

Artificial Intelligence and Soft Computing

(Code : CSC703)

**Semester VII : Computer Engineering
(Mumbai University)**

**Strictly as Per the Choice Based Credit and Grading System (Revise 2016)
of Mumbai University w.e.f academic Year 2019-2020**

Prof. Purva Raut

M. Tech. (Computer Engineering)
Assistant Professor,
Department of Information Technology,
D.J. Sanghvi College of Engineering, Mumbai,
Maharashtra, India

Dipali Y. Koshti

Assistant Professor,
Department of Computer Engineering
Fr. Conceicao Rodrigues College of
Engg.(Fr.CRCE), Mumbai.
Maharashtra, India.



Artificial Intelligence and Soft Computing

Prof. Purva Raut, Dipali Y. Koshti

(Semester VII : Computer Engineering) (Mumbai University))

Copyright © by Authors. All rights reserved. No part of this publication may be reproduced, copied, or stored in a retrieval system, distributed or transmitted in any form or by any means, including photocopy, recording, or other electronic or mechanical methods, without the prior written permission of the publisher.

This book is sold subject to the condition that it shall not, by the way of trade or otherwise, be lent, resold, hired out, or otherwise circulated without the publisher's prior written consent in any form of binding or cover other than which it is published and without a similar condition including this condition being imposed on the subsequent purchaser and without limiting the rights under copyright reserved above.

First Edition : July 2019 (TechKnowledge Publications)

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

ISBN 978-93-89424-35-5

Published by

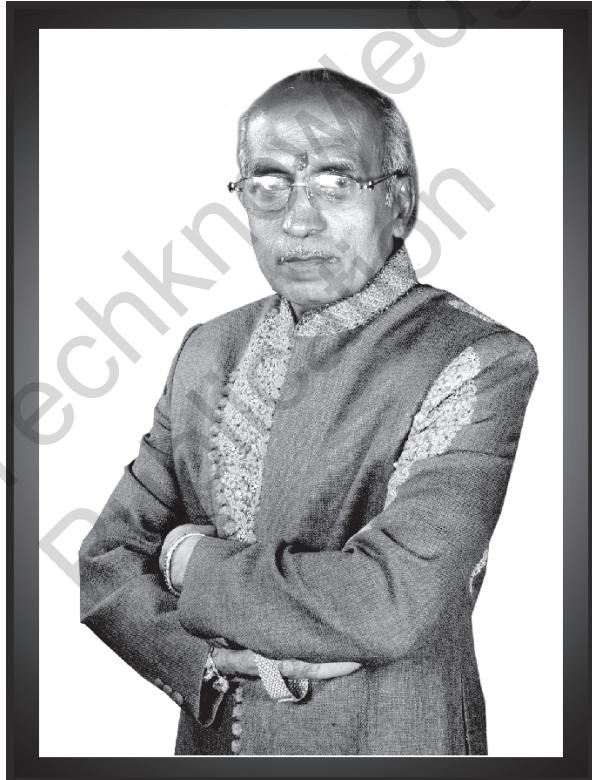
TechKnowledge Publications

Head Office : B/5, First floor, Maniratna Complex,
Taware Colony, Aranyeshwar Corner,
Pune - 411 009. Maharashtra State, India
Ph : 91-20-24221234, 91-20-24225678.

[CSC703] (FID : MO82) (Book Code : MO82A)

(Book Code : MO82A)

*We dedicate this Publication soulfully and wholeheartedly,
in loving memory of our beloved founder director
Late. Shri. Pradeepsheth Lalchandji Lunawat,
who will always be an inspiration, a positive force and strong support
behind us.*



Lt. Shri. Pradeepji L. Lunawat

*Soulful Tribute and Gratitude for all Your
Sacrifices, Hardwork and 40 years of Strong Vision.....*

Techknowledge
Publication

Preface

Dear Students,

We are extremely happy to present this book to you. We have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow and understanding of the subject.

We present this book in the loving memory of **Late Shri. Pradeepji Lunawat**, our source of inspiration and a strong foundation of "**TechKnowledge Publications**". He will always be remembered in our hearts and motivate us to achieve our new milestone.

We are thankful to Prof. Arunoday Kumar, Shri. Shital Bhandari & Shri. Chandroday Kumar for the encouragement and support that they have extended. We are also thankful to Seema Lunawat for ebooks and the staff members of TechKnowledge Publications and others for their efforts to make this book as good as it is. We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let us know, because that will help us to improve the book quality further.

We are also thankful to our family members and friends for their patience and encouragement.

- Authors



SYLLABUS

Course Code	Course Name	Credits
CSC703	Artificial Intelligence & Soft Computing	4

Course Objectives (CO):

1. To conceptualize the basic ideas and techniques of AI and SC.
2. To distinguish various search techniques and to make student understand knowledge representation and planning.
3. To become familiar with basics of Neural Networks and Fuzzy Logic.
4. To familiarize with Hybrid systems and to build expert system.

Course Outcomes : Students should be able to -

1. Identify the various characteristics of Artificial Intelligence and Soft Computing techniques.
2. Choose an appropriate problem solving method for an agent to find a sequence of actions to reach the goal state.
3. Analyse the strength and weakness of AI approaches to knowledge representation, reasoning and planning.
4. Construct supervised and unsupervised ANN for real world applications.
5. Design fuzzy controller system.
6. Apply Hybrid approach for expert system design.

Pre-requisites : Basic Mathematics, Algorithms

Module No.	Unit No.	Topics	Hrs.
1.0	1.1	Introduction to Artificial Intelligence(AI) and Soft Computing Introduction and Definition of Artificial Intelligence.	4
	1.2	Intelligent Agents : Agents and Environments ,Rationality, Nature of Environment, Structure of Agent, types of Agent	
	1.3	Soft Computing: Introduction of soft computing, soft computing vs. hard computing, various types of soft computing techniques. (Refer Chapter-1)	
2.0	2.1	Problem Solving Problem Solving Agent, Formulating Problems, Example Problems	10
	2.2	Uninformed Search Methods: Depth Limited Search, Depth First Iterative Deepening (DFID), Informed Search Method: A* Search	
	2.3	Optimization Problems: Hill climbing Search, Simulated annealing, Genetic algorithm (Refer Chapter-2)	

3.0	3.1	Knowledge, Reasoning and Planning Knowledge based agents	10
	3.2	First order logic: syntax and Semantic, Knowledge Engineering in FOL Inference in FOL : Unification, Forward Chaining, Backward Chaining and Resolution	
	3.3	Planning Agent, Types of Planning: Partial Order, Hierarchical Order, Conditional Order (Refer Chapter-3)	
4.0	4.1	Fuzzy Logic Introduction to Fuzzy Set: Fuzzy set theory, Fuzzy set versus crisp set, Crisp relation & fuzzy relations, membership functions,	12
	4.2	Fuzzy Logic: Fuzzy Logic basics, Fuzzy Rules and Fuzzy Reasoning	
	4.3	Fuzzy inference systems : Fuzzification of input variables, defuzzification and fuzzy controllers. (Refer Chapter-4)	
5.0	5.1	Artificial Neural Network Introduction – Fundamental concept– Basic Models of Artificial Neural Networks – Important Terminologies of ANNs – McCulloch-Pitts Neuron	12
	5.2	Neural Network Architecture: Perceptron, Single layer Feed Forward ANN, Multilayer Feed Forward ANN, Activation functions, Supervised Learning: Delta learning rule, Back Propagation algorithm.	
	5.3	Un-Supervised Learning algorithm: Self Organizing Maps (Refer Chapter-5)	
6.0	6.1	Expert System Hybrid Approach - Fuzzy Neural Systems	4
	6.2	Expert system : Introduction, Characteristics, Architecture, Stages in the development of expert system. (Refer Chapter-6)	
		Total	52

□□□

**Chapter 1 : Intro. to Artificial Intelligence(AI) and Soft Computing****1-1 to 1-33****Syllabus :**

Introduction and Definition of Artificial Intelligence. Intelligent Agents : Agents and Environments, Rationality, Nature of Environment, Structure of Agent, types of Agent. Soft Computing: Introduction of soft computing, soft computing vs. hard computing, various types of soft computing techniques.

1.1	Introduction to Artificial Intelligence	1-1
1.2	Foundations and Mathematical Treatments.....	1-1
1.2.1	Acting Humanly : The Turing Test Approach	1-1
1.2.2	Thinking Humanly : The Cognitive Modelling Approach.....	1-2
1.2.3	Thinking Rationally : The “Laws of Thought” Approach.....	1-3
1.2.4	Acting Rationally : The Rational Agent Approach	1-3
1.3	Categorization of Intelligent Systems.....	1-4
1.4	Components of AI.....	1-4
1.4.1	Computational Intelligence vs. Artificial Intelligence	1-6
1.5	History of Artificial Intelligence.....	1-6
1.6	Applications of Artificial Intelligence.....	1-7
1.7	Sub Areas/ Domains of Artificial Intelligence	1-8
1.8	Current Trends in Artificial Intelligence	1-9
1.9	Intelligent Agents.....	1-10
1.9.1	What is an Agent?.....	1-10
1.9.2	Definitions of Agent	1-12
1.9.3	Intelligent Agent.....	1-13
1.9.3(A)	Structure of Intelligent Agents	1-13
1.10	Rational Agent.....	1-14
1.11	Nature of Environment and PEAS Properties of Agent.....	1-15
1.11.1	Environments Types / Nature of Environment / Task Environment Properties	1-15
1.11.2	PEAS Properties of Agent	1-19
1.12	Structure of Agents / Types of Agents	1-21
1.12.1	Simple Reflex Agents	1-21
1.12.2	Model-Based Reflex Agents	1-23
1.12.3	Goal-Based Agents	1-24
1.12.4	Utility-Based Agents	1-25

1.12.5	Learning Agents	1-25
1.13	Introduction to Soft Computing	1-26
1.14	Soft Computing vs. Hard Computing	1-27
1.15	Various Types of Soft Computing Techniques.....	1-28
1.15.1	Introduction to Neural Networks	1-28
1.15.2	Introduction to Fuzzy Logic.....	1-30
1.15.3	Introduction to Genetic Algorithms.....	1-31

Chapter 2 : Problem Solving**2-1 to 2-61****Syllabus :**

Problem Solving Agent, Formulating Problems, Example Problems Uninformed Search Methods : Depth Limited Search, Depth First Iterative Deepening (DFID), Informed Search Method : A* Search Optimization Problems : Hill climbing Search, Simulated annealing, Genetic algorithm

2.1	Solving Problems by Searching.....	2-1
2.2	Formulating Problems	2-1
2.2.1	Components of Problems Formulation	2-2
2.2.2	Example Problems	2-2
2.3	Measuring Performance of Problem Solving Algorithm / Agent.....	2-5
2.4	Node Representation in Search Tree	2-5
2.5	Uninformed Search	2-5
2.6	Depth First Search (DFS)	2-6
2.6.1	Concept.....	2-6
2.6.2	Implementation.....	2-7
2.6.3	Algorithm.....	2-7
2.6.4	Performance Evaluation	2-7
2.7	Breadth First Search (BFS)	2-8
2.7.1	Concept.....	2-8
2.7.2	Process	2-8
2.7.3	Implementation.....	2-8
2.7.4	Algorithm	2-8
2.7.5	Performance Evaluation	2-9
2.8	Uniform Cost Search (UCS)	2-9
2.8.1	Concept.....	2-9
2.8.2	Implementation.....	2-9
2.8.3	Algorithm	2-9
2.8.4	Performance Evaluation	2-10
2.9	Depth Limited Search (DLS).....	2-10



2.9.1	Concept.....	2-10	2.16.4	Behaviour of A* Algorithm	2-26
2.9.2	Process	2-10	2.16.5	Admissibility of A*.....	2-27
2.9.3	Implementation.....	2-10	2.16.6	Monotonicity	2-28
2.9.4	Algorithm	2-10	2.16.7	Properties of A*	2-28
2.9.5	Pseudo Code.....	2-11	2.16.8	Example : 8 Puzzle Problem using A* Algorithm	2-29
2.9.6	Performance Evaluation	2-11	2.16.9	Caparison among Best First Search, A* search and Greedy Best First Search	2-31
2.10	Depth First Iterative Deepening (DFID).....	2-11	2.17	Local Search Algorithms and Optimization Problems	2-32
2.10.1	Concept.....	2-11	2.17.1	Hill Climbing	2-32
2.10.2	Process	2-13	2.17.1(A)	Simple Hill Climbing	2-33
2.10.3	Implementation.....	2-13	2.17.1(B)	Steepest Ascent Hill Climbing	2-34
2.10.4	Algorithm	2-13	2.17.1(C)	Limitations of Hill Climbing	2-34
2.10.5	Pseudo Code.....	2-13	2.17.1(D)	Solutions on Problems in Hill Climbing	2-36
2.10.6	Performance Evaluation	2-14	2.17.2	Simulated Annealing	2-36
2.11	Bidirectional Search	2-14	2.17.2(A)	Comparing Simulated Annealing with Hill Climbing ...	2-37
2.11.1	Concept.....	2-14	2.17.3	Local Beam Search.....	2-38
2.11.2	Process	2-14	2.17.4	Genetic Algorithms.....	2-40
2.11.3	Implementation.....	2-14	2.17.4(A)	Terminologies of GA.....	2-41
2.11.4	Performance Evaluation	2-15	2.17.4(B)	Genetic Operators	2-41
2.11.5	Pros of Bidirectional Search	2-15	2.17.4(C)	The Basic Genetic Algorithm	2-42
2.11.6	Cons of Bidirectional Search	2-15	2.17.4(D)	Example of Genetic Algorithm	2-42
2.12	Comparing Different Techniques	2-15	2.18	Adversarial Search	2-43
2.12.1	Difference between Unidirectional and Bidirectional Search.....	2-16	2.18.1	Environment Types	2-43
2.12.2	Difference between BFS and DFS.....	2-16	2.18.2	AI Game – Features.....	2-43
2.13	Informed Search Techniques.....	2-17	2.18.2(A)	Zero Sum Game.....	2-44
2.14	Heuristic Function.....	2-18	2.18.2(B)	Non-Zero Sum Game	2-44
2.14.1	Example of 8-puzzle Problem.....	2-19	2.18.2(C)	Positive Sum Game	2-44
2.14.2	Example of Block World Problem	2-19	2.18.2(D)	Negative Sum Game	2-44
2.14.3	Properties of Good Heuristic Function	2-21	2.19	Relevant Aspects of AI Game	2-45
2.15	Best First Search.....	2-22	2.20	Game Playing.....	2-45
2.15.1	Concept.....	2-22	2.20.1	Type of Game Strategies.....	2-46
2.15.2	Implementation.....	2-22	2.20.2	Type of Games.....	2-46
2.15.3	Algorithm : Best First Search.....	2-22	2.20.2(A)	Chess	2-47
2.15.4	Performance Measures for Best first search.....	2-23	2.20.2(B)	Checkers	2-48
2.15.5	Greedy Best First Search	2-23	2.20.3	What is Game Tree?	2-49
2.15.6	Properties of Greedy Best-First Search	2-24	2.21	Minimax Algorithm	2-50
2.16	A* Search.....	2-24	2.21.1	Minimax Algorithm.....	2-50
2.16.1	Concept.....	2-24	2.21.2	Properties of Minimax Algorithm.....	2-54
2.16.2	Implementation.....	2-25	2.22	Alpha Beta Pruning	2-54
2.16.3	Algorithm (A*).....	2-25			



