PSO (Coello et al, 2004).

Other work has involved the use of multiple swarms. This may allow each swarm to optimize a different section of the solution (van den Bergh & Engelbrecht, 2004). Alternatively, each swarm may be configured differently to take advantage of the strengths of different PSO variants (e.g. Jordan et al, 2008), in an attempt to create a more reliable algorithm that can be applied to a wide range of problem domains.

The themes of multi-objective optimization and hybridization equally apply to recent research into ant-colony optimisation. While multi-objective ACO is a more mature field than multi-objective PSO (see, for example, MARIANO & Morales, 1999), work continues in categorizing and comparing multi-objective approaches to ACO (e.g. Garcia-Martinez et al, 2007), in creating generic frameworks for multi-objective ACO and in creating new multi-objective variants (e.g. Alaya, 2007). Recent applications have seen single objective ACO hybridized with genetic algorithms (Lee et al, 2008), beam search (Blum, 2005b), and immune systems (Yuan et al, 2008) and multi-objective ACO used in combination with dynamic programming (Häckel et al, 2008) and integer linear programming (Doerner et al, 2006).

Other recent work has seen ACO adapted for use in continuous domains (Dreo Siarry, 2006; Socha & Dorigo, 2008), while research continues into variations of ACO and new algorithm features, for example, different types of pheromone and the use of dominance rules to warn ants from searching amongst solutions known to be of low quality (Lin et al, 2008).

Recent work on bacterial foraging algorithms has concentrated on exploiting the effectiveness of the local search ability of the algorithm, while adapting it improve the global search ability on high dimensional and multi-modal problems. With this aim, bacterial foraging has been hybridized with more effective global optimizers such as genetic algorithms (Chen et al, 2007; Kim et al, 2007) and particle swarm optimization (Tang et al, 2007; Biswas et al, 2007).

In conclusion, we have attempted to demystify the concept of swarm intelligence, and, after touring through the chief sources of natural inspiration, distilled the essence of its impact and presence in computer science down to two major families of algorithms for optimisation. No less intriguing and exciting additional topics in the swarm intelligence arena, that we have also discussed, are stigmergic construction, ant-based clustering, and swarm robotics. It is abundantly clear that the natural inspirations from swarming ants, bees and birds (among others) have provided us with new ideas for optimisation algorithms that have extended the state of the art in performance on many problems, sometimes with and sometimes without additional tailoring and hybridization. Ant-based clustering seems also to provide a valuable contribution, while swarm robotics, stigmergy based construction, and a variety of other emerging subtopics have considerable promise, and will doubtless develop in directions rather difficult to foresee.

## References

- Inés Alaya (2007) Ant Colony Optimization for Multi-objective Optimization Problems, in Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence, pages 450–457, 2007.
- Appleby, S., & Steward, S. (1994). Mobile software agents for control in telecommunications Networks. *BT Technology Journal*, Vol. 12, No.2.
- H. Berg and D. Brown (1972) Chemotaxis in escherichia coli analysed by three-dimensional tracking. *Nature*, **239**, pp. 500–504, 1972.
- Arijit Biswas, Sambarta Dasgupta, Swagatam Das, and Ajith Abraham. (2007), Synergy of PSO and Bacterial Foraging Optimization --- A Comparative Study on Numerical Benchmarks, in E. Corchado et al., ed., *Innovations in Hybrid Intelligent Systems*, no. 44 in *Advances in Soft computing*, pp. 255–263, 2007.
- Christian Blum (2005a) Ant colony optimization: Introduction and recent trends, *Physics of Life Reviews*, **2**(4), pp. 353–373, 2005
- Christian Blum (2005b) Beam-ACO---hybridizing ant colony optimization with beam search: an application to open shop scheduling, *Computers & Operations Research*, pages 1565–1591, 2005.
- Eric Bonabeau, Sylvain Guérin, Dominique Snyers, Pascale Kuntz and Guy Theraulaz. (2000) Three-dimensional architectures grown by simple ‘stigmergic’ agents, *Biosystems*, **56**, pp. 13–32, 2000.
- Eric Bonabeau, Guy Theraulaz, Jean-Louis Deneubourg, Serge Aron, and Scott Camazine (1997), Self-organization in social insects, *Trends in Ecology and Evolution*, **12**(5), pp. 188–193, 1997.
- Eric Bonabeau, Guy Theraulaz, Jean-Louis Deneubourg, Nigel R. Franks, Oliver Rafelsberger, Jean-Louis Joly and Stephane Blanco (1998) A model for the

- emergence of pillars, walls and royal chambers in termite nests *Philosophical Transactions of the Royal Society B: Biological Sciences*, **353**(1375), pp. 1561—1576, 1998.
- E. Budrene and H. Berg (1991) Complex patterns formed by motile cells of escherichia coli. *Nature*, **349**, pp. 630—633, 1991.
- Tai-Chen Chen, Pei-Wei Tsai, Shu-Chuan Chu, and Jeng-Shyang Pan (2007), *A Novel Optimization Approach: Bacterial-GA Foraging*, in *Proceedings of the Second International Conference on Innovative Computing, Information and Control (ICICIC)*, page 391. IEEE Computer Press, 2007.
- Vincent A. Cicirello and Stephen F. Smith (2001) Wasp nests for self-configurable factories, in *Proc. 5th Int. Conf. on Autonomous Agents*, pp. 473—480, ACM, 2001.
- Carlos A.~Coello Coello, Gregorio~Toscano Pulido, and Maximino~Salazar Lechuga (2004) Handling Multiple Objectives With Particle Swarm Optimization, *IEEE Transactions on Evolutionary Computation*, **8**(3):256--279, 2004.
- Colorni A., M. Dorigo & V. Maniezzo (1992a). Distributed Optimization by Ant Colonies. *Proceedings of the First European Conference on Artificial Life*, Paris, France, F.Varela and P.Bourgine (Eds.), Elsevier Publishing, 134-142.
- Colorni A., M. Dorigo & V. Maniezzo (1992b). An Investigation of some Properties of an Ant Algorithm. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92)*, Brussels, Belgium, R.Männer and B.Manderick (Eds.), Elsevier Publishing, 509-520.
- J.-L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain and L. Chretien. (1991) The dynamics of collective sorting: robot-like ants and ant-like robots, in *From Animals to Animats: Proc. 1st Int. Conf. on Simulation of Adaptive Behaviour*, Jean Arcady-Meyer and Stewart Wilson (eds.), MIT Press, Cambridge, pp. 356—365, 1991.
- S. Depickere, D. Fresneau and J.-L. Deneubourg (2004) Dynamics of aggregation in Lasius niger (Formicidae): influence of polyethism, *Insect. Sociaux*, **51**(1), 81–90, 2004.
- D. DeRosier (1998) The turn of the screw: The bacterial flagellar motor, *Cell*, **93**, pp. 17—20, 1998.
- Gianni Di Caro and Marco Dorigo (1998) AntNet: Distributed Stigmergetic Control for Communications Networks. *JAIR*, **9**, pp. 317—365, 1998.
- Giann Di Caro, Frederick Ducatelle and Luca M. Gambardella, Theory and practice of Ant Colony Optimization for routing in dynamic telecommunications networks, in Sala N., Orsucci F. (Eds.), *Reflecting Interfaces: the Complex Coevolution of Information Technology Ecosystems*, Idea Group, Hershey, PA, USA, 2008
- K.F. Doerner, W.J. Gutjahr, R.F. Hartl, C.Strauss, and C.Stummer (2006), Pareto ant colony optimization with ILP preprocessing in multiobjective project portfolio selection, *European Journal of Operational Research*, **171**:830--841, 2006.
- Dorigo M., V. Maniezzo & A. Colorni (1991). The Ant System: An Autocatalytic Optimizing Process. *Technical Report No. 91-016 Revised*, Politecnico di Milano, Italy
- Marco Dorigo, V. Maniezzo and A. Colorni (1996) Ant system: optimization by a colony of co-operating agents. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, **26**(1), pp. 29—41, 1996.
- J. Dreо and P. Siarry, An ant colony algorithm aimed at dynamic continuous optimization, *Applied Mathematics and Computation*, **181**:457–467, 2006.
- Ducatelle F., Förster A., Di Caro G., Gambardella L.M. New task allocation methods for robotic swarms, In *9th IEEE/RAS Conference on Autonomous Robot Systems and Competitions*, May, 2009

- N.R. Franks and A. Sendova-Franks, Brood sorting by ants: distributing the workload over the work-surface. *Behav. Ecol. Sociobiol.*, **30**(2), 109–123, 1992.
- Karl von Frisch, *The Dance Language and Orientation of Bees*, Harvard University Press, Cambridge, MA, 1967.
- Luca Maria Gambardella, Éric Taillard and Giovanni Agazzi, MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows, in *New Ideas in Optimization*, D. Corne, M. Dorigo, F. Glover (eds.), McGraw Hill, London, pp. 63–76, 1999.
- C. García-Martínez, O. Cordón, and F. Herrera, A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP, *European Journal of Operational Research*, **180**:116–148, 2007.
- Laurent Gaubert, Pascal Redou, Fabrice Harrouet and Jacques Tisseau, A first mathematical model of brood sorting by ants: Functional self organisation without swarm-intelligence. *Ecological Complexity*, **4**, pp. 234–241 2007.
- Pierre-Paul Grassé, La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes Natalensis et Cubitermes sp.* La théorie de la stigmergie: essai d'interprétation du comportement des termites constructeurs, *Insect Soc.*, **6**, pp. 41–84, 1959.
- Pierre-Paul Grassé, *Termitologia, Tome II -- Fondation des sociétés construction*, Paris: Masson, 1984.
- K. Guney and S. Basbug, Interference suppression of linear antenna arrays by amplitude-only control using a bacterial foraging algorithm, *Progress In Electromagnetics Research*, **79**, pp. 475–497, 2008.
- Walter J. Gutjahr, Mathematical runtime analysis of ACO algorithms: survey on an emerging issue, *Swarm Intelligence*, **1**(1), pp. 59–79, 2007.
- Sascha Häckel, Marco Fischer, David Zechel, and Tobias Teich, A Multi-Objective Ant Colony Approach for Pareto-Optimization Using Dynamic Programming, in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 33–40. ACM, 2008.
- Julia Handl and Bernd Meyer, Ant-based and swarm-based clustering, *Swarm Intelligence*, **1**(2), pp. 95–113, 2007.
- Julia Handl, Joshua Knowles and Marco Dorigo, Ant-based clustering and topographic mapping. *Artificial Life*, **12**(1), pp. 35–61, 2006.
- F. Heppner and U. Grenander, A stochastic nonlinear model for coordinated bird flocks, in *The Ubiquity of Chaos*, S. Krasner (ed.), AAAS Publications, Washington, 1990.
- Nicholas Holden and Alex A. Frietas, A Hybrid PSO/ACO Algorithm for Classification, in *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation*, pages 2745–2750, 2007.
- O. Hussein and T. Saadawi., Ant routing algorithm for mobile ad-hoc networks (ARAMA). In *Proc. IEEE Conf. on Performance, Computing and Communications*, pp. 281–290, 2003.
- Johannes Jordan, Sabine Helwig, and Rolf Wanka, Social Interaction in Particle Swarm Optimization, the Ranked FIPS, and Adaptive Multi-Swarms, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 49–56, 2008.
- P. Karlson and M. Luscher, Pheromones: a new term for a class of biologically active substances, *Nature*, **183**, pp. 155–176, 1959..
- James Kennedy and Russ Eberhart, Particle swarm optimization, in *Proc. IEEE International Joint Conference on Neural Networks*, IEEE Press, Piscataway, pp. 1942–1948, 1995.

- Dong-Hwa Kim, Ajith Abraham, and Jae-Hoon Cho, A hybrid genetic algorithm and bacterial foraging approach for global optimization, *Information Sciences*, **177**:3918–3937, 2007.
- Oliver Korb, Thomas Stützle and Thomas E. Exner, An ant colony optimization approach to flexible protein-ligand docking, *Swarm Intelligence*, **1**(2), pp. 115–134, 2007.
- Zne-Jung Lee, Shun-Feng Su, Chen-Chia Chuang, and Kuan-Hung Liu, Genetic algorithm with ant colony optimization (GA-ACO) for multiple sequence alignment, *Applied Soft Computing*, **8**:55–78, 2008.
- B.M.T. Lin, C.Y. Lu, S.J. Shyu, and C.Y. Tsai, Development of new features of ant colony optimization for flowshop scheduling, *International Journal of Production Economics*, **112**:742–755, 2008.
- E. Lumer and B. Faieta, Diversity and adaptation in populations of clustering ants, in *From Animals to Animats 3: Proc. of 3rd Int. Conf. on Simulation of Adaptive Behaviour*, D. Cliff et al (eds.), MIT Press, pp. 501–508, 1994.
- C.E. Mariano and E.-Morales, MOAQ: An Ant-Q algorithm for multiple objective optimization problems, in W. Banzhaf, J. Daida, A. E. Eiben, M.H. Garzon, V. Hnavař, M.-Jakielka, and R.E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computing Conference (GECCO 99)*, pages 894–901, 1999.
- Francesco Mondada, Luca Maria Gambardella, Dario Floreano, Stefano Nolfi, Jean-Louis Deneubourg and Marco Dorigo, The Cooperation of Swarm-Bots: Physical Interactions in Collective Robotics, in *IEEE Robotics and Automation Magazine*, **12**(2), pp. 21–28, 2005.
- S. Nakrani and C. Tovey, on honey bees and dynamic allocation in an internet server ecology, in *Proc. 2nd Int'l Workshop on the Mathematics and Algorithms of Social Insects*, 2003.
- B. Niu, Y. Zhu, X. He, and X. Zeng, Optimum design of PID controllers using only a germ of intelligence, in *Proc. 6th World Congress on Intelligent Control and Automation*, pp. 3584–3588, 2006.
- Rafael S. Parpinelli, Heitor S. Lopes and Alex A. Freitas, Data Mining With an Ant Colony Optimization Algorithm, *IEEE Transactions on Evolutionary Computation*, **6**(4), pp. 321–332, 2002.
- Partridge, B.L., The Structure and Function of Fish Schools, **Scientific American**, June 1982, pp. 114–123.
- Kevin M. Passino, Biomimicry of Bacterial Foraging for distributed optimization and control, *IEEE Control Systems Magazine*, pp. 52–68, June, 2002.
- Pini G., Brutschy A., Birattari M., Dorigo M. Interference Reduction Through Task Partitioning in a Robotic Swarm, In *6th International Conference on Informatics in Control, Automation and Robotics (ICINCO 09)*, 2009
- Potts, W.K., The Chorus-Line Hypothesis of Manoeuvre Coordination in Avian Flocks, letter in Nature, Vol. **309**, May 24, 1984, pp. 344–345.
- Nicanor Quijano and Kevin M. Passino, Honey Bee Social Foraging Algorithms for Resource Allocation, Part I: Algorithm and Theory, in *Proc. 2007 American Control Conference*, pp. 3383–3388, 2007a.
- Nicanor Quijano and Kevin M. Passino, Honey Bee Social Foraging Algorithms for Resource Allocation, Part II: Application, in *Proc. 2007 American Control Conference*, pp. 3389–3394, 2007b.
- Margarita Reyes-Sierra and Carlos A. Coello Coello, Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art, *International Journal of Computational Intelligence Research*, **2**(3), pp. 287–308, 2006.
- Craig Reynolds, Flocks, Herds and Schools: A Distributed Behavioral Model, *Computer Graphics*, **21**(4), pp. 25–34, 1987.

- Roberts J., Zufferey J., Floreano D. Energy Management for Indoor Hovering Robots In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2008)*, IEEE (editors), September, 2008.
- Laura Rosati, Matteo Bertioli and Gianluca Reali, On ant routing algorithms in ad hoc networks with critical connectivity, *Ad Hoc Networks*, **6**(6), pp. 827—859, 2008.
- Erol Sahin, Swarm Robotics: From Sources of Inspiration to Domains of Application, in *Swarm Robotics*, Springer LNCS 3342, pp. 10—20, 2005.
- R. Schoonderwoerd, O. Holland, J. Bruton, and L. Rothkrantz, Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, **5**(2):169—207, 1996.
- R. Schoonderwoerd, O. Holland, and J. Bruton. Ant-like agents for load balancing in telecommunications networks. In *Proceedings of the First International Conference on Autonomous Agents*, pages 209—216, ACM Press, 1997.
- J. Segall, S. Block and H. Berg, Temporal comparisons in bacterial chemotaxis, *PNAS*, **83**, pp. 8987—8991, 1986.
- X.H. Shi, Y.C. Liang, H.P. Lee, C.Lu, and L.M. Wang, An improved GA and a novel PSO-GA-based hybrid algorithm, *Information Processing Letters*, **93**:255—261, 2005.
- Krzysztof Socha and Marco Dorigo, Ant colony optimization for continuous domains, *European Journal of Operational Research*, **185**:1155—1173, 2008.
- W.J. Tang, Q.H. Wu, and J.R. Saunders, A Bacterial Swarming Algorithm For Global Optimization, in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 1207—1212, IEEE Service Center, Piscataway, NJ, 2007.
- Guy Theraulaz, Du super-organisme a l'intelligence en essaim: modeles et representations du fonctionnement des societes d'insectes, in *Intelligence Collective* (Eric Bonabeau and Guy Theraulaz, eds.), pp. 29—109, Paris: Hermes, 1994.
- Guy Theraulaz and Eric Bonabeau. Modelling the Collective Building of Complex Architectures in Social Insects with Lattice Swarms. *Journal of Theoretical Biology*, **177**(4), pp. 381—400, 1995.
- Guy Theraulaz, Eric Bonabeau, Stamatios C. Nicolis, Ricard V. Sole, Vincent Fourcassie, Stephane Blanco, Richard Fournier, Jean-Louis Joly, Pau Fernandez, Anne Grimal, Patrice Dalle and Jean-Louis Deneubourg., Spatial patterns in ant colonies. *PNAS*, **99**(15), 2002.
- M. Tripathy and S. Mishra, Bacteria foraging-based solution to optimize both real power loss and voltage stability limit, *IEEE Transactions on Power Systems*, **22**(1), 2007.
- Frans van den Bergh and Andries~P. Engelbrecht, A Cooperative Approach to Particle Swarm Optimization, *IEEE Transactions on Evolutionary Computation*, **8**(3):225—239, 2004.
- Robert K. Vander Meer and Leeanne E. Alonso, Pheromone Directed Behaviour in Ants, in *Pheromone Communication in Social Insects*, Vander Meer et al. (eds.), Westview Press, Boulder, CO, pp. 159--192, 1998a.
- Robert K. Vander Meer, M. Breed, M. Winston and K.E. Espelie (eds), *Pheromone Communication in Social Insects*, Westview Press, Boulder, CO, 368pp., 1998b.
- Waibel M., Keller L., Floreano D. (2009) Genetic Team Composition and Level of Selection in the Evolution of Multi-Agent Systems, *IEEE Transactions on Evolutionary Computation*, **13**(3): 648—660, 2009
- Reginald L. Walker, Dynamic Load Balancing Model: Preliminary Assessment of a Biological Model for a Pseudo-Search Engine, in *Parallel and Distributed Processing*, LNCS vol. 1800, Springer, pp. 620—627, 2000.
- Bo Yang, Yunping Chen and Zunlian Zhao, Survey on Applications of Particle Swarm Optimization in Electric Power Systems, in *IEEE Int'l Conf. on Control and Automation*, pp. 481—486, 2007.

- Peng-Yeng Yin, Fred Glover, Manuel Laguna, and Jia-Xian Zhu, Scatter PSO --- A More Effective Form of Particle Swarm Optimization, in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 2289—2296, 2007.
- Hua Yuan, Yi-Li, Wenhui Li, Kong Zhao, Duo Wang, and Rongqin Yi, *Combining Immune with Ant colony Algorithm for Geometric Constraint Solving*, in *Proceedings of the 2008 Workshop on Knowledge Discovery and Data Mining*, pages 524—527, IEEE Computer Society, 2008.
- Rui Zhang and Cheng Wu, An Effective Immune Particle Swarm Optimization Algorithm for Scheduling Job Shops, in *Proceedings of the 3rd IEEE Conference On Industrial Electronics and Applications*, pages 758—763, 2008.



## Review

## A survey: Ant Colony Optimization based recent research and implementation on several engineering domain

B. Chandra Mohan <sup>\*</sup>, R. Baskaran

*Department of Computer Science & Engineering, Anna University, Chennai, India*

## ARTICLE INFO

**Keywords:**  
 Swarm Intelligence  
 Ant Colony Optimization  
 Soft-computing  
 Engineering applications

## ABSTRACT

Ant Colony Optimization (ACO) is a Swarm Intelligence technique which inspired from the foraging behaviour of real ant colonies. The ants deposit pheromone on the ground in order to mark the route for identification of their routes from the nest to food that should be followed by other members of the colony. This ACO exploits an optimization mechanism for solving discrete optimization problems in various engineering domain. From the early nineties, when the first Ant Colony Optimization algorithm was proposed, ACO attracted the attention of increasing numbers of researchers and many successful applications are now available. Moreover, a substantial corpus of theoretical results is becoming available that provides useful guidelines to researchers and practitioners in further applications of ACO. This paper review varies recent research and implementation of ACO, and proposed a modified ACO model which is applied for network routing problem and compared with existing traditional routing algorithms.

© 2011 Elsevier Ltd. All rights reserved.

### 1. Introduction

Swarm Intelligence (SI) is a growing discipline of field of study which contains relatively optimal approach than the traditional approach for problem solving of almost all engineering domain. SI is developed from the imitations which are learned from the social behaviours of insects and animals, for example: ACO, Artificial Honey Bee (ABC), Fire Flies (FF), and Honey Bot. In which the ACO, is the field of “Ant Algorithm” studies models which is learned from the behavioural observation of real ant colonies. The ACO models used for the design of novel algorithms for the solution of optimization and distributed control problems. Foraging behaviour, division of labour, brood sorting, and co-operative transport are the several different aspects of the behaviour of ant colonies have inspired from the real ants’ and based on these inspiration different kinds of Ant Algorithms are proposed in the recent years. In which, the ACO is inspired by the foraging behaviour of ant colonies, and targets the discrete optimization problems.

The French Entomologist named Pierre-Paul Grasse observed that some species of termites react, which termed as “significant stimuli”. The term stigmergy is used to describes the particular type of communication in which the “workers are stimulated by the performance they have achieved”. Now the term, stigmergy is used for indirect, non-symbolic form of communication mediated by the environment. This stigmergy is achieved in the ACO using a chemical substance called pheromone, this chemical sub-

stance is deposited on the ground when ants walking to and from a food source. Other ants perceive the presence of this pheromone and tend to follow the routes where pheromone concentration is higher. Through this mechanism, ants are able to identify and transport food to their nest in a remarkably effective and easy way.

Pasteels, Deneubourg, and Goss (1987) thoroughly investigated the pheromone laying behaviour of the real ants in the experiment called as “double bridge experiment”. In this double bridge model, the nest was connected to a food source by two bridges of equal lengths. The author used the term argentine ants for the ants which identifies the route, simply says these ants are the predictor or scout of their colony. In such a setting, ants start to explore the surroundings of the nest and eventually reach the food source. Along their route between food source and nest, argentine ants deposit pheromone. Initially, each ant randomly chooses one of the two bridges. In the later stages due to random fluctuations, one of the two bridges presents a higher concentration of pheromone than the other bridge and therefore attracts more ants. This behaviour increases a further amount of pheromone on that bridge which makes more attractive. Therefore, after some time the whole colony converges toward for use the higher concentrated bridge for their transport.

Goss, Aron, Deneubourg, and Pasteels (1989) considered a variant of the above double bridge experiment in which one bridge is significantly longer than the other; refer the double bridge in the Fig. 1. In this case, the ants choosing by chance the short bridge are the first to reach the nest. Therefore, the short bridge receives more density of pheromones earlier than the long bridge and this fact will increases the probability of shorter bridge for choosing

\* Corresponding author. Mobile: +91 9976611188.

E-mail address: [abc.phd@hotmail.com](mailto:abc.phd@hotmail.com) (B. Chandra Mohan).



**Fig. 1.** A double bridge method for ACO with varying length routes R1 and R2.

by the further ants to select it. Deneubourg, Aron, Goss, and Pasteels (1990) developed a probability model of the observed behaviour of real ant. In which, assuming that at a given moment of time,  $m_1$  ants have used the first bridge and  $m_2$  ants have used the second one, the probability ' $p_1$ ' for an ant to choose the first bridge is:

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h}, \quad (1)$$

where parameters  $k$  and  $h$  are constant and which is to be fitted to the experimental data. By changing these  $k$  and  $h$  the impact of shorter path and the impact of less congestion path can be achieved. In the double bridge experiment, the probability of the other bridge is  $p_2$ , which is  $p_2 = 1 - p_1$ .

The real ant and artificial ants are differed in few assumptions, in the real ant behaviour the pheromone intensity is reduced over time as the pheromone is the chemical substance and so it evaporates over time. However, in the ACO, this can be set to a constant rate, this pheromone evaporation reduces the influence of the pheromones deposited in the early stages of the search, and this property is very useful for adaptive route search in such a situation that frequent path failures.

Ant System, Ant Colony System and Ant Net proposed by (Dorigo and Gambardella, 1997; Dorigo, Maniezzo, & Colomi, 1996; Dorigo & Stutzle, 2004) are the significant implementation of ACO. Dorigo et al. (1996) applied the simple probability rule and Dorigo and Gambardella (1997) applied the state transition rule for the decision model. According to Dorigo et al. (1996), Neto and Filho (2011), the following characteristics of Ant Model is described,

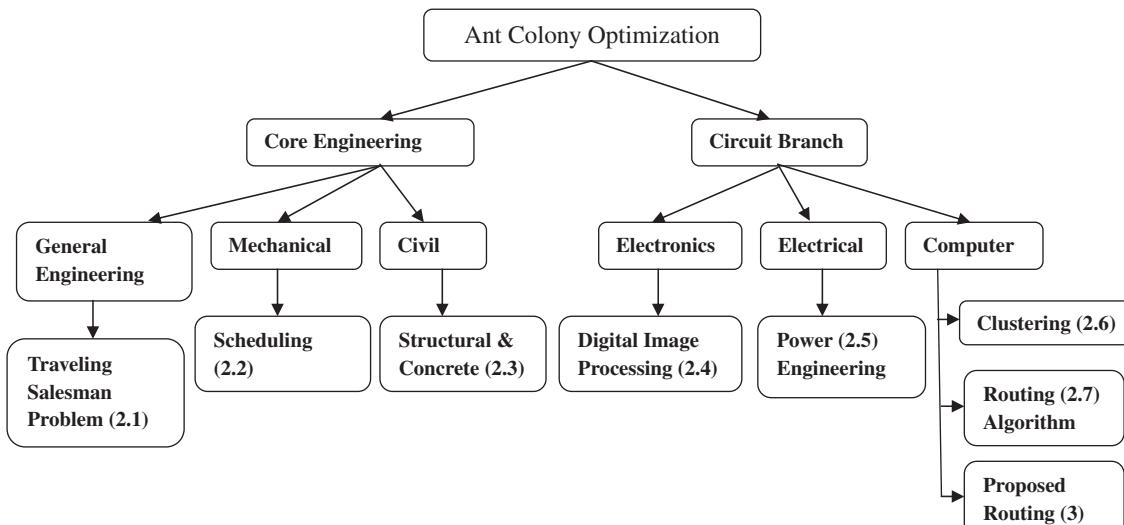
- The Ants exists in an environment represented mathematically as a graph, the ant always occupies a node in a graph which represents a search space. This node is called  $nf$ .

- It has an initial state.
- Although it cannot sense the whole graph, it can collect two kinds of information about the neighbourhood, (1) the weight of each trail linked to  $nf$ ; and (2) the characteristics of each pheromone deposited on this trail by other ants of the same colony.
- Moves toward a trail  $C_{ij}$  that connects nodes ' $i$ ' and ' $j$ ' of the graph.
- Also, it can alter the pheromones of the trail  $C_{ij}$ , in an operation called as "deposit of pheromone levels".
- It can sense the pheromone levels of all  $C_{ij}$  trails that connects a node  $i$ .
- It can determine a set of "prohibited" trails.
- It presents a pseudo-random behavior enabling the choice among the various possible trails.
- This choice can be (and usually is) influenced by the level of pheromone.
- It can move from node  $i$  to node  $j$ .

Dorigo and Stutzle (2004) redefined the pheromone update policy of ACO, and the term argentine ant is replaced with forward ant. Furthermore, there are some ACO approaches that adopt the privileged pheromone lying in which ants only deposit pheromones during their return trips. Simple ACO (SACO) uses two working ant model called forward ant and backward ant, the probabilistic forward ant generated in the nest and flooded towards food source. The forward ant do not deposit pheromone while moving, this helps in avoiding the formation of loops. Once the forward ant reaches its destination, it switched to the backward ant and copies the route information from the forward ant. Then the backward ant moves to the nest using the information copied from the forward node. Further in this paper, the research and application of ACO is explained, the organization of this paper is presented in the Fig. 2. This paper is the extended version of our previous reviews (Chandra Mohan & Baskaran, 2011e) with some more recent advancement in the survey of ACO and the proposed ACO based routing for the wide spread coverage and for helping the researchers community.

## 2. Review on recent research in ACO on various engineering domain

This paper reviews the recent systematic approach of ACO on various engineering field of domain on various factors like research



**Fig. 2.** Organization of sections (sub-section is shown in bracket).

issues, application and implementations, papers which are published in 2010 and after only considered in this survey.

## 2.1. Traveling salesman problem

Let  $V = \{a, \dots, z\}$  be a set of cities,  $A = \{(r, s) : r, s \in V\}$  be the edge set, and  $\delta(r, s) = \delta(s, r)$  be a cost measure associated with edge  $(r, s) \in A$ . The TSP is the problem of finding a minimal cost closed tour that visits each city once. In the case cities  $r \in V$  are given by their co-ordinates  $(x_r, y_r)$ , and  $\delta(r, s)$  is the Euclidean distance between  $r$  and  $s$ , then it is an Euclidean TSP. If  $\delta(r, s) \neq \delta(s, r)$  for at least some  $(r, s)$  then the TSP becomes an Asymmetric TSP (ATSP). The dynamic TSP (DTSP) is a TSP in which cities can be added or removed at run time. The goal is to find the shortest tour as early as possible after each and every iteration in order to update the link failures.

**Manuel and Blumb (2010)** proposed a Beam-ACO for the traveling salesman problem with time windows. This authors deals with the minimization of the travel-cost using a Beam-ACO algorithm, which is a hybrid method combining ACO with beam search. In general, Beam-ACO algorithms heavily rely on accurate and computationally inexpensive bounding information for differentiating between partial solutions. The Beam ACO uses the stochastic sampling as a useful alternative, which is evaluated on seven benchmark sets. The Beam-ACO algorithm is currently a state-of-the-art technique for the traveling salesman problem with time windows when travel-cost optimization is concerned. **Neumann and Witt (2010)**, proposed an ACO for the minimum spanning tree problem. The authors presented the first comprehensive rigorous analysis of a simple ACO algorithm for a combinatorial optimization problem which consider the minimum spanning tree (MST) problem and examines the effect of two construction graphs with respect to the runtime behaviour. **You, Liu, and Wang, (2010a)** proposed Parallel Ant Colony Optimization algorithm (PQACO) based on quantum dynamic mechanism for traveling salesman problem. In PQACO, the author explains the Parallel Evolutionary Algorithms (PEA) can be classified into three different models: Master-slaves PEA, Fine-grained PEA, Coarse-grained PEA and used this model for the travelling salesman problem of quantum computing applications.

**Andziulis, Dzemydiene, and Steponavičius (2011)** proposed a production scheduling problems, belongs to the class of ATSP. The authors proposes Nearest Neighbor (NN) and ACO for solving ATSP, it is tested on a specific real-life problem. The performances of the NN and the ACO techniques were evaluated and compared using two criteria, (1) the minimum value of the objective function achieved and (2) the CPU time. This paper concludes that the ACO algorithm works better than NN if looking at the achieved minimum values of the objective function and in the other hand, the computational time of the ACO algorithm is slightly longer.

For TSP, numerous techniques such as Genetic Algorithms (GA), Evolution Strategies (ES), Simulated Annealing (SA), Ant Colony Optimization (ACO), Particle Swarm Optimizers (PSO) are used to solve in the large-scale optimization problems. In this some of them are time consuming, and the time complexity algorithms could not find the optimal solution. Therefore, for making trade-off between time consumption and optimal solution combining two or more algorithms in order to improve solutions quality and reduce execution time is a mandatory process. **Elhaddad and Sallabi (2011)** propose new operations and techniques are used to improve the performance of GA and then combined the improved GA with SA to implement a hybrid algorithm (HGSAA) to solve TSP. The authors discussed that the TSP is the most well-known NP-hard problem and is used as a test bed to check the efficacy of any combinatorial optimization methods. In this work, the hybrid algorithm was tested using known instances from TSPLIB

using library of sample instances for the TSP at the internet and the results are compared against some recent related works. The comparison is shows that the HGSAA is effective in terms of results and time.

## 2.2. Scheduling

Job Scheduling problems have a vital role in recent years due to the growing consumer demand for variety, reduced product life cycles, changing markets with global competition and rapid development of new technologies. The Job Shop Scheduling Problem (JSSP) is one of the most popular scheduling models existing in practice, which is among the hardest combinatorial optimization problems. The definition of job scheduling problem is as follows:

- A number of independent (user/application) jobs to be scheduled.
- A number of heterogeneous machines candidates to participate in the planning.
- The workload of each job (in millions of instructions).
- The computing capacity of each machine (in mips).
- Ready time indicates when machine  $m$  will have finished the previously assigned jobs.
- The Expected Time to Compute (ETC) matrix ('nb' jobs  $\times$  'nb' machines) in which ETC  $[i][j]$  is the expected execution time of job ' $i$ ' in machine ' $j$ '.

Many approaches, such as, Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA), Ant Colony Optimization (ACO), Neural Network (NN), Evolutionary Algorithm (EA) and other heuristic approach have been successfully applied to JSSP. For improving the performance of EA, several researches integrated some optimization strategies into the EA. Also, the study of interaction between evolution and learning for solving optimization problems has been attracting much attention. The diversity of these approaches is shown in **Xing, Chen, Wang, Zhao, and Xiong (2010)**, in which the authors developed a frame work called Knowledge-Based Heuristic Searching Architecture (KBHSA). This framework integrates the knowledge model and the heuristic searching model to search an optimal solution. The performance of this architecture in the instantiation of the Knowledge-Based Ant Colony Optimization (KBACO) is applied to the common benchmark problems. The experimental results of KBACO algorithm outperform the previous approaches when solving the FJSSP.

**Chen, Shi, Teng, Lan, and Hu (2010)** proposed an efficient hybrid algorithm for resource-constrained project scheduling. This hybrid algorithm is known as the ACROSS algorithm which combines Scatter Search (SS) with ACO. Research on ACO has shown that improved performance can be obtained by stronger exploitation of the best solutions found during the search (**Berrichi, Yalaoui, Amodeo, & Mezghiche, 2010; Komarudin & Wong, 2010**). For improving the performance of ACO algorithms, the author proposes a combined and improved exploitation of the best solutions with an effective mechanism for avoiding early search stagnation. In this paper, as a first step, all ants in the ACO search the solution space and generate activity lists to provide the initial population for the SS. Then, although the SS improves all the ants' solutions, only the best solution (thus far) is used to update the pheromone trails. Finally, ACO searches the solution space again using the new pheromone trails. In other words, the SS uses the previous population constructed by ACO, which subsequently updates the pheromone trails using the best solution from the SS, and searches again. In addition, a local search strategy is employed to improve the quality of solutions generated by ACO, and also as the improvement method in the SS.

Several studies have been devoted to optimize the interdependent and controversy functions of the production scheduling and the maintenance planning. Production scheduling is a static whereas the maintenance scheduling is dynamic, therefore the static can be scheduled in advance however the dynamic scheduling requires more computational efforts. Providing a single model for both the problem is a challenging task and need combinatorial optimization solutions. [Berrichi et al \(2010\)](#) proposed a Bi-Objective ACO approach to optimize production and maintenance scheduling, in this paper, the author presents an algorithm based on ACO paradigm to solve the joint production and maintenance scheduling problem. This approach is focused to deal with the parallel machine case, to improve the quality of solutions; an algorithm based on Multi-Objective Ant Colony Optimization (MOACO) approach is developed. The goal is to simultaneously determine the best assignment of production tasks to machines as well as preventive maintenance periods of the production system, satisfying at best both objectives of production and maintenance.

[Chen, Zhang, Chung, Huang, and Liu \(2010\)](#) applied ACO in the cash flow monitoring and to control the project cost. The objective of the cash flow problem is to reduce the amount of cash being received and spent during a pre-defined time period. Without the positive cash flows, the basic obligations such as payments to suppliers and payrolls are become series issues. In project level, even a high-profit project may turn out to be a failure if cash short-fall suddenly occurs; therefore the cash flow management in the project level has attracted a considerable amount of research effort in recent years. A promising progress is to integrate cash flow management with the resource-constrained project-scheduling problem (RCPSP). The authors proposed a multimode RCPSP with discounted cash flows using ACO. The application of ACO requires setting up a construction graph and designing the pheromones and heuristic information. Based on the construction graph, the serial schedule generation scheme is applied for artificial ants to build solutions. In the process of this algorithm, each ant maintains a schedule generator and builds its solution following the rules of ACS using pheromones and heuristic information. For further studies, the comprehensive survey of [Herroelen, Reyck, and Demeulemeestre \(1998\)](#) and [Brucker, Drexl, Mohring, Neumann, and Pesch \(1999\)](#) are better choices.

The maintenance of green areas is important in most of the places like schools, parks, and recreational areas, this green area maintenance plays a heavy burden on both manpower and budget concern. [Lee, Tseng, Zheng, and Li \(2010\)](#) proposed a model to search for the minimum gardener manpower requirements and a near-optimal maintenance schedule for the green areas. The authors are implemented the proposed model using a decision support system called the Garden-Ant, in this proposal the authors grouped the ants into teams to represent gardeners and considered the route of an ant team for the schedule solution. In this paper, the proposed Garden Ant model provides an appropriate maintenance plan, including manpower estimation and an all-inclusive maintenance schedule and this proposal helps for overcoming the complicated calculations in the maintenance planning of green areas can be accomplished in an easy and efficient manner.

In geographic information systems, the spatial information takes different forms in different applications which ranging from accurate coordinates in to the qualitative abstractions. Therefore, the existing spatial information processing techniques will tend to be any one type of spatial information, and this is not extended with the heterogeneity of spatial information when the values are arises in practice, also the approximate boundaries of the spatial regions are to be constructed. In order to solve the spatial information processing, [Schockaert, Smart, and Twaroch \(2011\)](#) proposed a ACO model for heterogeneous spatial information processing, in this ACO model, genetic algorithm are used to find an optimal

ordering of variables and spatial constraints. Through hybridization with Ant Colony Optimization, this ACO algorithm is able to learn geometric constraints that are implicit in the available information. For example, the pheromone maps are generated and used to encode for the parts of the plane which likely to be contained/excluded from a certain region. Therefore, this technique with the popular technique of generating kernel density surfaces from possibly imperfect point data overcomes the existing techniques for processing imperfect information.

### 2.3. Structural and concrete engineering

The design of bridge piers is crucial for the design of prestressed concrete viaducts. The piers make up between 20% and 50% of the total cost of the viaduct depending on pier heights and foundation conditions. Rectangular hollow cross-sections as described in the present paper are most frequently used. Current designs of such reinforced concrete (RC) structures are highly conditioned by the experience of structural engineers. Design procedures usually adopt cross-section dimensions and material grades based on commonly sanctioned practice. Once the geometry and materials of the structure are specified, the reinforcement of the pier is tentatively defined according to experience. The first-order stress resultants are analyzed and second-order (buckling) stress resultants are then estimated according to simplified and conservative formulae or following a more general method that accounts for second-order deformations and includes the non-linear stiffness of the column. Tentative passive reinforcement must then satisfy the limit states prescribed by concrete codes. Should the dimensions, the material grades or the reinforcement be insufficient, the structure is redefined on a trial-and-error basis. This process leads to safe designs, but the cost of the RC pier is, consequently, highly dependent upon the experience of the structural designer. In contrast to designs based on experience, artificial intelligence has been applied to a variety of fields including the solution of constrained problems. The design of RC structures is a problem of selecting design variables as subject to structural constraints for which artificial intelligence is aptly suited. Exact methods and heuristic methods are the two main approaches to structural optimization. Exact methods are usually based on the calculation of optimal solutions following iterative techniques of linear programming of the expressions of the objective function and the structural constraints ([Twomey, Stützle, Dorigo, Manfrin, & Birattari, 2010](#); [Mocholi, Jaen, Catala, & Navarro, 2010](#); [Meneses, Gambardella, & Schirru, 2010](#)). These methods are computationally quite efficient when the number of variables is limited since they require a small number of iterations. However, they must solve the problem of linear conditioned optimization in each iteration of the analysis, which is computationally laborious when there are a large number of variables. In addition, exact methods require explicit expressions for the constraints which are not available in the present case of a non-linear buckling column. The second approach involves the heuristic methods based on artificial intelligence procedures. These methods include a wide range of artificial intelligence search algorithms, such as genetic algorithms, simulated annealing, threshold accepting, tabu search, ant colonies. These methods involve simple algorithms, but they also require a considerable computational effort, since they include a large number of iterations in which the objective function is evaluated and the structural constraints are checked.

[Martinez, González-Vidosa, Hospitaler, Yepes \(2010\)](#) proposes the ACO model for economic optimization of reinforced concrete (RC) bridge piers with hollow rectangular sections. The author describes the efficiency of their proposal in three heuristic algorithms which are variants of the ACO algorithm, the Genetic Algorithm (GA) and the Threshold Acceptance (TA) algorithm. The GA and

TA are used for comparison with the new ACO algorithms, the calibration of this ACO algorithm is recommended for a 250-member ant population and 100 stages which saved the 33% of project cost when compared with the existing design.

Biofilm processes are increasingly used for wastewater purification as they are environmental friendly and less energy intensive. Biofilms are agglomerations of microorganisms or bacteria, which due to their metabolic activity convert the contaminant components of the wastewaters into harmless products. The behaviour of a biofilm is determined by a variety of biological, chemical and physical processes internal to the film as well as interactions between the biofilm and its environment. [Rama Rao, Srinivasan, and Venkateswarlu \(2010\)](#) proposes ACO based novel optimization method for the determination of parameters involved in the kinetic and film thickness models of fixed bed anaerobic biofilm reactor. This proposal determined as a consequence of the validation of the process model with the aid of experimentally measured data. The results are evaluated with respect to the mathematical models, optimization methods, kinetic and film thickness expressions and the types of packing with the biofilm reactor.

Multi-target tracking (MTT) is a classic and an intractable problem which applied for sonar based tracking of submarines, radar based tracking of aircrafts, video based tracking of people for surveillance or security purposes. The issues of the MTT are the number of targets, the time that some target appears or dies, and the association of measurements with related targets, data association problems including measurements-to-measurements, measurements-to-tracks, and tracks-to-tracks. For MTT, various approaches are already applied, namely, the nearest neighbor (NN) method, the strongest neighbor method, the joint probabilistic data association (JPDA) method, the multiple hypothesis tracking (MHT) method. [Xu, Chen, Zhu, and Wang \(2011\)](#) applied ACO model for MTT, and the result shown are optimal than the above said methods.

#### 2.4. Digital Image Processing

Image segmentation is used to identify a spatially connected pixels or regions of a digital image. The objective of the image segmentation is to partition an input image into multiple segments so that objects and boundaries could be located and the image could become more meaningful and easier to analyze. In image processing and computer vision, the segmentation process is considered as one of the most important problems. Although many segmentation techniques is appeared in the literature, which can be classified into image-domain based, physics based and feature-space based techniques. These segmentation techniques are used extensively but each has its own advantages and limitations. Image-domain based techniques utilize both colour features and spatial relationship among colour in its homogeneity evaluation to perform segmentation. The Fuzzy C-means (FCM) algorithm has been used extensively to improve the compactness of the regions due to its clustering validity and simplicity of implementation. It is a pixels clustering process of dividing pixels into clusters so that pixels in the same cluster are as similar as possible and those in different clusters are as dissimilar as possible. This accords with segmentation application since different regions should be visually as different as possible. Recently, some feature-based segmentation techniques have employed the concept of ant colony algorithm (ACA) to carryout image segmentation. Recent researches concentrated on applying threshold with intelligent algorithms like ACO and fuzzy measures so that more adaptive and accurate decisions could be made to achieve better results. [Yu, Au, Zou, Yu, and Tian \(2010\)](#) proposes Ant Colony–Fuzzy C-means Hybrid Algorithm (AFHA) for adaptively clusters the image pixels which is viewed as three dimensional data pieces in the RGB colour space. The

Ant System (AS) algorithm is applied for intelligent initialization of cluster centroids with adaptivity. [Tan and Isa \(2011\)](#) proposes a novel histogram thresholding – fuzzy C-means hybrid (HTFCM) approach that could find different application in pattern recognition as well as in computer vision, particularly in colour image segmentation. The results shows that the execution time and pattern matches are optimal than AFHA.

Wavelet-based transformation techniques are widely used for performing image interpolation in the digital image processing. In this wavelet transform, a common assumption in the wavelet based image interpolation approach is that the input image is treated as the low frequency sub-bands of an unknown wavelet transformed high-resolution image. Then the unknown high resolution image can be reconstructed by estimating the wavelet coefficients of the high frequency sub bands, followed by applying the inverse wavelet transform. This model neglects the correlations among the sign information of the wavelet coefficients, since the Gaussian distribution is symmetrical around the zero. Inaccurate estimation of the sign of wavelet coefficients could result in poor performance in the reconstructed image. To tackle this challenge, [Tian, Ma, and Yu \(2011\)](#) propose a Three-Component Exponential Mixture (TCEM) model, by formulating the probability distribution of individual wavelet coefficient using three components: (i) a Gaussian component, (ii) a positive exponential component, and (iii) a negative exponential component. To address the parameter estimation challenge of the proposed TCEM model, the Ant Colony Optimization (ACO) technique is exploited to classify the wavelet coefficients into one of three components of the proposed TCEM model for estimating their parameters.

#### 2.5. Electrical Engineering

Economic dispatch(ED) problem is one of the mathematical optimization issues in the power system operation. With an increasing concern over the environmental pollution caused by thermal power plants, environmental/economic dispatch (EED) problem has drawn much more attention for a good dispatch scheme from it would not only result in great economical benefit, however it reduce the pollutants emission. Different techniques have been reported in the literature pertaining to EED problem, including conventional approaches such as weighted mini-max method, direct analytical solution method, linear programming and 1-constraint method, and artificial intelligence technology such as genetic algorithm, particle swarm optimization, fuzzy set theory and evolutionary programming. In principle, these approaches usually employed to deal with EED problems which can be classified into two categories, namely, Lagrange multiplier methods and a multi-objective stochastic search technique. Many researchers have performed studies in this field, a trade-off relations between cost and emission is the main objective of this problem. [Cai, Ma, Li, Li, and Peng \(2010\)](#) proposed a multi-objective chaotic ant swarm optimization for EED. This paper developed a multi-objective chaotic ant swarm optimization (MOCASO) method for solving the EED problems of thermal generators in power systems considering both economic and environmental issues. In MOCASO method, Pareto-dominance is employed to handle multi-objective problems, and fuzzifying, fitness sharing and turbulence factor perturbing techniques are also embedded. The proposed method was successfully employed to solve the EED problems in three test systems considering some constraints, such as power balance constraints and generation limits constraints. The author produced numerical simulation results which indicated that the MOCASO method is feasible and effective for solving EED problem for power systems.

[Pothiya, Ngamroo, and Kongprawechnon \(2010\)](#) proposed an Ant colony optimisation for economic dispatch problem with

non-smooth cost functions. This paper presents a novel and efficient optimisation approach based on the ACO for solving the economic dispatch (ED) problem with non-smooth cost functions. In order to improve the performance of ACO algorithm, three additional techniques, i.e. priority list, variable reduction, and zoom feature are presented. To show its efficiency and effectiveness, the proposed ACO is applied to two types of ED problems with non-smooth cost functions. first, the ED problem with valve-point loading effects consists of 13 and 40 generating units. Second, the ED problem considering the multiple fuels consists of 10 units. Additionally, the results of the proposed ACO are compared with those of the conventional heuristic approaches. The experimental results of this paper show that the proposed ACO approach is comparatively capable of obtaining higher quality solution and faster computational time.

Reactive power management is essential to transfer real energy and support power system security. Developing an accurate and feasible method for reactive power pricing is important in the electricity market. In conventional optimal power flow models the production cost of reactive power was ignored. [Ketabi, Alibabaei, and Feuillet \(2010\)](#) proposed an Application of the ant colony search algorithm to reactive power pricing in an open electricity market. In this paper, the production cost of reactive power and investment cost of capacitor banks were included into the objective function of the OPF problem. Then, using ant colony search algorithm, the optimal problem was solved. Marginal price theory was used for calculation of the cost of active and reactive power at each bus in competitive electric markets. The application of the proposed method on IEEE 14-bus system is confirms its validity and effectiveness. This ACO algorithm has the following features: (1) The points in feasible region are regard as "ants". After some iteration, the ants will centralize at the optimum points which could be one or more points. There are two choices for an ant in the each iteration: moving to other ants' point or searching in neighborhood. (2) The iteration would be guided by changing the distribution of intensity of pheromone in feasible region. 3) Sequential quadratic programming (SQP) is used as neighborhood-searching algorithm to improve the precision of convergence. The roulette wheel selection and disturbance are used to prevent the sub-optimization in ACO. The result of this paper shows that the effects of various factors on reactive power price on several case studies.

Fuel cell power plants (FCPPs) have been taken into a great deal of consideration in recent years. The continuing growth of the power demand together with environmental constraints is increasing interest to use FCPPs in power system. Since FCPPs are usually connected to distribution network, the effect of FCPPs on distribution network is more than other sections of power system. One of the most important issues in distribution networks is optimal operation management (OOM) which can be affected by FCPPs. [Niknam, Meymand, and Nayeripour \(2010\)](#) proposed a practical algorithm for optimal operation management of distribution network including fuel cell power plants. In this paper, the author proposes a new approach for optimal operation management of distribution networks including FCCPs. In the article, they consider the total electrical energy losses, the total electrical energy cost and the total emission as the objective functions which should be minimized. Whereas the optimal operation in distribution networks has a nonlinear mixed integer optimization problem, the optimal solution could be obtained through an evolutionary method. The author uses a new evolutionary algorithm based on Fuzzy Adaptive Particle Swarm Optimization (FAPSO) to solve the optimal operation problem and compared this method with Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Differential Evolution (DE), ACO and Tabu Search (TS) over two distribution test feeders.

## 2.6. Clustering

Clustering, also called set partitioning problem, is a basic and widely applied methodology various application which includes statistics; mathematical programming such as location selecting, graph theory, scheduling and assignment problems; and in computer science which includes pattern recognition, network partitioning, routing, learning theory, image processing and computer graphics. Clustering is mainly to group all objects into several mutually exclusive clusters in order to achieve the maximum or minimum of an objective function. Data clustering, which is an NP-complete problem of finding groups in heterogeneous data by minimizing some measure of dissimilarity, is one of the fundamental tools in data mining. There are many methods applied in clustering analysis, like hierarchical clustering, partition-based clustering, density-based clustering, and artificial intelligence-based clustering. K-means is used as a popular clustering method due to its simplicity and high speed in clustering large datasets. However, K-means has two shortcomings: (1) dependency on the initial state and convergence to local optima and (2) global solutions of large problems cannot found with reasonable amount of computation effort. In order to overcome local optima problem lots of studies have been done in clustering, many methods for local optimization are based on the notion of a direction of a local descent at a given point. [Maroosi and Amiri \(2010\)](#) proposed a Hybrid Global Optimization (HGOP) based on a dynamical systems approach algorithm. In this paper, the author proposed the application of hybrid global optimization algorithm based on a dynamical systems approach and compared the propose HGOP with other algorithms in clustering, such as GAK, SA, TS, and ACO, by implementing them on several simulation and real datasets.

Data mining is a field of study which lies under the knowledge engineering and knowledge discovery. Data mining is extracting useful knowledge from the large set of secondary data. The data mining techniques have been successfully applied in many different domains, such as breast-cancer detection in the biomedical sector, market basket analysis in the retail sector, prediction of future trends and credit scoring in the financial sector. [Verbeke, Martens, Mues, and Baesens \(2011\)](#) focuses on the use of data mining to predict customer churn. The Customer churn prediction models will aim to detect the customers with a high tendency in the market place. An accurate segmentation of the customer base allows a company to target the customers that are most likely to churn in a retention marketing campaign, which improves the efficient use of the limited resources for such a campaign. The author represents a fine and structured review of the churn prediction as a table. The author proposes the hybrid AntMiner+ and ALBA model for customer analysis and conclude the result is outperforms than existing methods.

## 2.7. Routing Algorithm

In the network routing, Ant-Net Routing using Ant Colony Optimization (ACO) technique provide a better result than others due to its real time computation and less control overhead. [Kwang and Weng \(2003\)](#) comparing all routing algorithms with ACO, concludes that ants are relatively small, can be piggybacked in data packets and more frequent transmission of ants may be possible in order to provide updates of routing information for solving link failures. Hence, using ACO for routing in dynamic network seems to be appropriate. Routing in ACO is achieved by transmitting ants rather than routing tables or by flooding LSPs. Even though it is noted that the size of an ant may vary in different systems/implementations,

depending on their functions and applications, in general, the size of ants is relatively small, in the order of 6 bytes.

**Laura, Matteo, and Gianluca (2008)** proposed a ACO algorithm which aims at minimizing complexity in the nodes at the expenses of the optimality of the solution, it results to be particularly suitable in environments where fast communication establishment and minimum signaling overhead are requested. However, this proposal is optimal for a less number of nodes in the cluster and also not suitable for adhoc network. A fault tolerant routing protocol (**Misra, Dhurandher, Obaidat, Verma, & Gupta 2010**) using greedy ACO routing mechanism may tend to choose single path. This routing achieves high packet delivery ratio and throughput whereas the packet loss on the link is not taken into consideration. The proposed RACO is restructured for considering the Packet loss of the link using additional traffic model and availability model for traffic free routing as well as avoiding frequent link failure nodes.

**Amilkar, Rafael, and Francisco (2010)** analyzed the performance of ACO on various case studies in the TSP using a two stage approach and concluded the performance of ACO is optimal than existing for TSP. The two-stage approach will converge quickly for lesser nodes whereas it requires more convergence time, if number of nodes increases. All the above ACO based routing algorithms identify and apply all possible 'n' no of paths, which degrade the performance of multipath routing algorithm (**Neumann and Witt 2010**). The author concluded that the number of possible routes increases, the relative performance of multi-path routing also increases till 'k' number of paths and when it exceeds the limit, the performance will be degraded. Therefore to choose only 'k' path is an important consideration for implementing multi path routing and the optimal value of 'k', may change in practice. In order to

avoid the traffic merging, and to allow only 'k' number of path, RLA algorithm along with ACO is proposed.

Wireless sensor networks (WSNs) consist of a large number of autonomous and resource constrained sensor nodes which are equipped with sensing devices, wireless communication interfaces, limited processing and energy resources. The WSNs are used for distributed and cooperative sensing of physical phenomena and events of interests. The WSN is referred as a robotic network and/or as a sensor-actor network. The WSNs can be employed in a wide spectrum of applications in both civilian and military scenarios, which includes the environmental monitoring, surveillance for safety and security, automated health care, intelligent building control, traffic control, object tracking. The routing in WSN are still an issue, **Saleem, Di Caro, and Farooq (2011)** proposes the ACO framework with the other SI technique ABC, in which, five main modules with additional sub modules are implemented. The main modules are: (i) mobile agents generation and management, (ii) routing information database (RID), (iii) agent structure, (iv) agent communications, and (v) packet forwarding. The main modules and sub-modules implements the architecture and the operations at the node router, the author concludes the hybrid ACO and ABC routing model is optimal in the WSN.

### 3. ACO implementation and performance evaluation

In this section, ACO is proposed for network routing, the ACO is optimally adapted for both wired and wireless routing as well as single path and multipath routing, so it is called as Single path and Multi path ACO (SMACO). Every node in the network can function as a source node, destination node, and/or intermediate node. Every node has a pheromone table and a routing table. The routing table can be constructed based on the state transition rule and pheromone update policy. The following random proportional rule is applied as State transition rule: for destination  $D$ , at node  $i$ , the probability of selecting a neighbor  $j$  is

$$\text{prob}(D, i, j) = \text{Fun}(TD, i, j, \eta) \quad \text{if}, \quad j \in N, \quad (2)$$

where TD is the pheromone value corresponding to neighbor  $j$  at node  $i$  and  $0 < TD < 1$  is the local heuristic value of the link  $(i,j)$  / or node  $j$ .  $0 < \eta < 1$  is the value can represent neighbor's information (i.e., neighbors queue delay, battery's remaining energy, processing power, link's signal-to-noise ratio, link's bandwidth, bit-error rate, etc.).  $\text{Fun}(TD, i, j, \eta)$  is a function in TD and  $\eta$  (this function value is high when TD and  $\eta$  are high).  $N$  is the set of all feasible neighbor nodes defined by the ant's information and the routing constraints (i.e., the guarantee of loop free). Assuming that at a given moment

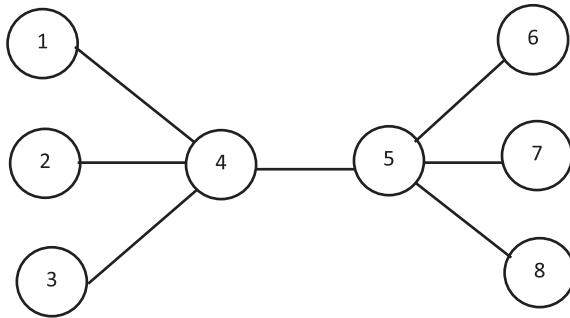


Fig. 3. Sample wired type 1 network.

**Table 1**  
Types of wired network used for simulation.

Type of network	No. of nodes	No. of source/destination node	No. of intermediate node	Normal load (source and destination node)	Medium load (source and destination node)	Heavy load (source and destination node)
Type 1	8	3	2	N1–N8, N2–N7, N3–N6, N8–N3, N7–N1	N1–N8, N2–N7, N3–N6, N8–N3, N7–N1, N1–N6, N2–N8, N3–N7, N8–N1, N7–N2	
Type 2	20	5	10	N1–N20, N2–N19, N3–N18	N1–N20, N2–N19, N3–N18, N4–N17, N5–N16, N6–N15, N7–N14, N8–N13, N9–N12, N10–N11	
Type 3	50	10	30	N1–N50, N2–N49, N3–N48, N4–N47, N5–N46	N1–N50, N2–N49, N3–N48, N4–N47, N5–N46, N6–N45, N7–N44, N8–N43, N9–N42, N10–N41	
Type 4	75	15	45	N1–N75, N2–N74, N3–N73	N1–N75, N2–N74, N3–N73, N4–N72, N5–N71, N6–N70, N7–N69, N8–N68, N9–N67, N10–N66	
Type 5	100	20	80	N1–N100, N2–N99, N3–N98, N4–N97, N5–N96	N1–N100, N2–N99, N3–N98, N4–N97, N5–N96, N6–N95, N7–N94, N8–N93, N9–N92, N10–N91	
Type 6	200	40	160	N1–N200, N2–N199, N3–N198, N4–N197, N5–N196, N6–N195, N7–N194, N8–N193, N9–N192, N10–N191		

in time  $m1$  ants have used the first bridge and  $m2$  the second one, the probability  $p1$  for an ant to choose the first bridge is:

$$Fun(TD, r, s) = \begin{cases} \frac{T(r,s) \cdot |\eta(r,s)|^\beta}{\sum T(r,s) \cdot |\eta(r,s)|^\beta} \rightarrow \text{if ... route ... found} \\ 0 \rightarrow \text{otherwise} \end{cases} \quad (3)$$

where  $T(r,s)$  is the pheromone deposited in the path between 'r' and 's',  $\eta(r,s)$  is the corresponding heuristic value which is the inverse of length of the particular path.  $\beta$  is a parameter which determines the relative importance of pheromone versus distance ( $\beta > 0$ ). The pheromone update policy is as follows:

$$T(r, s) \leftarrow (1 - \alpha) \cdot T(r, s) + \sum (1 - \alpha) \cdot T(r, s), \quad (4)$$

$$\Delta T_k(r, s) = \begin{cases} \frac{1}{L_k} \rightarrow \text{if ... route ... found} \\ 0 \rightarrow \text{otherwise} \end{cases}, \quad (5)$$

where,  $L$  is the length of tour,  $\alpha$  is the pheromone decay parameter which is lies between '0' and '1'. The pheromone values of each entry in the table can be initialized to equal values, thus providing nonbiased search for the best path. If some information about the best path is available, the pheromone values of the entry can be set to closer values to the optimum, thus, speed up the algorithm. The detailed proposed algorithm is given below:

### 3.1. The single path and multipath ACO (SMACO) algorithm

---

#### (1) //Initialization Phase

For each pair  $(r, s)$ , the value of  $T(r, s) := T_0$  End-for

For  $k := 1$  to  $m$  do

Let  $(r, k_1)$  be the starting city for an ant  $k$

$J_k(r_{k1}) := \{1, \dots, n\} - r_{k1}$

$r_k := r_{k1}$

End-for

#### (2) //This is the phase in which ants build their tours. The tour of ant $k$ : Tour <sub>$k$</sub>

For  $i := 1$  to  $n$  do

If  $i < n$

Then

For  $k := 1$  to  $m$  do

Choose the next city  $S_k$

$J_k(S_k) := J_k(r_k) - S_k$

Tour <sub>$k$</sub> ( $i$ ) :=  $(r_k, S_k)$

End-for

Else

For  $k := 1$  to  $m$  do

$S_k := r_{k1}$

Tour <sub>$k$</sub> ( $i$ ) :=  $(r_k, S_k)$

End-for

End-if

For  $k := 1$  to  $m$  do

$T(r, s) \leftarrow (1 - \alpha) \cdot T(r, s) + \sum (1 - \alpha) \cdot T(r, s)$

$r_k := s_k$  // New city for ant  $k$

End-for

End-for

#### (3) //In this phase global updating occurs and pheromone is updated

For  $k := 1$  to  $m$  do

Compute  $L_k$ / $L_k$  is the length of the tour done by ant  $k$

End-for

Compute  $L_{best}$

For each edge  $(r, s)$

$T(r, s) \leftarrow (1 - \alpha) \cdot T(r, s) + \sum (1 - \alpha) \cdot T(r, s)$

End-for

#### (4.1) //For Single Path Routing

For  $k := 1$  to  $m$  do

Sort the routing table based on pheromone values

{

Computes the Response Complexity ( $C_r$ ) of particular path where  $rtt_s$  – standard RTT (Round Trip Time),  $rtt_a$  – actual RTT

Delete the path having negative response complexity  
Choose the best path based on availability and priority

}

End-for

#### (4.2) //For Multi Path Routing

//In this compute Compound Probability

$k := 1$  to  $m$  do

Sort the routing table based on pheromone values

{

Computes the Response Complexity ( $C_r$ ) of particular path

$$C_r = \left\{ \frac{(rtt_s - rtt_a)}{rtt_s} \right\} - 6$$

where  $rtt_s$  – standard RTT (Round Trip Time),  $rtt_a$  – actual RTT

Delete the path having negative response complexity  
Compute Probability of packet communication based on compound probability rule.

Compound probability rule = [probability based on random proportional rule (equation 2) + Response Ratio Probability]/2.

$$\text{Response Ratio Probability} = C_r / \sum C_r$$

}

End-for

## 4. Result and analysis

The proposed ACO is implemented in Network Simulator 2 (NS2). The performance is tested in a variety of design and topology of network which include wired, wireless; in variety of network design which based on number of nodes; various load condition of network which is defined as normal load, medium load and heavy load; and using various transport protocol such TCP and UDP. Fig. 3 show the design of type 1 wired network, the Table 1 shows the various types of wired network used for the simulations. The simulation is implemented for 10 s. The throughput, response time and packet loss is calculated for entire 10 s and the mean value of each calculation is shown in the following tables. The average response time shown in the tables and figures are the combination of route discovery time, transmission time, propagation delay in each node, and waiting time in the intermediate queue.

Table 2 shows the comparison of packet loss in proposed and existing routing protocol. Table 3 shows the comparison of routing packet size of each routing protocol. In which the ACO has lesser packet size which reduces the routing overhead and avoids traffic due to control packet. The packet size of SMACO is slight higher than ACO due to two more two structures are introduced in the SMACO which occupy 8bits each. Therefore the implementation not requires any control overhead. The higher the routing packet size may increase the routing overhead and involves unnecessary traffic due to control packets, whereas the objectives of this thesis are to obtain reduced packet losses and response time, which is satisfied using SMACO.

**Table 2**

Performance analysis on number of packet losses.

Type of network	Normal load			Medium load			High load		
	OSPF OMP	ACO	SMACO	OSPF OMP	ACO	SMACO	OSPF OMP	ACO	SMACO
Type 1	3	3	3	4	5	4	67	34	16
Type 2	4	5	4	6	7	6	10	11	7
Type 3	6	7	5	8	10	8	14	16	10
Type 4	8	9	6	12	13	11	19	22	15
Type 5	12	13	10	17	19	15	27	31	21
Type 6	16	18	13	23	26	21	38	43	29

**Table 3**

Routing packet size (in bytes).

OSPF OMP	ACO	SMACO
44	16	32

## 5. Conclusion

ACO is implemented in always all engineering applications like continuous casting of steel, data reconciliation and parameter estimation in dynamic systems, gaming theory, In-Core Fuel Management Optimization in Nuclear Engineering, target tracking problem in signal processing, design of automatic material handling devices, Mathematical and kinetic modeling of bio-film reactor, optimization of a rail vehicle floor sandwich panel, software design, Vehicle routing design, Quadratic Assignment problem, mutation problem. The various level of experimental in the computer network using ACO as routing protocol shows (Chandra Mohan, Sandeep, & Sridharan, 2008; Chandra Mohan & Baskaran, 2010, 2011a, 2011b, 2011c, 2011d) that the ACO outperforms than the existing research methodologies. A minute redefinition, updation and or modification of the procedural steps of ACO also will raise the performance dramatically. The ACO remains open many research issues and the ACO are optimally suit many engineering domains.

## References

- Amilkar, P., Rafael, B., & Francisco, H. (2010). Analysis of the efficacy of a two-stage methodology for Ant Colony Optimization: Case of study with TSP and QAP". *Expert Systems with Applications* (Elsevier), 37, 5443–5453.
- Andziulis, A., Dzemydiene, D., & Steponavicius, R. (2011). Comparison of two heuristic approaches for solving the production scheduling problem. *Information Technology and Control*, 40(2), 118–122.
- Berrichi, A., Yalaoui, F., Amodeo, L., & Mezghiche, M. (2010). Computers Bi-Objective Ant Colony Optimization approach to optimize production and maintenance scheduling. *Operations Research*, 37, 1584–1596.
- Brucker, P., Drexel, A., Mohring, R., Neumann, K., & Pesch, E. (1999). Resource constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112, 3–41.
- Cai, J., Ma, X., Li, Q., Li, L., & Haipeng, P. (2010). A multi-objective chaotic ant swarm optimization for environmental/economic dispatch. *Electrical Power and Energy Systems*, 32, 337–344.
- Chandra Mohan, B., & Baskaran, R. (2011b) Reliable Barrier-free Services in Next Generation Networks, *Lecture Notes in Computer Science, Second International Conference on Advances in Power Electronics and Instrumentation Engineering, Nagpur, India, (PEIE 2011)*, Springer-Verlag Berlin Heidelberg, CCIS 148, pp. 79–82.
- Chandra Mohan, B., & Baskaran, R. (2011d). Energy aware and energy efficient routing protocol for adhoc network using restructured artificial bee colony system", HPACC 2011, Springer-Verlag Berlin Heidelberg, CCIS 169, pp. 480–491
- Chandra Mohan, B., & Baskaran, R. (2010). Improving network performance by optimal load balancing using ACO based redundant link avoidance algorithm. *International Journal of Computer Science Issues*, 7(3), 27–35. No. 6.
- Chandra Mohan, B., & Baskaran, R. (2011a). Reliable transmission for network centric military networks. *European Journal of Scientific Research*, 50(4), 564–574.
- Chandra Mohan, B., & Baskaran, R. (2011c). Priority and compound rule based routing using ant colony optimization. *International Journal of Hybrid Intelligent System* (Vol. 8). Netherland: IOS Press, pp. 93–97, No. 2.
- Chandra Mohan, B., & Baskaran, R. (2011e). Survey on recent research and implementation of Ant Colony Optimization in various engineering applications. *International Journal in Computational Intelligent Systems*, 4(4), 556–582.
- Chandra Mohan, B., Sandeep, R., & Sridharan, D. (2008). A data mining approach for predicting reliable path for congestion free routing using self-motivated neural network studies in computational intelligence (Vol. 149). Springer-verlag, pp. 237–246.
- Chen, W.-N., Zhang, J., Chung, H. S.- H., Huang, R.-Z., & Liu, O. (2010). "Optimizing Discounted Cash Flows in Project Scheduling—An Ant Colony Optimization Approach", *IEEE Transactions On Systems, Man, And Cybernetics—Part C: Applications And Reviews*, Vol. 40, No. 1, January 2010.
- Chen, W., Shi, Y.-J., Teng, H.-F., Lan, X.-P., & Hu, L.-C. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, 180, 1031–1039.
- Deneubourg, J. L., Aron, S., Goss, S., & Pasteels, J. M. (1990). The self-organizing exploratory pattern of the Argentine ant. *Insect Behaviour* (Elsevier), 3, 159–164.
- Dorigo, M., & Gambardella, L. M. (1997). Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1), 53–66.
- Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant System: Optimization by a colony of cooperating agents. *IEE Transactions on Systems, Man, and Cybernetics—Part B* (1), 29–41.
- Dorigo, M., & Stutzle, T. (2004). *Ant colony optimization*. Cambridge MA: MIT Press.
- Elhaddad, Y. R., & Sallabi, O. (2011). A novel approach for combining genetic and simulated annealing algorithms. *Lecture Notes in Electrical Engineering90 LNEE* (pp. 285–296).
- Goss, S., Aron, S., Deneubourg, J.-L., & Pasteels, J. M. (1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76, 579–581.
- Herroelen, W., Reyck, B., & Demeulemeester, E. (1998). Resource constrained project scheduling, a survey of recent developments. *Computational Operational Research*, 13(4), 279–302.
- Ketabi, A., Alibabaei, A., & Feuillet, R. (2010). Application of the ant colony search algorithm to reactive power pricing in an open electricity market. *Electrical Power and Energy Systems*, 32, 622–628.
- Komarudin, K., & Wong, Y. (2010). Applying ant system for solving unequal area facility layout problems. *European Journal of Operational Research*, 202, 730–746.
- Kwang, M. S., & Weng, H. S. (2003). Ant Colony Optimization for routing and load-balancing: Survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(5), 560–572.
- Laura, R., Matteo, B., & Gianluca, R. (2008). On ant routing algorithms in ad hoc networks with critical connectivity. *Ad Hoc Networks* (Elsevier), 6, 827–859.
- Lee, H.-Y., Tseng, H.-H., Zheng, M.-C., & Li, P.-Y. (2010). Decision support for the maintenance management of green areas. *Expert Systems with Applications*, 37, 4479–4487.
- Lopez-Ibanez, M., & Blum, C. (2010). Beam-ACO for the travelling salesmen problem with time windows. *Computers & Operations Research*, 37, 1570–1583.
- Maroosi, A., & Amiri, B. (2010). A new clustering algorithm based on hybrid global optimization based on a dynamical systems approach algorithm. *Expert Systems with Applications*, 37, 5645–5652.
- Martinez, F. J., González-Vidosa, F., Hospitaler, A., & Yepes, V. (2010). Heuristic optimization of RC bridge piers with rectangular hollow sections. *Computers and Structures*, 88, 375–386.
- Meneses, A. de. M., Gambardella, L. M., & Schirru, R. (2010). A new approach for heuristics-guided search in the In-core fuel management optimization. *Progress in Nuclear Energy*, 52, 339–351.
- Misra, S., Dhurandher, S. K., Obaidat, M. S., Verma, K., & Gupta, P. (2010). A low-overhead fault-tolerant routing algorithm for mobile ad hoc networks: A scheme and its simulation analysis. *Simulation Modelling Practice and Theory*, 18, 637–649.
- Mocholi, J. A., Jaen, J., Catala, A., & Navarro, E. (2010). An emotionally biased ant colony algorithm for route finding in games. *Expert Systems with Applications*, 37, 4921–4927.
- Neumann, F., & Witt, C. (2010a). Ant Colony Optimization and the minimum spanning tree problem. *Theoretical Computer Science*, 411, 2406–2413.
- Niknam, T., Meymand, H. Z., & Nayeripour, M. (2010). A practical algorithm for optimal operation management of distribution network including fuel cell power plants. *Renewable Energy*, 35, 1696–1714.

- Pasteels, J. M., Deneubourg, J.-L., & Goss, S. (1987). Self-organization mechanisms in ant societies (i): Trail recruitment to newly discovered food sources. *Experientia Supplementum*, 54, 155.
- Pothiya, S., Ngamroo, I., & Kongprawechnon, W. (2010). Ant colony optimisation for economic dispatch problem with non-smooth cost functions. *Electrical Power and Energy Systems*, 32, 478–487.
- Rama Rao, T., Srinivasan, C., & Venkateswarlu (2010). Mathematical and kinetic modeling of biofilm reactor based on Ant Colony Optimization. *Process Biochemistry*, 45, 961–972.
- Neto, R. F. T., & Filho, M. G. (2011). A software model to prototype Ant Colony Optimization algorithms. *Expert Systems with Applications*, 38 pp. 249–259.
- Saleem, M., Di Caro, G. A., & Farooq, M. (2011). Swarm Intelligence based routing protocol for wireless sensor networks: Survey and future directions. *Information Sciences*, 181, 4597–4624.
- Schockaert, S., Smart, P. D., & Twaroch, F. A. (2011). Generating approximate region boundaries from heterogeneous spatial information: An evolutionary approach. *Information Sciences*, 181, 257–283.
- Tan, K. S., & Isa, N. A. M. (2011). Color image segmentation using histogram thresholding - Fuzzy C-means hybrid approach. *Pattern Recognition*, 44, 1–15.
- Tian, J., Ma, L., & Yu, W. (2011). Ant Colony Optimization for wavelet-based image interpolation using a three-component exponential mixture model. *Expert Systems with Applications*, 38, 12514–12520.
- Twomey Stützle, T., Dorigo, M., Manfrin, M., & Birattari, M. (2010). An analysis of communication policies for homogeneous multi-colony ACO algorithms. *Information Sciences*, 180, 2390–2404.
- Verbeke, W., Martens, D., Mues, C., & Baesens, B. (2011). Building comprehensible customer churn prediction models with advanced rule induction techniques. *Expert Systems with Applications*, 38, 2354–2364.
- Xing, L.-N., Chen, Y.-W., Wang, P., Zhao, Q.-S., & Xiong, J. (2010). A knowledge-based Ant Colony Optimization for flexible job shop scheduling problems. *Applied Soft Computing*, 10, 888–896.
- Xu, B., Chen, Q., Zhu, J., & Wang, Z. (2010). Ant estimator with application to target tracking. *Signal Processing*, 90, 1496–1509.
- You, X.-M., Liu, S., & Wang, Y.-M. (2010a). Quantum dynamic mechanism-based Parallel Ant Colony Optimization algorithm. *International Journal of Computational Intelligence Systems (Suppl. 1)*, 101–113.
- Zhiding, Y., Au, O. C., Zou, R., Weiyu, Y., & Tian, J. (2010). An adaptive unsupervised approach toward pixel clustering and color image segmentation. *Pattern Recognition*, 43, 1889–1906.

# We are IntechOpen, the first native scientific publisher of Open Access books

**3,350**

Open access books available

**108,000**

International authors and editors

**1.7 M**

Downloads

**151**

Countries delivered to

**TOP 1%**

most cited scientists

**12.2%**

Contributors from top 500 universities



**WEB OF SCIENCE™**

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.

For more information visit [www.intechopen.com](http://www.intechopen.com)



# Introductory Chapter: Swarm Intelligence and Particle Swarm Optimization

---

Pakize Erdogmus

Additional information is available at the end of the chapter

<http://dx.doi.org/10.5772/intechopen.74076>

---

## 1. Introduction

In order to survive, the main objective of all creatures is foraging. Foraging behavior is cooperative in the same species. Each agent in the swarm communicates with others in such a way to find the food in the shortest time and way. This capability of all lively beings gives inspiration to the human being in order to find solutions to the optimization problems. Collective foraging behaviors of the lively beings are called swarm intelligence.

Most of the animals live as social groups in order to find foods easily and protect from the enemies to survive. Each individual lives in their habitat. Looking for food, they use their own experiences called cognitive movements as well as the experience of their leaders called social movements.

Optimization is to find the best solution to a given problem under some constraints. All disciplines use optimization for finding the best solution for their problems. Optimization is the first and foremost objective for engineers too. So especially in the future engineering applications, optimization will be an indispensable part of the product.

Optimization is everywhere. In the production of a new device, in a new artificial intelligence technique, in a big data application or in a deep learning network, optimization is the most important part of the application. To design a device with optimum sizes using minimum energy, to train a network, to minimize the error between the desired output and real output values, optimization is required.

Because of the difficulties of classical optimization algorithms, scientists have started to find an easy way to solve their problems in the last 1960s. The development of the computers made the efforts of the scientists easy, and completely new problem solution techniques are

---

studied. These techniques using heuristic information were derivative free, easy to implement, and shorten the solution time. The first product of these studies is genetic algorithm (GA) developed by Holland [1]. The evolutionary idea has been applied to the solution of the optimization problems. Instead of the evolving only one solution, a group of solutions called population has been used in the algorithm. Each solution is called individual. By this way, running such algorithms with multiple processors could be possible. After GA, simulated annealing [2] has been generally accepted as the second algorithm, inspired from the annealing process of physical materials. In high temperatures, particles move randomly in order to explore the solution space. While temperature is decreasing, particles try to create a perfect crystalline structure, only with local movements.

## 2. Particle swarm optimization

Particle swarm optimization (PSO) is accepted as the second population-based algorithm inspired from animals. Since James Kennedy (a social psychologist) and Russell C. Eberhart simulated the bird flocking and fish schooling foraging behaviors, they have used this simulation to the solution of an optimization problem and published their idea in a conference in 1995 [3] for the optimization of continuous nonlinear functions. There are two main concepts in the algorithm: velocity and coordinate for each particle. Each particle has a coordinate and an initial velocity in a solution space. As the algorithm progresses, the particles converge toward the best solution coordinates. Since PSO is quite simple to implement, it requires less memory and has no operator. Due to this simplicity, PSO is also a fast algorithm. Different versions of PSO have been developed, using some operators since the first version of PSO was published.

In the first versions of PSO, the velocity was calculated with a basic formula using current velocity, personal best and local best values in the formula, multiplying stochastic variables. The current particle updates its previous velocity, not only its previous best but also the global best. The total probability was distributed between local and global best using stochastic variables.

In the next versions, in order to control the velocity, an inertia weight was introduced by Shi and Eberhart in 1998 [4]. Inertia weight balances the local and global search ability of algorithm. Inertia weight specifies the rate of contribution of previous velocity to its current velocity. Researchers made different contributions to the inertia weight concept. Linearly, exponential or randomly decreasing or adaptive inertia weight was introduced by different researchers [5]. In the next version of PSO, a new parameter called constriction factor was introduced by Clerc and Kenedy [6, 7]. Constriction factor ( $K$ ) was introduced in the studies on stability and convergence of PSO. Clerc indicates that the use of a constriction factor insured convergence of the PSO. A comparison between inertia weight and constriction factor was published by Shi and Eberhart [8].

Nearly all engineering discipline and science problems have been solved with PSO. Some of the most studied problems solved with PSO are from Electrical Engineering, Computer Sciences, Industrial Engineering, Biomedical Engineering, Mechanical Engineering and Robotics. In Electrical Engineering, power distribution problem [9] is solved with PSO. Another most studied problem in Electrical Engineering is economic dispatch problem [10, 11]. In Computer Sciences, face localization [12], edge detection [13], image segmentation [14], image denoising

[15], image filtering [16] problems are solved with PSO. In Industrial Engineering, examination timetabling problems [17], traveling salesman problem [18], and job-shop scheduling problems [19] are solved with PSO. In Robotics, particle swarm optimization in coconut tree plucking robot is introduced [20], and path planning problem is solved with PSO [21]. In these studies, it has been proven PSO success in point of both performance and speed in most of the studies.

After the main PSO algorithm was studied and evolved with some parameters, hybrid algorithms were designed and developed by researchers. The most prominent ones are hybrid PSO with genetic algorithm (GA) [22] and particle swarm optimization with chaos [23–25] and quantum chaotic PSO [26]. In the next section, some of the recent hybrid PSO algorithms are presented.

### 3. Hybrid PSO algorithms using swarm intelligence

Hybrid algorithms are quite successful since they combine both algorithms' powerful sides. Since PSO is quite fast algorithm, nearly all newly developed algorithm combined with PSO. Some of the recent studies using swarm intelligence are crow search algorithm (CSA) [27], ant lion optimizer (ALO) [28], the whale optimization algorithm (WOA) [29], grey wolf optimizer (GWO) [30], monarch butterfly optimization (MBO) [31], moth flame optimization [32], selfish herd optimization (SHO) [33], and salp swarm optimization (SSO) [34]. Since the algorithms stated in the following paragraph are quite new, according to the Web of Science records, there is not any hybrid study combining PSO with them.

Firefly algorithm (FFA) [35], bacterial foraging optimization algorithm (BFOA) [36], ant colony optimization (ACO) [37], artificial bee colony (ABC) [38], and cuckoo search (CS) [39] are some of the algorithms using swarm intelligence improved in the last decade. There are hybrid versions of these algorithms with PSO.

#### 3.1. Firefly algorithm

FFA is improved by mimicking the flashing activity of fireflies. FFA is similar most of the swarm intelligence algorithm. Fireflies are located in a position in the solution space randomly initially. The fitness of the fireflies is calculated according to the light intensity. The next location of each firefly is calculated according to the current position, randomness and attractiveness. The hybrid algorithm combined PSO with firefly optimization [40] proposes a technique for the detection of Bundle Branch Block (BBB), one of the abnormal cardiac beat, using hybrid firefly and particle swarm optimization (FFPSO) technique in combination with Levenberg Marquardt Neural Network (LMNN) classifier.

#### 3.2. Bacterial foraging optimization algorithm

BFOA is inspired by the social foraging behavior of *Escherichia coli*. BFOA is an efficient algorithm in solving real-world optimization problems. Chemotaxis process simulates the movement of an *E. coli* cell through swimming and tumbling via flagella. *E. coli* cells like particles move in solution space and change their location with a formula-dependent previous

location, randomness and chemotactic step size. The big difference between PSO is that not only global best, but also global worst is also evaluated in the algorithm. Among *E. coli* cells, the least healthy one, eventually die. Each of the healthier bacteria splits into two bacteria, which are then placed in the same location. This keeps the swarm size constant. The hybrid algorithm using PSO and BFOA optimized PI controller, for multiobjective load frequency [41]. The authors developed a hybrid PSO with firefly, also developed a hybrid PSO with BFOA, for the detection of BBB. The same classifier is used in the study [42].

### 3.3. Ant colony optimization

ACO is inspired from the pheromone trails of ants. The first version improved by Dorigo is called ant system [43]. Although most of the ant species are blind, they can find the shortest path from their nest to food source using swarm intelligence. Ants are located in random positions in the solution space and moves with pheromone trail and randomness. In the first iterations, ants move stochastically. Eventually, pheromone increases in the path used most because ants prefer the path that contains more pheromone. This means that "Trace me." In order not to get trapped to the local convergence, pheromone evaporates related to time. ACO is generally used for the solution of combinatorial optimization problem solution such as travelling salesman problem (TSP) and some network problems. Hybrid PSO with ACO [44] solves routing problem.

### 3.4. Artificial bee colony

ABC optimization is developed by Karaboga in 2005. ABC is also a swarm intelligence algorithm based on the foraging behavior of honey bee swarms. The artificial bee colonies in the ABC algorithm consists of three groups: employed bees, onlookers and scouts. Employed bees search for food source and sharing this information to recruit onlooker bees. Onlooker bees select better food sources from those employed bees and further search around the selected food source. If a food source is not improved by some iteration, this employed bee will become a scout bee to search randomly for new food sources. In [45], a hybrid algorithm is developed combining PSO and ABC. Since PSO is fast convergent algorithm and ABC is slow convergent algorithm, hybrid algorithm uses the powerful sides of each algorithm.

### 3.5. Cuckoo search

CS is also another swarm intelligence algorithm inspired from cuckoos. CS is based on the interesting breeding behavior such as brood parasitism of certain species of cuckoos in combination with L'evy flight behavior of some birds. CS is successful for finding optimum values of multimodal functions. A hybrid algorithm is proposed finding for optimal design of multiband stop filters [46].

## 4. Conclusion

After GA, PSO is the first and foremost algorithm which is the most successful algorithm especially for the solution of the continuous optimization problems. Subsequent algorithms using swarm intelligence nearly have the same ideas. All of them are population based, and

all particles are distributed in the solution space. Particles have initial locations and velocities. They converge to the optimum solution using their swarm intelligence.

Although there have been improvements in the optimization algorithms using swarm intelligence, there is not a unique algorithm which is successful in all types of optimization problems. So the efforts trying to simulate the animal behaviors and swarm intelligence will continue. At the same time, developing hybrid algorithms will also continue until the best combination of the algorithms would be found.

## Author details

Pakize Erdogmus

Address all correspondence to: [pakizeerdogmus@duzce.edu.tr](mailto:pakizeerdogmus@duzce.edu.tr)

Engineering Faculty, Computer Engineering Department, Duzce University, Duzce, Turkey

## References

- [1] Holland JH. Adaptation in Natural and Artificial Systems. Second edition (First edition, 1975) ed. Cambridge, MA: MIT Press; 1975/1992
- [2] Kirkpatrick S, Gelatt C, Vecchi M. Optimization by simulated annealing. *Science*. 1983; **220**(4598):671-680 Retrieved from <http://www.jstor.org/stable/1690046>
- [3] Kennedy J, Eberhart R. Particle Swarm Optimization. 1995. pp. 1942-1948
- [4] Shi Y, Eberhart R. A Modified Particle Swarm Optimizer. In: IEEE International Conference on Evolutionary Computation Proceedings. 1998. pp. 69-73
- [5] C. Paper, I. Technology, and T. Kharagpur. Inertia weight strategies in particle swarm inertia weight strategies in particle swarm. no. May 2014, 2011
- [6] Clerc M. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In: Proceedings of the 1999 ICEC. Washington, DC. 1999. pp 1951-1957
- [7] Clerc M, Kennedy J. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. In: IEEE Transactions on Evolutionary Computation. Feb 2002; **6**(1):58-73. DOI: [10.1109/4235.985692](https://doi.org/10.1109/4235.985692)
- [8] Eberhart RC, Shi Y. Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512). La Jolla, CA. 2000; **1**:84-88. DOI: [10.1109/CEC.2000.870279](https://doi.org/10.1109/CEC.2000.870279)
- [9] Gao S, Wang H, Wang C, Gu S, Xu H, Ma H. Reactive power optimization of low voltage distribution network based on improved particle swarm optimization. In: Proceedings of the 2017 20th International Conference on Electrical Machines and Systems (ICEMS). Sydney, NSW. 2017. pp. 1-5

- [10] Abbas G, Gu J, Farooq U, Asad MU, El-Hawary M. Solution of an economic dispatch problem through particle swarm optimization: A detailed survey - part I. In: IEEE Access. 2017;5:15105-15141
- [11] Abbas G, Gu J, Farooq U, Raza A, Asad MU, El-Hawary ME. Solution of an economic dispatch problem through particle swarm optimization: A detailed survey – Part II. In: IEEE Access. 2017;5:24426-24445
- [12] Jois S, Ramesh R, Kulkarni AC. Face localization using skin colour and maximal entropy based particle swarm optimization for facial recognition. In: Proceedings of the 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON). Mathura, India. 2017. pp. 156-161
- [13] Chaudhary R, Patel A, Kumar S, Tomar S. Edge detection using particle swarm optimization technique. In: Proceedings of the 2017 International Conference on Computing, Communication and Automation (ICCCA). Greater Noida, India. 2017. pp. 363-367
- [14] Mozaffari MH, Lee WS. Convergent heterogeneous particle swarm optimisation algorithm for multilevel image thresholding segmentation. In: IET Image Processing. 2017; 11(8):605-619
- [15] Karami A, Tafakori L. Image denoising using generalised Cauchy filter. In: IET Image Processing. 2017;11(9):767-776
- [16] Zhou Xc. Color Image Filter Based on Predator-Prey Particle Swarm Optimization. 2009 International Conference on Artificial Intelligence and Computational Intelligence, Shanghai. 2009. pp. 480-484
- [17] Marie-Sainte SL. A new hybrid particle swarm optimization algorithm for real-world university examination timetabling problem. In: Proceedings of the 2017 Computing Conference. London, United Kingdom. 2017. pp. 157-163
- [18] Chang JC. Modified particle swarm optimization for solving traveling salesman problem based on a Hadoop MapReduce framework. In: Proceedings of the 2016 International Conference on Applied System Innovation (ICASI). Okinawa. 2016. pp. 1-4
- [19] Wenbin G, Yuxin L, Yi W. Energy-efficient job shop scheduling problem using an improved particle swarm algorithm. In: Proceedings of the 2017 IEEE 3rd Information Technology and Mechatronics Engineering Conference (ITOEC). Chongqing. 2017. pp. 830-834
- [20] Junaedy A, Sulistijono IA, Hanafi N. Particle swarm optimization for coconut detection in a coconut tree plucking robot. In: Proceedings of the 2017 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC). Surabaya, Indonesia. 2017. pp. 182-187
- [21] Roberge V, Tarbouchi M, Labonte G. Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. In: IEEE Transactions on Industrial Informatics. Feb 2013;9(1):132-141

- [22] Juang C. A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design. 2004;**34**(2):997-1006
- [23] Liu B, Wang L, Jin Y, Tang F, Huang D. Improved particle swarm optimization combined with chaos. *Chaos, Solitons & Fractals*. 2005;**25**:1261-1271
- [24] Alatas B, Akin E, Ozer AB. Chaos embedded particle swarm optimization algorithms. *Chaos, Solitons & Fractals*. 2009;**40**(4):1715-1734
- [25] Rong H. An adaptive chaos embedded particle swarm optimization algorithm. In: Proceedings of the 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering. Changchun. 2010. pp. 314-317
- [26] Emrah O, Sinan M, Turhan M. Chaotic quantum behaved particle swarm optimization algorithm for solving nonlinear system of equations. *Computers and Mathematics with Applications*. 2014
- [27] Askarzadeh A. A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. *Computers & Structures*. 2016;**169**:1-12, ISSN 0045-7949. <https://doi.org/10.1016/j.compstruc.2016.03.001>
- [28] Mirjalili S. The ant lion optimizer. *Advances in Engineering Software*. 2015;**83**:80-98, ISSN 0965-9978. <https://doi.org/10.1016/j.advengsoft.2015.01.010>
- [29] Mirjalili S, Lewis A. The whale optimization algorithm. *Advances in Engineering Software*. 2016;**95**:51-67. ISSN 0965-9978. <https://doi.org/10.1016/j.advengsoft.2016.01.008>
- [30] Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. *Advances in Engineering Software*. 2014;**69**:46-61. ISSN 0965-9978. <https://doi.org/10.1016/j.advengsoft.2013.12.007>
- [31] Wang GG, Deb S, Cui Z. *Neural Computing & Applications*. 2015. <https://doi.org/10.1007/s00521-015-1923-7>
- [32] Mirjalili S. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*. 2015;**89**:228-249. ISSN 0950-7051. <https://doi.org/10.1016/j.knosys.2015.07.006>
- [33] Fausto F, Cuevas E, Valdivia A, González A. A global optimization algorithm inspired in the behavior of selfish herds. *Biosystems*. 2017;**160**:39-55. ISSN 0303-2647. <https://doi.org/10.1016/j.biosystems.2017.07.010>
- [34] Mirjalili S, Amir H, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM. Salp swarm algorithm: A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*. 2017;**114**:163-191. ISSN 0965-9978. <https://doi.org/10.1016/j.advengsoft.2017.07.002>
- [35] Yang XS. Firefly Algorithms for Multimodal Optimization. In: Watanabe O, Zeugmann T, editors. *Stochastic Algorithms: Foundations and Applications*. Lecture Notes in Computer Science. Vol 5792. Springer, Berlin, Heidelberg: SAGA; 2009

- [36] Passino KM. Biomimicry of bacterial foraging for distributed optimization and control. In: IEEE Control Systems. Jun 2002;22(3):52-67. DOI: 10.1109/MCS.2002.1004010
- [37] Dorigo M, Maniezzo V, Colomi A. Ant system: Optimization by a colony of cooperating agents. In: IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics). Feb 1996;26(1):29-41. DOI: 10.1109/3477.484436
- [38] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. J. Global Optimiz. 2007;39:459-471
- [39] Yang X-S, Deb S. Cuckoo Search via Levy Flights. In: World Congress on Nature & Biologically Inspired Computing (NaBIC 2009). 2009. pp. 210-214
- [40] Padmavathi K, Sri Rama Krishna K. Hybrid firefly and Particle Swarm Optimization algorithm for the detection of Bundle Branch Block. International Journal of the Cardiovascular Academy. 2016;2(1):44-48. ISSN 2405-8181. <https://doi.org/10.1016/j.ijcac.2015.12.001>
- [41] Dhillon SS, Lather JS, Marwaha S. Multi objective load frequency control using hybrid bacterial foraging and particle swarm optimized PI controller. International Journal of Electrical Power & Energy Systems. 2016;79:196-209. ISSN 0142-0615, <https://doi.org/10.1016/j.ijepes.2016.01.012>
- [42] Kora P, Kalva SR. Hybrid bacterial foraging and particle swarm optimization for detecting bundle branch block. SpringerPlus. 2015;4(1):481
- [43] Vitorino LN, Ribeiro SF, Bastos-Filho CJA. A mechanism based on artificial bee Colony to generate diversity in particle swarm optimization. Neurocomputing. 2015;148:39-45. ISSN 0925-2312. <https://doi.org/10.1016/j.neucom.2013.03.076>
- [44] Cheng C-Y, Chen Y-Y, Chen T-L, Yoo JJ-W. Using a hybrid approach based on the particle swarm optimization and ant colony optimization to solve a joint order batching and picker routing problem. International Journal of Production Economics. (Part C). 2015;170:805-814. ISSN 0925-5273. <https://doi.org/10.1016/j.ijpe.2015.03.021>
- [45] Li Z, Wang W, Yan Y, Zheng L. PS-ABC: A hybrid algorithm based on particle swarm and artificial bee colony for high-dimensional optimization problems. Expert Systems with Applications. 2015;42(22):8881-8895. ISSN 0957-4174. <https://doi.org/10.1016/j.eswa.2015.07.043>
- [46] Dash J, Dam B, Swain R. Optimal design of linear phase multi-band stop filters using improved cuckoo search particle swarm optimization. Applied Soft Computing. 2017;52:435-445. ISSN 1568-4946. <https://doi.org/10.1016/j.asoc.2016.10.024>

## *Tabu Search: An introduction*



# A user's guide to tabu search\*

Fred Glover

*Graduate School of Business, University of Colorado, Boulder, CO, USA*

Eric Taillard and Dominique de Werra

*Department of Mathematics, Swiss Federal Institute of Technology,  
Lausanne, Switzerland*

## Abstract

We describe the main features of tabu search, emphasizing a perspective for guiding a user to understand basic implementation principles for solving combinatorial or nonlinear problems. We also identify recent developments and extensions that have contributed to increasing the efficiency of the method. One of the useful aspects of tabu search is the ability to adapt a rudimentary prototype implementation to encompass additional model elements, such as new types of constraints and objective functions. Similarly, the method itself can be evolved to varying levels of sophistication. We provide several examples of discrete optimization problems to illustrate the strategic concerns of tabu search, and to show how they may be exploited in various contexts. Our presentation is motivated by the emergence of an extensive literature of computational results, which demonstrates that a well-tuned implementation makes it possible to obtain solutions of high quality for difficult problems, yielding outcomes in some settings that have not been matched by other known techniques.

**Keywords:** Tabu search, heuristics, combinatorial optimization, artificial intelligence.

## 1. Introduction

The abundance of difficult optimization problems encountered in practical settings (e.g. telecommunications, logistics, financial planning, transportation and production) has motivated a proliferation of optimization techniques.

Researchers have adapted ideas from many areas in the quest to develop more efficient procedures. Network flow methods, for example, share a heritage with models exploiting ideas from electricity and hydraulics. Simulated annealing is based on a physical process in metallurgy, while so called genetic methods seek to imitate the biological phenomenon of evolutionary reproduction.

\*The research of this paper has been supported in part by the joint Air Force Office of Scientific Research and Office of Naval Research Contract No. F49620-90-C-0033, at the University of Colorado and by the Swiss National Research Council Grant No. 20-27926.89.

In this vein, the Tabu Search (TS) method elaborated in this paper may be regarded as a technique based on selected concepts from artificial intelligence. TS is a general heuristic procedure for guiding search to obtain good solutions in complex solution spaces. Its rules are sufficiently broad that it is often used to direct the operations of other heuristic procedures.

One of the main components of TS is its use of *flexible (adaptive)* memory, which plays an essential role in the search process. Discoveries of more refined ways to exploit this memory, and of more effective ways to apply it to special problem settings, provide one of the key research thrusts of the discipline and account for its growing success in treating hard problems.

The organization of this paper is as follows. First, we give a general presentation of TS in the next section. Then we describe some of the features that increase the efficiency of the technique in section 3. Finally, concluding observations are provided in section 4. The goal of our development is to disclose the applicability of TS as a general tool which may be adapted to a wide range of problems. As demonstrated in the literature, performance levels are achieved for the problem domains studied that frequently surpass those of the procedures previously established to be best.

We stress, however, that this paper is not intended as an exhaustive characterization of the elements of TS, but as a first introduction to the nonspecialist and as a foundation to suggest useful directions for advanced study. Our goal is to provide a basic understanding of the procedure that may permit its principles to be applied more easily to the solution of hard problems, and to give a glimpse of additional developments that lie ahead.

## 2. What is tabu search?

With roots going back to the late 1960's and early 1970's, TS was proposed in its present form a few years ago by Glover (see, e.g. [13]), and has now become an established optimization approach that is rapidly spreading to many new fields. The basic ideas of TS have also been sketched by Hansen [20]. Together with simulated annealing and genetic algorithms, TS has been singled out by the Committee on the Next Decade of Operations Research [3] as "extremely promising" for the future treatment of practical applications.

We begin by sketching the elements of TS in rough terms. The method can be viewed as an iterative technique which explores a set of problem solutions, denoted by  $\mathcal{X}$ , by repeatedly making moves from one solution  $s$  to another solution  $s'$  located in the neighborhood  $N(s)$  of  $s$ .

These moves are performed with the aim of efficiently reaching a solution that qualifies as "good" (optimal or near-optimal) by the evaluation of some objective function  $f(s)$  to be minimized.

As an initial point of departure, we may contrast TS with a simple descent method, which may be formulated as follows:

- (a) Choose an initial solution  $s$  in  $\mathcal{X}$   
 stop: = false

**Repeat:**

- (b) Generate a sample  $V^*$  of solutions in  $N(s)$ .

Find a best  $s'$  in  $V^*$  (i.e. such that

$f(s') \leq f(\bar{s})$  for any  $\bar{s}$  in  $V^*$ ).

**if**  $f(s') \geq f(s)$  **then** stop: = true

**else**  $s := s'$

(2.1)

Until stop

By narrowing the choice of  $s'$  to a best element of  $V^*$  we are restricting the class of descent methods to a proper subset of the larger collection sometimes called "improving methods"— a restriction that has important consequences for the goal of restructuring this approach by tabu search.

First note that the most straightforward implementation of the preceding approach is to take  $V^* = N(s)$ , i.e. to scan the entire neighborhood of  $s$ . In some circumstances, however, such a process is impractical because it is too time-consuming. Therefore, in such cases we stipulate that a set  $V^* \subset N(s)$  with  $|V^*| \ll |N(s)|$  be scanned. In the extreme, one may take  $|V^*| = 1$ , thereby eliminating a direct comparison among elements, and dissolving the distinction between a best element and an arbitrary member of  $V^*$ . This latter form of sampling in fact underlies the policy adopted by simulated annealing (which also may be viewed as a way of restricting a descent method), employing rejection criteria that may result in generating a number of sets with  $|V^*| = 1$ .

Stepping beyond the case where  $|V^*| = 1$ , the preceding descent technique can be given an added dimension of refinement by considering its implicit reliance on solving a local optimization problem (either by itemization or more intricate means), to find an  $s'$  that yields  $\min\{f(s') \mid s' \in V^* \subseteq N(s)\}$ , possibly in an approximate way. Building on this viewpoint, a key concept of tabu search is to constrain the definition of this local optimization problem in a strategic way – making systematic use of memory to exploit knowledge beyond that contained in the function  $f(s)$  and the neighborhood  $N(s)$ . This also leads to a general orientation whereby the set  $V^*$  is to be generated strategically rather than randomly, with the goal of allowing the minimum of  $f(s')$  over  $V^*$  to be more nearly representative of the minimum over  $N(s)$ , subject to the special restructuring of  $N(s)$  derived by reliance on memory. In this sense, we may call the method a metaheuristic, since at each step it uses a heuristic to move from one solution to the next, guiding the search in  $\mathcal{X}$ .

Why should such a search be guided? If the descent approach is applied as described in (2.1) it will (at best) lead to a local minimum of  $f$  where it will be trapped. This is the first type of difficulty that can arise with standard heuristic descent techniques.

To circumvent such an outcome, a guidance procedure must be able to accept a move from  $s$  to  $s'$  (in  $V^*$ ) even if  $f(s') > f(s)$ . But when a solution  $s'$  worse than  $s$  may be accepted, cycling may occur, causing the process again to be trapped by returning repeatedly to the same solution (perhaps after some interval of intermediate solutions). Simulated annealing attempts to provide guidance by accepting a disimproving  $s'$  (if it happens to be the one currently sampled) with a certain probability that depends upon  $f(s)$ ,  $f(s')$  and a parameter identified with temperature. The tabu search approach, by contrast, seeks to counter the danger of entrapment by incorporating a memory structure that forbids or penalizes certain moves that would return to a recently visited solution. The variant called *probabilistic tabu search* creates probabilities for accepting moves as a monotonic function of evaluations created by a penalty of the method.

The notion of using memory to forbid certain moves (i.e. to render them tabu) can be formalized in general by saying that the solution neighborhood depends on the time stream, hence on the iteration number  $k$ . That is, instead of  $N(s)$  we may refer to a neighborhood denoted  $N(s, k)$ . (More precise notation would indicate that  $N(s, k)$  depends on solutions and neighborhoods encountered on iterations prior to  $k$ .) For instance, suppose memory is employed that recalls solution transitions over some time horizon, and creates  $N(s, k)$  by deleting from  $N(s)$  each solution that was an immediate predecessor of  $s$  in one of these transitions. We may express the form of the procedure that uses the modified (tabu) neighborhoods as follows.

```

(a) Choose an initial solution  $s$  in  $X$ 
     $s^* := s$  and  $k := 1$ 
(b) While the stopping condition is not met do
     $k := k + 1$ 
    Generate  $V^* \subseteq N(s, k)$ 
    Choose the best  $s'$  in  $V^*$ 
     $s := s'$ 
    if  $f(s') < f(s^*)$ , then  $s^* := s'$ 
end while

```

(2.2)

There may be several possible stopping conditions, but simplest is some logical combination of the following:

- An optimal solution is found.
- $N(s, k + 1) = \emptyset$ .
- $k$  is greater than the maximum number of iterations allowed.
- The number of iterations performed since  $s^*$  last changed is greater than a specified maximum number of iterations.

It is to be emphasized again that a nontrivial, strategically generated sample of the solution neighborhood is examined at each step, and a best element from the

sample is selected (subject to avoiding moves that are classified tabu). The goal is therefore to make improving moves to the fullest extent allowed by the structure of  $N(s, k)$ , balancing trade-offs between solution quality and computational effort in examining larger samples.

Characterized in this way, TS may be viewed as a *variable neighborhood method*: each step redefines the neighborhood from which the next solution will be drawn, based on the conditions that classify certain moves as tabu.

A crucial aspect of the procedure involves the choice of an appropriate definition of  $N(s, k)$ . Due to the exploitation of memory,  $N(s, k)$  depends upon the trajectory followed in moving from one solution to the next. As a starting point, consider a form of memory embodied in a tabu list  $T$  that records the  $|T|$  solutions most recently visited, yielding  $N(s, k) = N(s) - T$ .

Such a *recency based* memory approach will prevent cycles of length less than or equal  $|T|$  from occurring in the trajectory. This formulation normally has only a theoretical interest, since keeping track of the  $|T|$  most recent solutions may be extremely space consuming. (It also represents a relatively limited form of *recency based* memory.)

As a basis for identifying a more practical and effective type of memory structure, we may amend our definition of  $T$  by defining the neighborhood  $N(s)$  in terms of moves for transforming the solution  $s$  into new solutions. Specifically, for each solution  $s$  consider a set  $M(s)$  of legal moves  $m$  which can be applied to  $s$  in order to obtain a new solution  $s' = s \oplus m$ . Then we have  $N(s) = \{s' | \exists m \in M(s) \text{ with } s' = s \oplus m\}$ . Generally in TS it is useful to employ a set of moves  $M(s)$  that is reversible, i.e.  $\forall m \in M(s), \exists m^{-1} \in M(M(s))$  such that  $(s \oplus m) \oplus m^{-1} = s$ . With this stipulation,  $T$  may consist of the last  $|T|$  "reverse moves" associated with the moves actually performed.

Instead of a single tabu list  $T$  it may be more convenient to use several lists  $T_i$ . Then a tabu status is assigned to some constituents  $T_i$  of  $m$  (or of  $s$ ) to indicate that these are currently forbidden for composing a move. In general, the tabu status of a move is a function of the tabu status of *attributes* that change in going from the current solution to the next (such as variables that change values, or elements that exchange positions, or items that are transferred from one set to another).

This leads to a collection of *tabu conditions* of the form

$$t_i(m, s) \in T_i \quad (i = 1, \dots, p). \quad (2.3)$$

A move  $m$  applied to the current solutions will then be a tabu move (i.e. implicitly a member of the list  $T$ ) if all conditions (2.3) are satisfied.

We now briefly illustrate some of the preceding concepts with a simple example [42] where TS has produced very good solutions; the quadratic assignment problem (QAP). Subsequently, we will use this example to motivate the discussion of additional elements fundamental to tabu search.

## QAP FORMULATION

$N$  items (workshops of a factory) have to be assigned to  $N$  different locations. The distance  $a_{ij}$  between locations  $i$  and  $j$  is given as well as the value  $b_{pq}$  of a flow (circulation of parts) between the items  $p$  and  $q$ .

An assignment of the items to the locations is sought that minimizes the sum of the products of distances and associated flow values. Mathematically, the problem consists in finding a permutation  $\phi$  of  $\{1, \dots, N\}$  to minimize

$$f(\phi) = \sum \sum a_{ij} b_{\phi(i)\phi(j)}. \quad (2.4)$$

For a complete formulation and some applications the reader is referred to Burkard [1]. Applying our previous notation, in this problem,  $X$  consists of the  $N!$  permutations of  $\{1, \dots, N\}$ ; a move  $m$  in  $M(\phi)$  may consist of the exchange of two items  $p$  and  $q$ . The permutation  $\pi$  obtained from  $\phi$  by move  $m$  then is given by

$$\left. \begin{array}{l} \pi = \phi \oplus m \\ \pi(i) = \phi(i) \\ \pi(p) = \phi(q) \\ \pi(q) = \phi(p) \end{array} \right\} \quad \text{for all } i \neq p, q.$$

The difference  $f(\phi \oplus m) - f(\phi)$  can be computed easily, permitting exploration of the entire neighborhood  $N(\phi)$  in time  $O(N^2)$  (see Finke et al. [7]).