2.22.1	Example of $\alpha\beta$ Pruning.....	2-56
2.22.3	Properties of $\alpha\beta$	2-60

**Chapter 3 : Knowledge, Reasoning
and Planning** **3-1 to 3-60**

Syllabus :

Knowledge based agents, First order logic: syntax and Semantic, Knowledge Engineering in FOL Inference in FOL : Unification, Forward Chaining, Backward Chaining and Resolution, Planning Agent, Types of Planning: Partial Order, Hierarchical Order, Conditional Order

3.1	A Knowledge Based Agent.....	3-1
3.1.1	Architecture of a KB Agent	3-2
3.2	The WUMPUS World Environment.....	3-3
3.2.1	Description of the WUMPUS World.....	3-4
3.2.2	PEAS Properties of WUMPUS World.....	3-5
3.2.3	Exploring a WUMPUS World.....	3-6
3.3	Logic	3-8
3.3.1	Role of Reasoning in AI.....	3-9
3.4	Representation of Knowledge using Rules	3-9
3.4.1	Ontology.....	3-12
3.5	Propositional Logic (PL)	3-12
3.5.1	Syntax	3-12
3.5.2	Semantics	3-13
3.5.3	What is Propositional Logic ?	3-14
3.5.4	PL Sentence - Example.....	3-14
3.5.5	Inference Rules	3-15
3.5.6	Horn Clause	3-16
3.5.7	Propositional Theorem Proving	3-17
3.5.8	Advantages of Propositional Logic.....	3-17
3.5.9	Disadvantages of Propositional Logic.....	3-17
3.6	First Order Predicate Logic.....	3-18
3.6.1	Syntactic Elements, Semantic and Syntax	3-18
3.7	Comparison between Propositional Logic and First Order Logic	3-19
3.8	Inference in FOL.....	3-20
3.8.1	Forward Chaining	3-20
3.8.2	Backward Chaining.....	3-21
3.8.3	Differentiate between Forward Chaining and Backward Chaining.....	3-22

3.9	Knowledge Engineering in First Order Logic.....	3-24
3.9.1	Knowledge Engineering Process	3-24
3.10	Unification and Lifting	3-24
3.10.1	Unification	3-24
3.10.2	Lifting	3-25
3.11	Resolution	3-27
3.11.1	The Resolution Procedure	3-27
3.11.2	Conversion from FOL Clausal Normal Form (CNF) ...	3-27
3.11.3	Facts Representation	3-28
3.11.4	Example	3-29
3.12	Planning	3-36
3.12.1	Introduction to Planning.....	3-36
3.12.2	Simple Planning Agent	3-37
3.13	Planning Problem	3-37
3.13.1	Problem Solving and Planning.....	3-38
3.14	Goal of Planning.....	3-38
3.14.1	Major Approaches	3-39
3.15	Planning Graphs.....	3-40
3.16	Planning as State-Space Search.....	3-41
3.16.1	Example of State Space Search.....	3-43
3.17	Classification of Planning with State Space Search	3-45
3.18	Progression Planners	3-45
3.19	Regression Planners	3-46
3.19.1	Heuristics for State-Space Search.....	3-47
3.20	Total Order Planning (TOP).....	3-47
3.21	Partial Order Planning	3-48
3.21.1	POP as a Search Problem	3-48
3.21.2	Consistent Plan is a Solution for POP Problem	3-50
3.22	Hierarchical Planning.....	3-50
3.22.1	POP One Level Planner	3-50
3.22.2	Hierarchy of Actions	3-51
3.22.3	Planner.....	3-51
3.23	Planning Languages.....	3-53
3.23.1	Example of Block World Puzzle.....	3-54
3.23.2	Example of the Spare Tire Problem.....	3-56
3.24	Planning and Acting in the Real World	3-56
3.25	Multi-Agent Planning	3-57
3.26	Conditional Planning.....	3-58

**Chapter 4 : Fuzzy Logic 4-1 to 4-89****Syllabus :**

Introduction to Fuzzy Set: Fuzzy set theory, Fuzzy set versus crisp set, Crisp relation & fuzzy relations, membership functions, Fuzzy Logic: Fuzzy Logic basics, Fuzzy Rules and Fuzzy Reasoning, Fuzzy inference systems: Fuzzification of input variables, defuzzification and fuzzy controllers.

4.1	Introduction to Fuzzy Set.....	4-1
4.2	Fuzzy Set vs. Crisp Set	4-1
4.3	Fuzzy Set Theory	4-3
4.3.1	Fuzzy Set : Definition.....	4-3
4.3.2	Types of Universe of Discourse.....	4-3
4.3.3	Different Notations for Representing Fuzzy Sets.....	4-5
4.3.4	Linguistic Variables and Linguistic Values	4-6
4.3.5	Important Terminologies related to Fuzzy Sets.....	4-7
4.3.6	Properties of Fuzzy Sets	4-10
4.3.7	Operations on Fuzzy Sets	4-11
4.4	Crisp Relation and Fuzzy Relations.....	4-13
4.4.1	Crisp Relation	4-13
4.4.1(A)	Cardinality of Classical Relation	4-14
4.4.1(B)	Operations on Classical Relations.....	4-14
4.4.1(C)	Properties of Crisp (Classical) Relations	4-15
4.4.1(D)	Composition of Crisp Relations	4-15
4.4.2	Fuzzy Relations.....	4-16
4.4.2(A)	Operations on Fuzzy Relation.....	4-17
4.4.2(B)	Properties of Fuzzy Relations.....	4-20
4.4.2(C)	Fuzzy Composition.....	4-20
4.5	Membership Functions	4-21
4.6	Fuzzy Logic	4-24
4.6.1	Fuzzy Logic Basics.....	4-24
4.6.2	Fuzzy Rules and Fuzzy Reasoning.....	4-25
4.7	Fuzzy Inference Systems	4-28
4.7.1	Construction and Working Principle of FIS.....	4-28
4.7.2	Fuzzification of Input Variables.....	4-33
4.7.3	Defuzzification.....	4-36

4.8	Fuzzy Controllers	4-40
4.8.1	Steps in Designing FLC.....	4-42
4.8.2	Advantages of FLSs.....	4-43
4.8.3	Disadvantages of FLSs	4-43
4.9	Solved Problems	4-43
4.10	Design of Controllers (Solved Problems)	4-66

Chapter 5 : Artificial Neural Network 5-1 to 5-60**Syllabus :**

Introduction : Fundamental concept : Basic Models of Artificial Neural Networks : Important Terminologies of ANNs : McCulloch-Pitts Neuron, Neural Network Architecture: Perceptron, Single layer Feed Forward ANN, Multilayer Feed Forward ANN, Activation functions, Supervised Learning : Delta learning rule, Back Propagation algorithm., Un-Supervised Learning algorithm : Self Organizing Maps

5.1	Fundamental Concept - Artificial Neural Networks.....	5-1
5.1.1	Artificial Neural Networks	5-1
5.1.2	Biological Neural Networks.....	5-2
5.1.3	Brain vs. Computer.....	5-4
5.1.4	Features of ANN.....	5-5
5.1.5	Disadvantages of ANN	5-5
5.1.6	Applications of ANN	5-5
5.2	Basic Models of Artificial Neural Networks	5-7
5.2.1	Connections	5-7
5.2.2	Learning	5-7
5.2.2(A)	Supervised Learning.....	5-7
5.2.2(B)	Unsupervised Learning.....	5-8
5.2.2(C)	Reinforced Learning	5-9
5.2.3	Activation Functions	5-10
5.3	Important Terminologies of ANN	5-13
5.3.1	Weights	5-13
5.3.2	Bias	5-14
5.3.3	Threshold	5-15
5.3.4	Learning Constant	5-15
5.3.5	Momentum Factor	5-15
5.3.6	Vigilance Parameter	5-16
5.4	McCulloch-Pitts Neuron Model	5-16
5.5	Neural Network Architecture	5-21



5.5.1	Single Layer Feed Forward Networks	5-21
5.5.2	Multilayer Feed Forward Networks	5-22
5.5.3	Single Layer Feedback Network	5-23
5.5.4	Multilayer Feedback Networks.....	5-24
5.6	Supervised Learning.....	5-24
5.6.1	Perceptron Learning Rule.....	5-24
5.6.1(A)	Rosenblatt's Perceptron	5-24
5.6.1(B)	The Perceptron Model.....	5-25
5.6.1(C)	Algorithm for Training a Single Perceptron Network	5-26
5.6.1(D)	Model for Multilayer Perceptron	5-27
5.6.1(E)	The EX-OR Problem and Need for Multi-layer Perceptron	5-27
5.6.1(F)	Linearly and Non-Linearly Separable Patterns.....	5-28
5.6.2	Delta Learning Rule.....	5-31
5.6.2(A)	Delta Learning Model	5-32
5.6.2(B)	Proofs.....	5-33
5.6.2(C)	Algorithm-Delta Learning.....	5-35
5.6.3	Back Propagation Algorithm	5-36
5.6.3(A)	Architecture of Back Propagation Network	5-37
5.6.3(B)	Algorithm (Error Back Propagation Training)	5-39
5.7	Un-Supervised Learning Algorithm : Self Organizing Maps	5-40
5.8	Solved Problems	5-42

Chapter 6 : Expert System**6-1 to 6-20****Syllabus :**

Hybrid Approach - Fuzzy Neural Systems Expert system : Introduction, Characteristics, Architecture, Stages in the development of expert system,

6.1	Hybrid Approach.....	6-1
6.1.1	Introduction to Hybrid Systems.....	6-1
6.1.2	Types of Hybrid Systems.....	6-1
6.1.3	Fuzzy Neural System	6-3
6.1.4	Adaptive Neuro - Fuzzy Inference System (ANFIS)	6-4
6.2	Expert System	6-6
6.2.1	Introduction	6-6
6.2.2	Characteristics of Expert Systems.....	6-7
6.2.3	Real Time Expert Systems	6-8
6.3	Building Blocks of Expert Systems	6-8
6.4	Development Phases of Expert Systems.....	6-11
6.5	Representing and Using Domain Knowledge.....	6-11
6.6	Expert System-Shell	6-15
6.7	Explanations.....	6-17
6.8	Knowledge Acquisition	6-17
6.8.1	Knowledge Elicitation	6-18
6.9	Expert System vs. Traditional System	6-19
6.10	The Applications of Expert Systems	6-19





Introduction to Artificial Intelligence(AI) and Soft Computing

Syllabus

- 1.1 Introduction and Definition of Artificial Intelligence.
- 1.2 Intelligent Agents : Agents and Environments ,Rationality, Nature of Environment, Structure of Agent, types of Agent
- 1.3 Soft Computing: Introduction of soft computing, soft computing vs. hard computing, various types of soft computing techniques.

1.1 Introduction to Artificial Intelligence

- John McCarthy who has coined the word “Artificial Intelligence” in 1956, has defined AI as “the science and engineering of making intelligent machines”, especially intelligent computer programs.
- **Artificial Intelligence (AI)** is relevant to any intellectual task where the machine needs to take some decision or choose the next action based on the current state of the system, in short act intelligently or rationally. As it has a very wide range of applications, it is truly a universal field.
- In simple words, Artificial Intelligent System works like a Human Brain, where a machine or software shows intelligence while performing given tasks; such systems are called **intelligent systems** or **expert systems**. You can say that these systems can “think” while generating output!!!
- AI is one of the newest fields in science and engineering and has a wide variety of application fields. AI applications range from the general fields like learning, perception and prediction to the specific field, such as writing stories, proving mathematical theorems, driving a bus on a crowded street, diagnosing diseases, and playing chess.
- AI is the study of how to make machines do things which at the moment people do better. Following are the four approaches to define AI.

1.2 Foundations and Mathematical Treatments

- In general, artificial intelligence is the study of how to make machines do things which at the moment human do better. Following are the four approaches to define AI.
- Historically, all four approaches have been followed by different groups of people with different methods.

1.2.1 Acting Humanly : The Turing Test Approach

MU - May 16

Q. Explain turing test designed for satisfactory operational definition of AI.	(May 16, 5 Marks)
--	-------------------

- **Definition 1 :** “The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil, 1990)
- **Definition 2 :** “The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight, 1991)

- To judge whether the system can act like a human, Sir Alan Turing had designed a test known as **Turing test**.
- As shown in Fig. 1.2.1, in Turing test, a computer needs to interact with a human interrogator by answering his questions in written format. Computer passes the test if a human interrogator, cannot identify whether the written responses are from a person or a computer. Turing test is valid even after 60 year of research.

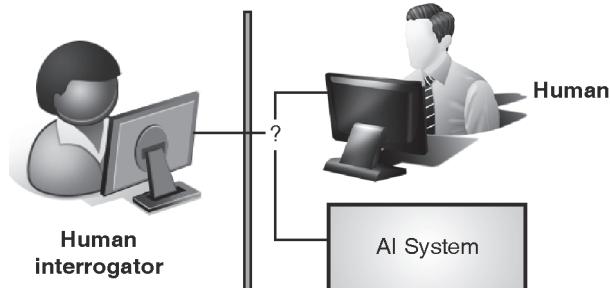


Fig.1.2.1 : Turing Test Environment

- For this test, the computer would need to possess the following capabilities:
 1. **Natural Language Processing (NLP)** : This unit enables computer to interpret the English language and communicate successfully.
 2. **Knowledge Representation** : This unit is used to store knowledge gathered by the system through input devices.
 3. **Automated Reasoning**: This unit enables to analyze the knowledge stored in the system and makes new inferences to answer questions.
 4. **Machine Learning**: This unit learns new knowledge by taking current input from the environment and adapts to new circumstances, thereby enhancing the knowledgebase of the system.
- To pass total Turing test, the computer will also need to have **computer vision**, which is required to perceive objects from the environment and **Robotics**, to manipulate those objects.

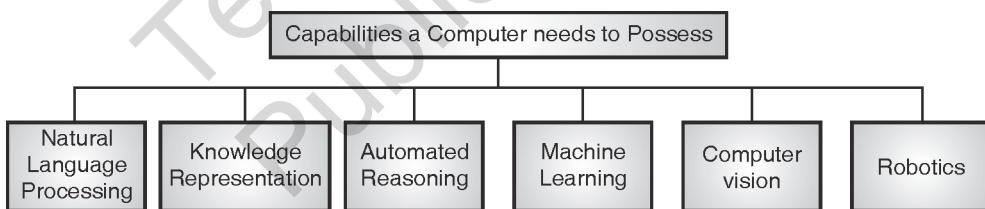


Fig. 1.2.2 : Capabilities a Computer needs to possess

- Fig. 1.2.2 lists all the capabilities a computer needs to have in order to exhibit artificial intelligence. Mentioned above are the six disciplines which implement most of the artificial intelligence.

1.2.2 Thinking Humanly : The Cognitive Modelling Approach

- **Definition1** : “The exciting new effort to make computers think ... machines with minds, in the full and literal sense”. (Haugeland, 1985)
- **Definition 2** : “The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning ...” (Hellman, 1978)
- **Cognitive science** :It is interdisciplinary field which combines computer models from Artificial Intelligence with the techniques from psychology in order to construct precise and testable theories for working of human mind.

- In order to make machines think like human, we need to first understand how human think. Research showed that there are three ways using which human's thinking pattern can be caught.
 1. **Introspection** through which human can catch their own thoughts as they go by.
 2. **Psychological experiments** can be carried out by observing a person in action.
 3. **Brain imaging** can be done by observing the brain in action.
- By catching the human thinking pattern, it can be implemented in computer system as a program and if the program's input output matches with that of human, then it can be claimed that the system can operate like humans.

1.2.3 Thinking Rationally : The “Laws of Thought” Approach

- **Definition1** : “The study of mental faculties through the use of computational models”. (Charniak and McDermott, 1985)
- **Definition2** : “The study of the computations that make it possible to perceive, reason, and act”.
- The laws of thought are supposed to implement the operation of the mind and their study initiated the field called logic. It provides precise notations to express facts of the real world.
- It also includes reasoning and “right thinking” that is irrefutable thinking process. Also computer programs based on those logic notations were developed to create intelligent systems.

There are two problems in this approach :

1. This approach is not suitable to use when 100% knowledge is not available for any problem.
2. As vast number of computations was required even to implement a simple human reasoning process; practically, all problems were not solvable because even problems with just a few hundred facts can exhaust the computational resources of any computer.

1.2.4 Acting Rationally : The Rational Agent Approach

- **Definition 1** :“Computational Intelligence is the study of the design of intelligent agents”. (Poole et at, 1998)
- **Definition 2**:“AI ... is concerned with intelligent behaviour in artifacts”.
(Nilsson, 1998)

Rational Agent

- Agents perceive their environment through sensors over a prolonged time period and adapt to change to create and pursue goals and take actions through actuators to achieve those goals. A rational agent is the one that does “right” things and acts rationally so as to achieve the best outcome even when there is uncertainty in knowledge.
- **The rational-agent approach has two advantages over the other approaches**
 1. As compared to other approaches this is the more general approach as, rationality can be achieved by selecting the correct inference from the several available.
 2. Rationality has specific standards and is mathematically well defined and completely general and can be used to develop agent designs that achieve it. Human behavior, on the other hand, is very subjective and cannot be proved mathematically.



- The two approaches namely, thinking humanly and thinking rationally are based on the reasoning expected from intelligent systems while; the other two acting humanly and acting rationally are based on the intelligent behaviour expected from them.
- In our syllabus we are going to study acting rationally approach.

1.3 Categorization of Intelligent Systems

As AI is a very broad concept, there are different types or forms of AI. The critical categories of AI can be based on the capacity of intelligent program or what the program is able to do. Under this consideration there are three main categories:

1. Artificial Narrow Intelligence/ Weak AI

Weak AI is AI that specializes in one area. It is not a general purpose intelligence. An intelligent agent is built to solve a particular problem or to perform a specific task is termed as narrow intelligence or weak AI. For example, it took years of AI development to be able to beat the chess grandmaster, and since then we have not been able to beat the machines at chess. But that is all it can do, which is does extremely well.

2. Artificial General Intelligence / Strong AI

Strong AI or general AI refers to intelligence demonstrated by machines in performing any intellectual task that human can perform. Developing strong AI is much harder than developing weak AI. Using artificial general intelligence machines can demonstrate human abilities like reasoning, planning, problem solving, comprehending complex ideas, learning from self experiences, etc. Many companies, corporations' are working on developing a general intelligence but they are yet to complete it.

3. Artificial Super Intelligence

As defined by a leading AI thinker Nick Bostrom, "Super intelligence is an intellect that is much smarter than the best human brains in practically every field, including scientific creativity, general wisdom and social skills." Super intelligence ranges from a machine which is just a little smarter than a human to a machine that is trillion times smarter. Artificial super intelligence is the ultimate power of AI.

1.4 Components of AI

AI is a vast field for research and it has got applications in almost all possible domains. By keeping this in mind, components of AI can be identified as follows: (Refer Fig.1.4.1)

1. Perception
2. Knowledge representation
3. Learning
4. Reasoning
5. Problem solving
6. Natural language processing (Language-understanding)

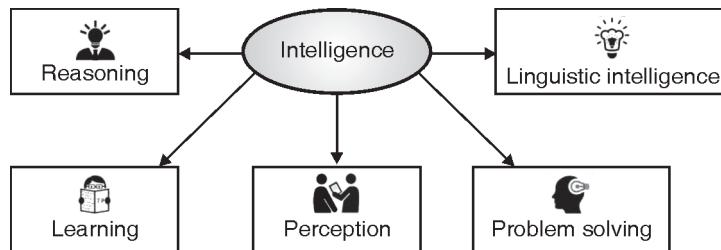


Fig. 1.4.1 : Components of AI

1. Perception

In order to work in the environment, intelligent agents need to scan the environment and the various objects in it. Agent scans the environment using various sense organs like camera, temperature sensor, etc. This is called as perception. After capturing various scenes, perceiver analyses the different objects in it and extracts their features and relationships among them.

2. Knowledge representation

The information obtained from environment through sensors may not be in the format required by the system. Hence, it need to be represented in standard formats for further processing like learning various patterns, deducing inference, comparing with past objects, etc. There are various knowledge representation techniques like Prepositional logic and first order logic.

3. Learning

Learning is a very essential part of AI and it happens in various forms. The simplest form of learning is by trial and error. In this form the program remembers the action that has given desired output and discards the other trial actions and learns by itself. It is also called as unsupervised learning. In case of rote learning, the program simply remembers the problem solution pairs or individual items. In other case, solution to few of the problems is given as input to the system, basis on which the system or program needs to generate solutions for new problems. This is known as supervised learning.

4. Reasoning

Reasoning is also called as logic or generating inferences form the given set of facts. Reasoning is carried out based on strict rule of validity to perform a specified task. Reasoning can be of two types, deductive or inductive. The deductive reasoning is in which the truth of the premises guarantees the truth of the conclusion while, in case of inductive reasoning, the truth of the premises supports the conclusion, but it cannot be fully dependent on the premises. In programming logic generally deductive inferences are used. Reasoning involves drawing inferences that are relevant to the given problem or situation.

5. Problem-solving

AI addresses huge variety of problems. For example, finding out winning moves on the board games, planning actions in order to achieve the defined task, identifying various objects from given images, etc. As per the types of problem, there is variety of problem solving strategies in AI. Problem solving methods are mainly divided into general purpose methods and special purpose methods. General purpose methods are applicable to wide range of problems while, special purpose methods are customized to solve particular type of problems.

6. Natural language processing

Natural Language Processing, involves machines or robots to understand and process the language that human speak, and infer knowledge from the speech input. It also involves the active participation from machine in the form of dialog i.e. NLP aims at the text or verbal output from the machine or robot. The input and output of an NLP system can be speech and written text respectively.

1.4.1 Computational Intelligence vs. Artificial Intelligence

Computational Intelligence (CI)	Artificial Intelligence (AI)
Computational Intelligence is the study of the design of intelligent agents	Artificial Intelligence is study of making machines which can do things which at presents human do better.
CI involves numbers and computations.	AI involves designs and symbolic knowledge representations.
CI constructs the system starting from the bottom level computations, hence follows bottom-up approach.	AI analyses the overall structure of an intelligent system by following top down approach.
CI concentrates on low level cognitive function implementation.	AI concentrates of high level cognitive structure design.

1.5 History of Artificial Intelligence

- The term **Artificial Intelligence (AI)** was introduced by John McCarthy, in 1955. He defined artificial intelligence as "The science and engineering of making intelligent machines".
- Mathematician Alan Turing and others presented a study based on logic driven computational theories which showed that any computer program can work by simply shuffling "0" and "1" (i.e. electricity off and electricity on). Also, during that time period, research was going on in the areas like Automations, Neurology, Control theory, Information theory, etc.
- This inspired a group of researchers to think about the possibility of creating an electronic brain. In the year 1956 a conference was conducted at the campus of Dartmouth College where the field of artificial intelligence research was founded.
- This conference was attended by John McCarthy, Marvin Minsky, Allen Newell and Herbert Simon, etc., who are supposed to be the pioneers of artificial intelligence research for a very long time. During that time period, Artificial Intelligence systems were developed by these researchers and their students.
- Let's see few examples of such artificial intelligent systems :
 1. **Game : Checkers** : Computer played as an opponent,
 2. **Education : Algebra** : For solving word problems,
 3. **Education : Math** :Proving logical theorems,
 4. **Education : Language** : Speaking English, etc.
- During that time period these founders predicted that in few years machines can do any work that a man can do, but they failed to recognize the difficulties which can be faced.

- Meanwhile we will see the ideas, viewpoints and techniques which Artificial Intelligence has inherited from other disciplines. They can be given as follows :
 1. **Philosophy** : Theories of reasoning and learning have emerged, along with the viewport that the mind is constituted by the operation of a physical system.
 2. **Mathematical** : Formal theories of logic, probability, decision making and computation have emerged.
 3. **Psychology** : Psychology has emerged tools to investigate the human mind and a scientific language which are used to express the resulting theories.
 4. **Linguistic** : Theories of the structure and meaning of language have emerged.
 5. **Computer science** : The tools which can make artificial intelligence a reality has emerged.

1.6 Applications of Artificial Intelligence

- You must have seen use of Artificial Intelligence in many SCI-FI movies. To name a few we have I Robot, Wall-E, The Matrix Trilogy, Star Wars, etc. movies. Many a times these movies show positive potential of using AI and sometimes also emphasize the dangers of using AI. Also there are games based on such movies, which show us many probable applications of AI.
- Artificial Intelligence is commonly used for problem solving by analyzing or/and predicting output for a system. AI can provide solutions for constraint satisfaction problems. It is used in wide range of fields for example in diagnosing diseases, in business, in education, in controlling a robots, in entertainment field, etc.

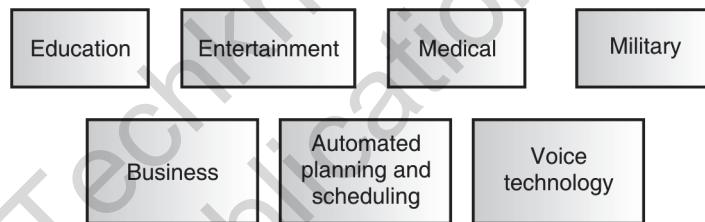


Fig. 1.6.1 : Fields of AI Application

- Fig. 1.6.1 shows few fields in which we have applications of artificial intelligence. There can be many fields in which Artificially Intelligent Systems can be used.

1. Education

Training simulators can be built using artificial intelligence techniques. Software for pre-school children are developed to enable learning with fun games. Automated grading, Interactive tutoring, instructional theory are the current areas of application.

2. Entertainment

Many movies, games, robots are designed to play as a character. In games they can play as an opponent when human player is not available or not desirable.

3. Medical

AI has applications in the field of cardiology (CRG), Neurology (MRI), Embryology (Sonography), complex operations of internal organs, etc. It can be also used in organizing bed schedules, managing staff rotations, store and retrieve information of patient. Many expert systems are enabled to predict the decease and can provide with medical prescriptions.

4. Military

Training simulators can be used in military applications. Also areas where human cannot reach or in life saving conditions, robots can be very well used to do the required jobs. When decisions have to be made quickly taking into account an enormous amount of information, and when lives are at stake, artificial intelligence can provide crucial assistance. From developing intricate flight plans to implementing complex supply systems or creating training simulation exercises, AI is a natural partner in the modern military.

5. Business and Manufacturing

Latest generation of robots are equipped well with the performance advances, growing integration of vision and an enlarging capability to transform manufacturing.

6. Automated planning and scheduling

Intelligent planners are available with AI systems, which can process large datasets and can consider all the constraints to design plans satisfying all of them.

7. Voice technology

Voice recognition is improved a lot with AI. Systems are designed to take voice inputs which are very much applicable in case of handicaps. Also scientists are developing an intelligent machine to emulate activities of a skillful musician. Composition, performance, sound processing, music theory are some of the major areas of research.

8. Heavy industry

Huge machines involve risk in operating and maintaining them. Human robots are better replacing human operators. These robots are safe and efficient. Robots are proven to be effective as compared to humans in the jobs of repetitive nature, humans may fail due to lack of continuous attention or laziness.

1.7 Sub Areas/ Domains of Artificial Intelligence

AI Applications can be roughly classified based on the type of tools/approaches used for inoculating intelligence in the system, forming sub areas of AI. Various sub domains/ areas in intelligent systems can be given as follows; Natural Language Processing, Robotics, Neural Networks and Fuzzy Logic. Fig. 1.7.1 shows these areas in Intelligent Systems.

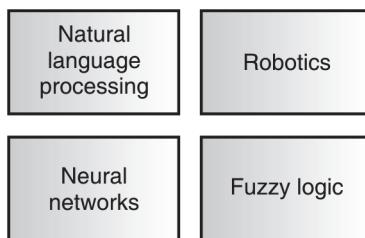


Fig. 1.7.1 : Sub-areas in Intelligent Systems

- Natural language processing :** One of the applications of AI is in the field of Natural Language Processing (NLP). NLP enables interaction between computers and human (natural) language. Practical applications of NLP are in machine translation (e.g. Google Translate), information retrieval, text categorization, etc. Few more applications are extracting 3D information using vision, speech recognition, perception, image formation.