One possibility for creating a recency based memory is to identify attributes of a move by the unordered pair of items  $(p, q)$ . The attribute pair that must be forbidden in order to prevent the reverse move likewise is  $(p, q)$  (disallowing these two items to be exchanged again, noting that an immediate exchange of this type would revisit the solution just left). Making this attribute pair the basis of a tabu restriction permits each of  $p$  and  $q$  independently to be exchanged at once with other items. Unfortunately, however, this type of restriction does not satisfy the property of forbidding a return to solutions already visited, since the sequence of moves  $(q, r)$   $(p, s)$   $(q, s)$   $(p, r)$   $(p, q)$   $(r, s)$  does not change anything.

Another way of identifying attributes of an exchange move is to introduce additional information, referring not only to items exchanged but to positions occupied by these items at the time of exchange. To do this without the memory burden of introducing 4-element vectors, we conceive a move to be composed of two half-moves  $(p, \phi(q))$ , the first placing item  $p$  in location  $\phi(q)$  and the second placing item  $q$  in location  $\phi(p)$ . The reverse move is prevented by disallowing the two attribute pairs  $(p, \phi(p))$  and  $(q, \phi(q))$  from occurring simultaneously, and it is these two pairs that are stored in the tabu list of our TS implementation for QAP.

More precisely, we define a move  $m$  (exchange of items  $p$  and  $q$ ) to be tabu if it sends both  $p$  and  $q$  to locations they occupied within the last  $t$  iterations. This

tabu condition admits some moves prevented by the previously illustrated condition, and also forbids some moves permitted by the previous condition, hence is neither uniformly more restrictive or less restrictive.

The current restriction has the appeal of being based not only on attributes of solutions (corresponding to items located in specified positions). In addition, we allow the size  $t$  of the primary tabu list, consisting of (item, location) pairs, to be changed during the solution process.

We now return to a discussion of general TS features before completing the QAP illustration.

## 2.1. ADDITIONAL TABU SEARCH ELEMENTS: ASPIRATION CRITERIA

It is not difficult to realize that tabu conditions based on selected attributes of moves and solutions may be too drastic in the sense that they may also forbid moves leading to unvisited solutions, and in particular to unvisited solutions that may be attractive. It is therefore necessary to overrule the tabu status of moves in certain situations. This is performed by means of *aspiration level conditions*.

As an example, consider a tabu move  $m$ ; if this move gives a new solution which is better than the best found so far, then one would be tempted to drop the tabu status of  $m$  and accept the move. This can be done by saying that  $m$  (applied to  $s$ ) may be accepted if it has a level of aspiration  $a(s, m)$  which is better than a threshold value  $A(s, m)$ . More generally, we may conceive of  $A(s, m)$  as defining a set of preferred values (for elements) for a function (or mapping)  $a(s, m)$ , and aspiration may be represented in the form

$$a_i(s, m) \in A_i(s, m) \quad (i = 1, \dots, I). \quad (2.5)$$

Then the tabu status of a move  $m$  will be rendered inoperative if at least one (or a specified number) of the conditions (2.5) is satisfied. If many aspiration conditions are simultaneously satisfied by more than one move, rules of choice must be defined.

In the QAP illustration, we use the simple type of aspiration criterion that accepts a tabu move if it produces a new best solution. Notationally, this corresponds to defining  $a(s, m) = f(s \oplus m)$  and letting  $A(s, m)$  be the interval of values smaller than  $f(s^*)$ , where  $s^*$  is the best solution found so far. In addition, for types of problems where the entries  $(b_{pq})$  of the flow matrix span a large sample of numerical values, we use a second type of aspiration condition which may be denoted by  $a'(s, m) \in A'(s, m)$ . Under this condition, a tabu move  $m$  that sends item  $p$  to location  $j$  and item  $q$  to location  $i$  is accepted (no matter what the value of  $f(s \oplus m)$  may be) if  $p$  was not at  $j$  and  $q$  was not at  $i$  within the last  $u$  iterations. In such a case,  $A'(s, m)$  contains every move (or "pair of half moves") that has not been performed during the preceding  $u$  iterations, and  $a'(s, m) = m$ . Obviously, the value of  $u$  has to be set at a value larger than the size of the neighborhood; for example, it turns out that a value of ten times the size of the neighborhood is convenient for most QAP instances.

## 2.2. INTENSIFICATION AND DIVERSIFICATION

Besides the above described components, TS requires a few additional ingredients in order to behave as an intelligent search technique. The use of a memory up to now has been limited to a short term horizon: TS remembers the most recently performed steps in order to avoid coming back to a solution visited earlier, or to one of a class of solutions visited earlier, where the class is implicitly determined by attributes used to define tabu status.

Memory is also used in TS in a kind of learning process: having visited several solutions, it is deemed worthwhile to observe whether the good solutions visited so far have some common properties (such as the presence or absence of certain elements). This generates an *intensification* scheme for the search. At some stage the neighborhood  $N(s, k)$  is restricted (or the criteria for evaluating moves are altered) to favor solutions with properties that occurred often in good solutions previously visited, or more generally in subsets of good solutions differentiated by clustering criteria. This form of bias can be used either to "structure" the current solution to satisfy such properties (as by a partial restart procedure), and then to discourage the properties from being violated, or can simply be used to encourage such properties to become incorporated as the method progresses. Some of the relevant considerations are discussed in the context of *consistent* and *strongly determined* variables in [12].

These ideas are lately finding favor in other procedures, and may provide a bridge for integrating components of tabu search with components of other methodologies. For example, a first level strategy to reinforce the incorporation of consistent elements from high quality solutions (without undertaking differentiation by clustered subsets) has proved beneficial in genetic algorithms. In the setting of the traveling salesman problem, where ingenuity makes it possible to identify "crossover" operations that preserve shared structures [45], these ideas have produced very good results. In particular, Muhlenbein [33] has noted the relevance of genetic algorithms of this type for classes of traveling salesman problems that satisfy a *building block property*, in which optimal (or exceedingly good) tours can be pieced together from segments of logically optimal tours. In this case, by augmenting a genetic algorithm approach with local optimization, and applying a "maximal edge-preserving crossover", impressive outcomes are achieved for a number of classical test problems. Similar uses of such ideas by Ulder et al. [43] and Kohlen and Pesch [27] have also produced very attractive results for these problems.

The effectiveness of such a special instance of the intensification strategy in the TSP context suggests that broader forms of the strategy deserve to be examined more closely in other settings, where the building block phenomenon may not apply. Approaches to reinforce elements of elite solutions based on clustering and statistical discrimination (e.g. incorporating frequency based memory over attribute groupings), illustrate natural foundations for such a strategy. From a more specialized perspective, approaches such as the use of *scatter search* and *structured*

combinations [17] provide an opportunity to establish links with genetic algorithms, while permitting the space of solutions to be exploited in ways unavailable to genetic crossover, and may prove useful in this context.

Intensification by itself is insufficient to yield the best outcomes for general classes of optimization problems. The complementary notion of *diversification* must be invoked to allow the most effective search over the set  $X$ . Strategic pursuit of solutions with varying characteristics provides an essential counterbalance to the intensification component of tabu search.

Thus, over the longer term, tabu search induces an alternation between sequences of steps that intensify the search in a promising region and steps that diversify the search across contrasting regions. One way to do this is by designing  $N(s, k)$  to incorporate solutions that are separated by a specified degree from solutions visited before. An evident form of diversification results by modifying the objective function to decrease  $f$  for solutions "far from  $s$ " and to increase  $f$  for solutions "close to  $s$ ". When  $s$  is replaced by a set  $S$  of previously generated solutions, the meanings of "far from" and "close to" become more subtle, and typically involve references to frequency based memory as well as recency based memory. Several specific ways of realizing diversification are discussed with the applications that follow.

The preceding elements summarize the basic ingredients needed to design a TS procedure. To derive a highly efficient metaheuristic, special issues of design and implementation must be addressed. We discuss some of these issues in the next section.

### 3. Refinements of TS

In considering refinements to obtain good solutions more efficiently than by an elementary TS procedure, we focus on easy-to-implement methods that have proved their merit, in the spirit of favoring techniques that are both easy to program and capable of achieving effective outcomes. Our illustrated adaptations to the QAP (introduced in section 2) provides a useful basis for exemplifying some of these features. The method succeeds in obtaining some of the best outcomes in the QAP literature by means of a Pascal program of approximately 2 pages!

We have already mentioned two diversification techniques that also turn out to be valuable in the QAP application. First is a simple strategy of varying the tabu list size. Second is an approach of variably penalizing the objective function to destroy the structure of the local minima one wants to escape, subsequently called the shifting penalty approach.

In general, an effective implementation of TS may be achieved by focusing on three levels:

- (1) tactical,
- (2) technical,
- (3) computational.

The tactical level seeks to embed greater intelligence in the search process. The technical and computational levels are chiefly concerned with limiting the possibility of cycling and with speeding-up the iterative computation of the search.

Ways to introduce improvements in each of the areas will be reviewed in the next sub-sections.

### 3.1. TACTICAL IMPROVEMENTS

The first question to ask in designing an iterative search procedure is related to the quality of the chosen neighborhood: "Are the defined moves good?" Frequently, the answer to this question must be based on intuitive good sense, and on practical experimentation with alternatives. However, the tools of statistical analysis and graphical simulation can be highly useful in this quest.

Simple statistics may disclose significant information about the nature of the moves performed by the search. For example, if the amplitude of objective function changes produced by the moves is very small, then it is reasonable to think that the search will have some trouble discovering good trajectories for escaping from local minima (since many trajectories look the same). On the other hand, if this amplitude is quite high, the search may have some difficulty finding local minima whose quality is close to that of a global minimum.

When both problem type and move type are susceptible to graphical visualization, exhibiting the solutions generated by the search can help to establish the appropriateness (or inappropriateness) of the neighborhood used to define admissible moves. This is relevant even where relatively simple moves are used as a foundation of a search procedure. While simple moves can be valuable for developing a method without a heavy investment of human time and resources (particularly for problems that have not been extensively studied, and whose subtleties are unknown), they are also susceptible to weaknesses that make it important to identify special "adjunct moves" that periodically can be invoked to counter these weaknesses. Such adjunct moves then fulfil the role of a diversification strategy, altering the terrain visited in a way that is unlikely to occur by applying the simple moves alone.

We give an example for the Euclidean traveling salesman problem, where graphical simulation led to identifying a useful diversification move as an adjunct to a rudimentary move structure (see Fiechter [6]). (Subsequently we identify additional ways to generate more powerful alternatives.) The problem in this case is defined to a collection of randomly placed cities on a unit square, where the goal is to obtain a shortest tour, that is, a closed path that passes exactly once through every city before returning to the first city visited.

A highly popular and very simple move for the traveling salesman problem is the "2-opt" move (see fig. 1), which is applied iteratively to transform an initial tour into one that is locally optimal (i.e. that cannot be further improved by such moves).

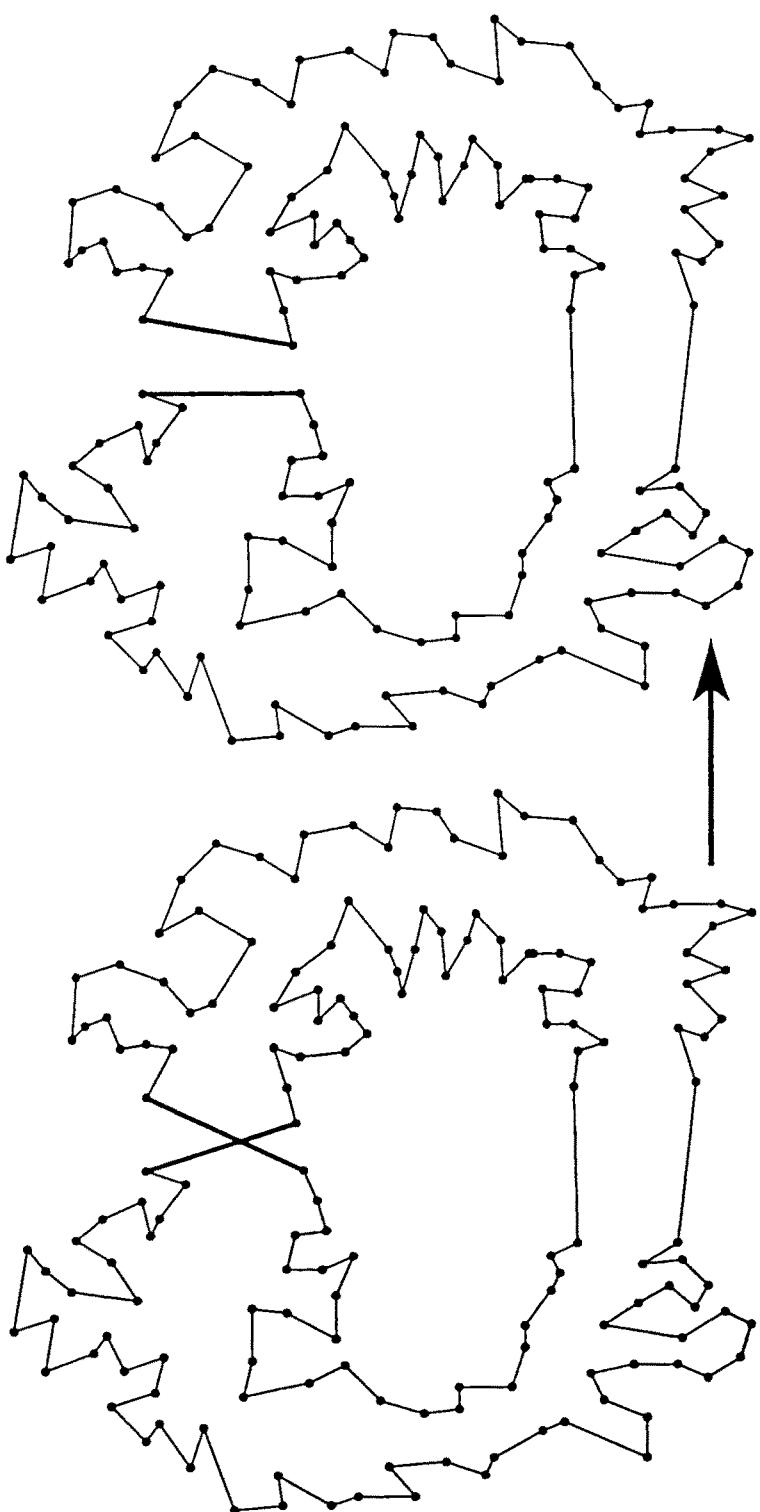


Fig. 1. 2-opt move.

Such 2-opt moves make it possible to quickly find reasonably good local optima of the problem. However, a number of other moves, requiring more effort to implement, are known to be stronger. A TS method based on 2-opt moves is partly affected by their short-sightedness, and typically will "cross" two short edges when it initiates a retreat from a local minimum. (Crossed edges are known to be undesirable in Euclidean traveling salesman problems.) If the tabu list size is too small (which implies that more of the small 2-opt changes are required to escape the basin of attraction than the size of the list), the search will cycle around a local minimum, successively creating and removing small crosses. Conversely, if the size of the list is somewhat larger, moves that allow escape from a local minimum will indeed be performed, but the tabu restrictions may render it impossible (without knowledge of an appropriate aspiration condition) to remove the small crosses previously created. Under these circumstances, 2-opt moves by themselves evidently do not define a sufficiently rich neighborhood to obtain very good results.

Using graphical representations of relatively small problems not only disclosed this phenomenon, but also disclosed a way to create a type of diversifying move to counter it. Specifically, comparing solutions obtained by a "2-opt neighborhood version" of TS with known optimal solutions, a relevant diversifying move was discovered to have the form shown in fig. 2.

The new move usefully modifies and expands the search alternatives at the local level, leading to configurations that are unlikely to be visited by a search based on 2-opt moves (since three 2-opt moves are needed to achieve the same outcome as one of the new moves, and one of the component moves may have a very high cost).

Once the diversification effect of the new move is achieved, the simpler 2-opt moves are used to refine the outcome by progressing to a good local optimum. Sequences of 2-opt moves leading to local optima are therefore alternated with sequences of diversifying moves to escape from these local optima.

It should be noted that, among more advanced moves that are incorporated into heuristics for TSPs, a class of compound moves due to Lin and Kernighan [30] has long been known for its effectiveness, and has been incorporated into a highly efficient procedure by Johnson [24]. Also, a new class of "generalized insert" moves recently has been proposed by Gendreau et al. [10] which shows considerable promise. At this writing, neither of these types of moves has been embedded in a tabu search procedure for TSPs, but gains may be expected by doing so, and a similar application of graphical analysis may offer the possibility to enhance such potential implementations.

### *3.1.1. Discovering improvements by target analysis*

Although graphical simulation is admittedly helpful, many types of optimization problems do not lend themselves readily to its use. Can anything be done to determine

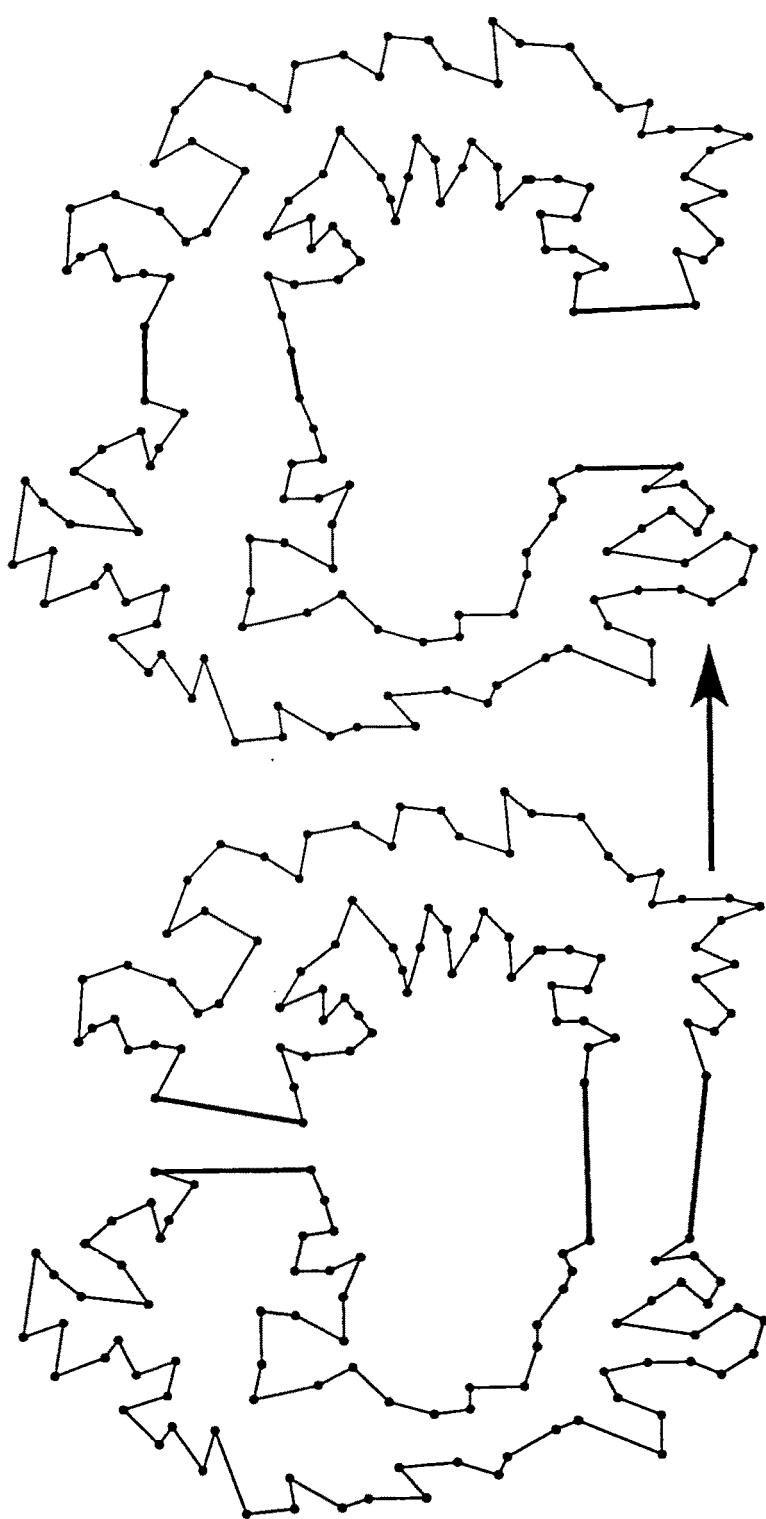


Fig. 2. A diversifying move.

the nature of improved strategies for these problems? Fortunately, the answer to this question is yes.

A learning approach called *target analysis* is conveniently suited for designing improved implementations of tabu search. To begin, target analysis launches an extensive effort to obtain optimal or very high quality solutions to sample problems from the class to be solved. This effort is based on investing greater solution time than normally would be appropriate, and may incorporate a variety of schemes to obtain the best outcomes currently possible. A set consisting of one or more of the best solutions generated for each problem during this initial phase provides the targets to be sought during the next phase of the approach.

The second phase re-solves each sample problem with the goal of collecting information to determine what rules will make the "right choices" to guide successive current solutions to a best solution (e.g. a target solution closest to the current one). Provisional rules are designed as parameterizations of the information collected. To focus on identifying relevant information about preferred solution trajectories, choices may be made during this second phase based on "illegitimate" information derived from hindsight, permitting uniformly good moves to be selected for approaching a target solution, although possibly no currently known rules would prescribe these choices.

The third phase consists of characterizing evaluation functions to give a master decision rule, based on establishing specific combinations and parameter values for the provisional rules of the second phase. Two useful ways to link the information of the second phase to the master decision rule of the third phase are by discriminant analysis [19] and the creation of *move scores* [29].

Move scores, which identify how moves preferably should be evaluated, in contrast to how they actually are evaluated, have provided a useful discovery for scheduling problems that appears potentially relevant for solving other kinds of problems. Tabu search evaluations using standard rules were shown on average to produce high scoring (good) moves when improving moves were unavailable, and to produce medium to low scoring moves when only non-improving moves were available. This phenomenon was exploited by using frequency based memory in a simple diversification approach that penalized frequently occurring solution attributes, activating the penalties precisely during the absence of admissible improving moves. Significant solution improvements resulted for the scheduling problems studied.

To date, these strategies have not yet been applied to develop methods for most other types of problems. A useful extension of target analysis in this context would be to subdivide its operation to develop different classes of rules according to whether the goal is intensification or diversification (e.g. focusing on reaching closest elite solutions during intensification, and on jumping from the vicinity of one elite solution to that of another during diversification).

### *3.1.2. Shifting penalty approach*

The shifting penalty tactic for diversifying the search is illustrated by the procedure of Hertz and de Werra [23] and Costa [4] for solving time table planning

problems. The goal of these problems is to find a feasible course schedule subject to numerous constraints belonging to several categories.

The shifting penalty tactic is applied in these instances to guide the search to discover a feasible solution. First, to define the problem objective, a penalty function is created for every constraint based on the degree to which it is violated for any given schedule. The global objective function then is expressed as a weighted sum of the penalty functions. An important constraint is given a higher weight than another of less importance.

At the beginning of the search, the procedure is driven to satisfy the most important constraints because this will provide the greatest improvements in the objective function. After that, the search undertakes to satisfy constraints associated with lower weights. In order to diversify the search, the shifting penalty tactic dramatically decreases the highest weights after a given number of iterations, and then maintains the decreased values for some number of iterations before reinstating the original values. This standardly causes the new solution to have a different structure than exhibited by the solutions visited before the diversification.

Gendreau et al. [11] have created an adaptation of TS for the vehicle routing problem that automates a rule to change the weights associated with each relaxed constraint. At the beginning of the search, every weight is set to 1. Then, after every 10 iterations, a weight associated with a constraint that was always violated during the past 10 iterations is multiplied by 2; a weight associated with a constraint that was never violated during these 10 iterations is divided by 2, and otherwise the weights stay unchanged.

### *3.1.3. Strategic oscillation and the principle of proximate optimality*

The shifting penalty tactic is an instance of a procedure called *strategic oscillation*, which represents one of the basic diversification approaches for tabu search [12]. The idea is to drive the search toward and away from selected boundaries of feasibility (or selected functional values), either by manipulating the objective function (e.g. with penalties and incentives) or simply by compelling the choice of moves that lead in specified directions. The oscillation may cross the boundary and penetrate to selected depths on either side, or may always approach and retreat from the same side. A common implementation is to alternate a series of constructive (or incrementing) moves with a series of destructive (or decrementing) moves. Recent applications of strategic oscillation have proved beneficial in solving graph partitioning problems [36] and course scheduling problems [32].

In certain settings, strategic oscillation also acquires additional power by an implementation linked to a concept called the Proximate Optimality Principle (POP). This principle stipulates that good solutions at one level are likely to be found close to good solutions at an adjacent level. The term "level" can refer to a stage of a

constructive or destructive process (such as incorporating a specified number of nodes, edges, or variables into a partial solution at that stage), or can refer to a given measure of distance from a specified boundary.

The POP notion is exploited in tabu search by remaining at each successive level for a chosen number of iterations, and then incorporating the best solution found (for the conditions defining the level) to initiate a move to the next level. For example, a level may be characterized by compelling values of a function to fall in a given range, or by requiring a given number of elements to be included in a partial construction. Then the process can be controlled to remain at a specified level by employing a neighborhood whose moves maintain the functional values (or numbers of elements) within the specified limits. This type of approach is sufficiently general to be applied in a TS branching algorithm for mixed integer programming problems [16]. An application of the method by Kelly et al. [25] demonstrates its ability to achieve outcomes that are significantly better than found with alternative procedures for a class of difficult confidentiality problems.

Interesting similarities and contrasts exist between the POP concept of "level" and the simulated annealing notion of "temperature". Each refers to a succession of adjacent states. However, temperature is a measure of energy, as reflected in an objective function change, and the SA mechanism for incorporating this measure is to bias acceptance criteria to favor moves that limit "negative change", according to the temperature value. The SA mechanism also does not seek to maintain a construction at a given level with a design for isolating and carrying forward best solutions to an adjacent level. The types of levels encompassed by the POP notion cover a wide range of strategic alternatives (by comparison to temperature, for example), as derived in reference to numbers of variables or constraints, parametric representations of costs or requirements, hierarchies of aggregation or disaggregation, and so forth.

The POP concept may be viewed as a heuristic counterpart of the so called principle of optimality in dynamic programming. However, it does not entail the associated *curse of dimensionality* manifested in the explosion of state variables that normally occurs when dynamic programming is applied to combinatorial problems. If the premise underlying this concept is valid, it suggests that systematic exploration of effective definitions of "levels" and "closeness", and the design of move neighborhoods for exploiting them, may disclose classes of strategies that usefully enlarge the options currently employed.

### 3.2. TECHNICAL IMPROVEMENTS OF THE SEARCH

From now on, we suppose that the types of moves and the criteria for evaluating them are fixed, that other constituents of TS are to be improved. We examine the issues of appropriate neighborhood sizes, tabu list structures and aspiration conditions.

### 3.2.1. Neighborhood sizes and candidate lists

A complete neighborhood examination with TS provides generally high quality solutions (see [37, 40–42]) but may be very expensive in terms of CPU-time. For this reason, it is often important to apply TS in conjunction with a strategy that isolates regions of the neighborhood containing moves with desirable features, putting these moves on a list of candidates for current examination. A few prominent strategies of this type that have found successful application with TS are as follows.

*Neighborhood decomposition strategies.* A useful candidate list approach is to decompose the neighborhood into coordinated subsets at each iteration. A TS aspiration threshold, or means of linking the examination of subsets, commonly is applied to limit the frequency of selecting moves from subsets whose current alternatives are gauged less attractive. This strategy generally makes it possible both to speed up the computation time and to generate some diversity in the search. Laguna et al. [28] have used this approach to limit the domains of jobs that are shifted in machine scheduling and Fiechter [6] has applied such a decomposition to limit exchanges in traveling salesman problems. More recently, in a practical vehicle routing application, Semet and Taillard [38] have succeeded in cutting down computation times by a factor of 3 while simultaneously obtaining better solutions, applying such a decomposition approach to cyclically scan about one fourth of all possible moves at each iteration.

*Elite evaluation candidate lists.* Another technique for scanning a subset of the neighborhood is to store a collection of the most promising, or "elite" (highest evaluation) moves on the candidate list. At a given iteration, the moves belonging to the candidate list are examined first, followed by a subset of the regular neighborhood, gradually replacing candidate list moves that are no longer attractive. Periodically, after a specified number of iterations or when the quality of moves on the candidate list deteriorates below a chosen threshold, a significantly larger portion of the current neighborhood is examined to reconstruct the candidate list. This technique is motivated by the assumption that a good move, if not performed at the present iteration, will still be a good move for some number of iterations. (More precisely, after an iteration is performed, the nature of a recorded move implicitly may be transformed. The assumption is that a useful proportion of these transformed moves will inherit attractive properties from their antecedents.)

A simplified variant of this strategy is to perform every move on the elite candidate list in succession, provided the move remains valid when its turn arrives. This approach, applied to TSPs in [6], permits a large number of moves to be performed scanning new neighborhoods and rebuilding the list, although at some risk of making less desirable moves. The approach may be improved by introducing an aspiration level threshold that moves must satisfy to be selected (see, e.g. [18]).

*Preferred attribute candidate lists.* In some applications it can be advantageous to isolate certain attributes of moves that are expected also to be attributes of good

solutions, and to limit consideration to those moves whose composition includes some portion of these "preferred" attributes. For example, in traveling salesman problems it often happens that good solutions are primarily composed of edges that are among the 10 or 20 shortest edges meeting one of their endpoints. Some studies have attempted to limit consideration entirely to tours constructed from such a collection of edges.

A preferred attribute candidate list, by contrast, seeks to organize moves so that they do not have to be composed entirely of such special elements. However, one or more of these elements are required to be incorporated in a "key segment" of a move, so that all moves containing such a segment can be generated very efficiently. This approach has been used by Gendreau et al. [10] and by Johnson [24] in application to TSPs without reference to TS. More recently, Gendreau et al. [11] have made an effective adaptation of this approach in the TS context for VRP problems. Related advances for VRPs are reported in the study of Osman [34].

*Sequential fan candidate lists.* A type of candidate list that is highly exploitable by parallel processing is a *sequential fan* candidate list. The basic idea is to generate some  $p$  best alternative moves at a given step, and then to create a fan of solution streams, one for each alternative. The several best available moves for each stream are again examined, and only the  $p$  best moves overall (where many or no moves may be contributed by a given stream) provide the  $p$  new streams at the next step.

In the setting of tree search methods such a sequential fanning process is sometimes called *beam search*. A useful refinement called *filtered beam search* has been proposed and studied by Ow and Morton [35] and other refinements (beyond the tree search setting) have been suggested by Glover [15]. A recent implementation for QAP problems that appears highly promising has been carried out by Gavish [9]. For use in the tabu search framework, it is to be noted that TS memory and restrictions can be carried forward with each stream and hence "inherited" in the selected continuations. In this case, a relevant variation is to permit the search of each stream to continue for some number of iterations until reaching a new local optimum. Then a subset of these can be selected and carried forward. Since a chosen solution can be assigned to more than one new stream, different streams can embody different missions in TS, as by giving different emphasis to intensification and diversification. It may be noted that the type of staging involved in successive solution runs of each stream may be viewed as a means of defining levels in the context of the Proximate Optimality Principle, and hence this way of implementing a parallel solution method may also give another approach for exploiting the POP notion.

### 3.2.2. Tabu list types

The choice of appropriate types of tabu lists depends on the problem. Although no single type of list is uniformly best for all applications, some guidelines are

possible. If the size of the neighborhood is small enough to store an item of information for each move attribute used to define a tabu restriction, it is generally worthwhile to store the iteration number that identifies when the tabu restriction associated with such an attribute may be discarded. This makes it possible to test the tabu status of a move in constant time (by reference to whether the move embodies a "tabu attribute"), and the necessary memory space depends on the neighborhood size and not on the "tabu list size" (i.e. the number of elements that might otherwise be recorded, one attribute group for each iteration, to cover the number of iterations that determine tabu status).

In a general way, it appears that tabu lists designed to insure the elimination of cycles of length proportional to the tabu list size provide very good results. (For reasons not mathematically identified, this "worst case" insurance translates into an empirical phenomenon where repeated cycles disappear entirely once the tabu list size achieves a critical length.) In the tabu list types used for very large problems (or complex attribute definitions), as in the TSP implementation of [6], it may not be possible to store an item of information for each (reverse) move. In this case attributes of moves may be stored in sequential tabu lists. In the indicated TSP study, every edge added to the current solution by a move was incorporated in a list  $T_{in}$  and every deleted edge was incorporated in a list  $T_{out}$ . A move was defined tabu if both the incoming edges belonged to  $T_{out}$  and the outgoing edges belonged to  $T_{in}$ . In another study using a different type of candidate list strategy [26], it appeared preferable to store only the shorter of the two added edges on  $T_{in}$  and the longer of the two deleted edges on  $T_{out}$ . In both studies  $|T_{out}|$  was set to a larger value than  $|T_{in}|$ , based on the fact that a TSP tour has  $O(N)$  edges and the total number of edges is in  $O(N^2)$ .

We now discuss the value to be given to the parameter  $t$  that embodies the tabu list size.

### 3.2.3. Tabu list size

Empirically, tabu list sizes that provide good results often grow with the size of the problem. However, no single rule (even an empirical one) gives good sizes for all classes of problems. This is partly because an appropriate list size depends on the strength of the tabu restrictions employed (where stronger restrictions are generally coupled with smaller sizes). The way to identify a good tabu size for a given problem class and choice of tabu restrictions is simply to watch for the occurrence of cycling when the size is too small and the deterioration in solution quality when the size is too large (caused by forbidding too many moves). Best sizes lie in an intermediate range between these extremes. In fact, the best approach is to allow the size in this intermediate range to vary.

These considerations may be clarified by our illustrated adaptation of TS for the QAP: in fig. 3 we plot the value of the best solution found and the mean cost

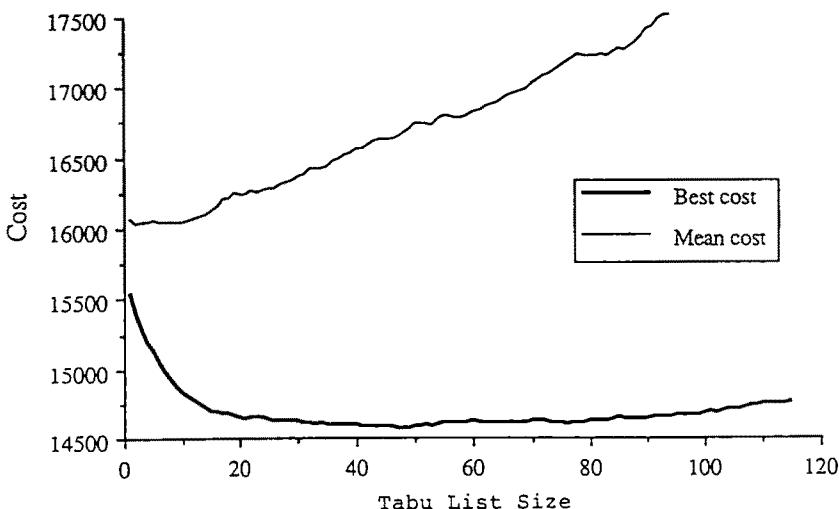


Fig. 3. Influence of tabu list size.

of the visited solution during a search with a given number of iterations. These values are plotted as functions of the tabu list size. We see that both curves decrease for small sizes and then increase as the size grows. However, the minima are reached for a smaller size for the mean cost than for the best cost. This translates into the fact that a small tabu list size is preferable for exploring the solution near a local optimum and a larger tabu list size is preferable for breaking free of the vicinity of this local minimum.

Varying the tabu list size during the search provides one way to take advantage of this effect. For the QAP, the approach of Taillard [42] selects this size randomly from an interval ranging from  $t_{\min} = \lfloor 0.9N \rfloor$  to the value  $t_{\max} = \lceil 1.1N \rceil$  (where  $N$  is the dimension of the problem). The chosen size is maintained constant for  $2t_{\max}$  iterations, and then a new size is selected by the same process.

Other simple types of dynamic tabu list approaches include systematically varying the list sizes over three different ranges (small, medium and large), as applied to telecommunications bandwidth packing problems by Laguna and Glover [29], and introducing "*moving tabu gaps*" as in the QAP approaches of Skorin-Kapov [39] and Chakrapani and Skorin-Kapov [2]. In each of these cases, the dynamic tabu list approach proved superior to using a static list of fixed size. Further improvements were obtained in these studies by incorporating a longer term frequency based memory as well as the short term recency based memory of the tabu list. We sketch how these improvements were obtained subsequently.

Finally, we note the relevance of two additional forms of dynamic tabu list strategies, one based on uses of *hashing functions*, as suggested by Hansen and Jaumard [21] and as explored in detail by Woodruff and Zemel [46], and the other

based on *sequential logic*, as proposed by Glover [16] and as studied in comparative implementations by Dammeyer and Voss [5].

### 3.2.4. Aspiration criteria

In most applications, an aspiration criterion takes the form: a tabu  $m$  is available (or compelled) to be selected if it can reach a solution  $s_k \oplus m$  better than the best solution  $s_k^*$  obtained up to iteration  $k$ , i.e. if

$$f(s_k^*) > f(s_k \oplus m). \quad (3.1)$$

It can also be useful to apply aspiration criteria differently at different phases of search. Such an approach is particularly motivated by findings involving the use of frequency based long term memory in solving machine scheduling problems [29], alluded to earlier in the discussion of target analysis. In this case the memory records the number of times move attributes (or solution attributes) occur over the search history – specifically, the number of times a job was moved to occupy a particular position for processing. The frequency counts are then weighted to penalize moves whose attributes have higher associated frequencies, applying the findings of target analysis by activating the penalties only after the search began to slow its rate of producing improved solutions, and then only in situations where no admissible (non-tabu) improving moves existed.

A surprising aspect of this implementation was that the best tabu list size did not appear to grow with problem size, and that a variable list size also became less important. Overall, the solutions obtained were considerably superior to those that did not incorporate the longer term memory activated in restricted non-improving phases.

The use of such an approach that penalizes frequently performed moves can be implemented as follows. First, one counts the number of times each move  $m$  is performed, in order to compute its frequency  $fm$ . Then, a penalty  $pm$  is associated with each move:

$$pm = \begin{cases} 0 & \text{if } m \text{ meets an aspiration criterion;} \\ w \cdot fm & \text{otherwise,} \end{cases} \quad (3.2)$$

where  $w$  is a constant. Then the value of a move is:

$$f(x + m) - f(x) + pm. \quad (3.3)$$

The weight  $w$  depends on the problem, on the move type and on the neighborhood. However, in several applications (VRP, QAP, electrical network design) we observed that this weight is approximately proportional to the square root of the size of the neighborhood multiplied by the standard deviation of the value (without penalty)

of every move tried during the search. In addition, such a frequency based memory is almost unaffected by the tabu list size; this means that a good weight  $w$  may be found by optimizing this parameter independently.

The asymmetry between improving and nonimproving search phases underlying the exploitation of such penalties suggests the merit of aspiration criteria that permit tabu status to be disregarded during improving phases, provided this status was created by a move which was also an improving move (or by an improving move of the current improving phase). Otherwise, tabu status is implemented in the usual manner during the improvement phase, subject to a secondary aspiration criterion. This criterion allows tabu moves to be selected if they are the only improving moves available (once an improving phase is in progress). The move then is chosen by selecting a "least tabu" or "most improving" tabu move, or by a varying mix of these criteria. Once a local optimum is reached, such an approach then shifts to a more rigid "better than best" aspiration criterion (e.g. accompanied by frequency based memory as previously indicated), and this criterion is maintained until a new improving phase is launched.

The application of such a strategy, and of related differential phase strategies that combine recency based and frequency based memory in more sophisticated ways, provide a means of exploiting aspiration levels with a considerable degree of adaptiveness, and constitute another area warranting fuller investigation.

### 3.3. COMPUTATIONAL IMPROVEMENTS

We now focus on the issue of speeding up the execution of each iteration of the search. A fundamental tactic, when it can be executed, is to replace the double evaluation of the objective function  $f(s_k)$  and  $f(s_k \oplus m)$ , needed to determine the attractiveness of move  $m$  applied to solution  $s_k$ , by the evaluation of the function  $\Delta(s_k, m)$  defined as the simplified algebraic expression  $f(s_k \oplus m) - f(s_k)$ . If the size of the neighborhood is small enough to store  $\Delta(s_k, m)$  for every move  $m$ , then it is often advantageous to express  $\Delta(s_k, m)$  as a function of  $\Delta(s_{k-1}, m)$  and the move  $m_{k-1}$  performed at the previous iteration. In other words, information computed at one step may help to accelerate the computation required at the next step. Computational simplification of the types have provided significant speed-ups for a number of applications, including the machine scheduling study of Laguna et al. [28] and the QAP study of Taillard [42] previously discussed. In the QAP study the refined objective function evaluation cut down the computation of the neighborhood from  $O(N^4)$  to  $O(N^3)$  and the stored evaluations further reduced the computations to  $O(N^2)$ .

Sometimes such simplifications are not feasible, and other approaches must be sought to speed up the search. The use of a profiler—a program that analyzes the time spent in the procedures and lines of an application—can be helpful in this regard.

In a practical vehicle routing problem [38], the search was accelerated by a factor of 1000 by isolating the most expensive routine with such an analysis, and

then significantly reducing the number of times the routine was invoked (by a combined strategy of algebraic simplification, partial neighborhood examination and a relaxed objective function computation). A similar approach has also proved useful in accelerating the solution of scheduling problems in [40].

### 3.3.1. Parallel processing

Parallelization methods provide the ultimate means of speeding up the search. At equivalent costs, distributed computers are potentially more powerful than sequential ones. Concurrent examination of different moves from the neighborhood often makes it possible to reach a speed-up factor close to the ideal one

$$\text{ideal speed-up} = \frac{T_n + T_u}{T_n/p + T_u}, \quad (3.4)$$

where  $p$  is the number of processors,  $T_n$  is the sequential time to perform the computation to be parallelized and  $T_u$  is the time associated with the non-parallelizable part of the algorithm. For example,  $T_n$  may correspond to the time spent to evaluate the neighborhood and  $T_u$  may correspond to the update of the information structure when a move is performed.

Using such a technique, significant speed-up can be achieved for many problems. Assuming that the problem and the type of move are adapted to perform many moves in parallel, such a technique may be useful where a great quantity of moves must be performed and where obtaining the global minimum is not a prime necessity. This may be illustrated in our traveling salesman example with the type of diversification move described previously. As noted, after such a move is executed, a series of 2-opt moves is performed to reach a new local optimum. The structure of the problem suggests that appropriate tour modifications will probably occur on portions of the tour that are near the edges modified by the new move (see fig. 4). If the end points of the four paths to be reoptimized are fixed, every path may be optimized simultaneously.

It has been found advantageous to perform several diversifying moves successively using an elite evaluation candidate list. As a result, there are many places where the tour has been optimized again. This fact is exploited in [6] by cutting the tour into subpaths having about the same number of vertices—approximately 50—and optimizing every subpath in parallel. Then, another way of cutting the tour is selected, repeating until improvements become negligible.

Unfortunately, the assumptions needed to be able to perform parallel moves based on such an implicit decomposition are extremely strong and are met only for a few problems. A quite natural and less restricted parallelization process, that works for every problem where we have undertaken to apply it, is to perform many independent searches at a time, each starting with a different initial solution or/and using a different set of parameters.

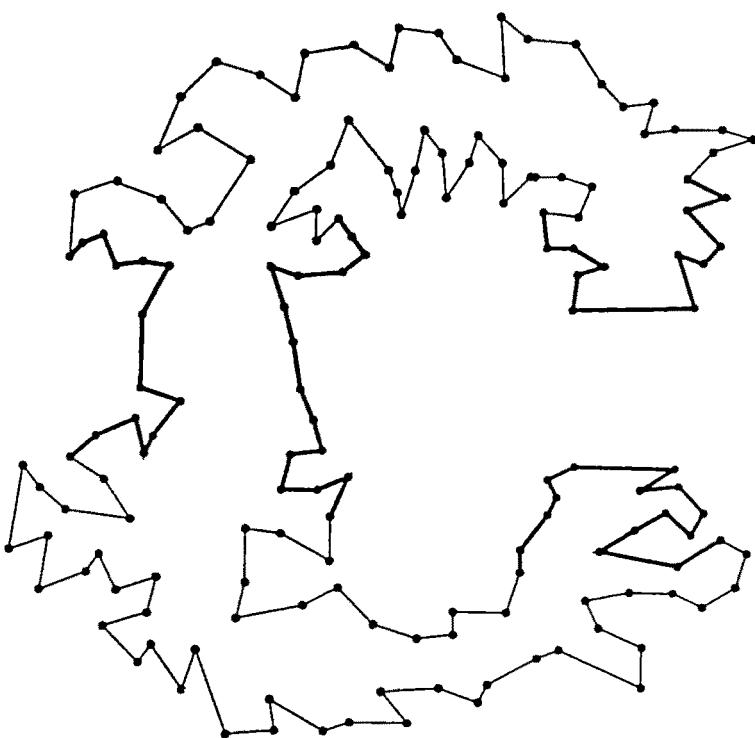


Fig. 4. Parts of the tour that probably have to be reoptimized after a new diversifying move.

Surprisingly, this straightforward type of parallelization is very efficient for a number of processors not exceeding a few dozen. It is also easy to implement, and may be efficient even when simulated on a sequential computer (as for example where the parameters and structuring of the method are not optimally tuned). Results from applying this type of parallel implementation may be found in [40–42]. We also recall that the type of parallel processing approach discussed in connection with sequential fan candidate lists merits fuller consideration.

#### 4. Concluding remarks

We have undertaken to present the main features of tabu search and to sketch some of the improvements that can be obtained by various refinements. Experiments have shown that TS is able to obtain results that match or surpass the best known outcomes in a variety of optimization settings. Nevertheless, our understanding of TS is not complete. New implementations continue to teach us new lessons and to suggest new refinements.

At the same time, mathematical analysis to explain (and ultimately enhance) the performance of tabu search is an area open for examination. Evidently, such an

analysis will embody elements of artificial intelligence (concerning integrated uses of memory) and likely will call upon concepts of probabilistic reasoning. In addition, graph theoretic arguments may prove relevant, based on representing solutions as nodes and moves as arcs—with the goal of determining why trajectories guided by certain forms of memory (in solution spaces with particular structures) turn out to be more effective than others. Finally, connections between TS and heuristic elaborations of dynamic programming (as embodied, for example in the POP notion and strategic oscillation) may provide a fruitful avenue for exploration. The present gap between empirical efficiency and mathematical demonstration invites an effort to bring these realms into closer harmony, with the possibility of creating a foundation for the "next generation" of tabu search.

## References

- [1] R.E. Burkard, Quadratic assignment problems, *Eur. J. Oper. Res.* 15(1984)283–289.
- [2] J. Chakrapani and J. Skorin-Kapov, Massively parallel tabu search for quadratic assignment problem, *Ann. Oper. Res.* (1993), this volume.
- [3] Committee on the Next Decade of Operations Research (Condor), *Operations research: The next decade*, *Oper. Res.* 36(1988).
- [4] D. Costa, A tabu search algorithm for computing an operational time table, Working Paper, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Switzerland (1990).
- [5] F. Dammeyer and S. Voss, Dynamic tabu list management using the reverse elimination method, *Ann. Oper Res.* (1993), this volume.
- [6] C.N. Fiechter, A parallel tabu search algorithm for large scale traveling salesman problems, Working Paper 90/1, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Switzerland (1990).
- [7] G. Finke, R.E. Burkard and F. Rendl, Quadratic assignment problems, *Ann. Discr. Math.* 31(1987)61–82.
- [8] A. Frieze, J. Yadegas, S. El-Horbaty and D. Parkinson, Algorithms for assignment problems on an array processor, *Parallel Comp.* 11(1989)151–162.
- [9] B. Gavish, Manifold search techniques applied to the quadratic assignment problem, Technical Report, Owen Graduate School of Management, Vanderbilt University (1991).
- [10] M. Gendreau, A. Hertz and G. Laporte, New intersection and post-optimization procedures for the traveling salesman problem, CRT-708, Centre de Recherche sur les Transports, Université de Montréal (1990).
- [11] M. Gendreau, A. Hertz and G. Laporte, A tabu search heuristics for the vehicle routing problem, CRT-777, Centre de Recherche sur les Transports, Université de Montréal (1991) to appear in *Manag. Sci.*
- [12] F. Glover, Heuristics for integer programming using surrogate constraints, *Dec. Sci.* 8(1977)156–166.
- [13] F. Glover, Future paths for integer programming and links to artificial intelligence, *Comp. Oper. Res.* 13(1986)533–549.
- [14] F. Glover, Tabu search—Part I, *ORSA J. Comput.* 1(1989)190–206.
- [15] F. Glover, Candidate list strategies and tabu search, CAAI Research Report, University of Colorado, Boulder (July 1989).
- [16] F. Glover, Tabu search—Part II, *ORSA J. Comput.* 2(1990)4–32.
- [17] F. Glover, Tabu search for nonlinear and parametric optimization (with links to genetic algorithms), Technical Report, Graduate School of Business and Administration, University of Colorado at Boulder (1991), to appear in *Discr. Appl. Math.*
- [18] F. Glover, R. Glover and D. Klingman, The threshold assignment algorithm, *Math. Progr. Study* 26(1986)12–37.

- [19] F. Glover, D. Klingman and N. Phillips, A network related nuclear power plant model with an intelligent branch and bound solution approach, *Ann. Oper. Res.* 21(1990)317–332.
- [20] P. Hansen, The steepest ascent mildest descent heuristic for combinatorial programming, *Congress on Numerical Methods in Combinatorial Optimization*, Capri, Italy (1986).
- [21] P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem, *Computing* 44(1990)279–303.
- [22] A. Hertz and D. de Werra, Using tabu search techniques for graph coloring, *Computing* 39(1987)345–451.
- [23] A. Hertz and D. de Werra, *Informatique et Horaires Scolaires*, Output 12(1989)53–56.
- [24] D. Johnson, Local optimization and the traveling salesman problem, *Proc. 17th Annual Colloquim on Automata, Languages and Programming* (Springer, 1990) pp. 446–461.
- [25] J.P. Kelly, B.L. Golden and A.A. Assad, Large-scale controlled rounding using tabu search with strategic oscillation, *Ann. Oper. Res.* (1993), this volume.
- [26] J. Knox, The application of tabu search to the symmetric traveling salesman problem, Ph.D. thesis, Graduate School of Business, University of Colorado.
- [27] A. Kohlen and D. Pesch, Genetic local search in combinatorial optimization, to appear in *Discr. Appl. Math.* (1991).
- [28] M. Laguna, J.W. Barnes and F. Glover, Tabu search methods for a single machine scheduling problem, *J. Int. Manufacturing* 2(1991)63–74.
- [29] M. Laguna and F. Glover, Integrating target analysis and tabu search for improved scheduling systems, School of Business, University of Colorado, Boulder (1991), to appear in *Expert Systems with Application: An International Journal*.
- [30] S. Lin and B.W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, *Oper. Res.* 21(1973)498–516.
- [31] M. Malek, M. Guruswamy, H. Owens and M. Pandya, Serial and parallel search techniques for the traveling salesman problem, *Ann. Oper. Res.* (1989).
- [32] E.L. Mooney and R.L. Rardin, Tabu search for a class of scheduling problems, *Ann. Oper. Res.* (1993), this volume.
- [33] H. Muhlenbein, Parallel genetic algorithms and combinatorial optimization, to appear in *SIAM J. Optim.* (1991).
- [34] I.H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, *Ann. Oper. Res.* (1993), this volume.
- [35] P.S. Ow and T.E. Morton, Filtered beam search in scheduling, *Int. J. Prod. Res.* 26(1988)35–62.
- [36] E. Rolland and H. Pirkul, Heuristic search for graph partitioning, *31st Joint National TIMS/ORSA Meeting*, Nashville, TN (1991).
- [37] F. Semet and I. Loewenton, The traveling salesman problem under accessibility constraints, Report ORWP 92/02, DMA, EPFL (1992).
- [38] F. Semet and E. Taillard, Solving real-life VRPS efficiently using TS, *Ann. Oper. Res.* (1993), this volume.
- [39] J. Skorin-Kapov, Extensions of a tabu search adaptation to the quadratic assignment problem, Harriman School Working Paper HAR-90-006, State University of New York at Stony Brook (1990).
- [40] E. Taillard, Parallel tabu search for the jobshop scheduling problem, Research Report ORWP 89/11, EPFL, DMA Lausanne, Switzerland (1989).
- [41] E. Taillard, Some efficient heuristic methods for the flowshop sequencing problem, *Euro. J. Oper. Res.* 47(1990)65–79.
- [42] E. Taillard, Robust taboo search for the quadratic assignment problem, *Parallel Comp.* 17(1991)443–455.
- [43] N. Ulder, E. Aarts, H.-J. Bandelt, P. van Laarhoven and E. Pesch, Genetic local search algorithms for the traveling salesman problem, *Proc. 1st Int. Workshop on Parallel Problem Solving*, ed. Schwefel and Manner, Lecture Notes in Computer Science 496(1991) pp. 109–116.
- [44] D. de Werra and A. Hertz, Tabu search techniques: A tutorial and an application to neural networks, *OR Spectrum* (1989)131–141.
- [45] D. Whitley, T. Starkweather and D. Fuguey, Scheduling problems and traveling salesman: The genetic edge recombination operator, *Proc. 3rd Int. Conf. of Genetic Algorithms*, Fairfax, VA (1989).
- [46] D.L. Woodruff and E. Zemel, Hashing vectors for tabu search, *Ann. Oper. Res.* (1993), this volume.



# Artificial & Computational Intelligence

**CLZG557**

**M2 : ACO & M3 : Game Playing**

V Indumathi  
Guest Faculty,  
BITS - WILP

**BITS** Pilani  
Pilani Campus

# ACO Pseudocode and notations



## General pseudo-code

### **Procedure ACO**

#### **Schedule Activities**

Initialization  
Construction  
Update Pheromone  
Daemon Actions {optional}  
    // local search, elitism

**End schedule activities**

**End ACO**

## Parameters used in ACO

Parameter	Description
$N$	Total No of ants ; $N > 1$
$\tau_0$	Initial pheromone amount
$\tau_{ij}$	Amount of pheromone deposited while traversing from i to j
$\eta_{ij}$	Cost of link (i,j)
$\alpha$	Importance coefficient of pheromone intensity
$\beta$	Importance coefficient of route cost
$\rho$	evaporation co-efficient; $0 < \rho < 1$
$visit_k$	Visited nodes table of $k^{\text{th}}$ ant
$Q$	Importance - Constant value pertaining to pheromone trail
$f_k$	Route cost obtained by ant k

## ① Initialization

Place predefined number of ants on starting point

Set values for parameters  $\alpha, \beta, \rho$ .

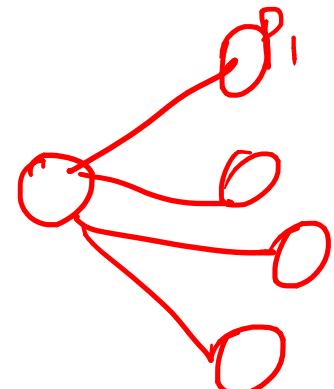
Set  $\tau_0$  to 0.

## ② Construction

Compute the next node transition probability

$$NTP_{ij} = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{h \notin \text{visit}_k} (\tau_{ih})^\alpha (\eta_{ih})^\beta}$$

$(\tau_{ij})^\alpha (\eta_{ij})^\beta$



## Pheromone updation:

Pheromone reinforcement & pheromone evaporation

Direct impact on the exploitation (enhancing found food path) & exploration (discovering new path) of ant algorithms

$$\tau_{ij}^{new} = (\rho)\tau_{ij}^{old} + \Delta\tau_{ij}^k$$

Amount of pheromone deposited on (i,j) by kth ant at that timestamp is given by

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{f_k} & \text{if } k^{th} \text{ ant passes } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

**Stopping criteria:** reaching predetermined number of iterations

**Problem: Reaching pre-determined number of iterations before reaching destination leading to ant drop**

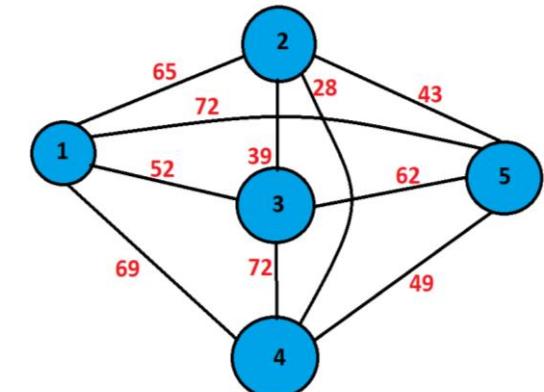
# Travelling Salesman Problem

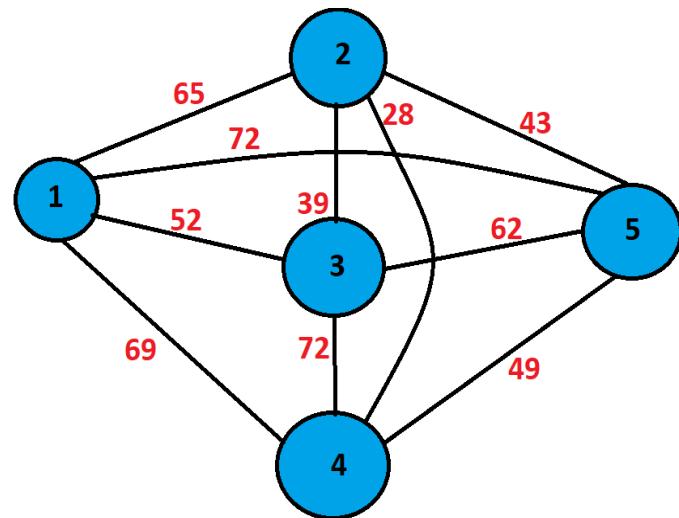


**Problem: Given n cities, the goal is to find shortest path going through all cities and visiting each exactly once**

- Consider a complete graph
- $d_{ij}$  is the route cost over  $(i,j)$   $\{f_k\}$

- Each ant builds its own tour from starting city
- Each ant chooses a town to go to with a probability
- Keep tabs on visit list of each ant
- When tour completed, lay pheromone on each edge visited
- Next city  $j$  after city  $i$  chosen according to probability rule





Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $0 \leq 1$ )

$$Q = 100$$

$$P = 0.1$$

PM

$$\underline{T = 0}$$

$$T_{12} = T_{21} = 0.54$$

$$T_{13} = T_{31} = 0.53$$

$$T_{14} = T_{41} = 0.35$$

$$T_{15} = T_{51} = 0.24$$

$$T_{23} = T_{32} = 0.53$$

$$T_{24} = T_{42} = 0.39$$

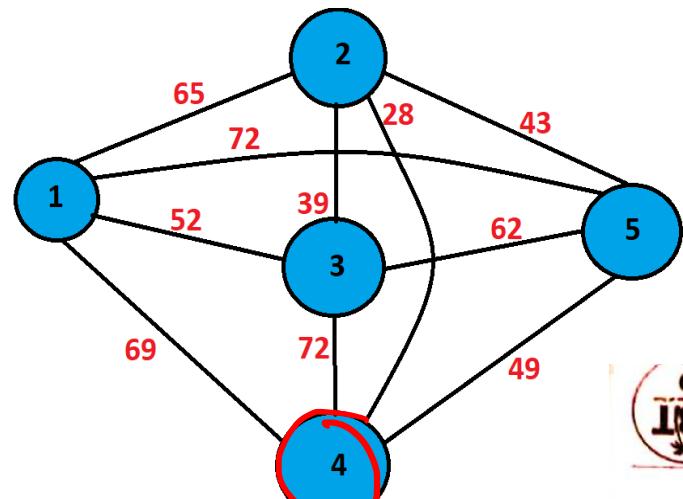
$$T_{25} = T_{52} = 0.18$$

$$T_{34} = T_{43} = 0.32$$

$$T_{35} = T_{53} = 0.90$$

$$T_{45} = T_{54} = 0.68$$

	1	2	3	4	5	dist
1	0	65	52	69	72	
2	65	0	39	28	43	
3	52	39	0	72	62	
4	69	28	72	0	49	
5	72	43	62	49	0	



$$\boxed{I = 0}$$

$$T_{12} = T_{21} = 0.54$$

$$T_{13} = T_{31} = 0.53$$

$$\boxed{T_{14} = T_{41} = 0.35}$$

$$T_{15} = T_{51} = 0.24$$

$$T_{23} = T_{32} = 0.53$$

$$T_{24} = T_{42} = 0.39$$

$$T_{25} = T_{52} = 0.18$$

$$T_{34} = T_{43} = 0.32$$

$$T_{35} = T_{53} = 0.90$$

$$T_{45} = T_{54} = 0.68$$

Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\sigma = 1$ )

$$Q = 100 \quad P = 0.1$$

4

{ } { }



Next Transition Probability

$$P_{41} = \frac{(T_{41})^\alpha (N_{41})^\beta}{(T_{41})^\alpha (N_{41})^\beta + (T_{42})^\alpha (N_{42})^\beta + (T_{43})^\alpha (N_{43})^\beta + (T_{45})^\alpha (N_{45})^\beta}$$

$$N_{41} = 0.014$$

$$N_{43} = 0.02$$

$$N_{45} = 0.014$$

$$N_{42} = 0.036$$

$$= \frac{0.35 \cdot 0.5}{0.5 \cdot 0.75} = 0.168$$

$$= (0.35) * (0.014) + (0.39) * (0.036)$$

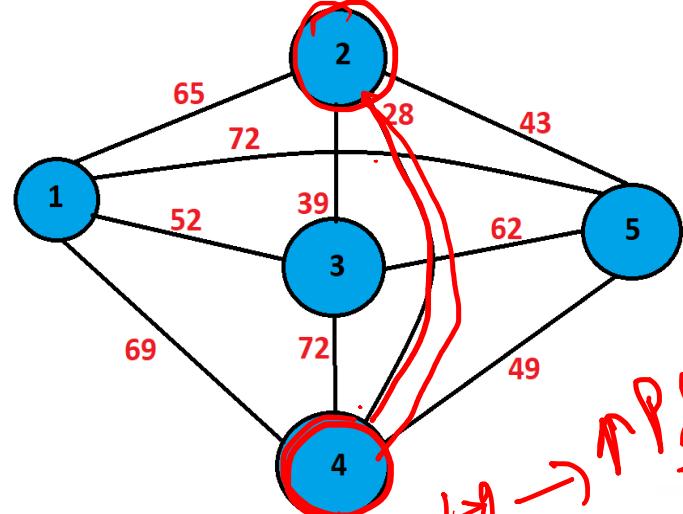
$$+ (0.32) * (0.02) + (0.68) * (0.014)$$

$$= 0.024 + 0.039 + 0.007 + 0.068 = 0.168$$

$P_{42}$

$$P_{42} = \frac{0.024}{0.168} = 0.143$$

$P_{43}$   $P_{45}$   $P_{42}$ ,  $P_{43}$   $P_{44}$



$$T = 0$$

$$T_{12} = T_{21} = 0.54$$

$$T_{13} = T_{31} = 0.53$$

$$T_{14} = T_{41} = 0.35$$

$$T_{15} = T_{51} = 0.24$$

$$T_{23} = T_{32} = 0.53$$

$$T_{24} = T_{42} = 0.39$$

$$T_{25} = T_{52} = 0.18$$

$$T_{34} = T_{43} = 0.32$$

$$T_{35} = T_{53} = 0.90$$

$$T_{45} = T_{54} = 0.68$$

↑ P deposit

4 → 2

Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )

$$Q = 100$$

$$\rho = 0.1$$

4-2  
{4, 2}

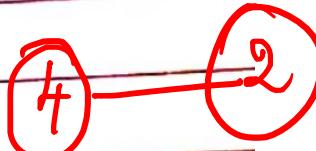
$$P_{41} = \frac{0.024}{0.143} = 0.168$$

$$P_{42} = \frac{0.053}{0.142} = 0.364$$

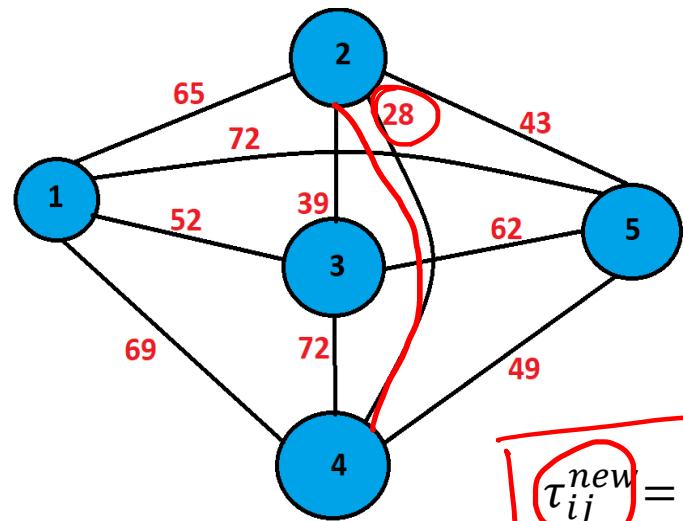
$$P_{43} = \frac{0.023}{0.143} = 0.160$$

$$P_{45} = 0.308$$

$$0.364$$



Since  $P_{42}$  is max, move from 4 to 2.



Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )  
 $Q = 100$   $P = 0.1$

$$\frac{Q}{f_k} = \frac{100}{28}$$

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{f_k} & \text{if } k^{\text{th}} \text{ ant passes } i \text{ and } j \\ 0 & \text{otherwise} \end{cases}$$

$$\tau_{ij}^{\text{new}} = (\rho)\tau_{ij}^{\text{old}} + \Delta\tau_{ij}^k$$

$$I = 0$$

$$\tau_{12} = \tau_{21} = 0.54 * 0.1 \Rightarrow 0.054$$

$$\tau_{13} = \tau_{31} = 0.53 * 0.1$$

$$\tau_{14} = \tau_{41} = 0.35 * 0.1$$

$$\tau_{15} = \tau_{51} = 0.24$$

$$\tau_{23} = \tau_{32} = 0.53$$

$$\tau_{24} = \tau_{42} = 0.39$$

$$\tau_{25} = \tau_{52} = 0.18$$

$$\tau_{34} = \tau_{43} = 0.32$$

$$\tau_{35} = \tau_{53} = 0.90$$

$$\tau_{45} = \tau_{54} = 0.68$$

Compute only for path  
4 - 2 - 1

24 or 4

4 - 2

current best list (4, 2)

Pheromone update (t=1)

$$\tau_{12} = \rho(\tau_{12}) + \Delta\tau_{12} = 0.054$$

$$\tau_{13} = 0.053 * 0.54 + 0 = 0.028$$

$$\tau_{14} = 0.035$$

$$\tau_{15} = 0.024$$

$$\tau_{23} = 0.05$$

$$\tau_{24} = 0.039 + 100/28 = 3.610$$

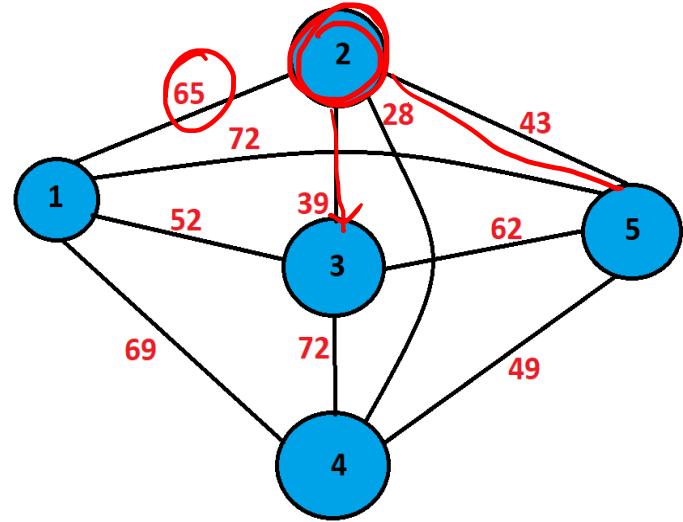
$$\tau_{25} = 0.018$$

$$\tau_{34} = 0.032$$

$$\tau_{35} = 0.09$$

$$\tau_{45} = 0.68$$

3.610



Current best list  $(4, 2)$

Pheromone updation ( $t=1$ )

$$\tau_{12} = P(\tau_{12}) + \Delta\tau_{12} = 0.054$$

$$\tau_{13} = 0.053$$

$$\tau_{14} = 0.035$$

$$\tau_{15} = 0.024$$

$$\tau_{23} = 0.05$$

$$\tau_{24} = 0.039 + 100/28 = 3.610$$

$$\tau_{25} = 0.018$$

$$\tau_{34} = 0.032$$

$$\tau_{35} = 0.09$$

$$\tau_{45} = 0.68$$

~~X~~

$\Rightarrow 0.320 //$

Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\gamma = 1$ )

$$Q = 100$$

$$P = 0.1$$

$S(1, 2)$

Now ant is at 2.

$$P_{21} = 0.171$$

$$P_{22} = 0.610$$

$$P_{23}$$

max

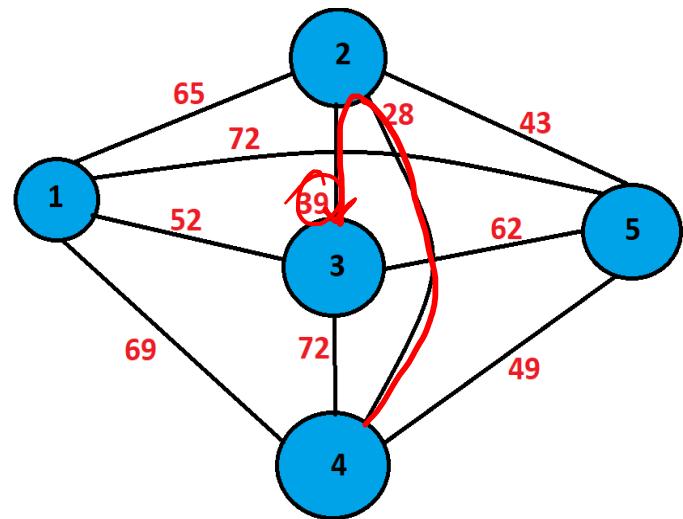
$$P_{25} = 0.026$$

$$P_{21} \Rightarrow (I_{21})^\alpha (M_{21})^\beta \rightarrow \text{Pathcost} = 1/65$$

$$0. (I_{21})^\alpha (M_{21})^\beta + (I_{23})^\alpha (M_{23})^\beta +$$

$$\Rightarrow (0.054)^{0.5} * (1/65)^{0.75} (I_{25})^\alpha (M_{25})^\beta$$

$$\Rightarrow (0.054)^{0.5} * (1/65)^{0.75} + (0.05)^{0.5} (1/39)^{0.75} + (0.018)^{0.5} * (1/43)^{0.75}$$



Initially No. of ants = No. of cities.  
 start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )  
 $Q = 100$   $\rho = 0.1$

23/32

Now ant moves to 3.

Pheromone updation ( $\Delta = ?$ )

$$\tau_{12} = 0.005 \quad \overbrace{P(\text{old})}^{\Delta = 0}$$

$$\tau_{13} = 0.005$$

$$\tau_{14} = 0.003$$

$$\tau_{15} = 0.002$$

$$\boxed{\tau_{23} = \rho(0.005) + 100/\beta \gamma = 0.012569}$$

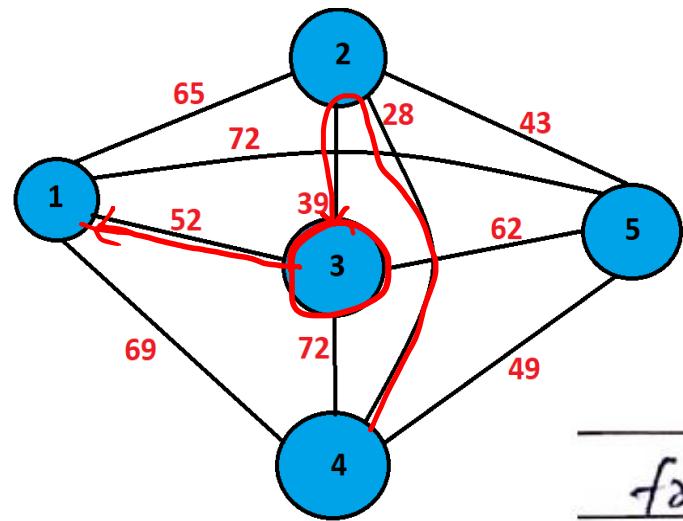
$$\tau_{24} = (0.1)(3.616) = 0.361$$

$$\tau_{25} = 0.001$$

$$\tau_{34} = 0.003$$

$$\tau_{35} = 0.009$$

$$\tau_{45} = 0.006$$



Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )

$$Q = 100$$

$$\rho = 0.1$$

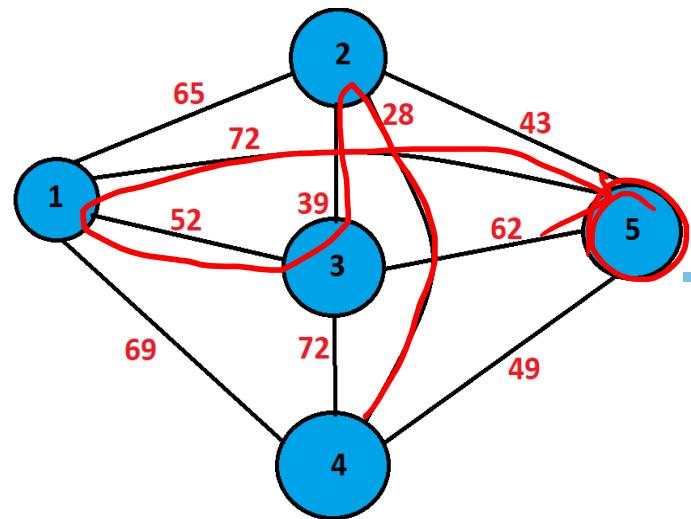
from 3, find  ~~$P_{31}, P_{35}$~~   $P_{31}, P_{35}$

$$P_{31} = 0.51 \leftarrow \text{max. Max}$$

$$P_{35} = 0.46.$$

{4, 2, 3,}

~~$P_{31}$~~   
 ~~$P_{35}$~~



(1)

{4, 2, 3, 1}

(5)

Initially No. of ants = No. of cities.  
start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )  
 $Q = 100$   $\rho = 0.1$

0.1

From 3, ant moves to 1.

Tabu list  $[4, 3, 1]$ .

Pheromone updation.  $\underline{l = 3}$ .

$$T_{12} = 0.0005$$

$$T_{13} = (0.1)(0.005) + 100/52 = 1.9235$$

$$T_{14} = 0.0003$$

$$T_{15} = 0.002$$

$$T_{23} = 0.25$$

$$T_{24} = 0.03$$

$$T_{25} = 0.0001$$

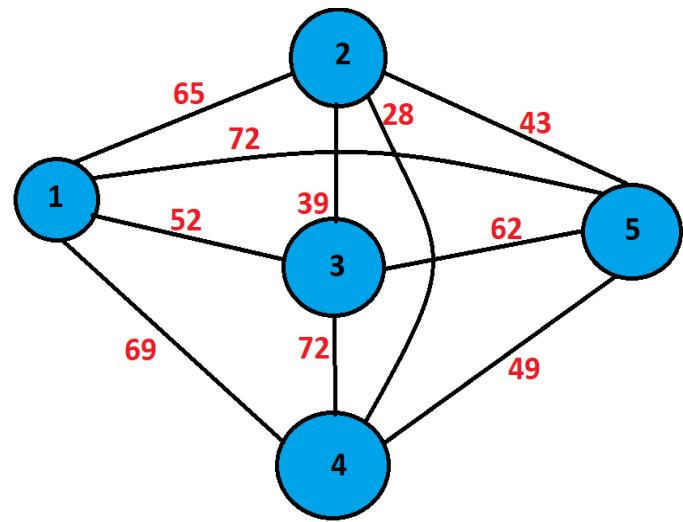
$$T_{34} = 0.0003$$

$$T_{35} = 0.0009$$

$$T_{45} = 0.0006$$

7.

From 1, Now move to 5. since it is the only non visited city.



Initially No. of ants = No. of cities.  
 start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )  
 $Q = 100$   $\rho = 0.1$

Pheromone Update,  $t = 1$ .

$$T_{12} = 0.00005$$

$$T_{13} = 0.019$$

$$T_{14} = 0.00003$$

$$T_{15} = 0.00002 + 100/72 = 1.388.$$

$$T_{23} = 0.025$$

$$T_{24} = 0.003$$

$$T_{25} = 0.00001$$

$$T_{34} = 0.00003$$

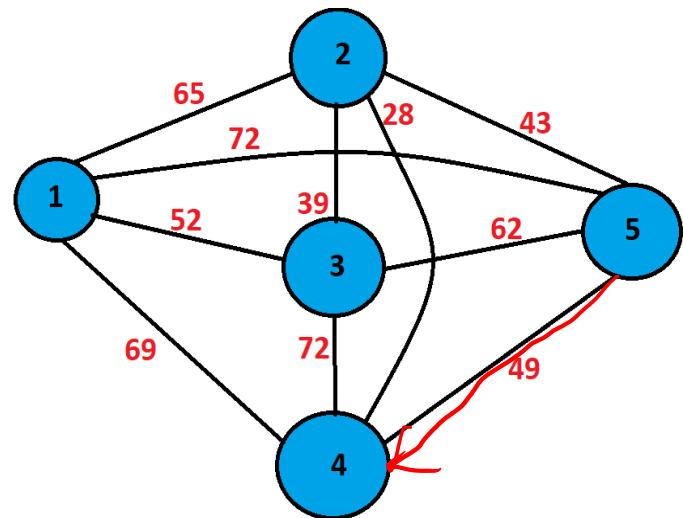
$$T_{35} = 0.00009$$

$$T_{45} = 0.00006.$$

Now back to origin since all states visited

$$\text{update } T_{45} = 0.000006 + 100/49$$

$$T_{45} = 2.0408.$$



4-2-3-1-5  
Path cost:

Initially No. of ants = No. of cities.  
 start at 4.  $\alpha = 0.5$   $\beta = 0.75$  ( $\rho = 1$ )  
 $Q = 100$   $\rho = 0.1$

Pheromone Matrix					
	$t=0$	$t=1$	$t=2$	$t=3$	$t=4$
$T_{12}$	0.54	0.054	0.0054	0.0005	0.00005
$T_{13}$	0.53	0.053	0.0053	0.00035	0.192
$T_{14}$	0.35	0.035	0.0035	0.0003	0.00003
$T_{15}$	0.24	0.024	0.0024	0.0002	0.138
$T_{23}$	0.53	0.053	0.0053	0.25	0.025
$T_{24}$	0.39	0.039	0.0039	0.036	0.0003
$T_{25}$	0.18	0.018	0.001	0.0001	0.00001
$T_{34}$	0.32	0.032	0.003	0.003	0.00003
$T_{35}$	0.90	0.090	0.009	0.0009	0.00009
$T_{45}$	0.68	0.068	0.006	0.0006	0.00006

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

**M3 Game Playing**

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

Adversarial  
Search

$\checkmark h(n) \rightarrow$  informed  
 $\times h(n) \rightarrow$  uninformed

Partially observable  
local search  
 ↳ HC  
 ↳ LS HC  
 ↳ Local beam search  
 ↳ Ant colony  
ACO

# Module 3 : Searching to play games

- A. Minimax Algorithm
- B. Alpha-Beta Pruning
- C. Making imperfect real time decisions

# Learning Objective

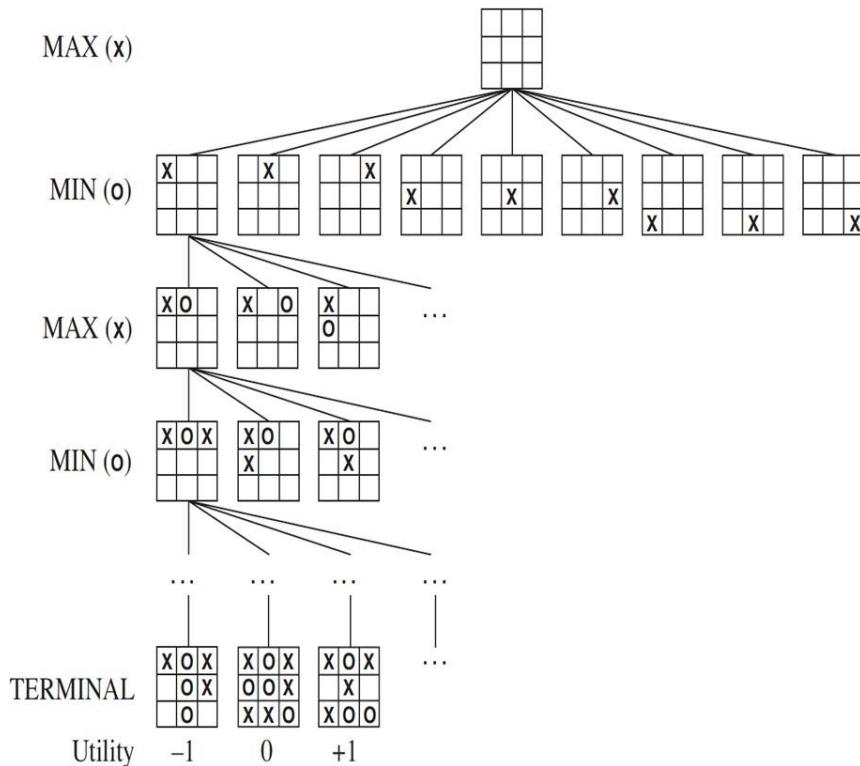
---

At the end of this class , students Should be able to:

1. Convert a given problem into adversarial search problem
  2. Formulate the problem solving agent components
  3. Design static evaluation function value for a problem
  4. Construct a Game tree
  5. Apply Min-Max
  6. Apply and list nodes pruned by alpha pruning and nodes pruned by beta pruning
-

# Task Environment

## Phases of Solution Search by PSA



Assumptions – Environment :

Static ✓

Observable → f | P ✓

Discrete

Deterministic ✓

Number of Agents : Single Multagent

# Game Problem✓

Study & design of games enables the computers to model ways in which humans think & act hence simulating human intelligence.

## AI for Gaming:

- Interesting & Challenging Problem
- Larger Search Space Vs Smaller Solutions
- Explore to better the Human Computer Interaction



## Characteristics of Games:

- Observability
- Stochasticity
- Time granularity
- Number of players



## Adversarial Games:

Goals of agents are in conflict where one's optimized step would reduce the utility value of the other.



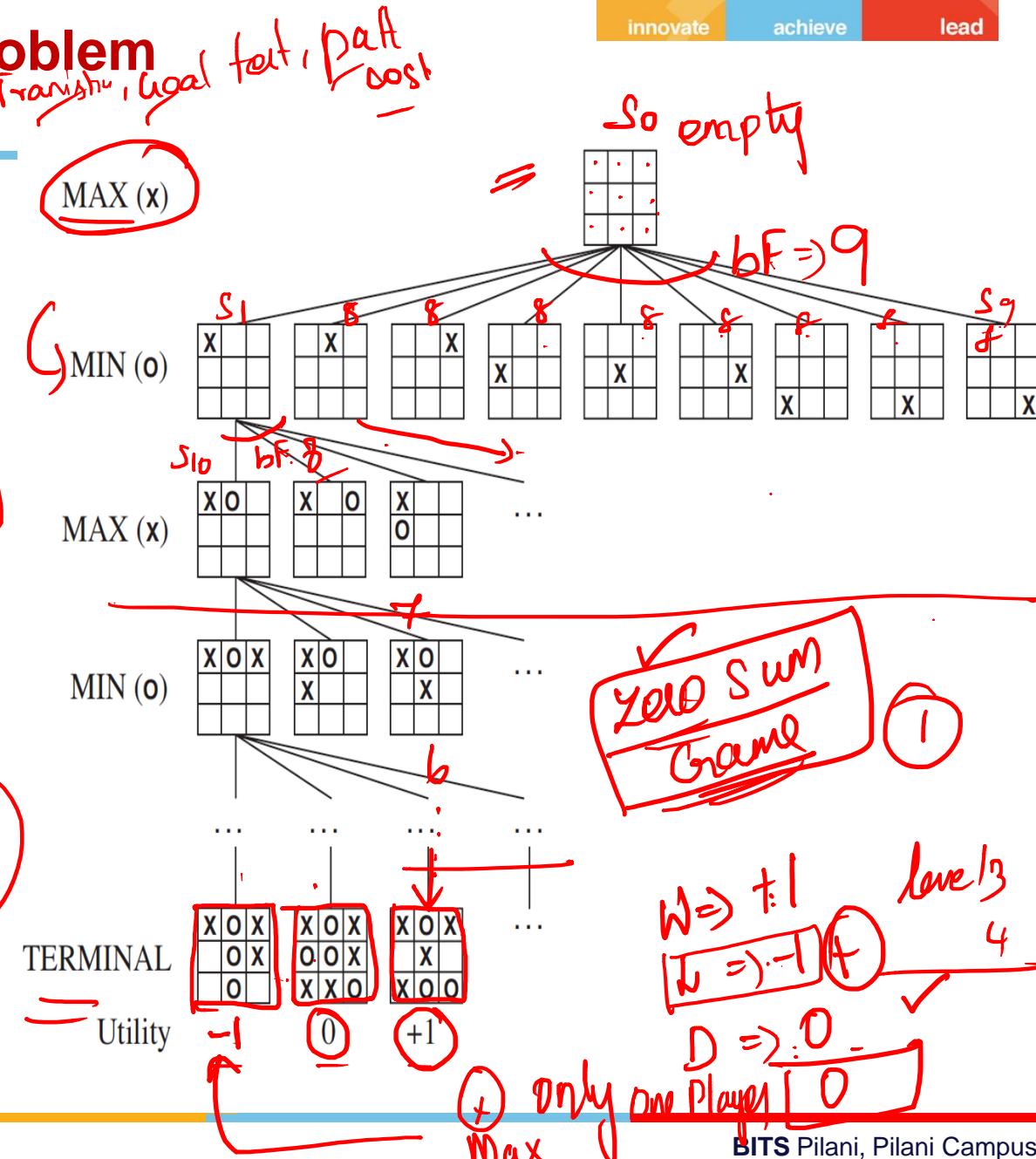
# Games as Search Problem

5 Components IS, Act, Trans, goal test, P cost

PSA : Representation of Game:

- ✓ INITIAL STATE:  $s_0$
- ✓ PLAYER(s)
- ✓ ACTIONS(s)
- ✓ RESULT(s, a)
- ✓ TERMINAL-TEST(s)
- ✓ UTILITY(s, p)

Eg., Tic Tac Toe



# Min-Max Algorithm

---

```
function MINIMAX-DECISION(state) returns an action
  return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$ 
```

---

```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
  return v
```

$\uparrow$  chance of winning  
 $\max$  utility

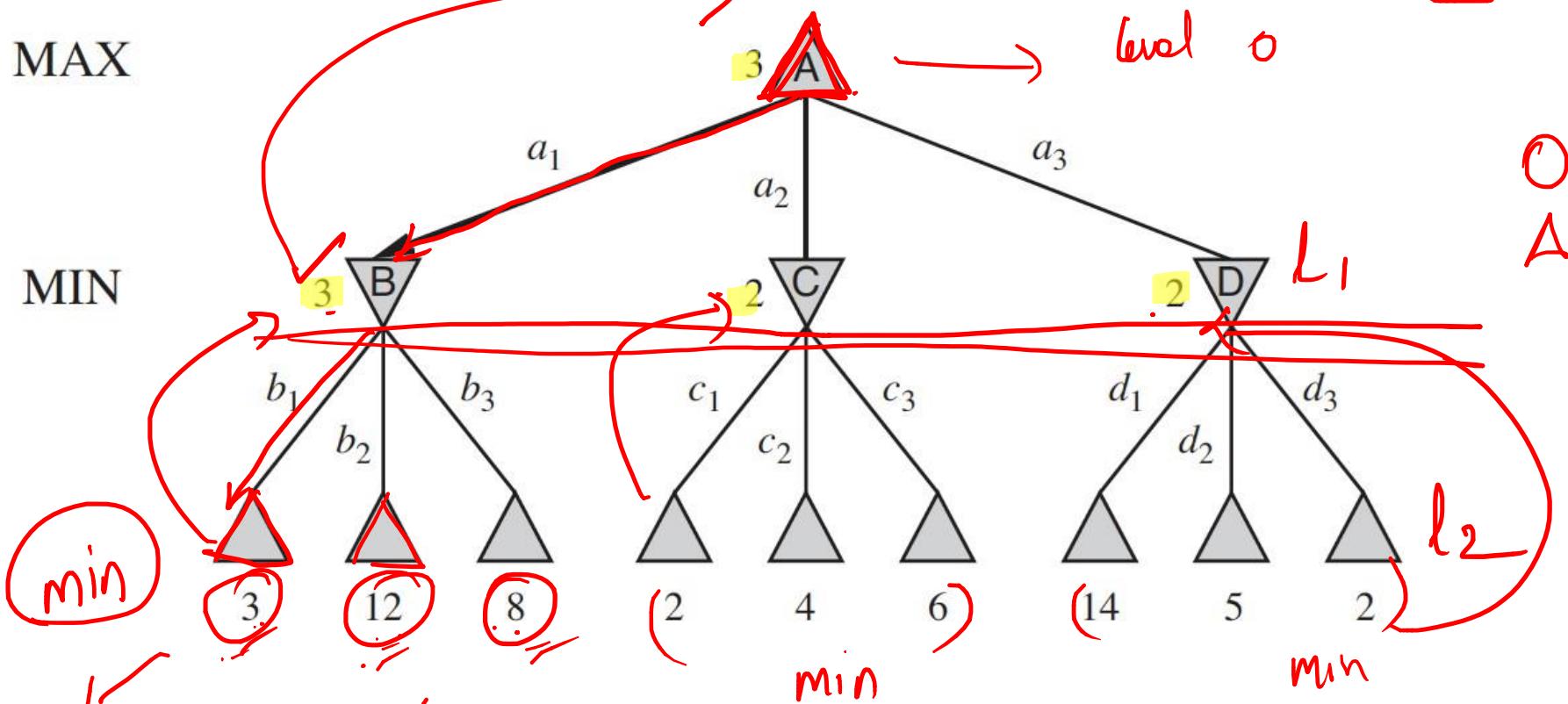
```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
  return v
```

$\downarrow$  chance of winning posn.  
 w.r.t. position  
 min utility

DFS approach

Book Example

$\Delta \rightarrow \text{Max}$   
 $\nabla \rightarrow \underline{\text{Min}}$



static eval  
function

# Design of Static Evaluation Values

start game  
innovate achieve lead

white coin → Max player  
Black coin → Min player

1	Knight → 3
801	→ 1
Queen	→ 9
Bishop	→ 3

18,000  
Set



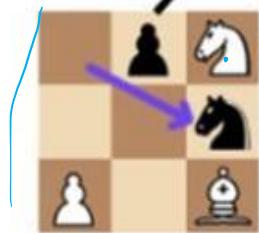
Score:  $\max(\min(3, 3, \dots), \min(-9, -7, \dots), \dots)$



Score:  $\min(3, 3, \dots)$



Score:  $\min(-9, -7, \dots)$



Score:  $1 + 3 + 3 - 3 - 1 = 3$



Score:  $1 + 3 + 3 - 3 - 1 = 3$



Score:  $1 + 3 - 3 - 1 - 9 = -9$



Score:  $3 + 3 - 1 - 3 - 9 = -7$

$$E(S) = h(n)$$

$$V(\text{max}) \leftarrow V(\text{min})$$

black  
3

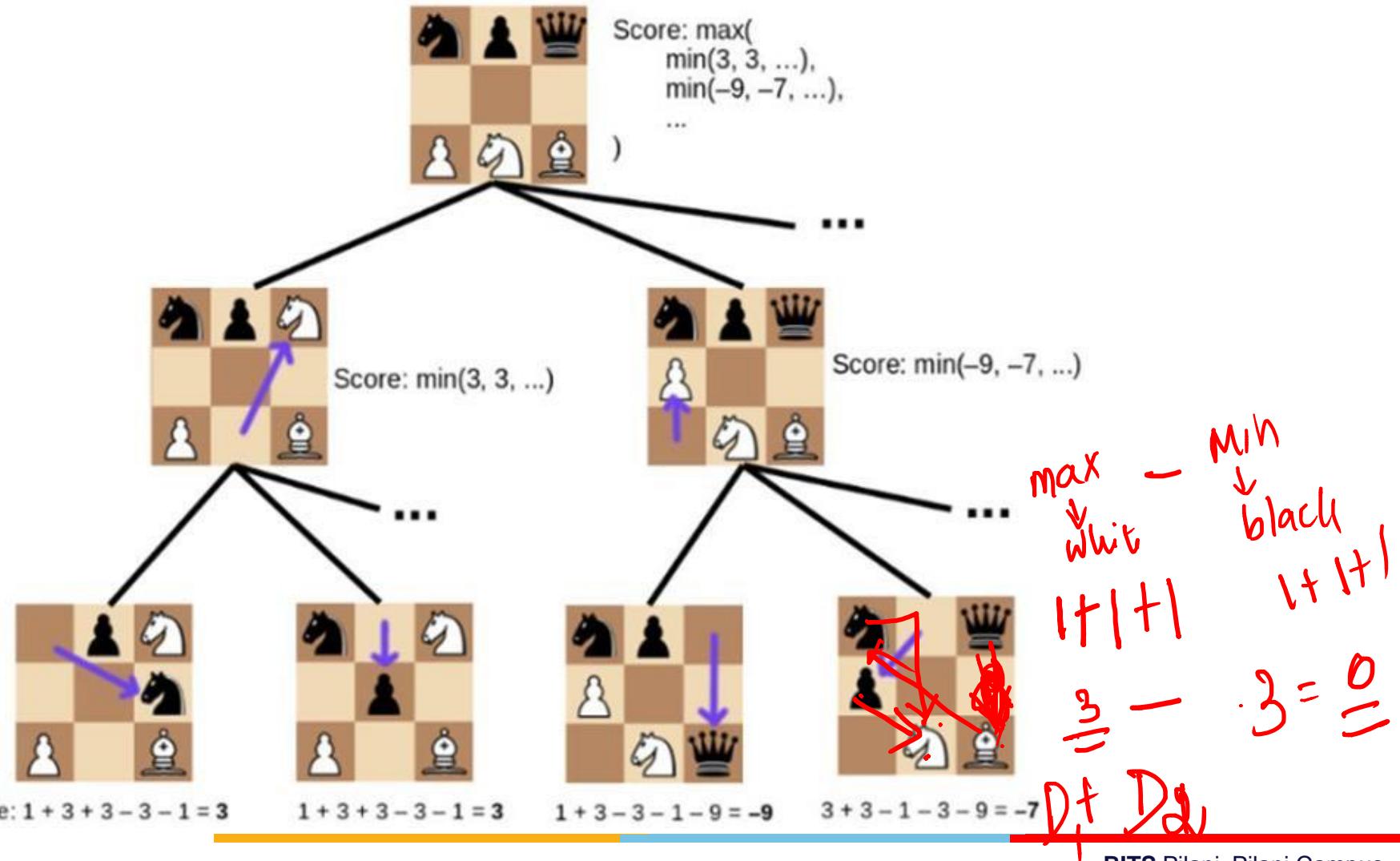
$$V(K+B) - \sum(K+B)$$

$$(3+3) - (3+9+1)$$

$$6 - 13 = -7$$

$$\underline{D_1} + \underline{D_2}$$

# Design of Static Evaluation Values



$$\text{Score: } 1 + 3 + 3 - 3 - 1 = 3$$

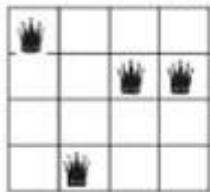
$$1 + 3 + 3 - 3 - 1 = 3$$

$$1 + 3 - 3 - 1 - 9 = -9$$

$$3 + 3 - 1 - 3 - 9 = -7$$

# Design of Static Evaluation Values

N-Queens



1 4 2 2 4

Tic-Tac-Toe

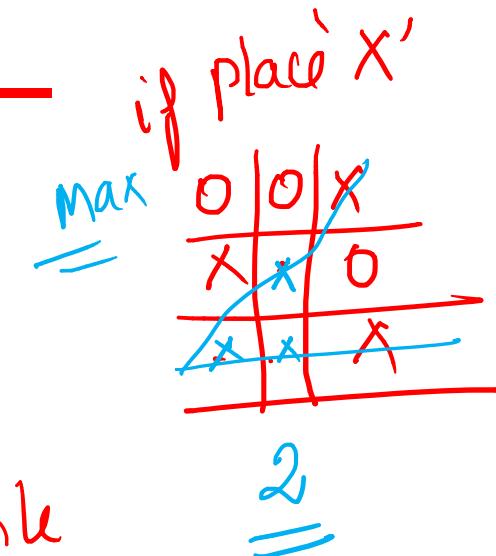
0	0	X
X	0	0
0	0	X

Max's Share	2
Min's Share	1
Board Value	1

N-Tile

2	8	3
1	6	4
7		5

1	2	3
8		4
7	6	5



$D_1 \Rightarrow$  NO of non conf pair  
 $\dots \dots \dots$   
 $D_2 \Rightarrow$  NO of conf pair  
 $D_3 \Rightarrow$  NO of safe pair

$$h(n) \Rightarrow U(\max) - U(\min)$$

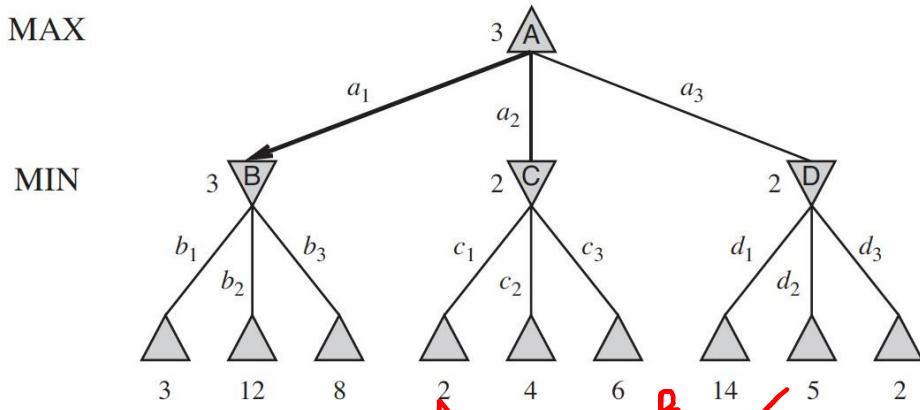
misplaced tile

$$2 - 1 = 1$$

$$\text{Eval}(S) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

$$= 0.6 (\text{MaxChance} - \text{MinChance}) + 0.4 (\text{MaxPairs} - \text{MinPairs})$$

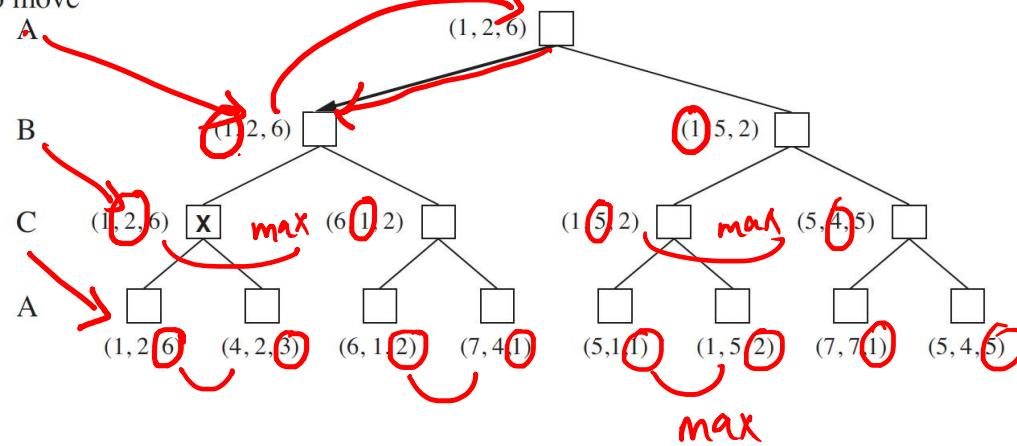
MAX



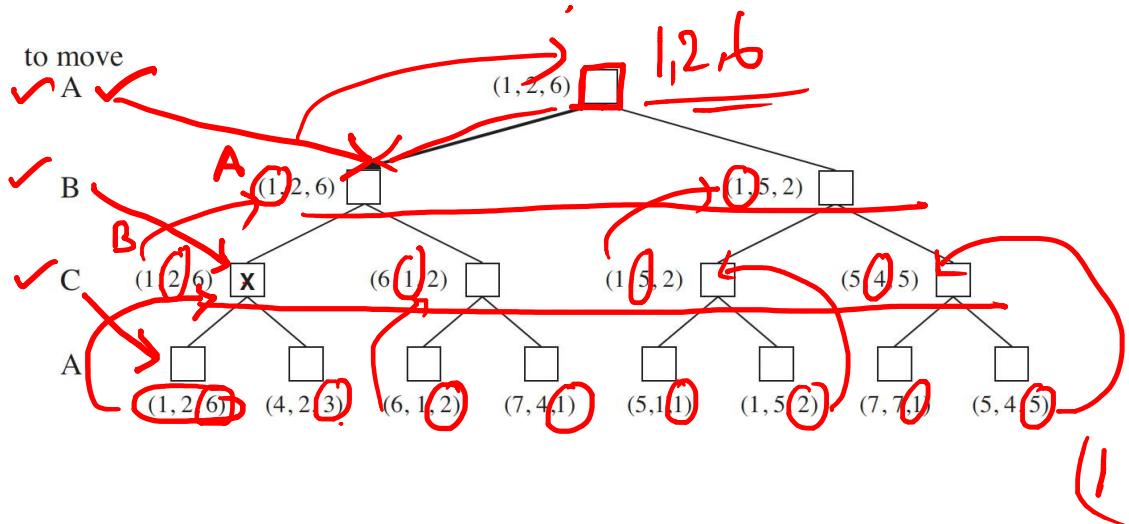
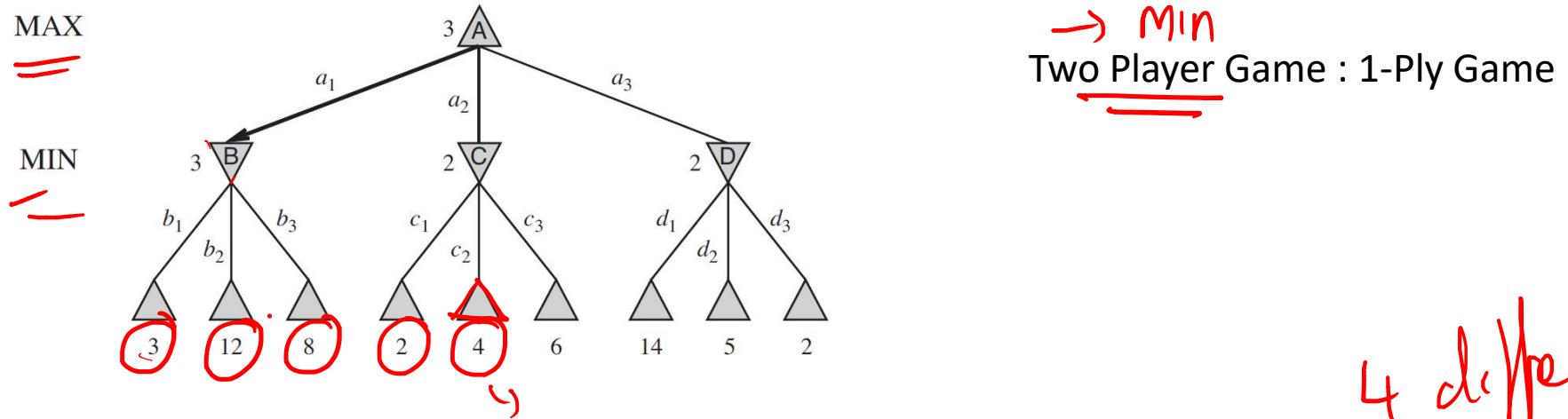
## Two Player Game : 1-Ply Game

A → 1 2 6 → C  
DFS

to move



## Multiplayer Game



Multiplayer Game

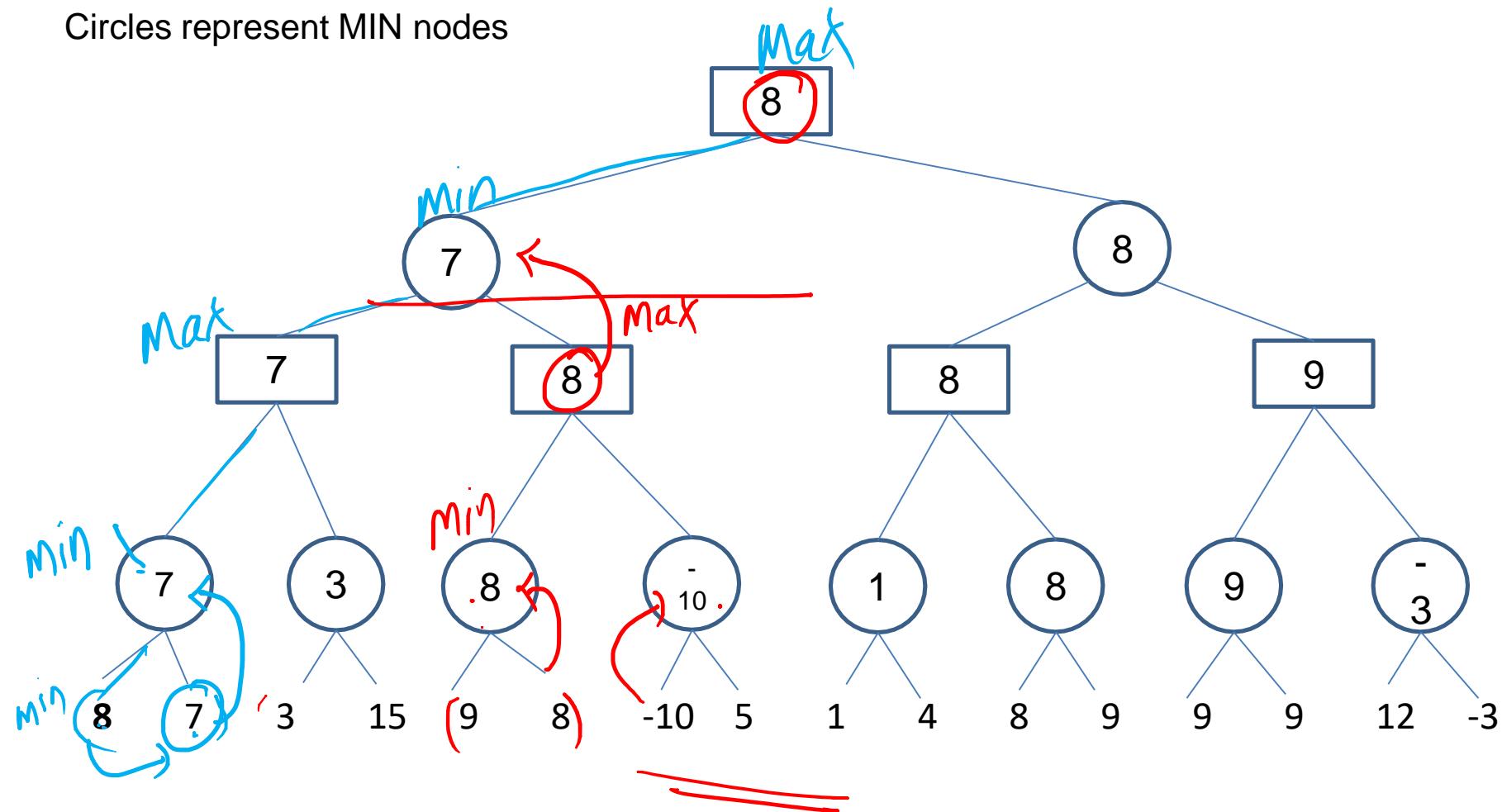
= 3 different value =

Max

# Min-Max Algorithm – Example -1

Squares represent MAX nodes

Circles represent MIN nodes

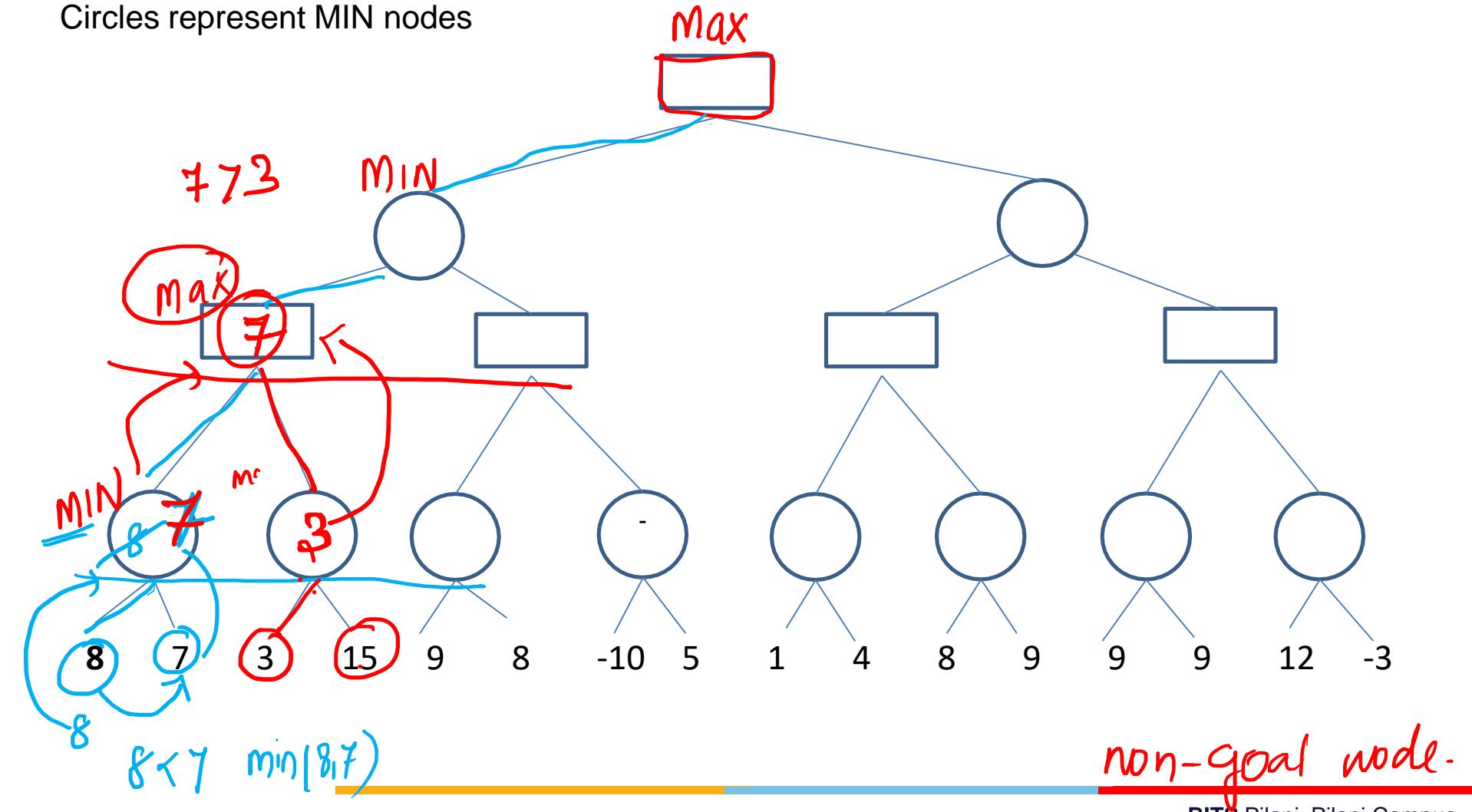


# Min-Max Algorithm – Example -1

Squares represent MAX nodes

Circles represent MIN nodes

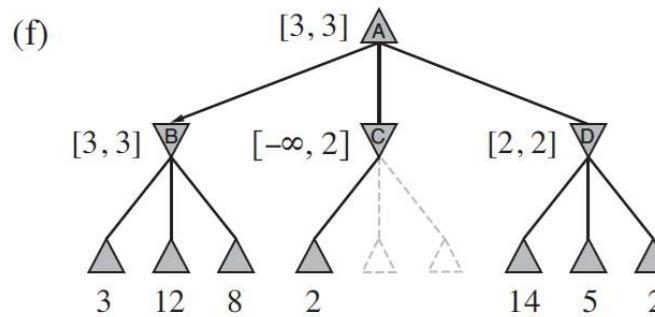
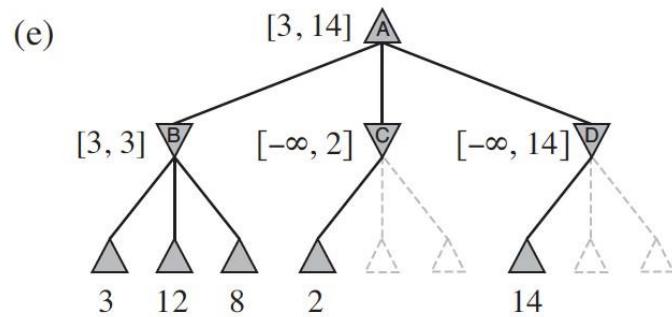
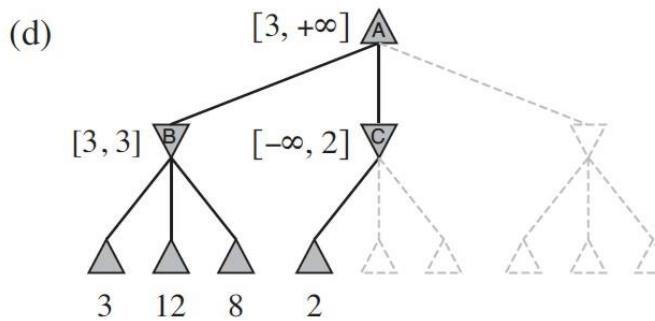
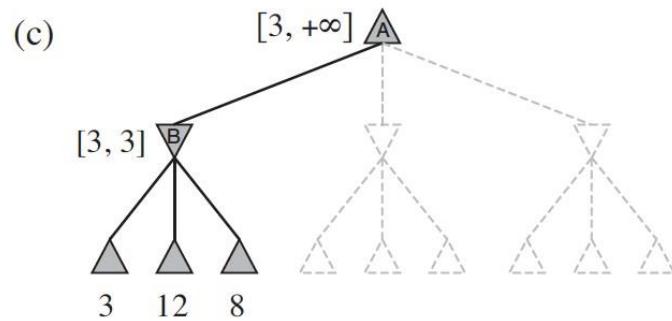
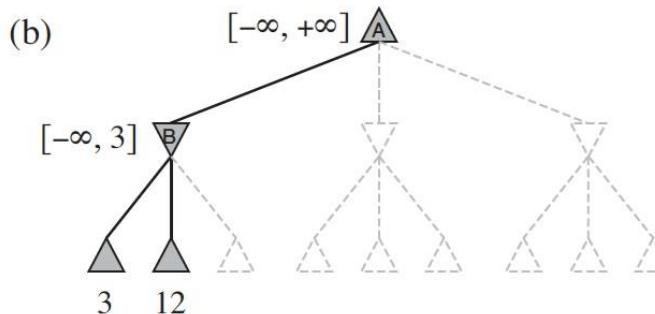
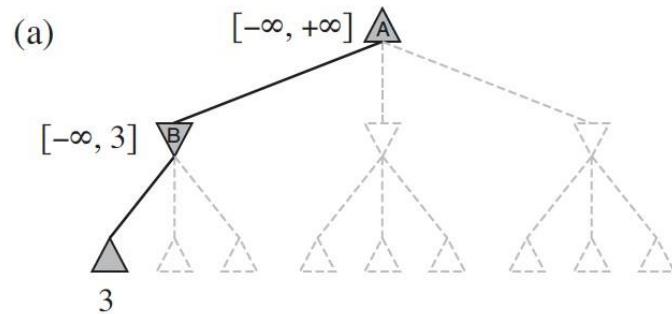
max value  
= min value





# Alpha Beta Pruning

## Book Example



## Alpha beta Modifications

---

```
function ALPHA-BETA-SEARCH(state) returns an action
```

```
    v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
```

```
    return the action in ACTIONS(state) with value v
```

---

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)
```

```
    v  $\leftarrow -\infty$ 
```

```
    for each a in ACTIONS(state) do
```

```
        v  $\leftarrow$  MAX(v, MIN-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
```

```
        if v  $\geq \beta$  then return v
```

```
         $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
```

```
    return v
```

---

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
```

```
    if TERMINAL-TEST(state) then return UTILITY(state)
```

```
    v  $\leftarrow +\infty$ 
```

```
    for each a in ACTIONS(state) do
```

```
        v  $\leftarrow$  MIN(v, MAX-VALUE(RESULT(s,a),  $\alpha$ ,  $\beta$ ))
```

```
        if v  $\leq \alpha$  then return v
```

```
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
```

```
    return v
```

Is it possible to compute the minimax decision for a node without looking at every successor node?