2. **Robotics** : One more major application of AI is in Robotics. Robot is an active agent whose environment is the physical world. Robots can be used in manufacturing and handling material, in medical field, in military, etc. for automating the manual work.
3. **Neural networks** : Another application of AI is using Neural Networks. Neural Network is a system that works like a human brain/nervous system. It can be useful for stock market analysis, in character recognition, in image compression, in security, face recognition, handwriting recognition, Optical Character Recognition (OCR), etc.
4. **Fuzzy logic** : Apart from these AI systems are developed with the help of Fuzzy Logic. Fuzzy Logic can be useful in making approximations rather than having a fixed and exact reasoning for a problem. You must have seen systems like AC, fridge, washing machines which are based on fuzzy logic (they call it “6th sense technology!”).

1.8 Current Trends in Artificial Intelligence

Artificial Intelligence has touched each and every aspect of our life. From washing machine, Air conditioners, to smart phones everywhere AI is serving to ease our life. In industry, AI is doing marvellous work as well. Robots are doing the sound work in factories. Driverless cars have become a reality. WiFi-enabled Barbie uses speech-recognition to talk and listen to children. Companies are using AI to improve their product and increase sales. AI saw significant advances in machine learning. Following are the areas in which AI is showing significant advancements.

1. Deep learning

Convolutional Neural Networks enabling the concept of deep learning is the top most area of focus in Artificial intelligence in todays' era. Many problems and applications areas of AI like, natural language and text processing, speech recognition, computer vision, information retrieval, and multimodal information processing empowered by multi-task deep learning.

2. Machine learning

The goal of machine learning is to program computers to use example data or past experience to solve a given problem. Many successful applications of machine learning include systems that analyze past sales data to predict customer behaviour, optimize robot behaviour so that a task can be completed using minimum resources, and extract knowledge from bioinformatics data.

3. AI replacing workers

In industry where there are safety hazards, robots are doing a good job. Human resources are getting replaced by robots rapidly. People are worries to see that the white color jobs of data processing are being done exceedingly well by intelligent programs. A study from The National Academy of Sciences brought together technologists and economists and social scientists to figure out what's going to happen.

4. Internet of Things (IoT)

The concepts of smarter homes, smarter cars and smarter world is evolving rapidly with the invention of internet of things. The future is no far when each and every object will be wirelessly connected to something in order to perform soma smart actions without any human instructions or interference. The worry is how the mined data can potentially be exploited.

5. Emotional AI

Emotional AI, where AI can detect human emotions, is another upcoming and important area of research. Computers ability to understand speech will lead to an almost seamless interaction between human and computer. With increasingly accurate cameras, voice and facial recognition, computers are better able to detect our emotional state. Researchers are exploring how this new knowledge can be used in education, to treat depression, to accurately predict medical diagnoses, and to improve customer service and shopping online.

6. AI in shopping and customer service

Using AI, customers' buying patterns, behavioral patterns can be studied and systems that can predict the purchase or can help customer to figure out the perfect item. AI cab be used to find out what will make the customer happy or unhappy. For example, if a customer is shopping online, like a dress pattern but needs dark shades and thick material, computer understand the need and brings out new set of perfectly matching clothing for him.

7. Ethical AI

With all the evolution happening in technology in every walk of life, ethics must be considered at the forefront of research. For example, in case of driverless car, while driving, if the decision has to be made between weather to dash a cat or a lady having both in an uncontrollable distance in front of the car, is an ethical decision. In such cases how the programming should decide who is more valuable, is a question. These are not the problems to be solved by computer engineers or research scientists but someone has to come up with an answer.

1.9 Intelligent Agents

1.9.1 What is an Agent?

- **Agent** is something that perceives its environment through sensors and acts upon that environment through effectors or actuators. Fig. 1.9.1 shows agent and environment.
- Take a simple example of a human agent. It has five senses : Eyes, ears, nose, skin, tongue. These senses sense the environment are called as **sensors**. Sensors collect percepts or inputs from environment and passes it to the processing unit.
- Effectors or actuators are the organs or tools using which the agent acts upon the environment. Once the sensor senses the environment, it gives this information to nervous system which takes appropriate action with the help of effectors.
- In case of human agents we have hands, legs as effectors or actuators.

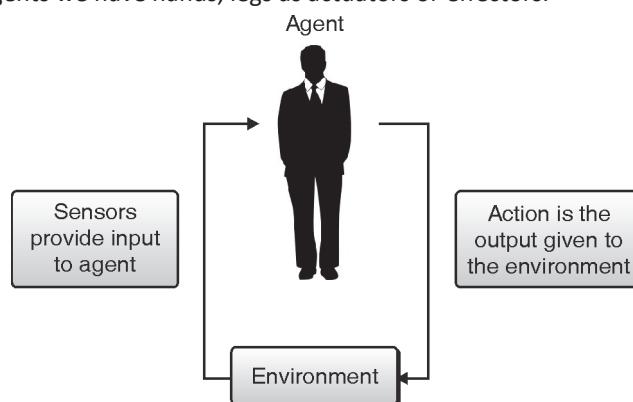


Fig. 1.9.1 : Agent and Environment

- Fig. 1.9.2 shows generic robotic agent structure.

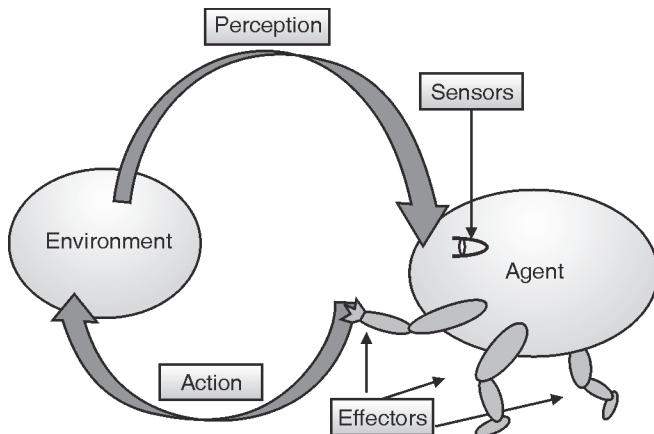


Fig.1.9.2 : Generic robotic agent architecture

- After understanding what an agent is, let's try to figure out sensor and actuator for a robotic agent, can you think of sensors and actuators in case of a robotic agent?
- The robotic agent has cameras, infrared range finders, scanners, etc. used as **sensors**, while various types of motors, screen, printing devices, etc. used as **actuators** to perform action on given input.

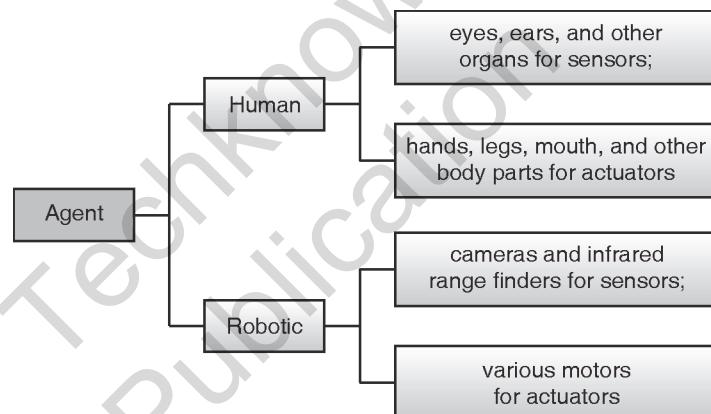


Fig.1.9.3 : Sensors and actuators in human and robotic agent

- The **agent function** is the **description of what all functionalities** the agent is supposed to do. The agent function provides mapping between percept sequences to the desired actions. It can be represented as $[f: P^* \Rightarrow A]$
- **Agent program** is a computer program that implements agent function in an architecture suitable language. Agent programs need to be installed on a device in order to run the device accordingly. That device must have some form of sensors to sense the environment and actuators to act upon it. Hence agent is a combination of the architecture hardware and program software.

Agent = Architecture + Program

- Take a simple example of vacuum cleaner agent. You might have seen vacuum cleaner agent in “WALL-E”(animated movie). Let's understand how to represent the percept’s (input) and actions (outputs) used in case of a vacuum cleaner agent.

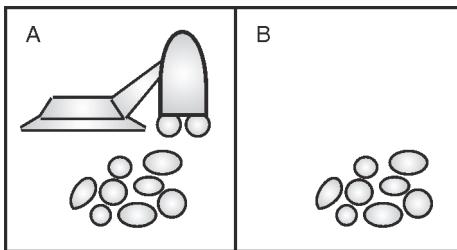


Fig.1.9.4 : Vacuum cleaner agent

- As shown in Fig. 1.9.4, there are two blocks A and B having some dirt. Vacuum cleaner agent supposed to sense the dirt and collect it, thereby making the room clean. In order to do that the agent must have a camera to see the dirt and a mechanism to move forward, backward, left and right to reach to the dirt. Also it should absorb the dirt. Based on the percepts, actions will be performed. For example : Move left, Move right, absorb, No Operation.
- Hence the sensor for vacuum cleaner agent can be camera, dirt sensor and the actuator can be motor to make it move, absorption mechanism. And it can be represented as :
[A, Dirty], [B, Clean], [A, absorb],[B, Nop], etc.

1.9.2 Definitions of Agent

There are various definitions exist for an agent. Let's see few of them.

- IBM states that **agents** are software entities that carry out some set of operations on behalf of a user or another program.
- FIPA** : Foundation for Intelligent Physical Agents (FIPA) terms that, an **agent** is a computational process that implements the autonomous functionality of an application.
- Another definition is given as “An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors”.

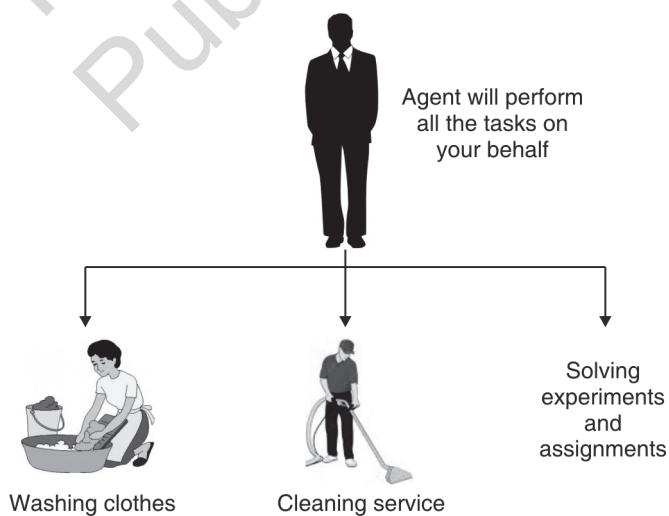


Fig.1.9.5 : Interactive Intelligent Agent

- By Russell and Norvig, F. Mills and R. Stuffle beam's definition says that “**An agent** is anything that is capable of acting upon information it perceives. An intelligent agent is an agent capable of making decisions about how it acts based on experience”.

- From above definitions we can understand that an agent is: (As per Terziyan, 1993)

- | | |
|---|---|
| <ul style="list-style-type: none">Goal-orientedAdaptiveSocial | <ul style="list-style-type: none">CreativeMobileSelf-configurable |
|---|---|

1.9.3 Intelligent Agent

- In the human agent example, we read that there is something called as "**Nervous System**" which helps in deciding an action with the assistance of effectors, based on the input given by sensors. In robotic agent, we have software's which demonstrates the functionality of nervous system.
- Intelligent agent** is the one which can take input from the environment through its sensors and act upon the environment through its actuators. Its actions are always directed to achieve a goal.
- The basic abilities of an intelligent agent are to exist to be self-governed, responsive, goal-oriented, etc.
- In case of intelligent agents, the software modules are responsible for exhibiting intelligence. Generally observed capabilities of an intelligent agent can be given as follows:
 - Ability to remain autonomous (Self-directed)
 - Responsive
 - Goal-Oriented
- Intelligent agent is the one which can take input from the environment through its sensors and act upon the environment through its actuators. Its actions are always directed to achieve a goal.

1.9.3(A) Structure of Intelligent Agents

- Fig.1.9.6 shows the general structure of an intelligent agent.

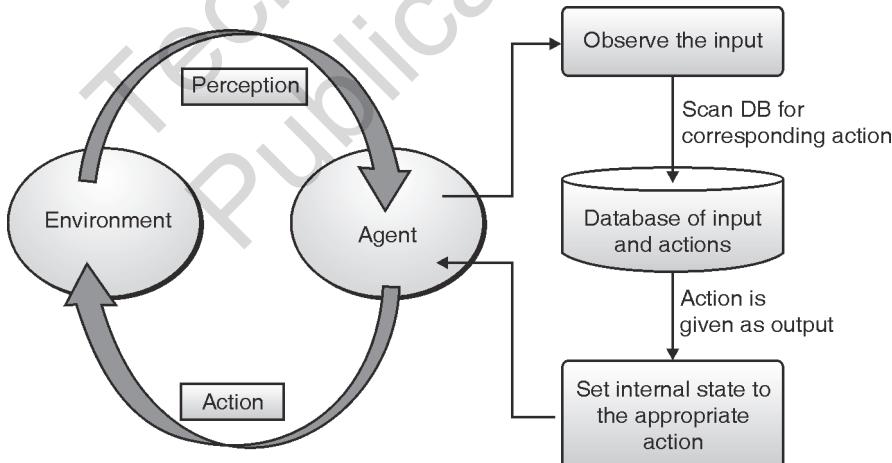


Fig.1.9.6: General structure of intelligent agent

- From Fig. 1.9.6, it can be observed how agent and environment interact with each other. Every time environment changes the agent first observes the environment through its sensors and get the input, then scans the database of input and actions for the corresponding action for given input and lastly sets the internal state to the appropriate action.
- Let's understand this working with a real life example. Consider you are an agent and your surroundings is an environment. Now, take a situation where you are cooking in kitchen and by mistake you touch a hot pan. We will see what happens in this situation step by step. Your touch sensors take input from environment (i.e. you have touched



some hot element), then it asks your brain if it knows “what action should be taken when you go near hot elements?” Now the brain will inform your hands (actuators) that you should immediately take it away from the hot element otherwise it will burn. Once this signal reaches your hand you will take your hand away from the hot pan.

- The agent keeps taking input from the environment and goes through these states every time. In above example, if your action takes more time then in that case your hand will be burnt.
- So the new task will be to find solution if the hand is burnt. Now, you think about the states which will be followed in this situation. As per Wooldridge and Jennings, “An intelligent agent is one that is capable of taking flexible self-governed actions”.
- They say for an intelligent agent to meet design objectives, flexible means three things:

- | | |
|-------------------|-------------------|
| 1. Reactiveness | 2. Pro-activeness |
| 3. Social ability | |

1. **Reactiveness** : It means giving reaction to a situation in a stipulated time frame. An agent can perceive the environment and respond to the situation in a particular time frame. In case of reactiveness, reaction within situation time frame is more important. You can understand this with above example, where, if an agent takes more time to take his hand away from the hot pan then agents hand will be burnt.
 2. **Pro-activeness** : It is controlling a situation rather than just responding to it. Intelligent agent show goal-directed behavior by taking the initiative. For example : If you are playing chess then winning the game is the main objective. So here we try to control a situation rather than just responding to one-one action which means that killing or losing any of the 16 pieces is not important, whether that action can be helpful to checkmate your opponent is more important.
 3. **Social ability** : Intelligent agents can interact with other agents (also humans). Take automatic car driver example, where agent might have to interact with other agent or a human being while driving the car.
- Following are few more features of an intelligent agent.
 - **Self-Learning** : An intelligent agent changes its behaviour based on its previous experience. This agent keeps updating its knowledge base all the time.
 - **Movable/Mobile** : An Intelligent agent can move from one machine to another while performing actions.
 - **Self-governing** : An Intelligent agent has control over its own actions.

1.10 Rational Agent

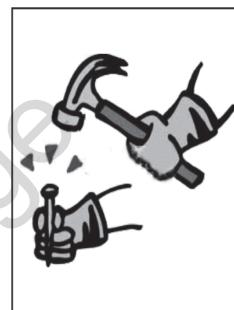
MU - Dec. 15

Q. Define rationality and rational agent. Give an example of rational action performed by any intelligent agent.

(Dec. 15, 5 Marks)

- For problem solving, if an agent makes a decision based on some logical reasoning, then, the decision is called as a “Rational Decision”. The way humans have ability to make right decisions, based on his/her experience and logical reasoning; an agent should also be able to make correct decisions, based on what it knows from the percept sequence and actions which are carried out by that agent from its knowledge.
- Agents perceive their environment through sensors over a prolonged time period and adapt to change to create and pursue goals and take actions through actuators to achieve those goals. **A rational agent** is the one that does “right” things and acts rationally so as to achieve the best outcome even when there is uncertainty in knowledge.
- A rational agent is an agent that has clear preferences, can model uncertainty via expected values of variables or functions of variables, and always chooses to perform the action with the optimal expected outcome for itself from among all feasible actions. A rational agent can be anything that makes decisions, typically a person, a machine, or software program.

- **Rationality** depends on four main criteria: First is the performance measure which defines the criterion of success for an agent, second is the agent's prior knowledge of the environment, and third is the action performed by the agent and the last one is agent's percept sequence to date.
- Performance measure is one of the major criteria for measuring success of an agent's performance. Take a vacuum-cleaner agent's example. The performance measure of a vacuum-cleaner agent can depend upon various factors like it's dirt cleaning ability, time taken to clean that dirt, consumption of electricity, etc.
- For every percept sequence a built-in knowledge base is updated, which is very useful for decision making, because it stores the consequences of performing some particular action. If the consequences direct to achieve desired goal then we get a good performance measure factor, else, if the consequences do not lead to desired goal state, then we get a poor performance measure factor.



(a) Agent's finger is hurt while using nail and hammer

(b) Agent is using nail and hammer efficiently

Fig. 1.10.1

- For example, see Fig.1.10.1. If agent hurts his finger while using nail and hammer, then, while using it for the next time agent will be more careful and the probability of not getting hurt will increase. In short agent will be able to use the hammer and nail more efficiently.
- Rational agent can be defined as an agent who makes use of its percept sequence, experience and knowledge to maximize the performance measure of an agent for every probable action. It selects the most feasible action which will lead to the expected results optimally.

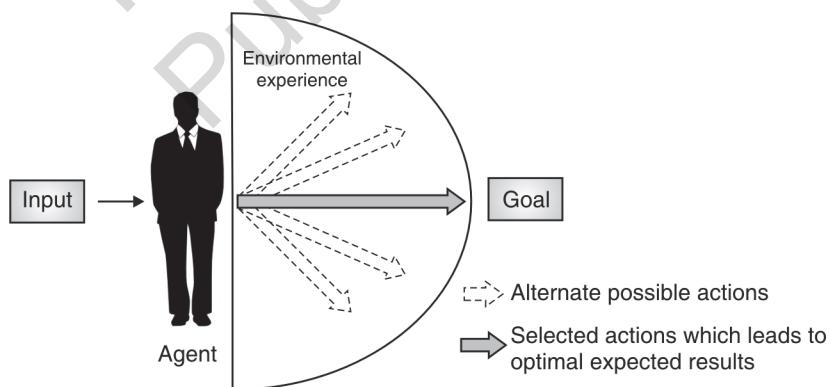


Fig.1.10.2 : Rational Agent

1.11 Nature of Environment and PEAS Properties of Agent

1.11.1 Environments Types / Nature of Environment / Task Environment Properties

MU -Dec. 13, May 15

Q. Describe different types of environments applicable to AI agents.

(Dec. 13, May 15, 10 Marks)

1. Fully observable vs. Partially observable

- The first type of environment is based on the observability. Whether the agent sensors can have access to complete state of environment at any given time or not, decides if it is a fully observable or partially observable environment.
- In **Fully observable** environments agents are able to gather all the necessary information required to take actions. Also in case of fully observable environments agents don't have to keep records of internal states. For example, Word-block problem, 8-puzzle problem, Sudoku puzzle, etc. in all these problem worlds, the state is completely visible at any point of time.

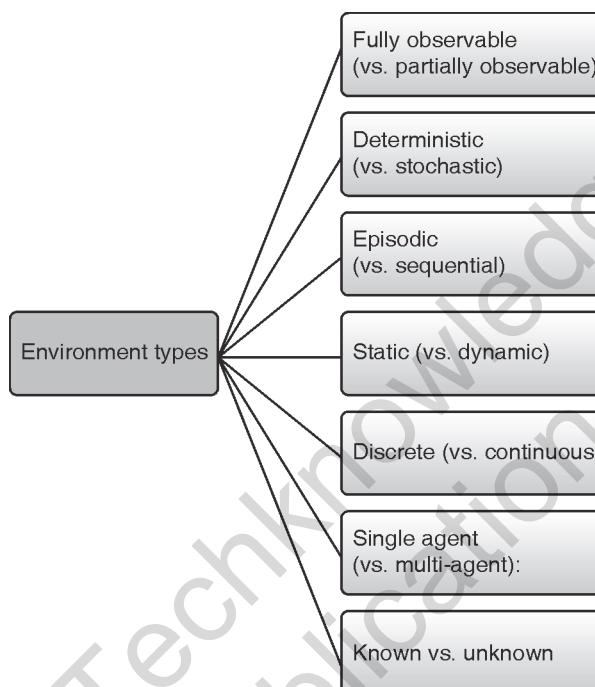


Fig.1.11.1: Environment types

- Environments are called **partially observable** when sensors cannot provide errorless information at any given time for every internal state, as the environment is not seen completely at any point of time.
- Also there can be unobservable environments where the agent sensors fail to provide information about internal states.
- For example, In case of an **automated car driver system**, automated car cannot predict what the other drivers are thinking while driving cars. Only because of the sensor's information gathering expertise it is possible for an automated car driver to take the actions.

2. Single agent vs. Multi-agent

- The second type of an environment is based on the number of agents acting in the environment. Whether the agent is operating on its own or in collaboration with other agents decides if it is a **Single agent** or a multi-agent environment.
- For example : An agent playing Tetris by itself can be a single agent environment, whereas we can have an agent playing checkers in a two-agent environment. Or in case of vacuum cleaner world, only one machine is working, so it's a single agent while in case of car driving agent, there are multiple agents driving on the road, hence it's a **multi-agent environment**.

- Multi-agent environment is further classified as Co-operative multi-agent and Competitive multi-agent. Now, you might be thinking in case of an automated car driver system which type of agent environment do we have?
- Let's understand it with the help of an automated car driving example. For a car driving system 'X', other car say 'Y' is considered as an Agent. When 'Y' tries to maximize its performance measure and the input taken by car 'Y' depends on the car 'X'. Thus it can be said that for an automated car driving system we have a cooperative multi-agent environment.
- Whereas in case of "chess game" when two agents are operating as opponents, and trying to maximize their own performance, they are acting in competitive multi agent environment.

3. Deterministic vs. Stochastic

- An environment is called **deterministic environment**, when the next state of the environment can be completely determined by the previous state and the action executed by the agent.
- For example, in case of vacuum cleaner world, 8-puzzle problem, chess game the next state of the environment solely depends on the current state and the action performed by agent.
- **Stochastic environment** generally means that the indecision about the actions is enumerated in terms of probabilities. That means environment changes while agent is taking action, hence the next state of the world does not merely depends on the current state and agent's action. And there are few changes happening in the environment irrespective of the agent's action. An automated car driving system has a stochastic environment as the agent cannot control the traffic conditions on the road.
- In case of checkers we have a multi-agent environment where an agent might be unable to predict the action of the other player. In such cases if we have partially observable environment then the environment is considered to be stochastic.
- If the environment is deterministic except for the actions of other agents, then the environment is **strategic**. That is, in case of game like chess, the next state of environment does not only depend upon the current action of agent but it is also influenced by the strategy developed by both the opponents for future moves.
- We have one more type of environment in this category. That is when the environment types are not fully observable or non-deterministic; such type of environment is called as **uncertain environment**.

4. Episodic vs. Sequential

- An **episodic task environment** is the one where each of the agent's action is divided into an atomic incidents or episodes. The current incident is different than the previous incident and there is no dependency between the current and the previous incident. In each incident the agent receives an input from environment and then performs a corresponding action.
- Generally, classification tasks are considered as episodic. Consider an example of pick and place robot agent, which is used to detect defective parts from the conveyor belt of an assembly line. Here, every time agent will make the decision based on current part, there will not be any dependency between the current and previous decision.
- In **sequential environments**, as per the name suggests, the previous decision can affect all future decisions. The next action of the agent depends on what action he has taken previously and what action he is supposed to take in future.

- For example, in checkers where previous move can affect all the following moves. Also sequential environment can be understood with the help of an automatic car driving example where, current decision can affect the next decisions. If agent is initiating breaks, then he has to press clutch and lower down the gear as next consequent actions.

5. Static vs. Dynamic

- You have learnt about static and dynamic terms in previous semesters with respect to web pages. Same way we have **static (vs. dynamic) environments**. If an environment remains unchanged while the agent is performing given tasks then it is called as a static environment. For example, Sudoku puzzle or vacuum cleaner environment are static in nature.
- If environment is not changing over the time but, an agent's performance is changing then, it is called as a **semi-dynamic** environment. That means, there is a timer exist in the environment who is affecting the performance of the agent.
- For example, In chess game or any puzzle like block word problem or 8-puzzle if we introduce timer, and if agent's performance is calculated by time taken to play the move or to solve the puzzle, then it is called as semi-dynamic environment.
- Lastly, if the environment changes while an agent is performing some task, then it is called **dynamic environment**.
- In this type of environment agent's sensors have to continuously keep sending signals to agent about the current state of the environment so that appropriate action can be taken with immediate effect.
- Automatic car driver example comes under dynamic environment as the environment keeps changing all the time.

6. Discrete vs. Continuous

- You have seen discrete and continuous signals in old semesters. When you have distinct, quantized, clearly defined values of a signal it is considered as discrete signal.
- Same way, when there are distinct and clearly defined inputs and outputs or precepts and actions, then it is called a **discrete environment**. For example : chess environment has a finite number of distinct inputs and actions.
- When a continuous input signal is received by an agent, all the precepts and actions cannot be defined beforehand then it is called **continuous environment**. For example : An automatic car driving system.

7. Known vs. Unknown

- In a **known environment**, the output for all probable actions is given. Obviously, in case of **unknown environment**, for an agent to make a decision, it has to gain knowledge about - how the environment works.
- Table 1.11.1 summarizes few task environment and their characteristics.

Table 1.11.1 : Task environments

Task environment	Car driving	Part – Picking Robot	Cross word puzzle	Soccer game	Checkers with clock
Observable	Partially	Partially	fully	Partially	Fully
Agents	Multi agent (cooperative)	Single agent	single	Multi agent (competitive)	Multi agent (competitive)
Deterministic	Stochastic	Stochastic	Deterministic	Strategic	Strategic



Task environment	Car driving	Part – Picking Robot	Cross word puzzle	Soccer game	Checkers with clock
Episodic	Sequential	Episodic	Sequential	Sequential	Sequential
Static	Dynamic	Dynamic	Static	Dynamic	Semi
Discrete	Continuous	Discrete	Discrete	Continuous	Discrete
Known and Unknown	Unknown	Known	Known	Known	Known

1.11.2 PEAS Properties of Agent

MU - May 13, Dec. 14, May 16

- | | |
|---|--|
| <p>Q. Give PEAS description for a robot soccer player. Characterize its environment.</p> <p>Q. What are PEAS descriptor ? Give PEAS descriptors for Part – picking Robot.</p> | <p style="margin: 0;">(May 16, 5 Marks)</p> <p style="margin: 0;">(May 13, Dec. 14, 3 Marks)</p> |
|---|--|

- **PEAS :** PEAS stands for **Performance Measure, Environment, Actuators, and Sensors**. It is the short form used for performance issues grouped under Task Environment.
- You might have seen driverless/ self driving car videos of Audi/ Volvo/ Mercedes, etc. To develop such driverless cars we need to first define PEAS parameters.
- **Performance Measure :** It the objective function to judge the performance of the agent. For example, in case of pick and place robot, number of correct parts in a bin can be the performance measure.
- **Environment :** It the real environment where the agent need to deliberate actions.
- **Actuators :** These are the tools, equipment or organs using which agent performs actions in the environment. This works as the output of the agent.
- **Sensors :** These are the tools, equipment or organs using which agent captures the state of the environment. This works as the input to the agent.
- To understand the concept of PEAS, consider following examples.