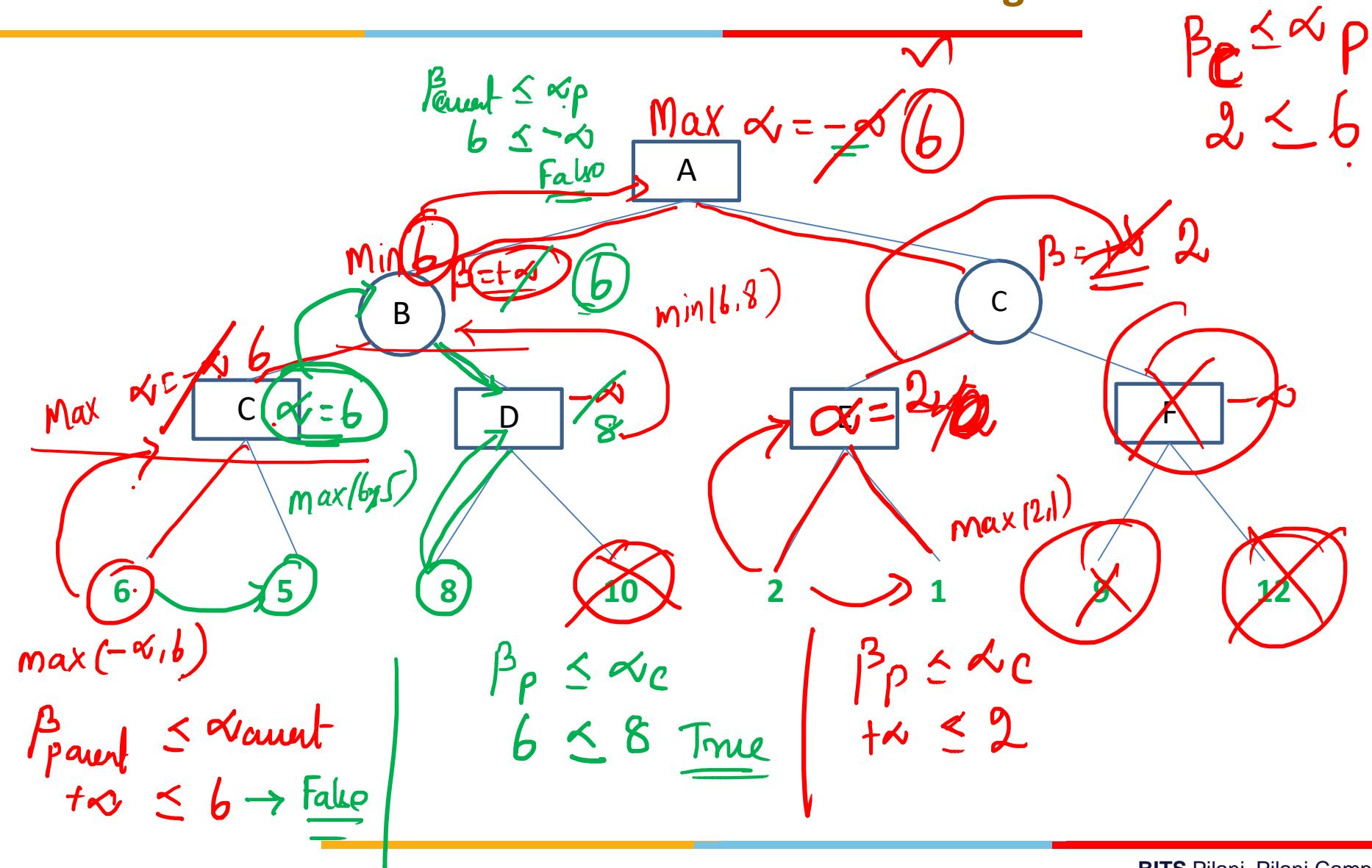
# Alpha – beta Pruning

## Steps in Alpha – Beta Pruning:

1. At root initialize alpha =  $-\infty$  and beta =  $+\infty$ . This is to set the worst case boundary to start the algorithm which aims to increase alpha and decrease beta as much as optimally possible
2. Navigate till the depth / limit specified and get the static evaluated numeric value.
3. For every value VAL being analyzed : Loop till all the leaf/terminal/specified state level nodes are analyzed & accounted for OR until **beta  $\leq$  alpha**.
  1. If the player is MAX :
    1. If VAL  $>$  alpha
    2. then reset alpha = VAL
    3. also check if beta  $\leq$  alpha **then** tag the path as unpromising (TO BE AVOIDED) **and** prune the branch from game tree. Rest of their siblings are not considered for analysis
  2. Else if the player is MIN:
    1. If VAL  $<$  beta
    2. then reset beta = VAL
    3. also check if beta  $\leq$  alpha **then** tag the path as unpromising (TO BE AVOIDED) **and** prune the branch from game tree. Rest of their siblings are not considered for analysis

# Alpha Beta Pruning - Another Example

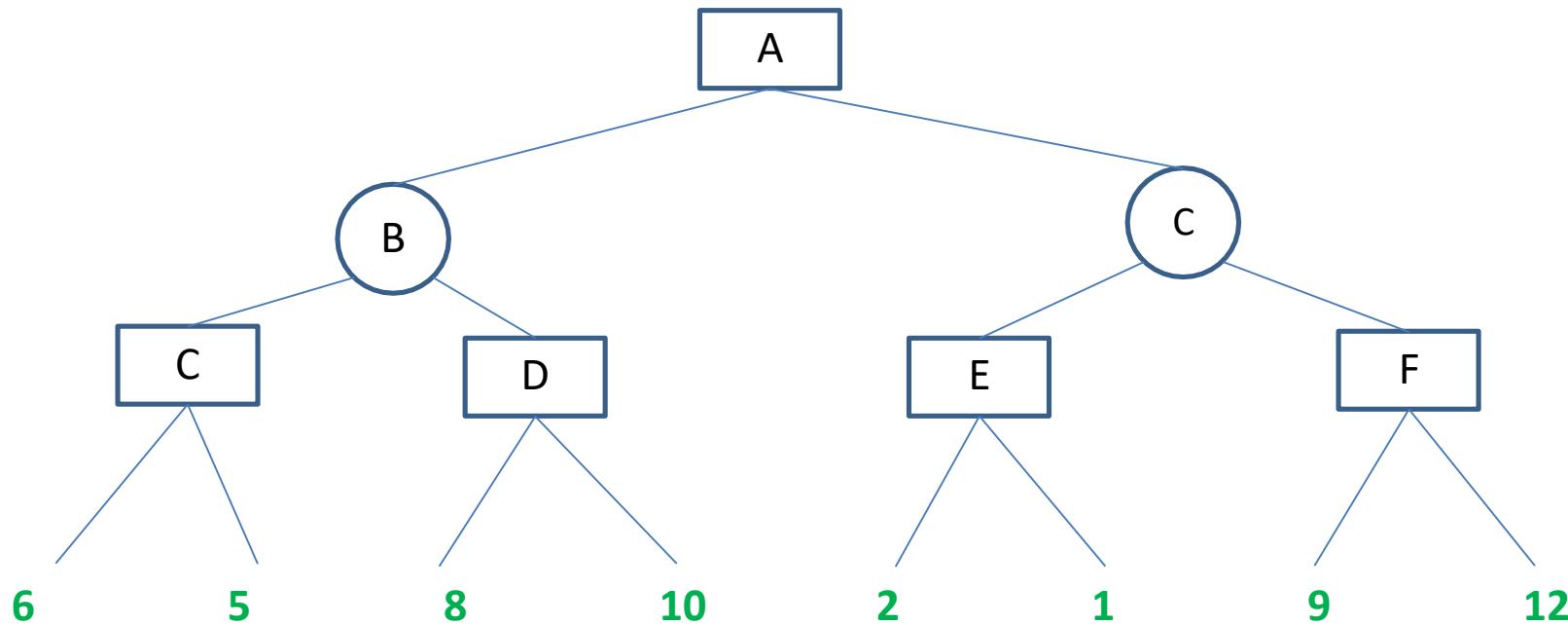
## Idea –Pruning



# Alpha Beta Pruning - Another Example



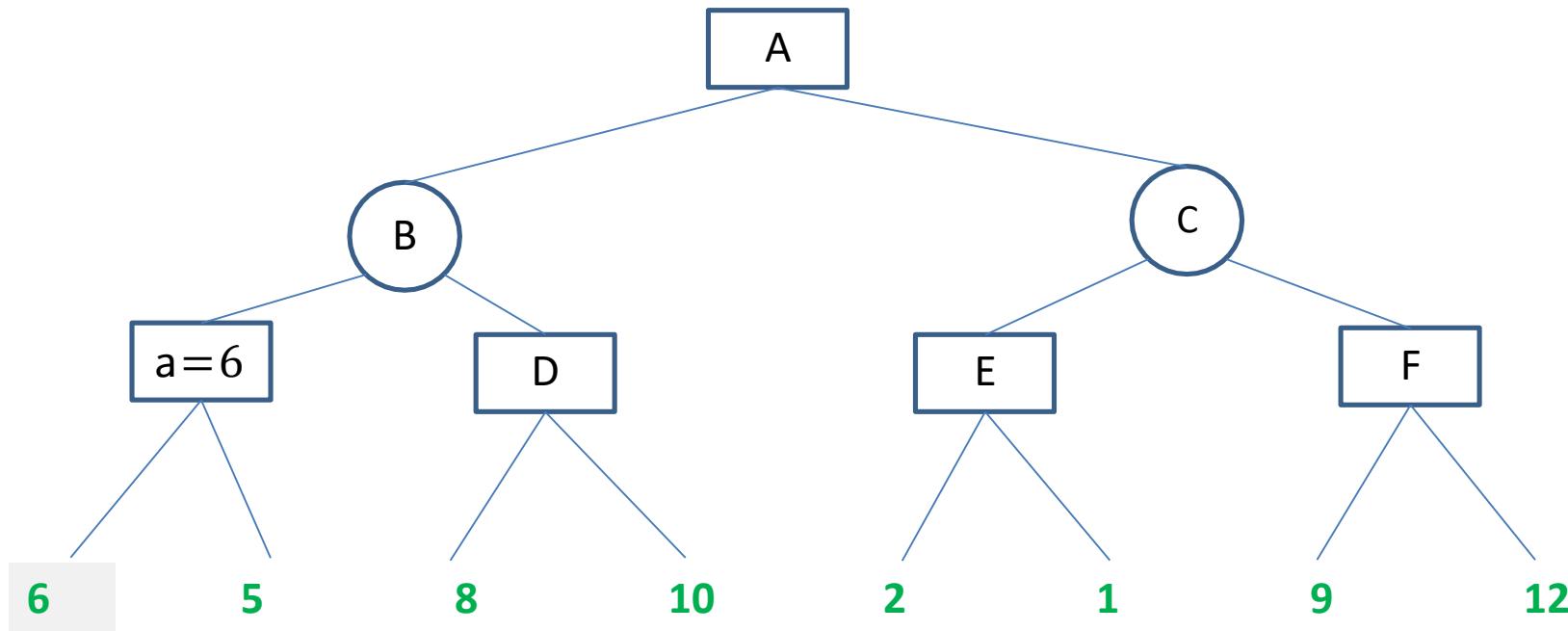
## Idea –Pruning



# Alpha Beta Pruning



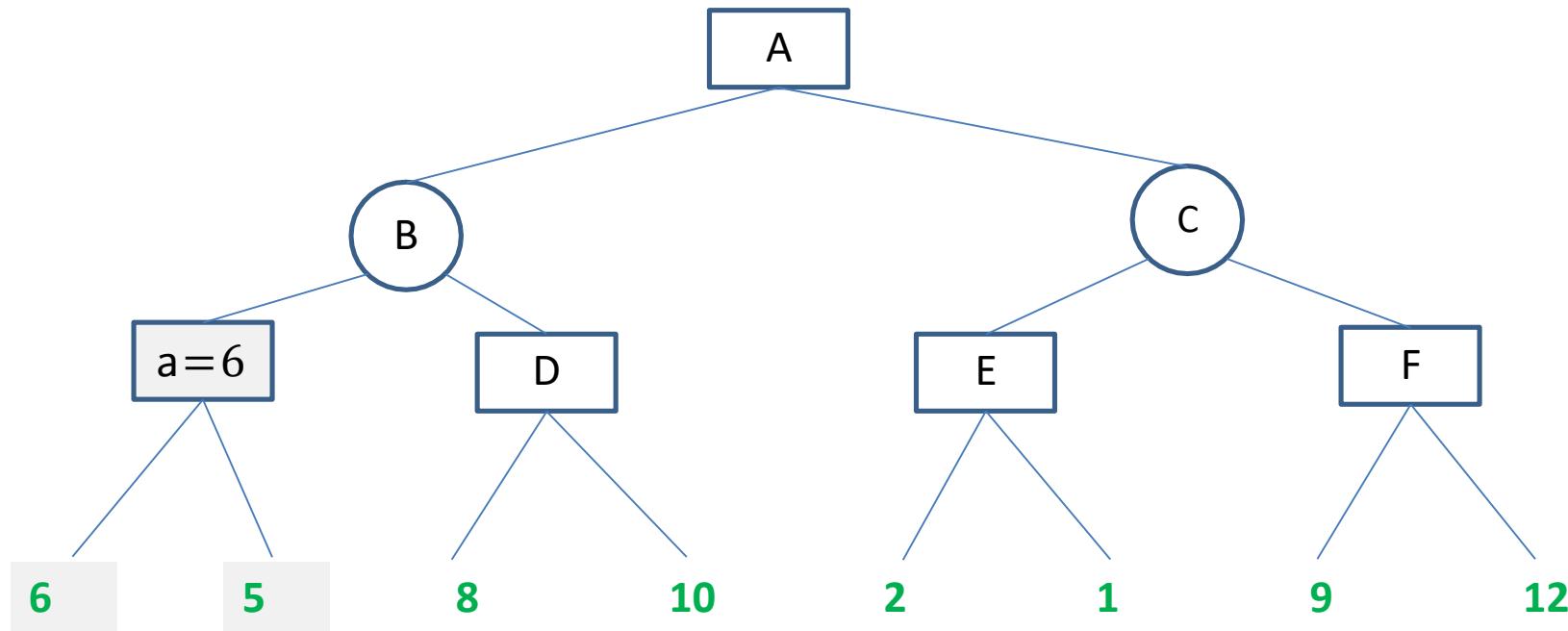
## Idea –Pruning



# Alpha Beta Pruning



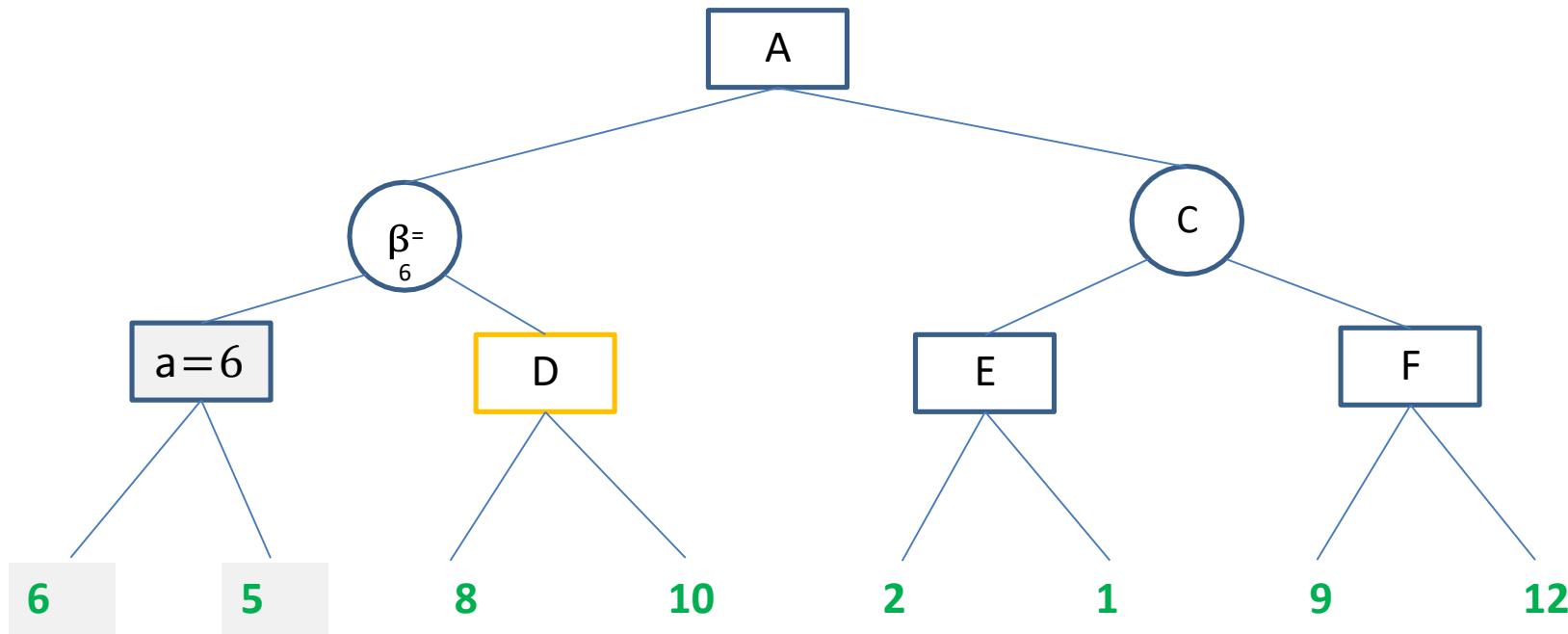
## Idea –Pruning



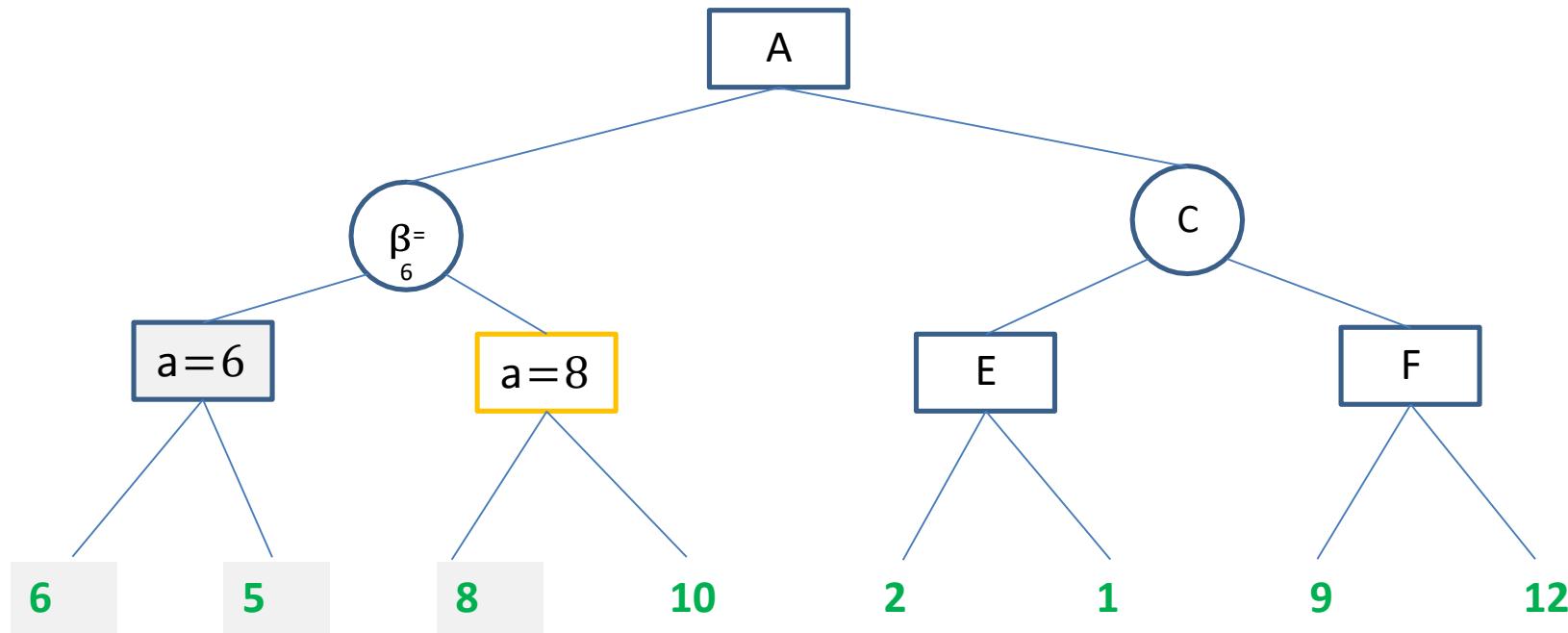
# Alpha Beta Pruning



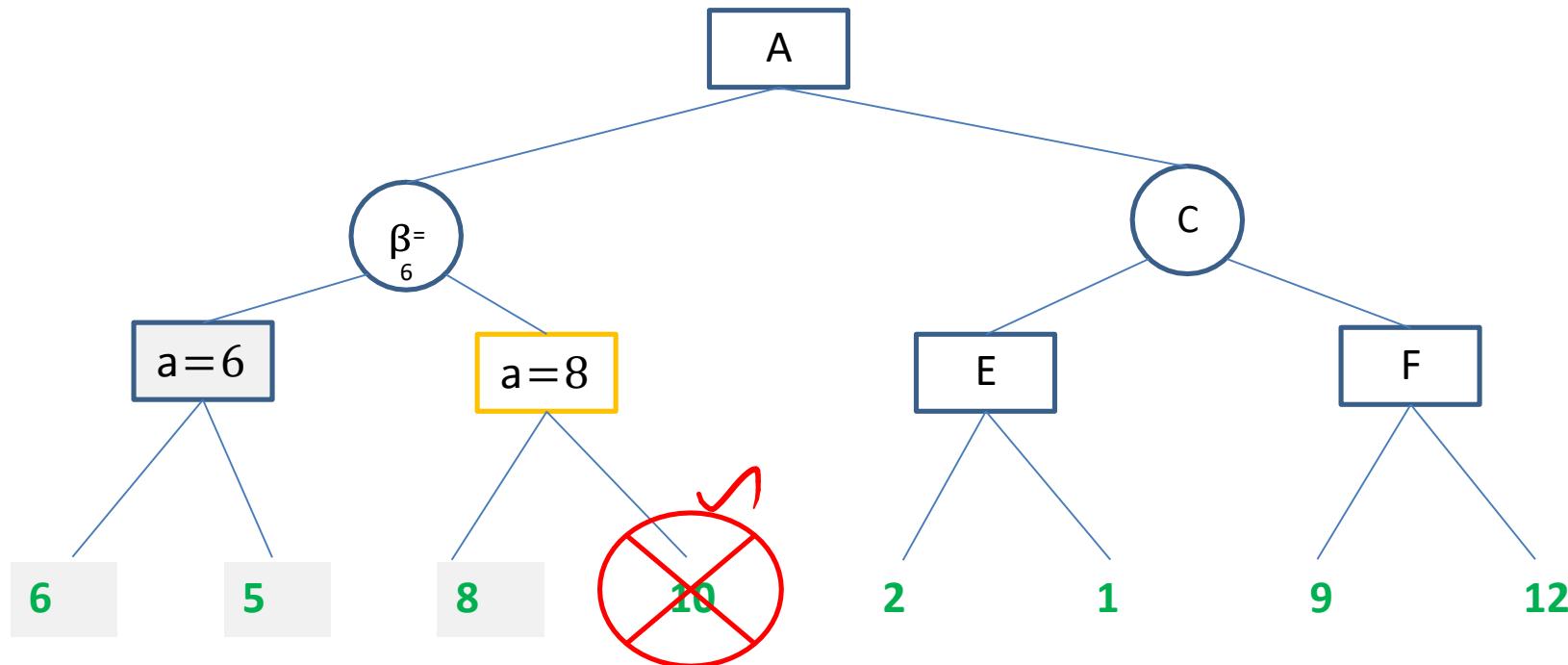
## Idea –Pruning



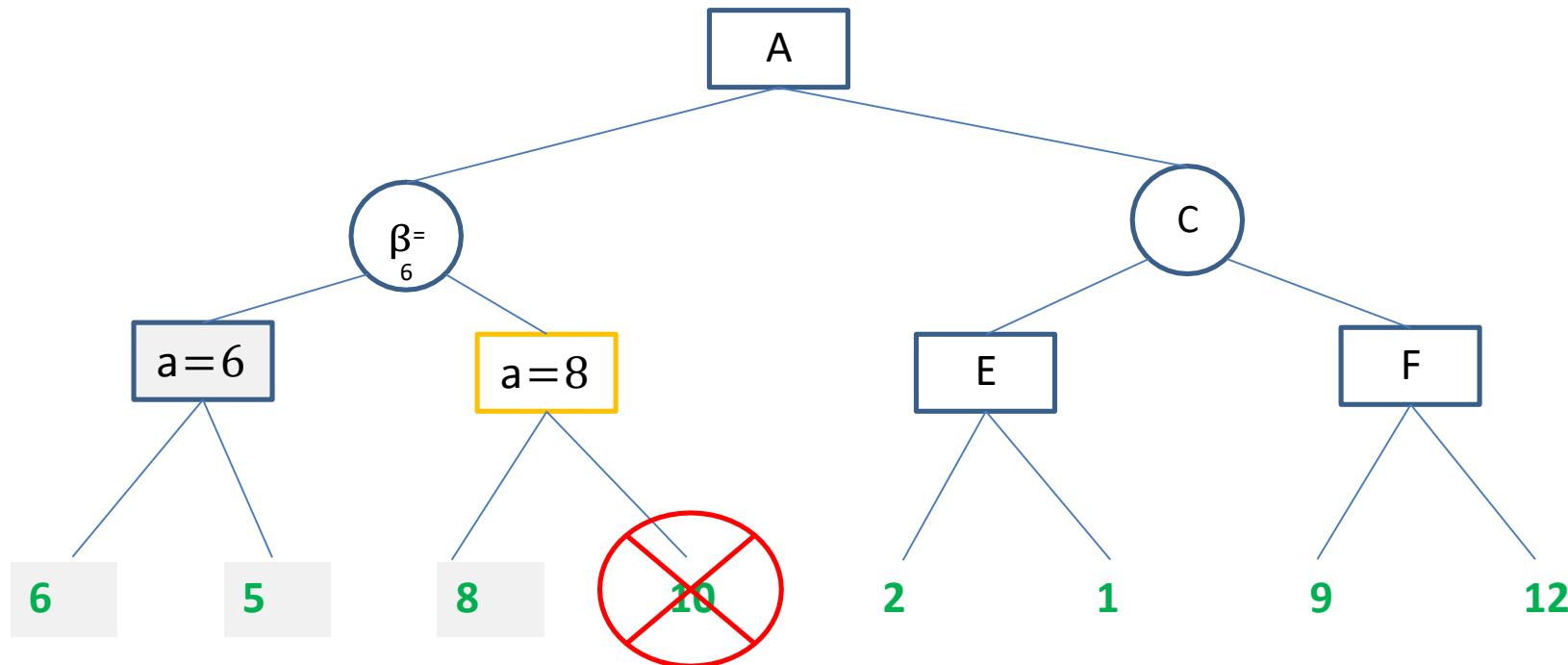
## Idea – Alpha Pruning



## Idea – Beta Pruning



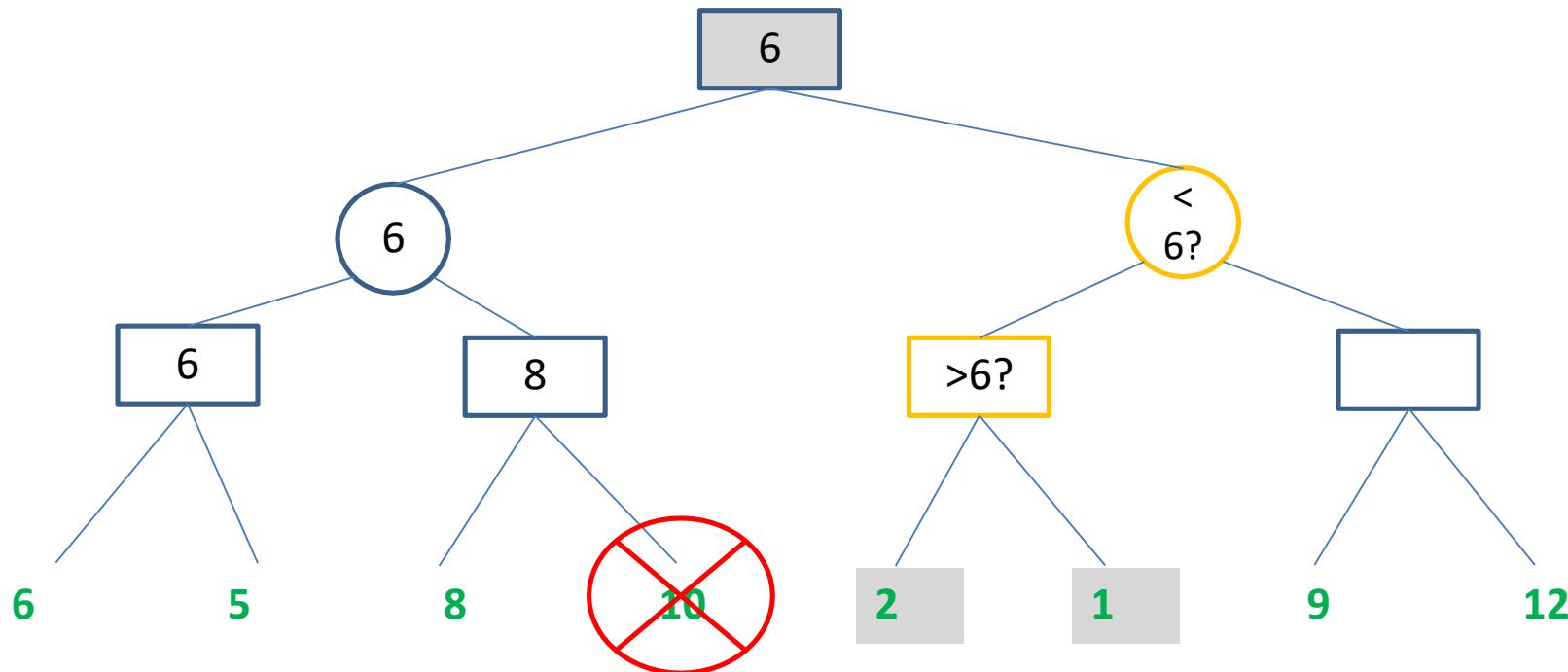
## Idea –Pruning



# Alpha Beta Pruning



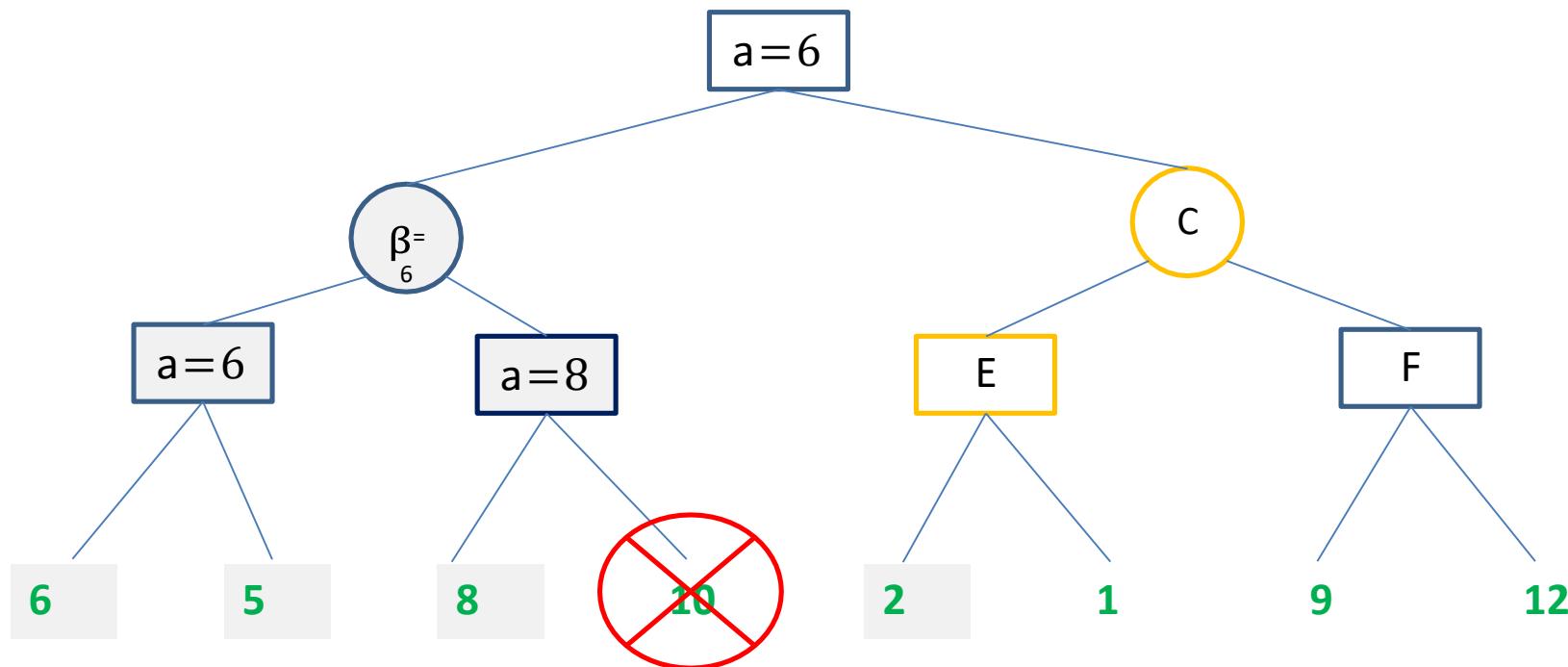
## Idea –Pruning



# Alpha Beta Pruning



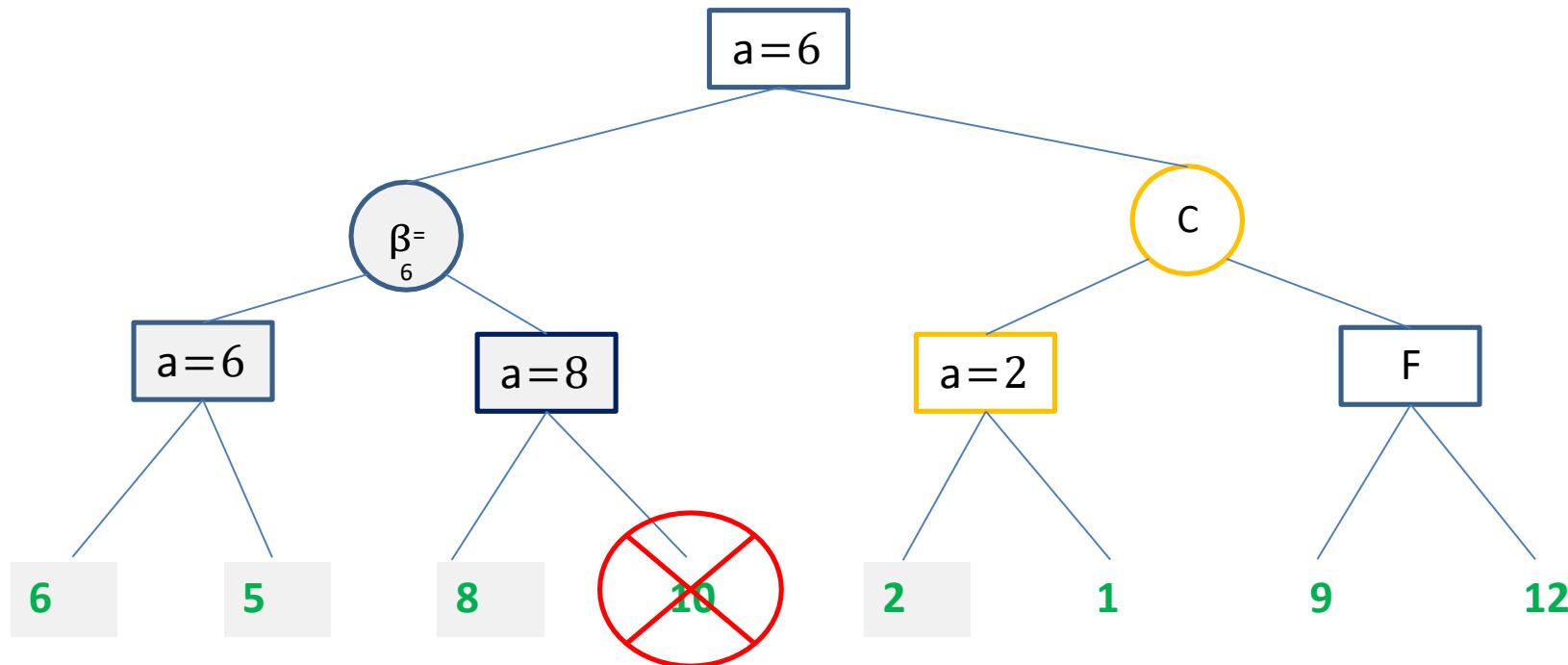
## Idea –Pruning



# Alpha Beta Pruning



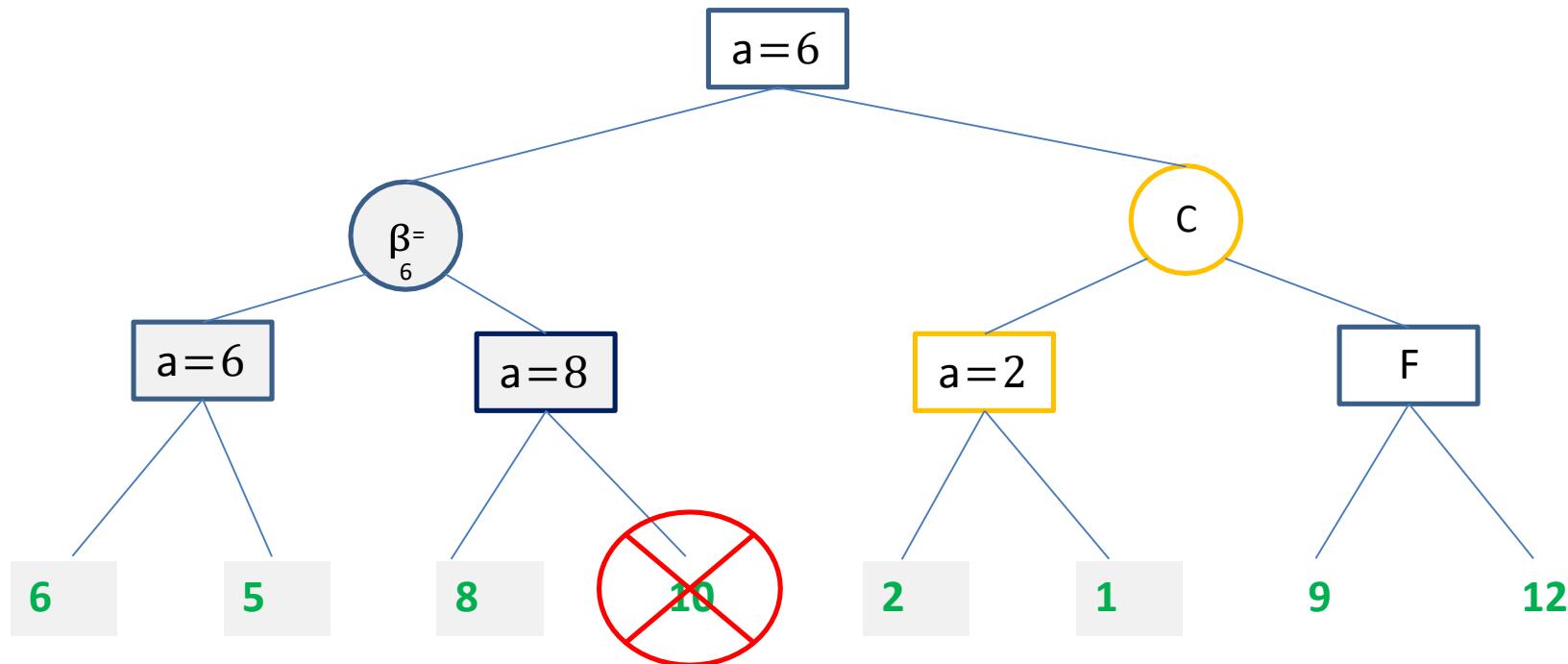
## Idea –Pruning



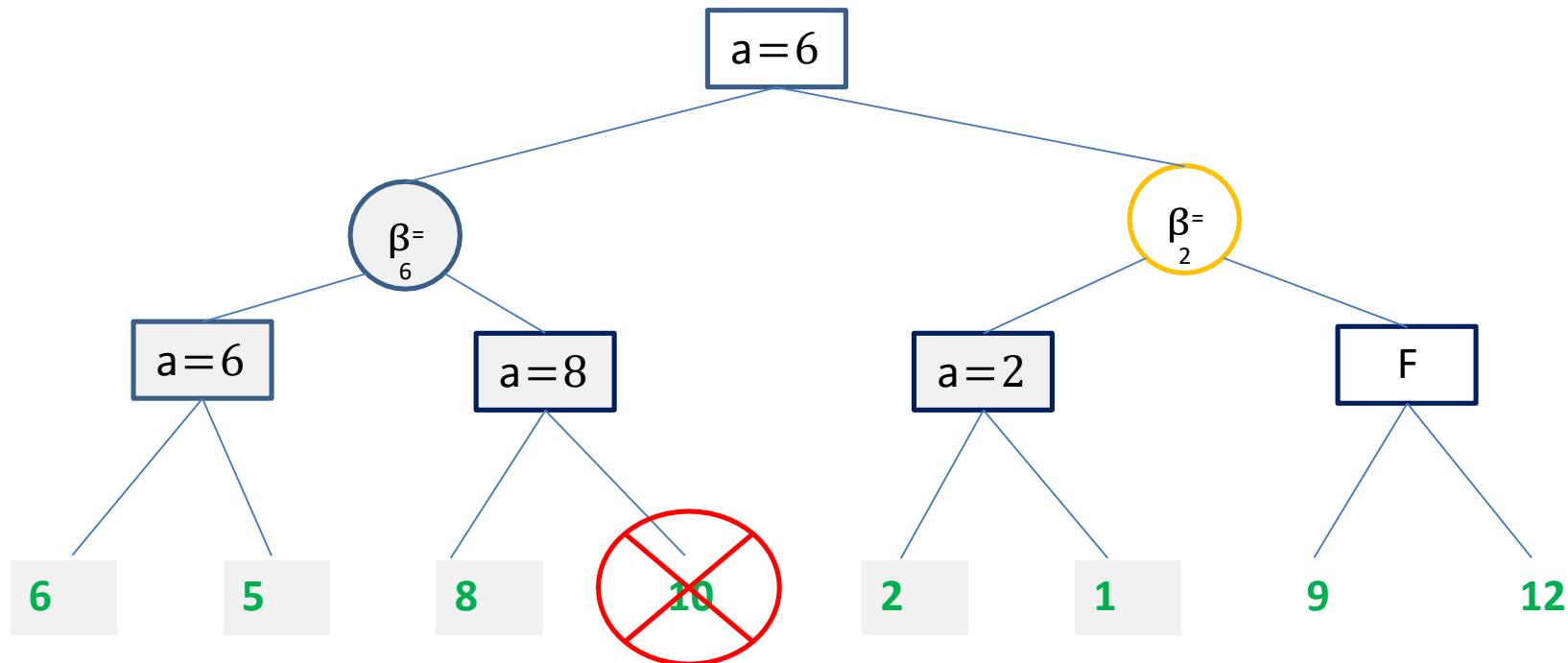
# Alpha Beta Pruning



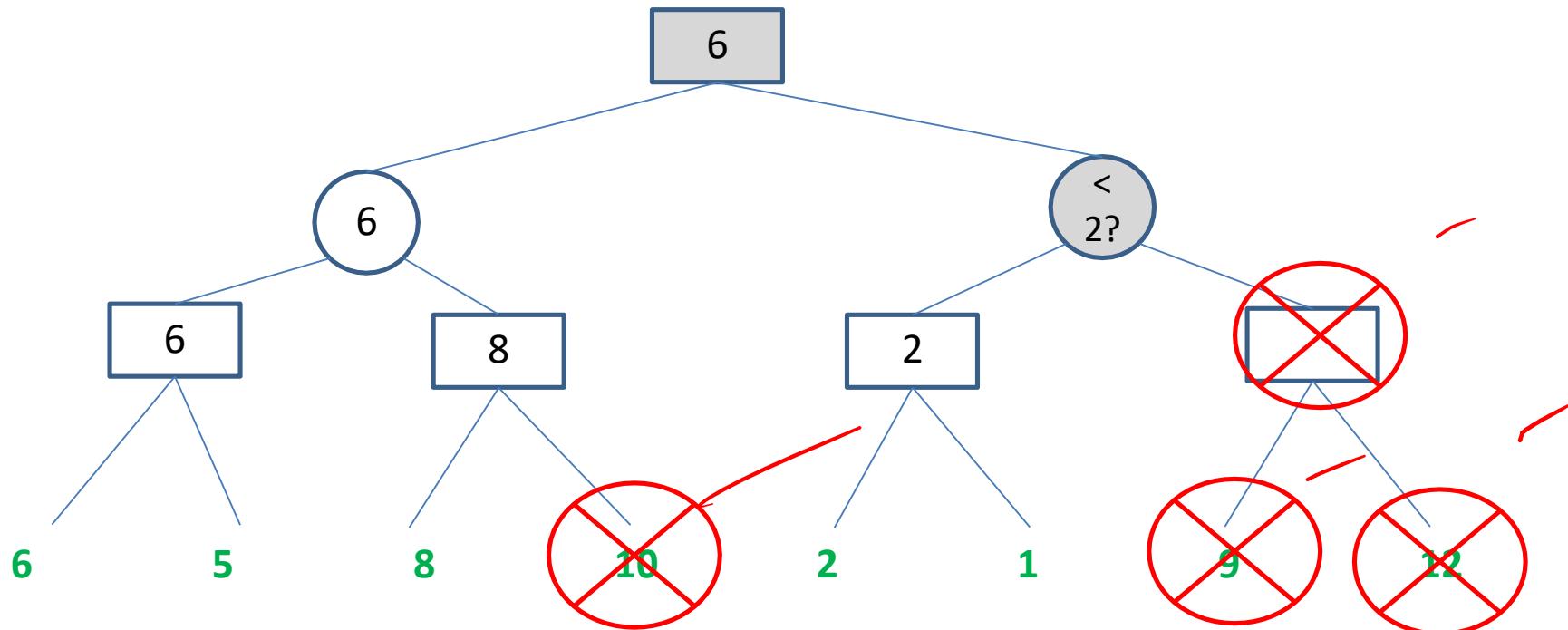
## Idea –Pruning



## Idea –Pruning



## Idea – Alpha Pruning



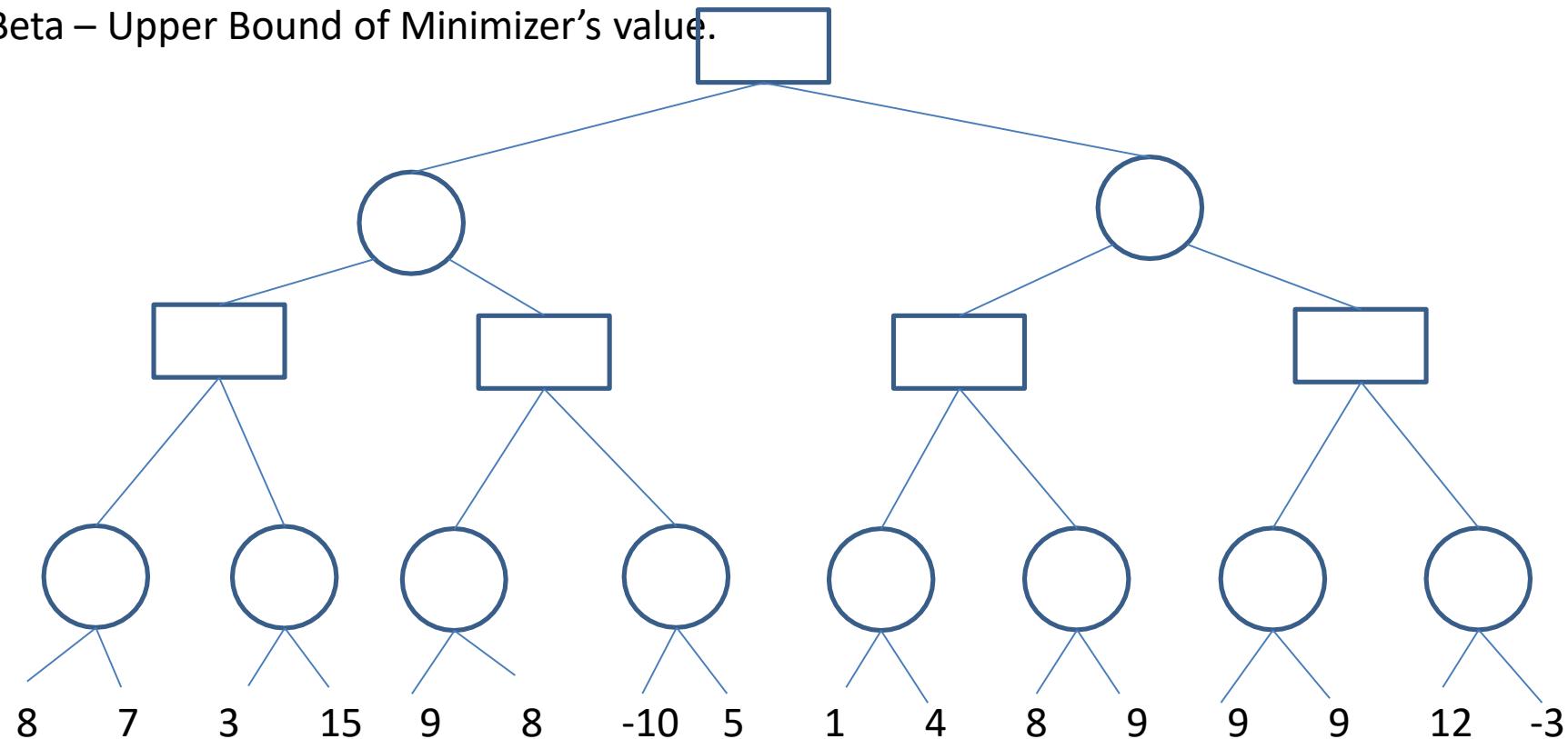
Alpha – Lower bound of Maximizer's value. Perceived value that Maximizer hopes to against a competitive Minimizer

## Alpha – beta Pruning – Example -4

Do for practice.

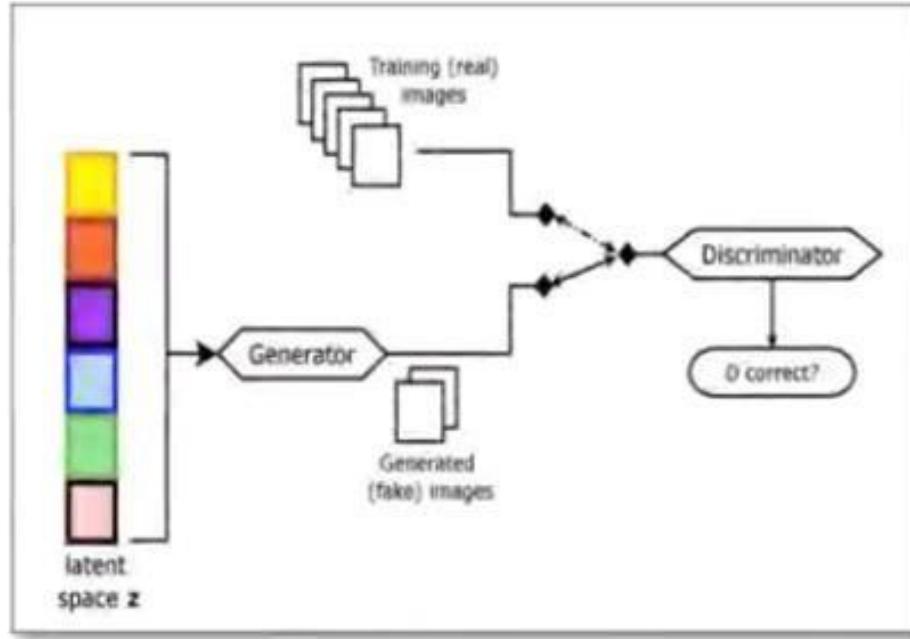
Alpha – Lower bound of Maximizer's value. Perceived value that Maximizer hopes to get with a competitive Minimizer

Beta – Upper Bound of Minimizer's value.



# Game Playing (Interesting Case Studies)

# Games in Image Processing



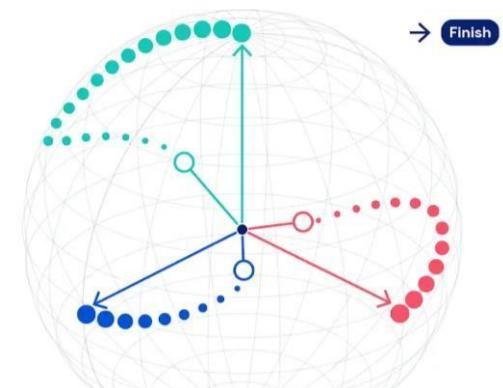
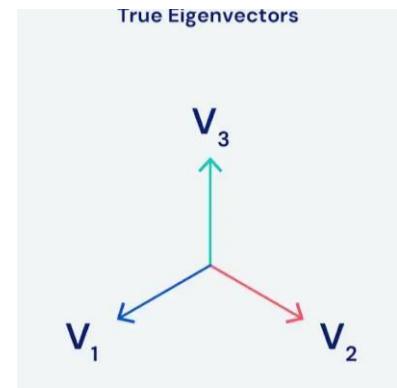
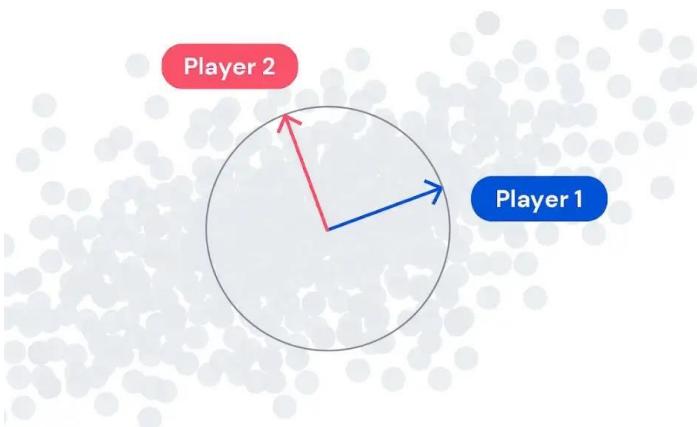
Source Credit:

[2019 - Analyzing and Improving the Image Quality of StyleGAN](#)

[Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, Timo Aila](#)

<https://thispersondoesnotexist.com/>

# Games in Feature Engineering



Source Credit:

<https://deepmind.com/blog/article/EigenGame>

2021 - EigenGame: PCA as a Nash Equilibrium , Ian Gemp, Brian McWilliams, Claire Vernade, Thore Graepel

# Games in Feature Engineering

$$\text{Utility}(v_i | v_{j < i}) = \boxed{\text{Var}(v_i)} - \sum_{j < i} \boxed{\text{Align}}(v_i, v_j)$$


Source Credit:

<https://deepmind.com/blog/article/EigenGame>

2021 - EigenGame: PCA as a Nash Equilibrium , Ian Gemp, Brian McWilliams, Claire Vernade, Thore Graepel

**Max**

## Gaming – Sample Question

Two **Robots A and B** competes to leave the maze through either of exits: **E1** and **E2**, as shown in the diagram. At each time step, each Robot move to an adjacent **free** square. Robots are not allowed to enter squares that other robots are moving into. The same exit cannot be used by both robots at once, but either robot may use either exit. A poisonous gas is left behind when a robot moves. No robot may enter the poisonous square for the duration of the gas's 1-time-step presence (ie., If the square is left free for one game round, the poison evaporates and is no longer dangerous). The poisonous squares are represented as **X** in the diagram. For utility calculation consider the below assumptions.

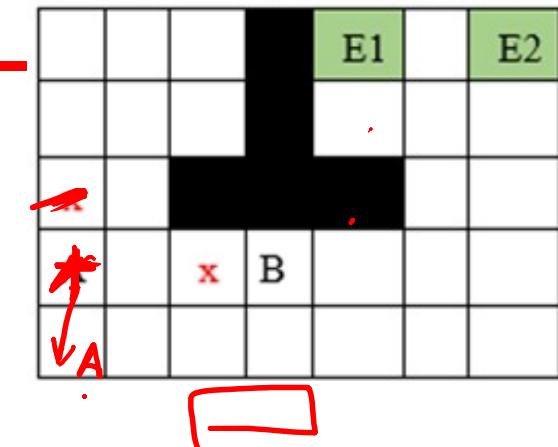
**B → MAX**

**4° free**

innovate

achieve

lead



$$H(n) = (\text{Max player's ability to win in the board position}) - (\text{Min player's ability to win in the board position})$$

Note: Player "Z's" ability to win is given by below:

$$\text{Utility} = \text{Minimum}(\text{manhattanDistance(Player "Z", Exit E1)}, \text{manhattanDistance(Player "Z", Exit E2)}) + \text{Penalty}$$

{  
Penalty = Number of unsafe cells (blockage+poisonous squares) with 4 degree of freedom(Up, Down, Right, Left) in the immediate neighborhood of the Player "Z's" position.

$$H(n) \rightarrow (\text{max}) - (\text{min})$$

$$\begin{aligned} \text{Agent} \\ \underline{A} & \rightarrow E_1 \rightarrow \text{dist}_1 \\ & \rightarrow E_2 \rightarrow \text{dist}_2 \end{aligned}$$

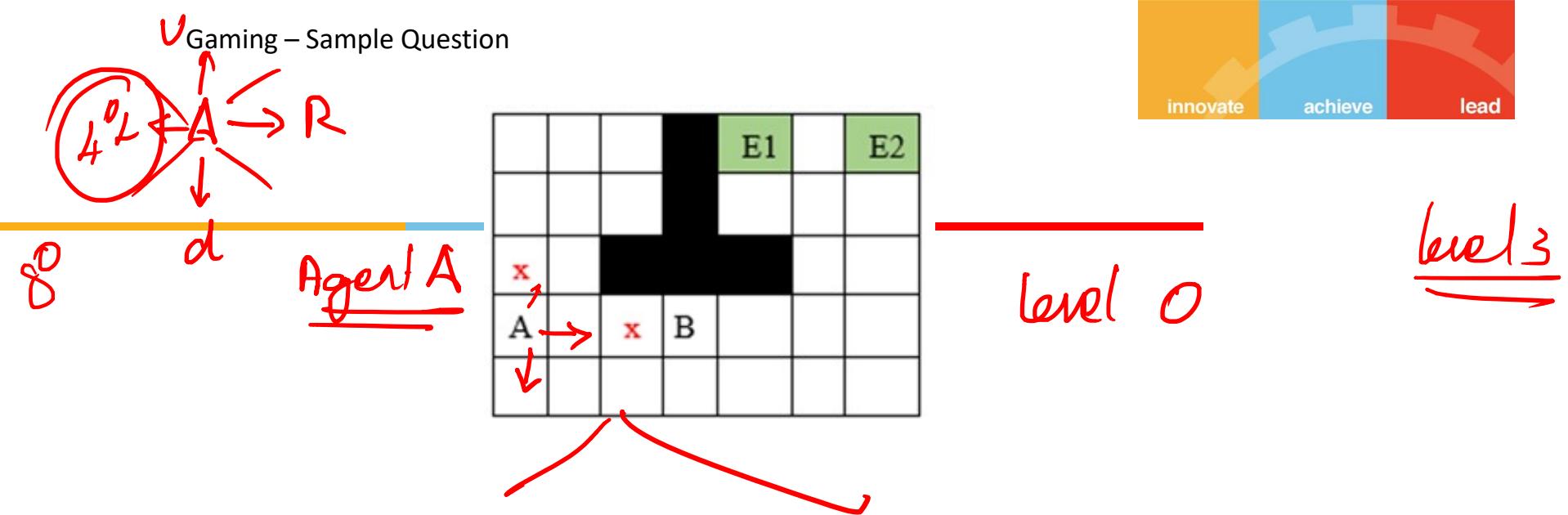
$$\min(\text{dist}_1, \text{dist}_2) + \text{Penalty}$$

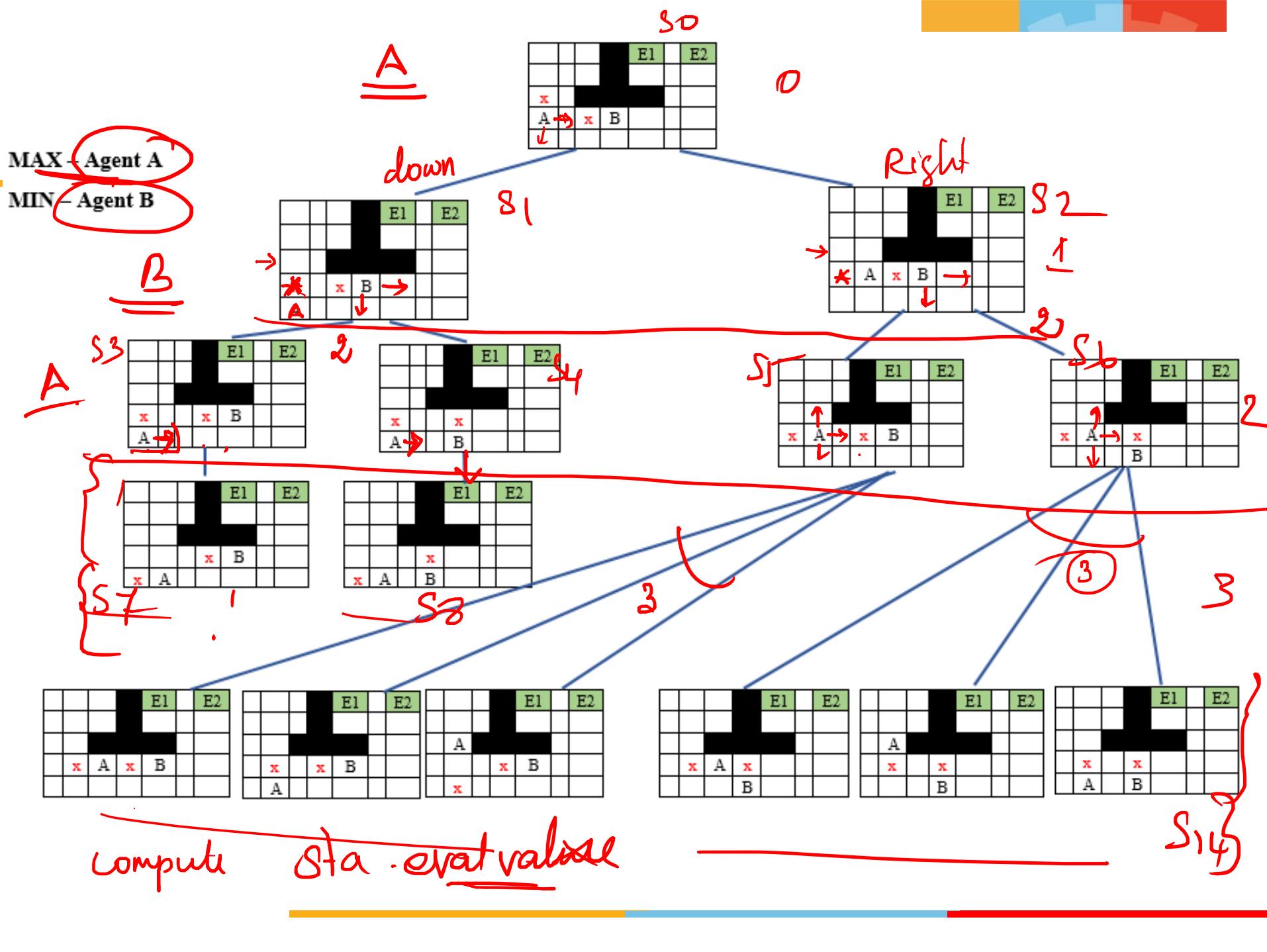
no of block

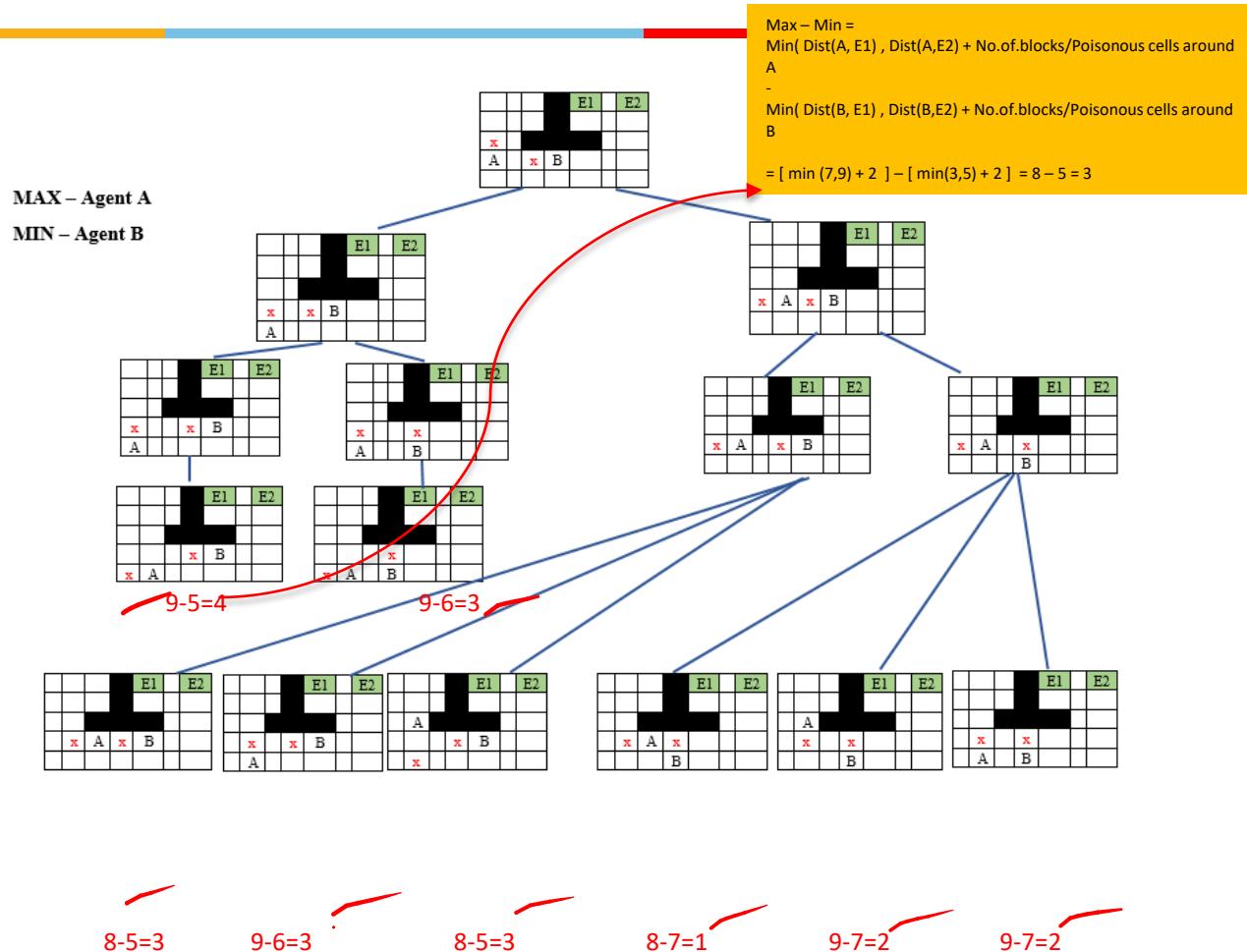
" " X 4° free

$$\begin{aligned} B \rightarrow E_1 \\ \rightarrow E_2 \end{aligned}$$

$$\underline{\min} + P$$







Max - Min =

$\text{Min}(\text{Dist}(A, E1), \text{Dist}(A, E2) + \text{No.of.blocks}/\text{Poisonous cells around A}$

$- \text{Min}(\text{Dist}(B, E1), \text{Dist}(B, E2) + \text{No.of.blocks}/\text{Poisonous cells around B})$

$$= [\min(7, 9) + 2] - [\min(3, 5) + 2] = 8 - 5 = 3$$

$\checkmark (\text{Max}) \rightarrow A$

$\min(\text{dist}_1, \text{dist}_2)$

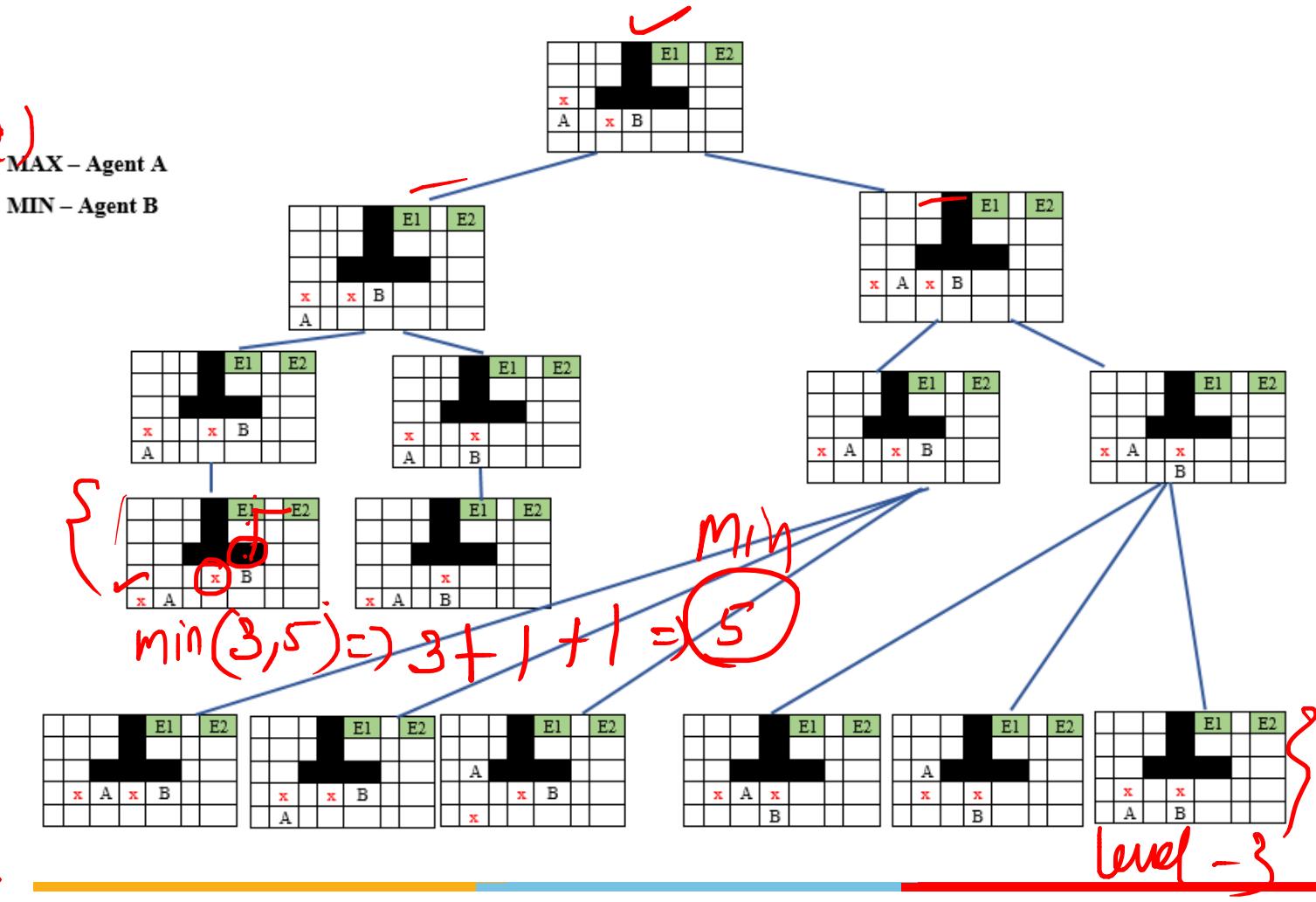
$7 + \text{Penalty}$

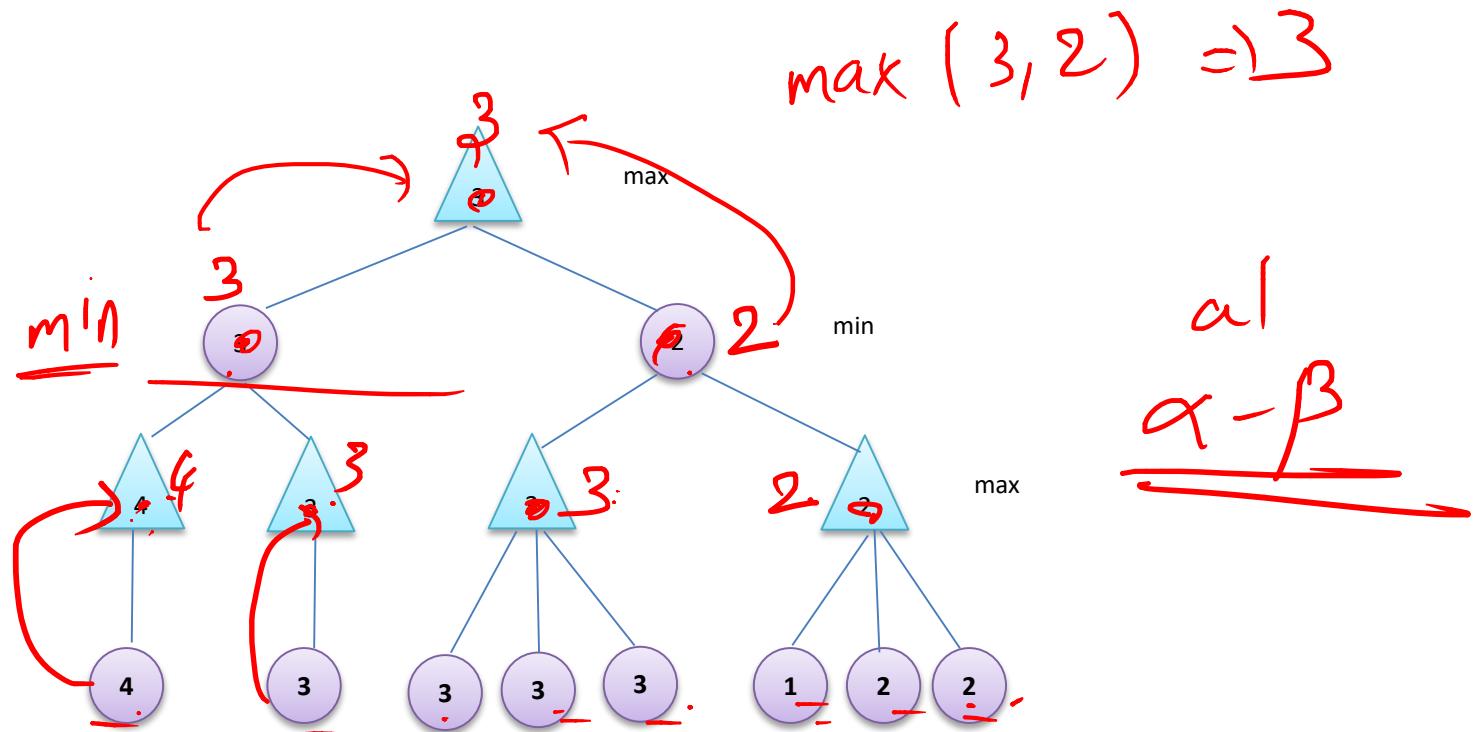
No of  
Poisonous  
Cells

$7 + 1 + 1$

$\Rightarrow 9$

$$\underline{\underline{9 - 5 = 4}}$$





---

**Required Reading: AIMA - Chapter # 4.1, #4.2, #5.1**

**Thank You for all your Attention**

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

DSE CLZG557

M4 : Knowledge Representation using Logics

Indumathi V  
Guest Faculty

BITS - CSIS

**BITS** Pilani  
Pilani Campus

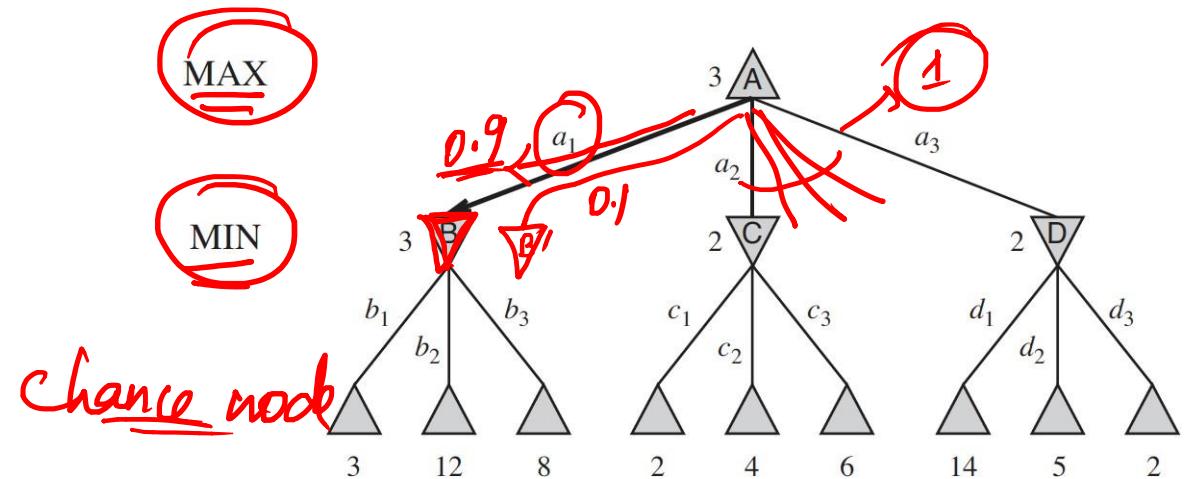
# Course Plan

- M1 Introduction to AI
- M2 Problem Solving Agent using Search
- M3 Game Playing → Adv Search
- M4 Knowledge Representation using Logics
- M5 Probabilistic Representation and Reasoning
- M6 Reasoning over time
- M7 Ethics in AI

# Gaming (Imperfect Decisions)

# Computational Efficiency

How games can be designed to handle imperfect decisions in real-time?



# Computational Efficiency

Idea : Chance Node:

Holds the expected values that are computed as a sum of all outcomes weighted by their probability (of dice roll)

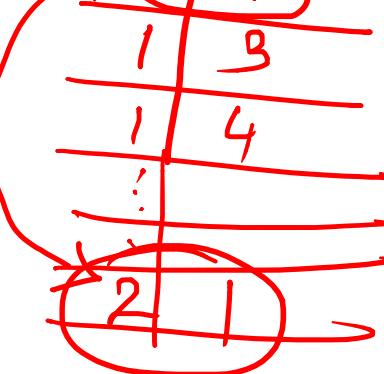
2 dice

36

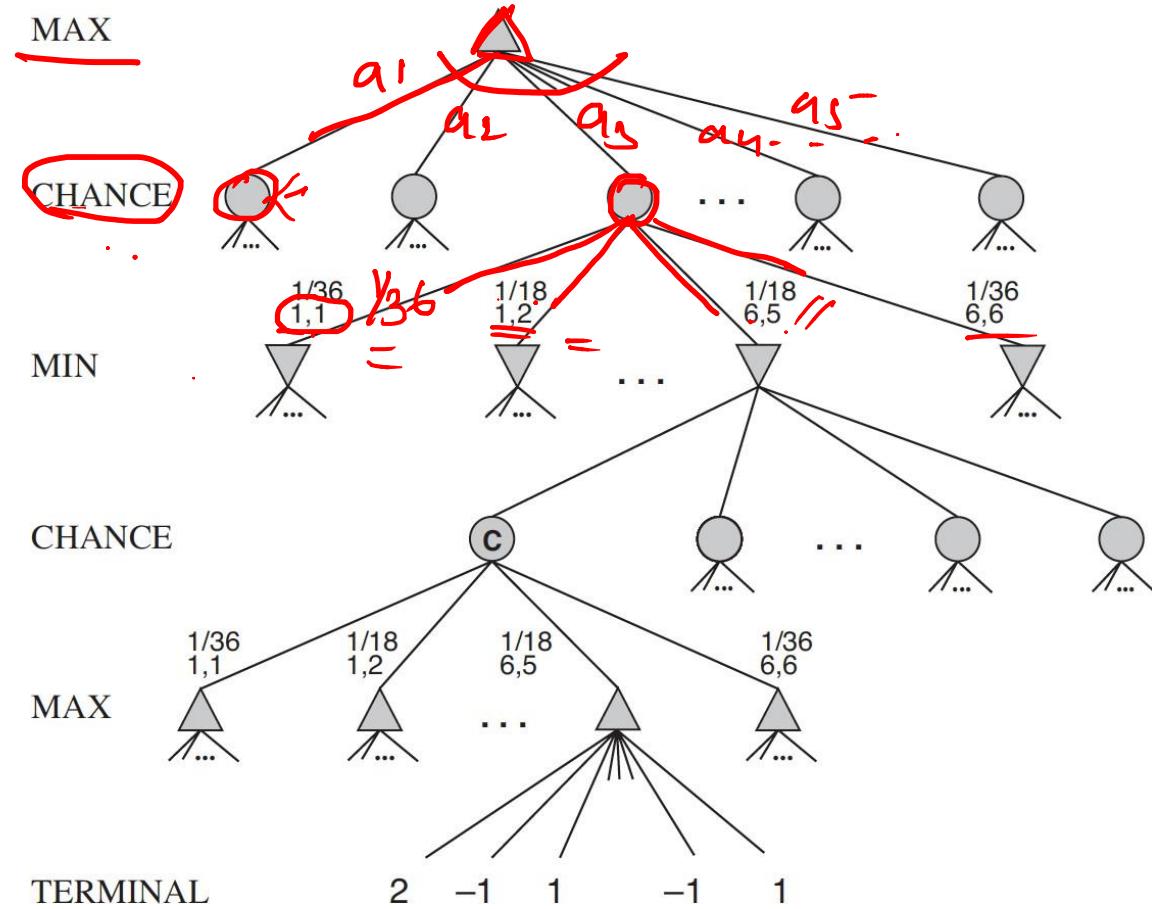
$d_1/d_2$

$\frac{1}{36}$

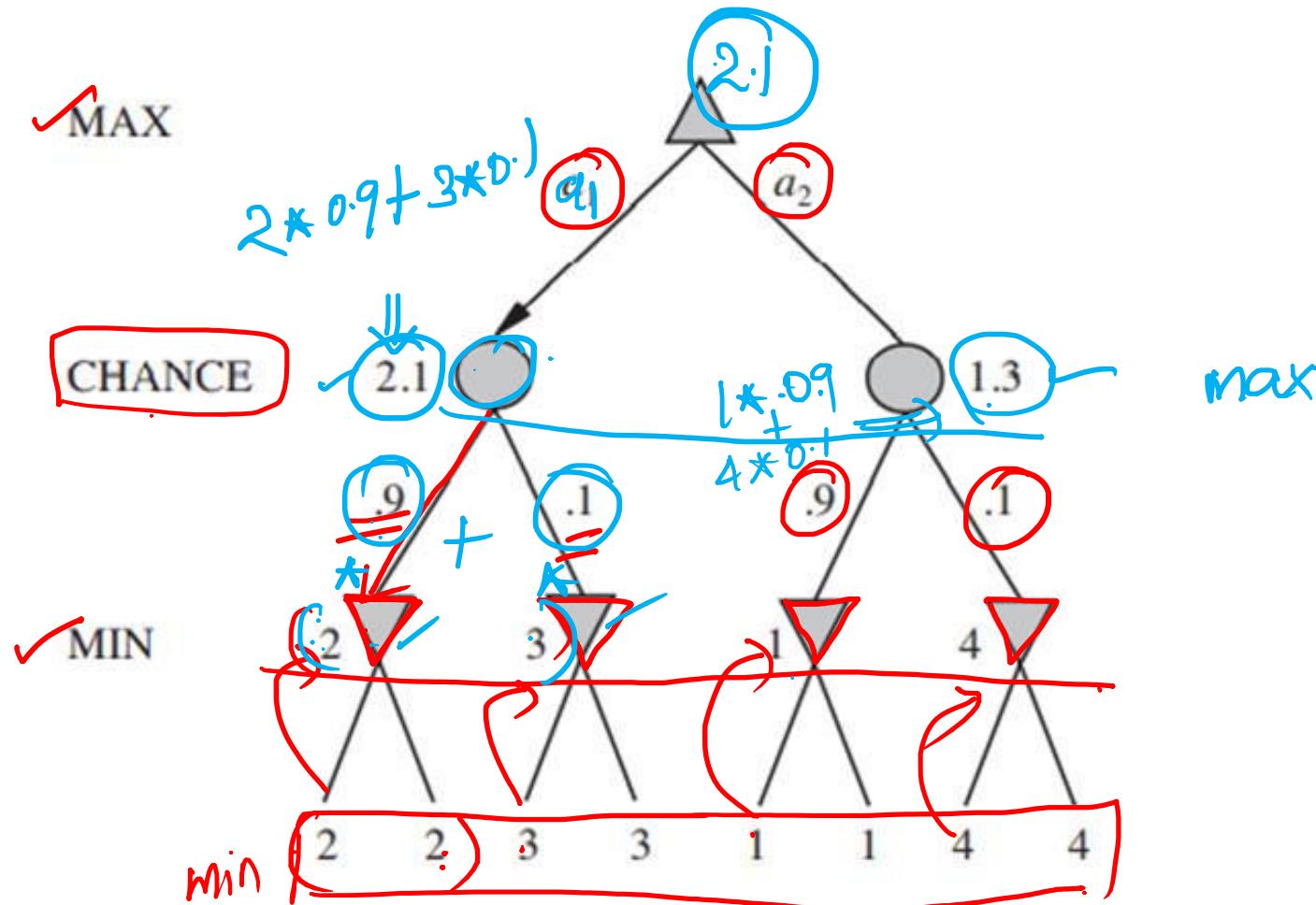
$(1,1) = \frac{1}{36}$



$2/36 = \frac{1}{18}$



# Expecti Mini Max Algorithm



$$4.4 * 0.3 + 3 * 0.7$$

$$= 4.4(0.3) + 3(0.7)$$

$$= 3.42$$

$$\min(4(0.8) + 6(0.2), 6)$$

$$= \min(4.4, 6)$$

$$= 4.4$$

0.3      0.7

b·4

~~b·4~~

~~max~~

$$4(0.6) + 10(0.4)$$

$$= 6.4$$

min

$$(4.4)$$

Min

0.8      0.2

(4)

$$(4 * 0.8 + b * 0.2)$$

b · max

min

min.

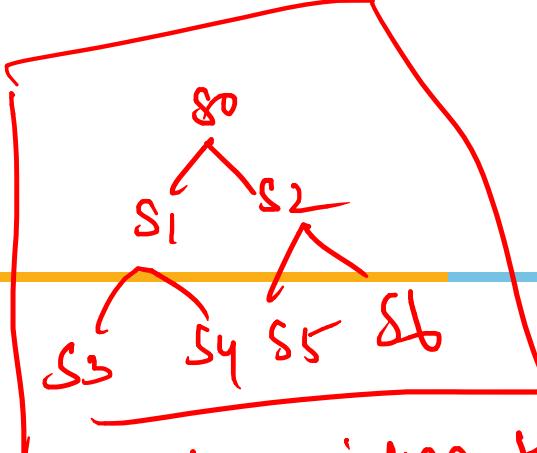
3

7

4

11

10

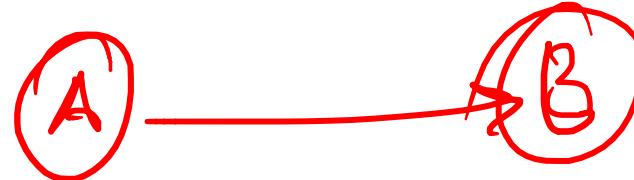
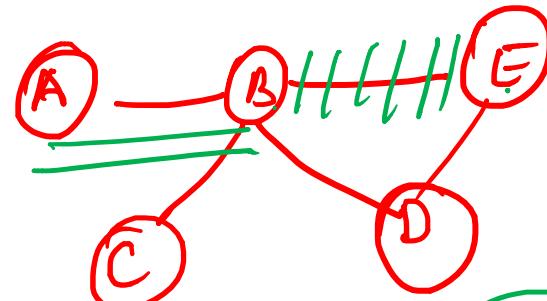


- ① store infmn = KB
- ② Future purpose.

ML

## Knowledge Representation Using Logics

A-B ||| E



Best path A to D  
ABD



Complex prob  $\rightarrow$  large dB

Reasoning

A-D  $\rightarrow$  ABED, ABD

# Learning Objective

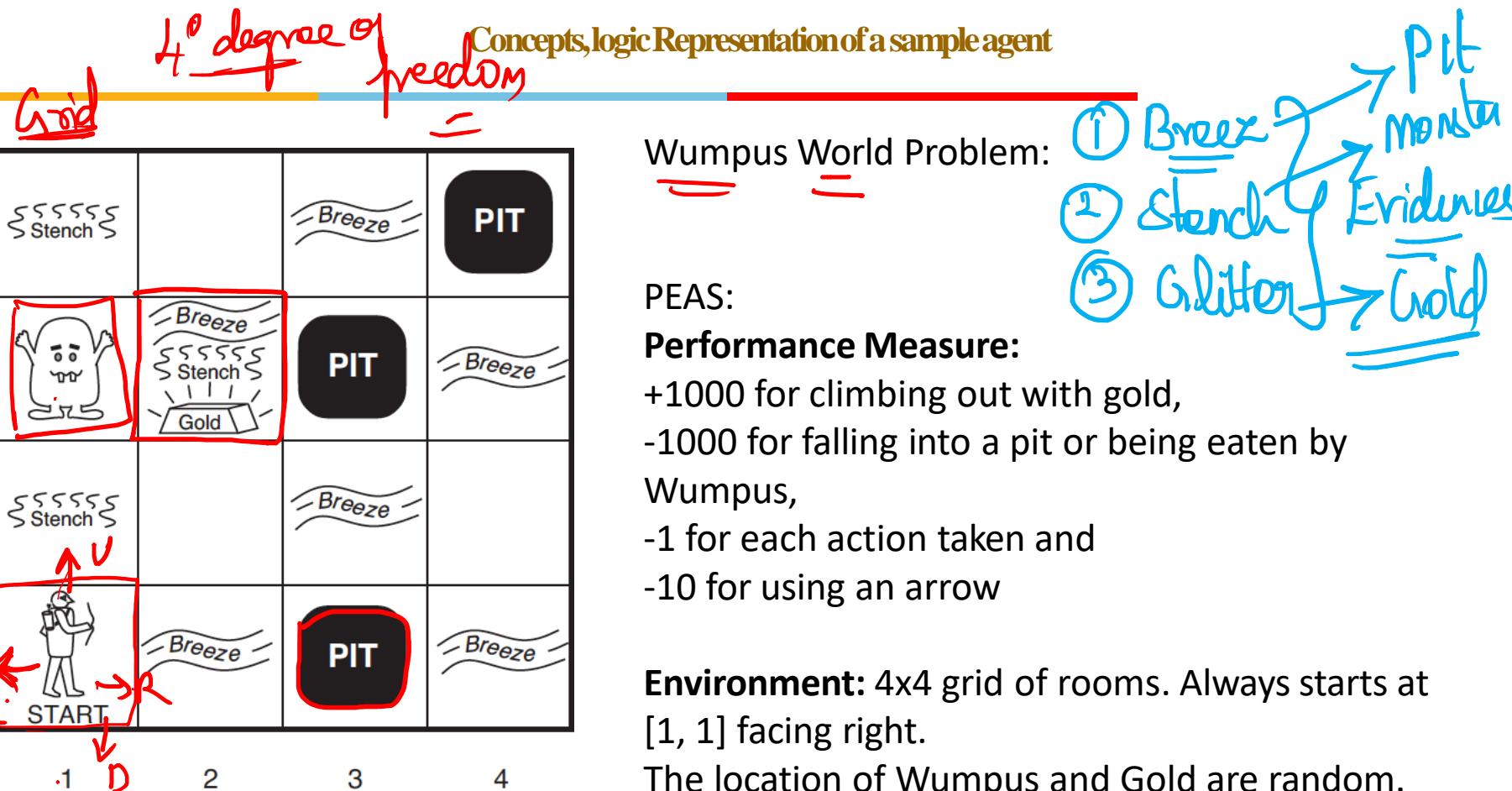
At the end of this class , students Should be able to:

1. Represent a given knowledge base into logic formulation
2. Infer facts from KB using Resolution
3. Infer facts from KB using Forward Chaining
4. Infer facts from KB using Backward Chaining



① semantic nets  
② Conceptual dependency  
Fuzzy

# Knowledge based Agent : Model & Represent



Agent Loc  $\rightarrow$  (1,1)  
Gold  $\rightarrow$  (2,3)

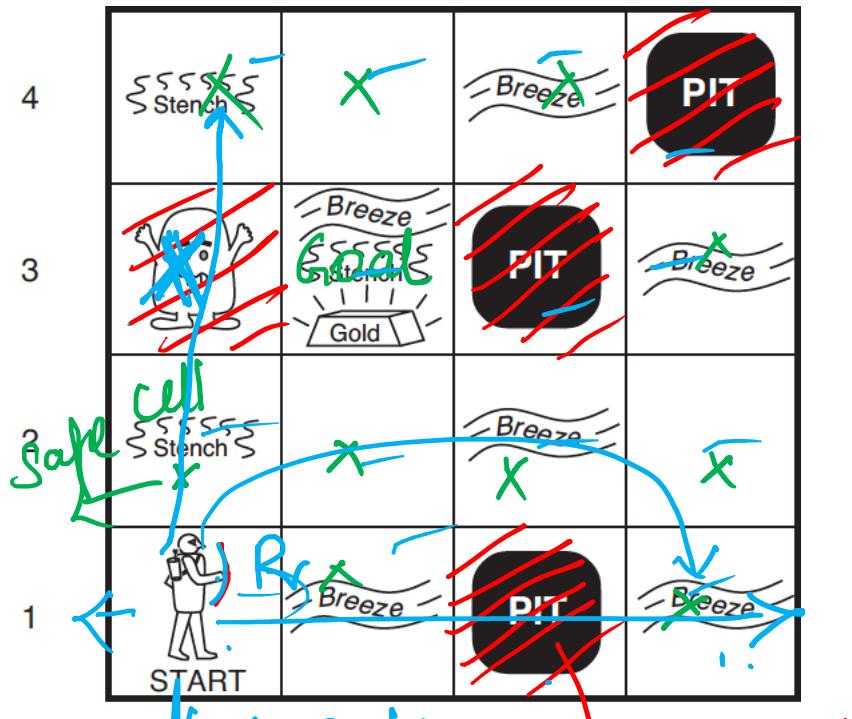
obstacles

$\checkmark \rightarrow$  PIT  
 $\checkmark \rightarrow$  Monster/Wumpus  $\rightarrow$  (1,3)  $\downarrow$

# Knowledge based Agent : Model & Represent



Concepts, logic Representation of a sample agent



Wumpus World Problem:

PEAS:

**Performance Measure:**

- +1000 for climbing out with gold,
- 1000 for falling into a pit or being eaten by Wumpus,
- 1 for each action taken and
- 10 for using an arrow

**Environment:** 4x4 grid of rooms. Always starts at [1, 1] facing right.

The location of Wumpus and Gold are random. Agent dies if entered a pit or live Wumpus.

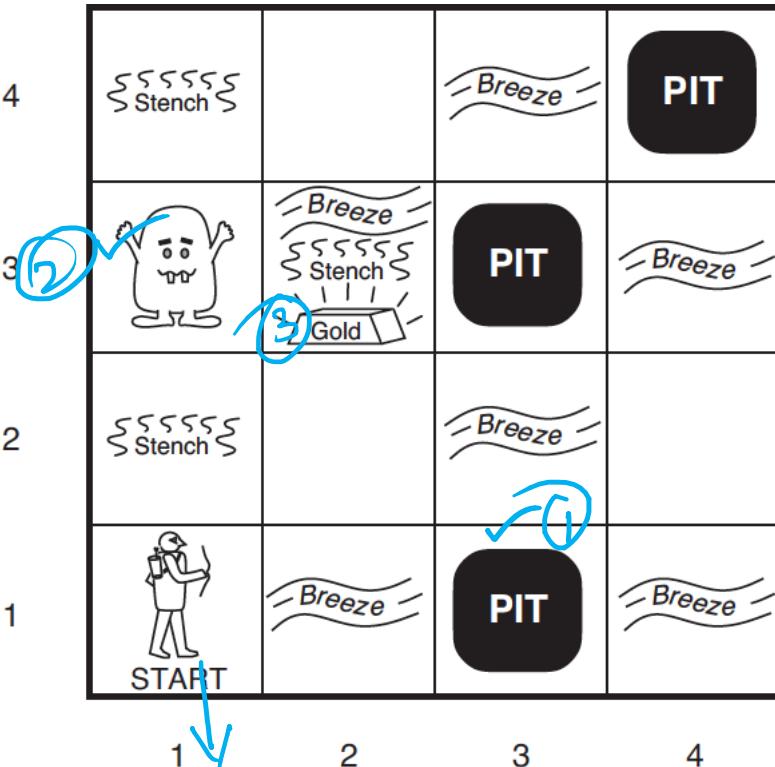
$\sqrt{A} \rightarrow \text{gold}$       ~~Not changed~~

T - M / W  
→ PITS

# Knowledge based Agent : Model & Represent



Concepts, logic Representation of a sample agent



Wumpus World Problem:

PEAS:

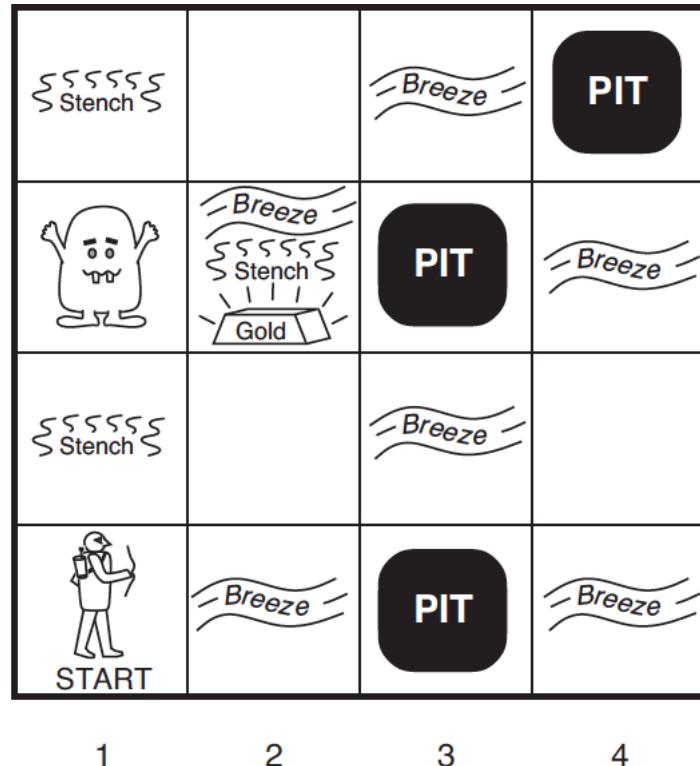
**Actuators –**

- Forward, ML
- TurnLeft by 90, MR
- TurnRight by 90, MU, MD
- Grab – pick gold if present,
- Shoot – fire an arrow, it either hits a wall or kills wumpus. Agent has only one arrow.
- Climb – Used to climb out of cave, only from [1, 1] ..