(A) Automated car driving agent

1. Performance measures which should be satisfied by the automated car driver:
 - (i) **Safety :** Automated system should be able to drive the car safely without dashing anywhere.
 - (ii) **Optimum speed :** Automated system should be able to maintain the optimal speed depending upon the surroundings.
 - (iii) **Comfortable journey :** Automated system should be able to give a comfortable journey to the end user, i.e. depending upon the road it should ensure the comfort of the end user.
 - (iv) **Maximize profits :** Automated system should provide good mileage on various roads, the amount of energy consumed to automate the system should not be very high, etc. such features ensure that the user is benefited with the automated features of the system and it can be useful for maximizing the profits.

2. Environment

- (i) **Roads :** Automated car driver should be able to drive on any kind of a road ranging from city roads to highway.

(ii) Traffic conditions : You will find different set of traffic conditions for different type of roads. Automated system should be able to drive efficiently in all types of traffic conditions. Sometimes traffic conditions are formed because of pedestrians, animals, etc.

(iii) Clients : Automated cars are created depending on the client's environment. For example, in some countries you will see left hand drive and in some countries there is a right hand drive. Every country/state can have different weather conditions. Depending upon such constraints automated car driver should be designed.

3. Actuators are responsible for performing actions/providing output to an environment.

In case of car driving agent following are the actuators :

(i) Steering wheel which can be used to direct car in desired direction (i.e. right/left)

(ii) Accelerator, gear, etc. can be useful to increase or decrease the speed of the car.

(iii) Brake is used to stop the car.

(iv) Light signal, horn can be very useful as indicators for an automated car.

4. Sensors: To take input from environment in car driving example cameras, sonar system, speedometer, GPS, engine sensors, etc. are used as sensors.

(B) Part-picking ARM robot

(i) Performance measures : Number of parts in correct container.

(ii) Environment : Conveyor belt used for handling parts, containers used to keep parts, and Parts.

(iii) Actuators : Arm with tooltips, to pick and drop parts from one place to another.

(iv) Sensors : Camera to scan the position from where part should be picked and joint angle sensors which are used to sense the obstacles and move in appropriate place.

(C) Medical diagnosis system

(i) Performance measures

a. **Healthy patient:** system should make use of sterilized instruments to ensure the safety (healthiness) of the patient.

b. **Minimize costs :** The automated system results should not be very costly otherwise overall expenses of the patient may increase, Lawsuits. Medical diagnosis system should be legal.

(ii) Environment : Patient, Doctors, Hospital Environment

(iii) Sensors : Screen, printer

(iv) Actuators : Keyboard and mouse which is useful to make entry of symptoms, findings, patient's answers to given questions. Scanner to scan the reports, camera to click pictures of patients.

(D) Soccer player robot

(i) Performance measures : Number of goals, speed, legal game.

(ii) Environment: Team players, opponent team players, playing ground, goal net.

(iii) Sensors: Camera, proximity sensors, infrared sensors.

(iv) Actuators : Joint angles, motors.

1.12 Structure of Agents / Types of Agents

- Depending upon the degree of intelligence and ability to achieve the goal, agents are categorized into five basic types. These five types of agents are depicted in the Fig. 1.12.1.

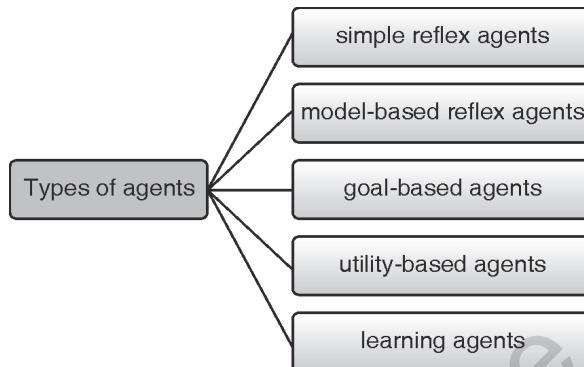


Fig.1.12.1 : Types of agents

- Let us understand these agent types one by one.

1.12.1 Simple Reflex Agents

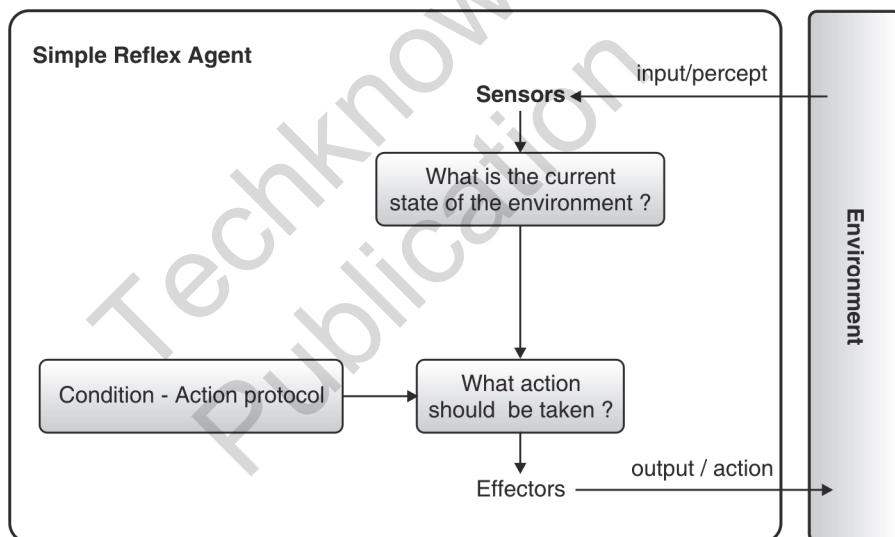


Fig.1.12.2 : Simple reflex agents

- An agent which performs actions based on the current input only, by ignoring all the previous inputs is called as **simple reflex agent**.
- It is a totally uncomplicated type of agent. The simple reflex agent's function is based on the situation and its corresponding action (condition-action protocol). If the condition is true, then matching action is taken without considering the percept history.
- You can understand simple reflexes with the help of a real life example, say some object approaches eye then, you will blink your eye. This type of simple reflex is called natural/innate reflex.
- Consider the example of the vacuum cleaner agent. It is a simple reflex agent, as its decision is based only on whether the current location contains dirt. The agent function is tabulated in Table 1.12.1.

- Few possible input sequences and outputs for vacuum cleaner world with 2 locations are considered for simplicity.

Table 1.12.1

Figure	Input sequence {location, content }	Output / action Right, left, suck, no-op
A B	{A, clean}	Right
A B	{B, clean}	Left
A B	{A, dirt}	Suck
A B	{B, dirt}	Suck
A B	Input sequence {location, content }	Output / action Right, left, suck, no-op
A B	{A, clean}{A, clean}	Right
A B	{A, clean}{A, dirt}	Suck
A B	:	:
	:	:
	:	:

- In case of above mentioned vacuum agent only one sensor is used and that is a dirt sensor. This dirt sensor can detect if there is dirt or not. So the possible inputs are ‘dirt’ and ‘clean’.
- Also the agent will have to maintain a database of actions, which will help to decide what output should be given by an agent. Database will contain conditions like : If there is dirt on the floor to left or right then find out if there is dirt in the next location and repeat these actions till the entire assigned area is cleaned then, vacuum cleaner should suck that dirt. Else, dirt should move. Once the assigned area is fully covered, no other action should be taken until further instruction.
- If the vacuum cleaner agent keeps searching for dirt and clean area, then, it will surely get trapped in an infinite loop. Infinite loops are unavoidable for simple reflex agents operating in partially observable environments. By randomizing its actions the simple reflex agent can avoid these infinite loops. For example, on receiving {clean} as input, the vacuum cleaner agent should either go to left or right direction.
- If the performance of an agent is of the right kind then randomized behaviour can be considered as rational in few multi-agent environments.

1.12.2 Model-Based Reflex Agents

- Partially observable environment cannot be handled well by simple reflex agents because it does not keep track on the previous state. So, one more type of agent was created that is model based reflex agent.
- An agent which performs actions based on the current input and one previous input is called as model-based agent. Partially observable environment can be handled well by model-based agent.
- From Fig. 1.12.3, it can be seen that once the sensor takes input from the environment, agent checks for the current state of the environment. After that, it checks for the previous state which shows how the world is developing and how the environment is affected by the action which was taken by the agent at earlier stage. This is termed as model of the world.

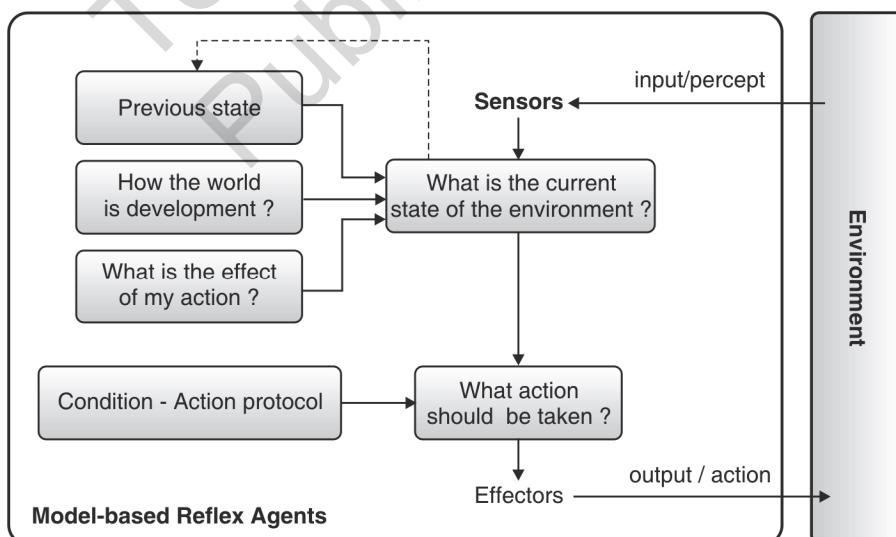


Fig.1.12.3 : Model-based reflex agents

- Once this is verified, based on the condition-action protocol an action is decided. This decision is given to effectors and the effectors give this output to the environment.

- The knowledge about “how the world is changing” is called as a model of the world. Agent which uses such model while working is called as the **“model-based agent”**.
- Consider a simple example of automated car driver system. Here, the world keeps changing all the time. You must have taken a wrong turn while driving on some or the other day of your life. Same thing applies for an agent. Suppose if some car “X” is overtaking our automated driver agent “A”, then speed and direction in which “X” and “A” are moving their steering wheels is important. Take a scenario where agent missed a sign board as it was overtaking other car. The world around that agent will be different in that case.
- Internal model based on the input history should be maintained by model-based reflex agent, which can reflect at least some of the unobserved aspects of the current state. Once this is done it chooses an action in the same way as the simple reflex agent.

1.12.3 Goal-Based Agents

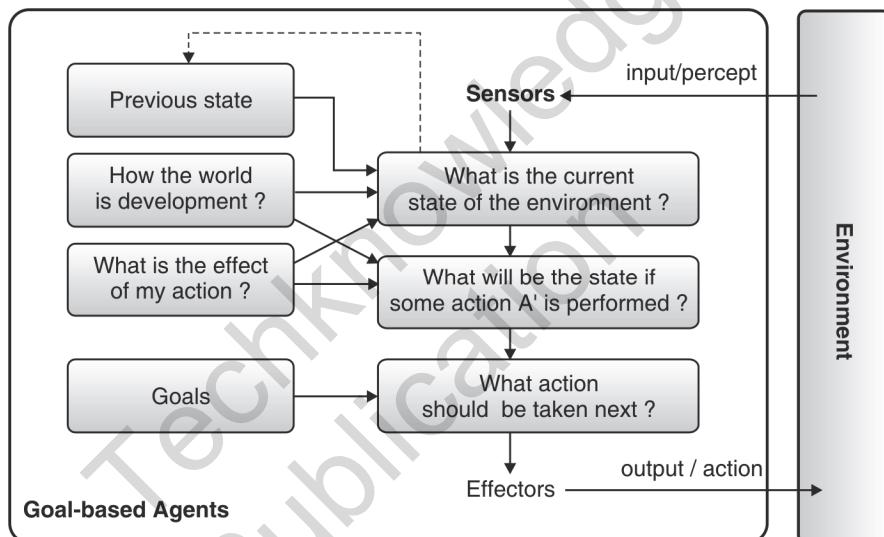
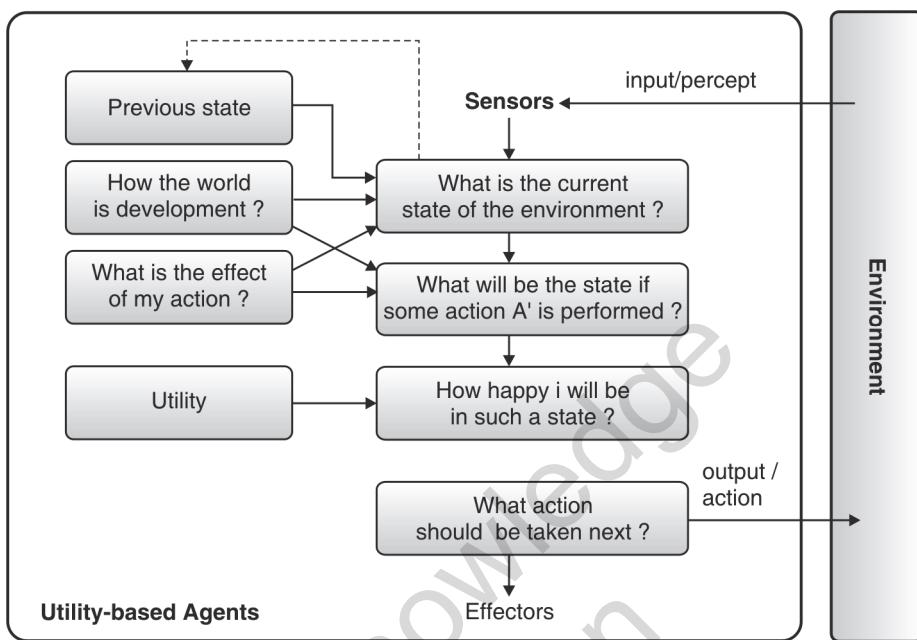


Fig.1.12.4 : Goal-based agents

- Model-based agents are further developed based on the “goal” information. This new type of agent is called as goal-based agent. As the name suggests, Goal information will illustrate the situations that is desired. These agents are provided with goals along with the model of the world. All the actions selected by the agent are with reference of the specified goals. Goal based agents can only differentiate between goal states and non-goal states. Hence, their performance can be 100% or zero.
- The limitation of goal based agent comes with its definition itself. Once the goal is fixed, all the actions are taken to fulfill it. And the agent loses flexibility to change its actions according to the current situation.
- You can take example of a vacuum cleaning robot agent whose goal is to keep the house clean all the time. This agent will keep searching for dirt in house and will keep the house clean all the time. Remember M-O the cleaning robot from Wall-E movie which keeps cleaning all the time no matter what is the environment or the Healthcare companion robot Baymax from Big Hero 6 which does not deactivate until user says that he/she is satisfied with care.

1.12.4 Utility-Based Agents

MU -May 13**Q.** Explain utility-based agents with the help of neat diagram.**(May 13, 10 Marks)****Fig.1.12.5:Utility-based agents**

- Utility function is used to map a state to a measure of utility of that state. We can define a measure for determining how advantageous a particular state is for an agent. To obtain this measure utility function can be used.
- The term utility is used to depict how “happy” the agent is to find out a generalized performance measure, various world states according to exactly how happy they would make an agent is compared.
- Take one example; you might have used Google maps to find out a route which can take you from source location to your destination location in least possible time. Same logic is followed by utility based automatic car driving agent.
- Goals utility based automatic car driving agent can be used to reach given location safely within least possible time and save fuel. So this car driving agent will check the possible routes and the traffic conditions on these routes and will select the route which can take the car at destination in least possible time safely and without consuming much fuel.

1.12.5 Learning Agents

MU -May 13, Dec. 13, May 14, May 15, Dec. 15, May 16**Q.** Explain the learning agent with the help of suitable diagram.**(May 13, 10 Marks)****Q.** Explain the structure of learning agent architecture. What is role of critic in learning ?**(Dec. 13, May 15, 10 Marks)****Q.** Discuss structure of learning agent.**(May 14, 5 Marks)****Q.** What are the basic building blocks of learning agent ? Explain each of them with a neat block diagram.**(Dec. 15, May 16, 8/10 Marks)**

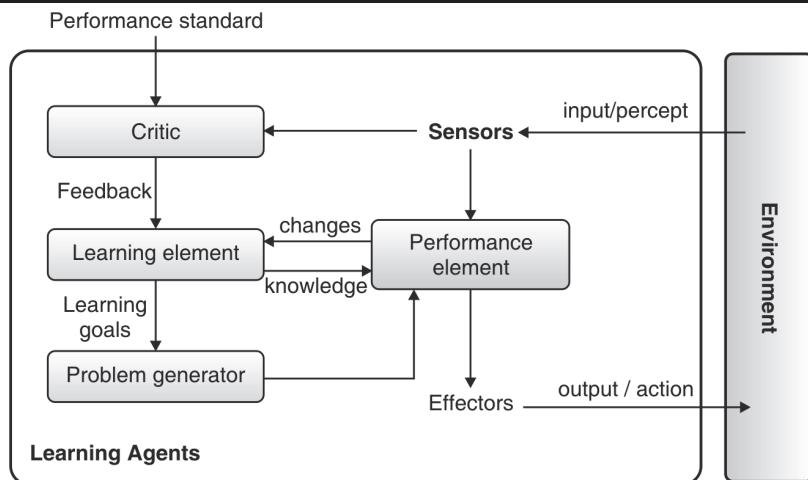


Fig. 1.12.6 : Learning agents

- Why do you give mock tests ? When you get less marks for some question, you come to know that you have made some mistake in your answer. Then you learn the correct answer and when you get that same question in further examinations, you write the correct answer and avoid the mistakes which were made in the mock test. This same concept is followed by the learning agent.
- Learning based agent is advantageous in many cases, because with its basic knowledge it can initially operate in an unknown environment and then it can gain knowledge from the environment based on few parameters and perform actions to give better results.
- Following are the components of learning agent :

1. Critic
2. Learning element
3. Performance element
4. Problem generator

1. **Critic** : It is the one who compares sensor's input specifying effect of agent's action on the environment with the performance standards and generate feedback for leaning element.
2. **Learning element** : This component is responsible to learn from the difference between performance standards and the feedback from critic. According to the current percept it is supposed to understand the expected behavior and enhance its standards
3. **Performance element** : Based on the current percept received from sensors and the input obtained by the learning element, performance element is responsible to choose the action to act upon the external environment.
4. **Problem generator** : Based on the new goals learnt by learning agent, problem generator suggests new or alternate actions which will lead to new and instructive understanding.

1.13 Introduction to Soft Computing

- Soft computing is a collection of all the techniques that help us to construct computationally intelligent systems. It has been now realized that, real world problems are complex, pervasively imprecise and uncertain. To solve such problems, we require computationally intelligent systems that combine knowledge, techniques and methodologies from various sources.

- There are three main requirements of any intelligent system.
 1. They must possess **human like expertise** within a specific domain.
 2. They should be able to **adapt and learn** to do better in changing environment.
 3. Should be capable of **making decisions** and taking actions accordingly.

1.14 Soft Computing vs. Hard Computing

- Hard computing involves the traditional methods of computing that require precisely stated analytical models. They often require more computational time. Examples of hard computing are:
 - Solving numerical problems (e.g. roots of polynomial, integration etc.)
 - Searching and sorting techniques
 - Solving Computational geometry problem etc.
- Unlike hard computing, soft computing techniques are tolerant of imprecision, uncertainty, partial truth and approximation that are present in the real world problems. Examples of soft computing techniques are Neural networks, Fuzzy logic, genetic algorithms etc.
- Following are differences between hard computing and soft computing.

Sr. No.	Hard computing	Soft computing
1.	Hard computing is a conventional type of computing that requires a precisely stated analytic model.	Soft computing techniques are imprecision, approximation and uncertainty tolerant.
2.	Hard computing requires programs to be written.	Soft computing techniques are model free. They can evolve their own models and programs.
3.	Hard computing is deterministic and uses two-valued logic.	Soft computing is stochastic and uses multi-valued logic such as fuzzy logic.
4.	Hard computing needs exact data to solve a particular problem.	Soft computing can deal with incomplete, uncertain and noisy data.
5.	Hard computing techniques perform sequential computation.	Soft computing allows parallel computations. E.g. Neural networks.
6.	The solution or output of hard computing is precise.	Soft computing can generate approximate output or solution.
7.	Hard computing is based on crisp logic, binary logic and numerical analysis.	Soft computing is based on neural networks, fuzzy logic, and evolutionary computations etc.
8.	Hard computing techniques are not fault tolerant. The reason is conventional programs and algorithms are built in such a way that errors have serious consequences, unless enough redundancy is added into the system.	Soft computing techniques are fault tolerant due to their redundancy, adaptability and reduced precision characteristics.

1.15 Various Types of Soft Computing Techniques

- Soft Computing is the fusion of different techniques that were designed to model and enable solutions to complex real world problems.
- These real world problems are the problems that are too difficult to model, mathematically.
- These problems result from the fact that our world seems to be imprecise, uncertain and difficult to categorize.
- The soft computing techniques are capable of handling such uncertainty, imprecision and vagueness present in the real world data.
- Most of the Soft computing techniques are based on some biological inspired methodologies such as human nervous systems, genetics, evolution, ant's behaviors etc.
- Soft Computing is the fusion of different techniques that were designed to model and enable solutions to complex real world problems, which are not modeled or too difficult to model, mathematically.
- Soft computing consist several computing paradigms mainly are:
 - Neural Network
 - Fuzzy Logic
 - Evolutionary Algorithms such as Genetic algorithm
- Every paradigm of soft computing mentioned above has its own strength. In order to build a computationally intelligent system, we may integrate multiple techniques or methodologies to take advantage of the strengths of each of them. Such systems are called **Hybrid soft computing** systems.
- Table 1.15.1 summarizes the soft computing methodologies and their strengths.

Table 1.15.1 : Soft computing constituents and their strengths

Sr. No.	Methodology	Strengths
1.	Neural networks	Has capability of learning and adaptation.
2.	Fuzzy set theory	Handles uncertainty and incorporates human-like reasoning into the system.
3.	Evolutionary algorithms	Has capability of finding optimum solution to a problem.

- The seamless integration of these methodologies forms the base of soft computing.
- **Neural networks** have the capability of recognizing patterns and adapting themselves to cope with changing environments.
- The **evolutionary algorithms** such as Genetic Algorithms are search and optimization techniques based on biological evolution that help us to optimize certain parameters in a given problem.
- **Fuzzy logic** incorporates human knowledge and performs inference and decision making.

1.15.1 Introduction to Neural Networks

- An Artificial Neural Network (ANN), inspired by the biological nervous system basically tries to mimic the working of a human brain.
- An ANN is composed of a large number of highly interconnected processing elements called neurons. All these neurons work in parallel to solve a specific problem.

- An ANN learns by examples the way humans learn by their experiences.
- ANN can be designed and configured for a specific application such as data classification, pattern reorganization, data clustering etc.

Advantages of Neural Networks

1. Neural networks provide human like artificial intelligence.
2. A neural network learns and does not need to be reprogrammed.
3. A neural network can do a task that a linear program cannot do.
4. Parallel organization of neural networks permits solutions to problems where multiple constraints must be satisfied simultaneously.
5. Because of its parallel nature, when an element of the neural network fails, it can continue without any problem.

Applications of Neural Networks

Neural networks have been successfully applied to a broad spectrum of data-intensive applications. Few of them are listed below.

(a) Forecasting

Neural network can be used very effectively in forecasting exchange rates, predicting stock values, inflation and cash forecasting, forecasting weather conditions etc. Researchers have proved that the forecasting accuracy of NN systems tend to excel over that of the linear regression model.

(b) Image compression

- Digital images require a large amount of memory for storage. As a result, the transmission of image from one computer to another can be very expensive in terms of time and bandwidth required.
- With the explosion of Internet, more sites are using images. Image compression is a technique that removes some of the redundant information present in the image without affecting its perceptibility, thus, reducing the storage size required to store the image.
- NN can be effectively used to compress the image. Several NN techniques such as Kohonen's self organizing maps, Back propagation algorithm, Cellular neural network etc. can be used for image compression.

(c) Industrial process control

- Neural networks have been applied successfully in industrial process control of dynamic systems.
- Neural networks (especially multi layer perceptrons) have been proved to be the best choice for modelling non-linear systems and implementing general – purpose non-linear controllers, due to their universal approximation capabilities. For example control and management of agriculture machinery.

(d) Optical character recognition

- Well known application using image recognition is the Optical Character Recognition (OCR) tools that are available with the standard scanning software for the home computer.
- **Scansoft** has had great success in combining NN with a rule based system for correctly recognizing both characters and words, to get a high level of accuracy.

(e) Customer relationship management

- Another popular application for NN is Customer Relationship Management (CRM).
- Customer Relationship Management requires key information to be derived from raw data collected for each individual customer. This can be achieved by building models using historical data information.
- Many companies are now using neural technology to help in their day to day business processes. They are doing this to achieve better performance, greater insight, faster development and increased productivity.
- By using Neural Networks for data mining in the databases, patterns, however complex, can be identified for the different types of customers, thus giving valuable customer information to the company.
- Also, NN could be useful for important tasks related to CRM, such as forecasting call centre loading, demand and sales levels, monitoring and analyzing the market, validating, completing and enhancing databases, clustering and profiling client base etc.
- One example is the airline reservation system AMT, which could predict sales of tickets in relation to destination, time of year and ticket price.

(f) Medical science

- Medicine is the field that has always taken benefits from the latest and advanced technologies.
- Artificial Neural Networks (ANN) is currently the next promising area of interest in medical science.
- It is believed that neural networks will have extensive application to biomedical problems in the next few years.
- ANN has already been successfully applied in medical applications such as diagnostic systems, bio chemical analysis, disease detection, image analysis and drug development.

1.15.2 Introduction to Fuzzy Logic

- **Fuzzy logic** is an approach to computing based on "**degrees of truth**" rather than the usual "true or false" (1 or 0) Boolean logic on which the modern computer is based.
- The idea of **fuzzy logic** was first proposed by Dr.Lotfi Zadeh of the University of California at Berkeley in the 1960s.
- The human brain can interpret imprecise, vague and incomplete information provided by sensory organs.
- Fuzzy logic is a powerful mathematical tool that can deal with such imprecise, incomplete and uncertain information present in complex real world problems.
- Using fuzzy logic, it is now possible to include vague human assessment in computing problems.
- Also, it provides an effective means for conflict resolution of multiple criteria and better assessment of options.
- Fuzzy logic can be used in the development of various applications such as pattern recognition, optimization, control applications, identification and any intelligent system for decision making.

A. Advantages of Fuzzy Logic Controllers

- simplicity and flexibility
- can handle problems with imprecise and incomplete data

- can model nonlinear functions of arbitrary complexity
- cheaper to develop,
- cover a wider range of operating conditions, more readily customizable in natural language terms.

B. Applications of Fuzzy Logic

- Fuzzy logic can be used in applications where human like decision making with an ability to generate precise solutions from certain or approximate information is required.
- Fuzzy logic has been extensively used in design of controllers for home appliances such as washing machine, vacuum cleaner, air conditioner etc.
- Fuzzy logic can also be used for other applications such as facial pattern recognition, anti-skid braking systems, transmission systems, control of subway systems and unmanned helicopters.
- Another application area of fuzzy logic is development of knowledge-based systems for multi objective optimization of power systems, weather forecasting systems, models for new product pricing or project risk assessment, medical diagnosis and treatment plans, and stock trading.
- Fuzzy logic has been successfully used in numerous fields such as control systems engineering, image processing, power engineering, industrial automation, robotics, consumer electronics and optimization.

1.15.3 Introduction to Genetic Algorithms

- Genetic Algorithms are ***adaptive heuristic search*** algorithms based on the evolutionary ideas of natural selection and genetics.
- GAs are often used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.
- GA can efficiently explore a large space of candidate designs and find optimum solutions.
- GAs are a subset of an Evolutionary Computation.
- GAs were developed by John Holland and his students and colleagues at the University of Michigan,
- In GAs, we select the initial pool or a population of possible solutions to the given problem.
- These solutions then undergo various GA operations like recombination and mutation which in turn produce new children.
- The process is repeated over various generations.
- Each individual or candidate solution is assigned a fitness value and the fitter individuals are given a higher chance to mate and yield more “fitter” individuals. This is in line with the Darwinian Theory of “Survival of the Fittest”.
- Thus GA keeps “evolving” better individuals or solutions over generations, till it reaches a stopping criterion.
- Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search.

A. Advantages of Genetic Algorithms

- GAs are easy to understand since they do not demand the knowledge of complex mathematics.
- Does not require any derivative information



- Good for noisy environment.
- Easy to discover global optimum.
- They can solve multimodal, non differentiable, non continuous or even NP-complete problems.
- GAs are inherently parallel and distributed.
- Provides a list of “good” solutions and not just a single solution.
- Flexible in forming building blocks for hybrid applications.
- Have substantial history and range of use.
- Useful when the search space is very large and there are a large number of parameters involved.