# Knowledge based Agent : Model & Represent



Concepts, logic Representation of a sample agent

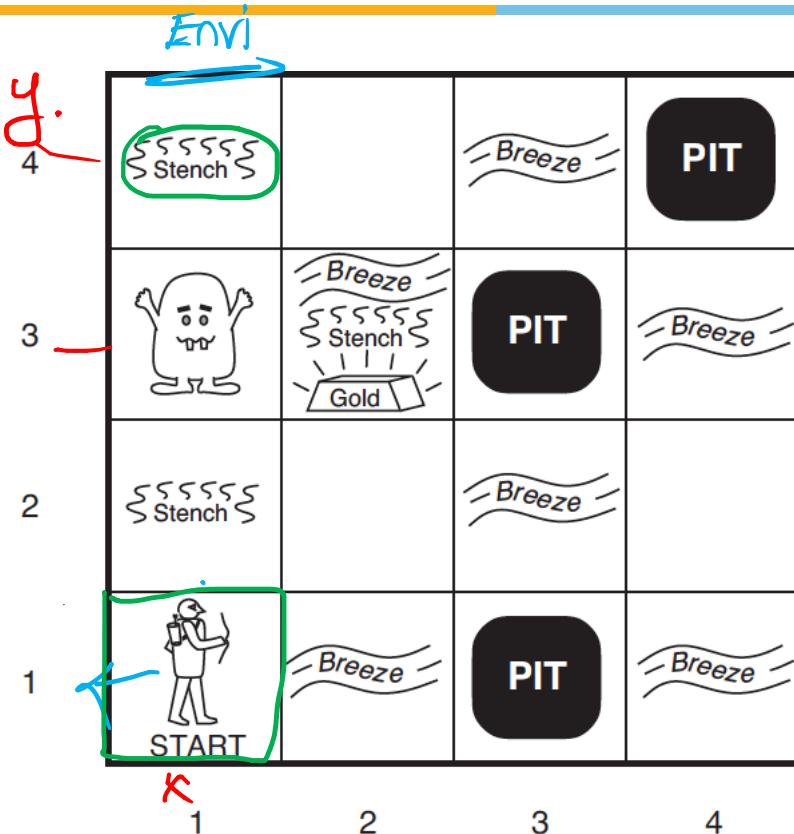


Why do we need Factored representation

- To reason about steps
- To learn new knowledge about the environment
- To adapt to changes to the existing knowledge
- Accept new tasks in the form of explicit goals
- To overcome partial observability of environment

# **Knowledge based Agent : Model & Represent**

## Concepts, logic Representation of a sample agent



## Wumpus World Problem:

PEAS: 

~~Sensors~~: The agent has five sensors

 Stench: In all adjacent (but not diagonal) squares of Wumpus

 Breeze: In all adjacent (but not diagonal) squares of a pit

 Glitter: In the square where gold is

**Bump:** If agent walks into a wall

Scream: When Wumpus is killed, it can be perceived everywhere

$S_{x,y}$  → val  
 $S_{1,4}$  → True ✓  
 $S_{1,1}$  → False

## Percept Format:

T, F [Stench?, Breeze?, Glitter?, Bump?, Scream?]  
E.g., [Stench, Breeze, None, None, None]  
(S), B, G, B, S, ] n, p, ~~q~~ Proposition Val

# Knowledge based Agent : Model & Represent

$S_1, B_1, G_1, B_3$

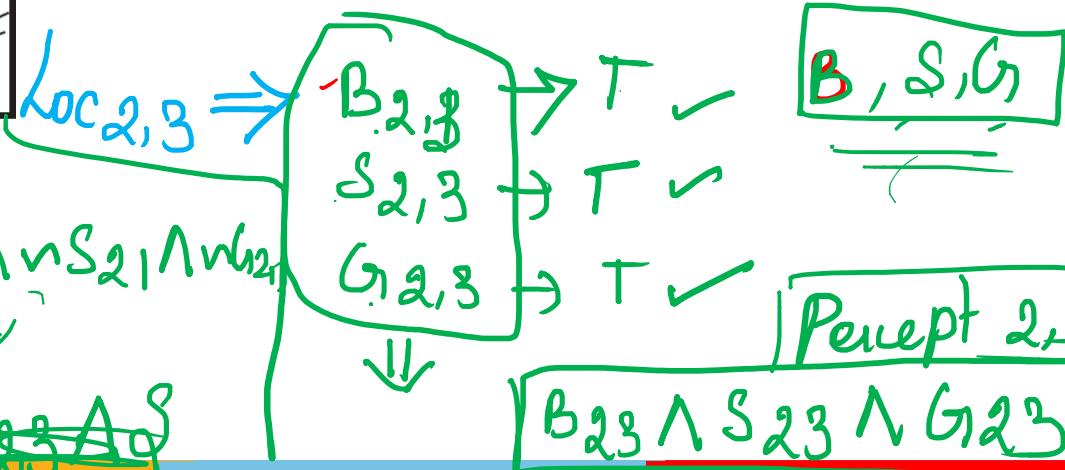
Scent  $\rightarrow$  ignore the mornk

$B_{x,y} \Rightarrow T$  or  $F$

$B_{2,1} \Rightarrow T \Rightarrow B_{2,1}$

$B_{2,2} \Rightarrow F \Rightarrow \neg B_{2,1}$

No  
False  
 $\neg B_{2,1}$

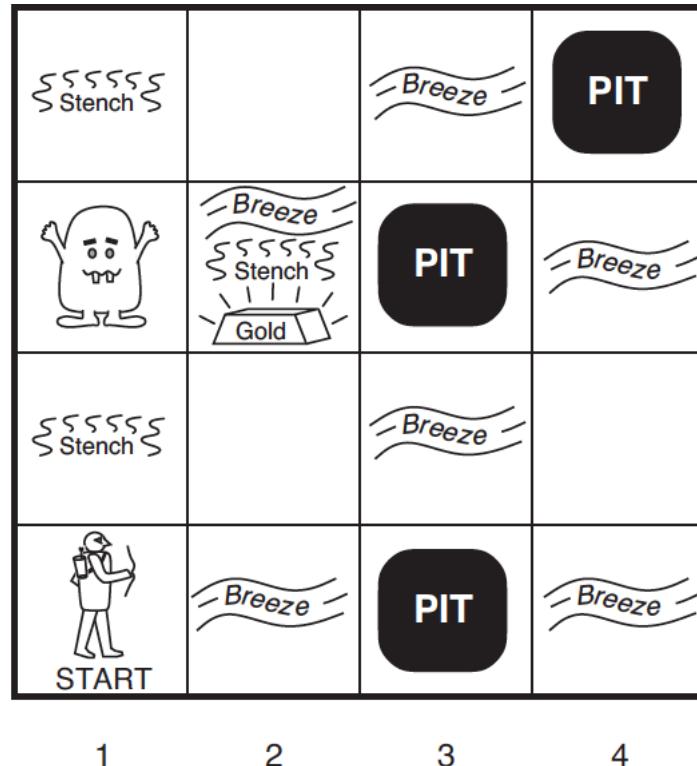


$Loc_{21} \Rightarrow B_{21} \Rightarrow T$

$\neg S_{21} \Rightarrow F \Rightarrow$   
 $\neg G_{21} \Rightarrow F$

~~B<sub>23</sub> A<sub>23</sub>~~

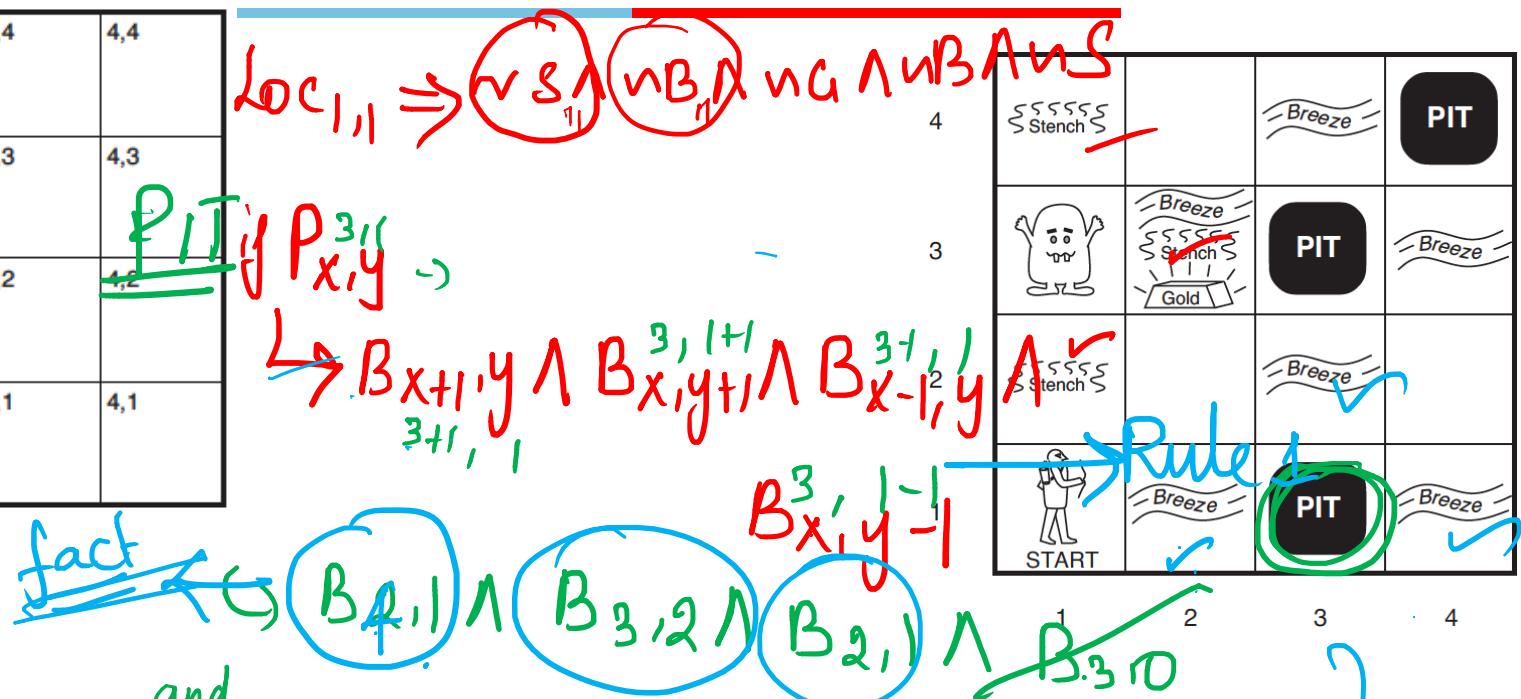
# Knowledge based Agent : Model & Represent



Percept 1: [None, None, None, None, None]

Action: Forward

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	<del>4,2</del>
OK			PIT
1,1	2,1	3,1	4,1



Homey

$W_{x,y} \rightarrow S_{x+1,y} \wedge S_{x,y+1} \wedge S_{x-1,y} \wedge S_{x,y-1} \rightarrow \text{Rule 2}$

$B_{x,y} \rightarrow P_{x+1,y} \vee P_{x,y+1} \vee P_{x-1,y} \vee P_{x,y-1} \rightarrow \text{Rule 3}$

$S_{x,y} \rightarrow W_{x+1,y} \vee W_{x,y+1} \vee W_{x-1,y} \vee W_{x,y-1} \rightarrow \text{Rule 4}$

Percept Format:  
 [Stench?, Breeze?, Glitter?, Bump?, Scream?]

Percept 1: [None, None, None, None, None]

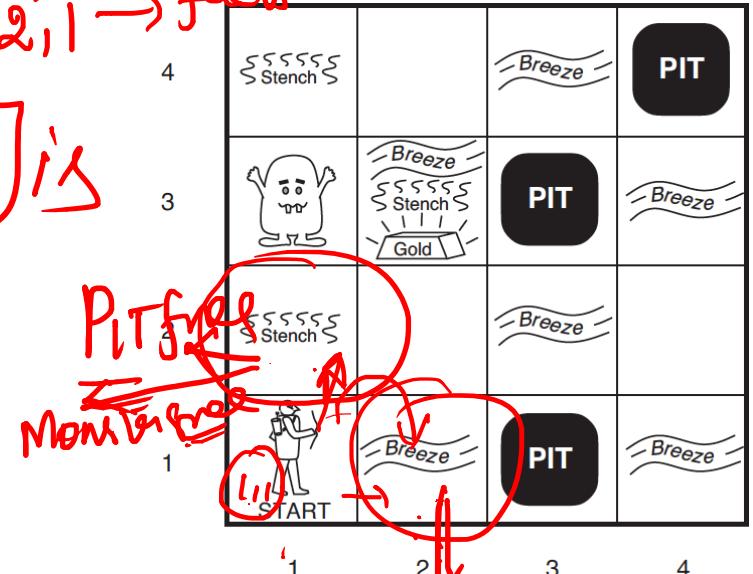
Action: Forward

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK	OK	OK	OK

A → ✓  
B → ✓

Loc 2,1

$\neg B_{1,1} \rightarrow \neg P_{1,2} \wedge \neg P_{2,1}$  facts → KB  
 $\neg S_{1,1} \rightarrow \neg W_{1,2} \wedge \neg W_{2,1}$  facts  
 Loc [1,2] & [2,1] is Safe cell



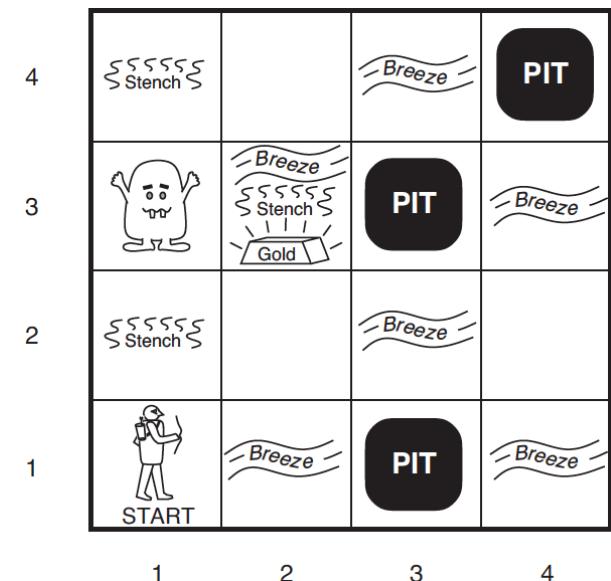
$\neg P_{1,1}$  →  $\neg P_{1,2}$   
 $\neg M_{1,1}$  →  $\neg M_{1,2}$   
 $\neg P_{2,2}$  →  $\neg P_{2,1}$   
 $\neg M_{2,2}$  →  $\neg M_{2,1}$   
 PIT free →  
 Monster free →  
 (1,1) → PLT free →  
 Monster free

Percept Format:  
 [Stench?, Breeze?, Glitter?, Bump?, Scream?]

Percept 1: [None, None, None, None, None]

Action: Forward

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A	2,1	3,1	4,1
OK	OK		



Percept Format:  
 [Stench?, Breeze?, Glitter?, Bump?, Scream?]

## Agents based on Propositional logic, TT-Entail for inference from truth table

# Syntax

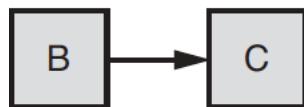
## Semantics

## Model

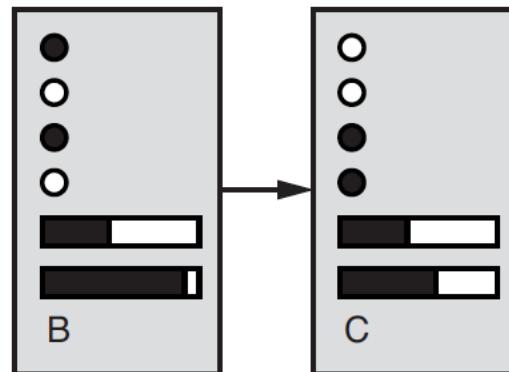
# Logic

## Propositional Logic

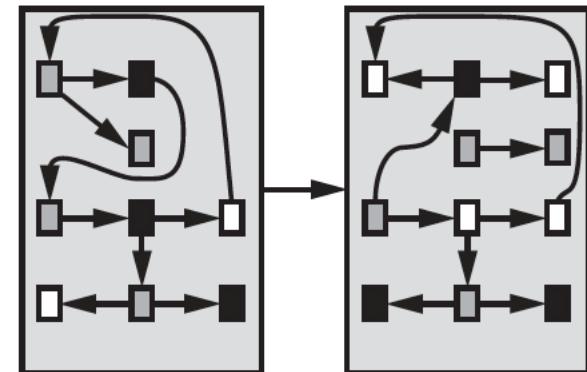
# Predicate Logic



### (a) Atomic



(b) Factored



### (b) Structured

# Search Strategies

## Propositional Logic

# First Order Logic

## Agents based on Propositional logic, TT-Entail for inference from truth table

A simple representation language for building knowledge-based agents

**Proposition Symbol** - A symbol that stands for a proposition.

E.g.,  $W_{1,3}$  - “Wumpus in [1,3]” is a proposition and  $W_{1,3}$  is the symbol

Proposition can be true or false

**Atomic** :  $W_{1,3}$

**Conjuncts** :  $W_{1,3} \wedge P_{3,1}$

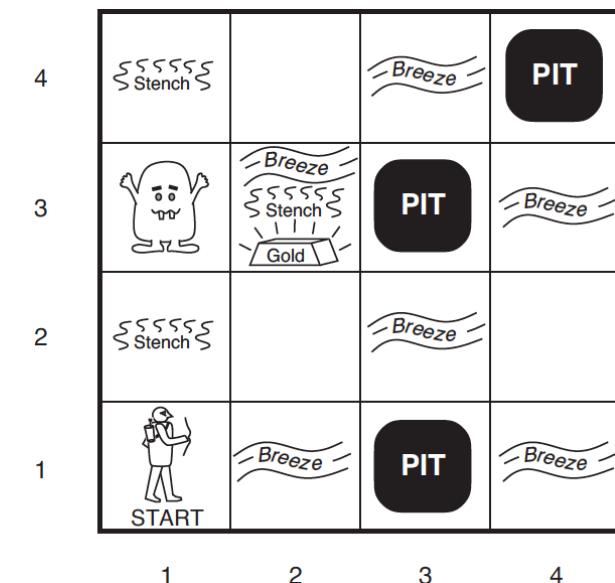
**Disjuncts** :  $W_{1,3} \vee P_{3,1}$

**Implications** :

$(W_{1,3} \wedge P_{3,1}) \Rightarrow \neg W_{2,2}$

**Biconditional** :  $W_{1,3} \Leftrightarrow \neg W_{2,2}$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	A OK	2,1 OK	3,1



Agents based on Propositional logic, TT-Entail for inference from truth table

Tie break in search:

$\neg$ ,  $\wedge$ ,  $\vee$ ,  $\Rightarrow$ ,  $\Leftrightarrow$

$(\neg A) \wedge B$  has precedence over  $\neg(A \wedge B)$

$P$	$Q$	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

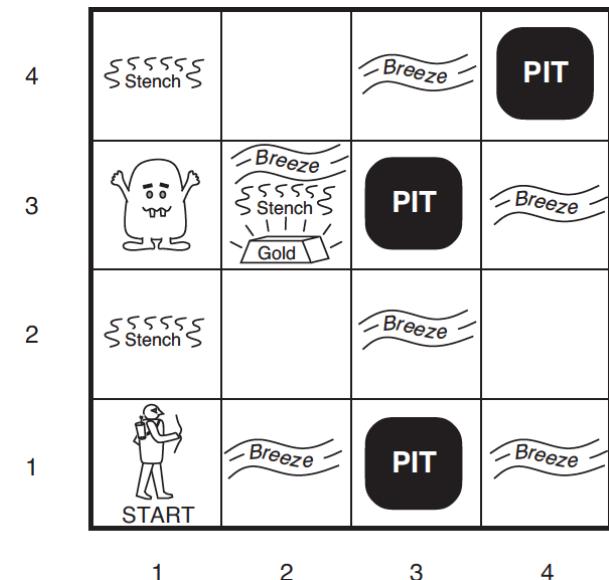
Percept 3: [Stench, None, None, None, None]



Action: Move to [2, 2]

Remembers (2,2) as possible PIT and no Stench.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1 A	2,1	3,1	4,1
OK	OK		



# Percept 3: [Stench, None, None, None, None]

Action: Move to [2, 2]

Remembers (2,2) as possible PIT and no Stench.

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	A	2,1	3,1
OK	OK		4,1

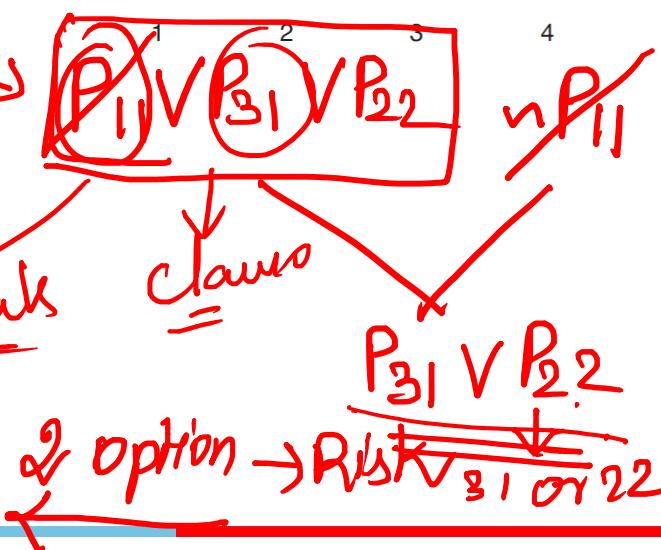
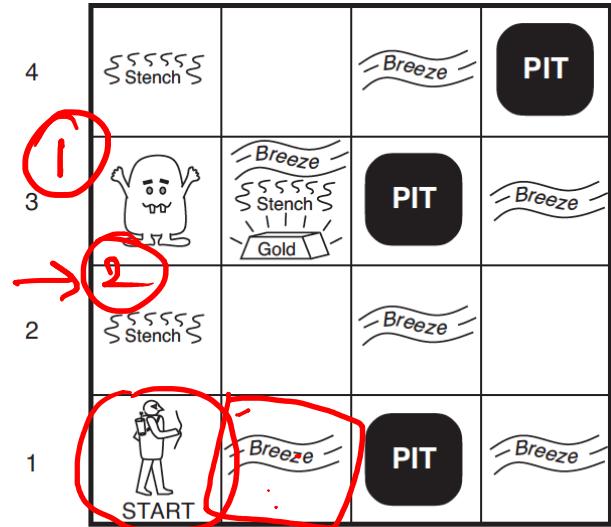
$$B_{2,1} \wedge \neg S_{2,1}$$

$$B_{2,1} \rightarrow P_{11} \vee P_{31} \vee P_{22} \rightarrow 1$$

$$\neg S_{2,1} \rightarrow \neg W_{11} \wedge W_{31} \wedge \neg W_{22} \rightarrow 2$$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	P?	2,2	3,2
OK			4,2
1,1	A	2,1	B
V	OK	V	OK

1,4	2,4	3,4	4,4
1,3	W!	2,3	3,3
1,2	S	2,2	3,2
OK		OK	4,2
1,1	V	2,1	B
OK		V	OK



# Percept 3: [Stench, None, None, None, None]

Action: Move to [2, 2]

Remembers (2,2) as possible PIT and no Stench.



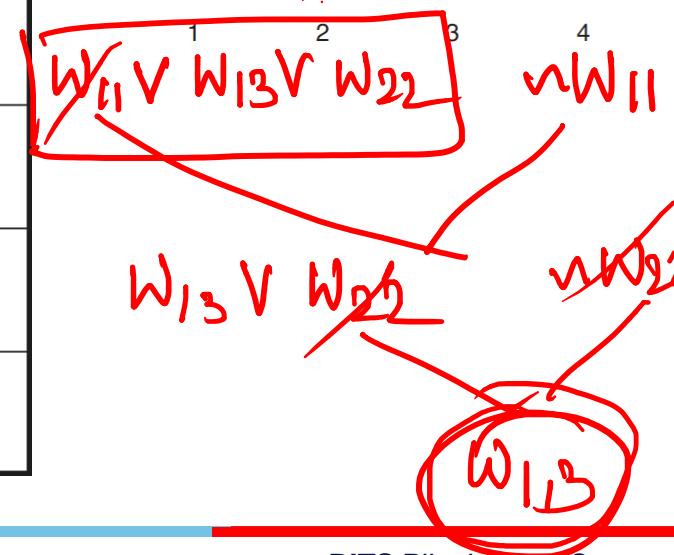
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	A	2,1	3,1
OK	OK		

$$\begin{aligned}
 L_{1,2} \Rightarrow S_{1,2} \wedge \neg B_{1,2} &\rightarrow \text{fact.} \\
 S_{1,2} \rightarrow W_{1,1} \vee W_{1,3} \vee W_{2,2} &\rightarrow \text{fact.} \\
 \neg B_{1,2} \rightarrow \neg P_{1,1} \wedge P_{1,3} \wedge P_{2,2} & \\
 \neg P_{1,1} \wedge P_{1,3} \wedge P_{2,2}
 \end{aligned}$$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	P?	3,2
OK			4,2
1,1	V	A	2,1
OK	B	OK	3,1
	P?		4,1

1,4	2,4	3,4	4,4
1,3	W?		3,3
1,2	2,2	S	3,2
OK		OK	4,2
1,1	V	A	2,1
OK	B	OK	3,1
	P?		4,1

4			
3			
2			
1			
START			





# Artificial & Computational Intelligence

**DSECSZG557**

## M4 : Knowledge Representation using Logics

Indumathi V

Guest Faculty,  
BITS -WILP

**BITS** Pilani  
Pilani Campus

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

① PL resolution  
② inference

③ TT Entailment

④ Theorem proving Mech  
DPLL Alg

# Representation by Propositional Logic

$$Q : \neg P_{1,2} \rightarrow \text{True}$$

For each  $[x, y]$  location

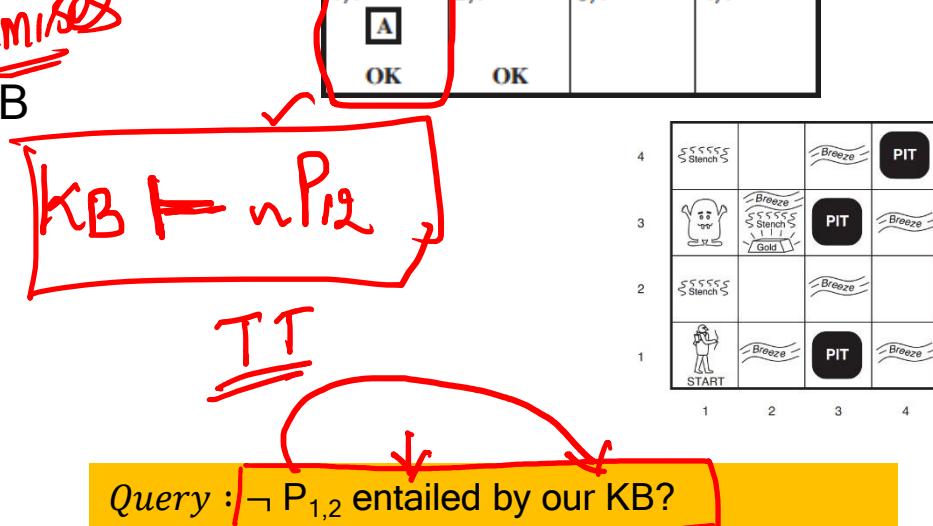
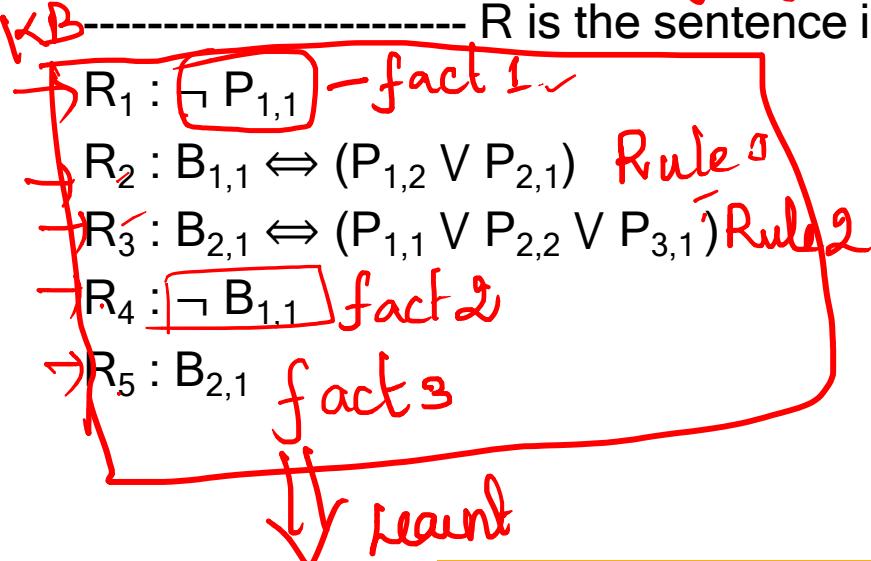
$P_{x,y}$  is true if there is a pit in  $[x, y]$

$W_{x,y}$  is true if there is a wumpus in  $[x, y]$

$B_{x,y}$  is true if agent perceives a breeze in  $[x, y]$

$S_{x,y}$  is true if agent perceives a stench in  $[x, y]$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1	2,1	3,1	4,1
OK	OK	OK	OK



# Representation by Propositional Logic

For each  $[x, y]$  location

$P_{x,y}$  is true if there is a pit in  $[x, y]$

$W_{x,y}$  is true if there is a wumpus in  $[x, y]$

$B_{x,y}$  is true if agent perceives a breeze in  $[x, y]$

$S_{x,y}$  is true if agent perceives a stench in  $[x, y]$

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
OK			
1,1	2,1	3,1	4,1
A	OK	OK	

R is the sentence in KB

- $R_1 : \neg P_{1,1}$
- $R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
- $R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
- $R_4 : \neg B_{1,1}$
- $R_5 : B_{2,1}$

KB  
↑ unique Lits



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

Query :  $\neg P_{1,2}$  entailed by our KB?

# TT – Entails Inference – Example

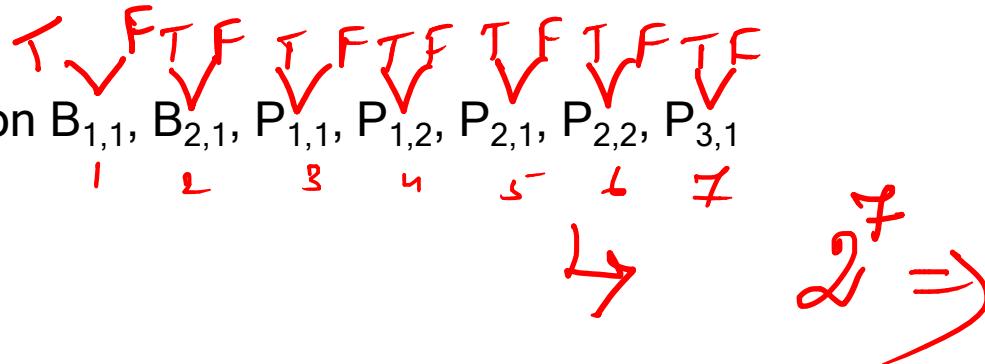


Agents based on Propositional logic, TT-Entail for inference from truth table

Q:  $\neg P_{1,2}$  entailed by our KB?

Way – 1 :

1. Get sufficient information  $B_{1,1}, B_{2,1}, P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, P_{3,1}$



$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$KB$
false	true	true	true	true	false	false						
false	true	true	false	true	false	false						
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	false	true	true	true	true	true	true
false	true	false	false	false	false	false	true	true	true	true	true	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	true	false	false	true	false	false	true	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	false	true	true	false	true	false						

# TT – Entails Inference – Example



Agents based on Propositional logic, TT-Entail for inference from truth table

$\neg P_{1,2}$  entailed by our KB?

Way – 1 :

1. Get sufficient information  $B_{1,1}, B_{2,1}, P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, P_{3,1}$
2. Enumerate all models with combination of truth values to propositional symbols

2<sup>7</sup>

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$KB$
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
:	:	:	:	:	:	:	:	:	:	:	:	:
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	false	true	true	true	true	true	true	false
:	:	:	:	:	:	:	:	:	:	:	:	:
true	false	true	true	false	true	false						

# TT – Entails Inference – Example

## Agents based on Propositional logic, TT-Entail for inference from truth table

$\neg P_{1,2}$  entailed by our KB?

## Way – 1 :

1. Get sufficient information  $B_{1,1}, B_{2,1}, P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, P_{3,1}$
  2. Enumerate all models with combination of truth values to propositional symbols
  3. In all the models, find those models where KB is true, i.e., every sentence  $R_1, R_2, R_3, R_4, R_5$  are true
  4. In those models where KB is true, find if query sentence  $\neg P_{1,2}$  is true No P1T →
  5. If query sentence  $\neg P_{1,2}$  is true in all models where KB is true, then it entails, otherwise it won't

$\rightarrow P_{12} \rightarrow T_{\text{me}}$

1

False

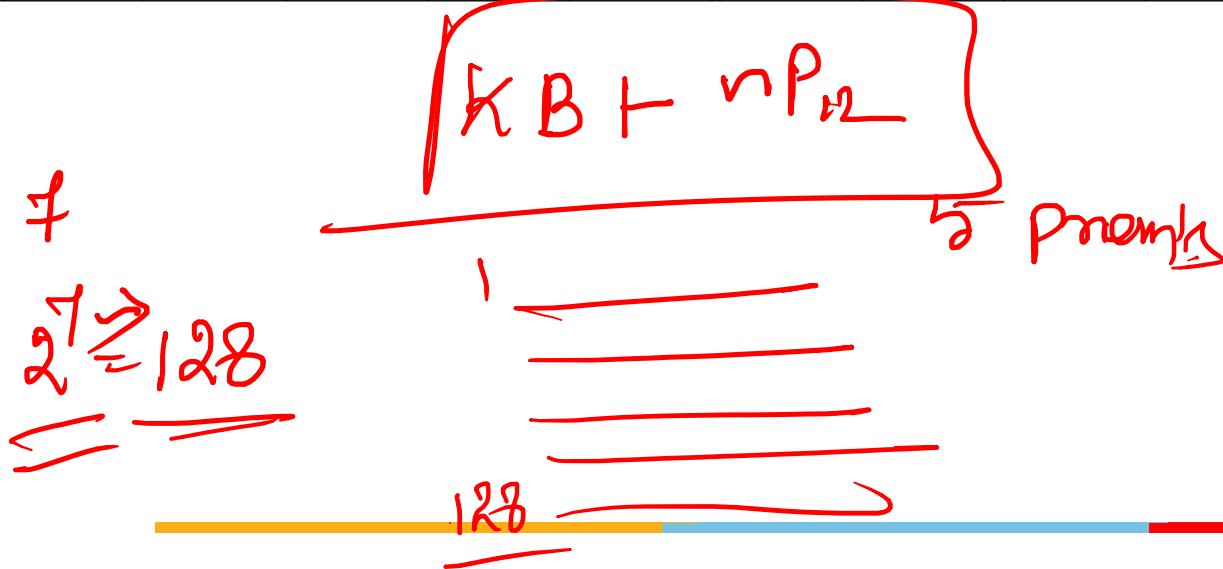
$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$KB$
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	true	false	false	true	false	false	true	true	false
false	true	false	false	false	false	false	true	true	true	true	true	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	false	true	true	false	true	false						

Time

# TT – Entails Inference – Example

Agents based on Propositional logic, TT-Entail for inference from truth table

$B_{1,1}$	$B_{2,1}$	$P_{1,1}$	$P_{1,2}$	$P_{2,1}$	$P_{2,2}$	$P_{3,1}$	$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$KB$
false	true	true	true	true	false	false						
false	false	false	false	false	false	true	true	true	false	true	false	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
false	true	false	false	false	false	false	true	true	false	true	true	false
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	false	false	true	true	true	true	true	true	true
false	true	false	false	true	false	false	true	false	false	true	true	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
true	false	true	true	false	true	false						



# Inference : Properties

- 
- 1. Entailment :  $\alpha \models \beta$
  - 2. Equivalence :  $\alpha \equiv \beta$  if and only if  $\alpha \models \beta$  and  $\beta \models \alpha$
  - 3. Validity
  - 4. Satisfiability

# Inference : Example – Theorem Proving (Self Study)



## Propositional theorem proving - Proof by resolution

Logical Equivalence rules can be used as inference rules

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \text{ commutativity of } \wedge$$

$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \text{ commutativity of } \vee$$

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \text{ associativity of } \wedge$$

$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \text{ associativity of } \vee$$

$$\neg(\neg\alpha) \equiv \alpha \text{ double-negation elimination}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \text{ contraposition}$$

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \text{ implication elimination}$$

$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \text{ biconditional elimination}$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \text{ De Morgan}$$

$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \text{ De Morgan}$$

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \text{ distributivity of } \wedge \text{ over } \vee$$

$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \text{ distributivity of } \vee \text{ over } \wedge$$

# Inference : Example – Theorem Proving

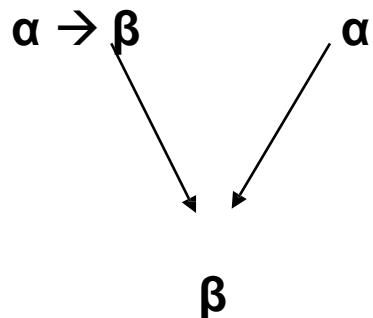
---

## 1. Modes Ponens

## 2. AND Elimination

$\alpha$  : I walk in rain without the umbrella

$\beta$  : I get wet



- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$  commutativity of  $\wedge$
- $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$  commutativity of  $\vee$
- $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$  associativity of  $\wedge$
- $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$  associativity of  $\vee$
- $\neg(\neg\alpha) \equiv \alpha$  double-negation elimination
- $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$  contraposition
- $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$  implication elimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$  biconditional elimination
- $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$  De Morgan
- $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$  De Morgan
- $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$  distributivity of  $\wedge$  over  $\vee$
- $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$  distributivity of  $\vee$  over  $\wedge$

# Inference : Example – Theorem Proving

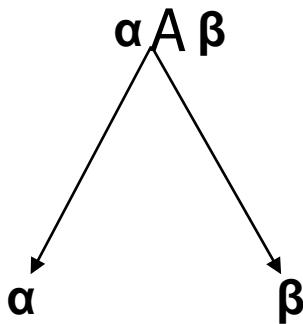
---

1. Modes Ponens

2. AND Elimination

$\alpha$  : I walk in rain without the umbrella

$\beta$  : I get wet



- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$  commutativity of  $\wedge$   
 $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$  commutativity of  $\vee$   
 $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$  associativity of  $\wedge$   
 $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$  associativity of  $\vee$   
 $\neg(\neg\alpha) \equiv \alpha$  double-negation elimination  
 $(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$  contraposition  
 $(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$  implication elimination  
 $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$  biconditional elimination  
 $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$  De Morgan  
 $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$  De Morgan  
 $(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$  distributivity of  $\wedge$  over  $\vee$   
 $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$  distributivity of  $\vee$  over  $\wedge$

# Inference : Example –

# Theorem Proving

KB

- $R_1 : \neg P_{1,1}$

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4 : \neg B_{1,1}$

$R_5 : B_{2,1}$

Query:  $\neg P_{1,2}$ . Can we prove if this sentence be entailed from KB using inference rules?

- $R_2: B_{11} \Leftrightarrow P_{12}^a \downarrow P_{21}^b \vee P_{21} \Rightarrow$   
 $R_6: (B_{11} \rightarrow (P_{12} \vee P_{21})) \wedge ((P_{12} \vee P_{21}) \rightarrow B_{11}) \vdash T$   
 $R_7: (P_{12} \vee P_{21}) \rightarrow B_{11}$   
 $R_8: \neg B_{11} \rightarrow \neg(P_{12} \vee P_{21})$

① Biconditional Elimination  
 And Elimination  
 Contraposition  
 Modus Ponens  
 Demorgans  
 And Elimination

$a \rightarrow T$   
 $a \rightarrow b \rightarrow T$   
 $b \rightarrow T$

# Inference : Example –

## Theorem Proving

$R_1 : \neg P_{1,1}$

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4 : \neg B_{1,1}$  fact  $\rightarrow T$   $(\neg B_{1,1} \rightarrow \text{True})$

$R_5 : B_{2,1}$

$\vdots$

Query:  $\neg P_{1,2}$ . Can we prove if this sentence be entailed from KB using  
Rs inference rules?

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_6 : (B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

$R_7 : ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

$R_8 : (\neg B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1}))$

$R_9 : \neg (P_{1,2} \vee P_{2,1})$

$R_{10} : \neg P_{1,2} \wedge \neg P_{2,1}$

$R_{11} : \neg P_{1,2}$

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$  commutativity of  $\wedge$

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$  commutativity of  $\vee$

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$  associativity of  $\wedge$

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$  associativity of  $\vee$

$\neg(\neg \alpha) \equiv \alpha$  double-negation elimination

$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$  contraposition

$(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$  implication elimination

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$  biconditional elimination

$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$  De Morgan

$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$  De Morgan

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$  distributivity of  $\wedge$  over  $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$  distributivity of  $\vee$  over  $\wedge$

True

b True

last step

Refuse

Biconditional Elimination

And Elimination

Contraposition

Modus Ponens

Demorgans

And Elimination

a : True

b  $\rightarrow$  b True

b True

b True

# Inference : Example –

## Theorem Proving

KB  
R<sub>1</sub> :  $\neg P_{1,1}$

R<sub>2</sub> :  $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

R<sub>3</sub> :  $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

R<sub>4</sub> :  $\neg B_{1,1}$

R<sub>5</sub> :  $B_{2,1}$

Query  $\neg P_{1,2}$  Can we prove if this sentence be entailed from KB using inference rules?

R<sub>11</sub> :  $P_{1,2}$

R<sub>12</sub> :  $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

R<sub>6</sub> :  $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

R<sub>7</sub> :  $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

R<sub>8</sub> :  $(\neg B_{1,1} \Rightarrow \neg (P_{1,2} \vee P_{2,1}))$

R<sub>9</sub> :  $\neg (P_{1,2} \vee P_{2,1})$

R<sub>10</sub> :  $\neg P_{1,2} \wedge \neg P_{2,1}$

R<sub>11</sub> :  $\neg P_{1,2}$

least fact

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$  commutativity of  $\wedge$

$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$  commutativity of  $\vee$

$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$  associativity of  $\wedge$

$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$  associativity of  $\vee$

$\neg(\neg \alpha) \equiv \alpha$  double-negation elimination

$(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$  contraposition

$(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$  implication elimination

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$  biconditional elimination

$\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$  De Morgan

$\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$  De Morgan

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$  distributivity of  $\wedge$  over  $\vee$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$  distributivity of  $\vee$  over  $\wedge$

KB  
~~negate~~  $(\neg P_{1,2}) \Rightarrow \neg(\neg P_{1,2}) = P_{1,2}$

Biconditional Elimination

And Elimination

Contraposition

Modus Ponens

Demorgans

And Elimination

Assumed fact

①

Derived fact

⑥

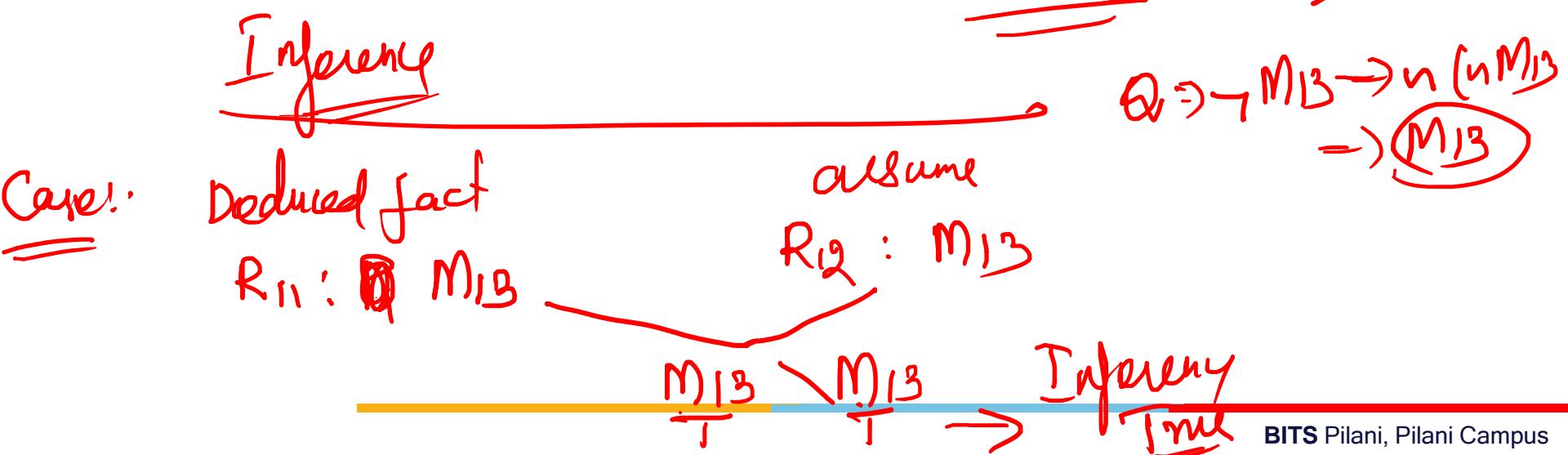
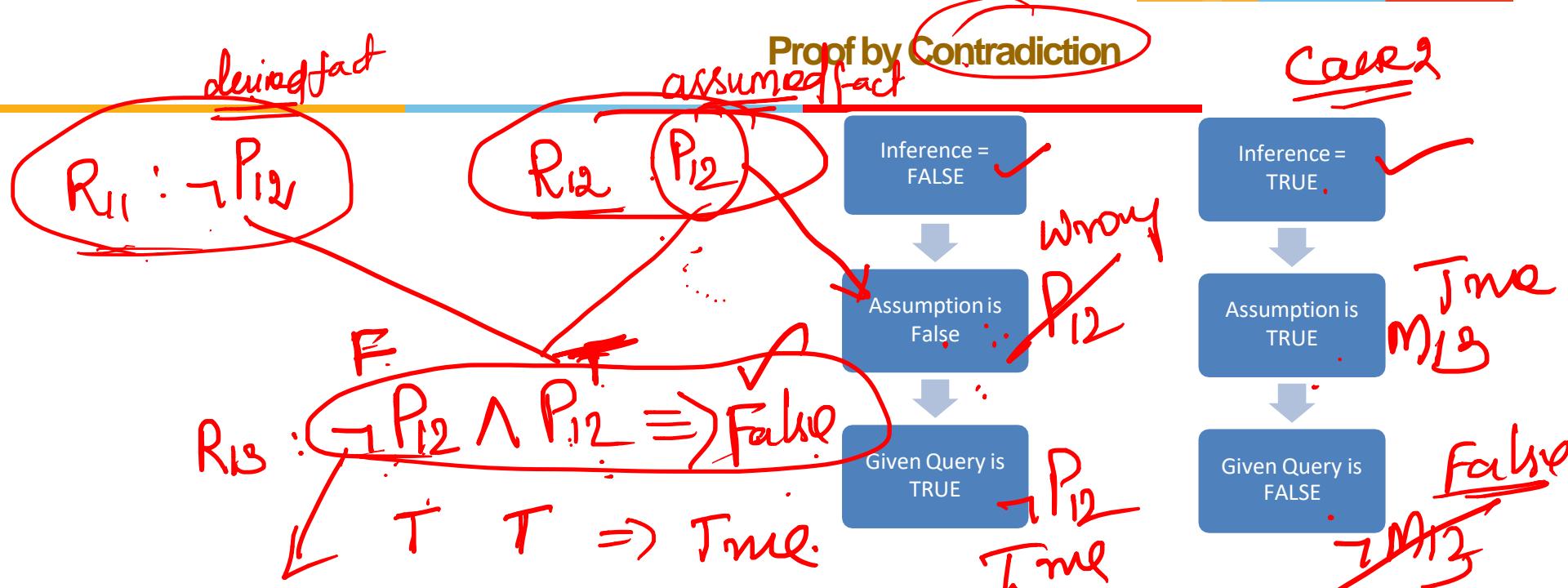
# Propositional Logic

$$Q \rightarrow N$$

innovate

achieve

lead



## Horn Clause

1. **Definite Clause** : A horn clause with exactly one positive literal
2. **Fact** : Definite clause with no negative literal / assertion
3. Rule
4. Inference by Chaining

*Modus Ponens + Contraposition*



Disjunctive Syllogism

① convert into CNF  $R_1 \dots R_5$   
~~Conjunction of disjunctions~~

4				
3				
2				
1				
	1	2	3	4

# PL-Resolution : CNF conversion

DB

$$R_1 : \neg P_{1,1}$$

$$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

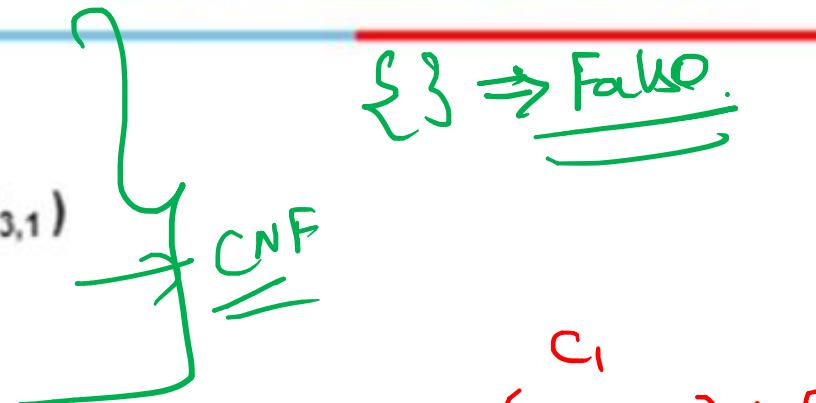
$$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$

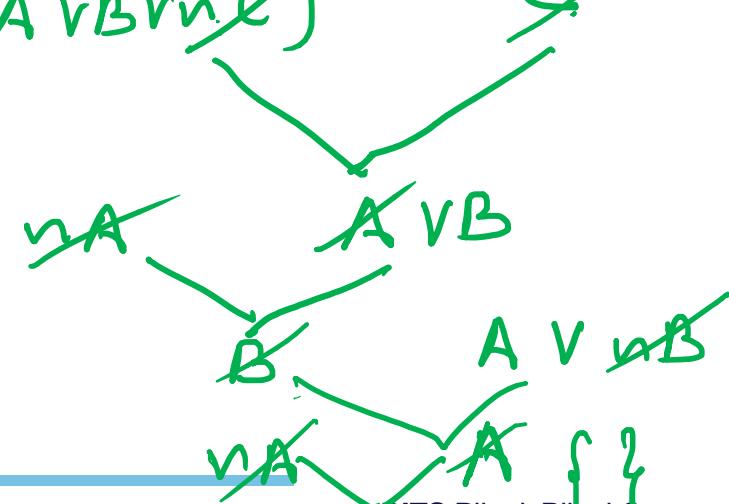
Query:  $\neg P_{1,2}$

Wumpus world Book example



$$\frac{c_1 \quad c_2}{(A \vee B) \wedge (A \vee B \vee \neg C) \wedge \neg A}$$

$$\frac{c_2}{(A \vee B \vee \neg C)}$$



Eg clause  $C_1 \wedge C_2 \wedge C_3$

Conjunctive Normal Form :

$$(A \vee \neg B) \wedge (A \vee B \vee \neg C) \wedge \neg A$$

Unit Resolution :  $\neg A$

Query : Is 'C' true?

↓

$$\frac{\neg A : \neg C \Rightarrow \neg(\neg C) \Rightarrow C}{\text{assumed fact}}$$

# PL-Resolution : CNF conversion



## Wumpus world Book example

R<sub>1</sub> :  $\neg P_{1,1}$

R<sub>2</sub> :  $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$  ✓

R<sub>3</sub> :  $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

R<sub>4</sub> :  $\neg B_{1,1}$

R<sub>5</sub> :  $B_{2,1}$

Query:  $\neg P_{1,2}$

} CNF conversion  
=   
↓ Theorem

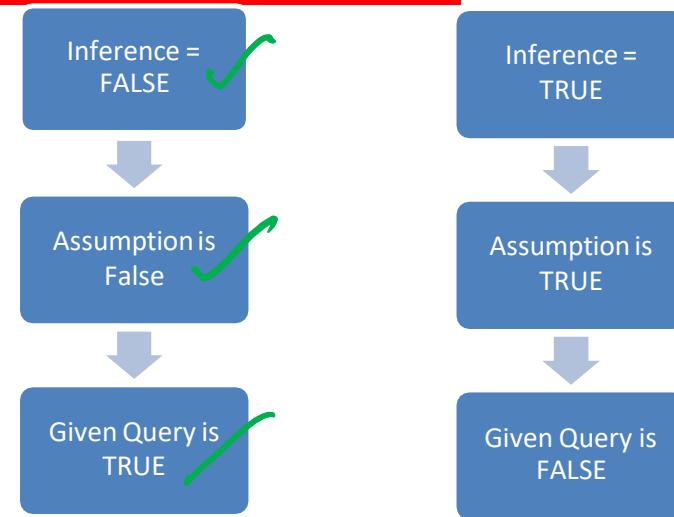
✓ Conjunctive Normal Form :

$$(A \vee \neg B) \wedge (A \vee B \vee \neg C) \wedge \neg A$$

Unit Resolution :  $\neg A$

Query : Is 'C' true?

## Proof by Contradiction



## Wumpus world Book example

$R_1 : \neg P_{1,1}$

$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$

$R_4 : \neg B_{1,1}$

$R_5 : B_{2,1}$

Query:  $\neg P_{1,2}$

Conjunctive Normal Form :

$(A \vee \neg B) \wedge (A \vee B \vee \neg C) \wedge \neg A$

Unit Resolution :  $\neg A$

Query : Is 'C' true?

# PL-Resolution

~~KB~~

- ~~R<sub>1</sub> :  $\neg P_{1,1}$~~
- ~~R<sub>2</sub> :  $B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$~~
- ~~R<sub>3</sub> :  $B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$~~
- ~~R<sub>4</sub> :  $\neg B_{1,1}$~~
- ~~R<sub>5</sub> :  $B_{2,1}$~~

Query:  $\neg P_{1,2}$

$R_6 \Rightarrow R_2$

- R<sub>6</sub> :  $\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}$
- R<sub>7</sub> :  $\neg P_{1,2} \vee B_{1,1}$
- R<sub>8</sub> :  $\neg P_{2,1} \vee B_{1,1}$
- R<sub>9</sub> :  $\neg B_{2,1} \vee P_{1,1} \vee P_{2,2} \vee P_{3,1}$
- R<sub>10</sub> :  $\neg P_{1,1} \vee B_{2,1}$
- R<sub>11</sub> :  $\neg P_{2,2} \vee B_{2,1}$
- R<sub>12</sub> :  $\neg P_{3,1} \vee B_{2,1}$

R<sub>2</sub>

- $(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$  commutativity of  $\wedge$
- $(\alpha \vee \beta) \equiv (\beta \vee \alpha)$  commutativity of  $\vee$
- $((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$  associativity of  $\wedge$
- $((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$  associativity of  $\vee$
- $\neg(\neg \alpha) \equiv \alpha$  double-negation elimination
- $(\alpha \Rightarrow \beta) \equiv (\neg \beta \Rightarrow \neg \alpha)$  contraposition
- $(\alpha \Rightarrow \beta) \equiv (\neg \alpha \vee \beta)$  implication elimination
- $(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$  biconditional elimination
- $\neg(\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$  De Morgan
- $\neg(\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$  De Morgan
- $(\alpha \wedge (\beta \vee \gamma)) = ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$  distributivity of  $\wedge$  over  $\vee$
- $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$  distributivity of  $\vee$  over  $\wedge$

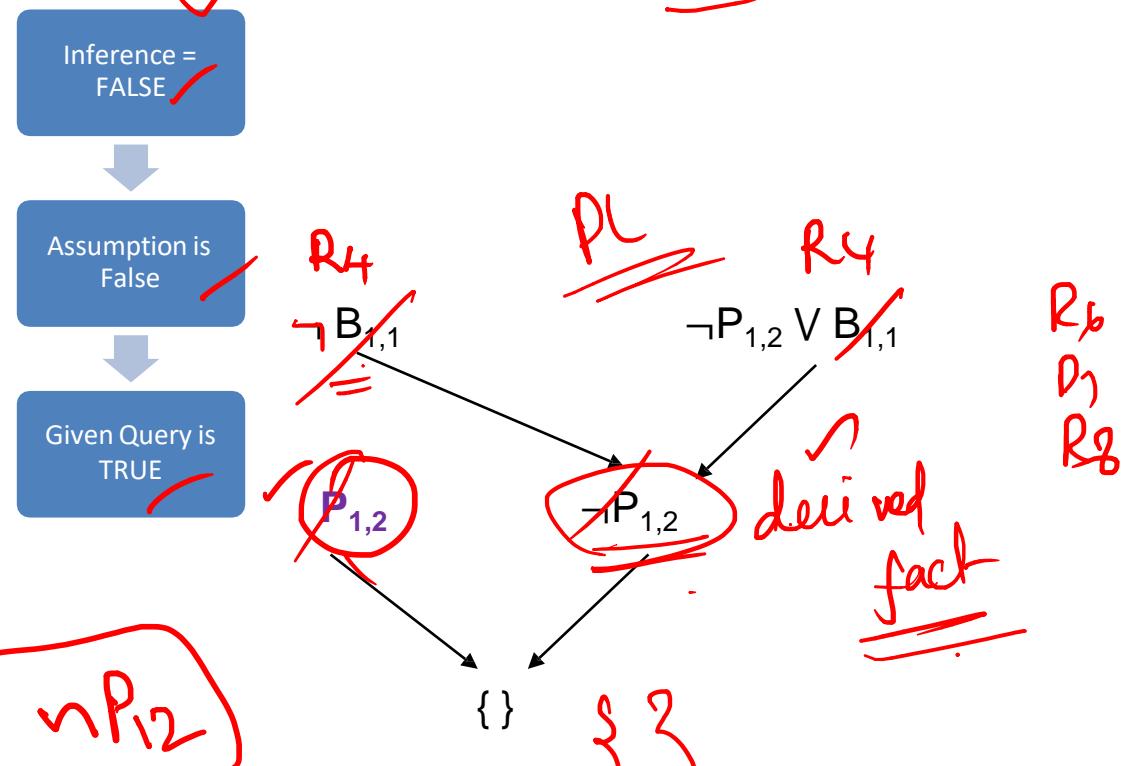
$R_3$  H [W/C]

Eliminate	$\Leftrightarrow$	$R_2 : B_{1,1} \leftrightarrow (P_{1,2} \vee P_{2,1})$	$R_3 : B_{2,1} \leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
$\leftrightarrow$	Biconditional Elimination	$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$	$(B_{2,1} \Rightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})) \wedge ((P_{1,1} \vee P_{2,2} \vee P_{3,1}) \Rightarrow B_{2,1})$
$\rightarrow$	Implication Elimination	$\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})$	$\neg B_{2,1} \vee (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
Clause level $\neg$	De Morgan	$(\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1}$	$(\neg P_{1,1} \wedge \neg P_{2,2} \wedge \neg P_{3,1}) \vee B_{2,1}$
CNF Form	Distributivity of $\vee$ over $\wedge$	$(\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$	$(\neg P_{1,1} \vee B_{2,1}) \wedge (\neg P_{2,2} \vee B_{2,1}) \wedge (\neg P_{3,1} \vee B_{2,1})$

$R_7 \wedge C_1$   $R_8 \Downarrow$ , CNF

## Unit Resolution: Query: $\neg P_{1,2}$

To find: Is there a pit in location (1,2) using the CNF obtained in previous slide



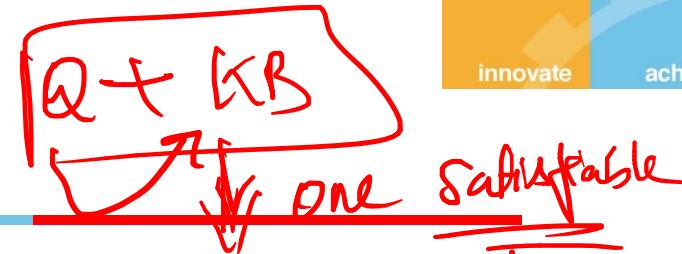
$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench		$\approx$ Breeze	$\blacksquare$ PIT
	$\approx$ Breeze $\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench 	$\blacksquare$ PIT	$\approx$ Breeze
$\Sigma\Sigma\Sigma\Sigma\Sigma$ Stench		$\approx$ Breeze	
 START	$\approx$ Breeze	$\blacksquare$ PIT	$\approx$ Breeze
1	2	3	4

# DPLL Algorithm

innovate

achieve

lead

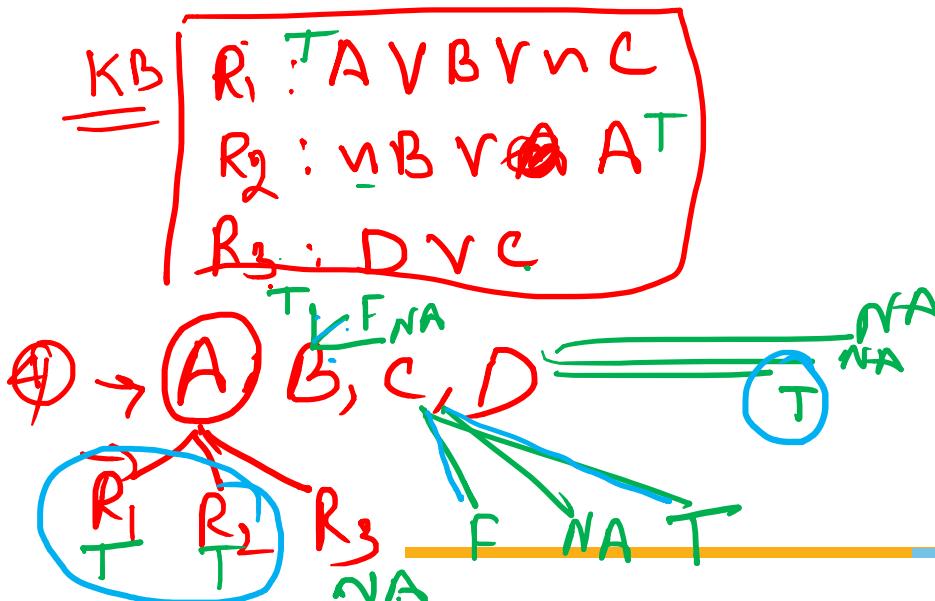


In logic and computer science, the Davis–Putnam–Logemann–Loveland (**DPLL**) algorithm is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form

DFS + backtracking

## Improvements:

1. Early Termination ✓
2. Pure Symbolic Heuristic ✓
3. Unit Clause Heuristic ✓



A & D are pure symbol

Time

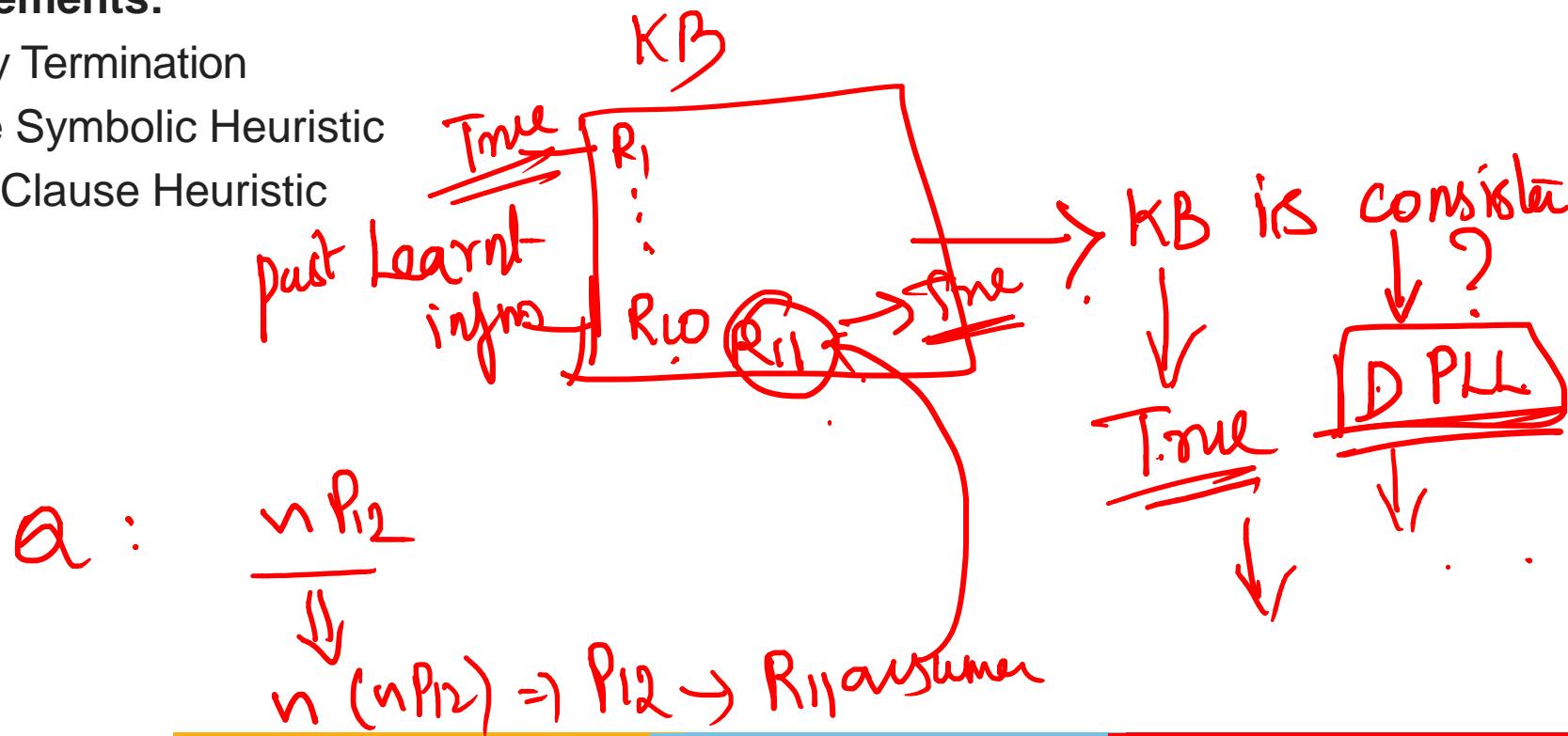
B, C

# DPLL Algorithm

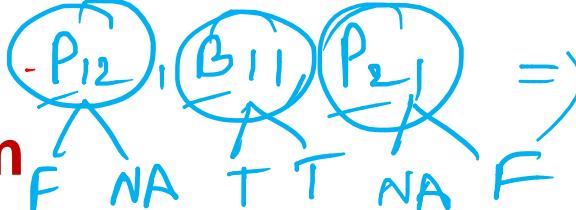
In logic and computer science, the Davis–Putnam–Logemann–Loveland (**DPLL**) algorithm is a complete, backtracking-based search **algorithm** for deciding the satisfiability of propositional logic formulae in conjunctive normal form

## Improvements:

1. Early Termination
2. Pure Symbolic Heuristic
3. Unit Clause Heuristic



## DPLL Algorithm



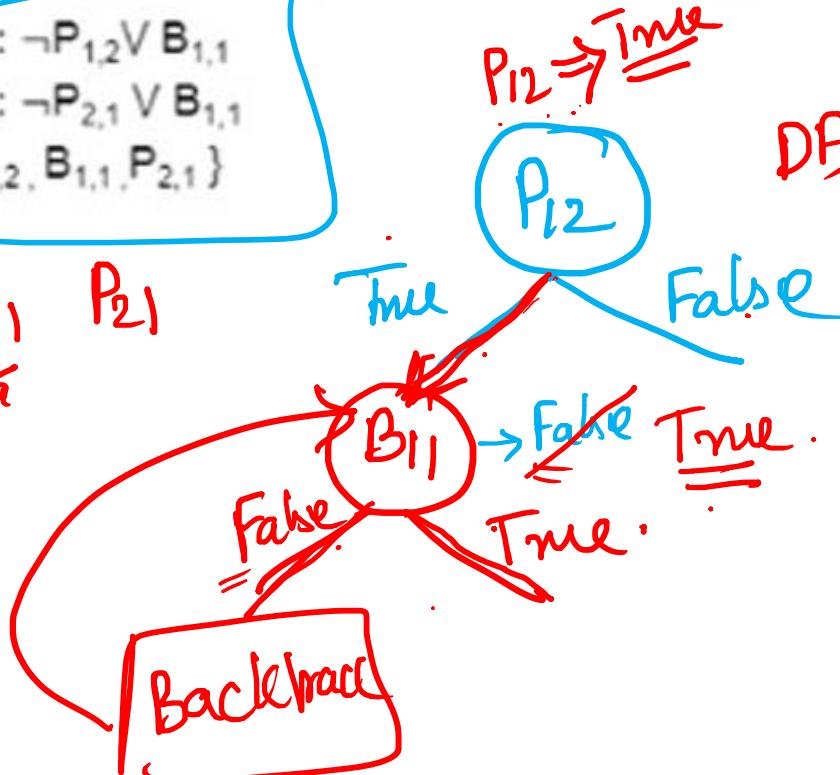
$KB + \alpha$

$$R_7 : \neg P_{1,2} \vee B_{1,1}$$

$$R_8 : \neg P_{2,1} \vee B_{1,1}$$

$$\{P_{1,2}, B_{1,1}, P_{2,1}\}$$

$$\begin{matrix} P_{1,2} & B_{1,1} & P_{2,1} \\ T & F & F \end{matrix}$$



DFS + Backtr.

$\text{KB}$

$$R_7 : \neg P_{1,2} \vee B_{1,1}$$

$$R_8 : \neg P_{2,1} \vee B_{1,1}$$

$\text{KB}$

$$R_7 : \neg P_{1,2} \vee B_{1,1} \rightarrow \text{False}$$

$$R_8 : \neg P_{2,1} \vee B_{1,1}$$

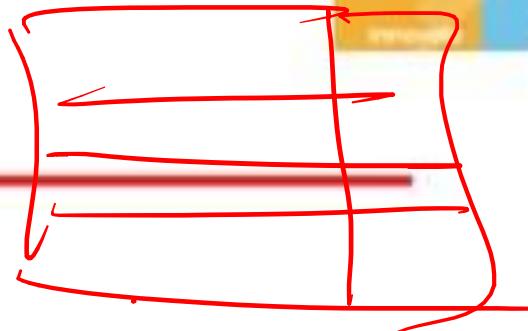
$$R_7 : \neg P_{1,2} \vee B_{1,1}$$

$$R_8 : \neg P_{2,1} \vee B_{1,1}$$

## DPLL Algorithm

~~3~~

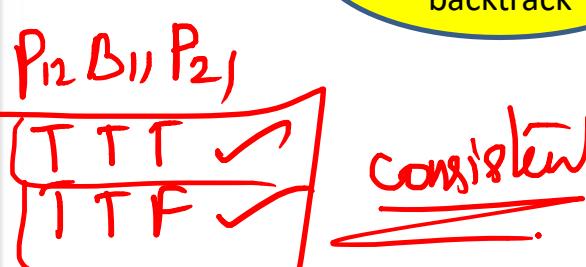
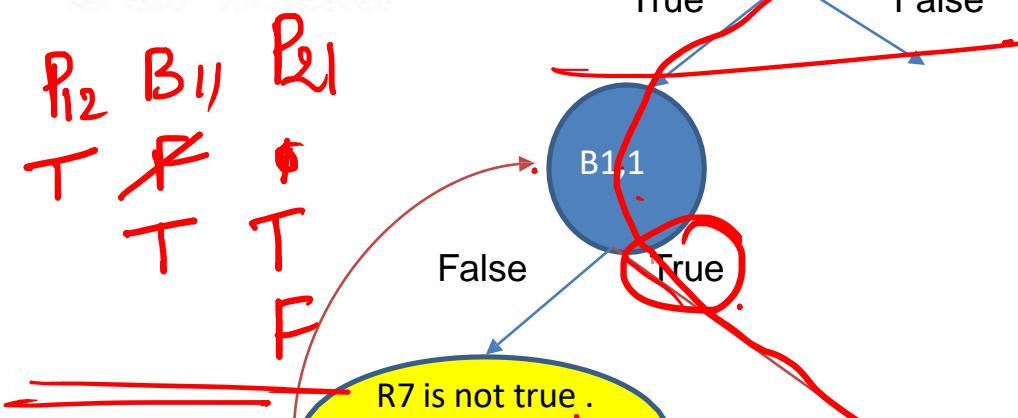
$2^3 \Rightarrow 8$



$$R_7 : \neg P_{1,2} \vee B_{1,1}$$

$$R_8 : \neg P_{2,1} \vee B_{1,1}$$

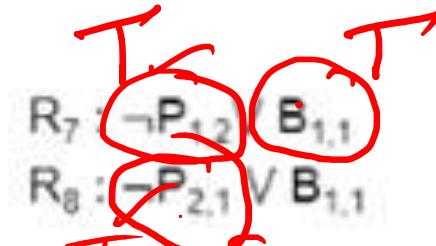
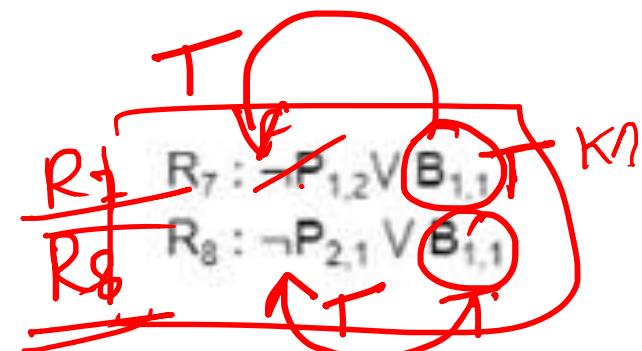
$$\{P_{1,2}, B_{1,1}, P_{2,1}\}$$



All premises satisfied in this DFS path. Returns result

$$R_7 : \neg P_{1,2} \vee B_{1,1}$$

$$R_8 : \neg P_{2,1} \vee B_{1,1}$$



---

## Required Reading: AIMA - Chapter # 4.1, #4.2, #5.1, #9

### Next Session Plan:

- (Prerequisite Reading : Refresh the basics of probability , Bayes Theorem , Conditional Probability, Product Rule, Conditional Independence, Chain Rule)
  - Inferences using Logic ( Forward / Backward Chaining / DPLL algorithm)
  - Bayesian Network
  - Representation
  - Inferences (Exact and approximate-only Direct sampling)
- Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

**DSECLZG557**

## M4 : Knowledge Representation using Logics

Indumathi V

Guest Faculty,  
BITS -WILP

**BITS** Pilani  
Pilani Campus

# Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time

M7 Ethics in AI

True  
False  
T V =  
Propositional Logic

# Towards Predicate Logic) First Order Logic

- ① All courses are offered and interesting →
- ① Multiple var-Val pair
  - ② Quantification
- ② All offered courses are interesting
- More expressible logic - FOL
- Predicat  
Logic
- ③ Some of the courses are offered and interesting [Atleast one of the offered courses is interesting]
- ④ Some of the offered courses are interesting

## Towards Predicate Logic $\rightarrow$ function definition

=

function( Val<sub>1</sub>, Val<sub>2</sub> .. - Val<sub>n</sub> )

i) All courses are offered and interesting

offer(x)

Interesting(x) common noun  
action verb

obj/entity

All offered courses are interesting

O(x)

I(x)

Unary Predicate - 1 arg

Predicate fn( v<sub>1</sub>, v<sub>2</sub> .. - v<sub>n</sub> )

↓ tuples [ aug.]

Some of the courses are offered and interesting [Atleast one of the offered courses is interesting]

Some of the offered courses are interesting

(1)  $A$  is teaching  $\equiv$  Could  $C$  subject.  $ACI$  for the student  $S$ .

(2) Teaching ( $A, ACI, S$ )

(3) Seller  $s$  sells ~~all~~<sup>some</sup> the product to customer.

#  $\exists p \text{ Sell } (S, P, C)$

→ all the product

3  $\exists p \text{ Sell } (S, P, C)$  → some of the product  
existential Quantification

## Towards Predicate Logic

All courses are offered and interesting  $\forall x O(x) \wedge I(x)$

② All offered courses are interesting  $\forall x O(x) \rightarrow I(x)$   
 1o  $\rightarrow$  1o  
 implication law  
 5

③ Some of the courses are offered and interesting [Atleast one of the offered courses is interesting]

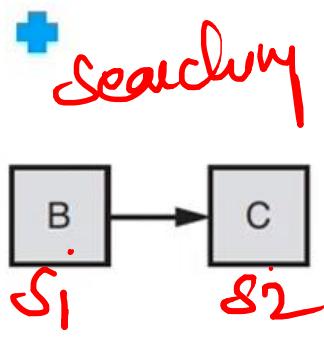
①  $S_1$   $O$   $I$   $\rightarrow \exists x [O(x) \wedge I(x)]$

④ Some of the offered courses are interesting

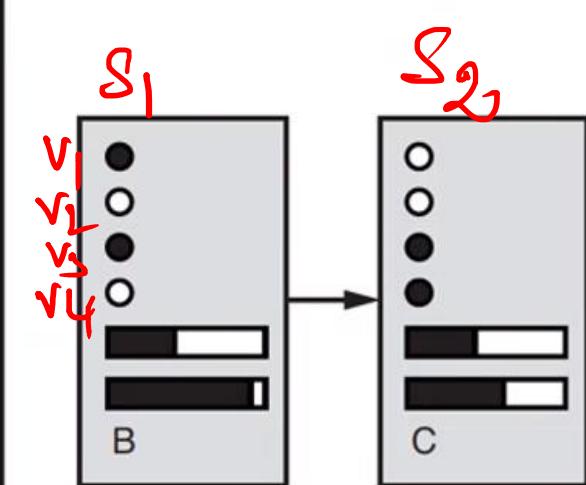
$\exists x O(x) \rightarrow I(x)$

None  $S_2$

# Towards Predicate Logic

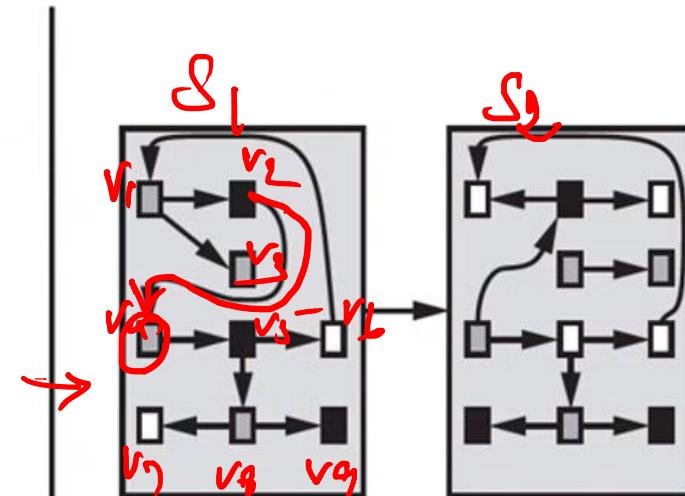


(a) Atomic



(b) Factored

PL resolution



(b) Structured

Predicate logic

# Predicate Logic

---

Squares neighboring the wumpus are smelly

~~Objects~~: squares, wumpus

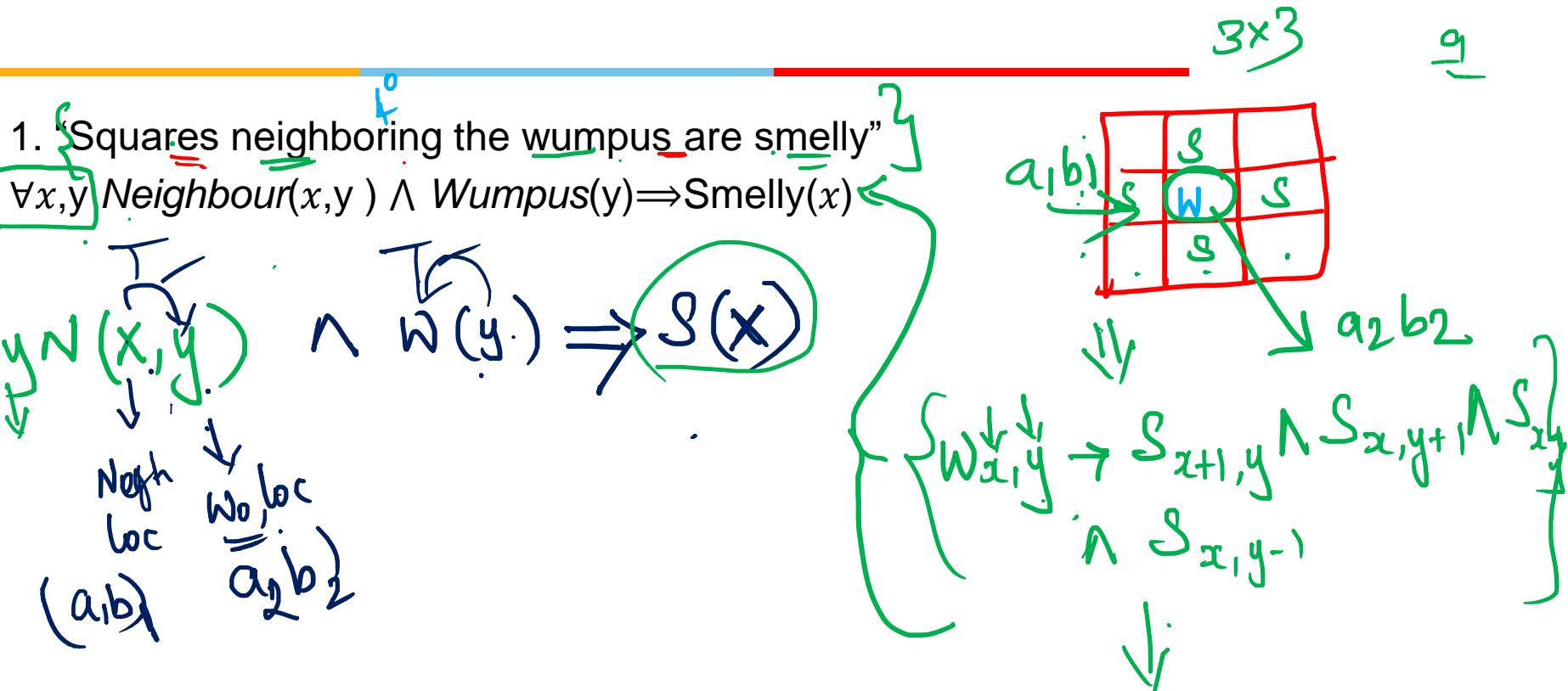
~~Unary Relation~~ (properties of an object): smelly N-ary

Relation (between objects): neighboring

~~Function~~:-

Primary difference between propositional and first-order logic lies in “ontological commitment” – the assumption about the nature of reality.

# Predicate Logic – Sample Modelling



# Predicate Logic – Sample Modelling

innovate

achieve

lead

- ✓ 2. "Everybody loves somebody"  $\forall x \exists y \text{ Loves}(x, y)$
3. "There is someone who is loved by everyone"  $\exists y \forall x \text{ Loves}(x, y)$
- Order of quantifiers is important

2 ways (Forward chaining)  
Convert the Predicate into proposition

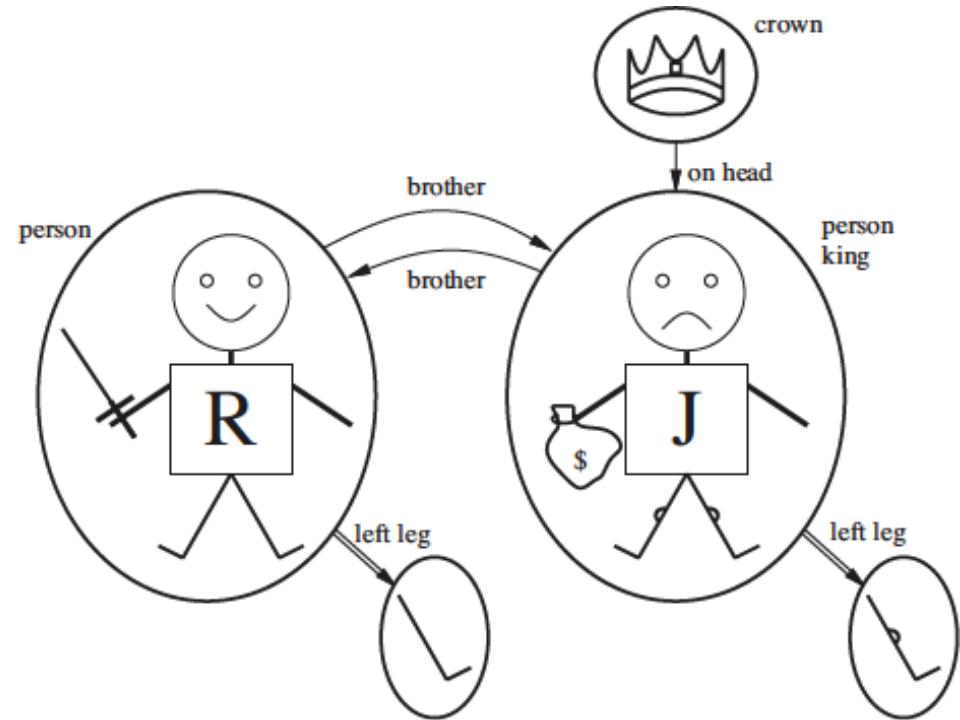
PL  
TT  
Then  
DPLL

# Predicate Logic – Sample Modelling

$\text{Brother}(\text{Richard}, \text{John}) \wedge \text{Brother}(\text{John}, \text{Richard})$

$\text{King}(\text{Richard}) \vee \text{King}(\text{John})$

$\neg \text{King}(\text{Richard}) \Rightarrow \text{King}(\text{John})$

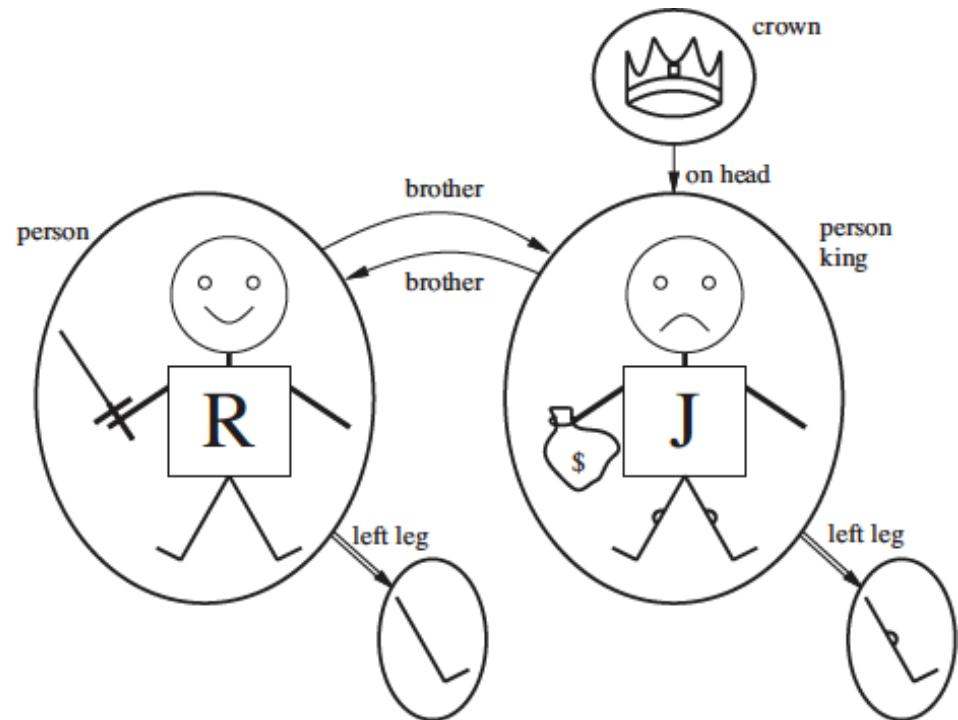


# Unification & Lifting

Brother(Richard, John)  $\wedge$  Brother(John, Richard)

King(Richard)  $\vee$  King(John)

$\neg$ King(Richard)  $\Rightarrow$  King(John)



# Predicate Logic – Sample Modelling

## Quantifiers

$\forall \exists \exists$

2 persons

Brother(Richard, John)  $\wedge$  Brother(John, Richard)

King(Richard)  $\vee$  King(John)

$\neg$ King(Richard)  $\Rightarrow$  King(John)

Removal of  $\forall$

Universal Instantiation

P

"All Kings are persons"

$\forall x \text{ King}(x) \rightarrow \text{Person}(x)$

King(Richard)  $\rightarrow$  Person(Richard)  
King(John)  $\rightarrow$  Person(John)

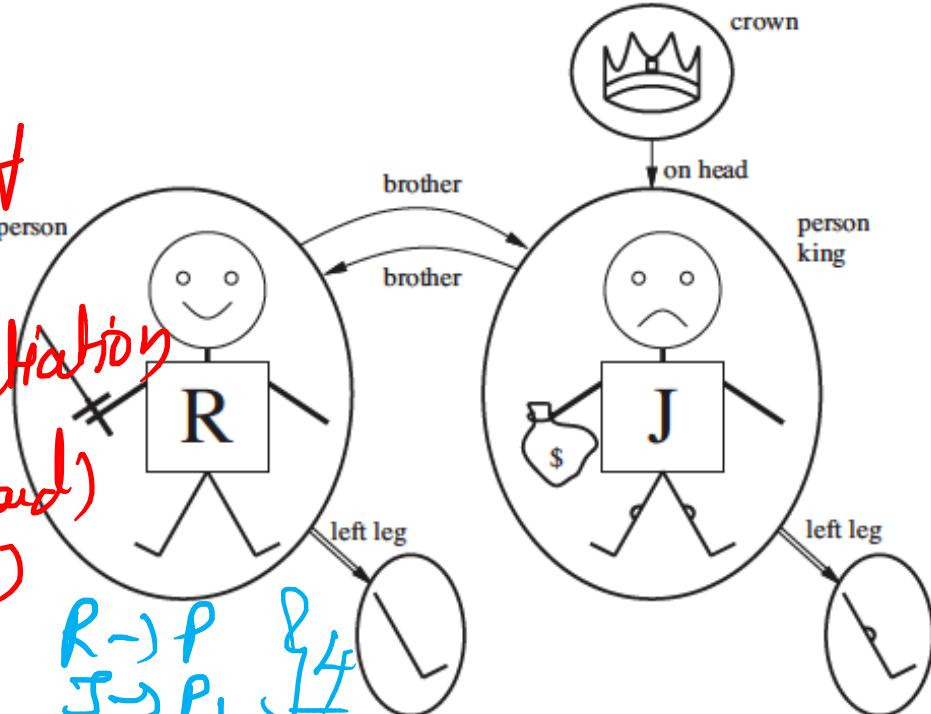
"King John has a crown on his head"

$\exists x \text{ Crown}(x) \wedge \text{OnHead}(x, John)$

Substitute temp var

Crown(F<sub>e</sub>)  $\wedge$  Onhead(F<sub>e</sub>, John)

Ground Term: A term with no variables. E.g., King(Richard)



$R \rightarrow P$   
 $J \rightarrow P$

Existential Instantiation

## Quantifiers

---

Brother(Richard, John)  $\wedge$  Brother(John, Richard)

King(Richard)  $\vee$  King(John)

$\neg$ King(Richard)  $\Rightarrow$  King(John)

"All Kings are persons"

$\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$

"King John has a crown on his head"

$\exists x \text{ Crown}(x) \text{ AOnHead}(x, \text{John})$

Ground Term: A term with no variables. E.g., King(Richard)

---

# Predicate Logic – Inference



1. Substitute for Quantifiers
2. Convert into Propositional Logic
3. Apply inference tech

$\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$

Richard the Lionheart is a king  $\Rightarrow$  Richard the Lionheart is a person

King John is a king  $\Rightarrow$  King John is a person

$\exists x \text{ Crown}(x) \text{ AOnHead}(x, \text{John})$

Crown(C<sub>1</sub>) A OnHead(C<sub>1</sub>, John) ||C<sub>1</sub> is imputed assumed fact

## Forward Chaining

- Consider the following problem:

KB  
~~The law says it is a crime for an American to sell weapons to hostile nations.~~ ~~The country Nono, an enemy of America, has some missiles (and all of its missiles were sold to it by Colonel West, who is American.)~~

Q:

- We will prove that West is a criminal.

American( $x$ )  $\wedge$  hostile( $y$ )  $\wedge$  weapon( $z$ )  $\wedge$

$\text{sell}(x, z, y) \rightarrow \text{Criminal}(x)$

Country (Nono)  
American (West)

## Forward Chaining

- ~~has some missile~~  $x \rightarrow \text{West}$
- "All of its missiles were sold to it by Colonel West"  
 $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
  - Missiles are weapons  
 $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
  - Hostile means enemy  
 $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$
  - "West, who is American"  
 $\text{American}(\text{West})$
  - "The country Nono, an enemy of America"  
 $\text{Enemy}(\text{Nono}, \text{America})$

# Forward Chaining

- First, we will represent the facts in First Order Definite Clauses

“... it is a crime for an American to sell weapons to hostile nations”

$$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$$

“Nono ... has some missiles”

$$\exists x \text{Owes}(\text{Nono}, x) \wedge \text{Missile}(x)$$

is transformed into two definite clauses by Existential Instantiation

$$\text{Owes}(\text{Nono}, M_1)$$

$$\text{Missile}(M_1)$$

Query: Criminal(West)

## Forward Chaining + Rule system

- ①  $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$
- ②  $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
- ③  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
- ④  $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

Starts from fact  
+ facts

- ~~✓~~ Missle(M1) f<sub>1</sub>
- ~~✓~~ Owns(Nono, M1) f<sub>2</sub>
- ~~✓~~ American (West) f<sub>3</sub>
- ~~✓~~ Enemy (Nono, America) f<sub>4</sub>

# Forward Chaining

- Consider the following problem:

*The law says it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.*

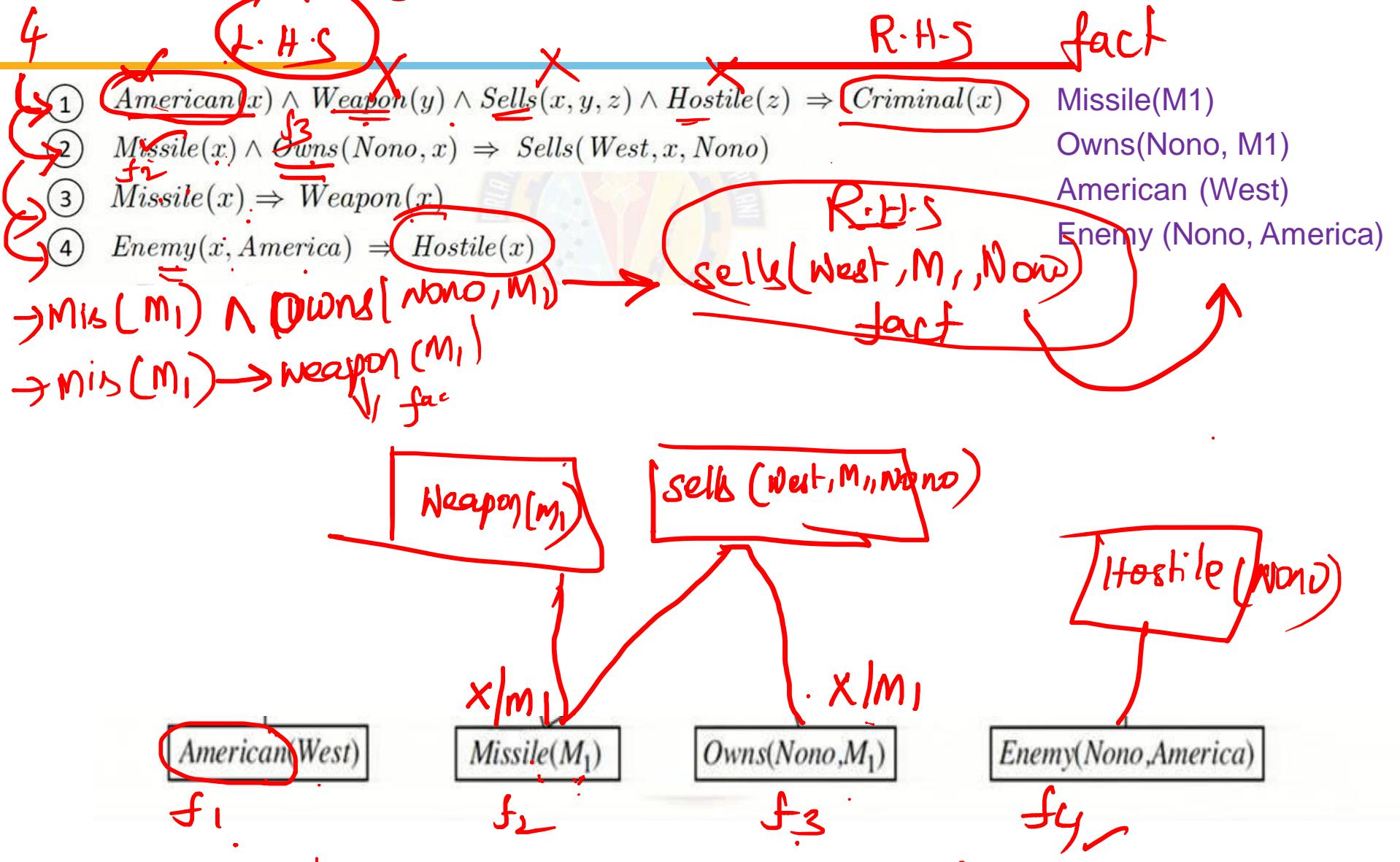
- We will prove that West is a criminal

## Algorithm:

- Start from the facts
- Trigger all rules whose premises are satisfied
- Add the conclusion to known facts
- Repeat the steps till no new facts are added or the query is answered

## Forward Chaining

Q : Criminal(west)



# fact oriented approach

## Forward Chaining

L.H.S

$f_1$   $f_2$

$f_3$

$f_4$

$\downarrow$

R.H.S

- (1)  $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- (2)  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- (3)  $Missile(x) \Rightarrow Weapon(x)$
- (4)  $Enemy(x, America) \Rightarrow Hostile(x)$

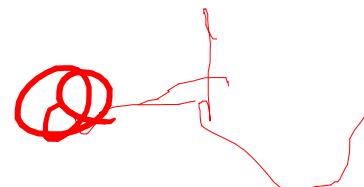
Missile(M1)  $f_1$

Owns(Nono, M1)  $f_2$

American (West)  $f_3$

Enemy (Nono, America)  $f_4$

Criminal (West)



$x/West$

$y/M1$

$x, y, z$   
West, M1, Nono

$z/Nono$

$f_1$

Weapon(M1)

Sells(West, M1, Nono)

$f_2$

$American(West)$

$Missile(M1)$

$Owns(Nono, M1)$

$Enemy(Nono, America)$

$f_3$

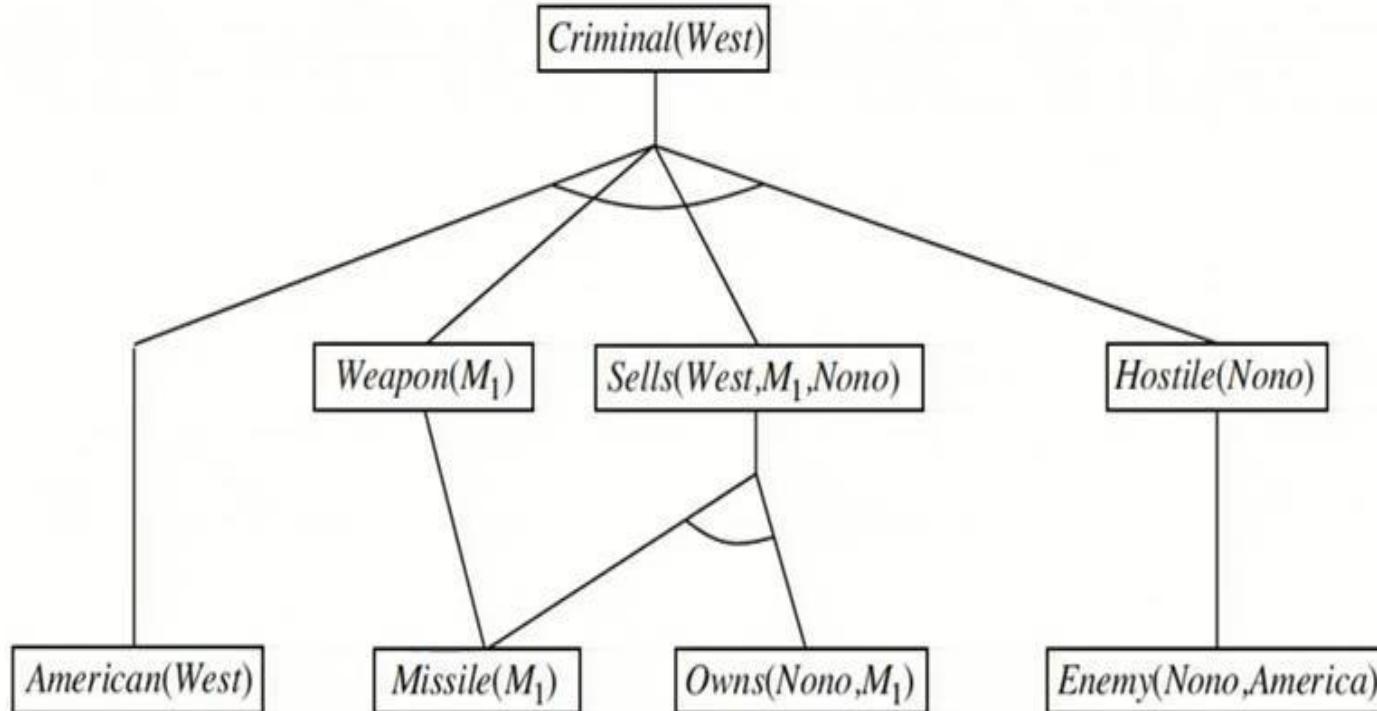
$f_4$

$f_5$

$f_6$

# Forward Chaining

- ①  $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- ②  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- ③  $Missile(x) \Rightarrow Weapon(x)$
- ④  $Enemy(x, America) \Rightarrow Hostile(x)$



# Forward Chaining

## Algorithm:

1. Start from the facts - Conjunct Ordering
2. Trigger all rules whose premises are satisfied - Pattern Matching
3. Add the conclusion to known facts – **Irrelevant Facts**
4. Repeat the steps till no new facts are added or the query is answered – Redundant Rule Matching

**Backward Chaining** → Goal oriented



### Algorithm:

1. Form Definite Clause
2. Start from the Goals
3. Search through rules to find the fact that support the proof
4. If it stops in the fact which is to be proved → Empty Set- LHS

Divide & Conquer Strategy  
Substitution by Unification

# Backward Chaining

6

R.H.S

fech **ICA**

- 1  $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$

2  $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$

3  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$

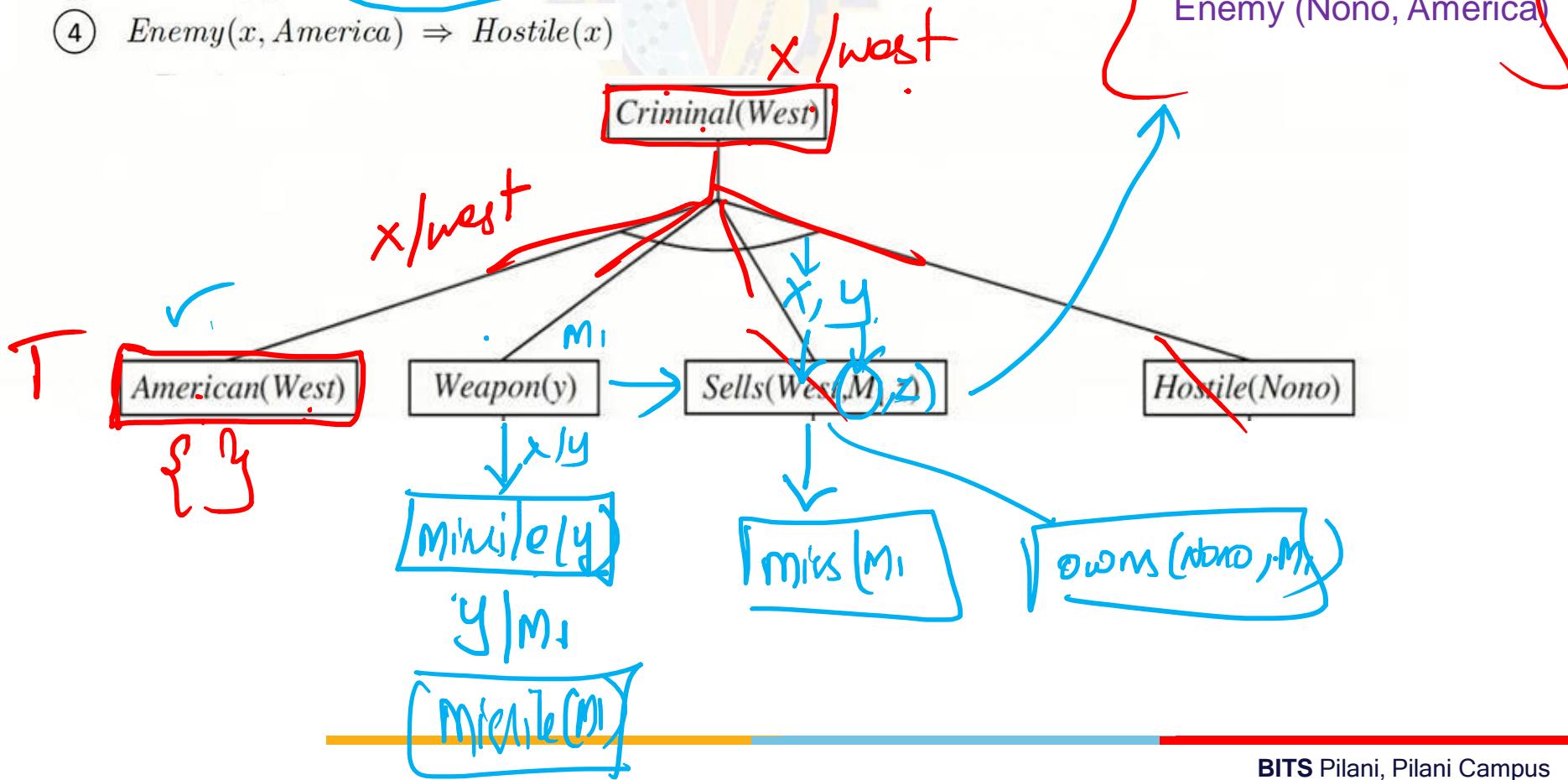
4  $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$

# Missile(M1)

Owns(Nono, M1)

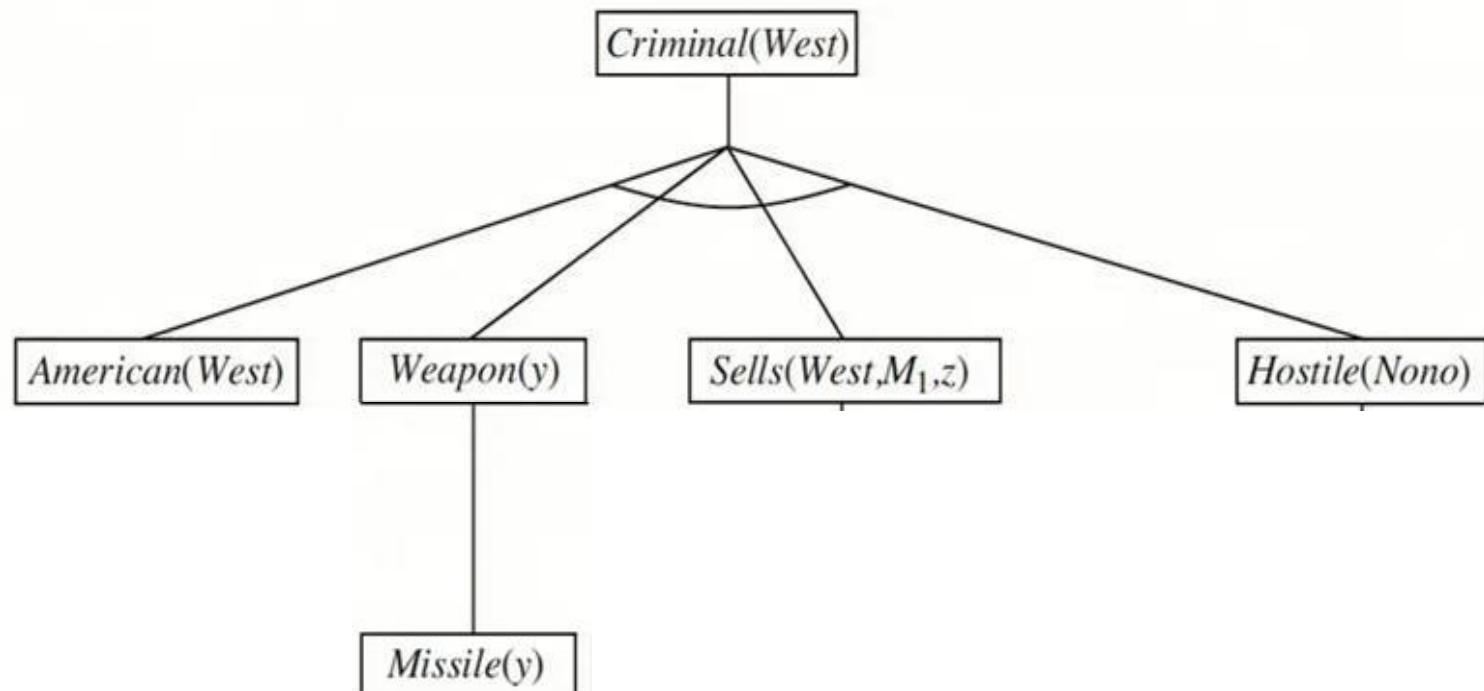
## American (West)

## Enemy (Nono, America)



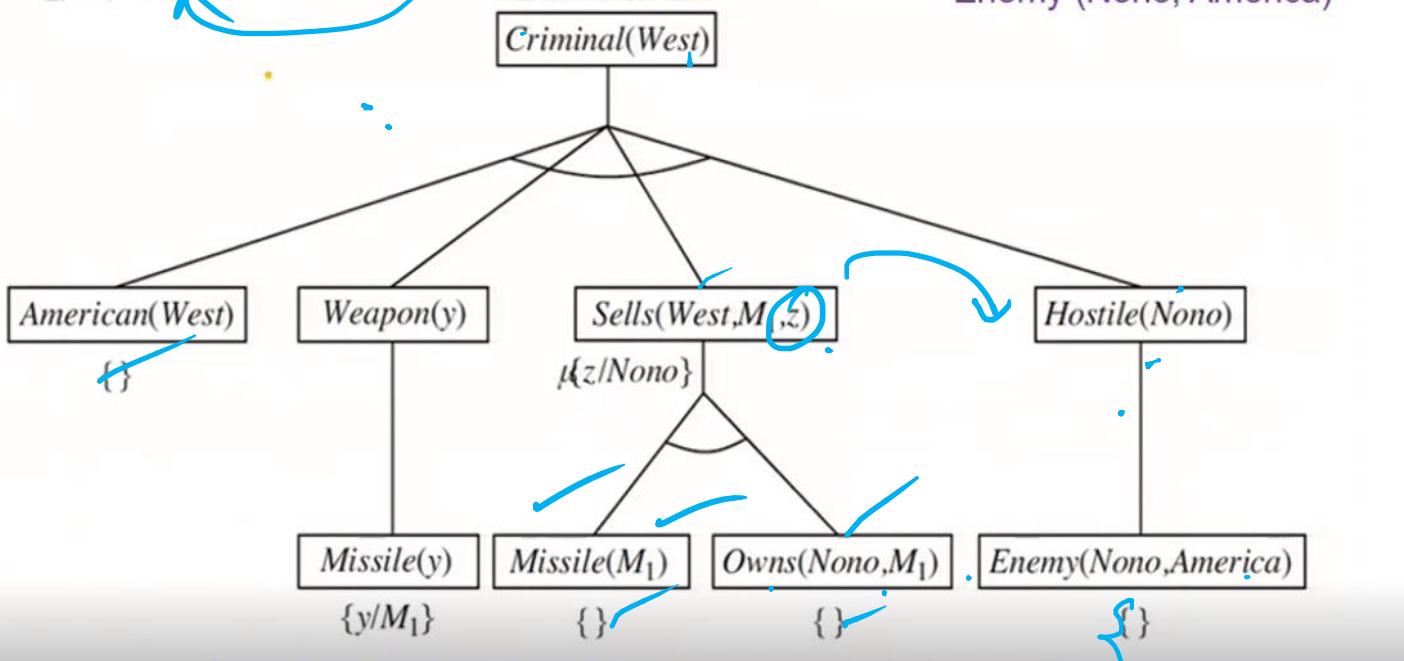
# Backward Chaining

- ①  $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- ②  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
- ③  $Missile(x) \Rightarrow Weapon(x)$
- ④  $Enemy(x, America) \Rightarrow Hostile(x)$
- Missile(M1)  
 Owns(Nono, M1)  
 American (West)  
 Enemy (Nono, America)



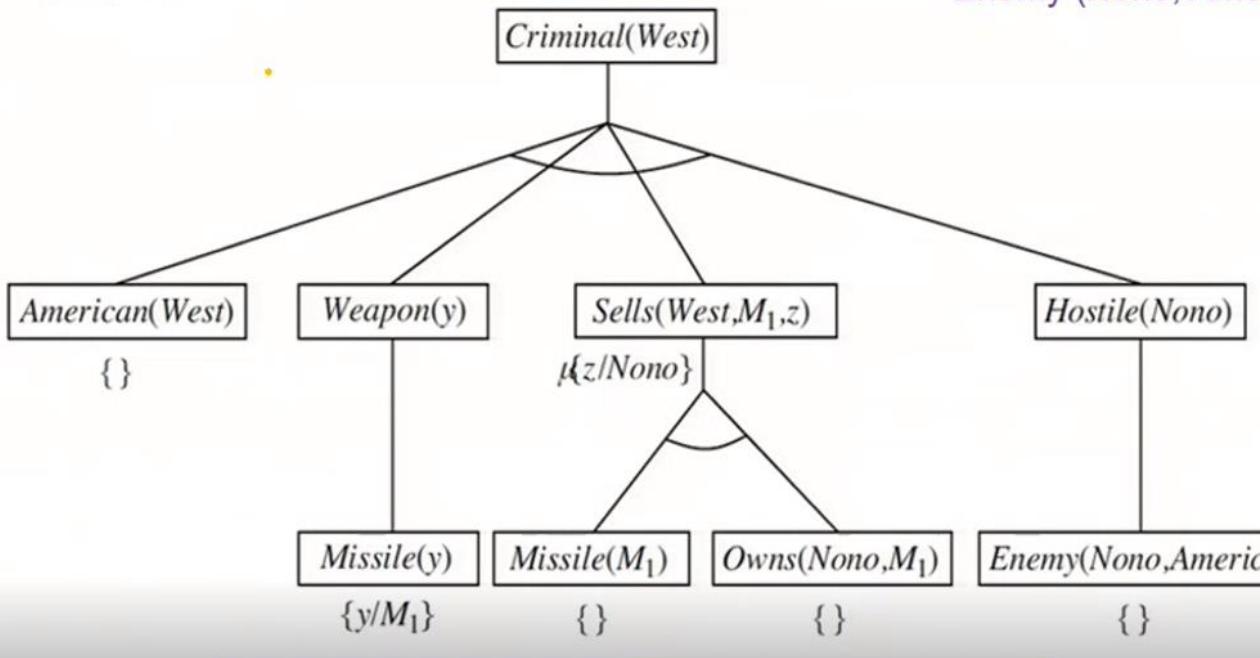
# Backward Chaining

- ①  $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Sells}(x, y, z) \wedge \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$   
 ②  $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$   
 ③  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$   
 ④  $\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x)$
- Missle(M1)  
 Owns(Nono, M1)  
 American (West)  
 Enemy (Nono, America)



# Backward Chaining

- ①  $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- ②  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$  Missle(M1)
- ③  $Missile(x) \Rightarrow Weapon(x)$  Owns(Nono, M1)
- ④  $Enemy(x, America) \Rightarrow Hostile(x)$  American (West)  
Enemy (Nono, America)





# Reasoning

## Module 5:

# Probabilistic Representation and Reasoning

A. Inference using full joint distribution

B. Bayesian Networks

I. Knowledge Representation

II. Conditional Independence

III. Exact Inference

IV. Introduction to Approximate Inference

- Monotonic Reasoning
- Non- Monotonic Reasoning

Dependency Directed Backtracking: when a statement is deleted as “ no more valid”, other related statements have to be backtracked and they should be either deleted or new proofs have to be found for them. This is called dependency directed backtracking (DDB)

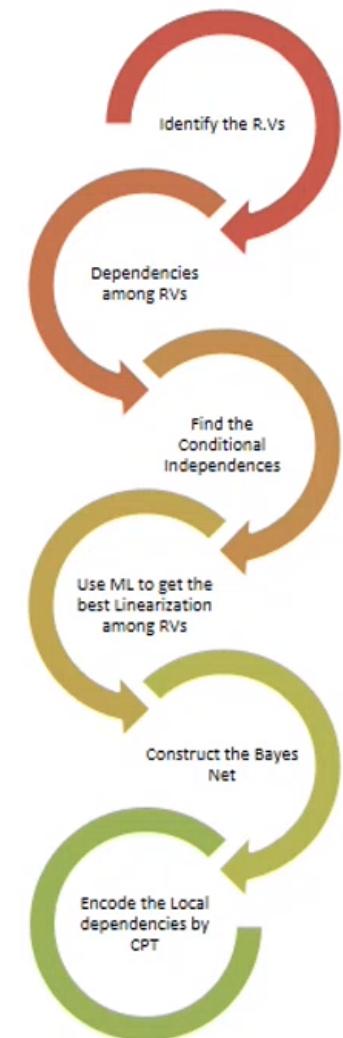
- Monotonic Reasoning
- Non- Monotonic Reasoning

Monotonic	Non-Monotonic
Consistent	Relaxed Consistency
Complete Knowledge	Incomplete Knowledge
Static	Dynamic
Discrete	Continuous & Learning Agent
Predicate Logic	Probabilistic Model

# Bayesian Net???

## Wumpus World Problem

- Wumpus Ghost traces of scent in the visited cell
- Earlier visited cell may become unsafe!!!
- **Problem:** Given the information that there is a possibility of apparition of Wumpus anywhere in the cave, AI agent needs to be safely travel with more caution!!



## Uncertainty

You can reach Bangalore Airport from MG Road within 90 mins if you go by route A.

- There is uncertainty in this information due to partial observability and non determinism
- Agents should handle such uncertainty

Previous approaches like Logic represent all possible world states

Such approaches can't be used as multiple possible states need to be enumerated to handle the uncertainty in our information

## Uncertainty

You can reach Bangalore Airport from MG Road within 90 mins if you go by route A.

Road Block	Festival Season	Weekend	Observation (20)	Prob
F	F	F	12	0.6
F	F	T	3	0.15
F	T	F	2	0.1
F	T	T	2	0.1
T	F	F	0	0
T	F	T	0	0
T	T	F	1	0.05
T	T	T	0	0
				=1

# Probability Theory

Basics

Conditional Probability

Chain Rule

Independence

Conditional Independence

Belief Nets

Joint Probability distribution

## Probability Basics - Refresher

---

**Sample Space:** Set of all possible outcomes.

- Ex: After tossing 2 coins, the set of all possible outcomes are
- $\{HH, HT, TH, TT\}$

**Event:** A subset of a sample space.

- An event of interest might be -  $\{HH\}$

## Probability Basics - Model

---

A fully specified **probability model** associates a numerical probability  $P(\omega)$  with each possible world.

### The **basic axioms**

1. Every possible world has a probability between 0 and 1
2. Sum of probabilities of possible worlds is 1  $P(\text{True}) = 1$   
 $P(\text{False}) = 0$
3.  $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

E.g.,  $P(HH) = 0.25$ ;  $P(HT) = 0.25$ ;  $P(TT) = 0.25$ ,  $P(TH) = 0.25$

$$0 \leq P(\omega) \leq 1 \text{ for every } \omega \text{ and } \sum_{\omega \in \Omega} P(\omega) = 1$$

# Probability Basics – Exclusive / Exhaustive events

---

## Mutually Exclusive Events:

- Two events (or propositions) are mutually exclusive or disjoint if they cannot both occur at the same time (be true).
- A clear example is the set of outcomes of a single coin toss, which can result in either heads or tails, but not both.

## Exhaustive Events:

- A set of events is jointly or collectively exhaustive if at least one of the events must occur.
- E.g., when rolling a six-sided die, the events 1, 2, 3, 4, 5, and 6 are collectively exhaustive.

## Probability Basics - Propositions

---

Probabilistic assertions (Propositions)

- Usually not a particular world event but about a set of them
- E.g., two dice when rolled, a proposition  $\varphi$  can be “the sum of dice is 11”

For any proposition  $\varphi$ ,

$$\begin{aligned} P(\varphi) &= P(\text{sum} = 11) &= P((5, 6)) + P((6, 5)) \\ &&= 1/36 + 1/36 = 1/18 \end{aligned}$$

## Probability Basics – Unconditional/Prior

---

Unconditional / Prior probabilities:

Propositions like  $P(\text{sum} = 11)$  or  $P(\text{two dices rolling equals})$  are called  
Unconditional or Prior probabilities

They refer to degree of belief in absence of any other information

$$P(a | b) = \frac{P(a \wedge b)}{P(b)}$$

$$P(a \wedge b) = P(a | b)P(b)$$

## Probability Basics - Conditional

---

However, most of the time we have some information, we call it **evidence**

E.g., we can interested in two dice rolling a double (i.e., 1,1 or 2,2, etc)

When one die has rolled 5 and the other die is still spinning

Here, we not interested in unconditional probability of rolling a double

Instead, the **conditional** or **posterior** probability for rolling a double given the first die has rolled a 5

$P(\text{doubles} \mid \text{Die}_1 = 5)$  where  $\mid$  is pronounced “given”

E.g., if you are going for a dentist for a checkup,  $P(\text{cavity}) = 0.2$

- If you have a toothache, then  $P(\text{cavity} \mid \text{toothache}) = 0.6$

## Independence

---

If we have two random variables, TimeToBnlrAirport and HyderabadWeather

$P(\text{TimeToBnlrAirport}, \text{HyderabadWeather})$

To determine their relation, use the product rule

$$= P(\text{TimeToBnlrAirport} | \text{HyderabadWeather}) / P(\text{HyderabadWeather})$$

However, we would argue that HyderabadWeather and TimeToBnlrAirport doesn't have any relation and hence

$$P(\text{TimeToBnlrAirport} | \text{HyderabadWeather}) = P(\text{TimeToBnlrAirport})$$

This is called Independence or Marginal Independence

Independence between propositions a and b can be written as

$$P(a | b) = P(a) \quad \text{or} \quad P(b | a) = P(b) \quad \text{or} \quad P(a \wedge b) = P(a)P(b)$$

## Bayes Rule

---

Using the product rule for propositions a and b

$$P(a \wedge b) = P(a | b)P(b) \quad \text{and} \quad P(a \wedge b) = P(b | a)P(a)$$

Equating the right hand sides and dividing by  $P(a)$

$$P(b | a) = \frac{P(a | b)P(b)}{P(a)}$$

This is called the Bayes Rule

# Conditional Independence

2 random variables A and B are conditionally independent given C iff

$$P(a, b | c) = P(a | c) P(b | c) \text{ for all values } a, b, c$$

More intuitive (equivalent) conditional formulation

- A and B are conditionally independent given C iff

$$P(a | b, c) = P(a | c) \text{ OR } P(b | a, c) = P(b | c), \text{ for all values } a, b, c$$

- Intuitive interpretation:

**P(a | b, c) = P(a | c) tells us that learning about b, given that we already know c, provides no change in our probability for a, i.e., b contains no information about a beyond what c provides**

$$P(R | F, P) = P(R | P)$$

## Joint Probability Distributions

---

Instead of distribution over single variable, we can model distribution over multiple variables, separated by comma

E.g.,  $P(A, B) = P(A | B) . P(B)$

$P(A, B)$  is the probability distribution over combination of all values of A and B

E.g., if A = Weather and B = Cavity

$$\begin{aligned}P(W = \text{sunny} \wedge C = \text{true}) &= P(W = \text{sunny}|C = \text{true}) P(C = \text{true}) \\P(W = \text{rain} \wedge C = \text{true}) &= P(W = \text{rain}|C = \text{true}) P(C = \text{true}) \\P(W = \text{cloudy} \wedge C = \text{true}) &= P(W = \text{cloudy}|C = \text{true}) P(C = \text{true}) \\P(W = \text{snow} \wedge C = \text{true}) &= P(W = \text{snow}|C = \text{true}) P(C = \text{true}) \\P(W = \text{sunny} \wedge C = \text{false}) &= P(W = \text{sunny}|C = \text{false}) P(C = \text{false}) \\P(W = \text{rain} \wedge C = \text{false}) &= P(W = \text{rain}|C = \text{false}) P(C = \text{false}) \\P(W = \text{cloudy} \wedge C = \text{false}) &= P(W = \text{cloudy}|C = \text{false}) P(C = \text{false}) \\P(W = \text{snow} \wedge C = \text{false}) &= P(W = \text{snow}|C = \text{false}) P(C = \text{false}) .\end{aligned}$$

# Probabilistic Inference

Computation of posterior probabilities given observed evidence, i.e., full joint probability distribution

		toothache		$\neg$ toothache	
		catch	$\neg$ catch	catch	$\neg$ catch
<i>cavity</i>	0.108	0.012	0.072	0.008	
$\neg$ cavity	0.016	0.064	0.144	0.576	

**Query:  $P(\text{cavity} \vee \text{toothache})$**

$$0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$$

# Conditional Probability

Towards Chain Rule:

$$P(a | b) = P(a,b) / P(b)$$

$$P(a, b) = P(a | b) P(b)$$

$$P(a, b, c) = P(a, x) \text{ where } x = b, c$$

$$P(a,x) = P(a | x) . P(x)$$

$$= P(a | bc) . P(b, c)$$

$$= P(a | bc) . P(b | c) . P(c)$$

$$\text{Hence : } P(a,b,c) = P(a | bc) . P(b | c) . P(c)$$

Chain Rule : Generalization

$$P(X_1, X_2, \dots, X_k) = \prod P(X_i | X_{i-1}, \dots, X_1)$$

Where  $i = k \text{ to } 1$  (reverse)

# Probability Theory

## Independence

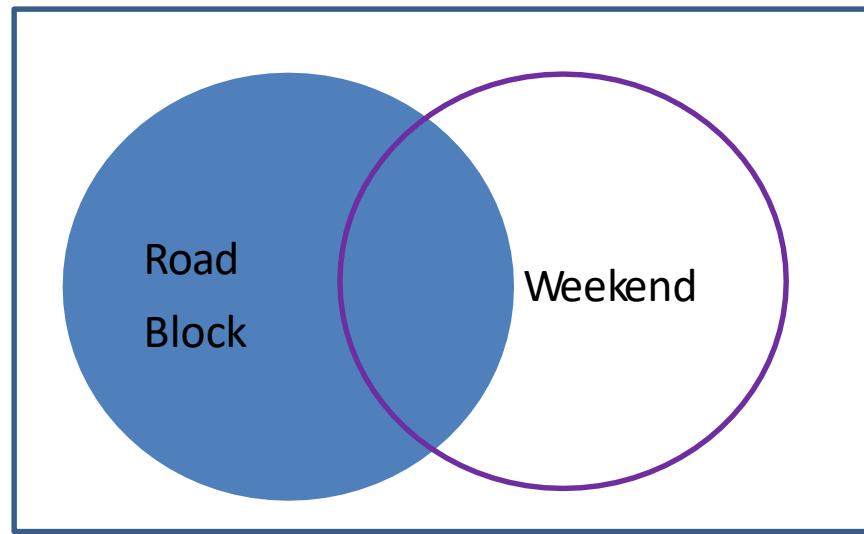
$$P(a | b) = P(a)$$

Implication:

$$P(a | b) = P(a,b) / P(b)$$

$$P(a) = P(a,b) / P(b)$$

$$\mathbf{P(a,b) = P(a) . P(b)}$$



## Conditional Independence

$$P(a | b c) = P(a | c)$$

# Probability Theory

## Conditional Independence

$$P(a | b c) = P(a | c)$$

Extension:

$$P(a b | c) = P(a | c) \cdot P(b | c)$$

---

## Required Reading: AIMA - Chapter # 4.1, #4.2, #5.1, #9

### Next Session Plan:

- (Prerequisite Reading : Refresh the basics of probability , Bayes Theorem , Conditional Probability, Product Rule, Conditional Independence, Chain Rule)
  - Inferences using Logic ( Forward / Backward Chaining / DPLL algorithm)
  - Bayesian Network
  - Representation
  - Inferences (Exact and approximate-only Direct sampling)
- Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

## DSECLZG557

**BITS** Pilani  
Pilani Campus

Indumathi V  
Guest Faculty  
BITS -WILP

## Probabilistic Representation and Reasoning

### A. Inference using full joint distribution

### B. Bayesian Networks

I. Knowledge Representation

II. Conditional Independence

III. Exact Inference

IV. Introduction to Approximate Inference

least info

KB  
fact

Time

A to D.  $\rightarrow$  A B C D

blocked

fact

→ false

① Delete

Non-Markovian

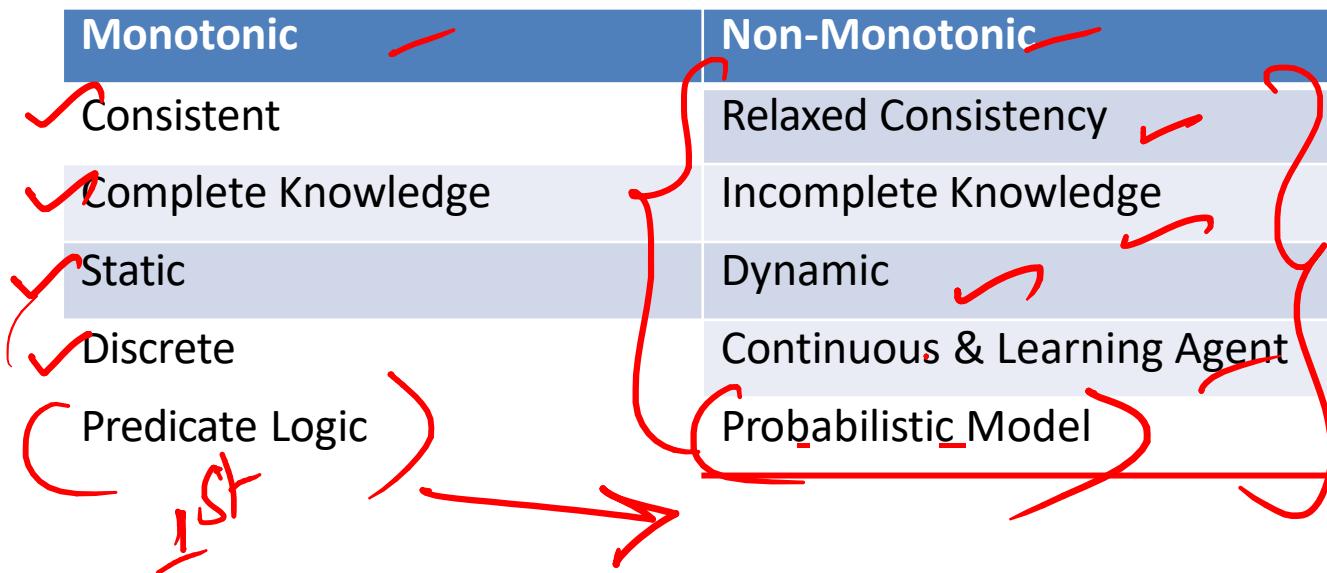
- Monotonic Reasoning

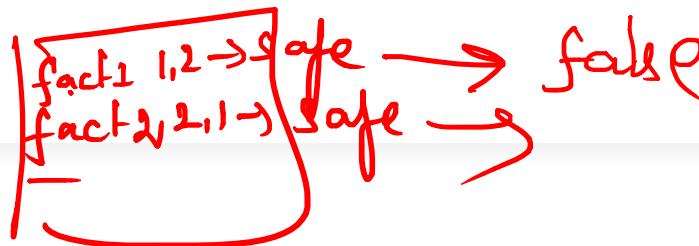
- Non-Monotonic Reasoning

Dependency Directed Backtracking: when a statement is deleted as “no more valid”, other related statements have to be backtracked and they should be either deleted or new proofs have to be found for them. This is called dependency directed backtracking (DDB)

- Monotonic Reasoning
- Non- Monotonic Reasoning



## Bayesian Net???



### Wumpus World Problem

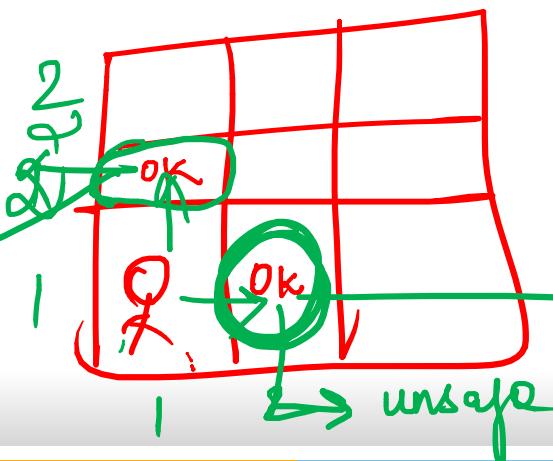
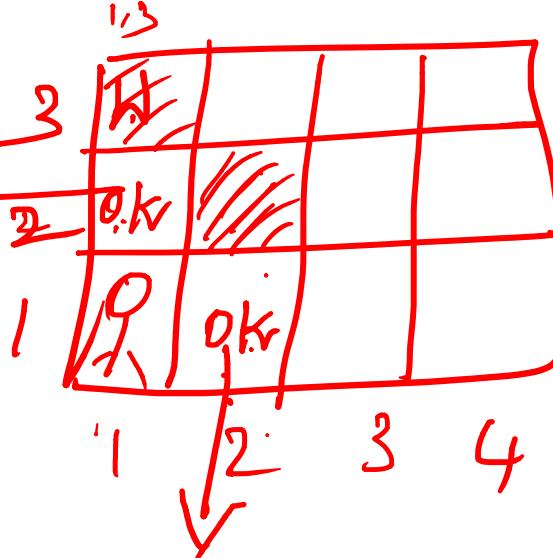
- Wumpus Ghost traces of scent in the visited cell
- Earlier visited cell may become unsafe!!!
- **Problem:** Given the information that there is a possibility of apparition of Wumpus anywhere in the cave, AI agent needs to be safely travel with more caution!!

Loc → Wumpus → 1,3 →

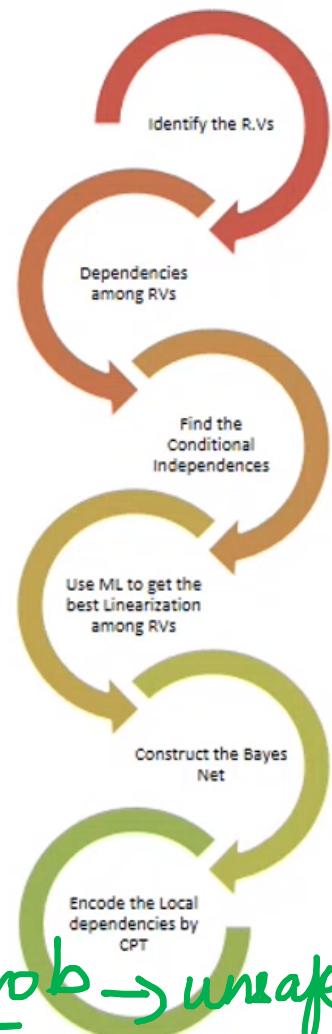
Agent → Loc

Wumpus ↗

unsafe ← 40%  
 unsafe ← 0.4  
 Safe → 60%



20% → 0.2 prob → unsafe  
 80% → 0.8 prob → safe.



## Uncertainty

You can reach Bangalore Airport from MG Road within 90 mins if you go by route A.

- There is uncertainty in this information due to partial observability and non determinism
- Agents should handle such uncertainty

Previous approaches like Logic represent all possible world states

Such approaches can't be used as multiple possible states need to be enumerated to handle the uncertainty in our information

## Uncertainty

You can reach Bangalore Airport from MG Road within 90 mins if you go by route A.

Road Block	Festival Season	Weekend	Observation (20)	Prob
F	F	F	12	0.6 ✓
F ✓	F ✓	T ✓	3 / 20	0.15 ✓
F	T	F	2	0.1
F	T	T	2	0.1
T	F	F	0	0
T	F	T	0	0
T	T	F	1	0.05
T	T	T	0	0
				= 1

Annotations in blue:

- $f_1$ ,  $f_2$ ,  $f_3$  are circled above the first three columns.
- "absent" is written next to the first row under "Road Block".
- "12/20" is written next to the value 12 in the "Observation (20)" column.
- A bracket on the left groups the first three rows.
- A bracket on the right groups the last four rows.
- A large bracket at the bottom groups all rows.
- The word "Compressed Data." is written diagonally across the bottom right.

# Probability Theory

---

**Basics**

**Conditional Probability**

**Chain Rule**

**Independence**

**Conditional Independence**

**Belief Nets**

**Joint Probability distribution**

## Probability Basics - Refresher

---

**Sample Space:** Set of all possible outcomes.

- Ex: After tossing 2 coins, the set of all possible outcomes are
- $\{HH, HT, TH, TT\}$

**Event:** A subset of a sample space.

- An event of interest might be -  $\{HH\}$

## Probability Basics - Model

---

A fully specified **probability model** associates a numerical probability  $P(\omega)$  with each possible world.

### The **basic axioms**

1. Every possible world has a probability between 0 and 1
2. Sum of probabilities of possible worlds is 1  $P(\text{True}) = 1$   
 $P(\text{False}) = 0$
3.  $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$

E.g.,  $P(HH) = 0.25$ ;  $P(HT) = 0.25$ ;  $P(TT) = 0.25$ ,  $P(TH) = 0.25$

$$0 \leq P(\omega) \leq 1 \text{ for every } \omega \text{ and } \sum_{\omega \in \Omega} P(\omega) = 1$$

# Probability Basics – Exclusive / Exhaustive events

---

## Mutually Exclusive Events:

- Two events (or propositions) are mutually exclusive or disjoint if they cannot both occur at the same time (be true).
- A clear example is the set of outcomes of a single coin toss, which can result in either heads or tails, but not both.

## Exhaustive Events:

- A set of events is jointly or collectively exhaustive if at least one of the events must occur.
- E.g., when rolling a six-sided die, the events 1, 2, 3, 4, 5, and 6 are collectively exhaustive.

## Probability Basics - Propositions

---

Probabilistic assertions (Propositions)

- Usually not a particular world event but about a set of them
- E.g., two dice when rolled, a proposition  $\varphi$  can be “the sum of dice is 11”

For any proposition  $\varphi$ ,

$$\begin{aligned} P(\varphi) &= P(\text{sum} = 11) &= P((5, 6)) + P((6, 5)) \\ &&= 1/36 + 1/36 = 1/18 \end{aligned}$$

## Probability Basics – Unconditional/Prior

---

Unconditional / Prior probabilities:

Propositions like  $P(\text{sum} = 11)$  or  $P(\text{two dices rolling equals})$  are called  
Unconditional or Prior probabilities

They refer to degree of belief in absence of any other information

$$P(a | b) = \frac{P(a \wedge b)}{P(b)}$$

$$P(a \wedge b) = P(a | b)P(b)$$

## Probability Basics - Conditional

---

However, most of the time we have some information, we call it **evidence**

E.g., we can interested in two dice rolling a double (i.e., 1,1 or 2,2, etc)

When one die has rolled 5 and the other die is still spinning

Here, we not interested in unconditional probability of rolling a double

Instead, the **conditional** or **posterior** probability for rolling a double given the first die has rolled a 5

$P(\text{doubles} \mid \text{Die}_1 = 5)$  where  $\mid$  is pronounced “given”

E.g., if you are going for a dentist for a checkup,  $P(\text{cavity}) = 0.2$

- If you have a toothache, then  $P(\text{cavity} \mid \text{toothache}) = 0.6$

## Independence

---

If we have two random variables, TimeToBnlrAirport and HyderabadWeather

$P(\text{TimeToBnlrAirport}, \text{HyderabadWeather})$

To determine their relation, use the product rule

$$= P(\text{TimeToBnlrAirport} | \text{HyderabadWeather}) / P(\text{HyderabadWeather})$$

However, we would argue that HyderabadWeather and TimeToBnlrAirport  
doesn't have any relation and hence

$$P(\text{TimeToBnlrAirport} | \text{HyderabadWeather}) = P(\text{TimeToBnlrAirport})$$

This is called Independence or Marginal Independence

Independence between propositions a and b can be written as

$$P(a | b) = P(a) \quad \text{or} \quad P(b | a) = P(b) \quad \text{or} \quad P(a \wedge b) = P(a)P(b)$$

## Bayes Rule

---

Using the product rule for propositions a and b

$$P(a \wedge b) = P(a | b)P(b) \quad \text{and} \quad P(a \wedge b) = P(b | a)P(a)$$

Equating the right hand sides and dividing by  $P(a)$

$$P(b | a) = \frac{P(a | b)P(b)}{P(a)}$$

This is called the Bayes Rule

# Conditional Independence

2 random variables A and B are conditionally independent given C iff

$$P(a, b | c) = P(a | c) P(b | c) \text{ for all values } a, b, c$$

More intuitive (equivalent) conditional formulation

- A and B are conditionally independent given C iff

$$P(a | b, c) = P(a | c) \text{ OR } P(b | a, c) = P(b | c), \text{ for all values } a, b, c$$

- Intuitive interpretation:

**P(a | b, c) = P(a | c) tells us that learning about b, given that we already know c, provides no change in our probability for a, i.e., b contains no information about a beyond what c provides**

$$P(R | F, P) = P(R | P)$$

## Joint Probability Distributions

---

Instead of distribution over single variable, we can model distribution over multiple variables, separated by comma

E.g.,  $P(A, B) = P(A | B) . P(B)$

$P(A, B)$  is the probability distribution over combination of all values of A and B

E.g., if A = Weather and B = Cavity

$$\begin{aligned}P(W = \text{sunny} \wedge C = \text{true}) &= P(W = \text{sunny}|C = \text{true}) P(C = \text{true}) \\P(W = \text{rain} \wedge C = \text{true}) &= P(W = \text{rain}|C = \text{true}) P(C = \text{true}) \\P(W = \text{cloudy} \wedge C = \text{true}) &= P(W = \text{cloudy}|C = \text{true}) P(C = \text{true}) \\P(W = \text{snow} \wedge C = \text{true}) &= P(W = \text{snow}|C = \text{true}) P(C = \text{true}) \\P(W = \text{sunny} \wedge C = \text{false}) &= P(W = \text{sunny}|C = \text{false}) P(C = \text{false}) \\P(W = \text{rain} \wedge C = \text{false}) &= P(W = \text{rain}|C = \text{false}) P(C = \text{false}) \\P(W = \text{cloudy} \wedge C = \text{false}) &= P(W = \text{cloudy}|C = \text{false}) P(C = \text{false}) \\P(W = \text{snow} \wedge C = \text{false}) &= P(W = \text{snow}|C = \text{false}) P(C = \text{false}) .\end{aligned}$$

# Probabilistic Inference

Computation of posterior probabilities given observed evidence, i.e., full joint probability distribution

	toothache		$\neg$ toothache	
	<i>catch</i>	$\neg$ <i>catch</i>	<i>catch</i>	$\neg$ <i>catch</i>
<i>cavity</i>	0.108	0.012	0.072	0.008
$\neg$ <i>cavity</i>	0.016	0.064	0.144	0.576

**Query:  $P(\text{cavity} \vee \text{toothache})$**

$$0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$$

# Conditional Probability

---

Towards Chain Rule:

$$P(a | b) = P(a,b) / P(b)$$

$$P(a, b) = P(a | b) P(b)$$

$$P(a, b, c) = P(a, x) \text{ where } x = b, c$$

$$P(a,x) = P(a | x) . P(x)$$

$$= P(a | bc) . P(b, c)$$

$$= P(a | bc) . P(b | c) . P(c)$$

$$\text{Hence : } P(a,b,c) = P(a | bc) . P(b | c) . P(c)$$

Chain Rule : Generalization

$$P(X_1, X_2, \dots, X_k) = \prod P(X_i | X_{i-1}, \dots, X_1)$$

Where  $i = k \text{ to } 1$  (reverse)

# Probability Theory

## Independence

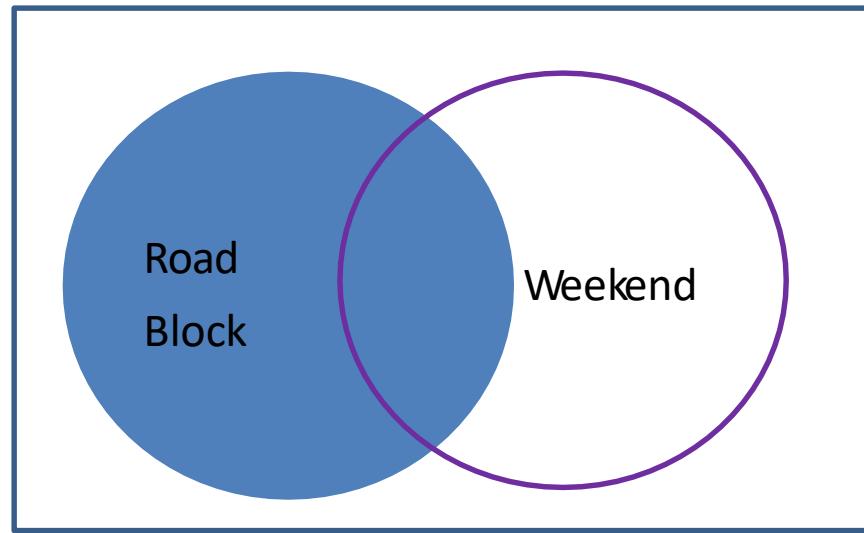
$$P(a | b) = P(a)$$

Implication:

$$P(a | b) = P(a,b) / P(b)$$

$$P(a) = P(a,b) / P(b)$$

$$\mathbf{P(a,b) = P(a) . P(b)}$$



## Conditional Independence

$$P(a | b c) = P(a | c)$$

# Probability Theory

## Conditional Independence

$$P(a | b c) = P(a | c)$$

Extension:

$$P(a b | c) = P(a | c) \cdot P(b | c)$$

20 → 20L    0,0,0.05

R, FS, W

Condition

$P(R|W)$

$P(R|FS)$

CPT →

## Building a Bayesian Network

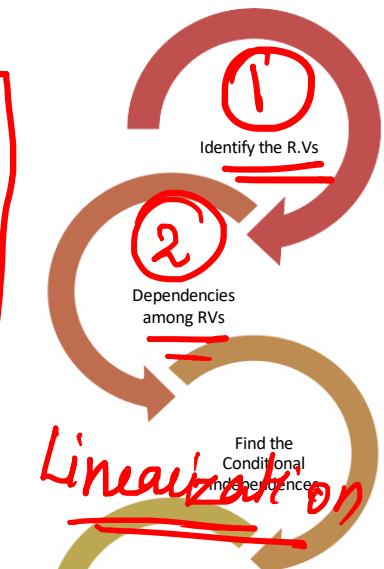
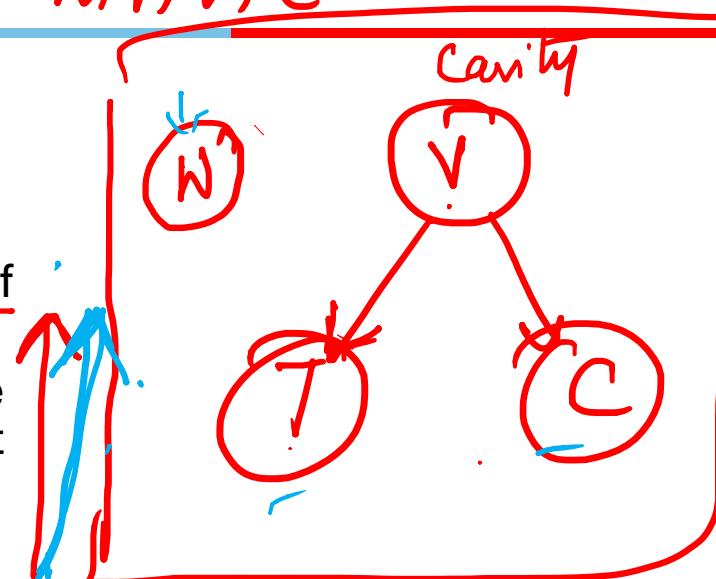
<u>W</u>	<u><math>P(R)</math></u>	<u><math>P(uR)</math></u>
T.	-	-
F	-	-

## Example Bayesian Net #1

$W, T, V, C$

A simple world with four random variables

- ✓ { Weather, Toothache, Cavity, Catch }  $\rightarrow$
- Weather is independent of other variables
- Toothache and Catch are conditionally independent given Cavity
- $P(\text{Toothache}, \text{Catch} | \text{Cavity}) = P(\text{Toothache} | \text{Cavity}) \cdot P(\text{Catch} | \text{Cavity})$
- ✓ Cavity is a direct cause of Toothache and Catch
- No direct relation between Toothache and Catch exists



Bayesian N/W

child should proceed first

Paren

- ③  $W T C V$
- ④  $W C T V$



②

$T C V W$

$T C V W$

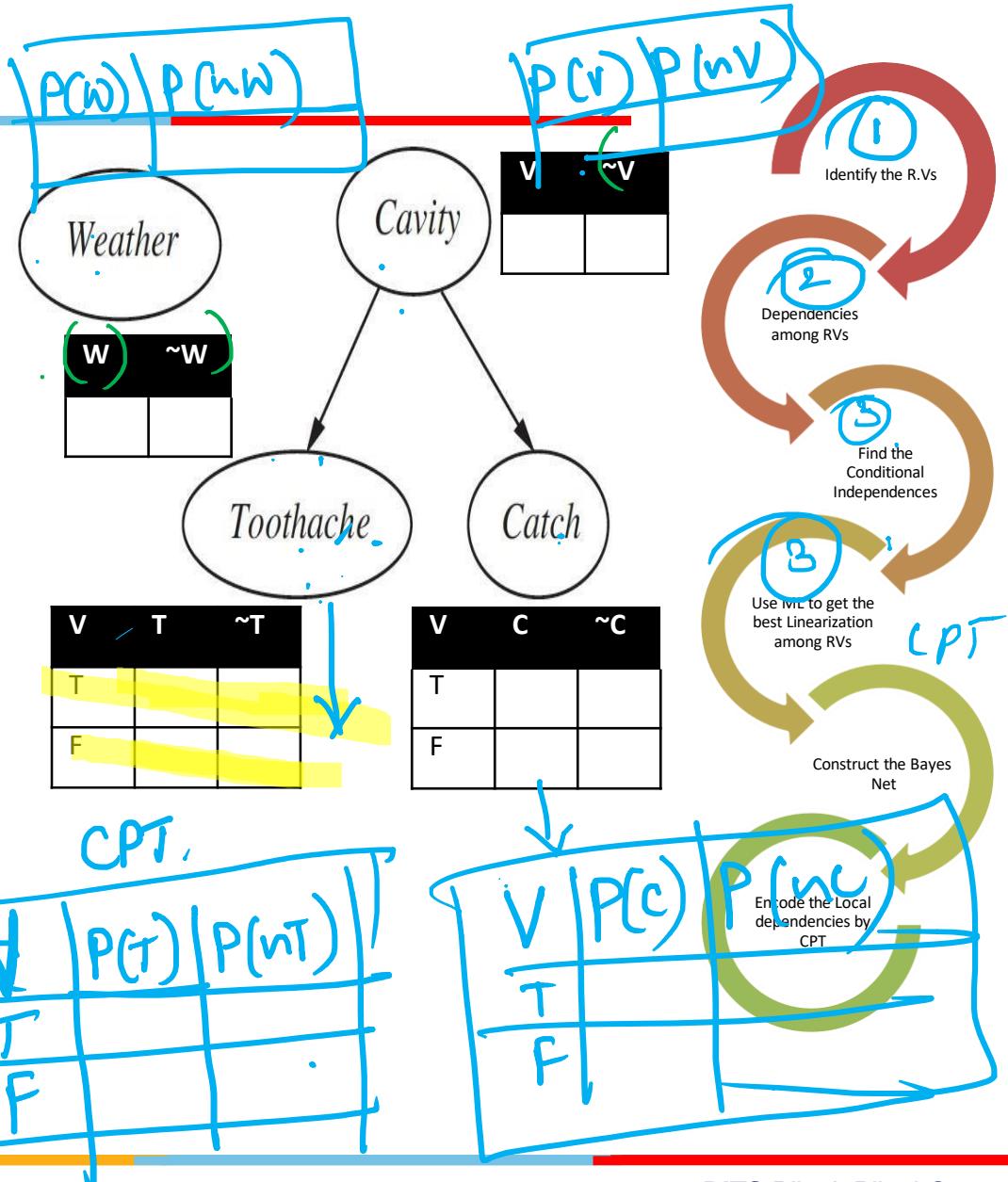
## Example Bayesian Net #1

A simple world with four random variables

- Weather, Toothache, Cavity, Catch
- Weather is independent of other variables
- Toothache and Catch are conditionally independent given Cavity
- $P(\text{Toothache}, \text{Catch} | \text{Cavity}) = P(\text{Toothache} | \text{Cavity}) \cdot P(\text{Catch} | \text{Cavity})$
- Cavity is a direct cause of Toothache and Catch
- No direct relation between Toothache and Catch exists

$$P(T|c)$$

		$P(T)$		$P(\neg T)$	
		T	F	T	F
	T	?			
	F				



T C V W

child Parent



## Example Bayesian Net #1

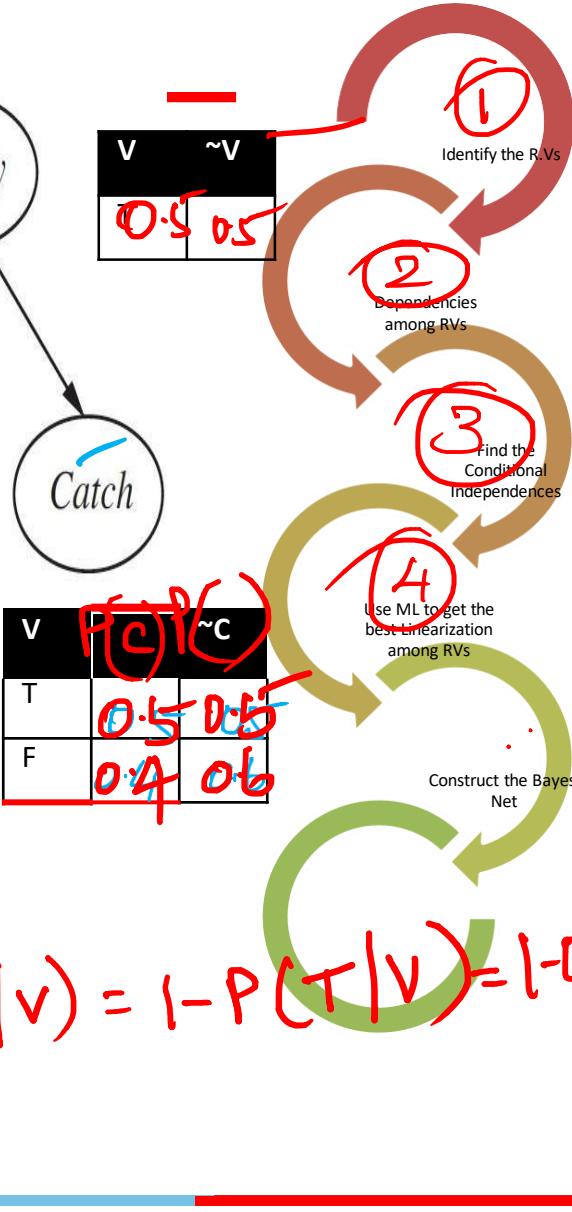
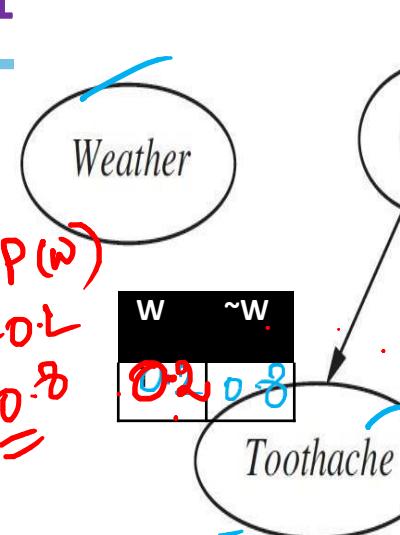
A simple world with four random variables

- Weather, Toothache, Cavity, Catch
- Weather is independent of other variables
- Toothache and Catch are conditionally independent given Cavity
- $P(\text{Toothache}, \text{Catch} | \text{Cavity}) = P(\text{Toothache} | \text{Cavity}) \cdot P(\text{Catch} | \text{Cavity})$
- Cavity is a direct cause of Toothache and Catch
- No direct relation between Toothache and Catch exists

$$\begin{aligned} P(\text{nw}) &= 1 - P(\text{w}) \\ &= 1 - 0.2 \\ &\Rightarrow 0.8 \end{aligned}$$

		CPT	
		V	$\neg V$
		T	$\neg T$
		T	0.6
		F	0.2
		V	0.4
		F	0.3

		CPT	
		V	$\neg V$
		T	$\neg T$
		T	0.5
		F	0.4
		V	0.5
		F	0.6



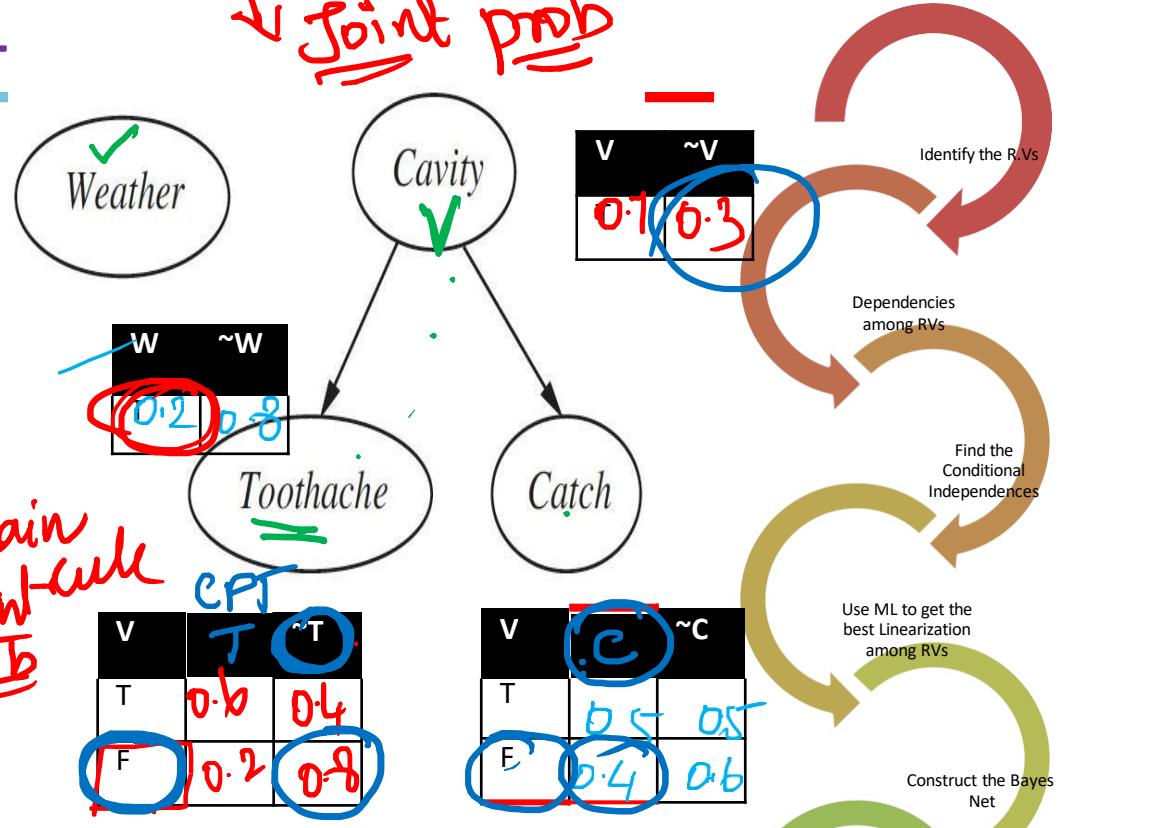
Query:  $P(w, nv, nt, c)$  → check proper linearization format

↓ joint prob

## Example Bayesian Net #1

A simple world with four random variables

- ~~$w \wedge v \wedge t \wedge c \rightarrow w \wedge t \wedge c \wedge v$~~
- Weather, Toothache, Cavity, Catch
- Weather is independent of other variables
- Toothache and Catch are conditionally independent given Cavity
- $P(\text{Toothache}, \text{Catch} | \text{Cavity}) = P(\text{Toothache} | \text{Cavity}) \cdot P(\text{Catch} | \text{Cavity})$
- Cavity is a direct cause of Toothache and Catch
- No direct relation between Toothache and Catch exists

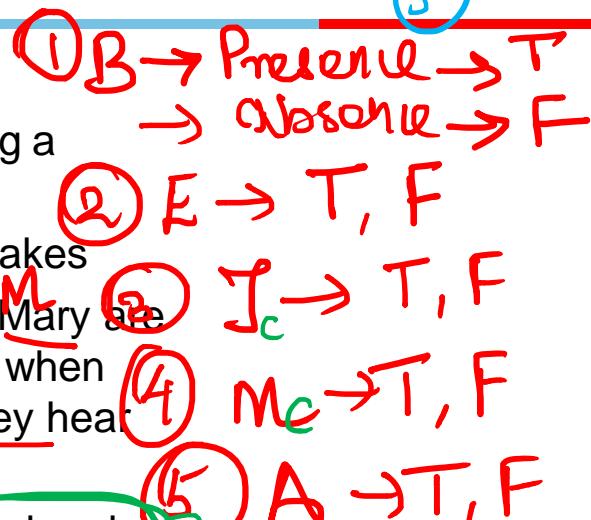


$$\begin{aligned}
 P(w, nv, nt, c) &\Rightarrow \\
 P(w | nt \wedge nv) \cdot P(nt | nv) \cdot P(c | nv) \cdot P(nv) &\Rightarrow \\
 P(w) \cdot P(nt | nv) \cdot P(c | nv) \cdot P(nv) &\Rightarrow \\
 0.2 * 0.8 * 0.4 * 0.3 &\Rightarrow 0.0192 //
 \end{aligned}$$

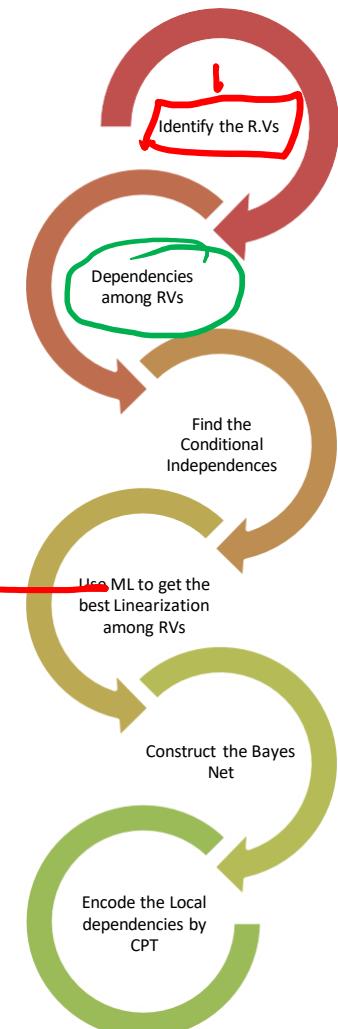
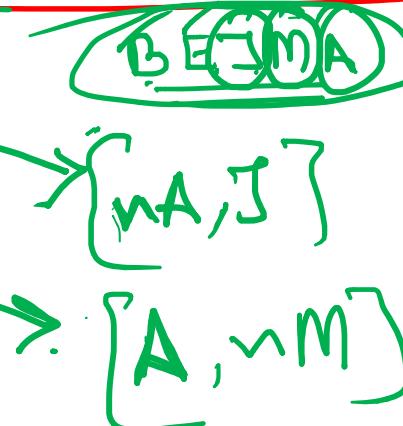
## Example Bayesian Net #2

### A Burglary Alarm System

- Fairly reliable on detecting a burglary
- Also responds to earthquakes
- Two neighbors John and Mary are asked to call you at work when Burglary happens and they hear the Alarm  $A$



- John nearly always calls when he hears the alarm, however sometimes confuses the telephone ring with alarm and calls then too
- Mary likes loud music and often misses the alarm altogether
- Problem:** Given the information that who has / has not called we need to estimate the probability of a burglary

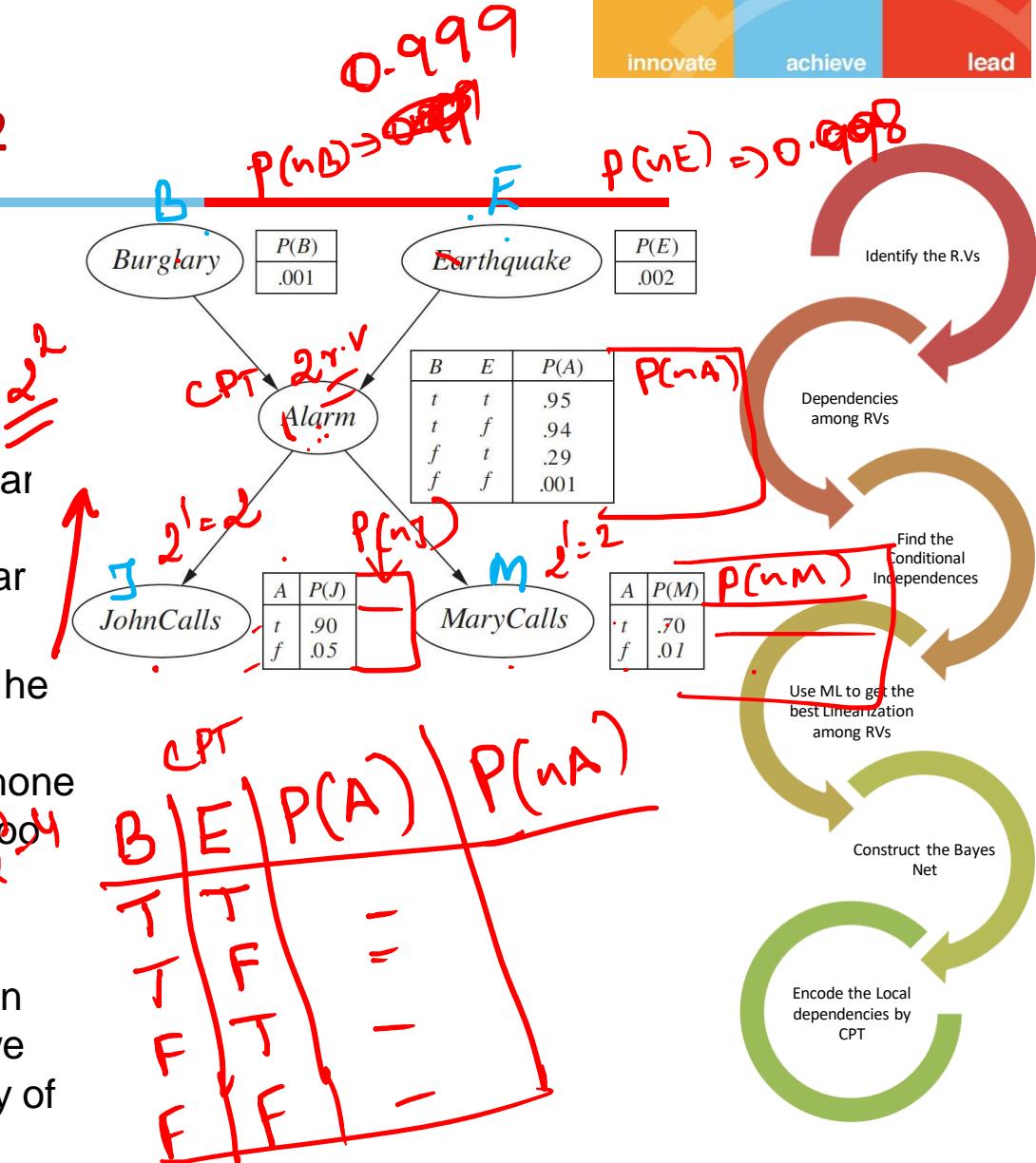


JMABE

# Example Bayesian Net #2

## A Burglary Alarm System

- Fairly reliable on detecting a burglary
  - Also responds to earthquakes
  - Two neighbors John and Mary are asked to call you at work when Burglary happens and they hear the Alarm
  - John nearly always calls when he hears the alarm, however sometimes confuses the telephone ring with alarm and calls then too
  - Mary likes loud music and often misses the alarm altogether
  - **Problem:** Given the information that who has / has not called we need to estimate the probability of a burglary



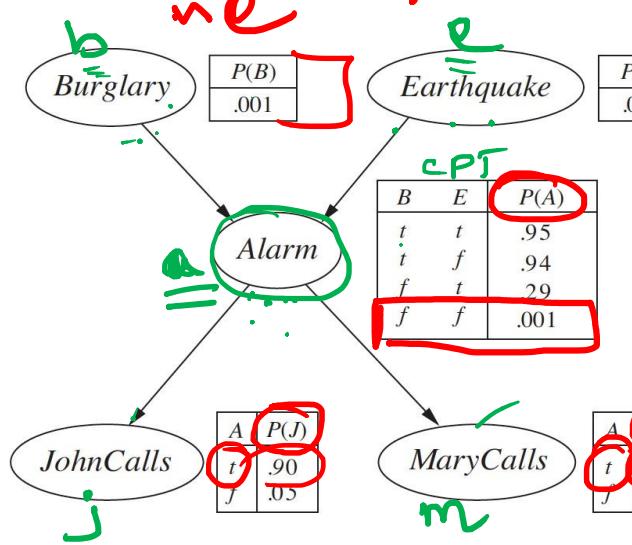
P(jmanbne) →

Query

a

nb

1. Calculate the probability that alarm has sounded, but neither burglary nor earthquake happened, and both John and Mary called



$$P(j, m, a, \neg b, \neg e) = P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e)$$

chain rule

$$\begin{aligned}
 & P(j|m a \neg b \neg e) \cdot P(m|a \neg b \neg e) \cdot P(a|\neg b \neg e) \\
 & \quad \downarrow \quad \downarrow \quad \downarrow \\
 & P(j|a) \cdot P(m|a) \cdot P(a|\neg b \neg e) \cdot P(\neg b) \cdot P(\neg e) \\
 & 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 = 0.000628
 \end{aligned}$$

⇒ 0.000628

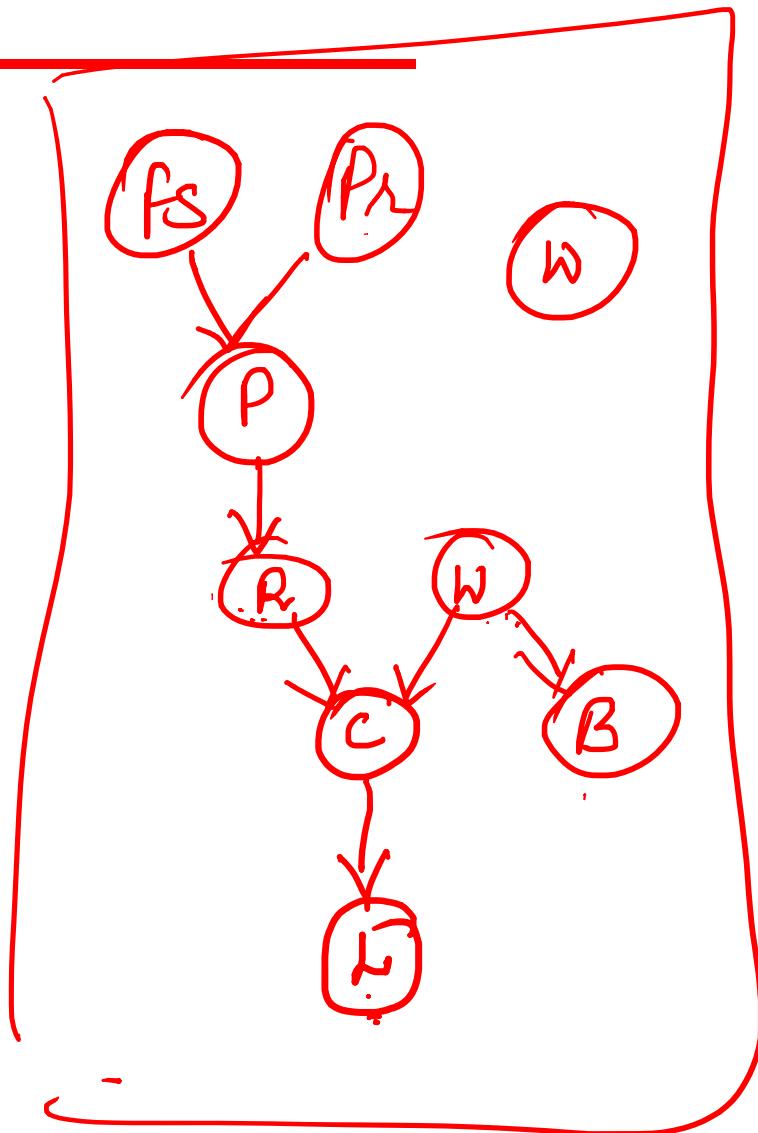
## Example Bayesian Net #3

Case study :-

Step1: R.V

### Traffic Prediction -Travel Estimation

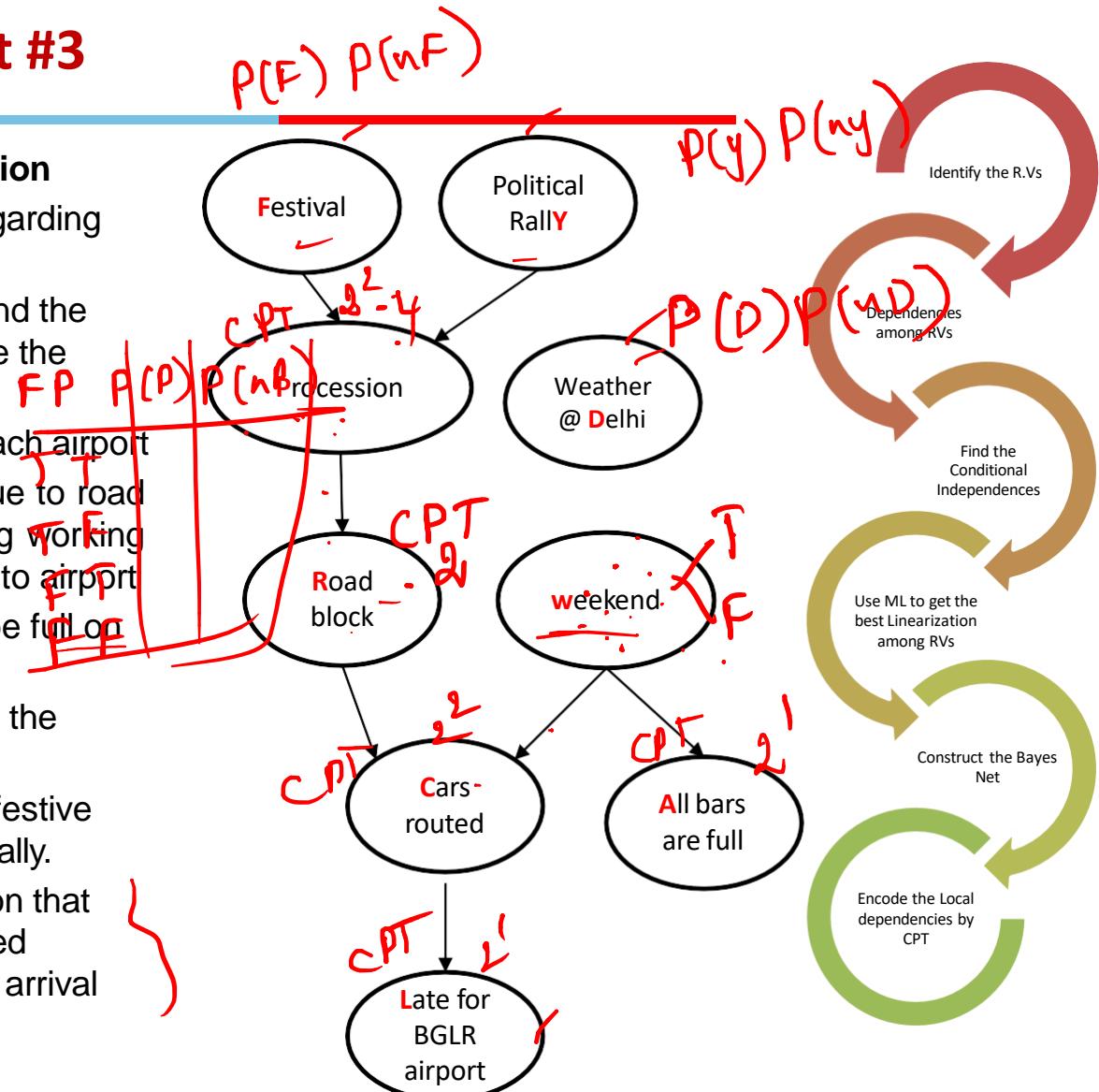
- AI system reminds traveler regarding start time
- Travel plan is to reach Delhi and the weather of Delhi may influence the accommodation plans
- Traveler always take car to reach airport
- Car may be rerouted either due to road block or weekday traffic during working hours which delays the arrival to airport
- Bars are always observed to be full on weekends
- Authorities block roads to safe the processions
- Processions observed during festive season or due to the political rally.
- **Problem:** Given the information that there is a political rally expected estimate the probability of late arrival



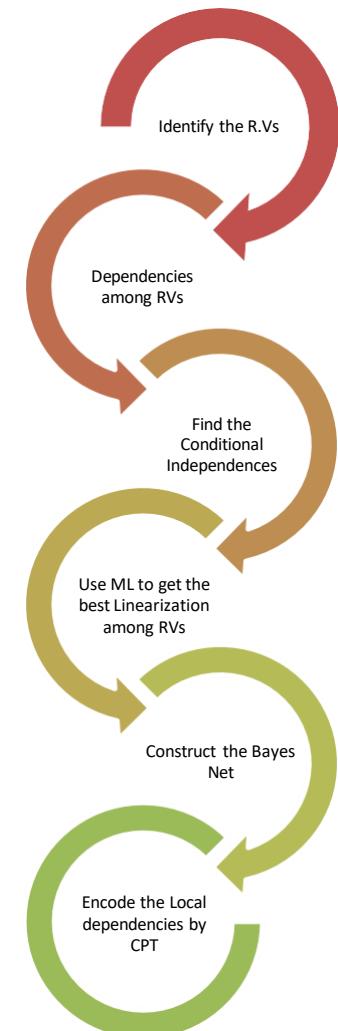
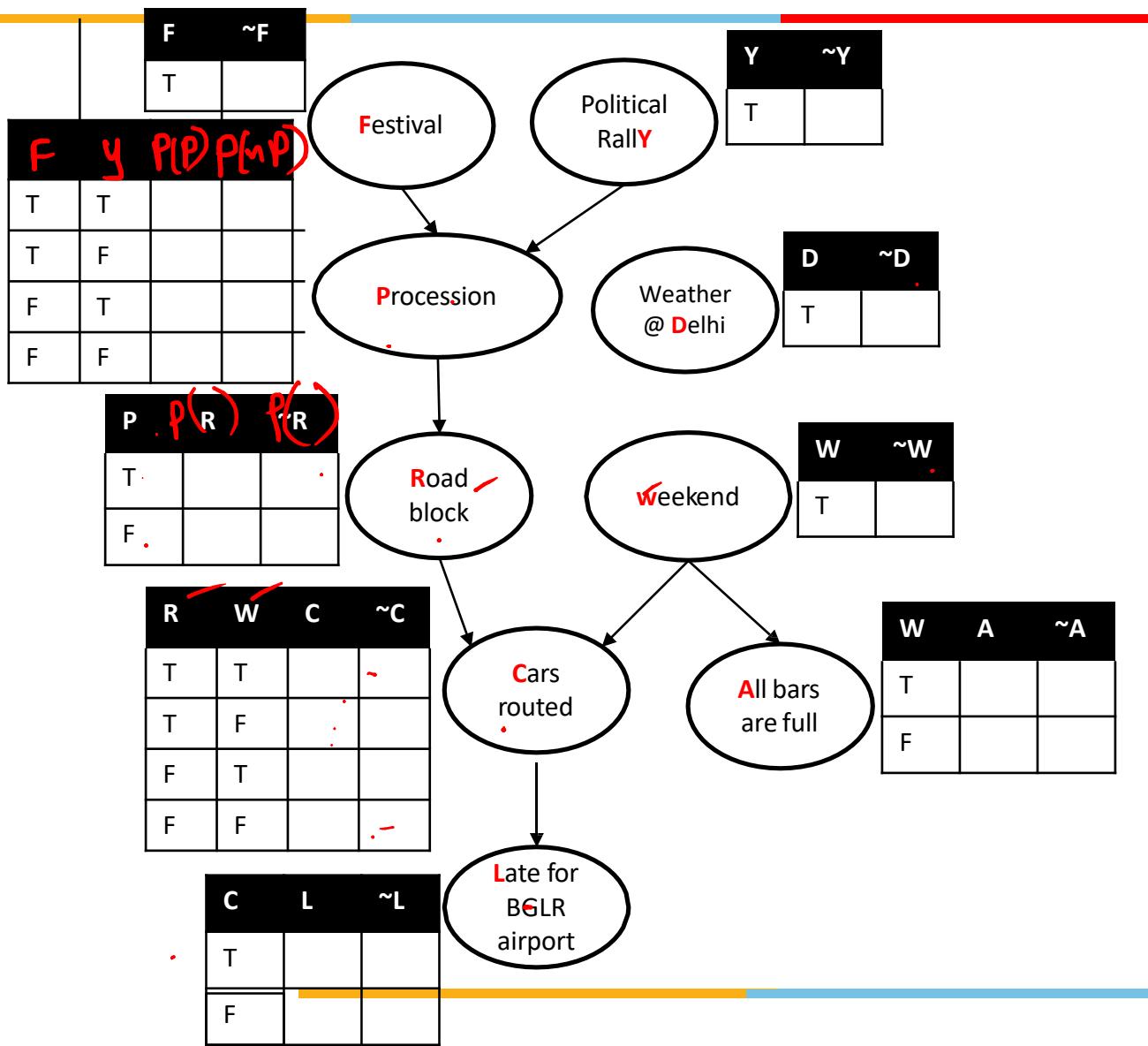
## Example Bayesian Net #3

### Traffic Prediction -Travel Estimation

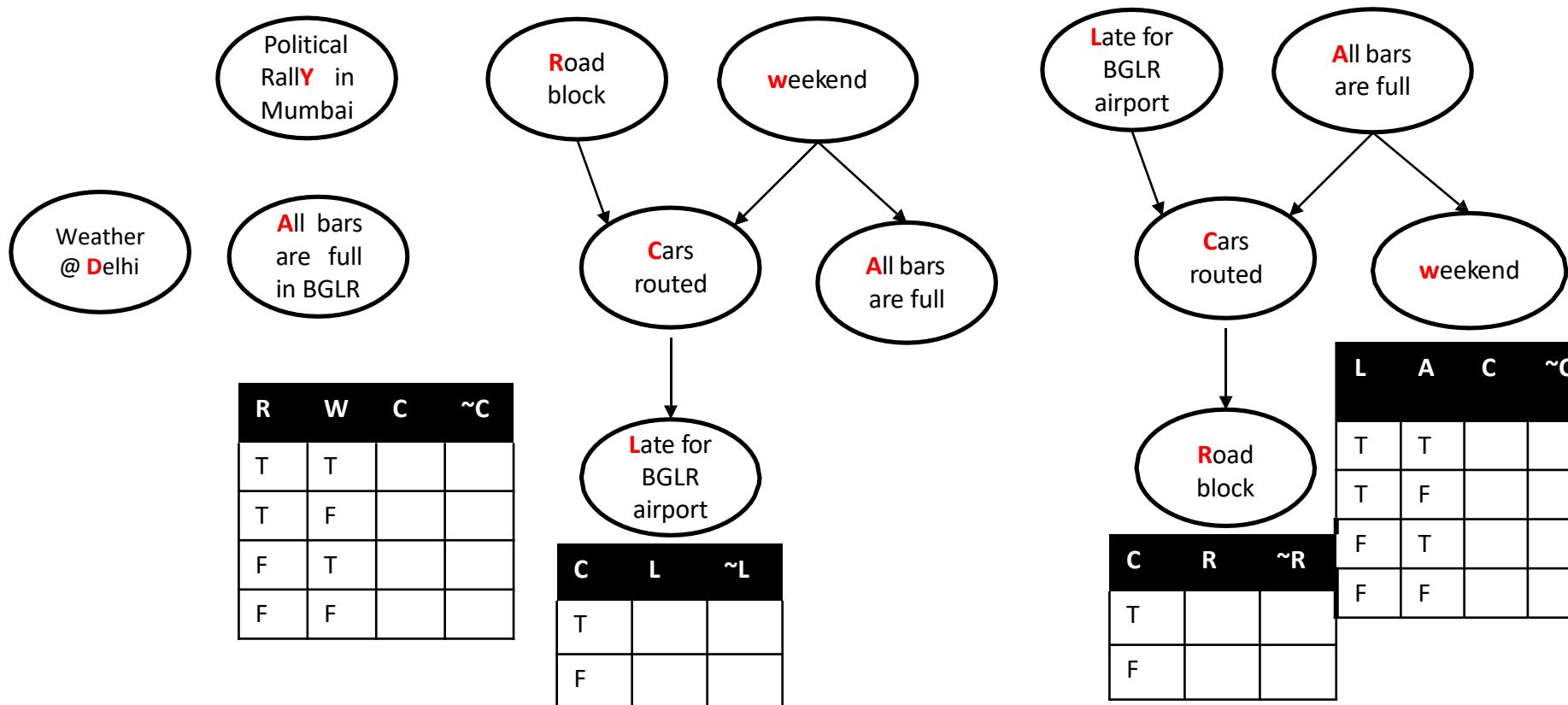
- AI system reminds traveler regarding start time
- Travel plan is to reach Delhi and the weather of Delhi may influence the accommodation plans
- Traveler always take car to reach airport
- Car may be rerouted either due to road block or weekday traffic during working hours which delays the arrival to airport
- Bars are always observed to be full on weekends
- Authorities block roads to safe the processions
- Processions observed during festive season or due to the political rally.
- **Problem:** Given the information that there is a political rally expected estimate the probability of late arrival



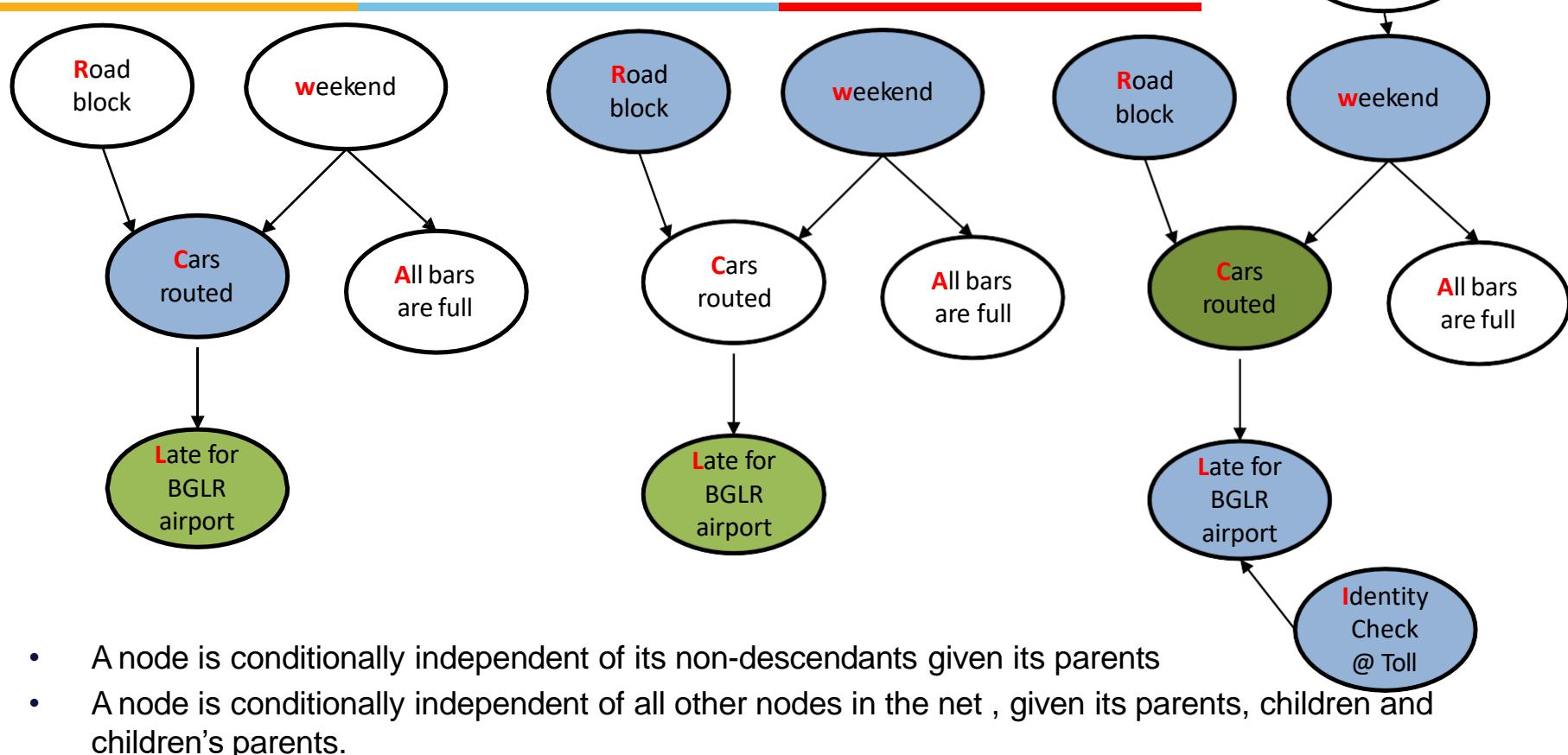
# Example Bayesian Net #3



## Example Bayesian Nets



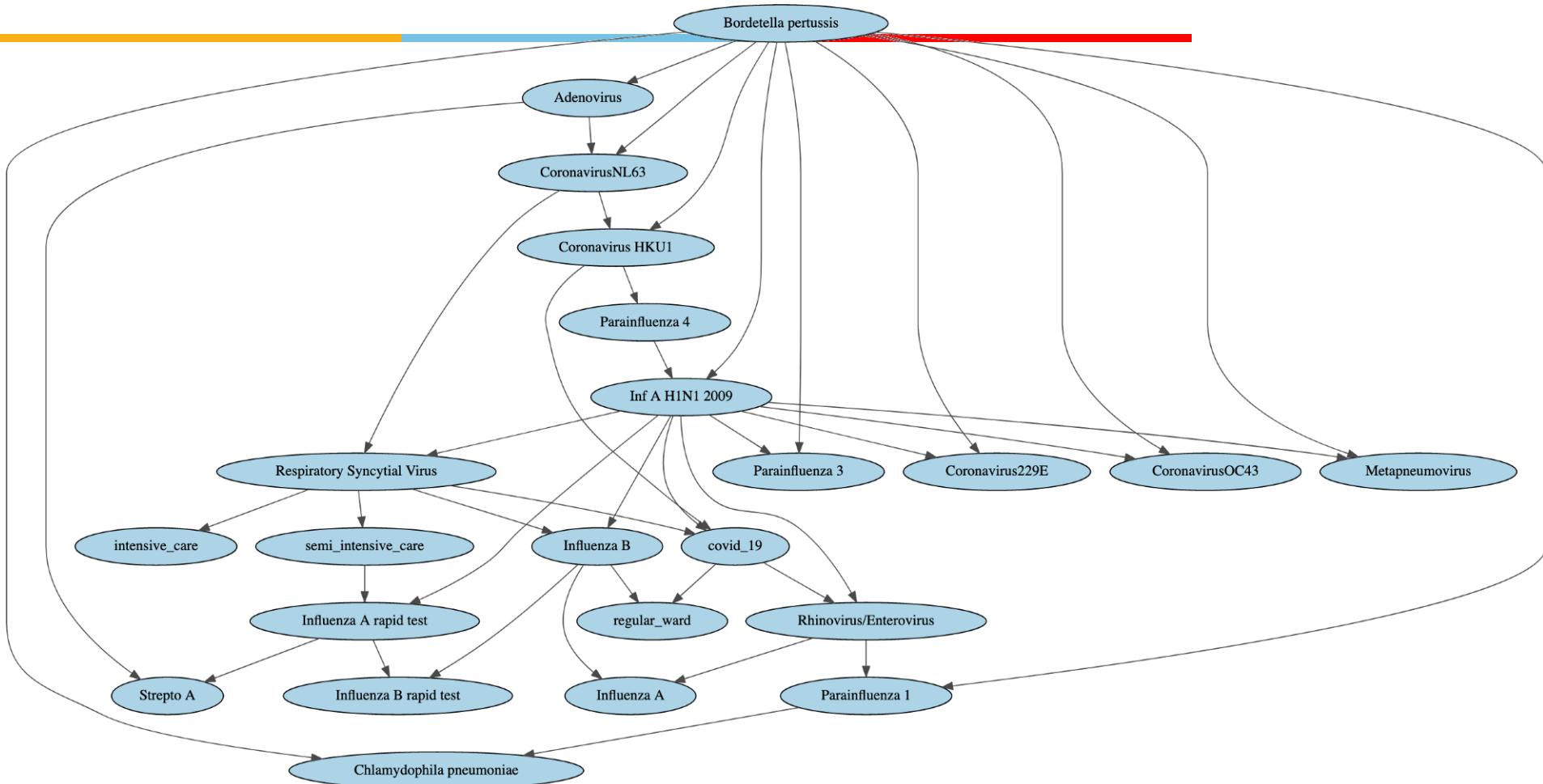
## Example Bayesian Nets



# Bayesian Nets

Interesting Case Study

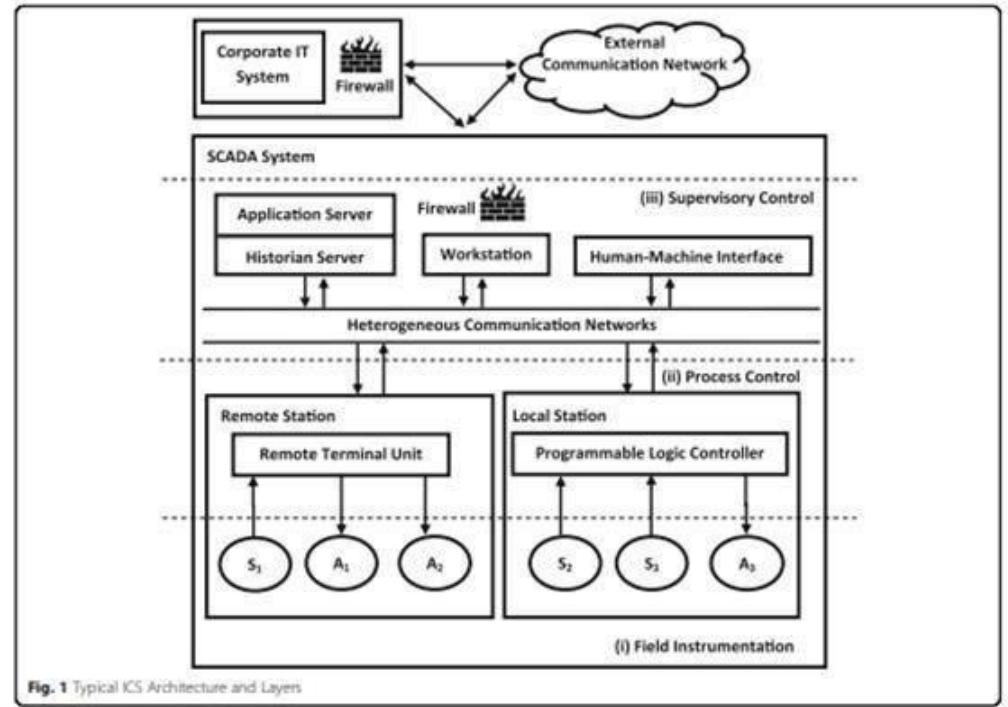
111-column wide dataset : 6347497291776 entries to store the JPD  
817 entries. Memory gain :99.9999998712879%!



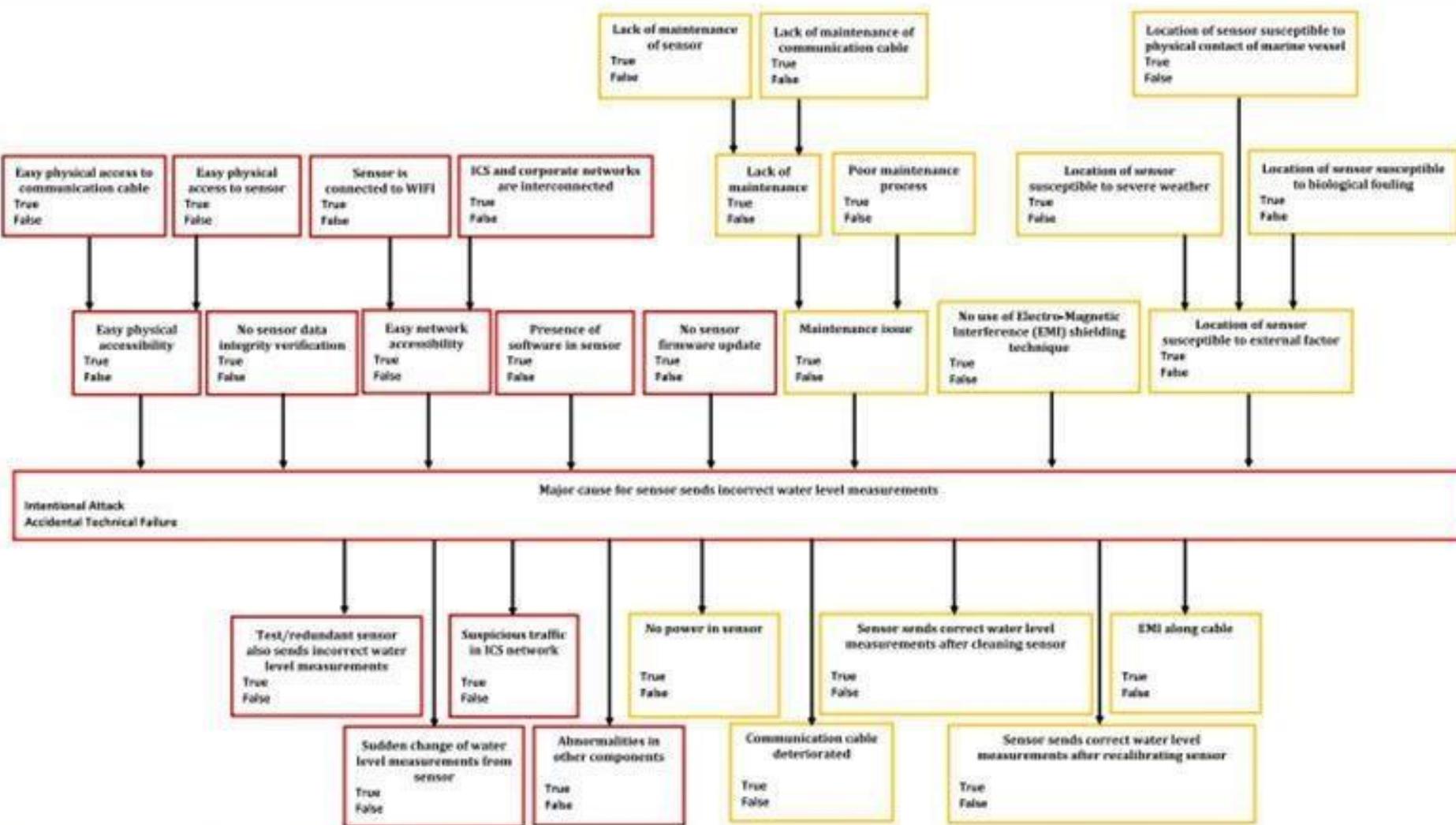
Source Credit : <https://www.kaggle.com/einsteindata4u/covid19>

# Bayesian Network

## Cyber Security



Source Credit : 2021 : Chockalingam, S., Pieters, W., Teixeira, A. et al. Bayesian network model to distinguish between intentional attacks and accidental technical failures: a case study of floodgates.



**Fig. 4** Constructed Qualitative BN Model. (In this Figure, the presence of contributory factors and observations (or test results) colored in dark red would increase the likelihood of the problem (colored in red) due to an attack on the sensor. Furthermore, the presence of contributory factors and observations (or test results) colored in orange would increase the likelihood of the problem due to sensor failure)

Source Credit : 2021 : Chockalingam, S., Pieters, W., Teixeira, A. et al. Bayesian network model to distinguish between intentional attacks and accidental technical failures: a case study of floodgates.

# Bayesian Network

## Cyber Security

**Table 2** CPT Excerpt – Problem Variable

$C_1$	$C_2$	$C_3$	$C_4$	$C_5$	$C_6$	$C_7$	$C_8$	$Y$	Attack	Failure
True	0.02	0.98								
True	False	0.09	0.91							
True	True	True	True	True	True	False	True	0.06	0.94	
True	True	True	True	True	True	False	False	0.24	0.76	
True	True	True	True	True	False	True	True	0.09	0.91	
True	True	True	True	True	False	True	False	0.38	0.62	
True	True	True	True	True	False	False	True	0.24	0.76	
True	True	True	True	True	False	False	False	0.97	0.03	
True	True	True	True	False	True	True	True	0.02	0.98	
True	True	True	True	False	True	True	False	0.09	0.91	

In this table,  $C_1$ : Easy physical accessibility,  $C_2$ : No sensor data integrity verification,  $C_3$ : Easy network accessibility,  $C_4$ : Presence of software in sensor,  $C_5$ : No sensor firmware update,  $C_6$ : Maintenance issue,  $C_7$ : No use of EMI shielding technique,  $C_8$ : Location of sensor susceptible to external factor and  $Y$ : Major cause for sensor sends incorrect water level measurements

Source Credit : 2021 : Chockalingam, S., Pieters, W., Teixeira, A. et al. Bayesian network model to distinguish between intentional attacks and accidental technical failures: a case study of floodgates.

# Inferences in Bayesian Nets

Enumeration

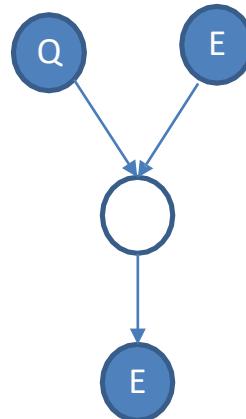
## Diagnostic



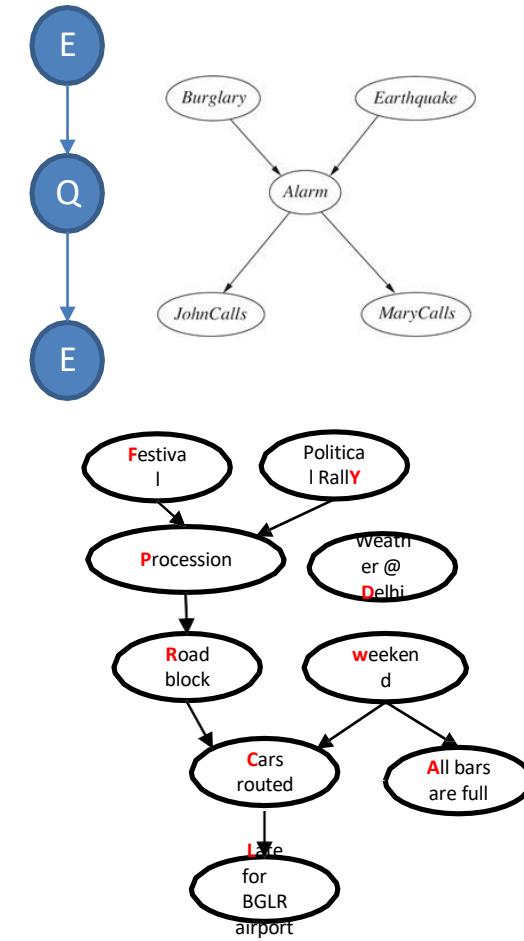
## Causal



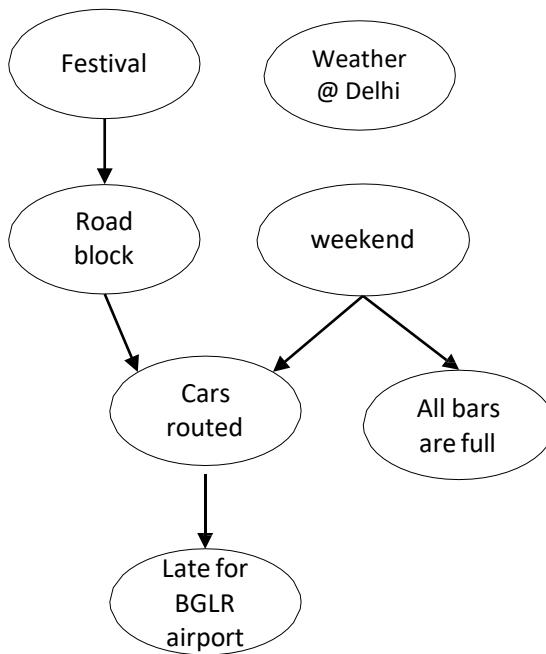
## Inter-Causal



## Mixed Inferences



## Examples



- $P(L|W \cap R \cap F \cap NC)$
1. Calculate the probability that arrival at airport was delayed during a weekend but there was no road block or festival and car was not routed anywhere.
  2. What is the probability that it is a festival season given cars were routed?  
 $P(F|C)$
  3. What is the probability that car arrived late at airport given it's a festival day?  
 $P(L|F)$

---

**Required Reading: AIMA - Chapter # 4.1, #4.2, #5.1, #9**

Next Session Plan:

- (Prerequisite Reading : Refresh the basics of probability , Bayes Theorem , Conditional Probability, Product Rule, Conditional Independence, Chain Rule)
- Bayesian Network
- Representation
- Inferences (Exact and approximate-only Direct sampling)

**Thank You for all your Attention**

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

**DSECLZG557**

## M5 : Probabilistic Representation and Reasoning

Indumathi V  
Guest Faculty  
BITS -WILP

**BITS** Pilani  
Pilani Campus

# Course Plan

- M1 Introduction to AI
- M2 Problem Solving Agent using Search
- M3 Game Playing
- M4 Knowledge Representation using Logics
- M5 Probabilistic Representation and Reasoning
- M6 Reasoning over time
- M7 Ethics in AI

## Probabilistic Representation and Reasoning

### A. Inference using full joint distribution

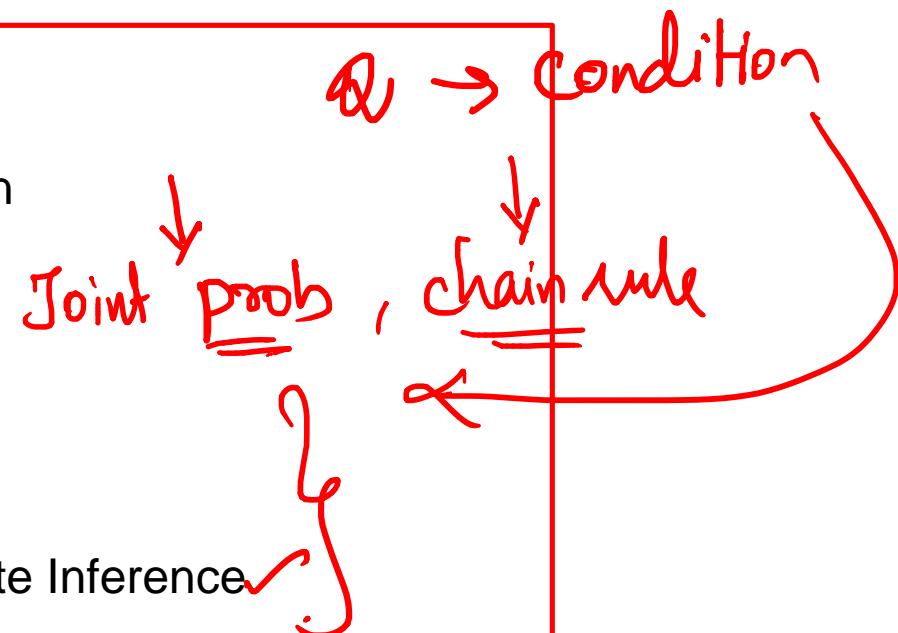
### B. Bayesian Networks

I. Knowledge Representation

II. Conditional Independence

III. Exact Inference

IV. Introduction to Approximate Inference



# Inferences in Bayesian Nets

Enumeration

# Belief Nets

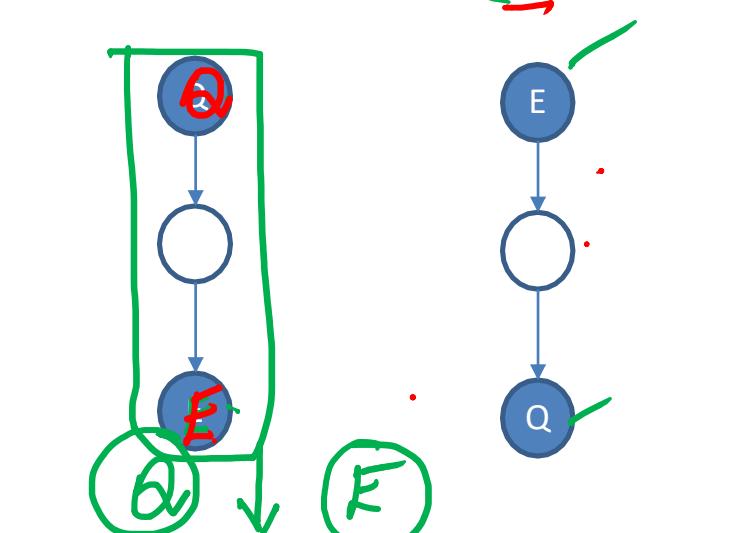
innovate

achieve

lead

~~Q & E~~

① Diagnostic



$$Q_1 : P(A | JM)$$

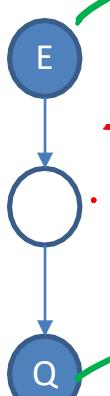
$$Q_2 : P(E | A)$$

$$Q_3 : P(A | E)$$

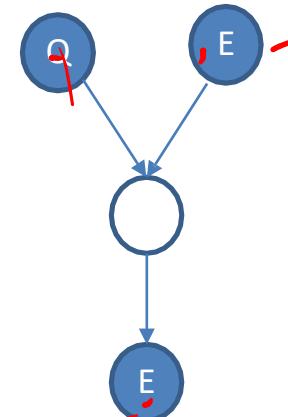
Some

} =

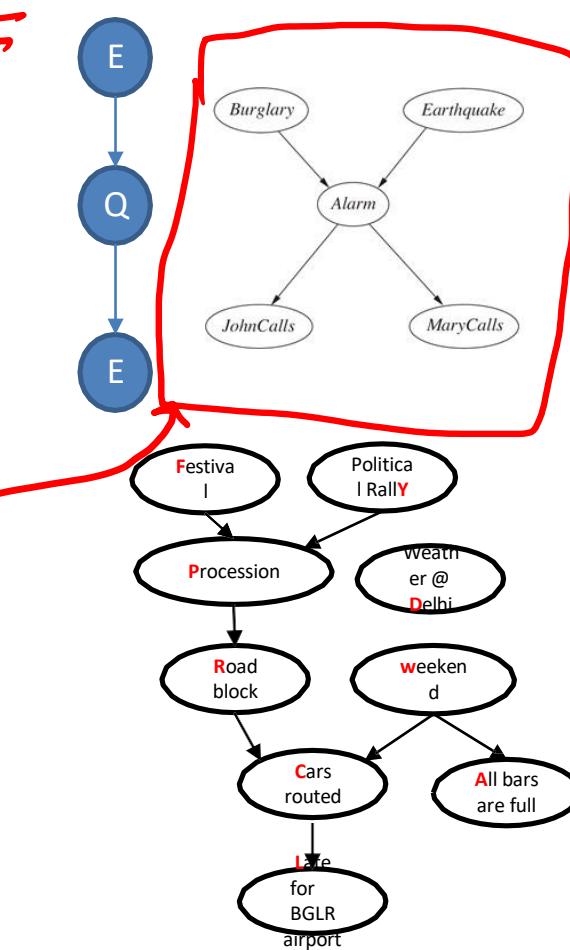
② Causal



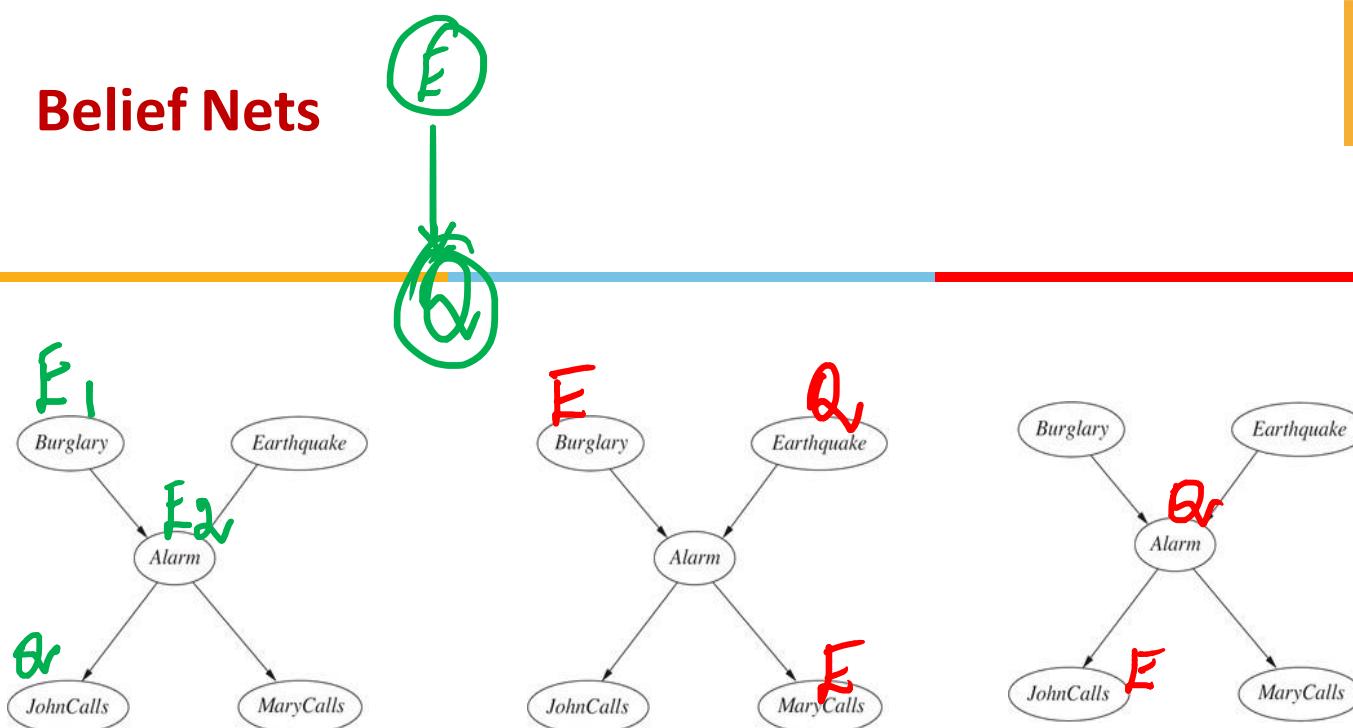
③ Inter-Casual



④ Mixed Inferences



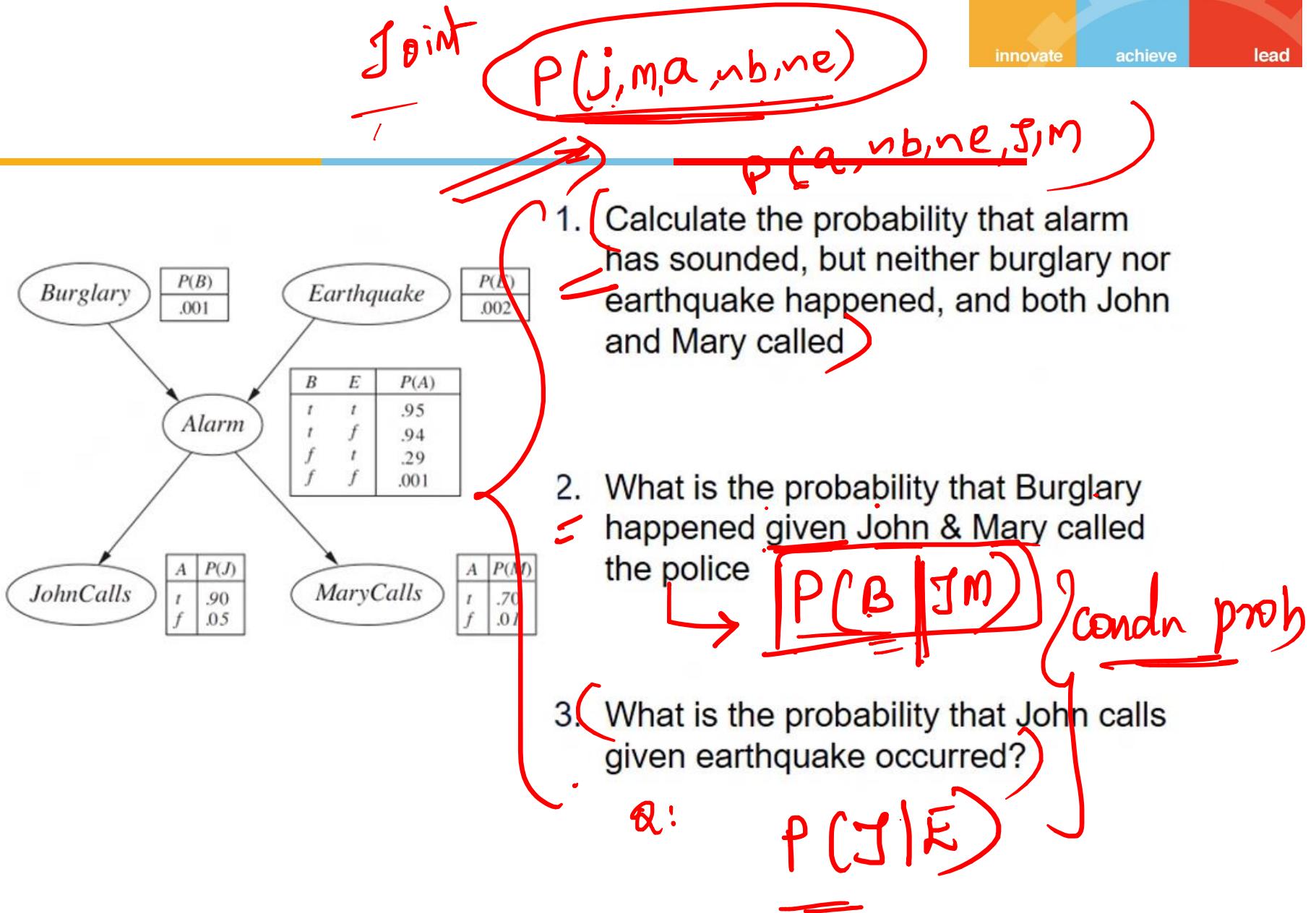
## Belief Nets



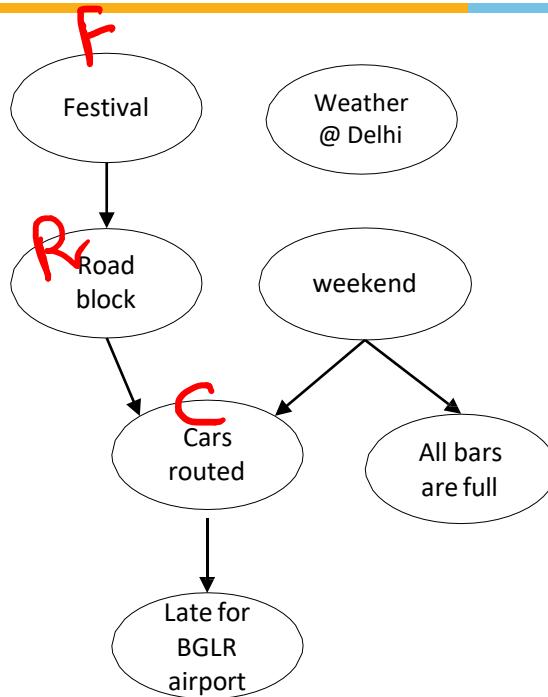
$P(J|B \wedge A)$   $\rightarrow$  causal

$P(E|BM)$   $\rightarrow$  DC

$P(A|J)$   $\rightarrow$  Diagnostic



## Examples



$\underline{P(L|W \wedge \neg R \wedge \neg F \wedge \neg C)}$

1. Calculate the probability that arrival at airport was delayed during a weekend but there was no road block or festival and car was not routed anywhere.
2. What is the probability that it is a festival season given cars where routed?  $\rightarrow \underline{\underline{P(F|C)}}$
3. What is the probability that car arrived late at airport given it's a festival day?  $\rightarrow \underline{\underline{P(L|F)}}$

# Examples

$P(B|JM)$  → Joint chain

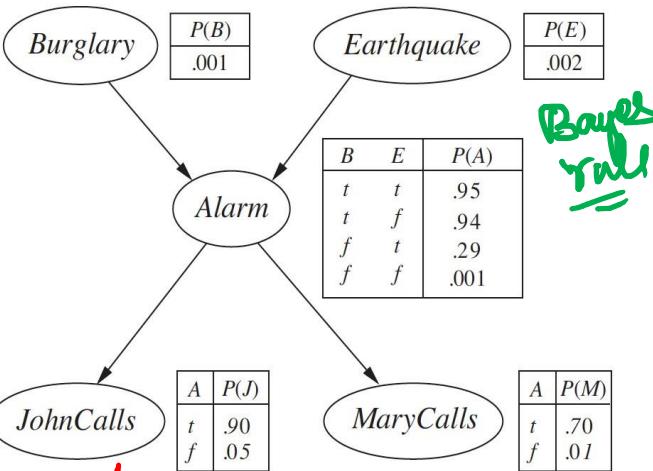


2. What is the probability that Burglary happened given John & Mary called the police?

From the definition of conditional probability, Bayes theorem can be derived for events as given below:

$$P(A|B) = P(A \cap B) / P(B), \text{ where } P(B) \neq 0$$

$$P(B|A) = P(B \cap A) / P(A), \text{ where } P(A) \neq 0$$



$$P(A) + P(\neg A) = 1$$

$$P(B|JM) + P(\neg B|JM) = 1$$

$$\frac{P(B|JM)}{P(JM)} + \frac{P(\neg B|JM)}{P(JM)} = 1 \rightarrow$$

$$\frac{P(B|JM)}{P(JM)} + \frac{P(\neg B|JM)}{P(JM)} = 1 \rightarrow$$

$$\text{let } P(JM) = 1 \Rightarrow [P(B|JM) + P(\neg B|JM)] = 1$$

$$P(JM)$$

$$\alpha = \frac{1}{P(B|JM) + P(\neg B|JM)}$$

$$\alpha = \frac{1}{P(B|JM) + P(\neg B|JM)} \rightarrow ①$$

$$P(B|JM) = \frac{P(B, JM)}{P(JM)}$$

$$P(B|JM) = \frac{\sum_{A, E} P(J, M, A, B, E)}{\sum_{A, B, E} P(J, M, A, B, E)}$$

$$P(B|JM) \Rightarrow P(B|JM) | \cdot \frac{1}{P(JM)} | \cdot ( )$$

$$\Rightarrow \frac{P(B|JM)}{P(B|JM) + P(\neg B|JM)} \Rightarrow$$

# Examples

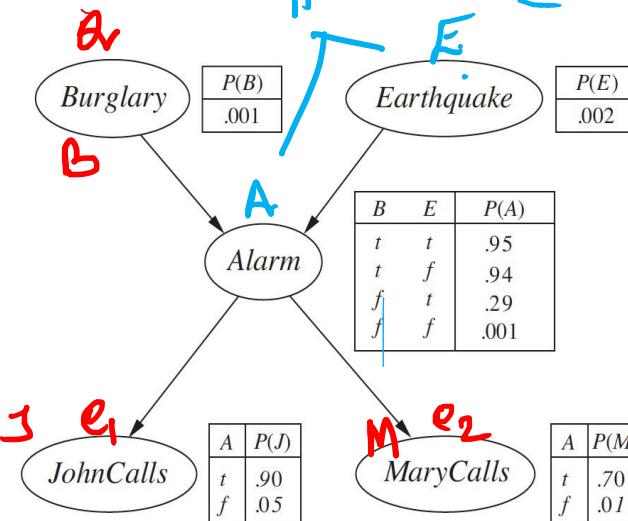
innovate

achieve

lead

2. What is the probability that Burglary happened given John & Mary called the police

hidden variables



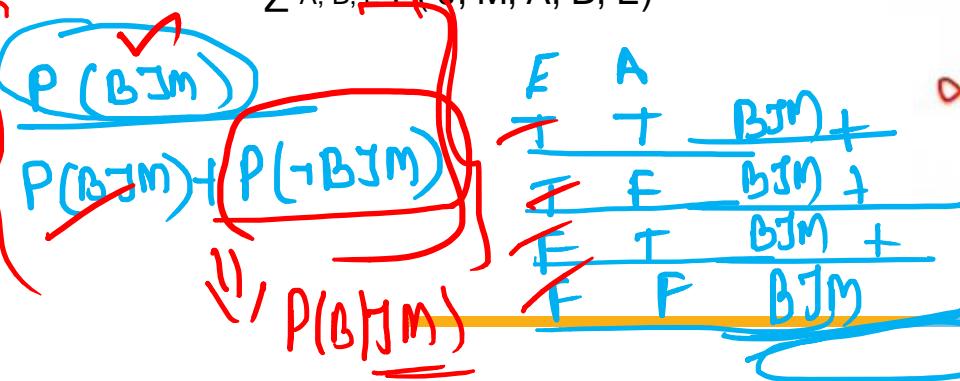
$$P(B|JM) + P(\neg B|JM) = 1$$

$$\frac{P(BJM)}{P(JM)} + \frac{P(\neg BJM)}{P(JM)} = 1$$

$$\frac{1}{P(JM)} [P(BJM) + P(\neg BJM)] = 1$$

$$P(B|JM) = \frac{P(B, JM)}{P(J, M)}$$

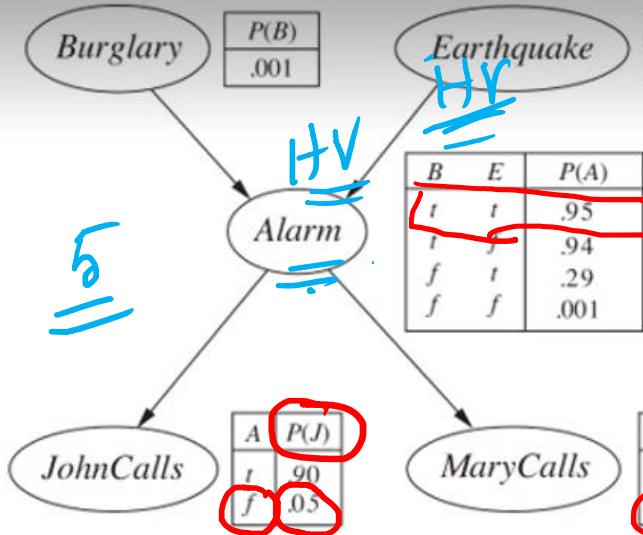
$$P(B|JM) = \frac{\sum_{A, E} P(J, M, A, B, E)}{\sum_{A, B, E} P(J, M, A, B, E)}$$



$$\text{let } \alpha = \frac{1}{P(JM)}$$

$$\alpha = \frac{1}{P(BJM) + P(\neg BJM)} \rightarrow ①$$

Marginalization



2. What is the probability that Burglary happened given John & Mary called the police

$$P(B | J, M) = \frac{P(B, J, M)}{P(J | M)}$$

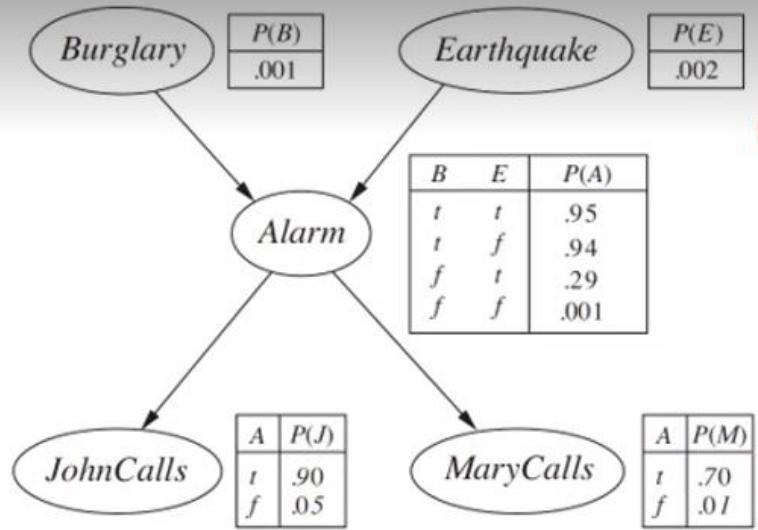
$$P(B | J, M) = \frac{\sum_{A, E} P(J, M, A, B, E)}{\sum_{A, B, E} P(J, M, A, B, E)}$$

$$\underline{P(B|J^M)} =$$

$$= \left[ P(J|A) \cdot P(M|A) \cdot P(A|B,E) \cdot P(B) \cdot P(E) \right] + \left[ P(J|\neg A) \cdot P(M|\neg A) \cdot P(\neg A|B,E) \cdot P(B) \cdot P(E) \right] \\ + \left[ P(J|A) \cdot P(M|A) \cdot P(A|\neg B,\neg E) \cdot P(\neg B) \cdot P(\neg E) \right] + \left[ P(J|\neg A) \cdot P(M|\neg A) \cdot P(\neg A|\neg B,\neg E) \cdot P(\neg B) \cdot P(\neg E) \right]$$

0.05 \* 0.01 \* (-0.95) \* 0.001 \* 0.002

$$\begin{aligned}
 \sum_{A,E} P(J,M,A,B,E) &= P(B|JM) = \sum_{A,E} P(J,M,A,B,E) \\
 &= \sum_{A,E} P(J|A,B,E) \cdot P(M|A,B,E) \\
 &\quad P(A|B,E) \cdot P(B|E) \cdot P(E) \\
 &= \sum_{A,E} P(J|A) \cdot P(M|A) \cdot P(A|B,E) \\
 &\quad P(B) \cdot P(E) \\
 &= \sum_{A,E} P(J|A) \cdot P(M|A) \cdot P(A|B,E) \cdot P(B) \cdot P(E) \\
 &= \sum_{A,E} [P(J|A) \cdot P(M|A) \cdot P(A|B,E) \cdot P(B) \cdot P(E)] \\
 &\quad [P(J|A) \cdot P(M|A) \cdot P(A|B,E) \cdot P(B) \cdot P(\neg E)] \\
 &\quad [P(J|A) \cdot P(M|A) \cdot P(A|\neg B,E) \cdot P(B) \cdot P(\neg E)] \\
 &\quad [P(J|\neg A) \cdot P(M|\neg A) \cdot P(\neg A|B,E) \cdot P(B) \cdot P(\neg E)]
 \end{aligned}$$



2. What is the probability that Burglary happened given John & Mary called the police

$$P(B|J, M) = \frac{P(B, J, M)}{P(J, M)}$$

$$P(B|J, M) = \frac{\sum_{A, E} P(J, M, A, B, E)}{\sum_{A, B, E} P(J, M, A, B, E)}$$

$$\begin{aligned}
 &= \sum_A \left\{ P(J|A) \cdot P(M|A) \cdot P(A|B, E) \cdot P(B) \cdot P(E) \right\} \\
 &\quad + \sum_{\neg A} \left\{ P(J|\neg A) \cdot P(M|\neg A) \cdot P(\neg A|B, E) \cdot P(B) \cdot P(E) \right\} \\
 &= \boxed{P(\neg B|M) \Rightarrow}
 \end{aligned}$$

$$\begin{aligned}
 P(B|J, M) &= \sum_{A, E} P(J, M, A, B, E) \\
 &= \sum_{A, E} P(J|A) \cdot P(M|A) \cdot P(A|B, E) \cdot P(B) \cdot P(E) \\
 &= \sum_{A, E} P(J|A) \cdot P(M|A) \cdot P(A|B, E) \cdot P(B) \cdot P(E)
 \end{aligned}$$

$$P(BJM) = \sum_{A,E} P(J, M, A, B, E)$$

$$P(B|JM) = \frac{P(BJM)}{P(BJM) + P(\neg BJM)}$$

$$\text{chain rule} = \sum_{A,E} P(J|a,b,c,e) \cdot P(M|a,b,c) \cdot P(a|bc) \cdot P(b|c) \cdot P(e)$$

$$\Rightarrow \sum_{A,E} P(J|a) \cdot P(M|a) \cdot P(a|bc) \cdot P(b) \cdot P(e)$$

$$\Rightarrow \sum_A (P(J|a) \cdot P(M|a) \cdot P(a|bc) \cdot P(b) \cdot P(e)) +$$

$$\sum_A P(J|\neg a) \cdot P(M|\neg a) \cdot P(\neg a|bc) \cdot P(b) \cdot P(\neg e)$$

$$\Rightarrow \left\{ \begin{array}{l} P(J|a) \cdot P(M|a) \cdot P(a|bc) \cdot P(b) \cdot P(e) \\ P(J|\neg a) \cdot P(M|\neg a) \cdot P(\neg a|bc) \cdot P(b) \cdot P(\neg e) \end{array} \right. \quad \oplus$$

$$P(J|a) \cdot P(M|a) \cdot P(a|bce) \cdot P(b) \cdot P(\neg e) \quad \oplus$$

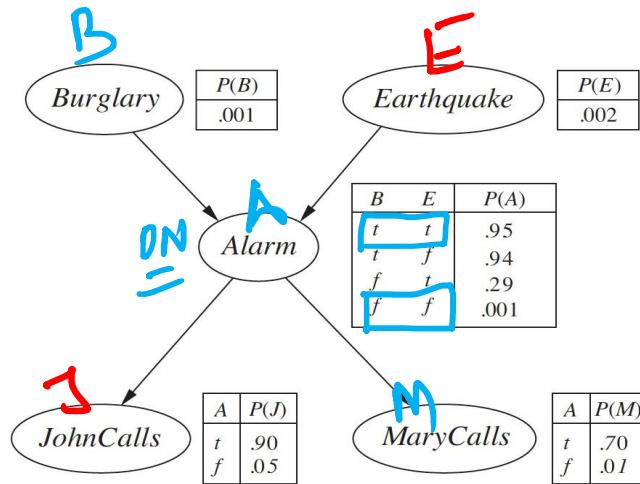
$$P(J|\neg a) \cdot P(M|\neg a) \cdot P(\neg a|bce) \cdot P(b) \cdot P(\neg e) \quad \oplus$$

$$P(BJM) = P(BJM) \Rightarrow 0$$

$$\frac{v_1}{v_1 + v_2}$$

Examples  $P(J|E) \Rightarrow \frac{P(J,E)}{P(E)}$

3. What is the probability that John calls given earthquake occurred?



Exact Inference

$$P(J|E) = \frac{P(J, E)}{P(E)}$$

$$P(J|E) = \frac{\sum_{M, A, B} P(J, M, A, B, E)}{\sum_{J, M, A, B} P(J, M, A, B, E)}$$

⋮

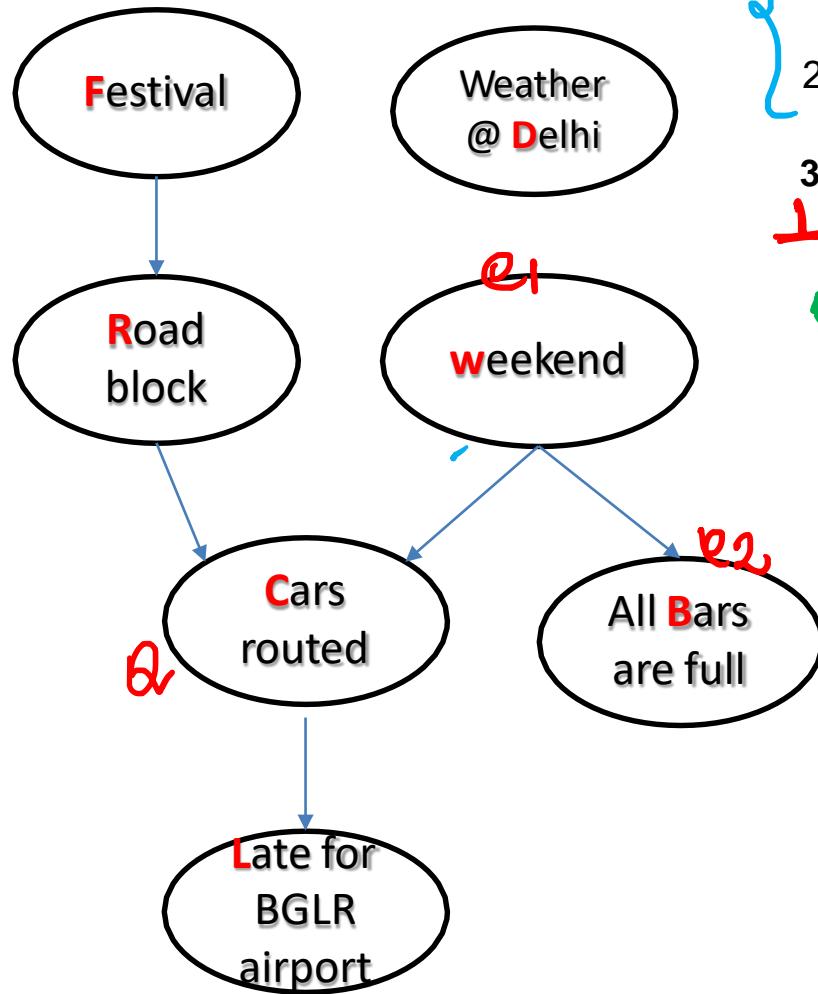
# Inferences in Bayesian Nets

Variable Elimination

Reduce Guaranteed Independent nodes

# D-Connectedness Vs D-Separation

5 steps

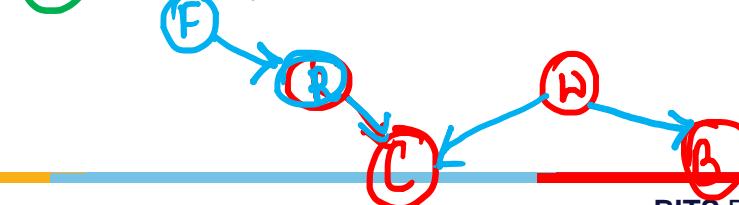


- } 1. Each variable is conditionally independent of its non-descendants, given its parents  
 } 2. Eliminate the hidden variables that is neither a query nor an evidence  
 } 3. Two variables are d-separated if they are conditionally independent given evidences

①  $\nexists C \perp B \mid W$

① List all evi & query node

② Add parent & ancestors



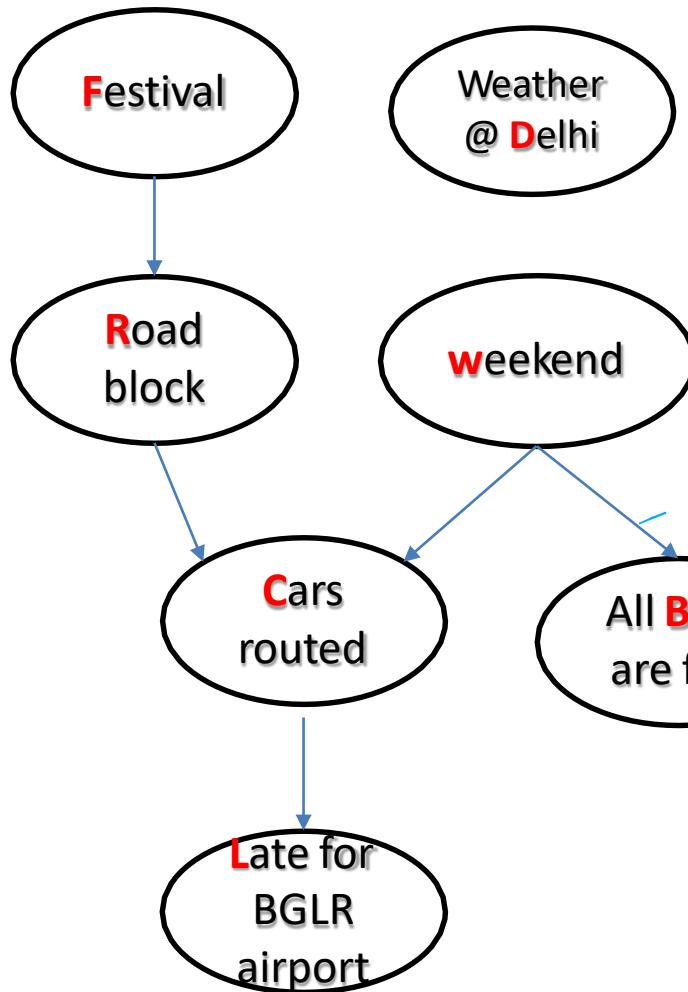
Yes

## D-Connectedness Vs D-Separation

C & B are desparated →

ignore

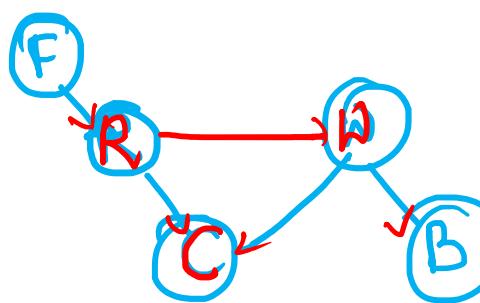
B in the presence of W



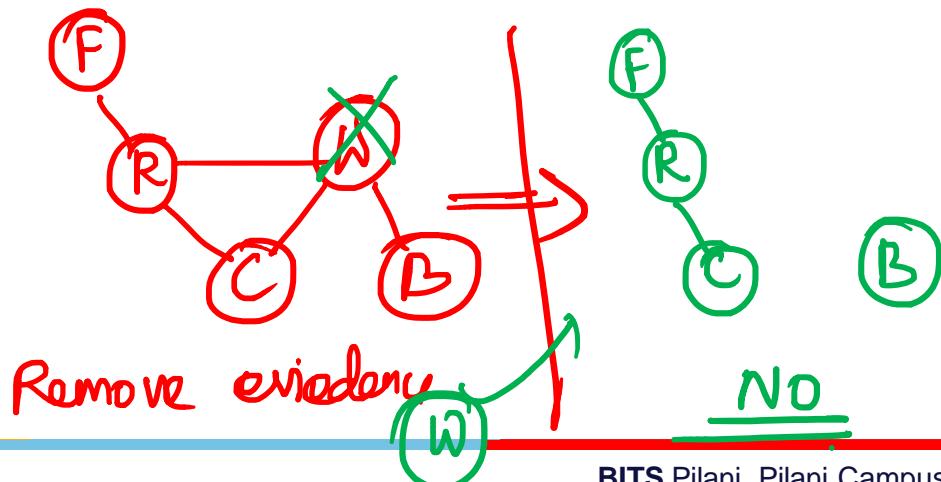
1. Each variable is conditionally independent of its non-descendants, given its parents
2. Eliminate the hidden variables that is neither a query nor an evidence
3. Two variables are d-separated if they are conditionally independent given evidences

3rd
Managing the parent

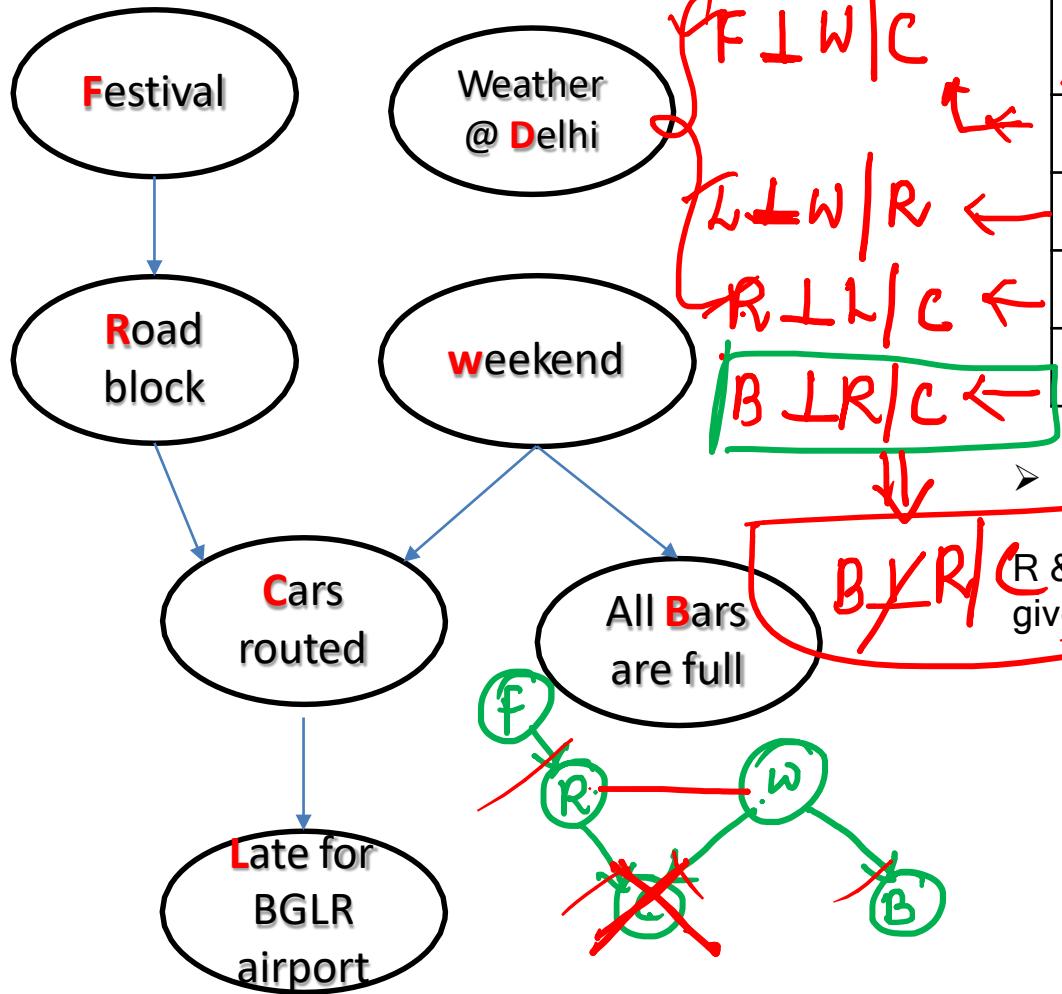
$C \perp B | W$

evidence

4th

⑤ Remove evidence


No

## Try it & Test



X	Y	Evidence Z	d-sep?
<del>R</del>	<del>W</del>	<del>C (e)</del>	
F	W	C (e)	No
L	W	R (e)	No
R	L	C	Yes
B	R	C	No ✓

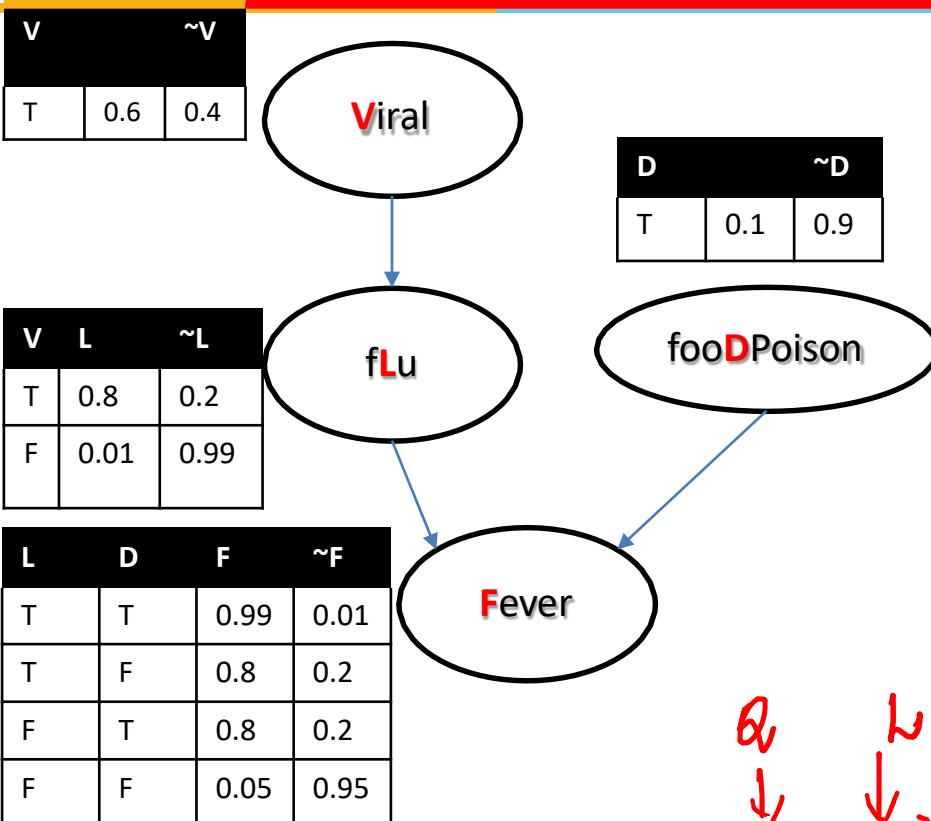
$$P(R | L, C) = P(R | L)$$

R & L are d-separated ie., conditionally independent given C

$B \perp R | C$   
Very remove it

$B \not\perp R$   
Not desopulated

## D-Separation in Inference

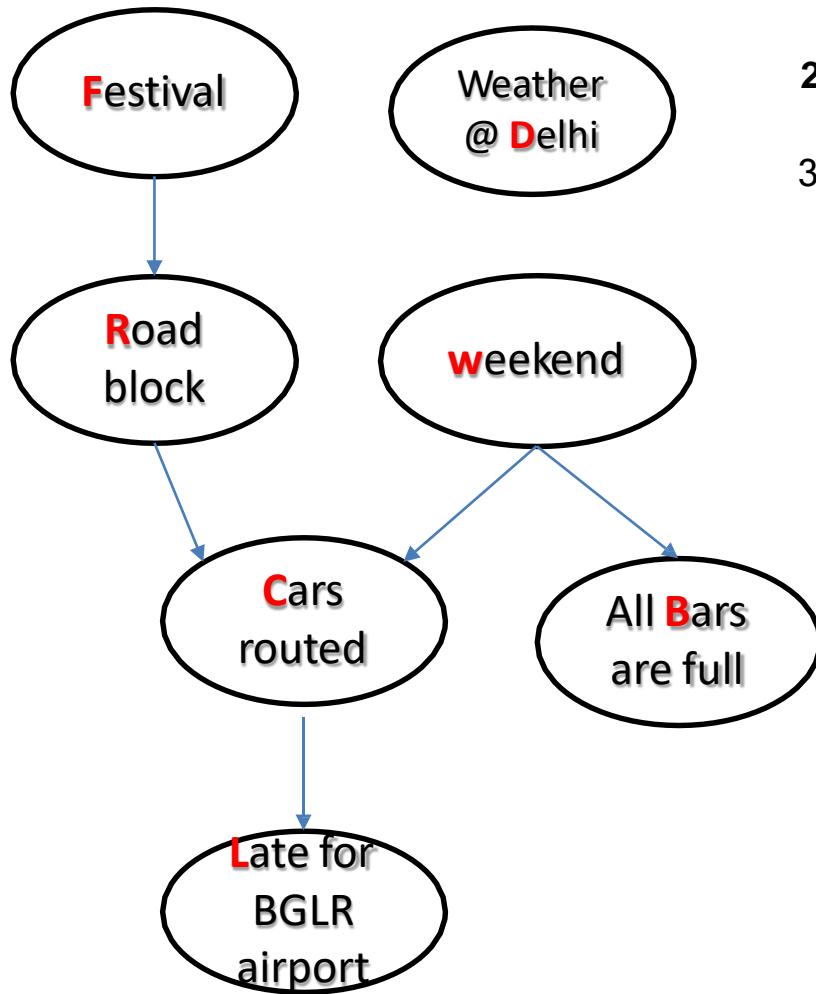


$$p(v|F, L) \xrightarrow{?} V \perp F | L \rightarrow$$

$$\begin{aligned} & P(V | F, L) \\ & P(V | D, L) \xrightarrow{?} V \perp D | L \end{aligned}$$

X	Y	Evidence Z	d-sep?
V	F	L	Yes
V	D	L	Yes

# Variable Elimination ✓



1. Each variable is conditionally independent of its non-descendants, given its parents
2. **Eliminate the hidden variables that is neither a query nor evidence**
3. Two variables are d-separated if they are conditionally independent given evidences

$$\begin{aligned}
 \mathbb{P}(B) &= \sum_{L, C, R, F} \mathbb{P}(L, C, B, W, R, F) \\
 &= \sum_L \sum_B \mathbb{P}(L|C) \cdot \mathbb{P}(B|W) \cdot \sum_W \mathbb{P}(C|W, R) \cdot \sum_R \mathbb{P}(R|F) \cdot \sum_F \mathbb{P}(F) \\
 &= \mathbb{P}(B|W)
 \end{aligned}$$

All other variables are hidden w.r.t to B as (L, C, R, F) are neither evidence nor query nor  $(L, C, R, F) \in \text{Ancestors}(W, B)$

This is variable elimination example targeting irrelevant nodes

Inference  
 $P(V) P(\bar{V})$

V	$\sim V$
T	0.6 0.4



CPT  
 $P(L) P(\bar{L})$

V	L	$\sim L$
T	0.8 0.2	
F	0.01 0.99	

L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95

$P(F)$

$\overline{P(F)} \Rightarrow$   
 $\overline{\overline{P(\bar{F})}} =$

$P(L) \Rightarrow \text{True}$   
 $P(\bar{L}) \Rightarrow \text{False}$

①

Variable Elimination: V

V	$\sim V$
T	0.6 0.4

V	$P(L)$	$P(\bar{L})$
T	0.8 0.2	0.01 0.99

CPT

V	L
T	0.48 0.12 0.004 0.396
F	0.12 0.396
T	0.004
F	0.48

$P(V) \rightarrow P(L|V)$

$$P(L|V) = P(L|V) \cdot P(V)$$

$$0.8 \cdot 0.6$$

$P(\bar{V}) \rightarrow P(\bar{L}|\bar{V})$

$$P(\bar{L}|\bar{V}) = P(\bar{L}|\bar{V}) \cdot P(\bar{V})$$

$$0.2 \cdot 0.6$$

$$0.48 + 0.004 \leftrightarrow 0.484$$

$$0.12 + 0.396 \leftrightarrow 0.516$$

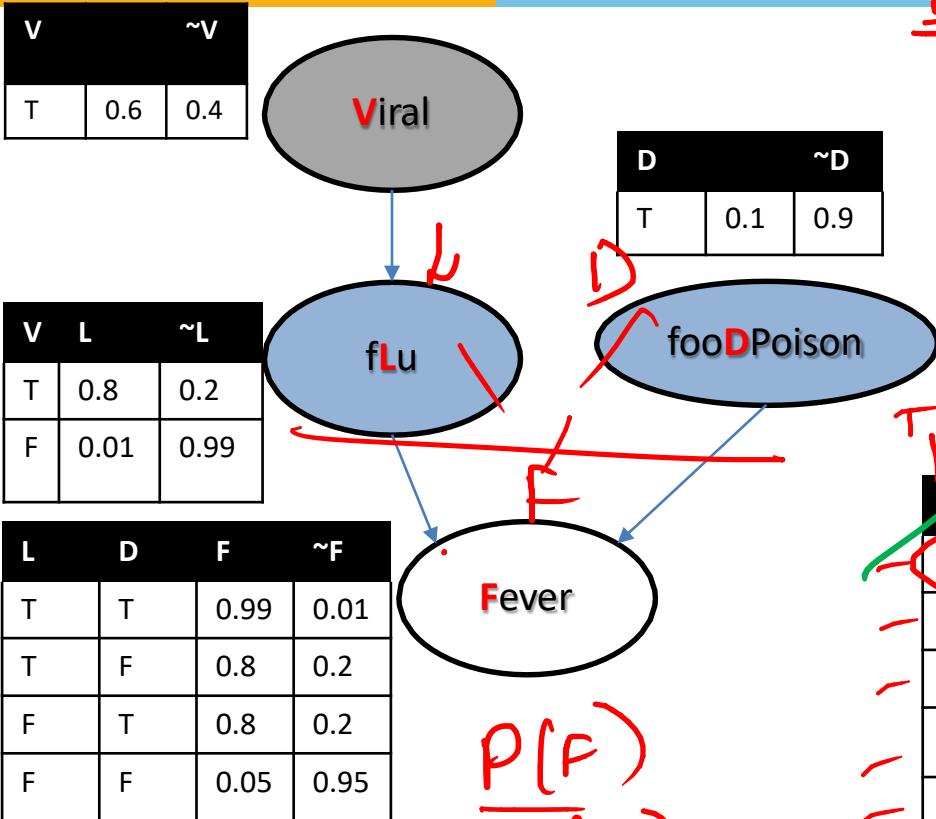
↓

$P(L)$   
 $P(\bar{L})$

$P(V)$   
 $P(L|V)$   
 $P(D)$   
 $P(F|L,D)$

# Inference

DLF



$P(L)$   
 $P(D)$

$$P(F|L,D) \Rightarrow P(D) \cdot P(L) \cdot P(F|LD)$$

DLF

Variable Elimination: L, D

CPT



L	D	~D
T	0.484	0.1
F	0.516	0.9

D	~D
T	0.1
F	0.9

L	D	F	~F
T	T	0.99	0.01
F	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95

D	L	F	~F
T	T	T	0.048
T	F	T	0.34852
F	T	T	0.04128
F	F	T	0.02322
T	T	F	0.00048
T	F	F	0.087
F	T	F	0.01032
F	F	F	0.44118

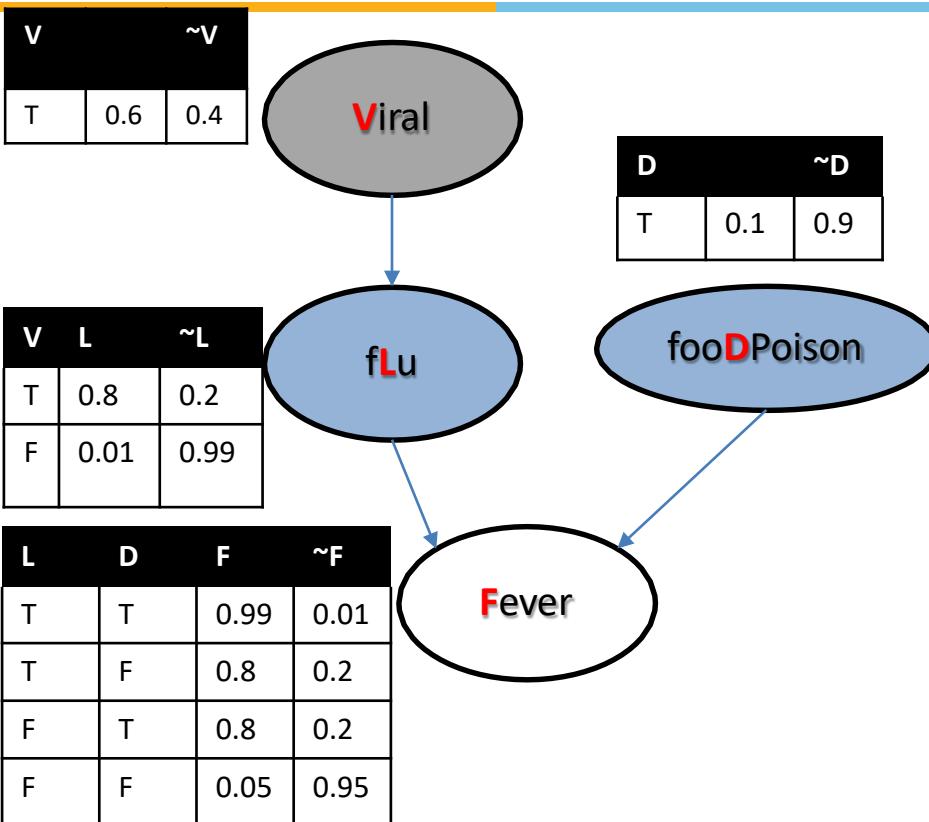
$$P(D) \cdot P(L) \cdot P(F|LD) = 0.1 \cdot 0.484 \cdot 0.99$$

L	F
T	0.08928
F	0.37174
T	0.0108
F	0.52818

$$P(\neg F | L \wedge D) = P(\neg F | L \wedge D) \cdot P(L) \cdot P(\neg F | LD) = 0.2 \cdot 0.484 \cdot 0.9 = 0.46102$$

# Inference

## Variable Elimination: L



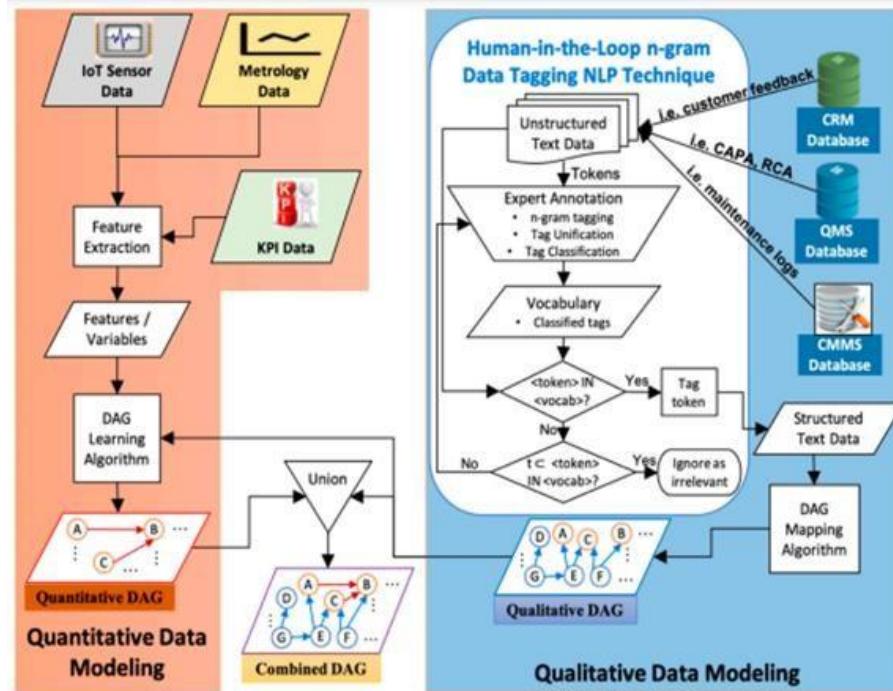
L	
T	0.484
F	0.516

L D F $\sim F$			
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95

$P(L)$   
 $P(D)$   
 $P(F|L,D)$

# Bayesian Network

## Fault Diagnostic System

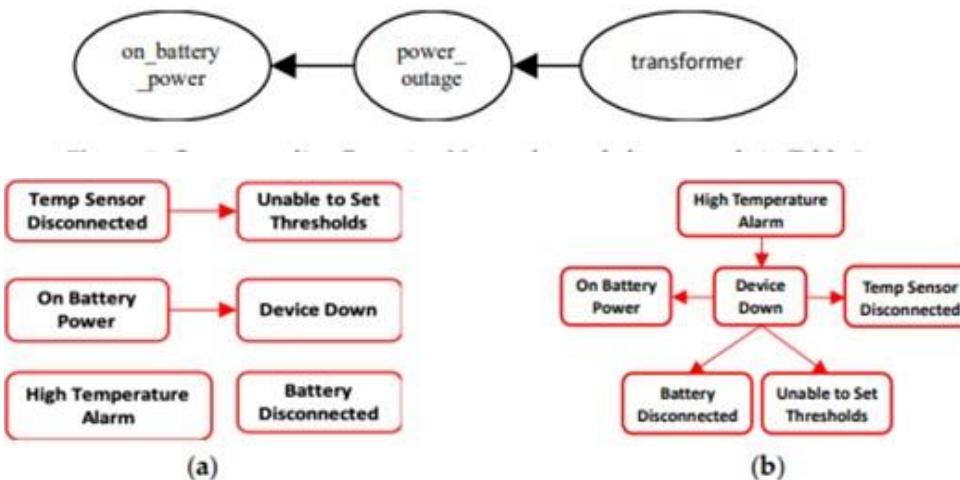


Source Credit : [Sensors 2021 : Fusion-Learning of Bayesian Network Models for Fault Diagnostics](#)

# Bayesian Network

## Fault Diagnostic System

Raw Data	Short Description		Resolution Notes		
	Symptom	Cause(s)	Link		
Classified Tags	on_battery_power	power_outage, transformer_fire	due_to		
BN Mapping	Child Variable	Child State	Parent Variable	Parent State	Ancestor Variable
	on_battery_power	yes	power_outage	yes	transformer
					Fire



Source Credit : [Sensors 2021 : Fusion-Learning of Bayesian Network Models for Fault Diagnostics](#)

# Bayesian Network

## Fault Diagnostic System

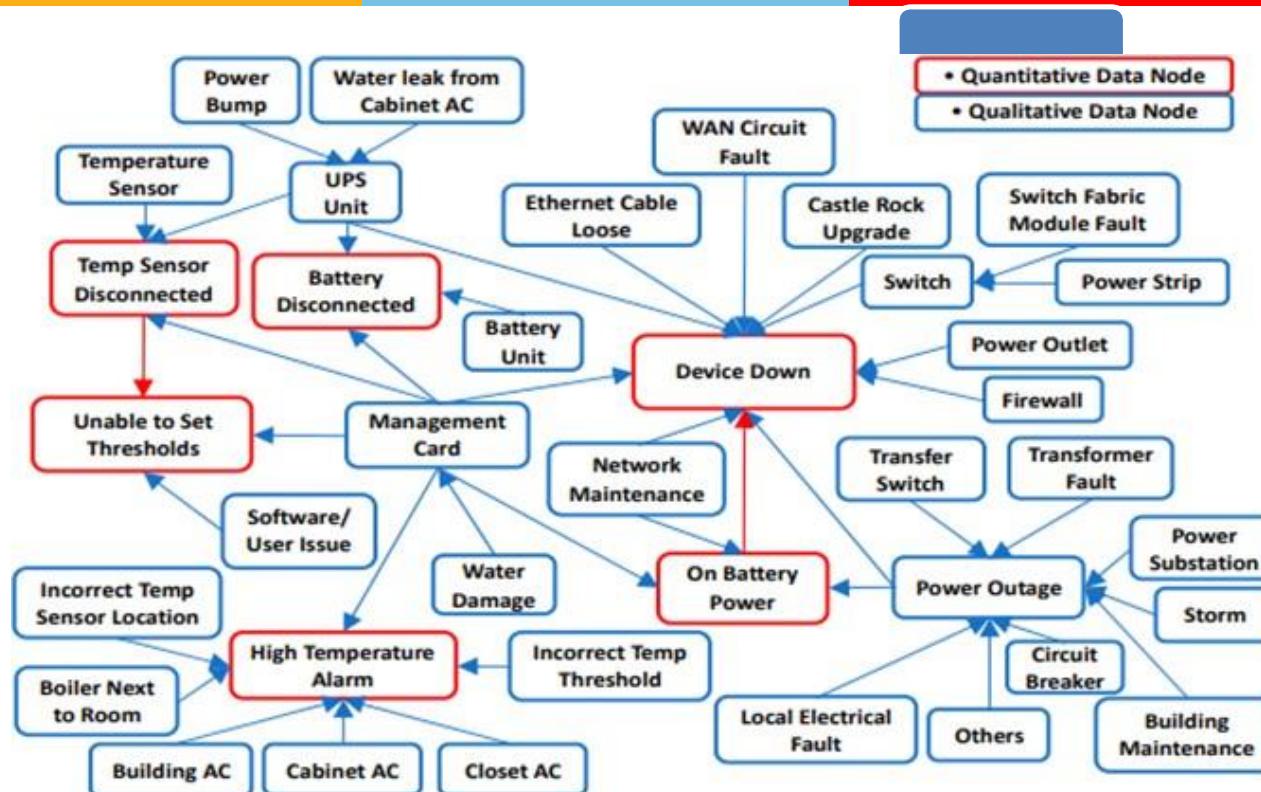
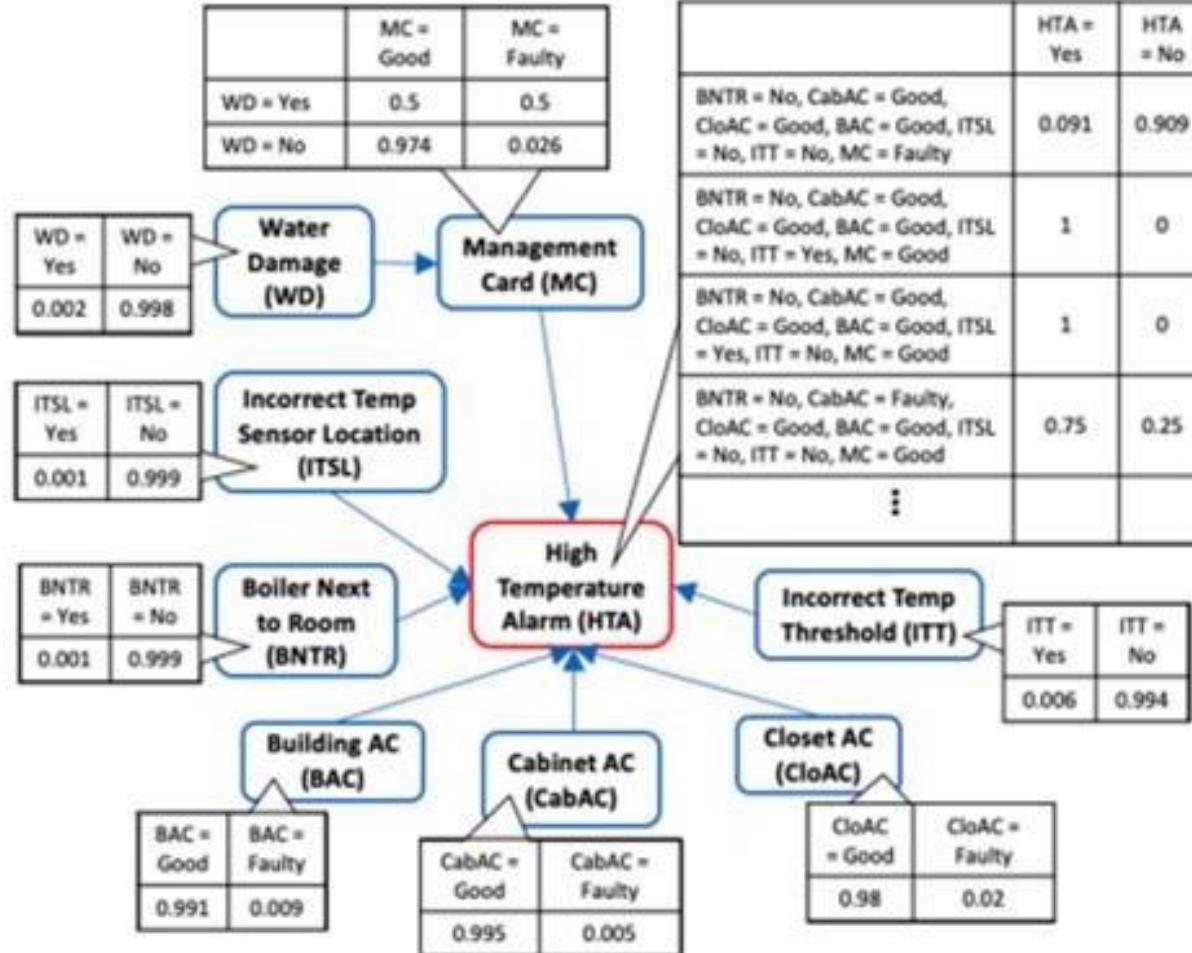


Figure 8. Fused Bayesian Network structure for top six occurring UPS messages.

Source Credit : [Sensors 2021 : Fusion-Learning of Bayesian Network Models for Fault Diagnostics](#)

# Bayesian Network

## Fault Diagnostic System



Source Credit : [Sensors 2021 : Fusion-Learning of Bayesian Network Models for Fault Diagnostics](#)

---

**Required Reading: AIMA - Chapter # 4.1, #4.2, #5.1, #9 Refer to the handout**

Next Session Plan:

- Hidden Markov Models

**Thank You for all your Attention**

Note : Some of the slides are adopted from AIMA TB materials

A Bayesian network diagram showing the following nodes and their relationships:
 

- Burglary** and **Earthquake** are parents of **Alarm**.
- Alarm** is a parent of **JohnCalls** and **MaryCalls**.
- Probability distributions:
  - Burglary**:  $P(B) = .001$
  - Earthquake**:  $P(E) = .002$
  - Alarm**:  $P(A|B, E)$  (Table)
 

B	E	$P(A)$
t	t	.95
t	f	.94
f	t	.29
f	f	.001
  - JohnCalls**:  $P(J|A)$  (Table)
 

A	$P(J)$
t	.90
f	.05
  - MaryCalls**:  $P(M|A)$  (Table)
 

A	$P(M)$
t	.70
f	.01

**innovate achieve lead**

2. What is the probability that Burglary happened given John & Mary called the police

$$P(B|J,M) = \frac{P(B, J, M)}{P(J, M)}$$

$$P(B|J,M) = \frac{\sum_{A,E} P(J, M, A, B, E)}{\sum_{A,B,E} P(J, M, A, B, E)}$$

$$= \frac{P(BJM)}{P(BJM) + P(\neg B JM)}$$

$P(BJM) + P(\neg B JM) = 1$   
 let  $\alpha = \frac{1}{P(JM)}$   
 $\alpha = \frac{1}{P(BJM) + P(\neg B JM)} \rightarrow \textcircled{1}$

Way 1:

A Bayesian network diagram showing the following nodes and their relationships:
 

- Burglary** and **Earthquake** are parents of **Alarm**.
- Alarm** is a parent of **JohnCalls** and **MaryCalls**.
- Probability distributions:
  - Burglary**:  $P(B) = .001$
  - Earthquake**:  $P(E) = .002$
  - Alarm**:  $P(A|B, E)$  (Table)
 

B	E	$P(A)$
t	t	.95
t	f	.94
f	t	.29
f	f	.001
  - JohnCalls**:  $P(J|A)$  (Table)
 

A	$P(J)$
t	.90
f	.05
  - MaryCalls**:  $P(M|A)$  (Table)
 

A	$P(M)$
t	.70
f	.01

**innovate achieve lead**

2. What is the probability that Burglary happened given John & Mary called the police

$$P(B|J,M) = \frac{P(B, J, M)}{P(J, M)}$$

$$P(B|J,M) = \frac{\sum_{A,E} P(J, M, A, B, E)}{\sum_{A,B,E} P(J, M, A, B, E)}$$

$$= \sum_{AE} P(J|MABE) \cdot P(M|ABE) \cdot P(A|BE) \cdot P(B|E) \cdot P(E)$$

$$= \sum_{AE} P(J|A) \cdot P(M|A) \cdot P(A|BE) \cdot P(B|E) \cdot P(E)$$

$$= \sum_{AE} P(J|A) \cdot P(M|A) \cdot P(A|BE) \cdot P(B|E) \cdot P(E)$$

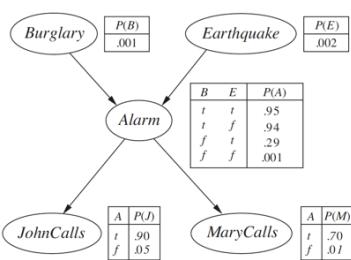
$$= \sum_A \left\{ P(J|A) \cdot P(M|A) \cdot P(A|BE) \cdot P(B|E) \cdot P(E) \right\}$$

$$+ \left\{ P(J|\neg A) \cdot P(M|\neg A) \cdot P(A|\neg BE) \cdot P(B|\neg E) \cdot P(\neg E) \right\}$$

$$= [P(J|A) \cdot P(M|A) \cdot P(A|BE) \cdot P(B|E)] + [P(J|\neg A) \cdot P(M|\neg A) \cdot P(A|\neg BE) \cdot P(B|\neg E)]$$

$$+ [P(J|A) \cdot P(M|A) \cdot P(A|\neg BE) \cdot P(B|E)] + [P(J|\neg A) \cdot P(M|\neg A) \cdot P(A|\neg BE) \cdot P(B|E)]$$

Way 2



2. What is the probability that Burglary happened given John & Mary called the police

$$P(B | J, M) = \frac{P(B, J, M)}{P(J, M)}$$

$$P(B | J, M) = \frac{\sum_{A, E} P(J, M, A, B, E)}{\sum_{A, B, E} P(J, M, A, B, E)}$$

$$P(\neg B | J, M) = \boxed{1 - P(B | J, M)}$$

$$P(B | J, M)$$

$$= \sum_{A} \sum_{E} P(J \wedge A \wedge B \wedge E)$$

$$= \sum_A P(J|A) \cdot P(M|A) \cdot P(B) \sum_E P(A|B \wedge E) \cdot P(E)$$

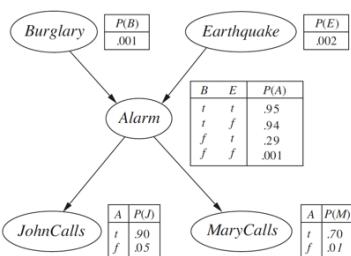
$$= P(B) \cdot \sum_A P(J|A) \cdot P(M|A) \cdot \left[ \sum_A P(A|B \wedge E) \cdot P(E) + P(A \neg B \wedge E) \cdot P(\neg E) \right]$$

$$= P(B) \cdot \left[ P(J|A) \cdot P(M|A) \cdot [P(A|B \wedge E) \cdot P(E) + P(A \neg B \wedge E) \cdot P(\neg E)] \right]$$

$$= P(B) \cdot \left[ P(J|A) \cdot P(M|A) \cdot [P(A|B \wedge E) \cdot P(E) + P(\neg A|B \wedge E) \cdot P(\neg E)] \right]$$

$$= \boxed{1 - P(B)}$$

BITs Pilani, Pilani Campus



3. What is the probability that John calls given earthquake occurred?

$$P(J | E) = \frac{P(J, E)}{P(E)}$$

$$P(J | E) = \frac{\sum_{M, A, B} P(J, M, A, B, E)}{\sum_{J, M, A, B} P(J, M, A, B, E)}$$

$$P(J | E) = \sum_{M, A, B} P(J | A) \cdot P(M | A) \cdot P(A | B \wedge E)$$

$$= P(E) \cdot \sum_M \sum_A P(J | A) \cdot P(M | A) \sum_B P(A | B \wedge E) \cdot P(B)$$

$$= P(E) \cdot \sum_A P(J | A) \sum_B P(A | B \wedge E) \cdot P(B)$$

$$P(J | E) + P(\neg J | E) = 1$$

$$P(J | E) + P(\neg J | E) = 1$$

$$\alpha = \frac{1}{P(E)} = \frac{1}{P(J | E) + P(\neg J | E)}$$

$$\text{Ans: } P(J | E) = \frac{P(J | E)}{P(J | E) + P(\neg J | E)}$$

BITs Pilani, Pilani Campus

:



# Artificial & Computational Intelligence

**DSECLZG557**

## **M6: Reasoning over time & Reinforcement Learning**

Indumathi V  
Guest Faculty,  
BITS - WILP

**BITS** Pilani  
Pilani Campus

# Course Plan

- M1 Introduction to AI
- M2 Problem Solving Agent using Search
- M3 Game Playing
- M4 Knowledge Representation using Logics
- M5 Probabilistic Representation and Reasoning
- M6 Reasoning over time
- M7 Ethics in AI

## Module 5:

# Probabilistic Representation and Reasoning

A. Inference using full joint distribution

$$\underline{\text{5 yrs}} \quad P(B|JM) \rightarrow \text{Joint}$$

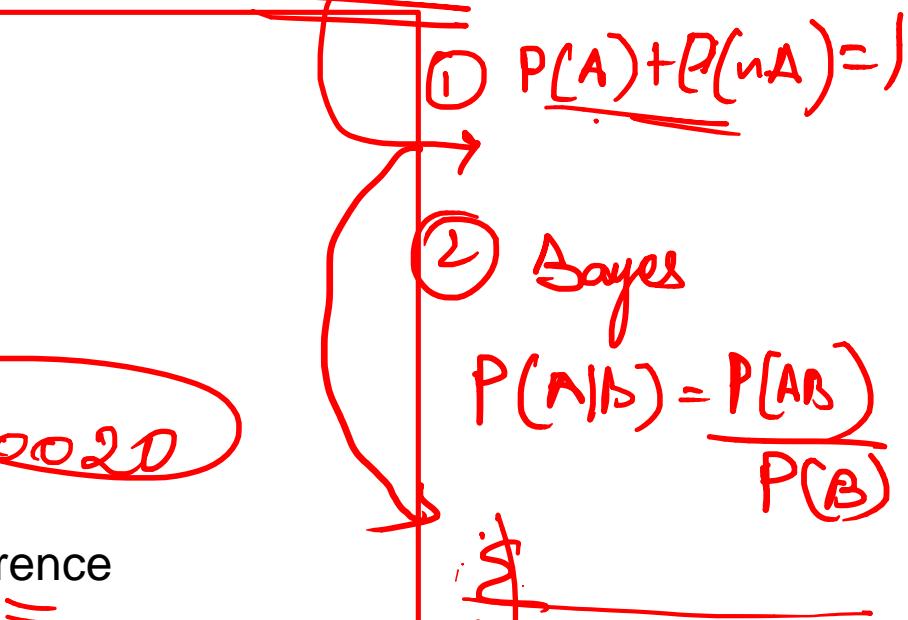
B. Bayesian Networks

I. Knowledge Representation ✓

II. Conditional Independence ✓

III. Exact Inference  $\Rightarrow 0.0020$

IV. Introduction to Approximate Inference



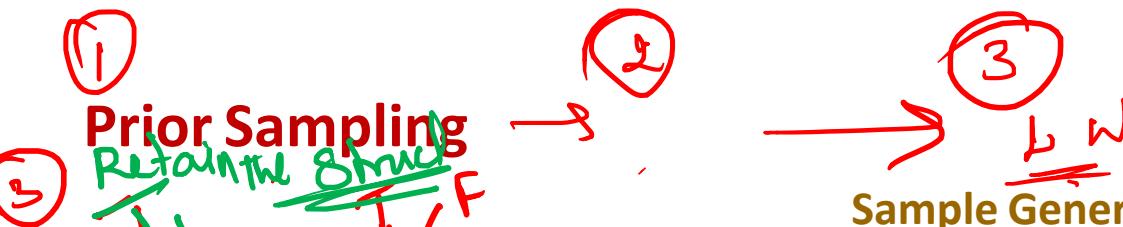
$$\underline{\text{Exp 1}} + \underline{\text{Exp 2}} + \underline{\text{Exp 3}} + \underline{\text{Exp 4}} = \underline{\text{A na E}}$$

# Approximate Inferences in Bayesian Nets

## Introduction



- Direct Sampling
- ↳ Prior sampling
- ↳ Rejection "
- ↳ Likelihood weighting



V	$\sim V$
T	0.6 0.4

Viral

V	$\sim V$
T	0.6 0.4

V	L	$\sim L$
T	0.8 0.2	

V	L	$\sim L$
F	0.01 0.99	

L	D	$\sim D$	CPT	$\sim F$
T	T	0.99	0.01	

L	D	$\sim D$	CPT	$\sim F$
T	F	0.8	0.2	

L	D	$\sim D$	CPT	$\sim F$
F	T	0.8	0.2	

L	D	$\sim D$	CPT	$\sim F$
F	F	0.05	0.95	

Random

no : 0.3 0.2, 0.6, 0.58,

0.73, 0.87, 0.15, 0.6, 0.57, 0.85, 0.12, 0.004, 0.93, 0.0002, 0.9, 0.55.....

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

V L D F

# Prior Sampling

## Sample Generation by Randomization

V	$\sim V$	
T	0.6	0.4



D	$\sim D$	
T	0.1	0.9

V	L	$\sim L$
T	0.8	0.2
F	0.01	0.99



L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95

V	L	D	F
T	T	F	T
F	F	F	F
T	F	F	T
F	T	F	T
..			
.....			

0.3, 0.2, 0.6, 0.58, 0.73, 0.87, 0.15, 0.6, 0.57, 0.85, 0.12, 0.004, 0.93, 0.0002, 0.9, 0.55.....



# Prior Sampling

V	$\sim V$
T	0.6
F	0.4



D	$\sim D$
T	0.1
F	0.9

V	L	$\sim L$
T	0.8	0.2
F	0.01	0.99

L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95



$$\checkmark P(L) = 3/8$$

$$P(F|L) = 3/8$$

$$P(L|F) = 3/5$$

$$P(\sim V|F) = 2/5$$

$$P(L|V\sim F) = 0$$

$$P(F|D) = \text{?????}$$

$$\textcircled{1} P(H|F) = \frac{\# \text{ matches w.r.t. query}}{\# \text{ matches w.r.t. evidence}} = \frac{L \rightarrow T}{F \rightarrow T} = \frac{3}{5}$$

$n \approx$

Inference

V	L	D	F
T	T	F	T
F	F	F	F
T	F	F	T
F	T	F	F
T	F	F	F
F	F	F	T
T	F	F	F

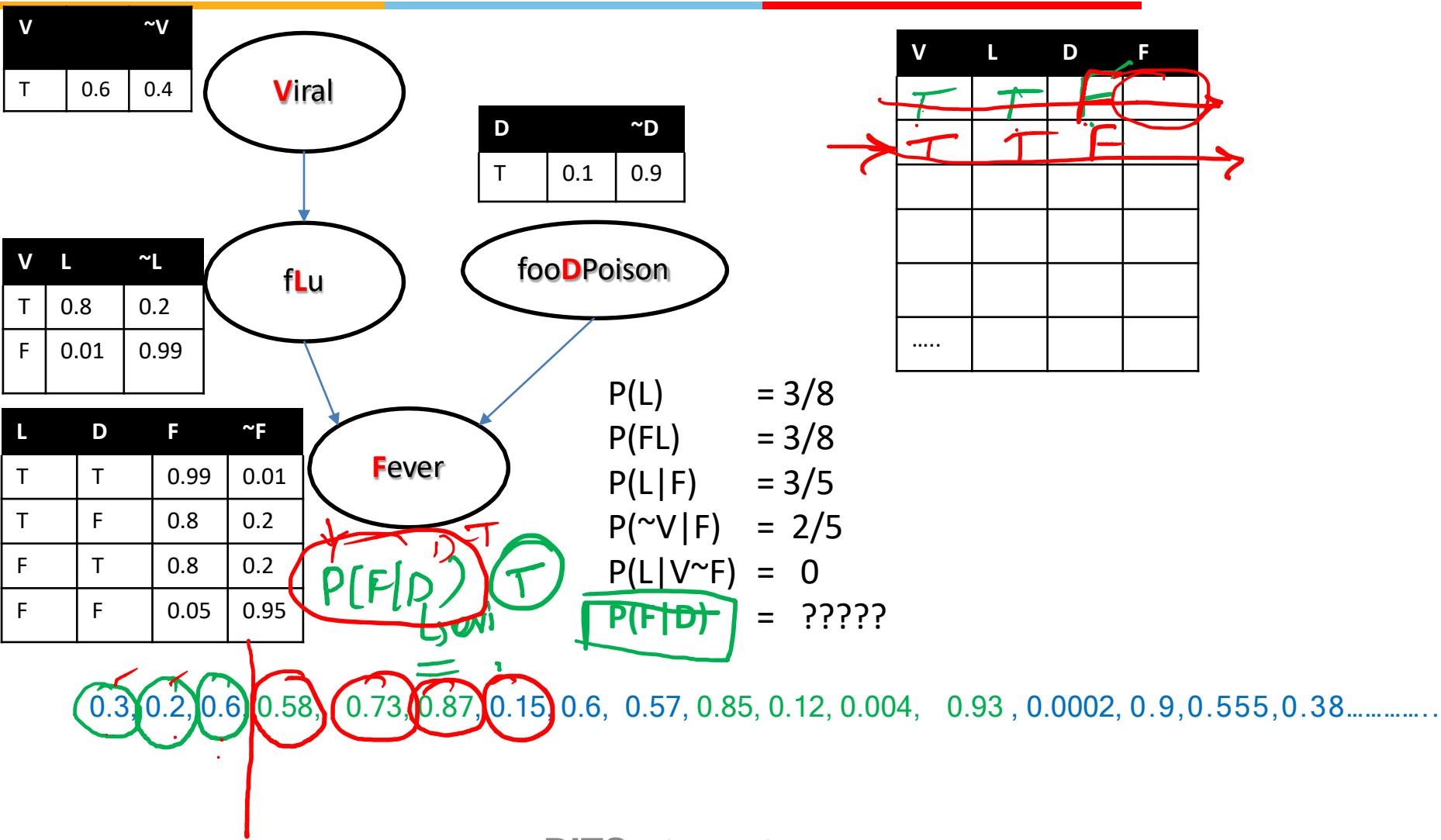
$$\frac{N}{F} = \frac{\text{False}}{\text{True}} \Rightarrow \frac{2}{5}$$

$$P(F|D) = ?$$

$$\textcircled{2} P(H|F) = \frac{\# \text{ matches w.r.t. query}}{\# \text{ matches w.r.t. evidence}} = \frac{L \rightarrow T}{F \rightarrow T} = \frac{3}{5}$$

# Rejection Sampling

## Sample Generation by Randomization



# Rejection Sampling

## Sample Generation by Randomization

N=100

V	$\sim V$	
T	0.6	0.4



D	$\sim D$	
T	0.1	0.9

V	L	$\sim L$
T	0.8	0.2
F	0.01	0.99



L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95



$$\begin{aligned}
 P(L) &= 3/8 \\
 P(F) &= 3/8 \\
 P(L|F) &= 3/5 \\
 P(\sim V|F) &= 2/5 \\
 P(L|V\sim F) &= 0 \\
 P(F|D) &= ??????
 \end{aligned}$$

V	L	D	F
T	T	F	
T	T	F	
T	T	F	
F	F	T	F
T	F	T	T

almost

N | L | D | V |   
 0.3, 0.2, 0.6 | 0.58, 0.73, 0.87 | 0.15, 0.6, 0.57 | 0.85, 0.12, 0.004, 0.93 | 0.0002, 0.9, 0.555, 0.38 .....  
 = = = = =

## Rejection Sampling

$\textcircled{1} \ P(\theta | e_1, e_2, e_3) \quad \text{Inference}$

V	$\sim V$
T	0.6
F	0.4



D	$\sim D$
T	0.1
F	0.9

V	L	$\sim L$
T	0.8	0.2
F	0.01	0.99

L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95



$$P(L) = 3/8$$

$$P(FL) = 3/8$$

$$P(L|F) = 3/5$$

$$P(\sim V|F) = 2/5$$

$$P(L|V\sim F) = 0$$

$$\boxed{P(F|D) = 5/8}$$

V	L	D	$\sim F$
T	T	T	T
F	F	T	F
T	L	F	T
F	T	T	F
T	T	T	T
T	F	T	F
F	F	T	T
T	F	T	F

$$\frac{5}{8}$$

# Rejection Sampling

## Sample Generation by Randomization

V	$\sim V$	
T	0.6	0.4



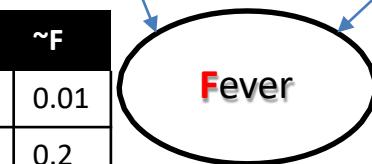
D	$\sim D$	
T	0.1	0.9

V	L	$\sim L$
T	0.8	0.2
F	0.01	0.99



fooDPoison

L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95



$$P(\sim V | F) = ?$$

V	L	D	F
T	T	F	T
F	F	F	
T	F	F	T
F	T	F	T
.....			

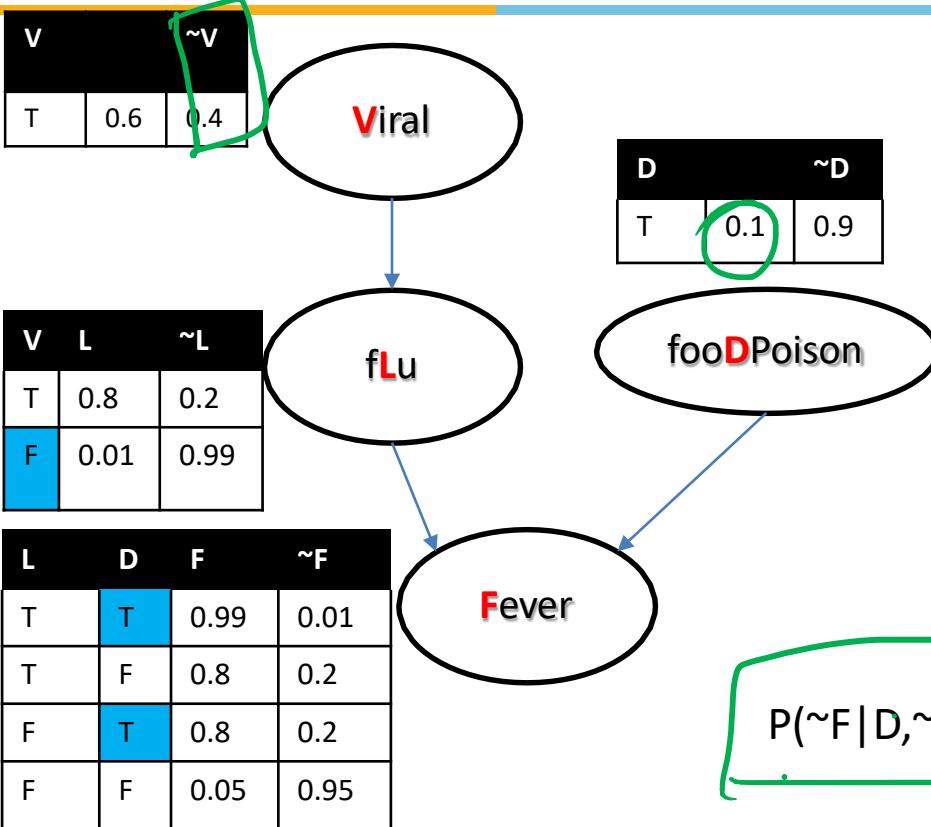
0.3, 0.2, 0.6, 0.58, 0.73, 0.87, 0.15, 0.6, 0.57, 0.85, 0.12, 0.004, 0.93, 0.0002, 0.9, 0.555, .....



evidence val  $\rightarrow$  wt from Prob Table  
 non evi val  $\rightarrow$  wt  $\rightarrow 1$



## Likelihood Weighing



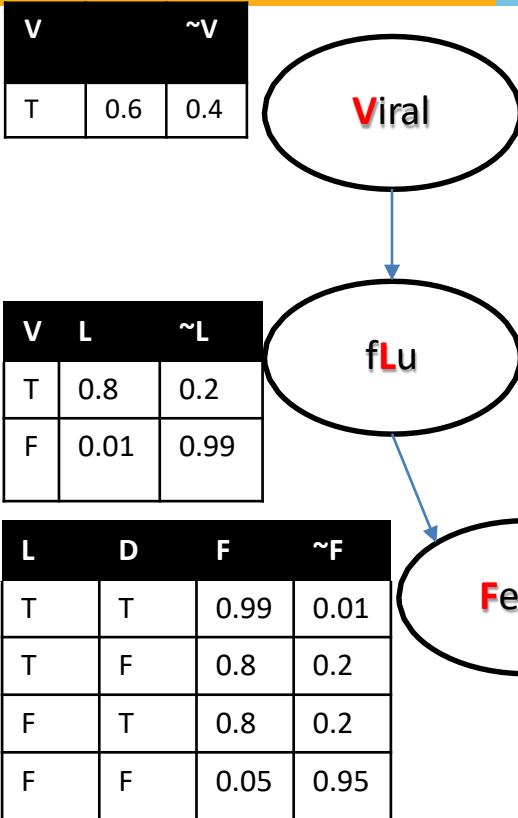
Sample Generation by Randomization

V	L	D	F	wgt
F	F	T	T	$0.4 * 0.1 * 0.01 = 0.004$
F	F	F	T	$0.4 * 1 * 0.01 = 0.04$
F	F	T	T	"
F	F	T	T	"
F	F	T	T	"
F	T	T	F	$0.04$

$E = 0.04 / 7 * 0.04$

$0.04 / 7 * 0.04 = 0.04 / 0.04 = 1$

# Likelihood Weighing



## Inference

Handwritten annotations:

- 1
- 2
- 3
- 4

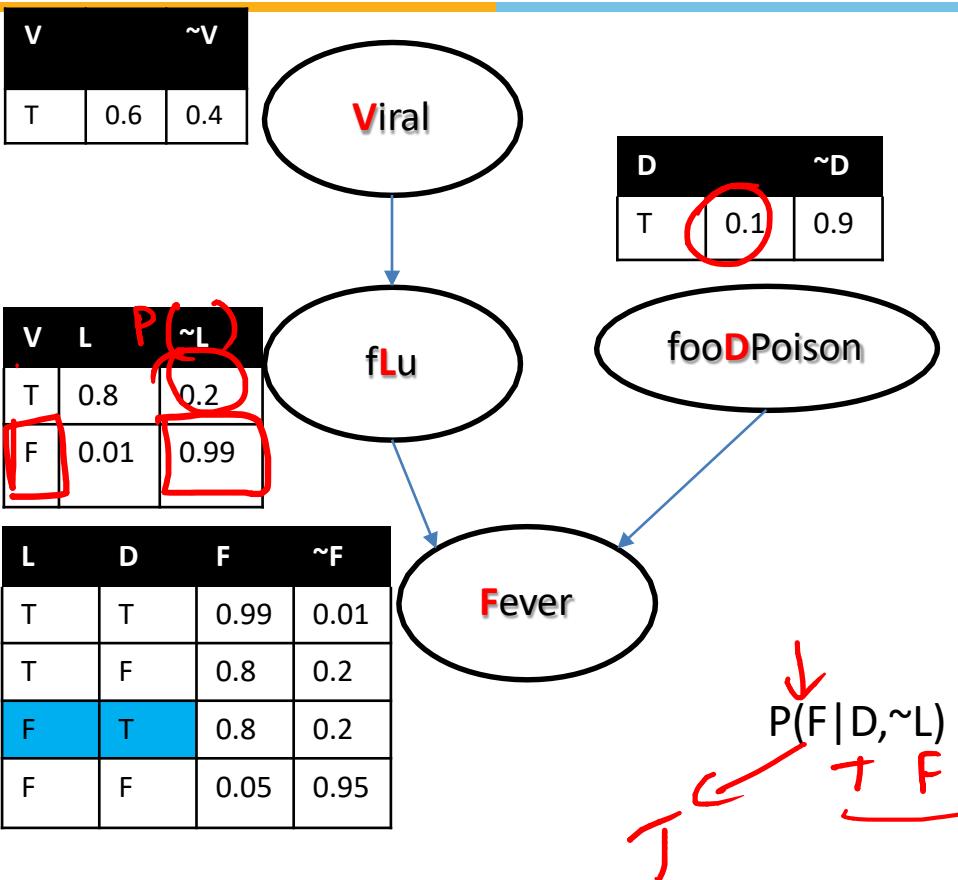
Probability table for inference:

V	L	D	F	wgt
F	F	T	F	0.4*1* 0.1 *1=
F	T	T	T	0.4*1* 0.1 *1=
F	F	T	T	0.4*1* 0.1 *1=
F	F	T	F	0.4*1* 0.1 *1=

$$P(F | D, \sim V)$$

$$= 0.04 + 0.04 / 4 * 0.04$$

# Likelihood Weighing



↓ Ne Inference ↓ NO ↓

V	L	D	F	wgt
F	I	F 0.99	T 0.1	$(1 * 0.99 * 0.1)$
F	I	F 0.99	T 0.1	"
F	I	F 0.99	T 0.1	"
T	I	F 0.2	T 0.1	$(1 * 0.2 * 0.1)$

$$\begin{aligned}
 Q_1 &= \frac{3 * 0.099 + 0.02}{(3 * 0.099 + 0.02)} \\
 &= 0.099 + 0.099 / 0.099 + 0.02 \\
 &\text{ev}
 \end{aligned}$$

# Likelihood Weighing

V	$\sim V$
T	0.6
F	0.4

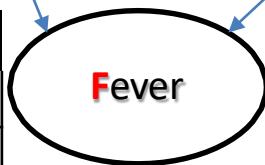


D	$\sim D$
T	0.1
F	0.9

V	L	$\sim L$
T	0.8	0.2
F	0.01	0.99



L	D	F	$\sim F$
T	T	0.99	0.01
T	F	0.8	0.2
F	T	0.8	0.2
F	F	0.05	0.95



$$P(F | D, \sim L)$$

## Inference

V	L	D	F	wgt
F	F	T	F	$1 * 0.99 * 0.1 * 1 =$
F	F	T	T	$1 * 0.99 * 0.1 * 1 =$
F	F	T	T	$1 * 0.99 * 0.1 * 1 =$
T	F	T	F	$1 * 0.2 * 0.1 * 1 =$

$$= 0.099 + 0.099 / (3 * 0.099 + 0.02)$$

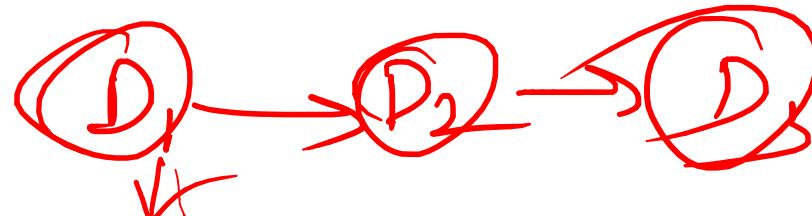
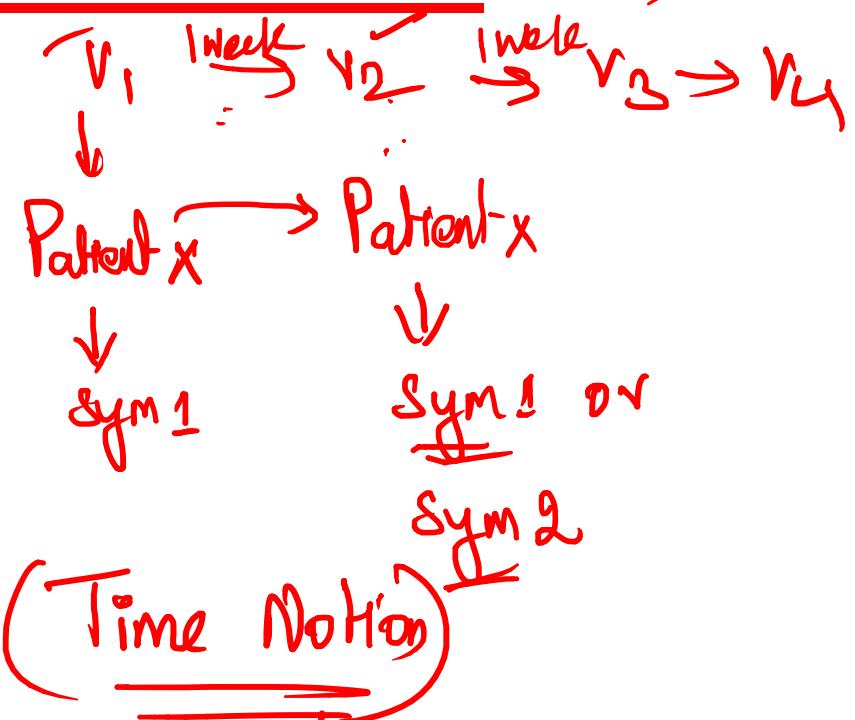
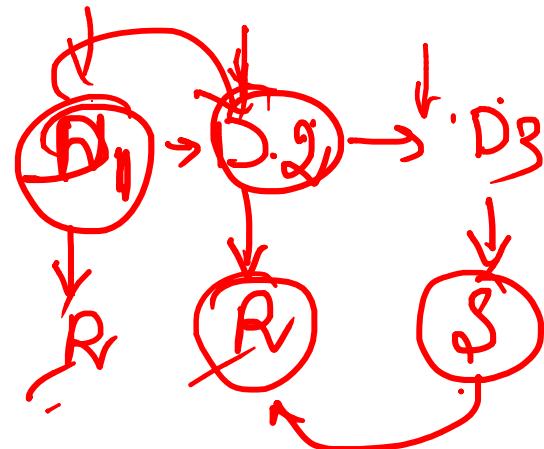
# Course Plan

- M1 Introduction to AI
- M2 Problem Solving Agent using Search
- M3 Game Playing
- M4 Knowledge Representation using Logics
- M5 Probabilistic Representation and Reasoning
- M6 Reasoning over time ↗
- M7 Ethics in AI

# Module 6: Reasoning over time & Reinforcement Learning

## Reasoning Over Time

- A. Time and Uncertainty ✓
- B. Inference in temporal models ↗
- c. Overview of HMM ✓



## Learning Objective

---

1. Understand the relationship between Time & Uncertainty
  1. Recognize the transition model of Markov Model
  1. Relate to the application of the Hidden Markov Model
-

# Module 6:

## Reasoning over time

### Reasoning Over Time

- A. Time and Uncertainty
- B. Inference in temporal models
- C. Introduction to Hidden Markov Model
- D. Applications of HMM

# Time & Uncertainty

## Learning Objective

- 
1. Understand the relationship between Time & Uncertainty
  1. Recognize the transition model of Markov Model
  1. Relate to the application of the Hidden Markov Model
-

# Sequential Decision Problems & Markov Decision Process

MDP → Max

# Markov Decision Process



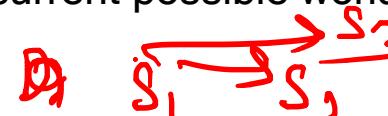
Modelling sequences of random events and transitions between states over time is known as Markov chain

Agents in partially observable environment should keep a track of current state to the extent allowed by sensors

E.g., Robot moving in a new maze

$$P(c|c) \Rightarrow 0.40$$

Agent maintains a **belief state** representing the current possible world states



~~Transition Model~~ ~~Probability Matrix~~ :

Using belief state and transition model, the agent can know how the world might evolve in next time step. To capture the degree of belief we will use Probability Theory. We model the change in world using a variable for each aspect of state and at each point in time.

Current state depends only finite number of previous states.

Previous

<del>C</del>	M	C	<del>M</del>
0.40	0.20	0.60	0.80

Current

# Markov Decision Process

## Time - Uncertainty | States - Observations

**Static World:** Each random variable would have a single fixed value

E.g., Diagnosing a broken car

**Dynamic World:** The state information keeps changing with time

E.g., treating a diabetic patient, tracking the location of robot, tracking economic activity of a nation

**Time slices:** World is observed in time slices. Each slice has a set of random variables, some observable and some not.

Assumption: We will assume same subset of random variables are observable in each time slice

$E_t$  - set of observable random variables at time t

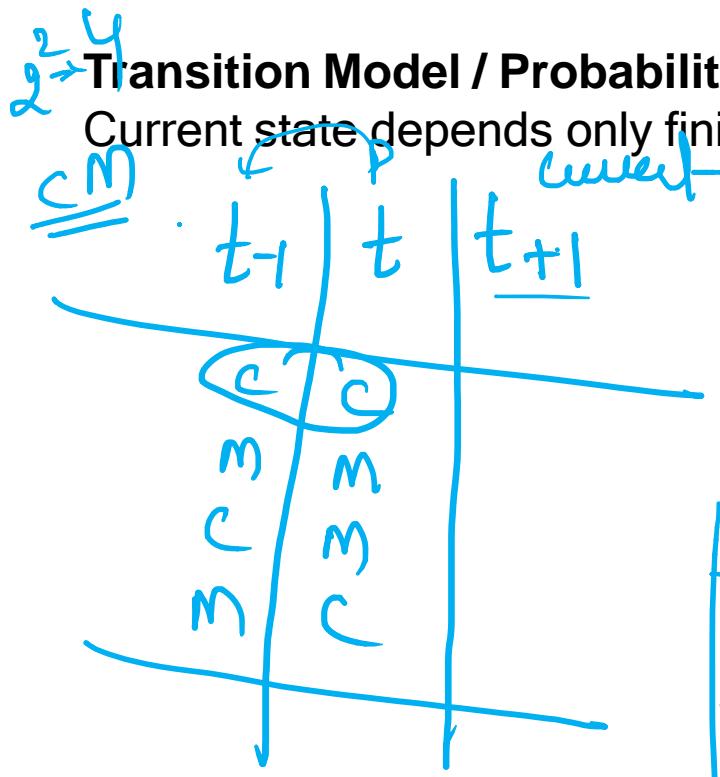
$X_t$  - set of unobserved random variables at time t

C	M	
0.40	0.20	C
0.60	0.80	M

# Markov Process

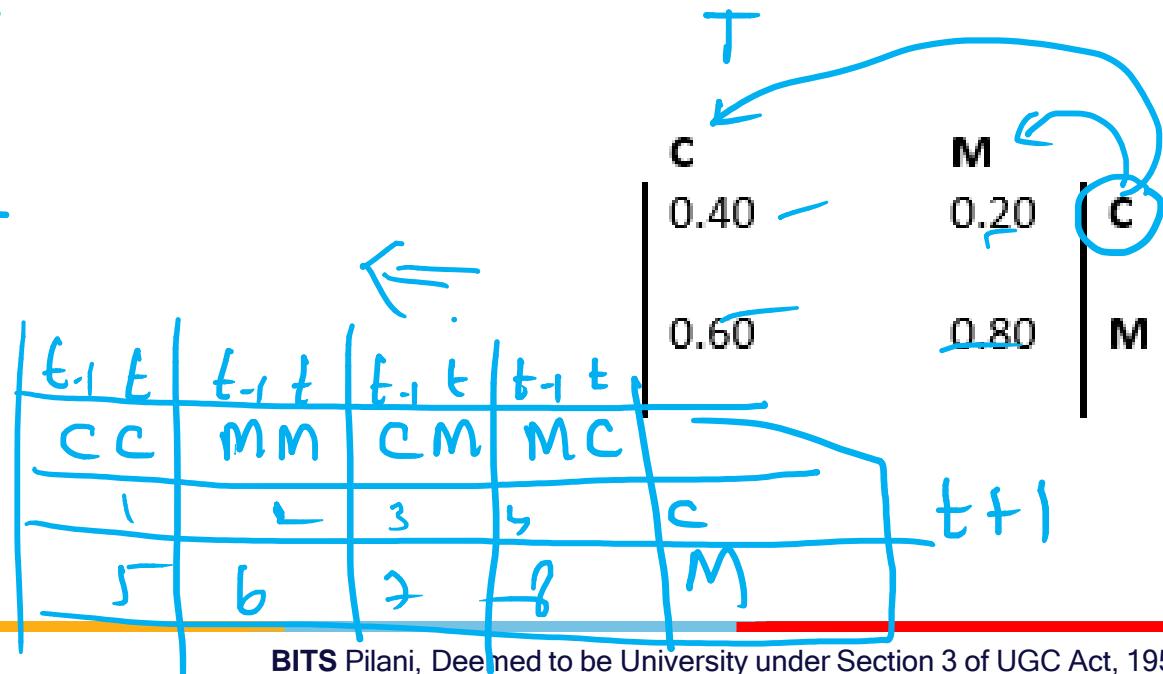
## States | Observations | Assumptions

Modelling sequences of random events and transitions between states over time is known as Markov chain



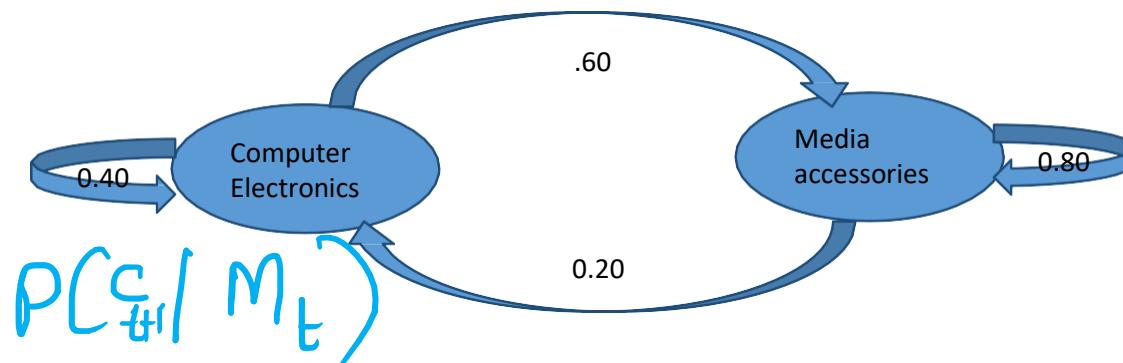
### Transition Model / Probability Matrix :

Current state depends only finite number of previous states. :



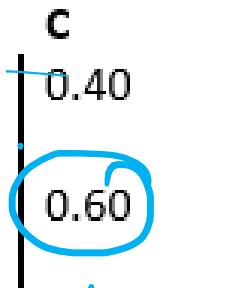
## Markov Model- Example 1

2 products C M

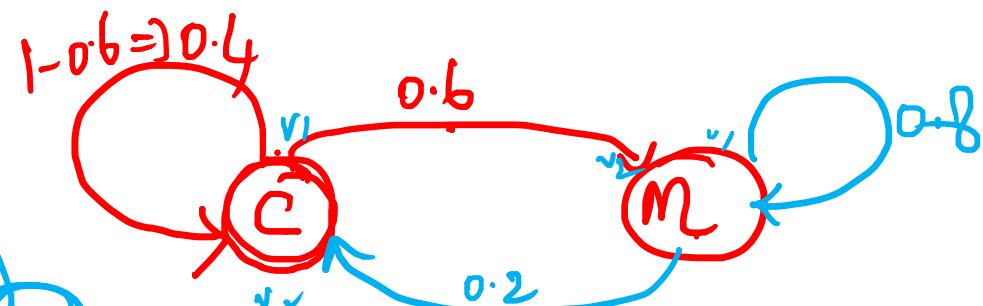


$$P(C_{t+1} | M_t)$$

Transition Model



$$P(M_{t+1} | C_t)$$

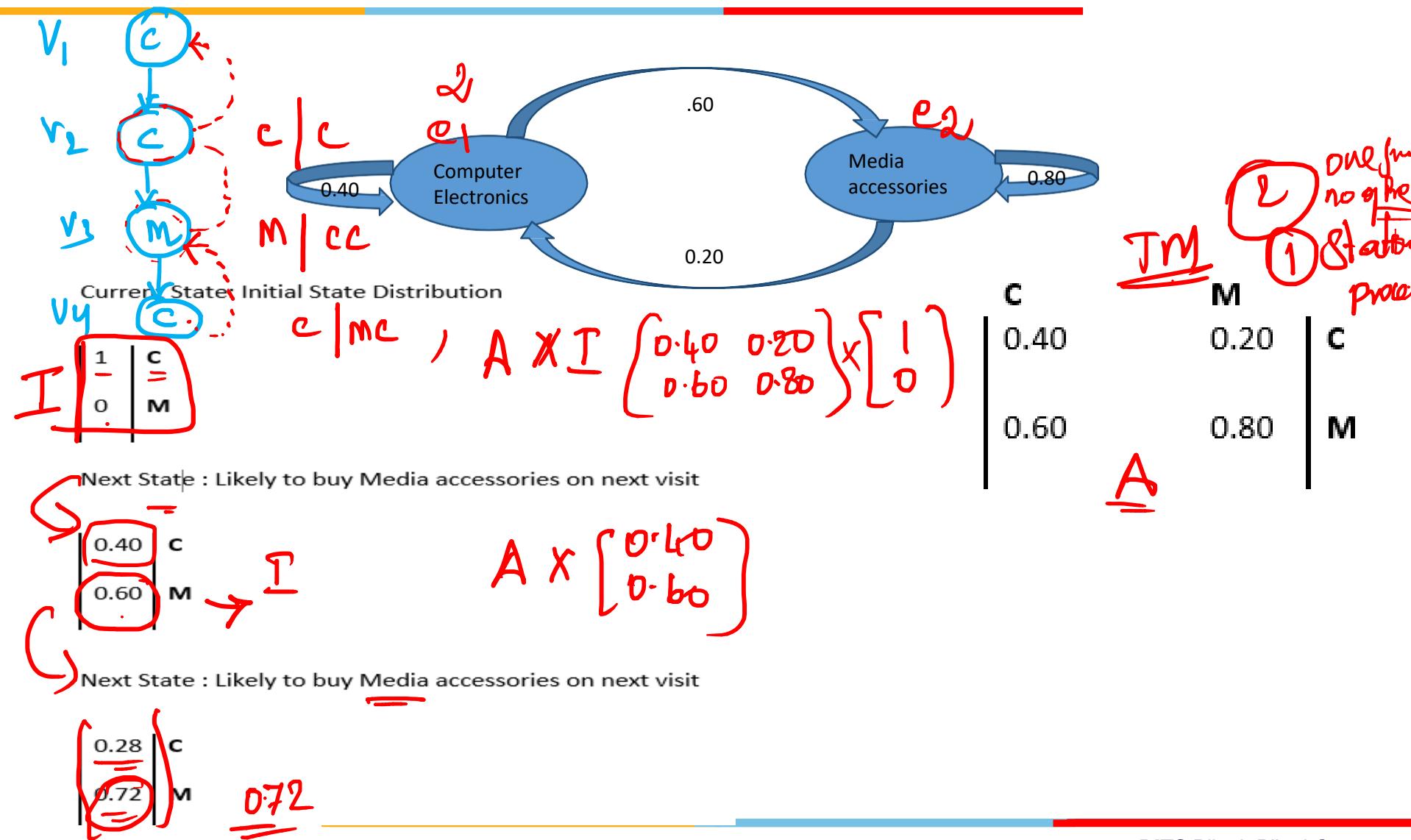


1<sup>st</sup> Markov Model  
2<sup>nd</sup> Markov Model  
Current → depends on  
the previous state

current → depends  
only on one previous

1st

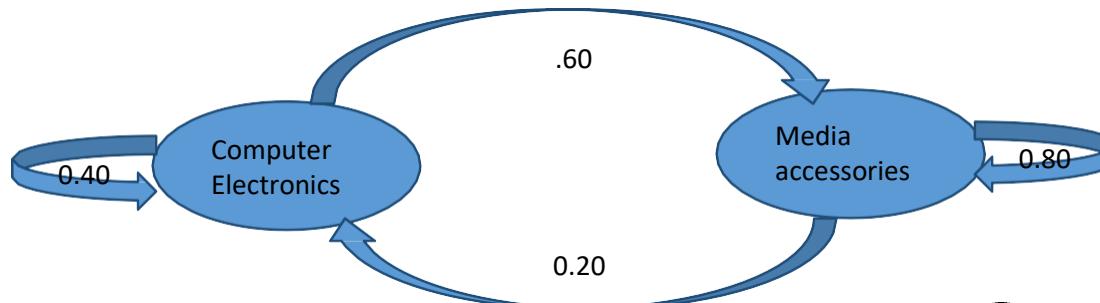
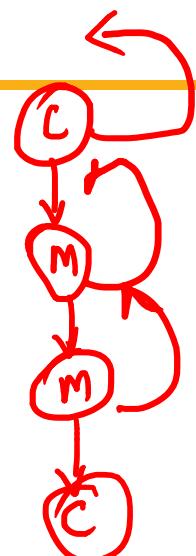
## Markov Model



# Inference in temporal Models

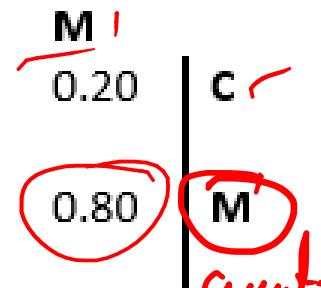
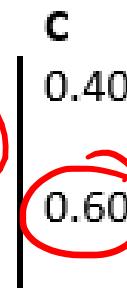
# Markov Model

Inference Type 1



Prev

$$P(C, M, M, C) \Rightarrow P(C) * P(M|C) * P(M|M) * P(C|M)$$



- ① What is the probability that the purchasing behaviour of the customer is in below sequential order only? Initial Probability Matrix is  $P(C) = 1, P(M) = 0$
- (Computer, Media, Media, Computer)  
 $v_1 \quad v_2 \quad v_3 \quad v_4$

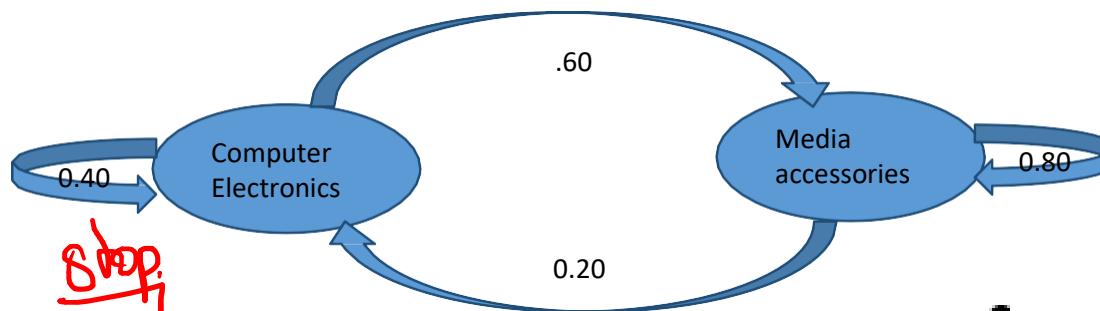
Apply Bayes chain rule:

$$P(\text{Computer, Media, Media, Computer}) = P(C) * P(M|C) * P(M|M) * P(C|M) = 0.096$$

$$\frac{1 * 0.6 * 0.8 * 0.2}{1 * 0.6 * 0.8 * 0.2} = 1$$

# Markov Model

## Inference Type 2



C	M	C
0.40	0.20	C
0.60	0.80	M

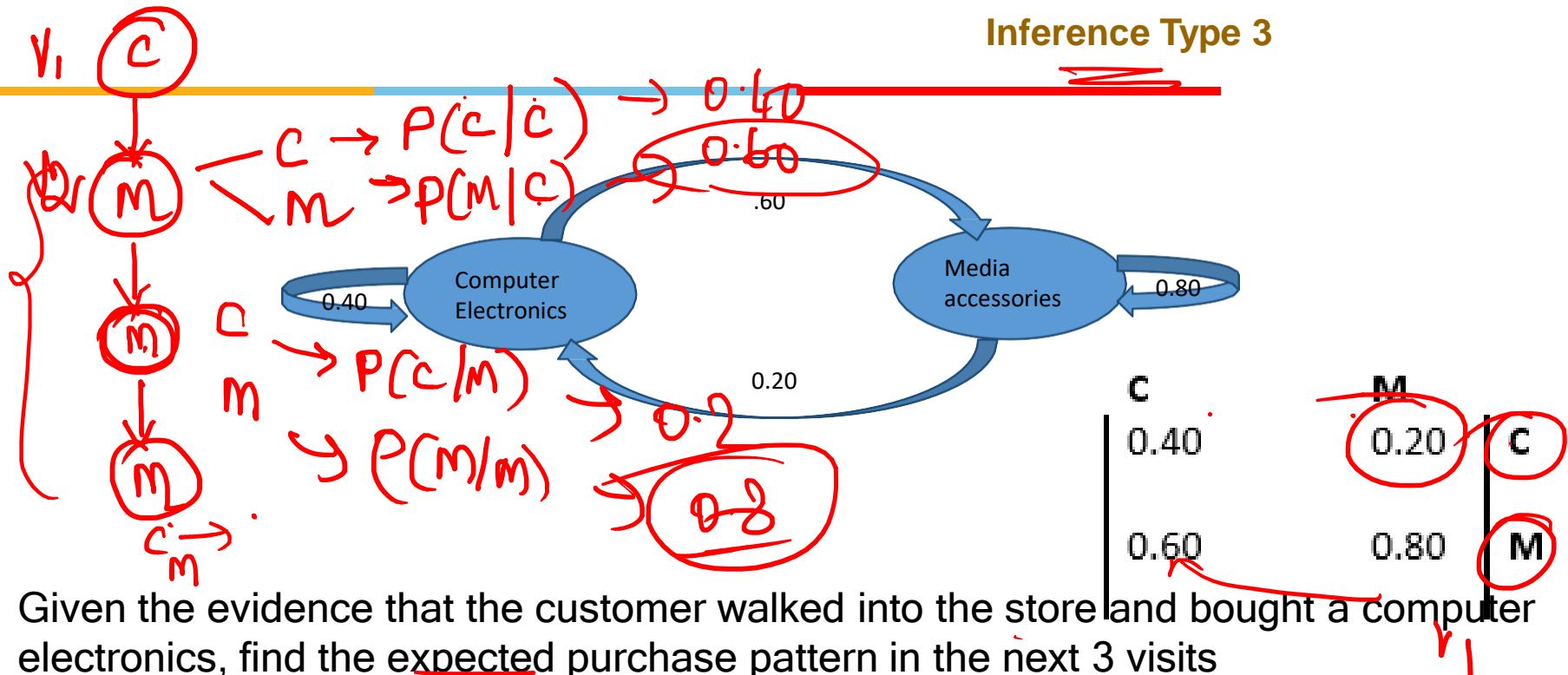
2) What is the probability that the customer who purchased Media accessories will keep coming back to purchase media accessories in the next 2 consecutive visits?

Derive Initial prob values & Apply Bayes chain rule on the pattern exhibited:

Initial Probability Matrix is  $P(M) = 1, P(C) = 0$

$$P(\text{Media, Media, Media, Computer}) = P(M) * P(M|M) * P(M|M) * P(C|M) = 0.128$$

# Markov Model



Derive Initial prob values & Apply Bayes chain rule and reverse predict the combination on the most likely pattern (Similar to Viterbi Algorithm):

Initial Probability Matrix is  $P(C) = 1, P(M) = 0$

$$P(\text{Computer}, X, Y, Z) = P(\text{Computer}) * P(X|\text{Computer}) * P(Y|X) * P(Z|X) =$$

$1 * 0.6 * 0.8 * 0.8 \rightarrow \text{Produces max values}$

Ans : Pattern = (Computer, Media, Media, Media)

---

**Required Reading:** AIMA - Chapter #15.1, #15.2, #15.3, #20.3.3

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

DSE CLZG557

## M6: Reinforcement Learning

Indumathi V

Guest Faculty,

BITS -WILP

**BITS** Pilani  
Pilani Campus

# Module 6:

# Reinforcement Learning

- I. Introduction
- II. Q-Learning algorithm
- III. Applications of reinforcement learning to games / search

# Agent Architectures

Simple Reflex Agents



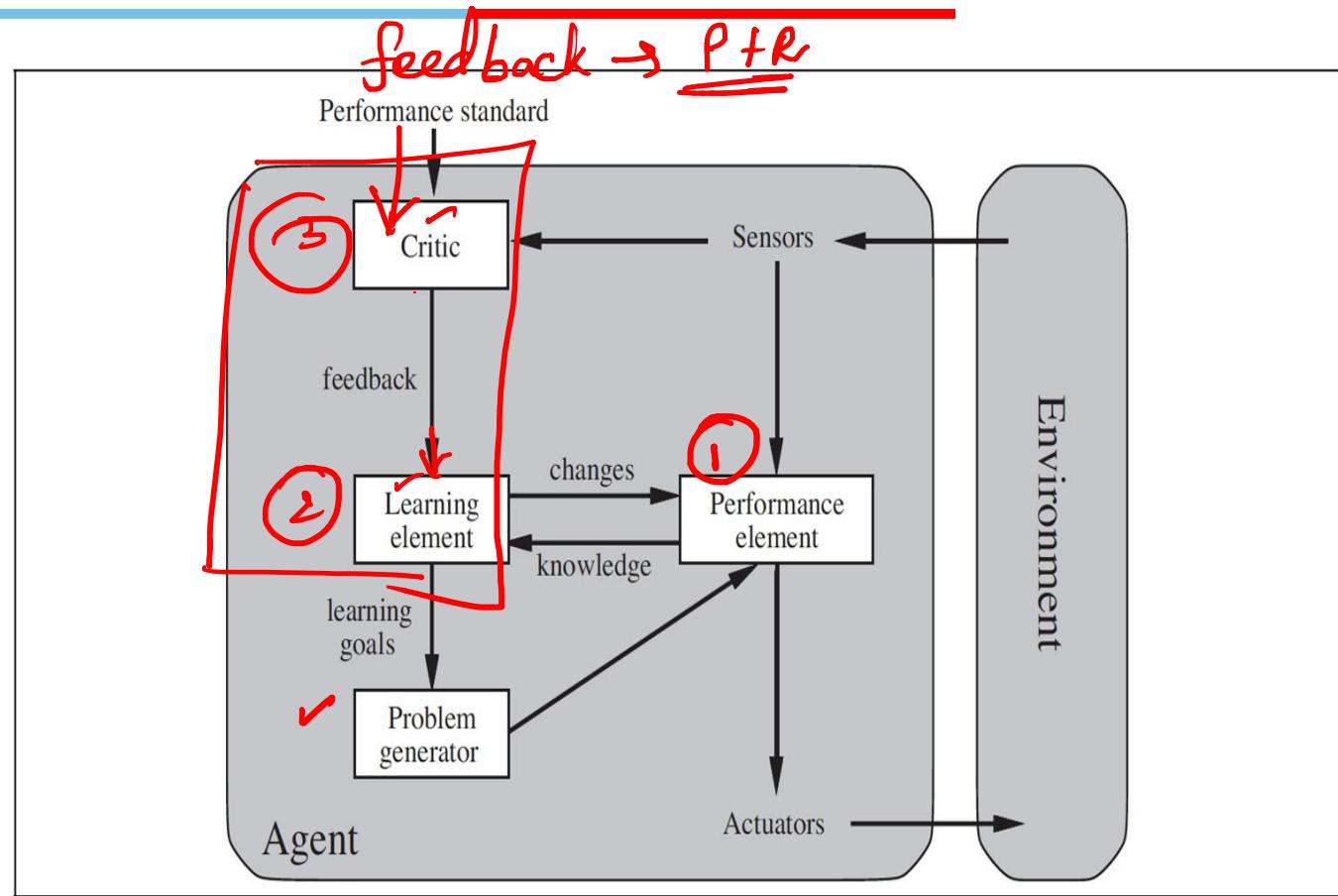
Model Based Agents



Goal Based Agents

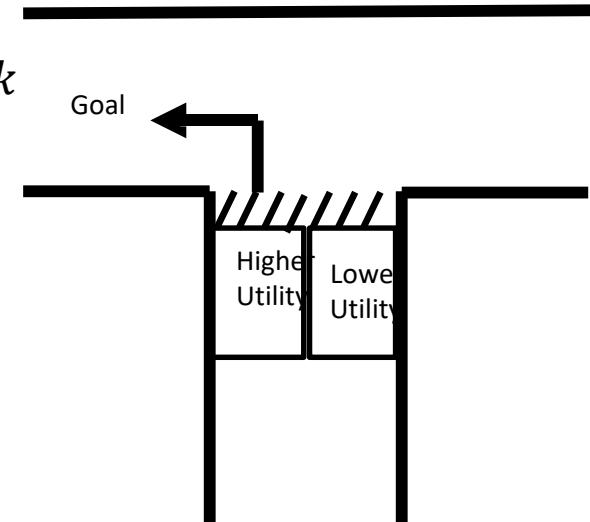


Utility Based



## Role of Learning

Performance Element – Takes decision on action based on percept

$$f(\text{red signal}, \quad \text{distance} = 15k \text{ N brak } \\ (\text{ e } \quad \quad \quad ) \\ \text{distance} = f' \text{ percept sequence } \\ f \text{ percepts, distance, raining} \\ (\quad \quad \quad ) \\ - (\quad f \text{ state}_0, a) = 0.45 \\ \text{ctionB}$$


Learning Element – Make the performance element select better actions such that the utility function is optimized

Critic – Provides feedback on the actions taken

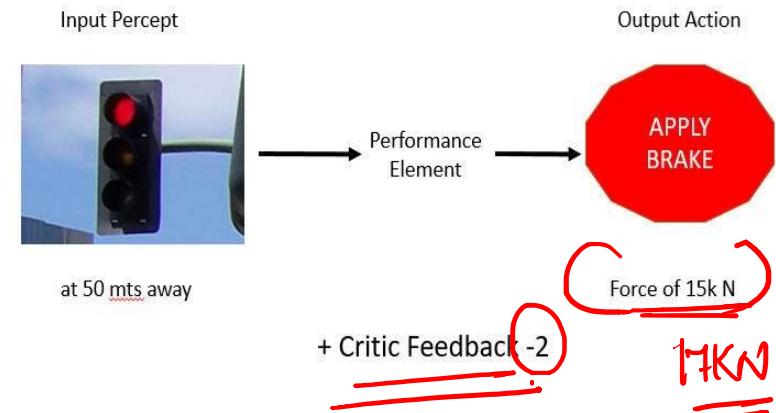
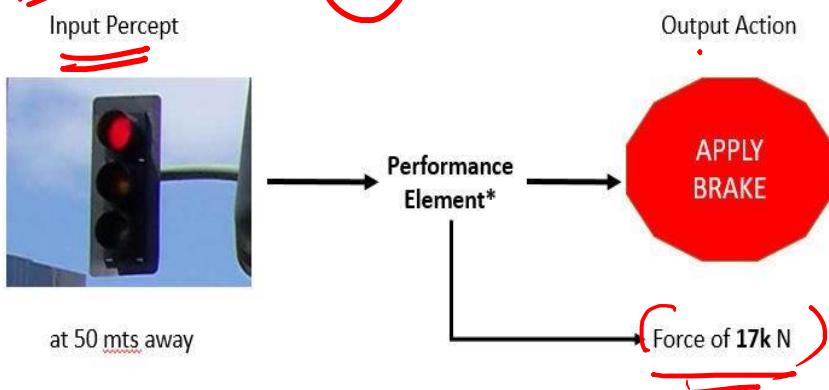
Problem Generator – Make the Performance Element select sub-optimal actions such that you would learn from unseen actions

# Role of Learning

Critic – Provides feedback on the actions taken

Learning :

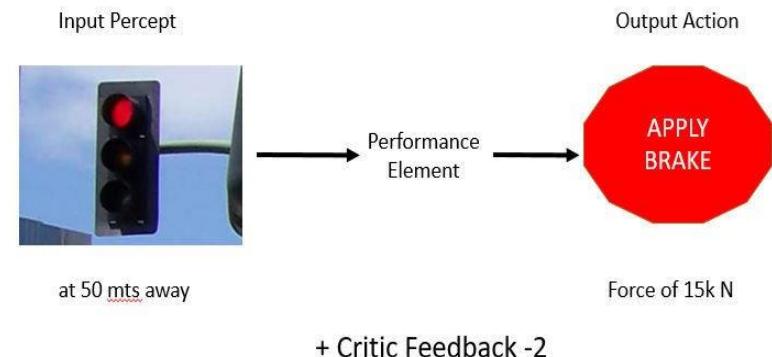
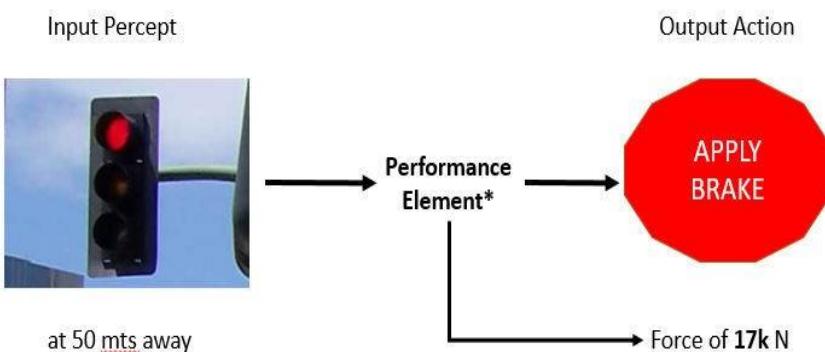
- ✓ Supervised Vs Unsupervised Vs
- ✓ Reinforcement 3.



# Towards Reinforcement Learning

# Supervised Vs Unsupervised Vs Reinforcement Learning

<b>Feedback = - 2 Suggestion = Apply Brake @ XXX</b>	<b>No Feedback</b>	<b>Feedback = -2 (Delayed Feedback)</b>
Supervised	Unsupervised	Reinforcement



# Reinforcement Learning

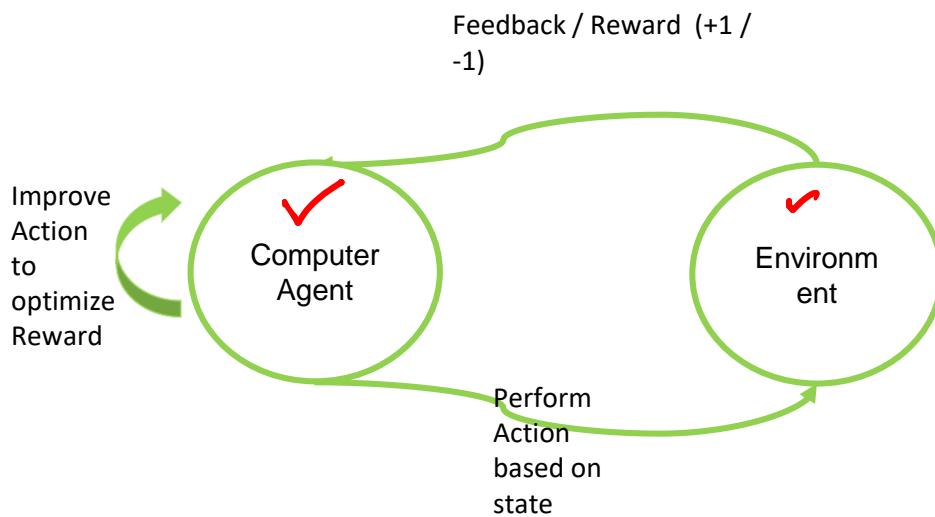


The goal : To Identify an optimal policy that maximizes the expected total reward

Path = Searching → path  
Gramy → Strategy

Task: Use observed rewards to learn an optimal policy for the environment

V - Q - learning



Action Critic

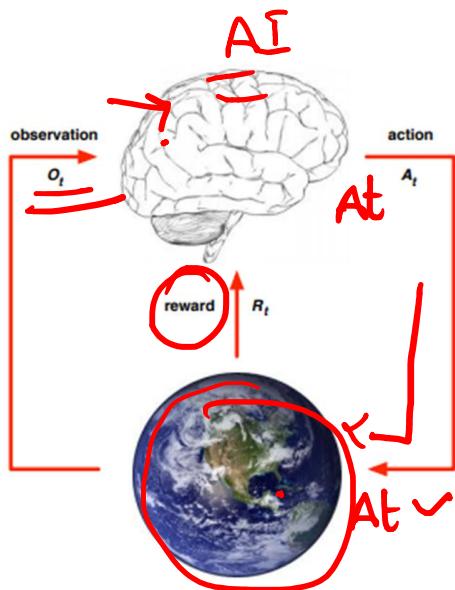


# Reinforcement Learning

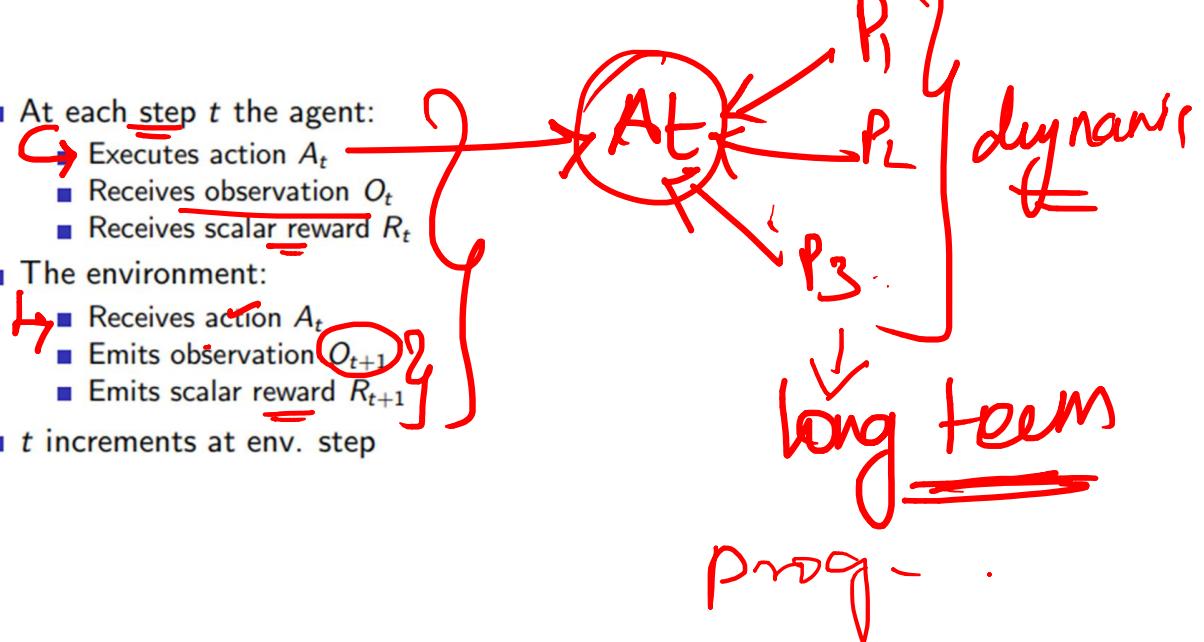


Goal oriented learning

MD



- At each step  $t$  the agent:
  - ↳ Executes action  $A_t$
  - Receives observation  $O_t$
  - Receives scalar reward  $R_t$
- The environment:
  - Receives action  $A_t$
  - Emits observation  $O_{t+1}$
  - Emits scalar reward  $R_{t+1}$
- $t$  increments at env. step



# Reinforcement Learning

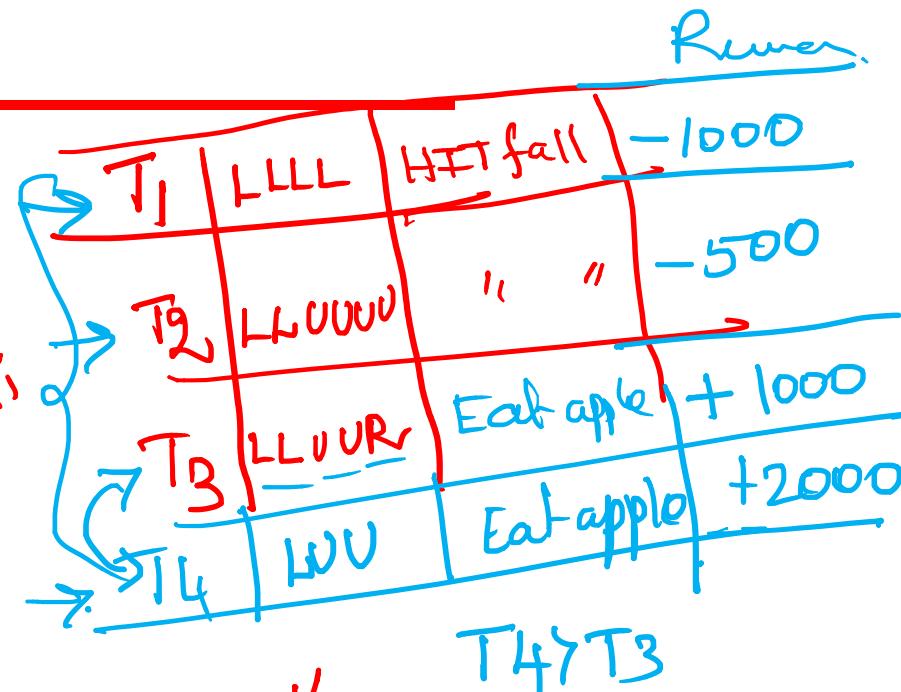
At each step 't' the agent :

- Executes an action  $A_t$
- Receives observation  $O_t$
- Receives scalar Reward  $R_t$

**Components:**

- Policy
- Value Function
- Model

seq st train



Sample Instructions of the Snake Game:

- Eat apple(red circle) to get points.
- Eat cherries(pink circle to get bonus points).
- If you hit wall or eat yourself you die.
- Controls are UP, DOWN, LEFT, RIGHT

Termination



L V R  
D

# Reinforcement Learning

## Components: Value Function

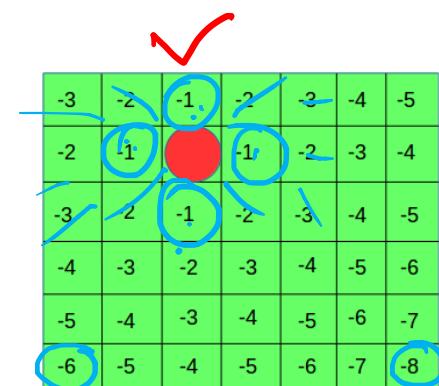
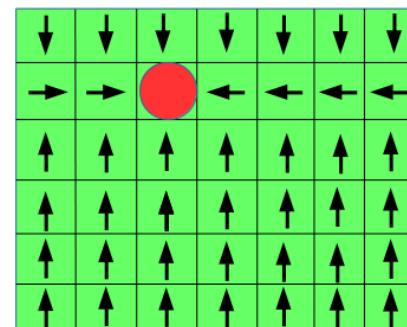
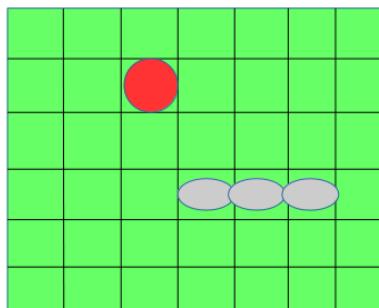
$$V_{\pi}(s) = E_{\pi}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots | S_t = s]$$

's' is the current state

' $\pi$ ' is the policy

'R' is the reward

' $\gamma$ ' is the discount



# Reinforcement Learning

## Components:

- Policy : ' $\Pi$ ' is just the policy function that simply gives an output state given your current state:
- Stochastic Policy
- Deterministic Policy :

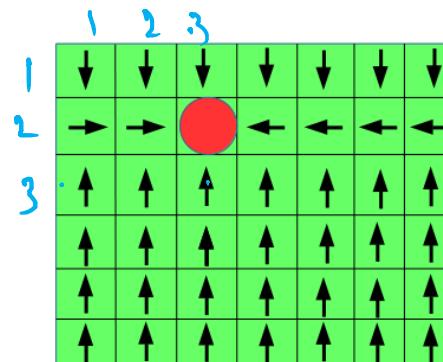
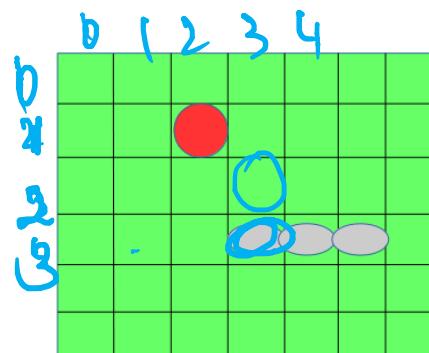
$$T((3,3), V) \Rightarrow (2,3)$$

$$\Pi(a|s) = P[A = a|S = s]$$

$$a = \Pi(s)$$

$$T((3,3), U)$$

2,3	b-b
2,2	0.4



# Reinforcement Learning

At each step 't' the agent :

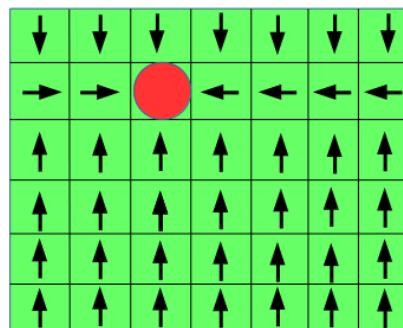
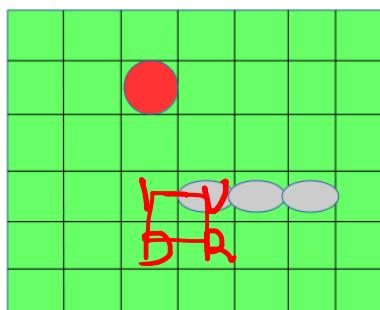
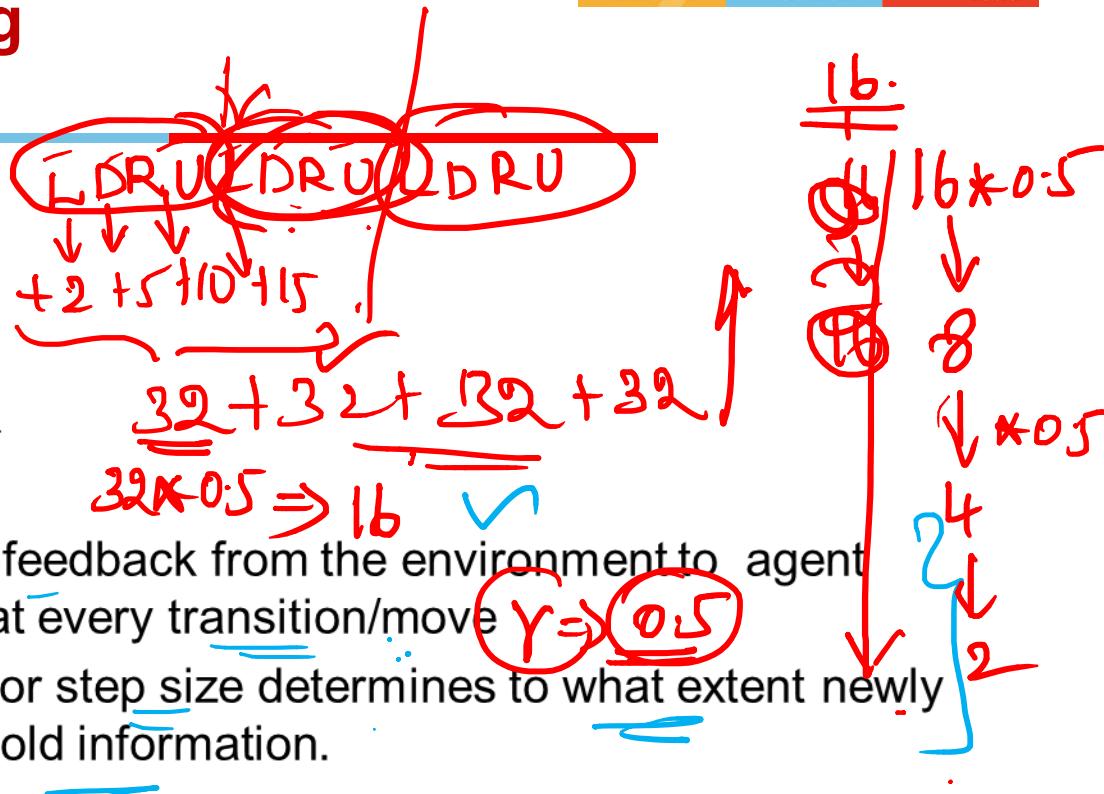
- Executes an action  $A_t$
- Receives observation  $O_t$
- Receives scalar Reward  $R_t$

## Components:

Reward: Utility value which is the feedback from the environment to agent

Discount: Probability to succeed at every transition/move

Learning Rate: The learning rate or step size determines to what extent newly acquired information overrides old information.



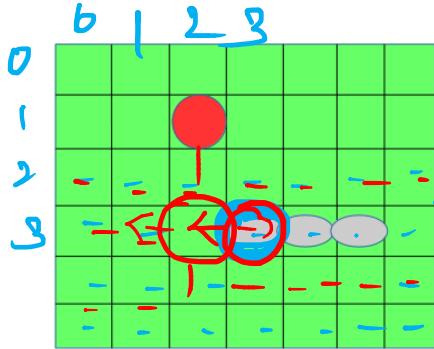
-3	-2	-1	-2	-3	-4	-5
-2	-1	0	0	-1	-2	-3
-3	-2	-1	-2	-3	-4	-5
-4	-3	-2	-3	-4	-5	-6
-5	-4	-3	-4	-5	-6	-7
-6	-5	-4	-5	-6	-7	-8

# Reinforcement Learning : Q-Learning

## Multiple actions – Single state transition

~~1,000~~

Action : moveLeft or moveRight or moveUp or moveDown.



$V_{3,3}$  old info

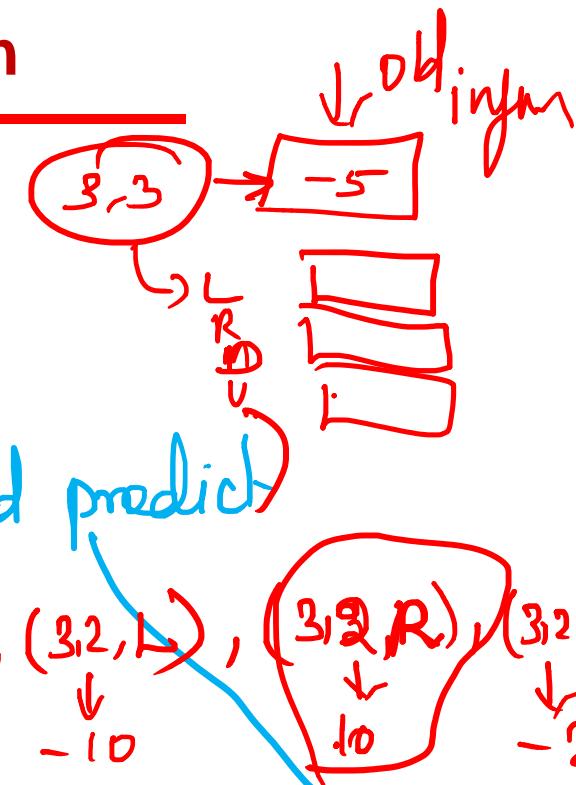
1 left action

$((V_{3,3}), L)$  one step ahead predict

$\max((3,2), (3,2, R), (3,2, D))$

$\max$

(-5)



• U, D, L, R

Bellman's equation

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \alpha \cdot \underbrace{\left( r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right)}_{\text{estimate of optimal future value}}$$

+ learning rate

~~if  $\alpha = 1$ , Exploration  
 $\alpha = 0$ , Exploitation  
0.6 to 0.9~~

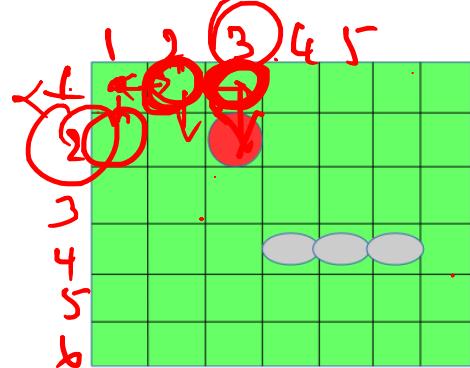
~~Exploration~~

# Reinforcement Learning

## Multiple actions – Single state transition

moveLeft or moveRight etc., . . .

Transition Model ✓



D

Location1,1 Top left	Location1,2	Loc2,1	Loc7,7 Bottom Right	Loc2,3 Apple	
No transition possible	1,1	.....	7,6	2,2	moveLeft
1,2	1,3	.....	No transition possible	2,4	moveRight
No transition possible	No transition possible	.....	6,7	1,3	moveUp
2,1	2,2	.....	No transition possible	3,3	moveDown

Reward table      Sample

Location1,1	Location1,2	Loc2,1	Loc5,5	Loc2,3 Apple	Sample Value similar to adjacent setup
-20	-10	-10	-10	+100	Reward

-3	-20	-1	-2	-3	-4	-5
-2	-1	-1	-2	-3	-4	-4
-3	-2	-1	-2	-3	-4	-5
-4	-3	-2	-3	-4	-5	-6
-5	-4	-3	-4	-5	-6	-7
-6	-5	-4	-5	-6	-7	-8

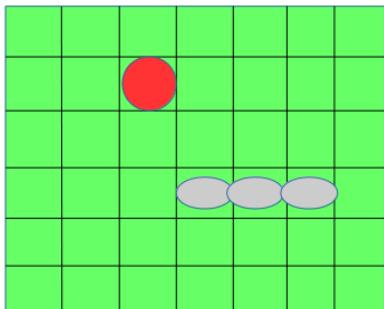
+100  
too

# Reinforcement Learning



## Multiple actions – Single state transition

moveLeft or moveRight etc., .



~~-20~~

-3	-2	-1	-2	-3	-4	-5
-2	-1	<span style="background-color: red; border-radius: 50%; width: 10px; height: 10px; display: inline-block;"></span>	-1	-2	-3	-4
-3	-2	-1	-2	-3	-4	-5
-4	-3	-2	-3	-4	-5	-6
-5	-4	-3	-4	-5	-6	-7
-6	-5	-4	-5	-6	-7	-8

### ① Q-table Initialize

Locotion1, 1 Top left	Location1, 2	Loc2,1`	Loc7,7 Bottom Right	Loc2,3 Apple	
No <del>-1000</del>	0	.....	0	0	moveLeft
Yes <del>0</del>	0	.....	-1000	0	moveRight
No <del>-1000</del>	-1000	.....	0	0	moveUp
Yes <del>0</del>	0	.....	0	0	moveDown

~~= -1000~~

Locotion1,1	Location1,2	Loc2,1`	Loc5,5	Loc2,3 Apple	Sample Value similar to adjacent setup
-30	-20	-10	-40	+100	Reward

# Reinforcement Learning

$$Q(L_{11}, R) \leftarrow (1-\alpha) \cdot Q(L_{11}, R)$$

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot \underbrace{r_t}_{\text{old value}} + \gamma \cdot \max_a Q(s_{t+1}, a)$$

Iteration 1:  $\alpha = 1$   $\gamma = 0.9$ . Trail 1:

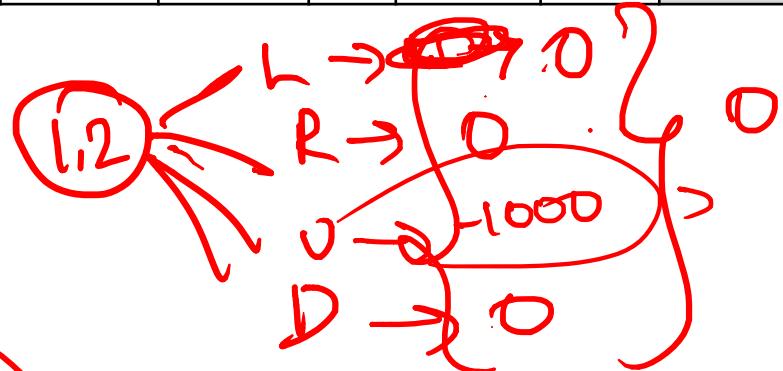
R → R → D

$$0 * 0 + 0 + 1 [-20 + 0.9 (\max \{ 0, -1000, 0 \})]$$

$$\begin{aligned} Q(L_{11}, R) &= (1 - \alpha) Q(L_{11}, R) + \alpha [R(T(L_{11}, R)) + \\ &\gamma (\text{Max}\{Q(L_{12}, L), Q(L_{12}, R), Q(L_{12}, U), Q(L_{12}, D)\})] \\ &= 0 + 1 * [-20 + 0.9 (\text{Max}\{0, -1000, 0\})] \\ &= -20 \end{aligned}$$

-20

Location1,1 Top left	Location1,2 Top right	Loc2,1 Bottom Left	Loc7,7 Bottom Right	Loc2,3 Apple	
-1000	0	.....	0	0	moveLeft
0	0	.....	-1000	0	moveRight
-1000	-1000	.....	0	0	moveUp
0	0	.....	0	0	moveDown



Action1,1 Top left	Location1,2 Top right	Loc2,1` Bottom Left	Loc7,7 Bottom Right	Loc2,3 Apple	
No transition possible	1,1	.....	7,6	2,2	moveLeft
1,2	1,3	.....	No transition possible	2,4	moveRight
No transition possible	No transition possible	.....	6,7	1,3	moveUp
2,1	2,2	.....	No transition possible	3,3	moveDown

Loction1,1	Location1,2	Loc2,1`	Loc5,5	Loc2,3	
-30	-20	-10	-40	+100	Reward



# Reinforcement Learning

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \underbrace{\alpha}_{\text{learning rate}} \left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \max_a Q(s_{t+1}, a) \right)$$

Iteration 1:  $\alpha = 1$   $\gamma = 0.9$ : Trail 1:



$Q(L1,2, R)$

$$= (1 - \alpha)Q(L1,2, R) + \alpha[R(T(L1,2, R)) + \gamma (\text{Max}\{Q(L1,3, L), Q(L1,3, R), Q(L1,3, U), Q(L1,3, D)\})]$$

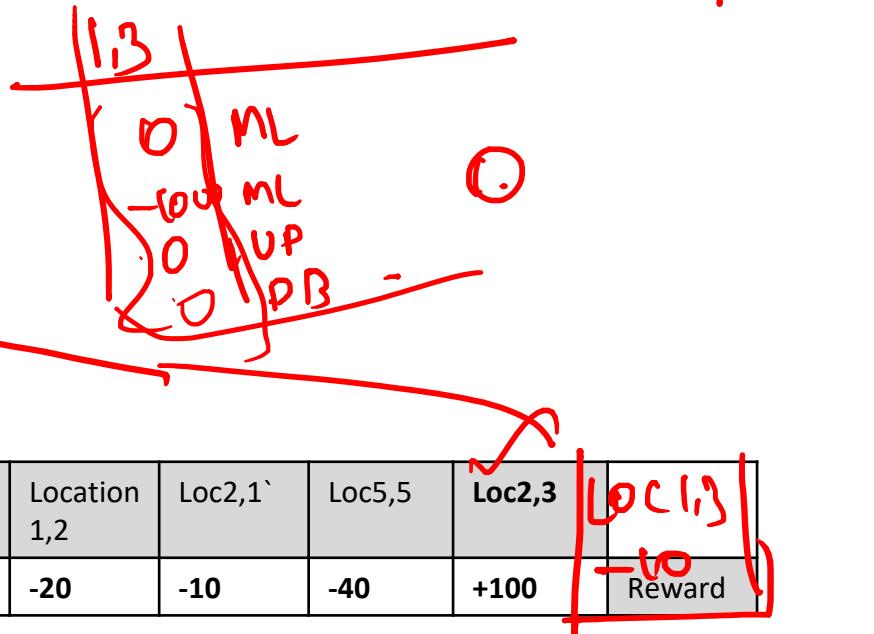
$$= 0 + 1 * [-10 + 0.9 (\text{Max}\{0, -1000, 0, 0\})] \\ = -10$$

$-10.$

~~$Q_{table}$~~

$$\begin{aligned} & 0 * 0 + 1 * (-10 + 0.9 \cdot \max Q(s_{t+1}, a)) \\ & \text{old value} \quad \text{learning rate} \quad \text{reward} \quad \text{discount factor} \quad \text{estimate of optimal future value} \end{aligned}$$

Loction1,1 Top left	Location 1,2	Loc 2,1	Loc7,7 Bottom Right	Loc2, 3 Apple	...
-1000	0	.....	0	0	moveLeft
<del>-20</del>	0	.....	-1000	0	<del>moveRight</del>
-1000	-1000	.....	0	0	moveUp
0	0	.....	0	0	moveDown



# Reinforcement Learning

$Q^{new}$

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{(r_t + \gamma \cdot \max_a Q(s_{t+1}, a))}_{\text{estimate of optimal future value}}$$

Iteration 3:  $\alpha = 1$   $\gamma = 0.9$ : Trail 1:

$R \rightarrow R \rightarrow D$

Loc : L3, D

$Q(L1,3, D)$

$$= (1 - \alpha)Q(L13, D) + \alpha [ R(T(L13, D)) + \gamma (\text{Max}\{Q(L23, L), Q(L23, R), Q(L23, U), Q(L23, D)\}) ]$$

$$= 0 + 1 \cdot [ +100 + 0.9 (\text{Max}\{0, -1000, 0, 0\}) ] \\ = +100$$

Loction1,1 Top left	Location1,2	Loc2,1`	Loc7,7 Bottom Right	Loc2,3 Apple	
-1000	0	.....	0	0	moveLeft
-20	-10	.....	-1000	0	moveRight
-1000	-1000	.....	0	0	moveUp
0	0	.....	0	0	moveDown

Loction1,1 Top left	Location1,2	Loc2,1`	Loc7,7 Bottom Right	Loc2,3 Apple	
No transition possible	1,1	.....	7,6	2,2	moveLeft
1,2	1,3	.....	No transition possible	2,4	moveRight
No transition possible	No transition possible	.....	6,7	1,3	moveUp
2,1	2,2	.....	No transition possible	3,3	moveDown

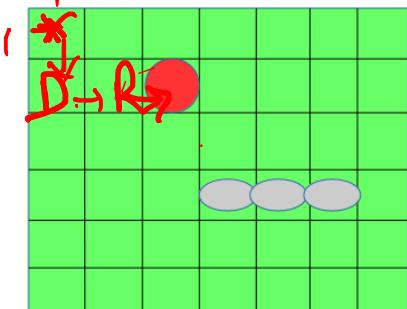
Loction1,1	Location1,2	Loc2,1`	Loc5,5	Loc2,3	
-30	-20	-10	-40	+100	Reward

# Reinforcement Learning

## Multiple actions – Single state transition

Start : Location 1,1 & find the path

~~Down → Right → Right.~~



Loc : 1,1  
D

~~Q-table~~ Comp to

Location1 ,1 Top left	Location 1,2	Loc2, 1`	L2,2	Loc2, 3 Apple	
-1000	20	-1000	-50	0	moveLeft
55	-40	40	+200	0	moveRight
-1000	-1000	-30	-30	0	moveUp
60	30	-30	-60	0	moveDown

Loc : 2,1  
R

D → R → . R

Loc : 2,2  
R

-3	-2	-1	-2	-3	-4	-5
-2	-1	1	-1	-2	-3	-4
-3	-2	-1	-2	-3	-4	-5
-4	-3	-2	-3	-4	-5	-6
-5	-4	-3	-4	-5	-6	-7
-6	-5	-4	-5	-6	-7	-8

# Reinforcement Learning

Idea: Multiple actions : Multiple state transitions

1 $S_1$	2 $S_L$	L (depicted as 1) Transition Prob
0.7	0.4	1 $S_1$
0.3	0.6	2 $S_2$

$S_1$	$S_2$	L (depicted as 1) Reward
6	7	1 $R_1$
-5	12	2 $R_2$

Left  $\rightarrow$  1  
Right  $\rightarrow$  2

action

(1,0.7,6)

(1,0.3,-5)

(2,0.1,17)

$S_1$

$S_2$

(2,0.9,10)

(2,0.2,-14)

R

(2,0.8,13)

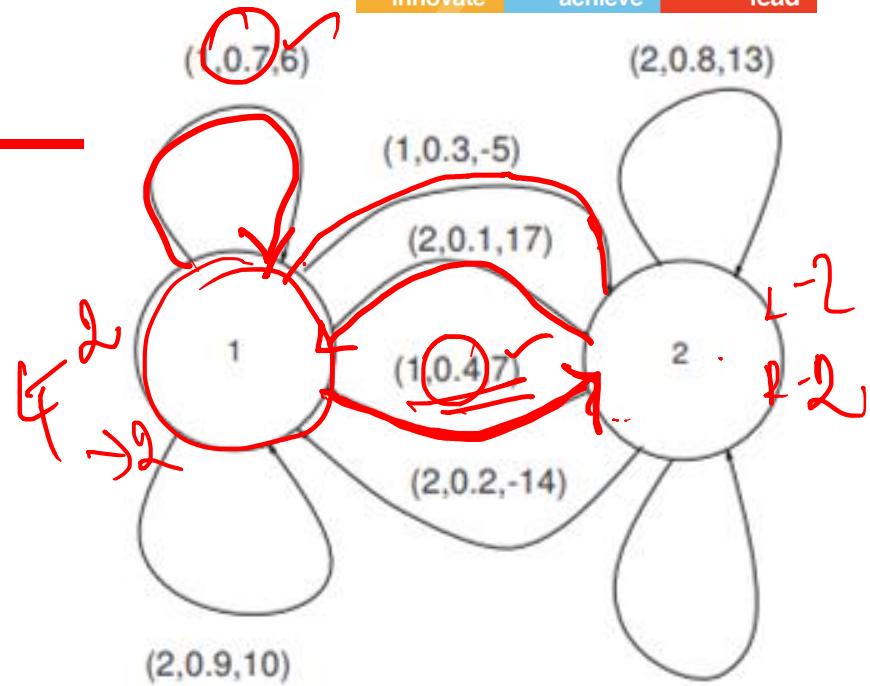
(1,0.6,12)

Legend:

(a,p,r); a = action  
p = transition probability  
r = immediate reward

$a_1$

# Reinforcement Learning



1	2	L (depicted as 1) Transition Prob
0.7	0.4	1
0.3	0.6	2

1	2	L (depicted as 1) Reward
6	7	1
-5	12	2

1	2	R (depicted as 2) Transition Prob
0.9	0.2	1
0.1	0.8	2

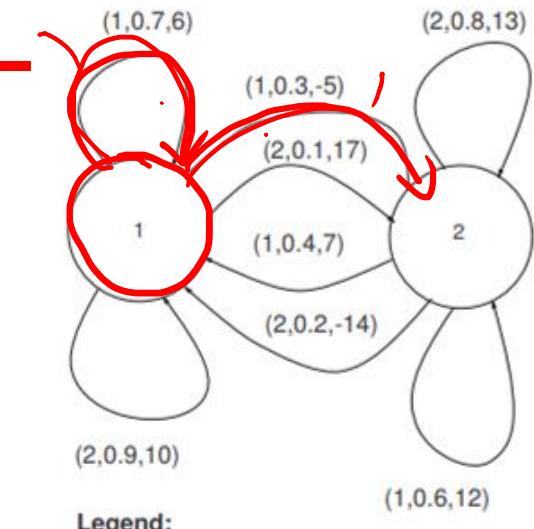
1	2	R (depicted as 2) Reward
10	-14	1
17	13	2

d:  
 a = action  
 p = transition probability  
 r = immediate reward

# Reinforcement Learning

1	2	L (depicted as 1) Transition Prob
0.7	0.4	1
0.3	0.6	2

1	2	L (depicted as 1) Reward
6	7	1
-5	12	2



Legend:  
 (a,p,r): a = action  
 p = transition probability  
 r = immediate reward

$$Q_{n+1}(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \times \max_{a'} Q(s', a')]$$

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\substack{\text{learned value} \\ \text{estimate of optimal future value}}} \right)$$

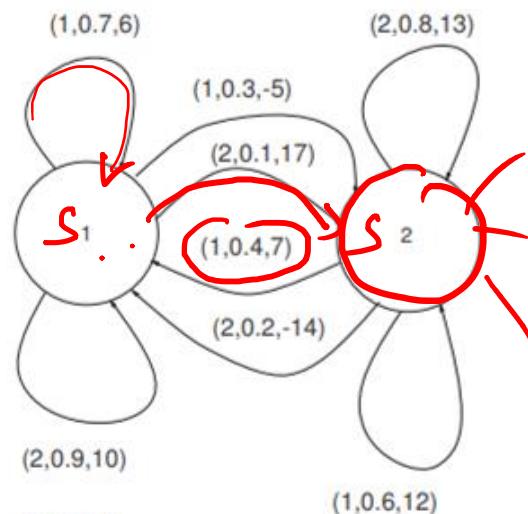
# Reinforcement Learning

Iteration 1:  $\alpha = 0.6 \gamma = 0.9$ : Trail 1:

Start State : State1

1	2	L (depicted as 1) Transition Prob
0.7	0.4	1
0.3	0.6	2

1	2	L (depicted as 1) Reward
6	7	1
-5	12	2



Legend:  
 $(a,p,r)$ : a = action  
 p = transition probability  
 r = immediate reward

$$\begin{aligned}
 Q(\text{State1}, \text{Left}) &= (1 - \alpha)Q(\text{State1}, \text{Left}) + \alpha \left( \right. \\
 &\quad \cancel{\sum T(\text{State1}, \text{Left}, \text{State1}) * [R(\text{State1}, \text{Left}, \text{State1}) + \gamma (\max\{Q(\text{State1}, \text{Left}), Q(\text{State1}, \text{Right})\})]} \\
 &\quad \cancel{+ T(\text{State1}, \text{Left}, \text{State2}) * [R(\text{State1}, \text{Left}, \text{State2}) + \gamma (\max\{Q(\text{State2}, \text{Left}), Q(\text{State2}, \text{Right})\})]} \\
 &\quad \left. \right) \\
 &= (0.4)Q(\text{State1}, \text{Left}) + 0.6 \left( \right. \\
 &\quad 0.7 * [6 + 0.9 (\max\{Q(\text{State1}, \text{Left}), Q(\text{State1}, \text{Right})\})] \\
 &\quad + \\
 &\quad 0.3 * [-5 + 0.9 (\max\{Q(\text{State2}, \text{Left}), Q(\text{State2}, \text{Right})\})] \\
 &\quad \left. \right)
 \end{aligned}$$

$$Q_{n+1}(s, a) = \sum T(s, a, s') [R(s, a, s') + \gamma \times \max_{a'} Q(s', a')]$$

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\left( \underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}_{\text{learned value}}$$

# Reinforcement Learning Case Study



## Discuss

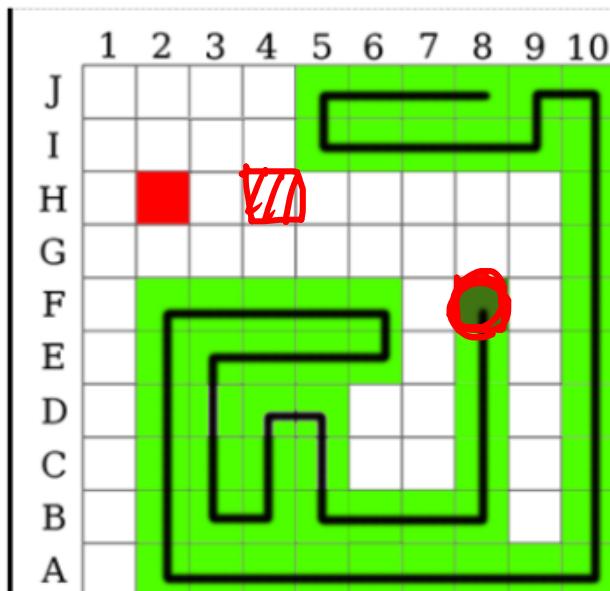
---

Source Credit :

<https://www.capecstart.com/resources/blog/reinforcement-learning-in-health-care-why-its-important-and-how-it-can-help/>

# Reinforcement Learning –Q learning Algorithm -DSE

Consider the below problem and to apply the reinforcement learning with initial Q-Table initialized to value = “-20” learning rate=0.8 and discount factor=0.5. It’s mandatory to show the update of Q-table at the end of every iteration.



In this snake game (head of the snake is represented by dark green cell and body is depicted by green shaded ones), the snake has to learn to eat as many apples (represented in red colored cell) as possible without dying. Eating apple makes it grow lengthier. Both its head hitting the wall or its own body is penalized by death leading to game restart.

For each action it gains a reward of -5 and eating apple adds a reward of +50. Assume that the snake start at the location H4 and performs two actions moveLeft → moveLeft, answer the below questions.

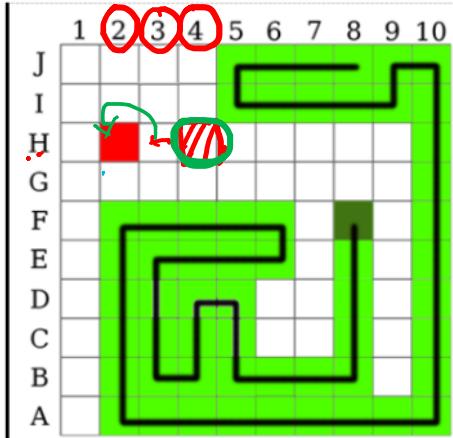
I. Q table  
Reward

ML - ML

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{(1 - \alpha) \cdot Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{\left( r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \right)}_{\substack{\text{learned value} \\ \text{reward} \\ \text{discount factor} \\ \text{estimate of optimal future value}}}.$$

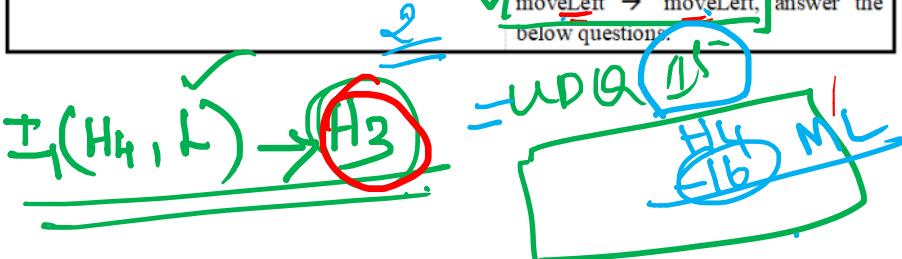
# Reinforcement Learning –Q learning Algorithm -DSE

Consider the below problem and to apply the reinforcement learning with initial Q-Table initialized to value = “-20”, learning rate=0.8 and discount factor=0.5. It’s mandatory to show the update of Q-table at the end of every iteration.



In this snake game (head of the snake is represented by dark green cell and body is depicted by green shaded ones), the snake has to learn to eat as many apples (represented in red colored cell) as possible without dying. Eating apple makes it grow lengthier. Both its head hitting the wall or its own body is penalized by death leading to game restart.

For each action it gains a reward of -5 and eating apple adds a reward of +50. Assume that the snake starts at the location H4 and performs two actions moveLeft → moveLeft, answer the below question.



$$Q^{\text{new}}(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{(r_t + \gamma \cdot \max Q(s_{t+1}, a))}_{\text{learned value}}$$

$\Rightarrow -5 + 0.8(-5 + 0.5 \cdot 50) \rightarrow H_2$

$\alpha = 0.8, \gamma = 0.5$

Q table

$A_1$	$B_1$	$H_2$	$H_3$	$H_4$	$J_5$	$M_6$	$M_7$
-20	-20	-20	-20	-20	-20	-20	-20
-20	-20	-20	-20	-20	-20	-20	-20
-20	-20	-20	-20	-20	-20	-20	-20
-20	-20	-20	-20	-20	-20	-20	-20
-20	-20	-20	-20	-20	-20	-20	-20
-20	-20	-20	-20	-20	-20	-20	-20
-20	-20	-20	-20	-20	-20	-20	-20

$\rightarrow$  Eat  $\rightarrow +50$

$$Q(H_4, L) \leftarrow (1 - \alpha) Q(H_4, L) + \alpha (r_t + \gamma * \max(Q(H_3, L)))$$

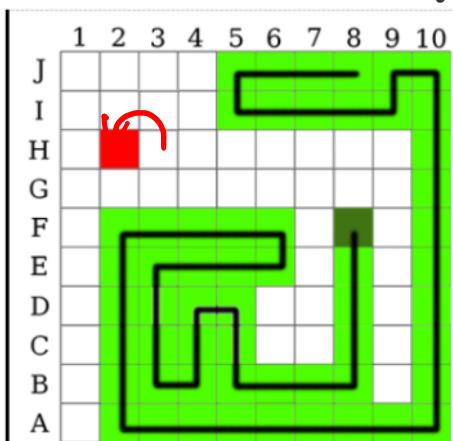
$$\leftarrow (1 - 0.8)(-20) + 0.8(-5) + (0.5) * \max(-20, -20, \bar{Q}(H_3, R))$$

$$\leftarrow 0.2 * -20 + 0.8(-5) + 0.5 * -20$$

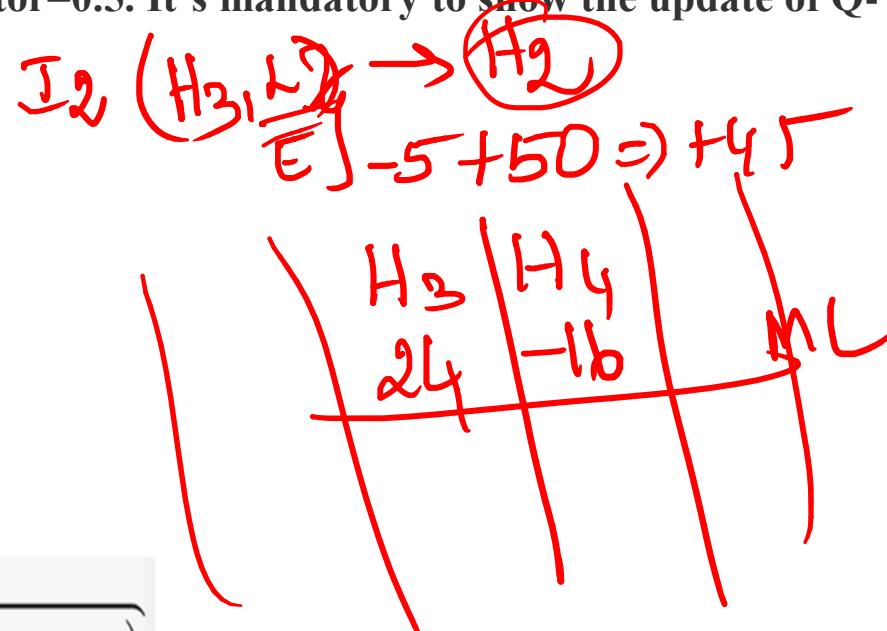
$$\leftarrow -4 + 0.8(-5 - 10) \Rightarrow -4 + 0.8(-15) \Rightarrow -12 \Rightarrow -16$$

# Reinforcement Learning –Q learning Algorithm -DSE

Consider the below problem and to apply the reinforcement learning with initial Q-Table initialized to value = “-20”, learning rate=0.8 and discount factor=0.5. It’s mandatory to show the update of Q-table at the end of every iteration.



In this snake game (head of the snake is represented by dark green cell and body is depicted by green shaded ones), the snake has to learn to eat as many apples (represented in red colored cell) as possible without dying. Eating apple makes it grow lengthier. Both its head hitting the wall or its own body is penalized by death leading to game restart. For each action it gains a reward of -5 and eating apple adds a reward of +50. Assume that the snake starts at the location H4 and performs two actions moveLeft → moveLeft, answer the



$Q(A_3, L)$

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \underbrace{(r_t + \gamma \cdot \max_a Q(s_{t+1}, a))}_{\text{learned value}}$$

$\leftarrow (1 - 0.8) * (-20) + (0.8) * (45 + 0.5 * \max(H_2, L, H_2, R, H_2, U, H_2, D, H_2, E))$

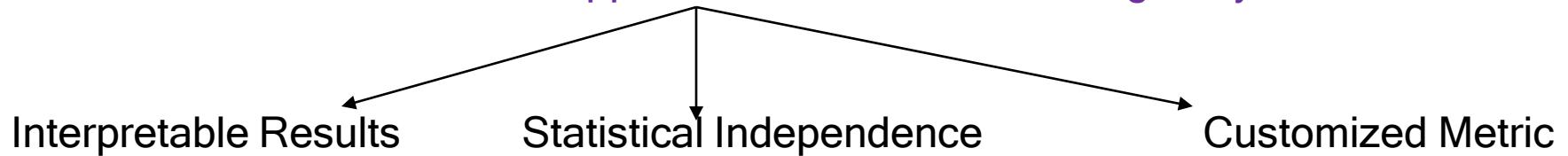
 $= (0.2) * (-20) + (0.8) * (45 + -10)$ 
 $= -4 + 0.8 * 35$ 
 $\Rightarrow -4 + 28 \Rightarrow 24$



# Ethics in Artificial Intelligence

# Building a Fair Model

1. Is it fair to make an AI-ML system?
2. Is there a better technical approach to convert an existing AI system fair?



# Interpretable Models

Are the results obtained by the AI system fair?

Interpretable models helps to trust the AI system by answering transparently to the specific questions like “Why the system is behaving under certain scenarios?”

- If a loan gets rejected, do we know the reasons?
- If a job application is accepted, is it biased towards a gender?
- If a bail is granted to an accused, is it based on their race?
- If a patient is diagnosed with a disease, what factors made the algorithm to classify it?

# Interpretable Models

Example Based Explanations:

If SymptomInX  $\equiv$  SymptomInY

if DiseaseA infected X

then probably DiseaseB might have infected Y

If CustomerX  $\equiv$  CustomerY

if CustomerX purchased P1

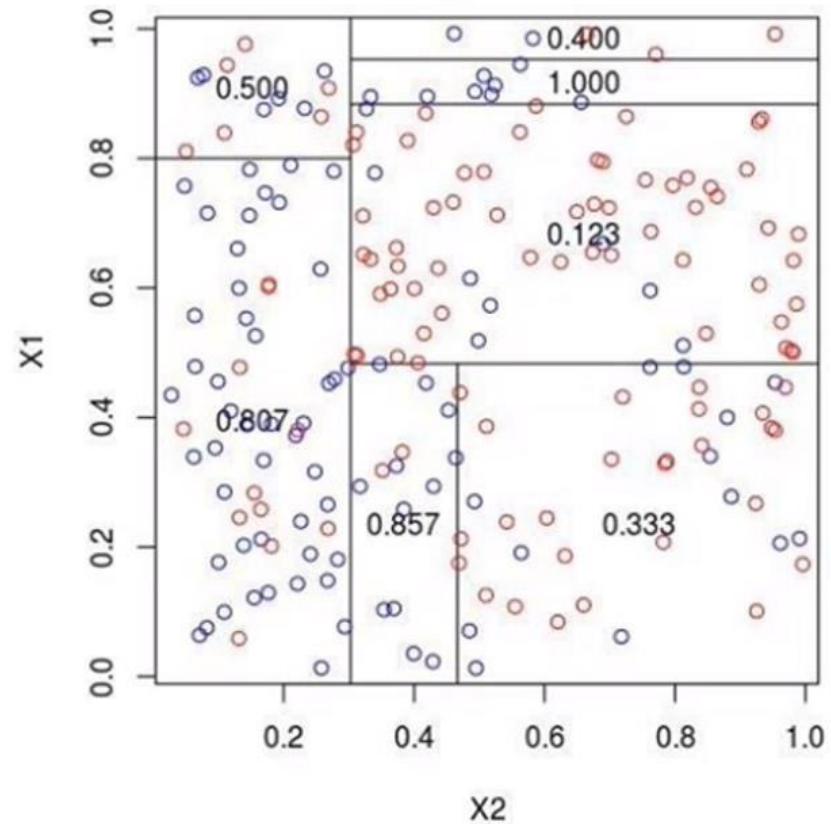
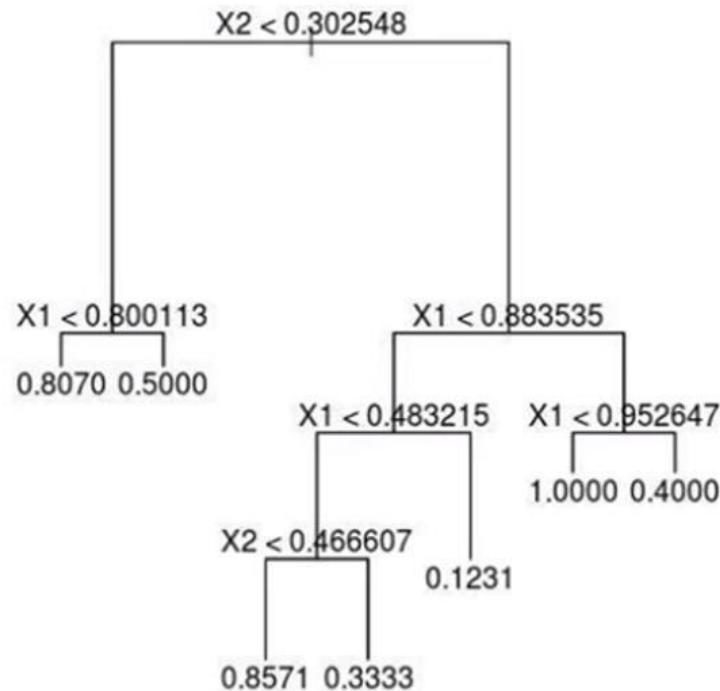
then probably CustomerY will purchase P1

Counterfactual Explanations:

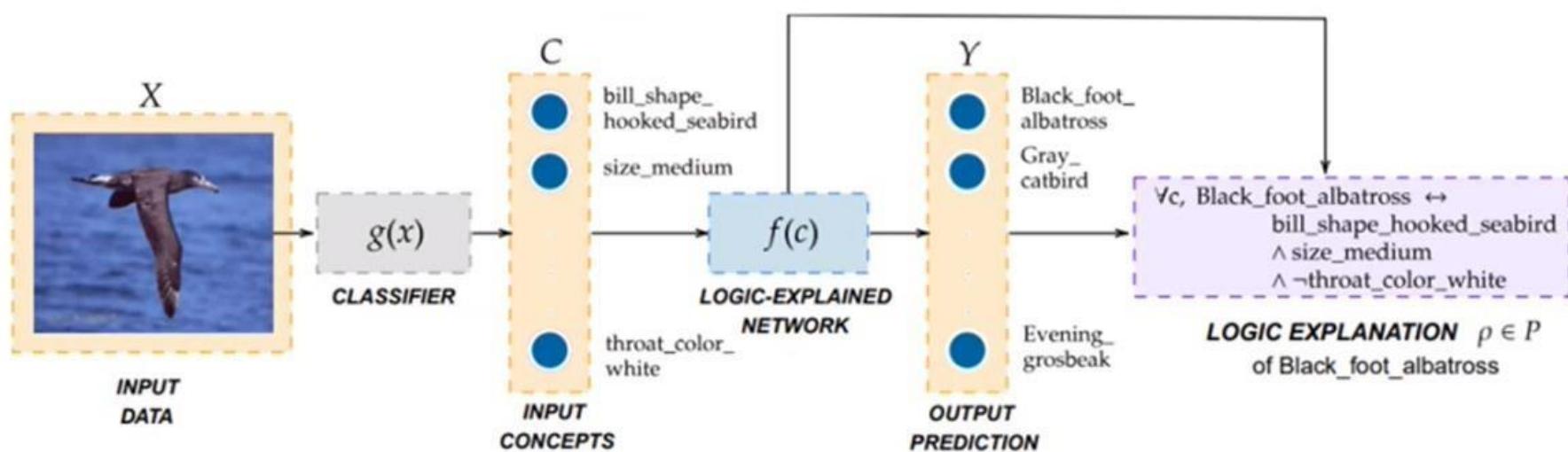
If customerX's income level had not been less than L3

then the customer's Loan might not have been rejected

# Interpretable Models



## In Deep Learning



---

**Required Reading:** AIMA - Chapter #15.1, #15.2, #15.3

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials



# Artificial & Computational Intelligence

**DSECLZG557**

## Sample Problems

**BITS** Pilani  
Pilani Campus

Indumathi V  
Guest Faculty  
BITS -WILP

# Logics

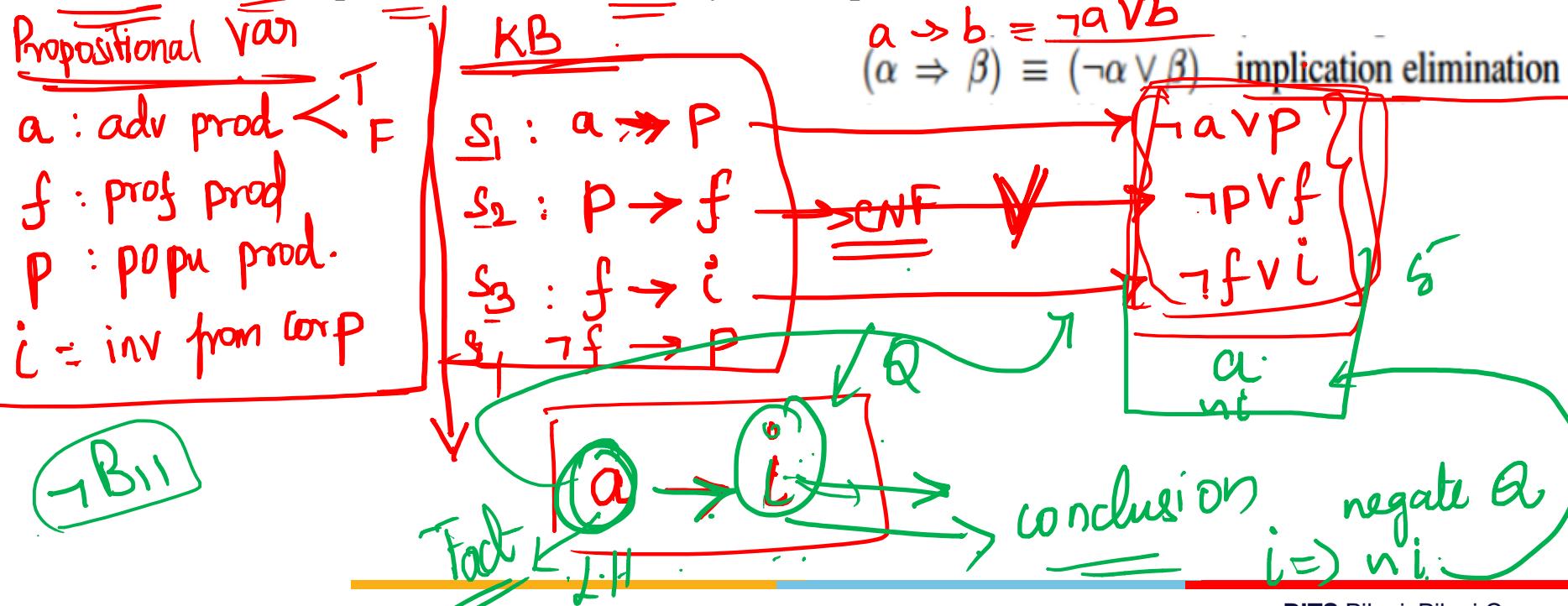
"In the marketing industry, all advertised products gain popularity. Not all profitable products have been popular, but all the popular products have been always profitable. Profitable products attract investments from corporates."

$$T \xrightarrow{a} P$$

P fn (tuple)  
Predicate  
 $\forall \exists$

- ✓ 1. Represent the knowledge base using propositional logic (without quantifiers)
- ✓ 2. convert KB into CNF and find any three sample complete BSAT (Binary Satisfiability) solutions to the variables using DPLL algorithm.
- ✓ 3. Using the result of part a), prove the below using equivalence laws and resolution.

"All the advertised products are invested by the corporates"



# Logics

$S_1: \neg a \vee p$   
 $S_2: \neg p \vee f$   
 $S_3: \neg f \vee i$

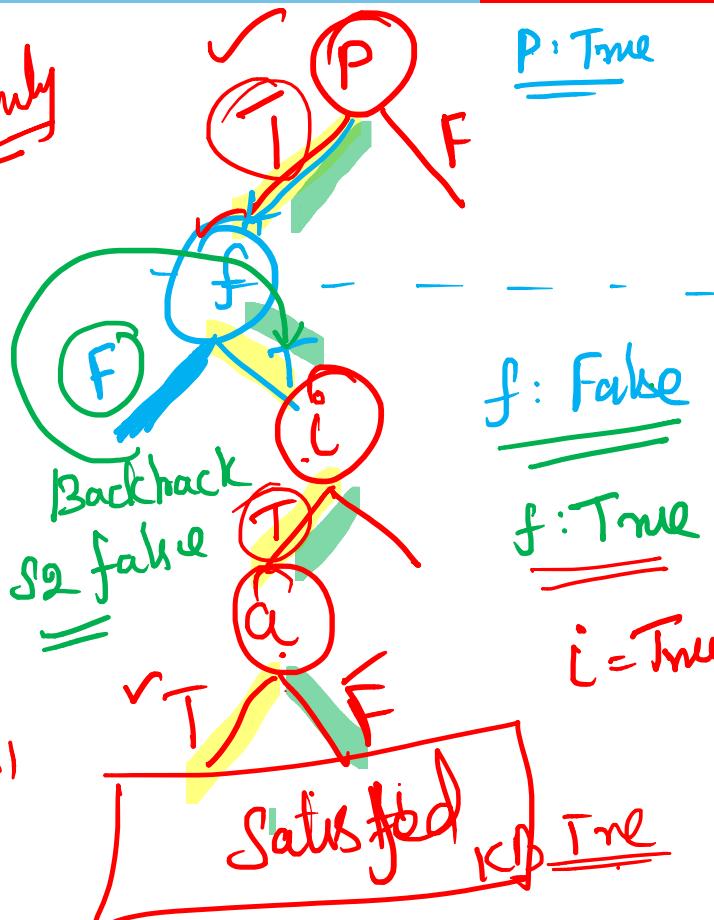
Pafi → randomly

f

fa

Pfia  
 $\neg a$     $\neg f$     $i$     $a$   
 $\neg T$     $\neg T$     $T$     $T$

Path1  
 $\neg T$     $\neg T$     $F$   
Soln Path2



KB  
 $S_1: \neg a \vee p \checkmark \rightarrow \text{Time}$   
 $S_2: \neg p \vee f \cancel{\checkmark} \rightarrow \text{Time}$   
 $S_3: \neg f \vee i \cancel{\checkmark} \rightarrow \text{Time}$

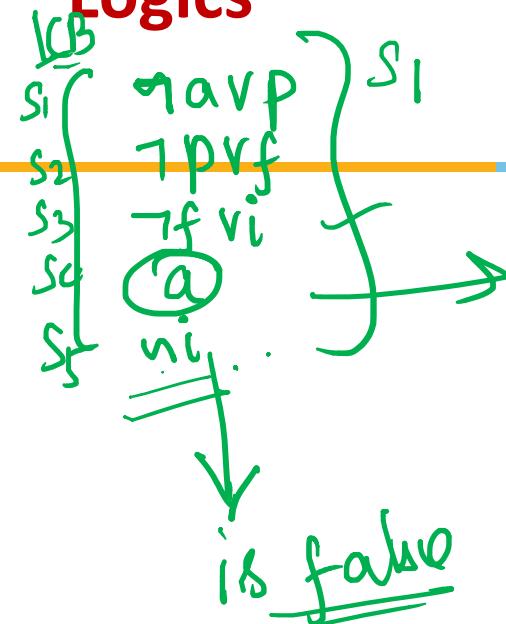
KB False

$S_1: \neg a \vee p \checkmark \rightarrow \text{Time}$   
 $S_2: \neg p \vee f \cancel{\checkmark} \rightarrow \text{False}$   
 $S_3: \neg f \vee i \cancel{\checkmark}$

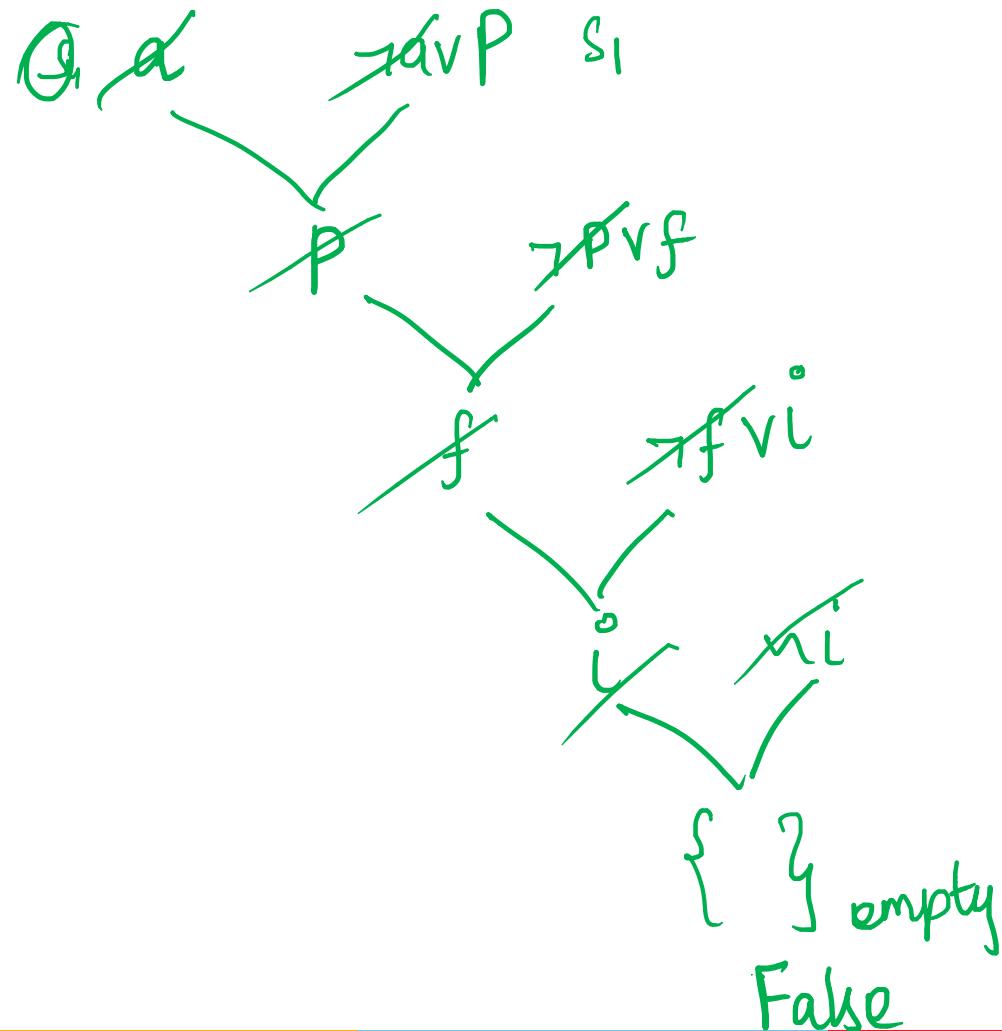
$\neg a \vee p \checkmark \rightarrow \text{Time}$   
 $\neg p \vee f \cancel{\checkmark} \rightarrow \text{Time}$   
 $\neg f \vee i \cancel{\checkmark} \rightarrow \text{Time}$

KB → Time

## Logics



$a \rightarrow i$  prof



# Logics $\rightarrow$ predicate logic

pred fn



(2) H/W

function

"A software intern is always trained in a project as a project member. Every Project member is involved in development team and support team. Every development member is involved in unit testing. Every support member is involved in user acceptance testing. A project member trained in both unit testing and acceptance testing is certified as skilled in testing."

Convert the above into predicate logic.

Prove by backward chaining that "Prove that all the software interns will be certified as skilled in testing." using the results of part a. Show the steps by step inferences using neat diagram with direction.

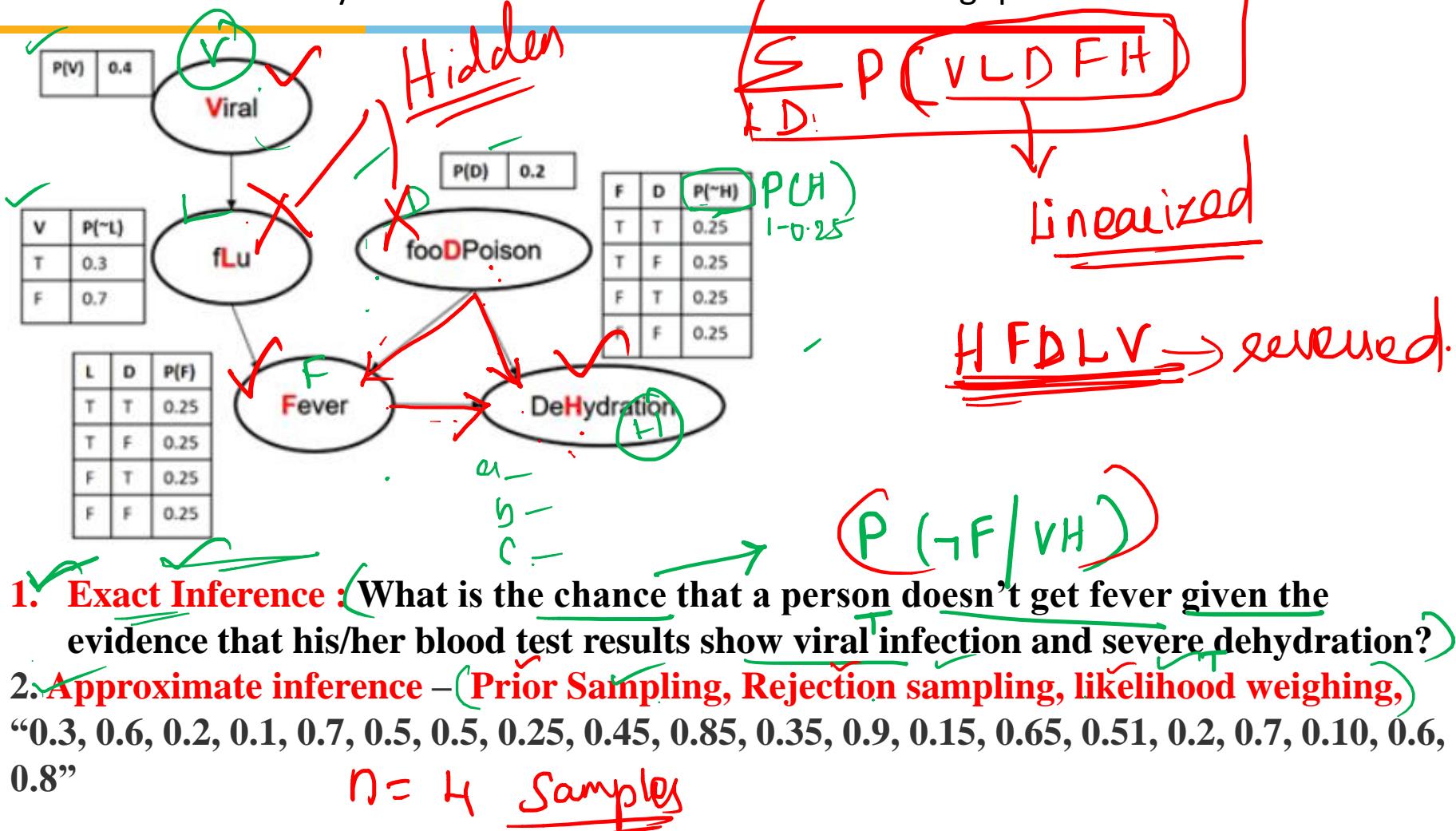
$R_1$   
 $R_2$   
 $R_3$   
 $R_4$

$f_1$   
 $f_2$   
 $f_3$

$O(x) \rightarrow I(x)$

# Bayesian Network

Consider the below Bayesian Network and answer the following questions:



- Exact Inference :** What is the chance that a person doesn't get fever given the evidence that his/her blood test results show viral infection and severe dehydration?
- Approximate inference –** (Prior Sampling, Rejection sampling, likelihood weighing, "0.3, 0.6, 0.2, 0.1, 0.7, 0.5, 0.5, 0.25, 0.45, 0.85, 0.35, 0.9, 0.15, 0.65, 0.51, 0.2, 0.7, 0.10, 0.6, 0.8")

$n = 4$  Samples

# Bayesian Network

$$\underline{P(e) + P(\bar{e}) = 1}$$

Q Bayesian Network :-

1)  $\underline{P(nF|VH)} \rightarrow \text{Exact inference}$

$$\Rightarrow P(nF|VH) + P(F|VH) = 1$$

apply Bayes

$$\text{Rule: } \frac{P(nF|VH)}{P(VH)} + \frac{P(F|VH)}{P(VH)} = 1 \rightarrow$$

$$P(A|B) = \frac{P(AB)}{P(B)}$$

$$\frac{1}{P(VH)} = \left( \frac{1}{P(nF|VH) + P(F|VH)} \right) \rightarrow ①$$

Query:  $P(nF|VH) \Rightarrow \frac{P(nF|VH)}{P(VH)} \quad (\text{Bayes rule})$

$$\Rightarrow \frac{P(nF|VH)}{P(nF|VH) + P(F|VH)}$$

Substitute the value.

L, D are hidden Variable.

# Bayesian Network

*H then F  
X.  
see diagram*

$$\Rightarrow P(\text{HFVH}) \Rightarrow \sum_{LD} P(\underline{H} \underline{\text{F}} \underline{\text{FDL}} \underline{V})$$

*apply chain rule:*

$$P(H | nFD) * P(nF | LD) * P(L | V) * P(V) * P(D)$$
$$P(H | nFD) * P(nF | LD) * P(L | V) * P(V) * P(D) +$$
$$P(H | nFD) * P(nF | LD) * P(nL | V) * P(V) * P(nD)$$
$$0.0315 \Rightarrow P(H | nFD) * P(nF | LD) * P(L | V) * P(V) * P(D) +$$
$$0.0135 \Rightarrow P(H | nFD) * P(nF | LD) * P(nL | V) * P(V) * P(D) +$$
$$0.1260 \Rightarrow P(H | nFD) * P(nF | LD) * P(L | V) * P(V) * P(nD) +$$
$$0.0540 \Rightarrow P(H | nFD) * P(nF | LD) * P(nL | V) * P(V) * P(nD) +$$

*0.2250*

# Bayesian Network

$P(\text{uFVH}) \Rightarrow 0.2250$

Use same approach to find  $\underline{\underline{P(FVH)}}$

$P(FVH) = 0.075$

Final answer

$P(\text{uF|VH}) = \frac{0.2250}{0.2250 + 0.075}$

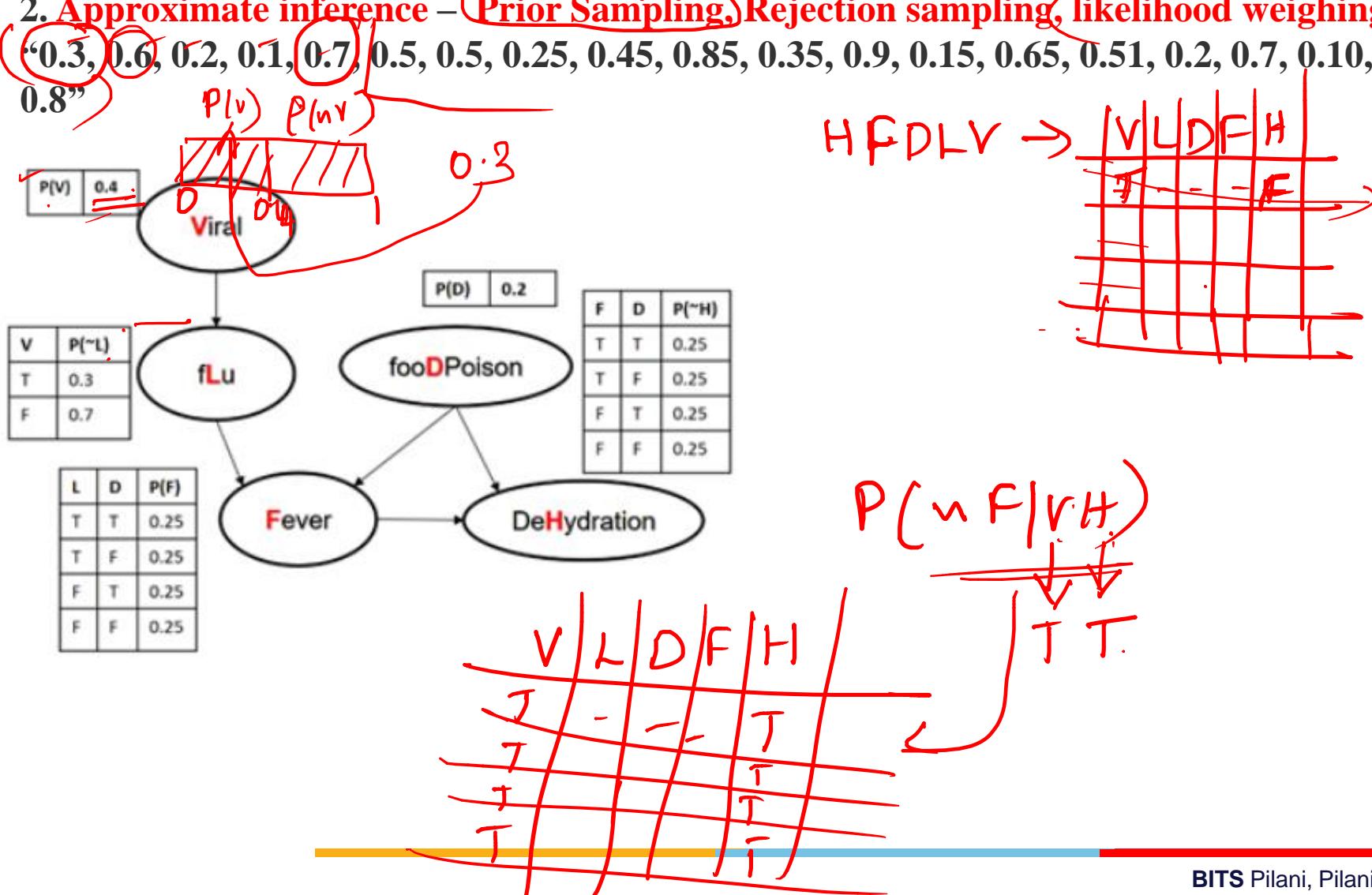
$P(\text{uF|VH}) = \frac{0.2250}{0.3}$

$\boxed{P(\text{uF|VH}) = \underline{\underline{0.75}}}$

# Bayesian Network

1. **Exact Inference :** What is the chance that a person doesn't get fever given the evidence that his/her blood test results show viral infection and severe dehydration?

2. **Approximate inference – Prior Sampling, Rejection sampling, likelihood weighing,**  
 “0.3, 0.6, 0.2, 0.1, 0.7, 0.5, 0.5, 0.25, 0.45, 0.85, 0.35, 0.9, 0.15, 0.65, 0.51, 0.2, 0.7, 0.10, 0.6, 0.8”

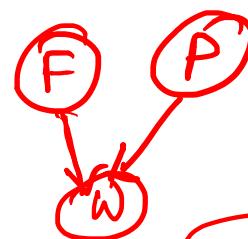


# Bayesian Network

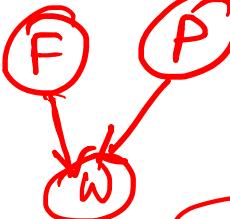
innovate

achieve

lead



F



BN|W

Most of the WILP students are fans (F) of cricket irrespective of their gender. With the new season of IPL (Indian Premier League) having started on the exam month almost every cricket fans spend time to watch(W) the live play. Sometimes being a parent (P) reduces the probability of watching the IPL live season. A likely consequence of watching matches is reduced concentration(C) on the following days. A consequence of the reduced concentration is increased stress(S) with work environment leading to reduced productivity(D) in project. Lack of concentration might also be caused by viral (V) infection which is common in this rainy season(R). WILP students have the comprehensive exams and reduced concentration would reduce the probability of good grades(G) in the exam which reflects the performance of students in examination. Assume an AI agent is fed this information and it answers to certain queries that can be inferred. Assume all the events (conditional or unconditional) are equally likely to occur:

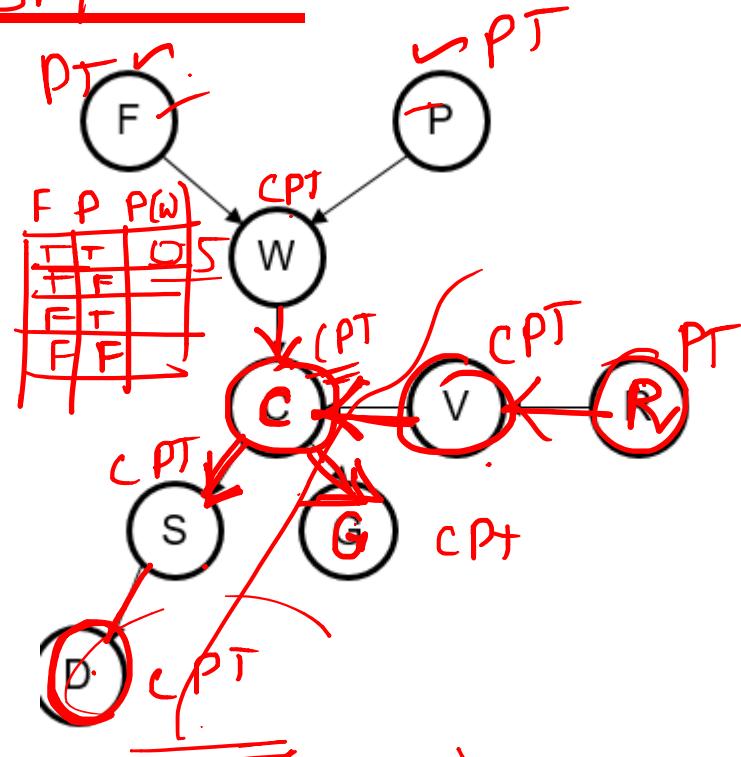
Example Joint Prob. Distribution Query :

What is the chance that "an ardent fan of cricket who is a parent of two kids, never misses an IPL match, doesn't get stressed in work environment, is affected by viral infection and performs well in the comprehensive examination"?

$$P(F) = 0.5 \quad P(\neg F) = 0.5$$

$$P(F, P, W, \neg S, V, G)$$

D-Separation: Performance of in the examination is independent of stress in work environment given its known that the student is affected by viral infection



# Bayesian Network

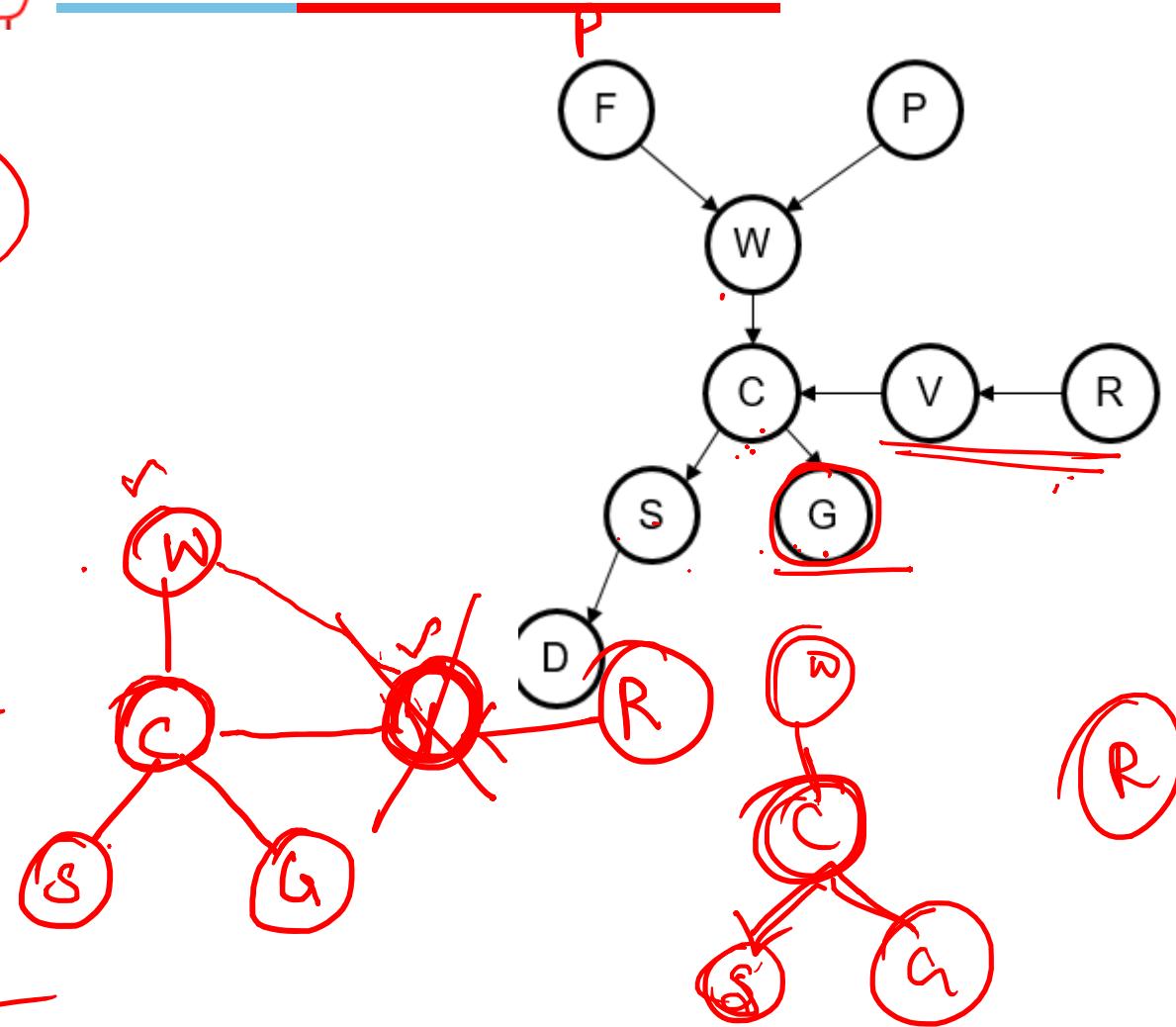
~~t, is affected by viral infection a~~

$$P(F, P, W, nS, V, G)$$

$(G \cap S \vee WFP)$

$P(G | nS \vee WFP)$

~~$G \perp S | V$~~   $\text{remove } G \perp S | V$   $\text{False}$



Initially - no. of ants =  
no. of cities

Start at 4.

Given,  $\alpha = 100$

$\alpha = 0.5$

$p = 0.75$

$Q = 0.1$

Take random values for  $T_{ij}$  initially

$$T_{12} = T_{21} = 0.54$$

$$T_{13} = T_{31} = 0.53$$

$$T_{14} = T_{41} = 0.35$$

$$T_{15} = T_{51} = 0.24$$

$$T_{23} = T_{32} = 0.53$$

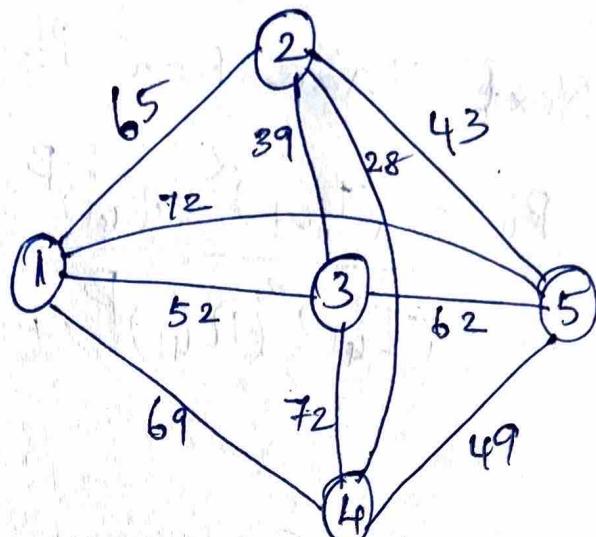
$$T_{24} = T_{42} = 0.39$$

$$T_{25} = T_{52} = 0.18$$

$$T_{34} = T_{43} = 0.32$$

$$T_{35} = T_{53} = 0.90$$

$$T_{45} = T_{54} = 0.68$$



## Next Transition Probability

$$P_{41} = \frac{(T_{41})^{\alpha} (n_{41})^{\beta}}{(T_{41})^{\alpha} (n_{41})^{\beta} + (T_{42})^{\alpha} (n_{42})^{\beta} + (T_{43})^{\alpha} (n_{43})^{\beta} + (T_{45})^{\alpha} (n_{45})^{\beta}}$$

$$= \frac{(0.35)^{0.5} (0.014)^{0.75}}{(0.35)^{0.5} (0.014)^{0.75} + (0.398)^{0.5} (0.036)^{0.75} + (0.326)^{0.5} (0.014)^{0.75} + (0.683)^{0.5} (0.02)^{0.75}}$$

$$= \frac{0.024}{0.143} = 0.168$$

$$\eta_{ij} = Y_{dij}$$

$$n_{41} = Y_{69} = 0.014$$

$$n_{42} = Y_{28} = 0.036$$

$$n_{43} = Y_{72} = 0.044$$

$$n_{45} = Y_{49} = 0.021$$

$$P_{42} = \frac{0.053}{0.142} = 0.364 \text{ Max.}$$

$$P_{43} = \frac{0.023}{0.143} = 0.160$$

$$P_{45} = 0.308$$

Since  $P_{42}$  is max, move from 4 to 2.

current ~~list~~ visit list (4, 2)

## Pheromone Updation ( $t=1$ )

$$\tau_{12} = \rho(\tau_{12}) + 4\tau_{12} \quad \text{excepting for } (4,2) \text{ & } (2,4)$$

$$= 0.054$$

$$\tau_{13} = 0.053, \quad \tau_{14} = 0.035,$$

$$4\tau_{ij} \text{ is zero for all other } (ij)$$

$$\tau_{15} = 0.024, \quad \tau_{23} = 0.05,$$

$$\tau_{24} = 0.039 + \theta/d_{ij} = 0.039 + 100/28 = 3.610$$

$$\tau_{25} = 0.018, \quad \tau_{34} = 0.032, \quad \tau_{35} = 0.09$$

$$\tau_{45} = 0.068.$$

Now ant is at 2.

$$P_{21} = 0.320$$

$$P_{23} = 0.429 \quad \text{Max}$$

$$P_{25} = 0.250$$

Now ant moves to 3!

## Pheromone Updation ( $t=2$ )

$$\tau_{12} = 0.005, \quad \tau_{13} = 0.005, \quad \tau_{14} = 0.003,$$

$$\tau_{15} = 0.002, \quad \tau_{23} = 0.005 + 100/39 = 2.569$$

$$\tau_{24} = \rho(3.616) = 0.361, \quad \tau_{25} = 0.001$$

$$\tau_{34} = 0.003, \quad \tau_{35} = 0.009, \quad \tau_{45} = 0.006$$

From 3, find  $P_{31}, P_{35}$

$$P_{31} = 0.500$$

$$P_{35} = \textcircled{0.5571} - \text{Max}$$

From 3, ant moves to 5.

Tabu list

4, 2, 3, 5

Pheromone updation  $t=3$

$$\tau_{12} = 0.0005, \tau_{13} = 0.0005, \tau_{14} = 0.0003,$$

$$\tau_{15} = 0.0002, \tau_{23} = 0.25, \tau_{24} = 0.03,$$

$$\tau_{25} = 0.0001, \tau_{34} = 0.0003, \tau_{35} = 0.0009 + 100/62$$

$$\tau_{45} = 0.0006$$

From 5, move to 1 since it is the only non-visited city. Update pheromone.

$$\tau_{12} = 0.0005, \tau_{13} = 0.00005, \tau_{14} = 0.00003,$$

$$\cancel{\tau_{15} = 0.00002}, \cancel{\tau_{23} = 0} \quad \tau_{15} = 1.388, \tau_{23} = 0.025$$

$$\tau_{24} = 0.003, \tau_{25} = 0.00001, \tau_{34} = 0.00003$$

$$\tau_{35} = 0.16, \tau_{45} = 0.0006$$

Now back to origin, since all the states are visited.

$$\text{update } T_{14} = T_{41} = 0.00003 \times 100 / 69 \\ = 1.449$$

Final route,

$$4 - 2 - 3 - 5 - 1 - 4$$



# Artificial & Computational Intelligence



## A\* Algorithm

Rashmi K

**BITS** Pilani  
Pilani Campus



# Informed Search

## A\*

## A\* Search

Expands the node which lies in the closest path (estimated cheapest path) to the goal

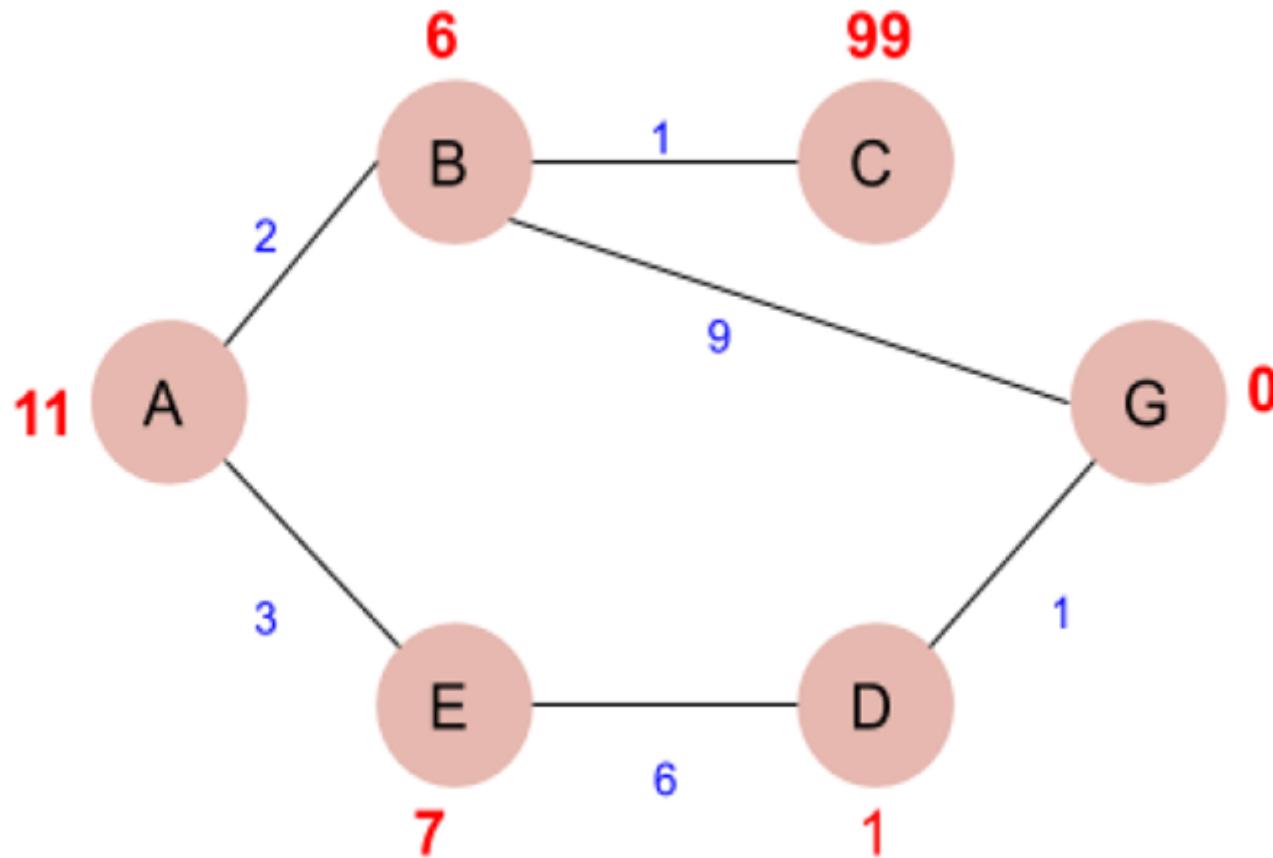
Evaluation function  $f(n) = g(n) + h(n)$

$g(n)$  – the cost to reach the node / path cost

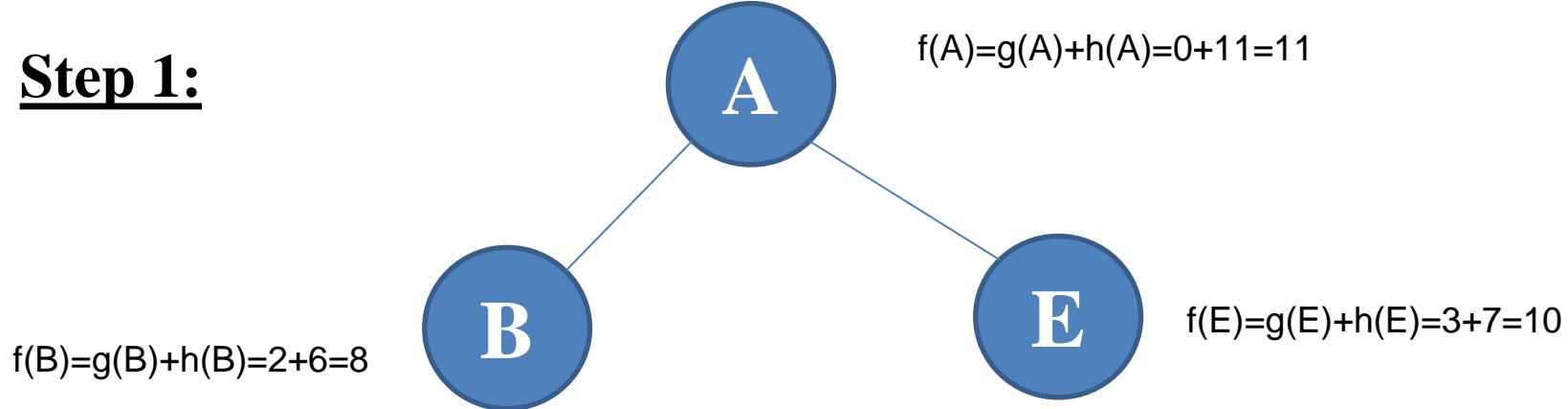
$h(n)$  – the expected cost to go from node to goal

$f(n)$  – estimated cost of cheapest path through node n

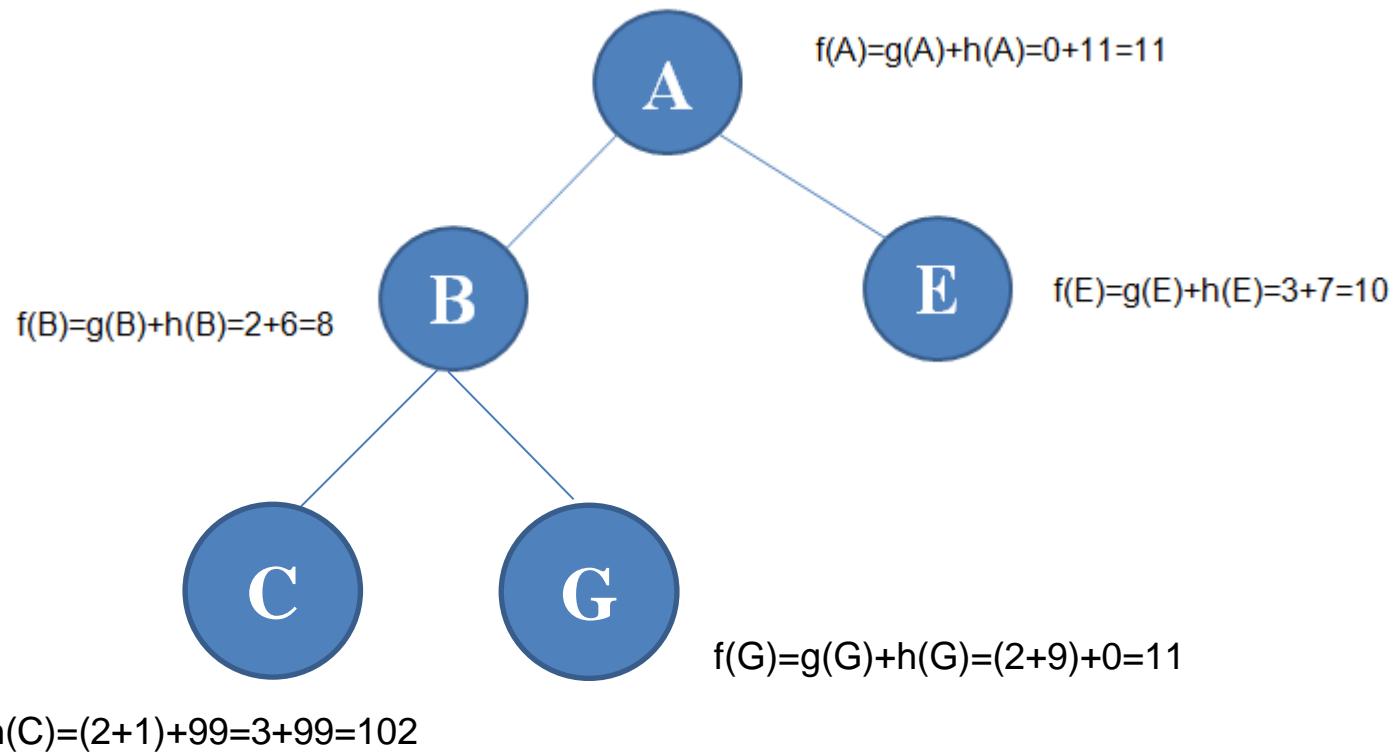
# Example 1



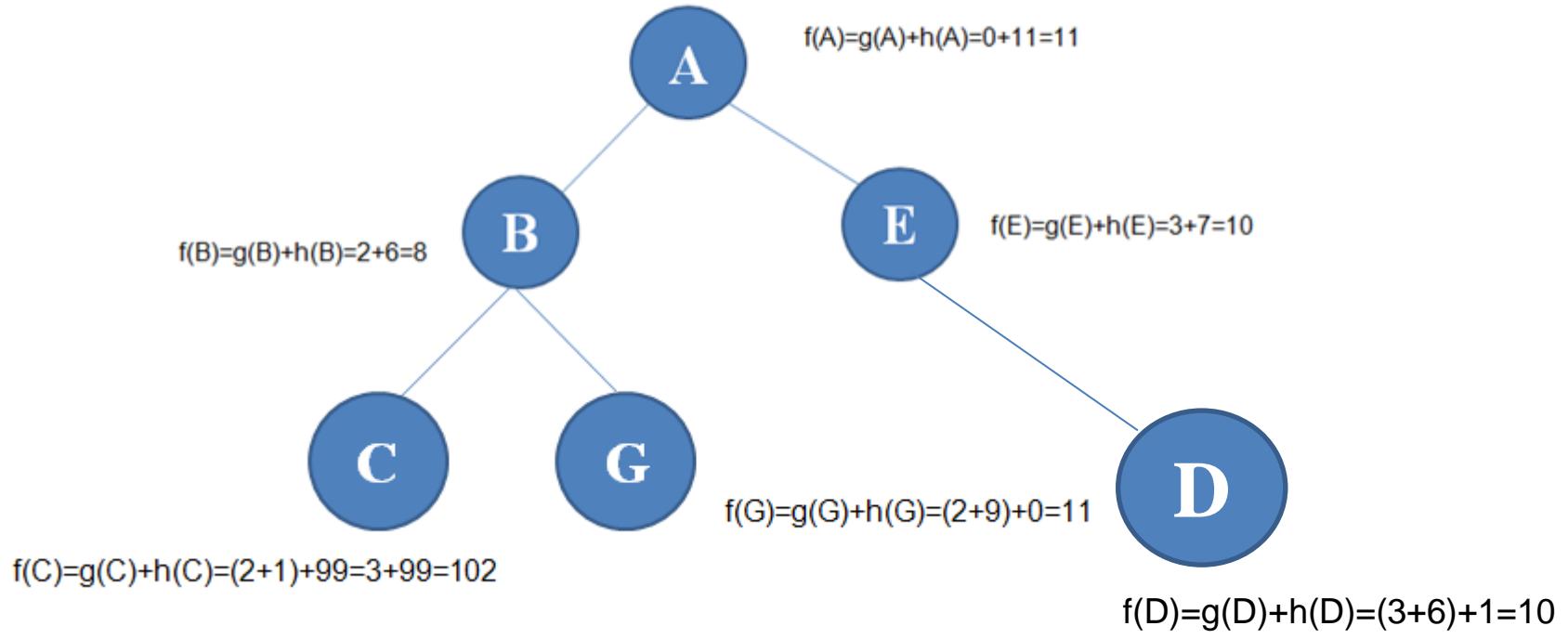
## Step 1:



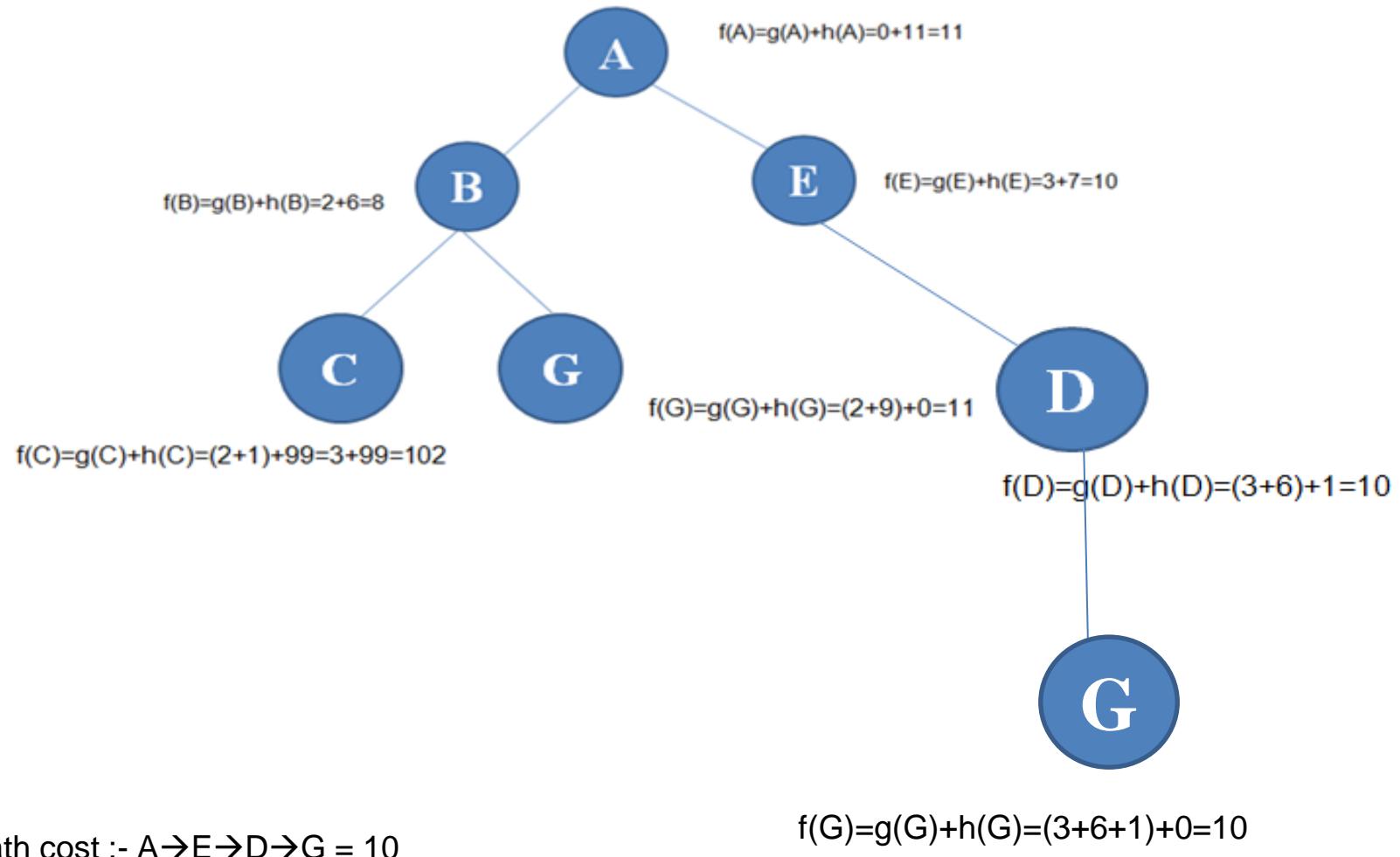
## Step 2:



## Step 3:



## Step 4:



# Optimality of A\*

## Test for Admissibility

Expands the node which lies in the closest path (estimated cheapest path) to the goal

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

$g(n)$  – the cost to reach the node

$h(n)$  – the expected cost to go from node to goal

$f(n)$  – estimated cost of cheapest path through node n

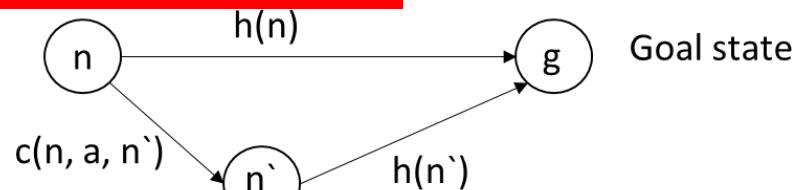
A heuristic is admissible or optimistic if ,  $0 \leq h(n) \leq h^*(n)$ , where  $h^*(n)$  is the actual cost to reach the goal

# A\* Search

## Optimal on condition

$h(n)$  must satisfies two conditions:

- Admissible Heuristic – one that never overestimates the cost to reach the goal
- Consistency – A heuristic is consistent if for every node  $n$  and every successor node  $n'$  of  $n$  generated by action  $a$ ,  $h(n) \leq c(n, a, n') + h(n')$



## Complete

- If the number of nodes with cost  $\leq C^*$  is finite
- If the branching factor is finite
- A\* expands no nodes with  $f(n) > C^*$ , known as pruning

Time Complexity -  $\mathcal{O}(b^\Delta)$  where the absolute error  $\Delta = h^* - h$



**BITS** Pilani  
Pilani Campus

# ACI Lab Session 2





# ACO-Pants

# ACO Pants

---

A Python3 implementation of the Ant Colony Optimization Meta-Heuristic.

**Pants** provides you with the ability to quickly determine how to visit a collection of interconnected nodes such that the work done is minimized.

Nodes can be any arbitrary collection of data while the edges represent the amount of “work” required to travel between two nodes.

The **world** is built from a list of nodes and a function responsible for returning the length of the edge between any two given nodes.

The length function need not return actual length.

Instead, “length” refers to that amount of “work” involved in moving from the first node to the second node - whatever that “work” may be.

For a silly, random example, it could even be the number of dishes one must wash before moving to the next station at a least dish-washing dish washer competition.

---

# Ant Colony Optimization – How it works

---

Solutions for problems are found through an iterative process.

In each iteration, several ants are allowed to find a solution that “visits” every node of the world.

The amount of pheromone on each edge is updated according to the length of the solutions in which it was used.

The ant that traveled the least distance is considered to be the local best solution.

If the local solution has a shorter distance than the best from any previous iteration, it then becomes the global best solution.

The elite ant(s) then deposit their pheromone along the path of the global best solution to strengthen it further, and the process repeats.

---

# World module

## **Class pants.world.Edge(*start*, *end*, *length=None*, *pheromone=None*)**

This class represents the link between starting and ending nodes.

In addition to *start* and *end* nodes, every Edge has length and pheromone properties.

**length** represents the static, *a priori* information, whereas

**pheromone** level represents the dynamic, *a posteriori* information.

### **Parameters:**

- **start** ([node](#)) – the node at the start of the Edge
- **end** ([node](#)) – the node at the end of the Edge
- **length** ([float](#)) – the length of the Edge (default=1)
- **pheromone** ([float](#)) – the amount of pheromone on the Edge (default=0.1)

# World module

## **Class pants.world.World(nodes, Ifunc, \*\*kwargs)**

The nodes and edges of a particular problem.

Each World is created from a list of nodes, a length function, and optionally, a name and a description.

Additionally, each World has a UID.

The length function must accept nodes as its first two parameters, and is responsible for returning the distance between them.

It is the responsibility of the **create\_edges()** to generate the required Edges and initialize them with the correct *length* as returned by the length function.

Once created, World objects convert the actual nodes into node IDs, since solving does not rely on the actual data in the nodes.

These are accessible via the nodes property.

To access the actual nodes, simply pass an ID obtained from nodes to the **data()** method which will return the node associated with the specified ID.

**Note:** **\*\*kwargs** allows us to pass a variable number of keyword arguments to a Python function.

# World module

Edges are accessible in much the same way, except two node IDs must be passed to the data() method to indicate which nodes start and end the Edge

For example:

```
ids = world.nodes  
assert len(ids) > 1  
node0 = world.data(ids[0])  
node1 = world.data(ids[1])  
edge01 = world.data(ids[0], ids[1])  
assert edge01.start == node0  
assert edge01.end == node1
```

# World Module

## **reset\_pheromone()** method

This method provides an easy way to reset the pheromone levels of every Edge contained in a World to a given *level*.

It should be invoked before attempting to solve a World unless a “blank slate” is not desired.

Also note that it should *not* be called between iterations of the solver because it effectively erases the memory of the Ant colony solving it.

Parameters:

**nodes** (list) – a list of nodes

**lfunc** (callable) – a function that calculates the distance between two nodes

**name** (str) – the name of the world (default is “world#”, where “#” is the uid of the world)

**description** (str) – a description of the world (default is None)

# World Module

## **create\_edges()**

Create edges from the nodes.

The job of this method is to map node ID pairs to Edge instances that describe the edge between the nodes at the given indices.

All of the Edges are created within this method.

**Returns:** a mapping of node ID pairs to Edge instances.

Return type: dict

# World Module

## **data(idx, idy=None)**

Return the node data of a single id or the edge data of two ids.

If only idx is specified, return the node with the ID idx.

If idy is also specified, return the Edge between nodes with indices idx and idy.

**Parameters:**

**idx** (int) – the id of the first node

**idy** (int) – the id of the second node (default is None)

**Returns:** the node with ID idx or the Edge between nodes with IDs idx and idy.

**Return type:** node or Edge

Here Nodes denote Node IDs.

# World Module

## **reset\_pheromone(level=0.01)**

Reset the amount of pheromone on every edge to some base level.

Each time a new set of solutions is to be found, the amount of pheromone on every edge should be equalized to ensure un-biased initial conditions.

Parameters:

level (float) – amount of pheromone to set on each edge (default=0.01)

# Ant module

## **Class pants.ant.Ant(alpha=1, beta=3)**

A single independent finder of solutions to a World.

Each Ant finds a solution to a world one move at a time.

They also represent the solution they find, and are capable of reporting which nodes and edges they visited, in what order they were visited, and the total length of the solution.

Two properties govern the decisions each Ant makes while finding a solution:

-alpha and beta.

- alpha controls the importance placed on pheromone while

- beta controls the importance placed on distance.

In general, beta should be greater than alpha for best results.

Ants also have a uid property that can be used to identify a particular instance.

# Ant module

Using the `initialize()` method, each Ant must be initialized to a particular World, and optionally may be given an initial node from which to start finding a solution.

If a starting node is not given, one is chosen at random.

A few examples of instantiation and initialization assuming existence of an already created World might look like:

```
ant = Ant()
```

```
ant.initialize(world)
```

```
ant = Ant().initialize(world)
```

```
ant = Ant(alpha=0.5, beta=2.25)
```

```
ant.initialize(world, start=world.nodes[0])
```

# Ant module

Once an Ant has found a solution (or at any time), the solution may be obtained and inspected by accessing its `tour` property, which returns the nodes visited in order, or its `path` property, which returns the edges visited in order.

Also, the total distance of the solution can be accessed through its `distance` property.  
Ants are even sortable by their distance:

```
ants = [Ant() for ...]  
# ... have each ant in the list solve a world  
ants = sorted(ants)  
for i in range(1, len(ants)):  
    assert ants[i - 1].distance < ants[i].distance
```

Ants may be cloned, which will return a shallow copy while not preserving the `uid` property.

If this behavior is not desired, simply use the `copy.copy()` or `copy.deepcopy()` methods as necessary.

# Ant module

The remaining methods mainly govern the mechanics of making each move.

**can\_move()** determines whether all possible moves have been made,

**remaining\_moves()** returns the moves not yet made,

**choose\_move()** returns a single move from a list of moves,

**make\_move()** actually performs the move, and **weigh()** returns the weight of a given move.

The **move()** method governs the move-making process by gathering the remaining moves, choosing one of them, making the chosen move, and returning the move that was made.

# Ant module

## **can\_move()**

Return True if there are moves that have not yet been made.

Return type: bool

## **choose\_move(choices)**

Choose a move from all possible moves.

**Parameters:** choices (list) – a list of all possible moves

**Returns:** the chosen element from choices

Return type: node

## **clone()**

Return a shallow copy with a new UID.

If an exact copy (including the uid) is desired, use the `copy.copy()` method.

**Returns:** a clone

Return type: Ant

# Ant module

## **initialize(world, start=None)**

Reset everything so that a new solution can be found.

Parameters:

world (World) – the world to solve

start (Node) – the starting node (default is chosen randomly)

Returns: self, Return type: Ant

## **make\_move(dest)**

Move to the dest node and return the edge traveled.

When dest is None, an attempt to take the final move back to the starting node is made. If that is not possible (because it has previously been done), then None is returned.

Parameters: dest (node) – the destination node for the move

Returns: the edge taken to get to dest, Return type: Edge

# Ant module

## **move()**

Choose, make, and return a move from the remaining moves.

Returns: the Edge taken to make the move chosen

Return type: Edge

## **node**

Most recently visited node.

## **path**

Edges traveled by the Ant in order.

## **remaining\_moves()**

Return the moves that remain to be made.

Return type: list

## **tour**

Nodes visited by the Ant in order.

# Ant module

## **weigh(edge)**

Calculate the weight of the given edge.

The weight of an edge is simply a representation of its perceived value in finding a shorter solution.

Larger weights increase the odds of the edge being taken, whereas smaller weights decrease those odds.

Parameters: edge (Edge) – the edge to weigh

Returns: the weight of edge

Return type: float

# Solver module

## `class pants.solver.Solver(**kwargs)`

This class contains the functionality for finding one or more solutions for a given World.

### Parameters:

**alpha** (float) – relative importance of pheromone (default=1)

**beta** (float) – relative importance of distance (default=3)

**rho** (float) – percent evaporation of pheromone (0..1, default=0.8)

**q** (float) – total pheromone deposited by each Ant after each iteration is complete (>0, default=1)

**t0** (float) – initial pheromone level along each Edge of the World (>0, default=0.01)

**limit** (int) – number of iterations to perform (default=100)

**ant\_count** (float) – how many Ants will be used (default=10)

**elite** (float) – multiplier of the pheromone deposited by the elite Ant (default=0.5)

# Solver module

## **aco(colony)**

Return the best solution by performing the ACO meta-heuristic.

This method lets every Ant in the colony find a solution, updates the pheromone levels according to the solutions found, and returns the Ant with the best solution.

This method is not meant to be called directly. Instead, call either solve() or solutions().

Parameters: colony (list) – the Ants to use in finding a solution

Returns: the best solution found, Return type: Ant

# Solver module

## **create\_colony(world)**

Create a set of Ants and initialize them to the given world.

If the ant\_count is less than 1, round\_robin\_ants() are used and the number of Ants will be equal to the number of nodes. Otherwise, random\_ants() are created instead, and the number of Ants will be equal to the ant\_count.

Parameters: world (World) – the world from which the Ants will be given starting nodes.

Returns: list of Ants

Return type: list

# Solver module

## **find\_solutions(ants)**

Let each Ant find a solution.

Makes each Ant move until each can no longer move.

Parameters: ants (list) – the ants to use for solving

## **global\_update(ants)**

Update the amount of pheromone on each edge according to the fitness of solutions that use it.

This accomplishes the global update performed at the end of each solving iteration.

**Note:** This method should never let the pheromone on an edge decrease to less than its initial level.

Parameters: ants (list) – the ants to use for solving

# Solver module

## **local\_update(edge)**

Evaporate some of the pheromone on the given edge.

Note: This method should never let the pheromone on an edge decrease to less than its initial level.

Parameters: edge (Edge) – the Edge to be updated

# Solver module

## **random\_ants(world, count, even=False)**

Returns a list of Ants distributed to the nodes of the world in a random fashion.

Note that this does not ensure at least one Ant begins at each node unless there are exactly as many Ants as there are nodes.

This method is used to create the Ants before solving if ant\_count is not 0.

### **Parameters:**

world (World) – the World in which to create the ants.

count (int) – the number of Ants to create

even (bool) – True if random.random() should avoid choosing the same starting node multiple times (default is False)

Returns: the Ants initialized to nodes in the World, Return type: list

# Solver module

---

`reset_colony(colony)`

Reset the colony of Ants such that each Ant is ready to find a new solution.

Essentially, this method re-initializes all Ants in the colony to the World that they were initialized to last.

Internally, this method is called after each iteration of the Solver.

Parameters: `colony` (list) – the Ants to reset

# Solver module

## **round\_robin\_ants(world, count)**

Returns a list of Ants distributed to the nodes of the world in a round-robin fashion.

Note that this does not ensure at least one Ant begins at each node unless there are exactly as many Ants as there are nodes.

However, if ant\_count is 0 then ant\_count is set to the number of nodes in the World and this method is used to create the Ants before solving.

### **Parameters:**

world (World) – the World in which to create the Ants

count (int) – the number of Ants to create

Returns: the Ants initialized to nodes in the World

Return type: list

# Solver module

## **solutions(world)**

Return successively shorter paths through the given world.

Unlike solve(), this method returns one solution for each improvement of the best solution found thus far.

Parameters: world (World) – the World to solve

Returns: successively shorter solutions as Ants

Return type: list

# Solver module

---

**solve(world)**

Return the single shortest path found through the given world.

Parameters: world (World) – the World to solve

Returns: the single best solution found

Return type: Ant

# Solver module

## **trace\_elite(ant)**

Deposit pheromone along the path of a particular ant.

This method is used to deposit the pheromone of the elite Ant at the end of each iteration.

**Note:**

This method should never let the pheromone on an edge decrease to less than its initial level.

**Parameters:** ant (Ant) – the elite Ant



# Reference

1. <https://aco-pants.readthedocs.io/en/latest/>
  
2. <https://pypi.org/project/ACO-Pants/>



**BITS** Pilani  
Pilani Campus

# ACI – Lab Session 2





# **Hill Climbing Algorithm Autonomous car driving agent**

**BITS Pilani**  
Pilani Campus

Optimizing attributes for fuel economy

# Hill Climbing Algorithm

---

Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing value to find the peak of the mountain or best solution to the problem.

It terminates when it reaches a peak value where no neighbor has a higher value.

One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.

---

# Hill Climbing Algorithm

---

It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.

A node of hill climbing algorithm has two components which are **state** and **value**.

Hill Climbing is mostly used when a good heuristic is available.

In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

# Hill climbing - function

---

**function HILL-CLIMBING(problem) returns a state that  
is a local maximum**

current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)

**loop do**

neighbor  $\leftarrow$  a highest-valued successor of current

if neighbor.VALUE  $\leq$  current.VALUE then return  
    current.STATE

current  $\leftarrow$  neighbor

---

# The problem - Attributes for fuel economy

---

**Performance measure** of an automated driver

Desirable qualities include getting to the correct destination; **minimizing fuel consumption** and wear and tear; minimizing the trip time or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

Fuel economy is calculated by using the formula

Average Fuel Consumption (Km/L) = Distance travelled / Litres of fuel consumed

Fuel consumption depends on other attributes like speed of the vehicle, time taken for the travel, type of roads, traffic, driving behavior, weather, weight, air conditioning, etc.

# The attributes/objectives for optimization in the example

---

Time to travel

Speed

Fuel consumed

Distance travelled

Optimizing these objectives is known as Multi-objective optimization

# Multi-objective optimization problem

---



A problem that takes more than one objective for optimization

The objectives can be contradicting ones

Meaning, it is desirable to increase the value of one objective while decreasing the value of others

The objectives are represented as a vector and can be evaluated using

- The vector as a whole
  - Defining a evaluation function using the objectives
-

# Cost function / Evaluation function

---

In the current autonomous car agent problem the cost or evaluation function is taken as follows:

$$\text{Cost} = \text{Time} + \text{Speed} + \text{Fuel} - \text{Distance}$$

Time travelled and fuel consumed must have lower values for cost reduction.

Speed must be average (not less not more)

Distance travelled must be maximum to reduce the cost

# Heuristic

---

Choose the optimal attribute values to get minimum cost

Goal = 0 cost

Heuristic – Hill climbing

Choose attribute vectors at random (Stochastic hill climbing)

Evaluate them

Retain the best while forgetting the less optimal values

# PEAS

**Performance** – Safety, time, legal drive, comfort.

- In the algorithm, reducing the cost of travel by optimizing the attributes taken

**Environment** – Roads, other cars, pedestrians, road signs.

- Acceptable range of values for the attributes Time, Speed, Fuel and Distance travelled represented as vector of combination of these values

**Actuators** – Steering, accelerator, brake, signal, horn.

- Initial solution, Candidate solution, Evaluation function.

**Sensors** – Fuel indicator, Speedometer, Digital clock in the display

- Candidate solution with the acceptable range of values (from the sensors), number of iterations.

# Hill climbing algorithm - Steps

---

**Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.

**Step 2:** Loop Until a solution is found or there is no new operator left to apply.

**Step 3:** Select and apply an operator to the current state.

**Step 4:** Check new state

**Step 5:** Exit.

# Code Walk through



**Stochastic hill climbing in  
PYTHON**

# #PEAS and the initial states

---

```
import numpy as np
```

```
import random;
```

## #The PEAS

```
#Time, Speed, Fuel, Distance
```

```
#Init_solution = [10, 100, 25, 10];
```

```
#Init_solution = [2,10,5,100]
```

```
#Init_solution = [0, 0, 0, 0];
```

```
Iterations = 0
```

```
max_itr=1000 #Can be increased or decreased
```

```
Final_solution = None #Stores best of each iteration
```

# #Class : Solution space

---

#Contains an objective vector of attributes to be optimized and the cost associated with the objectives

```
class Solu_space:  
    def __init__(self):  
        self.obj_vector = None  
        self.cost = None  
    def setObj_vector(self, val):  
        self.obj_vector = val  
    def setCost(self, cost):  
        self.cost = cost  
    def getAttr(self):  
        return {'Time, Speed, Fuel, Distance': self.obj_vector, 'Cost':  
        self.cost }
```

# #Function for cost evaluation

---

```
# Function for calculating the cost - The evaluation function
```

```
def calc_cost(objVector):
```

```
    cost = round((objVector[0] + objVector[1] +  
    objVector[2] - objVector[3]),2)
```

```
    #print("Cost"+str(cost))
```

```
    return abs(cost)
```

# #Function to generate new solution

```
def gen_new_solution():
    distance_step=0
    speed_step=random.randint(10,100)
    time_step=round((random.randint(1,100))/60,2)
    try:
        distance_step=random.randint(1,round(speed_step*time_step,2))
    except:
        pass
    fuel_step=random.randint(1,25)
    objVector=[time_step,speed_step,fuel_step,distance_step]
    #print("new solution: "+str(objVector))
    return objVector
```

# #Function Hill climb

---

```
def Hill_Climb(objVector):  
    soln_space = Solu_space()  
    soln_space.setObj_vector(objVector)  
    oldCost = calc_cost(objVector);  
    soln_space.setCost(oldCost)  
    newobjVector = gen_new_solution()  
    newcost= calc_cost(newobjVector)
```

# #Function Hill climb – compare solution costs

---

```
if(newcost < oldCost):  
    soln_space.setObj_vector(newobjVector)  
    soln_space.setCost(newcost)  
    print(str(soln_space.getAttr()))  
    return soln_space.obj_vector  
print(str(soln_space.getAttr()))  
return soln_space.obj_vector
```

---

# #Function – The stopping criterion

---

```
def stopit():
    globals()
    cost=calc_cost(Final_solution)
    if cost == 0:
        return True
    if Iterations == max_itr:
        return True
    else:
        return False
```

**CALLING ALL THE DEFINED  
FUNCTIONS**

# #Generate initial solution

---

```
#Time,speed,fuel,distance
```

```
Init_soln=
```

```
[random.randint(1,10),  
 random.randint(1,100),  
 random.randint(1,25),  
 random.randint(1,500)]
```

# #Loop the Hill Climb

---

```
Final_solution= Hill_Climb(Init_solution) #Initialize variable Final_solution
print(Final_solution)
while not stopit():
    solution=Final_solution
    Final_solution=Hill_Climb(solution)
    print("Iteration no: "+str(Iterations))
    Iterations+=1
print("Solution")
print(str(Final_solution[0])+ " Hrs " +str(Final_solution[1])+
      " Km/Hr " + str(Final_solution[2]) + " Litres " +
      str(Final_solution[3]) + " Kms")
print("Cost" + str(calc_cost(Final_solution)))
```

---

# Partial sample output

```
{"Time, Speed, Fuel, Distance": [1.3, 70, 12, 55], "Cost": 28.3} Iteration no: 35
{"Time, Speed, Fuel, Distance": [1.3, 70, 12, 55], "Cost": 28.3} Iteration no: 36
{"Time, Speed, Fuel, Distance": [1.3, 70, 12, 55], "Cost": 28.3} Iteration no: 37
{"Time, Speed, Fuel, Distance": [0.13, 10, 2, 0], "Cost": 12.13} Iteration no: 172
{"Time, Speed, Fuel, Distance": [0.13, 10, 2, 0], "Cost": 12.13} Iteration no: 173
{"Time, Speed, Fuel, Distance": [0.13, 10, 2, 0], "Cost": 12.13} Iteration no: 174
{"Time, Speed, Fuel, Distance": [0.13, 10, 2, 0], "Cost": 12.13} Iteration no: 175
{"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 991
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 992
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 993
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 994
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 995
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 996
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 997
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 998
 {"Time, Speed, Fuel, Distance": [1.5, 66, 2, 70], "Cost": 0.5} Iteration no: 999
```

Solution 1.5 Hrs 66 Km/Hr 2 Litres 70 Kms Cost 0.5

# Food for thought

---

Add code to convert stochastic hill climb into Random restart hill climb

Solve the problem using Simulated annealing

Implement the solution for the given problem using genetic algorithm

**That's all for Hill Climbing**

---



**BITS** Pilani  
Pilani Campus

# ACI – Lab Session 2



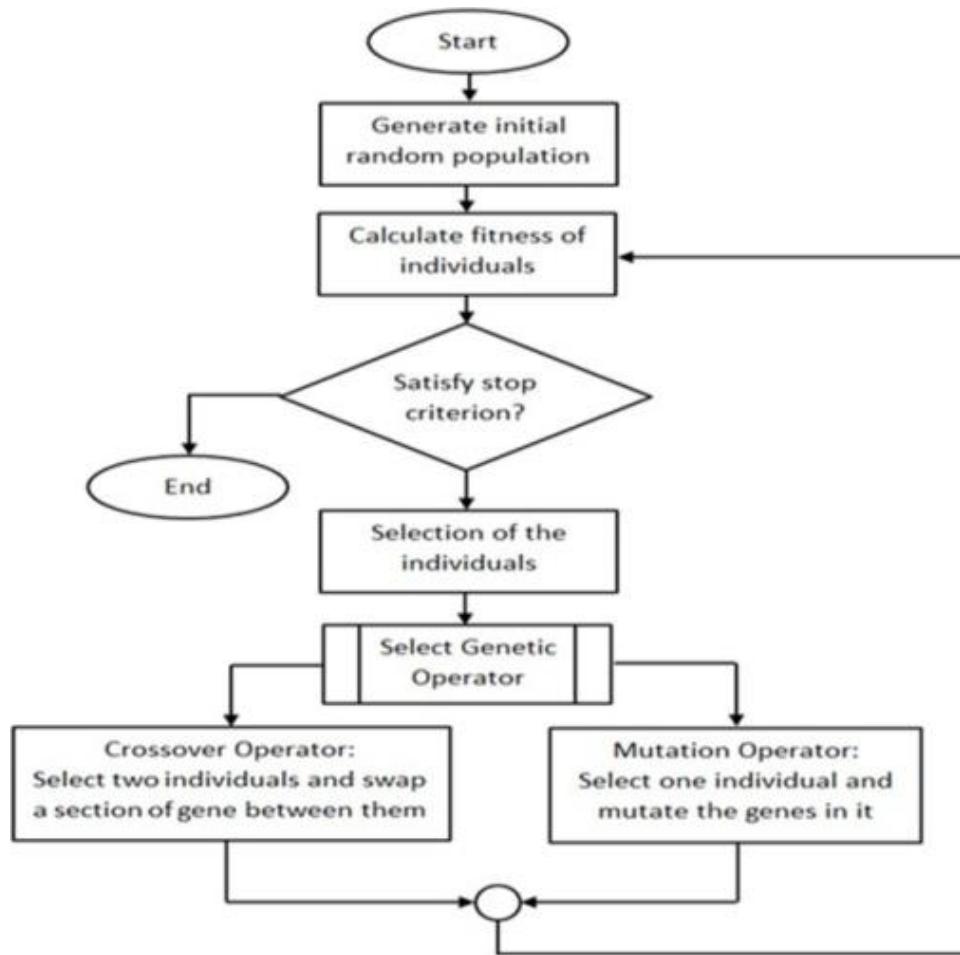


# Genetic Algorithm For N Queens problem



**BITS** Pilani  
Pilani Campus

# GA - Flowchart



# Genetic Algorithm

**function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual**

**inputs: population, a set of individuals**

**FITNESS-FN, a function that measures the fitness of an individual**

**repeat**

new population  $\leftarrow$  empty set

**for i = 1 to SIZE(population) do**

x  $\leftarrow$  RANDOM-SELECTION(population, FITNESS-FN)

y  $\leftarrow$  RANDOM-SELECTION(population, FITNESS-FN)

child  $\leftarrow$  REPRODUCE(x , y)

**if (small random probability) then child  $\leftarrow$  MUTATE(child )**

add child to new population

population  $\leftarrow$  new population

**until some individual is fit enough, or enough time has elapsed**

**return the best individual in population, according to FITNESS-FN**

# Genetic Algorithm - Reproduction

---

**function REPRODUCE(x , y) returns an individual**

**inputs: x , y, parent individuals**

n←LENGTH(x );

c←random number from 1 to n

**return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))**

**Note:** This reproduce function produces a single child

# N Queens problem

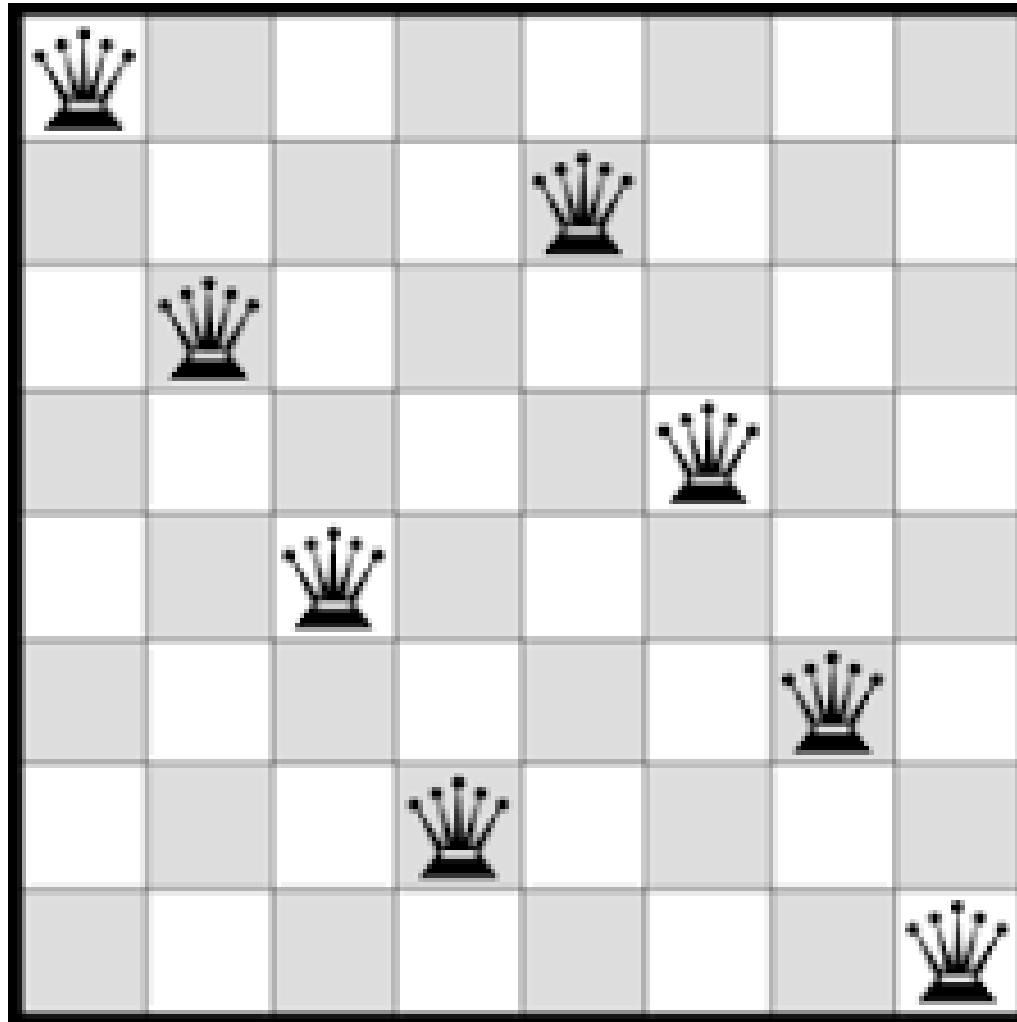
---

The **eight queens puzzle** is the **problem** of placing **eight chess queens** on an **8×8** chessboard so that no two **queens** threaten each other.

The solution requires that no two **queens** share the same row, column, or diagonal.

---

# N Queens problem



# N Queens problem - Representation

	0	1	2	3	4	5	6	7
0	♛							
1					♛			
2							♛	
3						♛		
4			♛					
5						♛		
6		♛						
7				♛				

The board configuration is represented as (0,4,7,5,2,6,1,3)  
 Note: Here row indicates the queen number while column indicates the queens position. Vice versa can also be used.

# N Queens problem - PEAS

---

- **States:** All possible arrangements of n queens ( $0 \leq n \leq 7$ ), one per column in the leftmost n columns, with no queen attacking another.
- **Actions:** Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

The goal of the **8-queens problem** is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.)

---



# Code Walk through

**GA for n queens - Python**

# #Import Libraries and Initialize states

---



```
#Import libraries numpy and sys  
import numpy as np  
import sys
```

```
#The initial states -- PEAS  
nQueens = 8  
STOP_CTR = 28  
MUTATE_FLAG = True  
MAX_ITER = 10000  
POPULATION = None
```

# #Representation of the States - Define class Boardposition



```
#Define class Boardposition
```

```
class BoardPosition:
```

```
    def __init__(self):
```

```
        self.sequence = None
```

```
        self.fitness = None
```

```
        self.survival = None
```

```
    def setSequence(self, val):
```

```
        self.sequence = val
```

```
    def setFitness(self, fitness):
```

```
        self.fitness = fitness
```

```
    def setSurvival(self, val):
```

```
        self.survival = val
```

```
    def getAttr(self):
```

```
        return {'sequence': self.sequence, 'fitness': self.fitness, 'survival':  
            self.survival}
```

# #Calculate fitness of the candidate solution

---

"" To test for conflicts, we check for  
-> row conflicts -> columnar conflicts -> diagonal conflicts  
The ideal case can yield up to 28 arrangements of non  
attacking pairs (i.e. 28 solutions).

for iteration 0 -> there are 7 non attacking queens

for iteration 1 -> there are 6 non attacking queens .... and  
so on

Therefore max fitness =  $7 + 6 + 5 + 4 + 3 + 2 + 1 = 28$

Hence fitness value returned will be  $28 - <\text{number of clashes}>$  ""

# #Calculate fitness of the candidate solution - Row and Column clashes

---

```
# calculate row and column clashes
# just subtract the unique length of array from total length
# of array
# E.g. [1,1,1,2,2,2] - [1,2] => 4 clashes
def fitness(chromosome=None):
    clashes = 0
    row_col_clashes = abs(len(chromosome) -
    len(np.unique(chromosome)))
    clashes += row_col_clashes
```

# #Calculate fitness of the candidate solution

---

## # calculate diagonal clashes

```
for i in range(len(chromosome)):  
    for j in range(len(chromosome)):  
        if (i != j):  
            dx = abs(i - j)  
            dy = abs(chromosome[i] - chromosome[j])  
            if (dx == dy):  
                clashes += 1  
return 28 - clashes
```

# #Chromosome (Candidate solution generation)

---

```
def generateChromosome():
    # randomly generates a sequence of board states.
    global nQueens
    init_distribution = np.arange(nQueens) #nQueens = 8
    np.random.shuffle(init_distribution)
    return init_distribution
```

# #Generate a population of candidate solutions

---

```
def generatePopulation(population_size=100):
    global POPULATION
    POPULATION = population_size
    population = [BoardPosition() for i in range(population_size)]
    for i in range(population_size):
        population[i].setSequence(generateChromosome())
        population[i].setFitness(fitness(population[i].sequence))
    summation_fitness = np.sum([x.fitness for x in population])
    for each in population:
        each.survival = each.fitness / (summation_fitness * 1.0)
    return population
```

# #Get parents for reproduction

---

```
def getParent():
    globals()
    parent1, parent2 = None, None
# parent is decided by random probability of survival.
# since the fitness of each board position is an integer >0,
# we need to normalize the fitness in order to find the
# solution
```

---

# #Get parents for reproduction

## – Parent1

---

```
while True:
```

```
    parent1_random = np.random.rand()
```

```
#parent1_rn contains all the candidate solutions whose survival <
parent1_random
```

```
    parent1_rn = [x for x in population if x.survival <=
parent1_random]
```

```
    try:
```

```
#Choose the first element from parent1_rn as the first parent
```

```
        parent1 = parent1_rn[0]
```

```
        break
```

```
    except:
```

```
        pass
```

# #Get parents for reproduction

## – Parent2

```
while True:
```

```
    parent2_random = np.random.rand()
```

```
#parent2_rn contains all the candidate solutions whose survival < parent2_random
```

```
    parent2_rn = [x for x in population if x.survival <= parent2_random]
```

```
    try:
```

```
#Generate random number from 0 to no. of candidates in parent2_rn
```

```
        t = np.random.randint(len(parent2_rn))
```

```
#Choose the candidate at position t in parent2_rn as parent2
```

```
        parent2 = parent2_rn[t]
```

```
        if parent2 != parent1:
```

```
            break
```

```
        else:
```

```
            continue
```

```
    except:
```

```
        continue
```

```
if parent1 is not None and parent2 is not None:
```

```
    return parent1, parent2
```

```
else:
```

```
    sys.exit(-1)
```

# #Reproduction - Crossover

---

```
def reproduce_crossover(parent1, parent2):
    globals()
    n = len(parent1.sequence)
    c = np.random.randint(0, n) #Random position
    child = BoardPosition()
    child.sequence = []
    #Swap the contents starting from random position
    child.sequence.extend(parent1.sequence[0:c])
    child.sequence.extend(parent2.sequence[c:])
    child.setFitness(fitness(child.sequence))
    return child
```

---

# #Reproduction - Mutation

---

```
def mutate(child):
    """
        - according to genetic theory, a mutation will take
        place when there is an anomaly during cross over state
        - since a computer cannot determine such anomaly, we
        can define the probability of developing such a mutation
    """

if child.survival < MUTATE:
    c = np.random.randint(8)
    child.sequence[c] = np.random.randint(8)
# print "Inside Mutation"
return child.sequence
```

# #The GA – Calls crossover and mutation for producing new population

---



```
def GA(iteration):
    print("#" * 10, "Executing Genetic generation : ", iteration, "#" * 10)
    globals()
    newpopulation = []
    for i in range(len(population)):
        parent1, parent2 = getParent()
        child = reproduce_crossover(parent1, parent2)
        newpopulation.append(child)
    summation_fitness = np.sum([x.fitness for x in newpopulation])
    for each in newpopulation:
        each.survival = each.fitness / (summation_fitness * 1.0)
```

# #The GA – Calls crossover and mutation for producing new population

---

```
if (MUTATE_FLAG):  
    for each in newpopulation:  
        presentVal = each.sequence  
        mightBeChangedVal = mutate(each)  
        if presentVal!=mightBeChangedVal:  
            each.sequence = presentVal  
            each.fitness =  
            each.setFitness(fitness(each.sequence))  
        summation_fitness = np.sum([x.fitness for x in  
        newpopulation])  
        for each in newpopulation:  
            each.survival = each.fitness / (summation_fitness * 1.0)  
    return newpopulation
```

# #The stopping criteria

---

```
def stop():
    globals()
    fitnessvals = [pos.fitness for pos in population]
    if STOP_CTR in fitnessvals: #STOP_CTR = 28
        return True
    if MAX_ITER == iteration:
        return True
    return False
```

---

# #The main iteration of GA

---

```
population_size = 100
MUTATE = 1/population_size
population = generatePopulation(population_size)
iteration = 0
while not stop():
    # keep iterating till you find the best position
    population = GA(iteration)
    iteration += 1
    print("Iteration Number: ", iteration)
    for each in population:
        if each.fitness == 28:
            print(each.sequence)
```

# Sample partial output

```
# ##### Executing Genetic generation : 4293 #####
# ##### Executing Genetic generation : 4294 #####
# ##### Executing Genetic generation : 4295 #####
# ##### Executing Genetic generation : 4296 #####
# ##### Executing Genetic generation : 4297 #####
# ##### Executing Genetic generation : 4298 #####
# ##### Executing Genetic generation : 4299 #####
# ##### Executing Genetic generation : 4300 #####
# ##### Executing Genetic generation : 4301 #####
# ##### Executing Genetic generation : 4302 #####
# ##### Executing Genetic generation : 4303 #####
# ##### Executing Genetic generation : 4304 #####
# ##### Executing Genetic generation : 4305 #####
# ##### Executing Genetic generation : 4306 #####
# ##### Executing Genetic generation : 4307 #####
# ##### Executing Genetic generation : 4308 #####
# ##### Executing Genetic generation : 4309 #####
Iteration Number: 4310 [1, 4, 6, 0, 2, 7, 5, 3]
```

# Food for thought

---

Write implementation to solve the problem in Python using Hill climbing algorithm

Change the number of queens for example take the number of queens to be 4, 9, 10, etc. getting the number of queens as input from the user.



**BITS** Pilani  
Pilani Campus

# Lab Session 2



# Ant Colony Optimization

# ACO Pants

---

A Python3 implementation of the Ant Colony Optimization Meta-Heuristic.

**Pants** provides you with the ability to quickly determine how to visit a collection of interconnected nodes such that the work done is minimized.

Nodes can be any arbitrary collection of data while the edges represent the amount of “work” required to travel between two nodes.

The **world** is built from a list of nodes and a function responsible for returning the length of the edge between any two given nodes.

The length function need not return actual length.

Instead, “length” refers to the amount of “work” involved in moving from the first node to the second node - whatever that “work” may be.

# Ant Colony Optimization – How it works

---

Solutions for problems are found through an iterative process.

In each iteration, several ants are allowed to find a solution that “visits” every node of the world.

The amount of pheromone on each edge is updated according to the length of the solutions in which it was used.

The ant that traveled the least distance is considered to be the local best solution.

If the local solution has a shorter distance than the best from any previous iteration, it then becomes the global best solution.

The elite ant(s) then deposit their pheromone along the path of the global best solution to strengthen it further, and the process repeats.

---

# The Problem

---

The **Travelling Salesman Problem (TSP)** is one of the most famous problems in computer science for studying optimization.

The objective is to find a complete route with the following conditions:

- The route must connect all the nodes of a network.
  - The nodes must be visited only once and
  - Return to the starting point
  - The total distance of the route must be the minimum.
-

# The Solution - ACO

The basic idea underlying all the ant-based algorithm is to use a positive feedback mechanism based on the laying pheromone.

The pheromone component allows the best solutions found to be kept in memory, which can be used to make up better solutions.

To avoid stagnation of the algorithm a form of negative feedback is implemented through pheromone evaporation, but it must not evaporate too fast in order to make cooperation behavior emerge.

In the TSP the goal is to find the tour with minimal length connecting  $n$  given cities, and each city must be visited only once.

# The Solution - Distance

The distance between cities can be defined by Euclidean distance or other distance functions.

In the graph the cities would be the nodes and the connections between the cities are the edges of the graph.

The graph does not need to be fully connected, all the nodes may not be connected to all the other nodes.

And the distances may not be symmetric, distance  $ij$  may be different from that of distance  $ji$ .

# The Solution - Transition

---

In order to solve the TSP using ACO the transitions of the ants from city to city depends on the following premises:

- Whether or not the city has already been visited. Each ant has a memory or tabu list to make sure each city is visited once per tour.
- The inverse of the distance between two nodes (visibility). Visibility is based on local information and represents the heuristic desirability of choosing city  $j$  when in city  $i$ .
- The amount of virtual pheromone on the edges. It is a global type of information, represents the learned desirability of choosing city  $j$  when in city  $i$ .

# The Solution - Parameters

---

Other things to take into account are:

- The transition rule: the probability for an ant to go from city  $i$  to city  $j$  while building a tour.
- Pheromone decay: Without pheromone decay the algorithm would lead to amplification of the initial random fluctuation, that will produce non optimal solutions.
- Total number of ants: It is an important parameter since too many ants will reinforce non optimal solutions, while too few ants would not produce cooperative effect due to pheromone decay. It is suggested to use a number of ants equal to the number of cities in the graph.



# **Code Walk through – Finding the shortest path around Indian cities**

# 1. Import packages

---

```
# Installing ACO-Pants  
pip install ACO-Pants  
#import packages  
import pants  
import math  
import random
```

## 2. Input

---

The input for ACOPants is a list of coordinates ( $x, y$ ) of the nodes, and providing a length function to the algorithm this is able to calculate the distances from node  $i$  to  $j$ .

Here we have get a csv .file that contains information about cities all around the world from the webpage: <http://simplemaps.com/data/world-cities>.

We will work with the cities from India and with the coordinates in decimal degrees (latitude and longitude).

## 2. Input

---

```
import pandas as pd
import numpy as np
cities = pd.read_csv('worldcities.csv', decimal=".")
INDIAcities = cities.loc[cities['country'] == 'India']
#only the cities that belong to INDIA
print('Dimension INDIAcities:', INDIAcities.shape)
#dimension of INDIAcities dataset
```

---

### 3. View the data

---

```
#Get sample of 100 cities  
Indiancities = INDIAcities.sample(100) #to get a sample of  
# 100 rows to work with  
print('Dimension Indiancities:', Indiancities.shape)  
#dimension USCities dataset  
Indiancities.head() #first rows from the new dataset
```

---

## 4. Define the Distance between edges

---

#To calculate the distances from node  $i$  to  $j$ , we are going to use Euclidean distance, which is the straight-line distance between two points or nodes.

```
def euclidean(a, b):  
    return math.sqrt(pow(a[1] - b[1], 2) + pow(a[0] - b[0],  
2))
```

# # 5. Node, Edge initialization

---

```
#Since the input is a list of nodes(x,y) :  
#Initialize the nodes using latitude and longitude  
x = Indiancities['lat']  
y = Indiancities['lng']  
DD = list(zip(x,y)) #Indian cities represented in decimal  
degrees  
print(DD)
```

# Hyper Parameters

---

Optional arguments:

- a A, --alpha A relative importance placed on pheromones; default=1
- b B, --beta B relative importance placed on distances; default=3
- l L, --limit L number of iterations to perform; default=100
- p P, --rho P ratio of evaporated pheromone ( $0 \leq P \leq 1$ ); default=0.8
- e E, --elite E ratio of elite ant's pheromone; default=0.5
- q Q, --Q Q total pheromone capacity of each ant ( $Q > 0$ ); default=1
- t T, --t0 T initial amount of pheromone on every edge ( $T > 0$ ); default=0.01
- c N, --count N number of ants used in each iteration ( $N > 0$ ); default=10

Arguments are very important and they can affect the result. Usually, it is used as many number of ants (N) as nodes. Also, is better to use a higher value of beta(distance) than beta(pheromone).

---

# # 6. Define the world

---

#Parameters

#Here we will use a number of ants less than number of nodes (N= 5) .

#Number of iterations L = 5 instead of 100.

#Alpha and beta with the same relative importance (A, B = 1)

```
world = pants.World(DD, euclidean, N = 5, L = 5 , A = 1, B = 1)
```

# # 7. The solver and the solution

---

```
solver = pants.Solver()  
solution = solver.solve(world)  
print('DISTANCE:', solution.distance) #total distance of  
#the tour performed  
tour = solution.tour      #nodes visited in order  
print(tour)  
#To get the names of the cities visited from the nodes  
values:  
UScities.set_index(['lat','lng'])['city'].loc[tour].tolist()  
)
```

---



# Reference

1. <https://aco-pants.readthedocs.io/en/latest/>
2. <https://pypi.org/project/ACO-Pants/>



Thank you  
Questions?

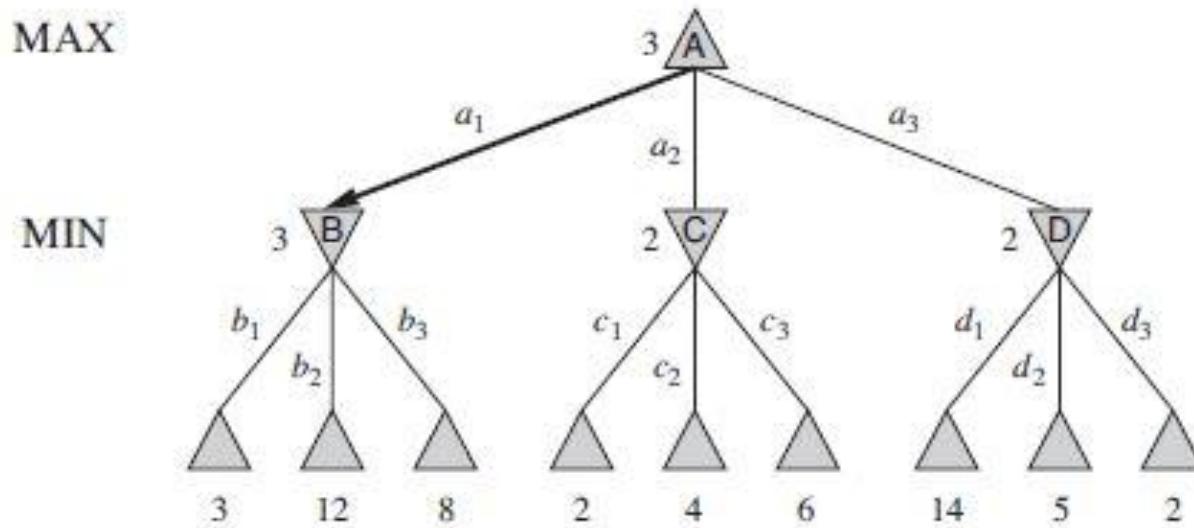
# **ACI – Lab Session 2**

**Min Max algorithm with  
Alpha-Beta pruning  
For  
Tic Tac Toe**

# Min Max algorithm

- An algorithm for calculating minimax decisions.
- It returns the action corresponding to the best possible move, that is, the move that leads to the outcome with the best utility, under the assumption that the opponent plays to minimize utility.
- The functions MAX-VALUE and MIN-VALUE go through the whole game tree, all the way to the leaves, to determine the backed-up value of a state.
- The notation  $\text{argmax}_{a \in S} f(a)$  computes the element  $a$  of set  $S$  that has the maximum value of  $f(a)$ .

# Min Max algorithm



**Figure 5.2** A two-ply game tree. The  $\triangle$  nodes are “MAX nodes,” in which it is MAX’s turn to move, and the  $\diamond$  nodes are “MIN nodes.” The terminal nodes show the utility values for MAX; the other nodes are labeled with their minimax values. MAX’s best move at the root is  $a_1$ , because it leads to the state with the highest minimax value, and MIN’s best reply is  $b_1$ , because it leads to the state with the lowest minimax value.

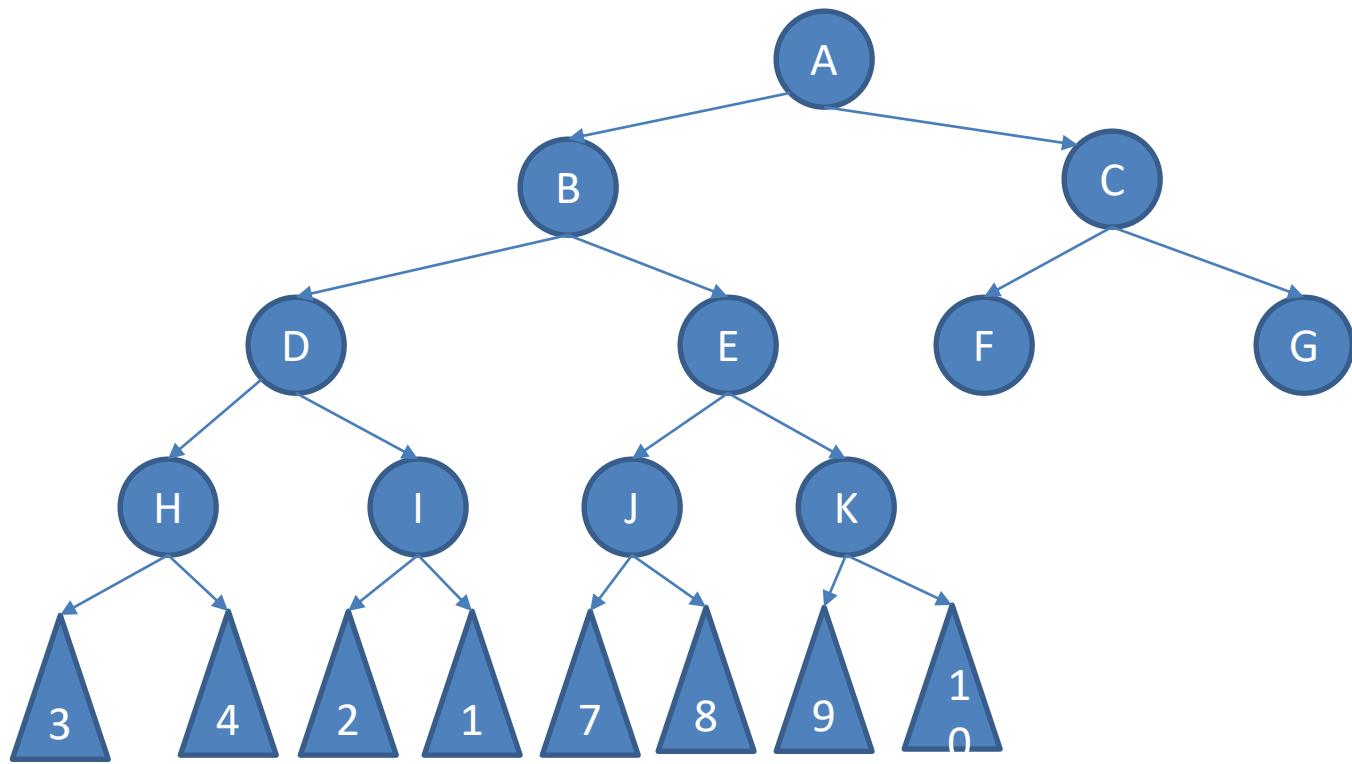
# Alpha Beta pruning

- The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree.
- We can effectively cut it in half without looking at every node in the game tree.
- The particular technique we examine is called **ALPHA–BETA pruning**.
- **When applied to a standard PRUNING** minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

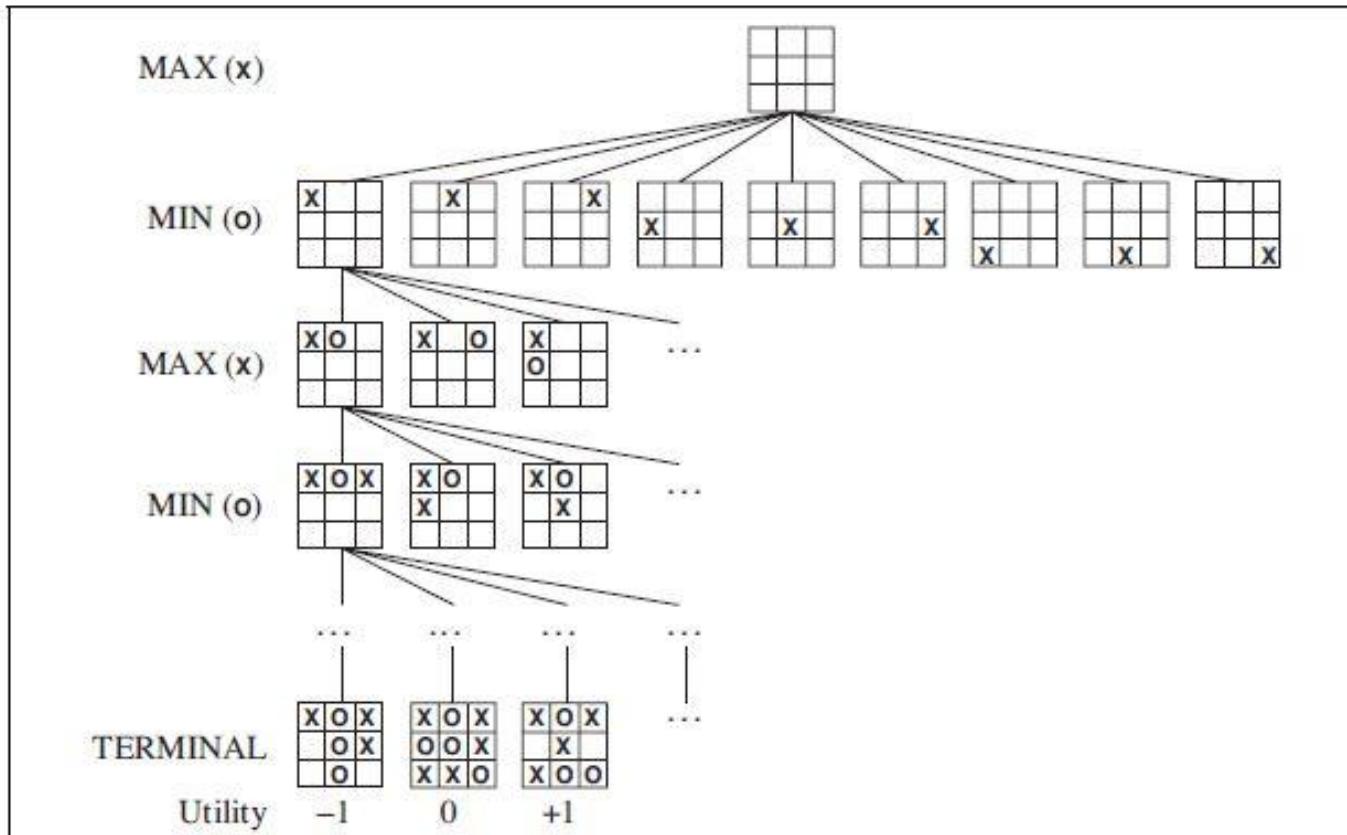
# Alpha Beta pruning

- $\alpha$  = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.
- $\beta$  = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.
- Alpha–beta search updates the values of  $\alpha$  and  $\beta$  as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current  $\alpha$  or  $\beta$  value for MAX or MIN, respectively

# Example

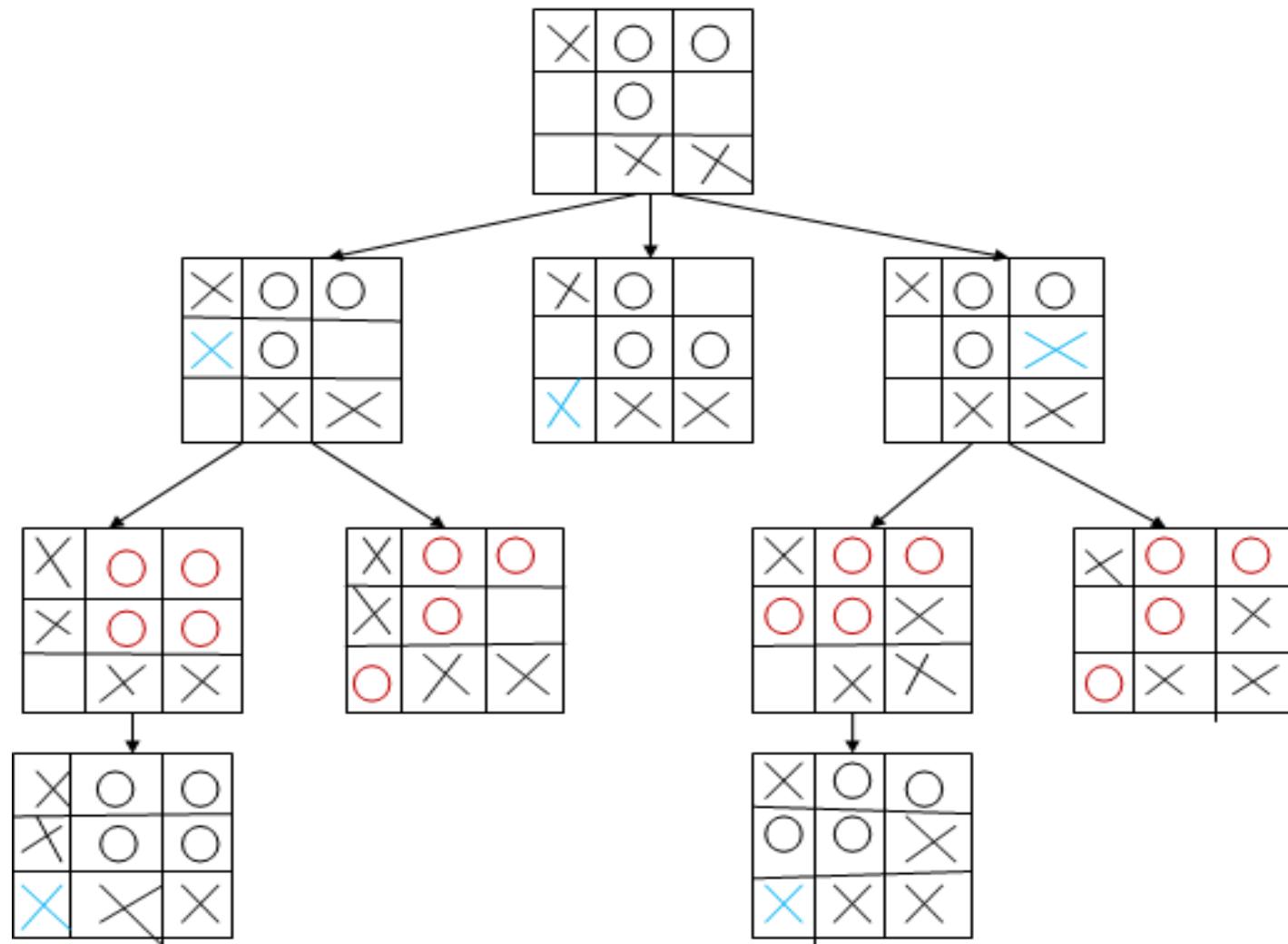


# The problem –Tic Tac Toe game



**Figure 5.1** A (partial) game tree for the game of tic-tac-toe. The top node is the initial state, and MAX moves first, placing an X in an empty square. We show part of the tree, giving alternating moves by MIN (O) and MAX (X), until we eventually reach terminal states, which can be assigned utilities according to the rules of the game.

# Tic Tac Toe game



**Code Walk through**

**TIC TAC TOE WITH MIN MAX  
ALGORITHM AND ALPHA BETA  
PRUNING IN PYTHON**

# #Class TicTacToe

- from math import inf as infinity
- class TicTacToe:
- def \_\_init\_\_(self):
- self.initialize\_board()
- def initialize\_board(self):
- self.current\_state = [['-','-','-'],
- ['-','-','-'],
- ['-','-','-']]
- # Player X always plays first
- self.player\_turn = 'X'

# #Class TicTacToe – Draw the board

- def draw\_board(self):
- for i in range(0, 3):
- for j in range(0, 3):
- print('{}'.format(self.current\_state[i][j]),  
end=" ")
- print()
- print()

# # Determines if the made move is a legal move

- def is\_validmove(self, px, py):
- if px < 0 or px > 2 or py < 0 or py > 2:
- return False
- elif self.current\_state[px][py] != '-':
- return False
- else:
- return True

# Checks if the game has ended and returns the winner in each case - #**is\_endofgame()** function

- def **is\_endofgame(self):**
- # Check for column-wise win
- for i in range(0, 3):
- if (self.current\_state[0][i] != '-' and
- self.current\_state[0][i] ==
- self.current\_state[1][i] and
- self.current\_state[1][i] ==
- self.current\_state[2][i]):
- return self.current\_state[0][i]

# is\_endofgame() function

- # Check for row-wise win
- for i in range(0, 3):
- if (self.current\_state[i] == ['X', 'X', 'X']):
- return 'X'
- elif (self.current\_state[i] == ['O', 'O', 'O']):
- return 'O'

# # is\_endofgame() function

- # Check for win over the Main diagonal
- if (self.current\_state[0][0] != '-' and
- self.current\_state[0][0] ==  
self.current\_state[1][1] and
- self.current\_state[0][0] ==  
self.current\_state[2][2]):
- return self.current\_state[0][0]

# # is\_endofgame() function

- # Check for win over the Second diagonal
- if (self.current\_state[0][2] != '-' and
- self.current\_state[0][2] ==  
self.current\_state[1][1] and
- self.current\_state[0][2] ==  
self.current\_state[2][0]):
- return self.current\_state[0][2]

# # is\_endofgame() function

- # Is whole board full?
  - for i in range(0, 3):
    - for j in range(0, 3):
- # There's an empty field, we continue the game
  - if (self.current\_state[i][j] == '-'):
    - return None
- # It's a tie!
  - return '-'

# #Max player

- def max\_alpha\_beta(self, alpha, beta):
  - maxv = -infinity
  - px = None
  - py = None
- #Check for end of game
  - result = self.is\_endofgame()
- if result == 'X':
  - return (-1, 0, 0)
- elif result == 'O':
  - return (1, 0, 0)
- elif result == '-':
  - return (0, 0, 0)

# #Max player - #Testing for best remaining moves, calculating the utility function and updating alpha

- for i in range(0, 3):
  - for j in range(0, 3):
    - if self.current\_state[i][j] == '-':
      - self.current\_state[i][j] = 'O'
      - (m, min\_i, min\_j) = self.min\_alpha\_beta(alpha, beta)
      - if m > maxv: #Update Maxvalue
        - maxv = m
        - px = i
        - py = j
      - #After checking reset to empty state
        - self.current\_state[i][j] = '-'

## #Max player - #Testing for best remaining moves, calculating the utility function and updating alpha

- # Next two ifs in Max and Min are the only difference between regular algorithm and alpha-beta
- if maxv >= beta:
  - return (maxv, px, py)
  - #Update alpha
  - if maxv > alpha:
    - alpha = maxv
- return (maxv, px, py)

# #Min player

- def min\_alpha\_beta(self, alpha, beta):
- minv = infinity
- qx = None
- qy = None
- #Check for end of game
- result = self.is\_endofgame()
- if result == 'X':
- return (-1, 0, 0)
- elif result == 'O':
- return (1, 0, 0)
- elif result == '-':
- return (0, 0, 0)

# #Min player - #Testing for best remaining moves, calculating the utility function and updating beta

- for i in range(0, 3):
  - for j in range(0, 3):
    - if self.current\_state[i][j] == '-':
      - self.current\_state[i][j] = 'X'
      - (m, max\_i, max\_j) =
        - self.max\_alpha\_beta(alpha, beta)
      - if m < minv: #update Minvalue
        - minv = m
        - qx = i
        - qy = j
      - self.current\_state[i][j] = '-'

#Min player - #Testing for best remaining moves,  
calculating the utility function and updating beta

- #Alpha Beta pruning
- if minv <= alpha:
- return (minv, qx, qy)
- #Update beta
- if minv < beta:
- beta = minv
- return (minv, qx, qy)

**CALLING ALL GAME**

# #Play tic tac toe

- def play\_alpha\_beta(self):
- while True:
- self.draw\_board()
- self.result = self.is\_endofgame()
- if self.result != None:
- if self.result == 'X':
- print('The winner is X!')
- elif self.result == 'O':
- print('The winner is O!')
- elif self.result == '-':
- print("It's a tie!")
- self.initialize\_board()
- return

# #Playing the game (Min turn)

- #px, py, qx and qy represent the board positions as x and y coordinates
- if self.player\_turn == 'X':
  - while True:
    - (m, qx, qy) = self.min\_alpha\_beta(-infinity, infinity)
    - px = int(input('Insert the X coordinate: '))
    - py = int(input('Insert the Y coordinate: '))
    - qx = px
    - qy = py
    - if self.is\_validmove(px, py):
      - self.current\_state[px][py] = 'X'
      - self.player\_turn = 'O'
      - break
    - else:
      - print('The move is not valid! Try again.')

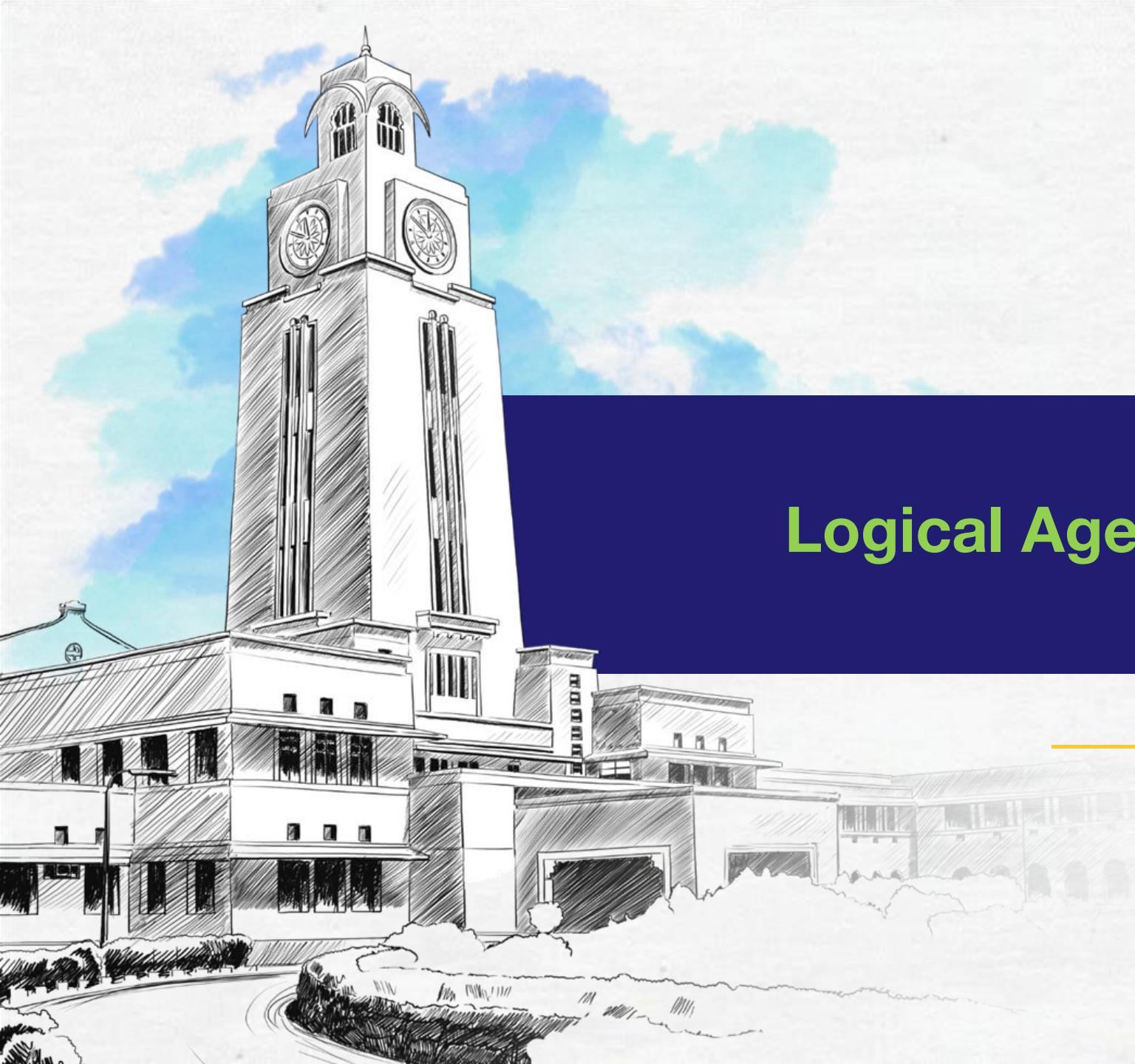
# #Playing the game (Max turn)

- else:
- (m, px, py) =
- self.max\_alpha\_beta(-infinity, infinity)
- self.current\_state[px][py] = 'O'
- self.player\_turn = 'X'
-

# #Defining and Calling the main function

- def main():
  - g = TicTacToe()
  - g.play\_alpha\_beta()
- 
- if \_\_name\_\_ == "\_\_main\_\_":
  - main()

Thank you  
That's all for the day



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# ACI Webinar #3

## Logical Agents And Introduction to Prolog

**Shyamala B**  
WILP Division, BITS-Pilani

# Session-1 Objectives

1. Solving some problems on
  - Propositional Logic – Truth tables , Propositional formulation
  - Normal Forms
2. Introduction to Prolog
  - What is Prolog Programming ?
  - Basics of Prolog
    - Facts & Rules
    - Terms
    - Queries
  - Simple Prolog programs
  - How Prolog Works ?
3. Exercises
4. Feedback!



# Problems on Basic Concepts

*Which of the following are well formed propositional formulas?*

1.  $\vee pq$
2.  $(\neg(p \rightarrow (q \wedge p)))$
3.  $(\neg(p \rightarrow (q = p)))$
4.  $(\neg(\Diamond(q \vee p)))$
5.  $(p \wedge \neg q) \vee (q \rightarrow r)$
6.  $p \neg r$

*Well formed formulas: 2. and 5.*

# Problems on Basic Concepts

Use the truth tables method to determine whether  $(p \rightarrow q) \vee (p \rightarrow \neg q)$  is valid.

$p$	$q$	$p \rightarrow q$	$\neg q$	$p \rightarrow \neg q$	$(p \rightarrow q) \vee (p \rightarrow \neg q)$
$T$	$T$	$T$	$F$	$F$	$T$
$T$	$F$	$F$	$T$	$T$	$T$
$F$	$T$	$T$	$F$	$T$	$T$
$F$	$F$	$T$	$T$	$T$	$T$

The formula is valid since it is satisfied by every interpretation.

# Problem on Normal Forms

Reduce to Conjunctive Normal Form (CNF) the formula

$$\neg(\neg p \vee q) \vee (r \rightarrow \neg s)$$

## Solution.

1.  $\neg(\neg p \vee q) \vee (\neg r \vee \neg s)$

2.  $(\neg\neg p \wedge \neg q) \vee (\neg r \vee \neg s)$

3.  $(p \wedge \neg q) \vee (\neg r \vee \neg s)$  NNF

4.  $(p \vee \neg r \vee \neg s) \wedge (\neg q \vee \neg r \vee \neg s)$

# An Introduction to Prolog Programming

## What is Prolog ?

- Prolog (*programming in logic*) is a logic-based programming language: programs correspond to sets of logical formulas and the Prolog interpreter uses logical methods to resolve queries.
- Prolog is a *declarative* language: you specify *what* problem you want to solve rather than *how* to solve it.
- Prolog is very useful in *some* problem areas, such as artificial intelligence, natural language processing, databases, . . . , but pretty useless in others, such as for instance graphics or numerical algorithms.
- The objective of this first lecture is to introduce you to the most basic concepts of the Prolog programming language.



**SWI Prolog**

<https://www.swi-prolog.org/>

# Prolog Programming

## What is Prolog ?

- It's declarative: very different from imperative style programming like Java, C++, Python,..
- A program is partly like a database but much more powerful since we can also have general rules to infer new facts!
- A Prolog interpreter can follow these facts/rules and answer queries by sophisticated search.
- Prolog is mainly suitable for programs involve symbolic or non-numeric computations.
- The following are very useful for writing a program in Prolog:
- **1. Knowledge Base, 2. Facts, 3. Rules, and 4. Queries.**

# Knowledge Base, Facts, Rules and Queries

- Facts, rules and queries are the main building blocks for a logical programming.
- We can form a Knowledge Base by collecting the facts and rules as a whole.
- **Knowledge Base** can be defined as collection of facts and rules.
- **Fact** is an explicit relationship between objects and properties. So, facts are unconditionally true. **Ex: Orange is a fruit**
- **Rule** is an implicit relationship between objects. So, rules are conditionally true.  
**Ex: John is happy if he drives car.**
- Queries are some questions on the relationships between objects and object properties. **Ex: Is orange a fruit? and Is John happy?**

# Facts & Queries

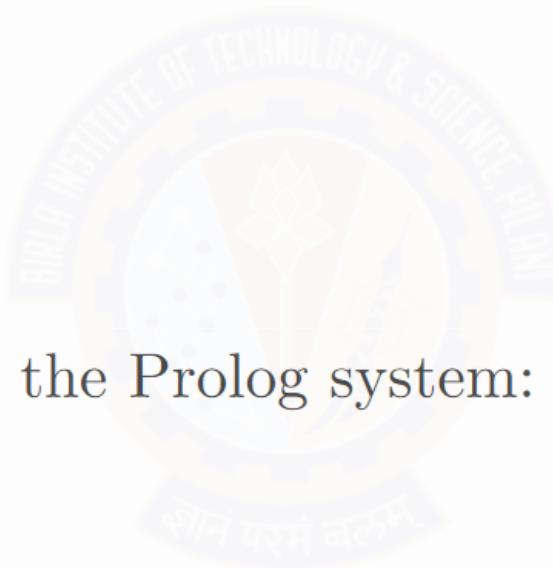
A little Prolog program consisting of four *facts*:

```
bigger(elephant, horse).
```

```
bigger(horse, donkey).
```

```
bigger(donkey, dog).
```

```
bigger(donkey, monkey).
```



After compilation we can *query* the Prolog system:

```
?- bigger(donkey, dog).
```

Yes

```
?- bigger(monkey, elephant).
```

No

# Issue?

The following query does not succeed!

```
?- bigger(elephant, monkey).
```

No

The *predicate* `bigger/2` apparently is not quite what we want.

What we'd really like is the transitive closure of `bigger/2`. In other words: a predicate that succeeds whenever it is possible to go from the first animal to the second by iterating the previously defined facts.

*How to solve this ? **Rules***

# Rules

The following two *rules* define `is_bigger/2` as the transitive closure of `bigger/2` (via recursion):

```
is_bigger(X, Y) :- bigger(X, Y).
```

```
is_bigger(X, Y) :- bigger(X, Z), is_bigger(Z, Y).
```

↑

“if”

↑

“and”

# Rules

```
?- is_bigger(elephant, monkey).
```

Yes

Even better, we can use the *variable X*:

```
?- is_bigger(X, donkey).
```

X = horse ;

X = elephant ;

No

Press ; (semicolon) to find alternative solutions. No at the end indicates that there are no further solutions.

# Terms

Prolog *terms* are either *numbers*, *atoms*, *variables*, or *compound terms*.

*Atoms* start with a lowercase letter or are enclosed in single quotes:

elephant, xYZ, a\_123, 'Another pint please'

*Variables* start with a capital letter or the underscore:

X, Elephant, \_G177, MyVariable, \_

*Compound terms* have a *functor* (an atom) and a number of *arguments* (terms):

is\_bigger(horse, X)  
f(g(Alpha, \_), 7)  
'My Functor'(dog)

Atoms and numbers are called *atomic terms*.

Atoms and compound terms are called *predicates*.

Terms without variables are called *ground terms*.

# Facts & Rules

*Facts* are predicates followed by a dot. Facts are used to define something as being unconditionally true.

```
bigger(elephant, horse).
```

```
parent(john, mary).
```

*Rules* consist of a *head* and a *body* separated by `:-`. The head of a rule is true if all predicates in the body can be proved to be true.

```
grandfather(X, Y) :-
```

```
    father(X, Z),
```

```
    parent(Z, Y).
```

# Programs and Queries

*Programs:* Facts and rules are called *clauses*. A Prolog program is a list of clauses.

*Queries* are predicates (or sequences of predicates) followed by a dot. They are typed in at the Prolog prompt and cause the system to reply.

```
?- is_bigger(horse, X), is_bigger(X, dog).
```

```
X = donkey
```

```
Yes
```

# Answering Queries

Answering a query means proving that the goal represented by that query can be satisfied (according to the programs currently in memory).

*Recall:* Programs are lists of facts and rules. A fact declares something as being true. A rule states conditions for a statement being true.

## Steps:

- If a goal matches with a *fact*, then it is satisfied.
- If a goal matches the *head of a rule*, then it is satisfied if the goal represented by the rule's body is satisfied.
- If a goal consists of several *subgoals* separated by commas, then it is satisfied if all its subgoals are satisfied.

# Example: Mortal Philosophers

Consider the following argument:

All men are mortal.

Socrates is a man.

---

Hence, Socrates is mortal.

It has two *premises* and a *conclusion*.

# Translating it into Prolog

The two premises can be expressed as a little Prolog program:

```
mortal(X) :- man(X).  
man(socrates).
```

The conclusion can then be formulated as a query:

```
?- mortal(socrates).
```

Yes

# Goal Execution

- (1) The query `mortal(socrates)` is made the initial goal.
- (2) Prolog looks for the first matching fact or head of rule and finds `mortal(X)`. Variable instantiation: `X = socrates`.
- (3) This variable instantiation is extended to the rule's body, i.e. `man(X)` becomes `man(socrates)`.
- (4) New goal: `man(socrates)`.
- (5) Success, because `man(socrates)` is a fact itself.
- (6) Therefore, also the initial goal succeeds.

# Simple Prolog Program

We can ask **queries** after loading our program:

?-male(albert).

Yes. %the above was true

?-male(victoria).

No. %the above was false

?-male(mycat).

No.

?-male(X). %X is a **variable**, we are asking “**who** is male?”

X=albert; %right! now type **semicolon** to ask for more answers.

X=edward;

No %no more answers

?-father(F,C). %F & C are variables, we are asking “**who** is father of **whom**”

F=albert, C=edward; %this is awesome! How did Prolog get this?

No

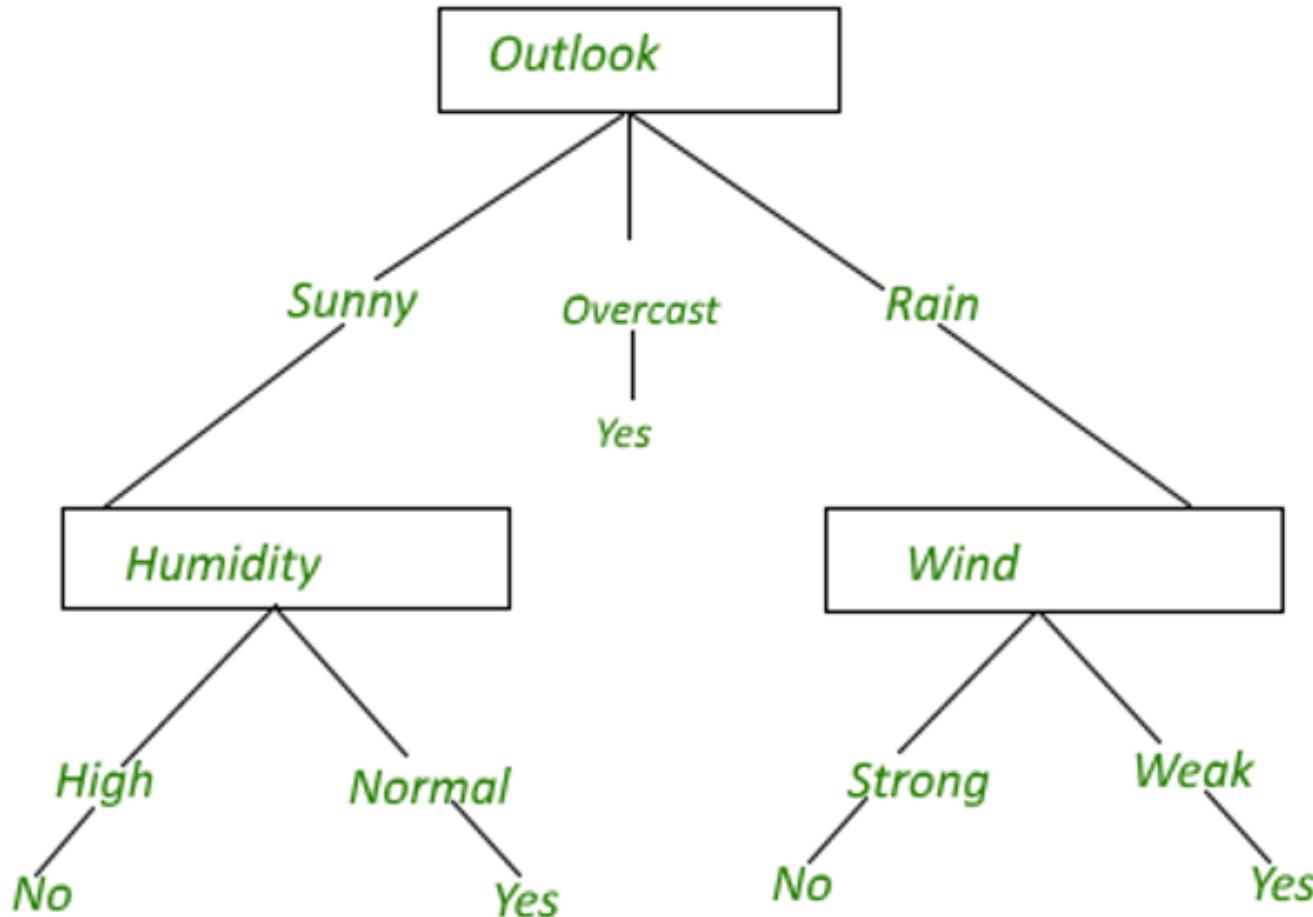
```
male(albert).    %this is our family.pl program
male(edward).
female(alice).
female(victoria).
parent(albert,edward).
parent(victoria,edward).
father(X,Y):- parent(X,Y), male(X).
mother(X,Y):- parent(X,Y), female(X).
```

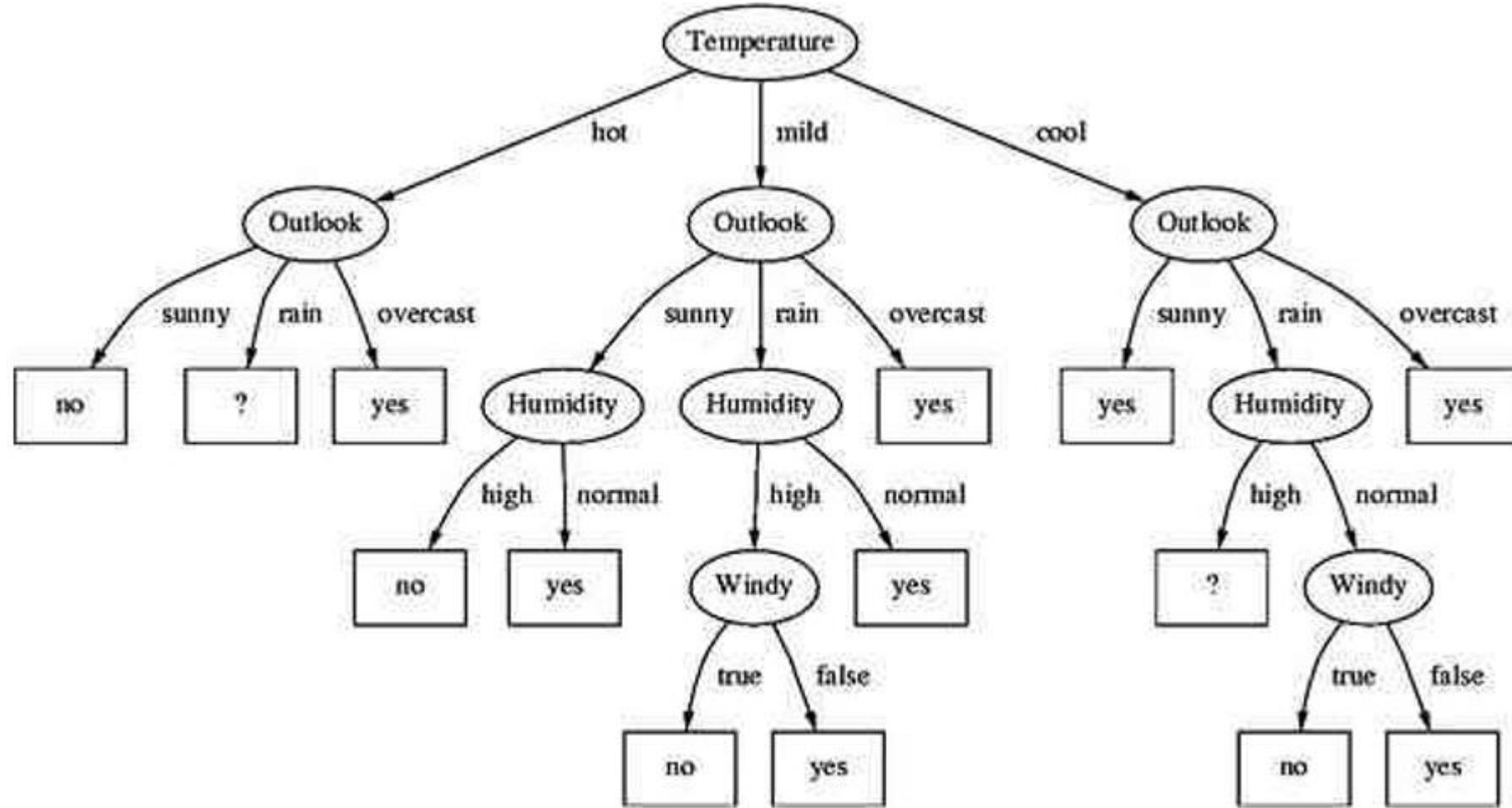
## Decision Tree to rules:

Day	<i>Outlook</i>	<i>Temperature</i>	<i>Humidity</i>	<i>Wind</i>	<i>PlayTennis</i>
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

# Decision Tree to rules:

Example:





- Lab Session 4



# Pomegranate – Bayesian Network

**BITS** Pilani

Pilani Campus

# Bayesian Networks

Bayesian networks are a probabilistic model for inference given incomplete data.

They consist of

- A directed graphical model and
- A set of probability distributions.

The edges encode dependency statements between the variables

Each node encodes a probability distribution.

Root nodes encode univariate probability distributions and inner/leaf nodes encode conditional probability distributions.

# Bayesian Networks

---

The Bayesian network has two main components: the **causal component** and the **numerical component**.

The causal component represents the **causal relationships** between the variables in the system.

The numerical component provides the actual **probabilities** that are used to make predictions and to calculate probabilities.

Currently, pomegranate only supports discrete Bayesian networks.

# API Reference

---

## ***Class pomegranate***

pomegranate is a Python package that implements fast and flexible probabilistic models ranging from individual probability distributions to compositional models such as Bayesian networks and hidden Markov models.

A Bayesian network is a directed graph where

- Nodes represent variables,
- Edges represent conditional dependencies of the children on their parents, and
- The lack of an edge represents a conditional independence.

# A Bayesian Network Model.

---

```
class Pomegranate.BayesianNetwork.BayesianNetwork
```

## **Parameters:**

`name: str, optional - The name of the model. (Default None)`

## **Attributes:**

`states:- A list of all states`

`graph:- The underlying graph object.`

# Bayesian Network Model - Methods



Method	Meaning
add_edge()	Add a transition from state A to state B which indicates that B is dependent on A
add_transition()	Transitions and edges are the same.
add_node()	Add a node to the graph.
add_nodes()	Add multiple states to the graph.
add_state()	Another name for a node.
add_states()	Another name for a node.

# A Bayesian Network Model - Methods



## bake()

- Finalize the topology of the model.
- Assign a **numerical index to every state** and create the underlying arrays corresponding to the states and edges between the states.
- This method must be **called before any of the probability-calculating methods**.
- This includes **converting conditional probability tables into joint probability tables** and creating a list of both marginal and table nodes.

# A Bayesian Network Model - Methods

---



## fit()

- Fit the model to data using MLE estimates.
- Fit the model to the data by updating each of the components of the model, which are univariate or multivariate distributions.

**Returns:** self:BayesianNetwork

- The fit Bayesian network object with updated model parameters.



# **Network learning – using samples, structure, summaries**

# Bayesian Network Model – Methods – Learning the structure

---

## **from\_samples()**

- Learn the structure of the network from data.

### **Parameters**

X: The data to fit the structure to, where each row is a sample and each column corresponds to the associated variable.

### **weights: ( optional)**

- The weight of each sample as a positive double.  
Default is None.

**max\_parents:int, optional, Default is -1 ->no max parents**

- The maximum number of parents a node can have

# A Bayesian Network Model – Learning the structure



## `from_samples()` method - Parameters

**algorithm:** str, one of ‘chow-liu’, ‘greedy’, ‘exact’, ‘exact-dp’ optional

- The algorithm to use for learning the Bayesian network.
- Default is ‘greedy’ that greedily attempts to find the best structure.

**include\_edges:** list or None, optional

- A list of (parent, child) tuples (edges) that must be present in the found structure.

**exclude\_edges:** list or None, optional

- A list of (parent, child) tuples (edges) which cannot be present in the found structure.

**Returns:** model:BayesianNetwork

- The learned BayesianNetwork.

# A Bayesian Network Model - Methods – Learning the structure

## **from\_structure()**

- Return a Bayesian network from a predefined structure.
- Passed as a tuple of tuples
- The tuple should contain n tuples, one for each node in the graph.
- Each inner tuple is the parent for that node.
- For example, a three node graph where both node 0 and 1 have node 2 as a parent would be specified as ((2,), (2,), ()).

### **Parameters:**

X:The data to fit the structure to, where each row is a sample and each column corresponds to the associated variable.

structure:tuple of tuples

- The parents for each node in the graph. If a node has no parents, then do not specify any parents.

# A Bayesian Network Model - Methods – Learning the structure

---

## **from\_summaries()**

- Use MLE on the stored sufficient statistics to train the model.

### **Parameters:**

Inertia : double, optional, Default is 0.0

- The inertia for updating the distributions, passed along to the distribution method.

Pseudocount : double, optional, Default is 0

- A pseudo-count to add to the emission of each distribution.
- This effectively smoothes the states to prevent 0 probability.

**Returns:** None

# A Bayesian Network Model – Methods - Prediction

---



## **predict()** method

- Predict missing values of a data matrix using MLE.
- Run each sample through the algorithm (`predict_proba`) and replace missing values with the maximally likely predicted emission.

**Returns:** The data matrix with the missing values imputed.

## **marginal()**

- Return the marginal probabilities of each variable in the graph.
- This will calculate the probability of each variable when nothing is known.

# A Bayesian Network Model – Methods - Prediction

---



## **predict\_proba()** method

- Returns the probabilities of each variable in the graph given evidence.
- This calculates the marginal probability distributions for each state given the evidence.

## **probability()**

- Return the probability of the given symbol under this distribution.

# Monty Hall Problem

---

The Monty Hall problem arose from the game show "**Let's Make a Deal**", where a guest had to choose one of three doors which had a prize behind it.

The twist was that after the guest chose, the host, originally Monty Hall, would then open one of the doors the guest did not pick and ask if the guest wanted to switch which door they had picked.

Guess the chance of the guest switching the doors or not and whether the guest gets the car.

# Initialization

---

Bayesian networks can be initialized in two ways, depending on whether the underlying graphical structure is known or not.

For Bayesian networks, fitting can be done quickly by just summing counts through the data.

For each variable, only that **column of data** (node) and the columns corresponding to that variables **parents** are considered.

# Bayesian Net for Monty Hall Problem



The network for this problem will have three nodes:

- one for the guest,
- one for the prize, and
- one for the door Monty chooses to open.

The door the guest initially chooses and the door the prize is behind are uniform random processes across the three doors.

The door which Monty opens is dependent on both the door the guest chooses (it cannot be the door the guest chooses), and the door the prize is behind (it cannot be the door with the prize behind it).

# Conditional distribution

---

The conditional distribution must be explicitly spelled out in this example

Followed by a list of the parents in the same order as the columns take in the table that is provided

In our examples, the columns in the table correspond to guest, prize, monty.

Here, A, B, C denotes the door picked by the guest, prize containing door, and Monty's door.

# PYTHON CODE – Initialization

---

```
from pomegranate import *
# The guests initial door selection is
# completely random
guest = DiscreteDistribution({'A': 1./3, 'B':
    1./3, 'C': 1./3})
# The door the prize is behind is also
# completely random
prize = DiscreteDistribution({'A': 1./3, 'B':
    1./3, 'C': 1./3})
# Monty is dependent on both the guest and the
# prize.
```

# PYTHON CODE - Initialization

```
monty =  
    ConditionalProbabilityTable  
([[['A', 'A', 'A', 0.0],  
     ['A', 'A', 'B', 0.5],  
     ['A', 'A', 'C', 0.5],  
     ['A', 'B', 'A', 0.0],  
     ['A', 'B', 'B', 0.0],  
     ['A', 'B', 'C', 1.0],  
     ['A', 'C', 'A', 0.0],  
     ['A', 'C', 'B', 1.0],  
     ['A', 'C', 'C', 0.0],  
     ['B', 'A', 'A', 0.0],  
     ['B', 'A', 'B', 0.0],  
     ['B', 'A', 'C', 1.0],  
     ['B', 'B', 'A', 0.5],  
     ['B', 'C', 'A', 1.0],  
     ['B', 'C', 'B', 0.0],  
     ['B', 'C', 'C', 0.0],  
     ['C', 'A', 'A', 0.0],  
     ['C', 'A', 'B', 1.0],  
     ['C', 'A', 'C', 0.0],  
     ['C', 'B', 'A', 1.0],  
     ['C', 'B', 'B', 0.0],  
     ['C', 'B', 'C', 0.0],  
     ['C', 'C', 'A', 0.5],  
     ['C', 'C', 'B', 0.5],  
     ['C', 'C', 'C', 0.0]],  
     [guest, prize])
```

# PYTHON CODE - Initialization

---

Next, we pass these distributions into state objects along with the name for the node.

```
s1 = Node(guest, name="guest")
s2 = Node(prize, name="prize")
s3 = Node(monty, name="monty")
```

# PYTHON CODE - Initialization

---

Then we add the states to the network.

```
# Create the Bayesian network object with a useful name  
model = BayesianNetwork("Monty Hall Problem")  
# Next Add the three states to the network  
model.add_states(s1, s2, s3)
```

# PYTHON CODE - Initialization

---

Then we need to add edges to the model.

The edges represent which states are parents of which other states.

Edges are added from parent -> child by calling `model.add_edge(parent, child)`.

# Monty is dependent on both guest and prize

```
model.add_edge(s1, s3)
```

```
model.add_edge(s2, s3)
```

# PYTHON CODE - Initialization

---

Lastly, the model must be baked to finalize the internals.

bake() method produces a factor graph which is a probabilistic graph

**model.bake()**

# Conditional distribution – From samples

---

The conditional distribution can also be initialized based completely on data.

```
from pomegranate import *
import numpy
X = numpy.load('data.npy')
model = BayesianNetwork.from_samples(X,
    algorithm='exact')
```

Note: Not related to Monty problem

# Predicting Probabilities using Probability0 method

You can calculate the probability of a sample under a Bayesian network as the product of the probability of each variable given its parents, if it has any.

This can be expressed as

$$P = \prod_{i=1}^d P(D_i|Pa_i)$$

for a sample with d dimensions.

For example, in the Monty Hall problem, the probability of a show is the probability of the guest choosing the respective door, times the probability of the prize being behind a given door, times the probability of Monty opening a given door given the previous two values.

# Predicting Probabilities using Probability0 method

For example, using the manually initialized network above, the following statement using the probability() method will give the probabilities of the events as follows:

```
print(model.probability([[['A', 'A', 'A'],
                           ['A', 'A', 'B'],
                           ['C', 'C', 'B']]]))
```

Output: [ 0. **0.05555556** **0.05555556** ]

# Predicting Missing values using predict0 method

Bayesian networks are frequently used to infer/impute the value of missing variables given the observed values.

In the case of Bayesian networks, when data is passed in for prediction it should be in the format as a matrix with **None** in the missing variables that need to be inferred.

The return is thus a filled in matrix where the Nones have been replaced with the imputed values. For example:

```
print(model.predict([[ 'A', 'B', None],  
                     [ 'A', 'C', None],  
                     [ 'C', 'B', None]]))
```

Output: [[ 'A' 'B' 'C']  
 [ 'A' 'C' 'B']  
 [ 'C' 'B' 'A']]

# Predicting Probabilities – Some Examples

---

We can calculate probabilities of a sample under the Bayesian network in the same way that we can calculate probabilities under other models.

In this case, let's calculate the probability that the guest initially chooses door A, Monty then opened door B, but the car was behind door C.

```
model.probability([[ 'A', 'B', 'C']])
```

Output: 0.1111111111111109

# Predicting Probabilities – Some Examples

- Next, let's look at an impossible situation. What is the probability of initially guest choosing door A, Monty chooses door B, and that the car was actually behind door B.
- `model.probability([[ 'A', 'B', 'B']])`
- Output: `0.0`
- The reason that situation is impossible is because Monty can't open a door that has the car behind it.

# Prediction VS Inference

---

**Prediction** is the process of using a model to make a prediction about something that is yet to happen.

**Inference** is the process of evaluating the relationship between the predictor and response variables.

## **Examples for prediction and inference**

- **Prediction:** You want to predict future ozone levels using historic data.
- **Inference:** You want to understand how ozone levels are influenced by temperature, solar radiation, and wind.

# Performing Inference

---

We can run inference using

the `predict_proba()` method and passing in a dictionary of values, where the key is the name of the state and the value is the observed value for that state.

If we don't supply any values, we get the **marginal of the graph**, which is just the frequency of each value for each variable over an infinite number of randomly drawn samples from the graph.

# Performing Inference – Marginal probabilities

---

Lets see what happens when we look at the marginal of the Monty hall network.

```
model.predict_proba({}) #Refer Run1
```

We were returned three DiscreteDistribution objects, each representing the marginal distribution for each variable, in the same order they were put into the model.

In this case, they represent the guest, prize, and monty variables respectively.

We see that everything is equally likely.

We can also pass in an array where None (or np.nan) correspond to the unobserved values.

# Performing Inference

Now lets do something different, and say that the guest has chosen door 'A'.

We do this by passing a **list** to predict\_proba() method

```
model.predict_proba([[ 'A', None, None]]) #Run2
```

We can see that now Monty will not open door 'A', because the guest has chosen it.

At the same time, the distribution over the prize has not changed, it is still equally likely that the prize is behind each door.

Unknown values can be predicted using the predict\_proba() method as follows:

```
print(model.predict([[ 'A', 'B', None], ['A', 'C', None], ['C', 'B', None]])) #Run3
```

# Performing Inference

---

Now, lets say that Monty opens door 'C' and see what happens.

Here we use a **dictionary** rather than a list simply to show how one can use both input forms depending on what is more convenient.

```
model.predict_proba([{'guest': 'A', 'monty':  
'C'}]) #Run4
```

We see that the distribution over prizes has changed. It is now twice as likely that the car is behind the door labeled 'B'.

This demonstrates that when on the game show, it is always better to change your initial guess after being shown an open door.

# References for further reading

- <https://pomegranate.readthedocs.io/en/latest/BayesianNetwork.html>
- <https://pomegranate.readthedocs.io/en/latest/faq.html>
- [https://notebook.community/jmschrei/pomegranate/tutorials/B\\_Model\\_Tutorial\\_4\\_Bayesian\\_Networks](https://notebook.community/jmschrei/pomegranate/tutorials/B_Model_Tutorial_4_Bayesian_Networks)
- <https://www.analyticsvidhya.com/blog/2020/08/probability-conditional-probability-monty-hall-problem/>
- <https://www.mygreatlearning.com/blog/bayesian-network/>
- <https://www.edureka.co/blog/bayesian-networks/>