B. Applications of Genetic Algorithms

1. **Automotive design :** Genetic algorithms can be used to design composite materials and aerodynamic shapes for race cars to provide faster, lighter, more fuel efficient and safer vehicles for all the things we use vehicles for.
2. **Engineering design :** GA are most commonly used to optimize the structural and operational design of buildings, factories, machines, etc. GA s are used for optimizing the design of robot gripping arms, satellite booms, building trusses turbines, , flywheels or any other computer-aided engineering design application. 3.
3. **Robotics :** GAs have found applications that span the range of architectures for intelligent robotics. GAs can be used to design the entirely new types of robots that can perform multiple tasks and have more general application.

Review Questions

- Q. 1 Define artificial intelligence.
- Q. 2 Write a short note on: Applications of artificial intelligence.
- Q. 3 Explain the various artificial intelligence problems and artificial intelligence techniques.
- Q. 4 What is artificial intelligence ?
- Q. 5 What are the components of AI?
- Q. 6 What are the various AI techniques ?
- Q. 7 Explain various applications of Artificial Intelligence.
- Q. 8 Explain PEAS representation with example.
- Q. 9 Define agent and give classification of agents.
- Q. 10 What is intelligent agent ?
- Q. 11 Write a short note on: Rational agent.
- Q. 12 Write a short note on : Structure of intelligent agents.
- Q. 13 Give types of agents.



Q. 14 What are various agent environments ? Give PEAS representation for an agent.

Q. 15 Define in your own words, the following terms :

- | | |
|------------------|-------------------|
| 1. Agent | 2. Agent function |
| 3. Agent program | 4. Autonomy |

Q. 16 Explain various types of intelligent agents, state limitations of each and how it is overcome in other type of agent.

Q. 17 What do you mean by PEAS? Explain properties of task environment.

Q. 18 Explain detail architecture of goal based agent.

Q. 19 Explain Simple reflex agent architecture.

Q. 20 Explain learning agent architecture.

Q. 21 What are the constituents of Soft Computing ? Explain each in brief.

Q. 22 Differentiate between hard computing and soft computing.

Q. 23 Explain the basics of genetic algorithm along with its applications.

Q. 24 What are the applications of neural networks ?

□□□



Problem Solving

Syllabus

- 2.1 Problem Solving Agent, Formulating Problems, Example Problems
- 2.2 Uninformed Search Methods : Depth Limited Search, Depth First Iterative Deepening (DFID), Informed Search Method : A* Search
- 2.3 Optimization Problems : Hill climbing Search, Simulated annealing, Genetic algorithm

Search is an indivisible part of intelligence. An intelligent agent is the one who can search and select the most appropriate action in the given situation, among the available set of actions. When we play any game like chess, cards, tic-tac-toe, etc.; we know that we have multiple options for next move, but the intelligent one who searches for the correct move will definitely win the game. In case of travelling salesman problem, medical diagnosis system or any expert system; all they required to do is to carry out search which will produce the optimal path, the shortest path with minimum cost and efforts. Hence, this chapter focuses on the searching techniques used in AI applications. Those are known as un-informed and informed search techniques.

2.1 Solving Problems by Searching

- Now let us see how searching play a vital role in solving AI problems. Given a problem, we can generate all the possible states it can have in real time, including start state and end state. To generate solution for the same is nothing but searching a path from start state to end state.
- Problem solving agent is the one who finds the goal state from start state in optimal way by following the shortest path, thereby saving the memory and time. It's supposed to maximize its performance by fulfilling all the performance measures.
- Searching techniques can be used in game playing like **Tic-Tac-Toe** or navigation problems like **Travelling Salesman Problem**.
- First, we will understand the representation of given problem so that appropriate searching techniques can be applied to solve the problem.

2.2 Formulating Problems

MU - Dec. 12

Q. Explain how you will formulate search problem.

(Dec. 12, 3 Marks)

- Given a goal to achieve; problem formulation is the process of deciding what states to be considered and what actions to be taken to achieve the goal. This is the first step to be taken by any problem solving agent.
- **State space** : The state space of a problem is the set of all states reachable from the initial state by executing any sequence of actions. State is representation of all possible outcomes.
- The state space specifies the relation among various problem states thereby, forming a directed network or graph in which the nodes are states and the links between nodes represent actions.



- **State Space Search :** Searching in a given space of states pertaining to a problem under consideration is called a state space search.
- **Path :** A path is a sequence of states connected by a sequence of actions, in a given state space.

2.2.1 Components of Problems Formulation

MU - May 15**Q.** Explain steps in problem formulation with example.**(May 15, 10 Marks)**

Problem can be defined formally using five components as follows :

- | | |
|-----------------------|--------------|
| 1. Initial state | 2. Actions |
| 3. Successor function | 4. Goal test |
| 5. Path cost | |

1. **Initial state :** The initial state is the one in which the agent starts in.
2. **Actions :** It is the set of actions that can be executed or applicable in all possible states. A description of what each action does; the formal name for this is the transition model.
3. **Successor function :** It is a function that returns a state on executing an action on the current state.
4. **Goal test :** It is a test to determine whether the current state is a goal state. In some problems the goal test can be carried out just by comparing current state with the defined goal state, called as **explicit goal test**. Whereas, in some of the problems, state cannot be defined explicitly but needs to be generated by carrying out some computations, it is called as **implicit goal test**.

For example : In Tic-Tac-Toe game making diagonal or vertical or horizontal combination declares the winning state which can be compared explicitly; but in the case of chess game, the goal state cannot be predefined but it's a scenario called as "Checkmate", which has to be evaluated implicitly.

5. **Path cost :** It is simply the cost associated with each step to be taken to reach to the goal state. To determine the cost to reach to each state, there is a cost function, which is chosen by the problem solving agent.

Problem solution : A well-defined problem with specification of initial state, goal test, successor function, and path cost. It can be represented as a data structure and used to implement a program which can search for the goal state. A solution to a problem is a sequence of actions chosen by the problem solving agent that leads from the initial state to a goal state. Solution quality is measured by the path cost function.

Optimal solution : An optimal solution is the solution with least path cost among all solutions.

A general sequence followed by a simple problem solving agent is, first it formulates the problem with the goal to be achieved, then it searches for a sequence of actions that would solve the problem, and then executes the actions one at a time.

2.2.2 Example Problems

MU - Dec. 12**Q.** Formulate 8-puzzle problem.**(Dec. 12, 3 Marks)**

1. Example of 8-puzzle problem

- Fig. 2.2.1 depicts a typical scenario of 8-puzzle problem. It has a 3×3 board with tiles having 1 through 8 numbers on it. There is a blank tile which can be moved forward, backward, to left and to right. The aim is to arrange all the tiles in the goal state form by moving the blank tile minimum number of times.

1	2	3
4	8	-
7	6	5

1	2	3
4	5	6
7	8	-

Initial State Goal State

Fig. 2.2.1 : A scenario of 8-Puzzle Problem

- This problem can be formulated as follows :

States : States can be represented by a 3×3 matrix data structure with blank denoted by 0.

1. **Initial state** : $\{\{1, 2, 3\}, \{4, 8, 0\}, \{7, 6, 5\}\}$
2. **Actions** : The blank space can move in Left, Right, Up and Down directions specifying the actions.
3. **Successor function** : If we apply “Down” operator to the start state in Fig. 2.2.1, the resulting state has the 5 and the blank switching their positions.
4. **Goal test** : $\{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 0\}\}$
5. **Path cost** : Number of steps to reach to the final state.

Solution :

$\{\{1, 2, 3\}, \{4, 8, 0\}, \{7, 6, 5\}\} \rightarrow \{\{1, 2, 3\}, \{4, 8, 5\}, \{7, 6, 0\}\} \rightarrow \{\{1, 2, 3\},$

$\{4, 8, 5\}, \{7, 0, 6\}\} \rightarrow \{\{1, 2, 3\}, \{4, 0, 5\}, \{7, 8, 6\}\} \rightarrow \{\{1, 2, 3\}, \{4, 5, 0\}, \{7, 8, 6\}\} \rightarrow \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 0\}\}$

Path cost = 5 steps

2. Example of missionaries and cannibals problem

- The problem statement as discussed in the previous section. Let's formulate the problem first.
 - **States** : In this problem, state can be data structure having triplet (i, j, k) representing the number of missionaries, cannibals, and canoes on the left bank of the river respectively.
1. **Initial state** : It is $(3, 3, 1)$, as all missionaries, cannibals and canoes are on the left bank of the river.
 2. **Actions** : Take x number of missionaries and y number of cannibals
 3. **Successor function** : If we take one missionary, one cannibal the other side of the river will have two missionaries and two cannibals left.
 4. **Goal test** : Reached state $(0, 0, 0)$
 5. **Path cost** : Number of crossings to attain the goal state.

Solution :

The sequence of actions within the path :

$(3, 3, 1) \rightarrow (2, 2, 0) \rightarrow (3, 2, 1) \rightarrow (3, 0, 0) \rightarrow (3, 1, 1) \rightarrow (1, 1, 0) \rightarrow (2, 2, 1) \rightarrow (0, 2, 0) \rightarrow (0, 3, 1) \rightarrow (0, 1, 0) \rightarrow (0, 2, 1) \rightarrow (0, 0, 0)$

Cost = 11 crossings

3. Vacuum-cleaner problem

States : In vacuum cleaner problem, state can be represented as [**block**, clean] or [**block**, dirty]. The agent can be in one of the two blocks which can be either clean or dirty. Hence there are total 8 states in the vacuum cleaner world.

1. **Initial State** : Any state can be considered as initial state. For example, [A, dirty]
2. **Actions** : The possible actions for the vacuum cleaner machine are left, right, absorb, idle.

3. **Successor function :** Fig. 2.2.2 indicating all possible states with actions and the next state.

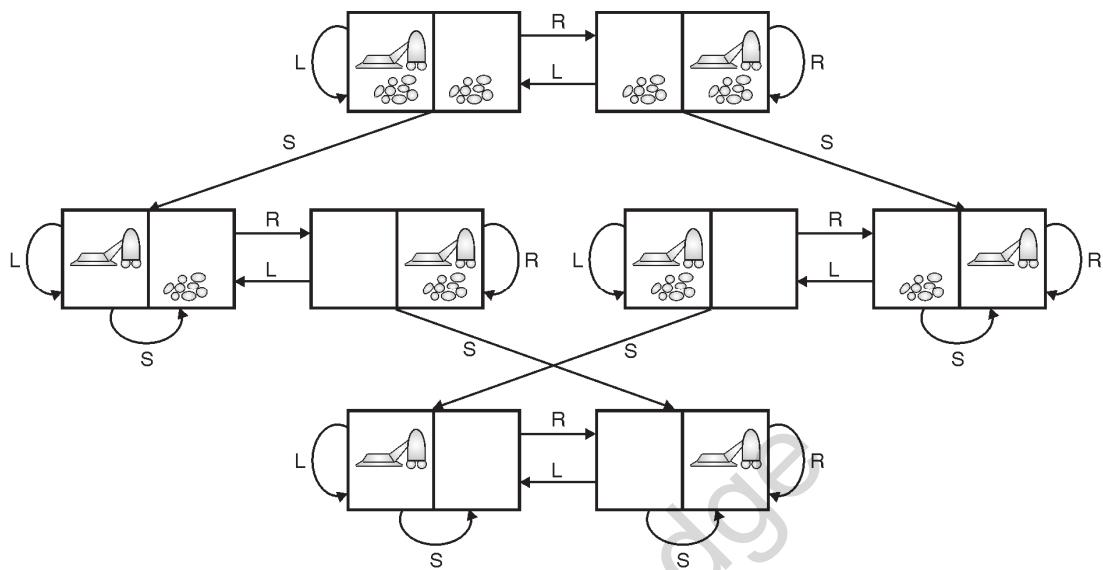


Fig. 2.2.2 : The state space for vacuum world

4. **Goal Test :** The aim of the vacuum cleaner is to clean both the blocks. Hence the goal test if [A, Clean] and [B, Clean].
5. **Path Cost :** Assuming that each action/ step costs 1 unit cost. The path cost is number of actions/ steps taken.

4. Example of real time problems

- There are varieties of real time problems that can be formulated and solved by searching. Robot Navigation, Rout Finding Problem, Traveling Salesman Problem (TSP), VLSI design problem, Automatic Assembly Sequencing, etc. are few to name.
- There are number of applications for route finding algorithms. Web sites, car navigation systems that provide driving directions, routing video streams in computer networks, military operations planning, and airline travel-planning systems are few to name. All these systems involve detailed and complex specifications.
- For now, let us consider a problem to be solved by a travel planning web site; the airline travel problem.
- **State :** State is represented by airport location and current date and time. In order to calculate the path cost state may also record more information about previous segments of flights, their fare bases and their status as domestic or international.

1. **Initial state :** This is specified by the user's query, stating initial location, date and time.
2. **Actions :** Take any flight from the current location, select seat and class, leaving after the current time, leaving enough time for within airport transfer if needed.
3. **Successor function :** After taking the action i.e. selecting fight, location, date, time; what is the next location date and time reached is denoted by the successor function. The location reached is considered as the current location and the flight's arrival time as the current time.
4. **Goal test :** Is the current location the destination location?
5. **Path cost :** In this case path cost is a function of monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards and so on.

2.3 Measuring Performance of Problem Solving Algorithm / Agent

There are variety of problem solving methods and algorithms available in AI. Before studying any of these algorithms in detail, let's consider the criteria to judge the efficiency of those algorithms. The performance of all these algorithms can be evaluated on the basis of following factors.

1. **Completeness** : If the algorithm is able to produce the solution if one exists then it satisfies completeness criteria.
2. **Optimality** : If the solution produced is the minimum cost solution, the algorithm is said to be optimal.
3. **Time complexity** : It depends on the time taken to generate the solution. It is the number of nodes generated during the search.
4. **Space complexity** : Memory required to store the generated nodes while performing the search.

Complexity of algorithms is expressed in terms of three quantities as follows :

1. **b** : Called as branching factor representing maximum number of successors a node can have in the search tree.
2. **d** : Stands for depth of the shallowest goal node.
3. **m** : It is the maximum depth of any path in the search tree.

2.4 Node Representation in Search Tree

- In order to carry out search, first we need to build the search tree. The nodes are the various possible states in the state space.
- The connectors are the indicators of which all states are directly reachable from current state, based on the successor function.
- Thus the parent child relation is build and the search tree can be generated. Fig. 2.4.1 shows the representation of a tree node as a data structure in 8-puzzle problem.

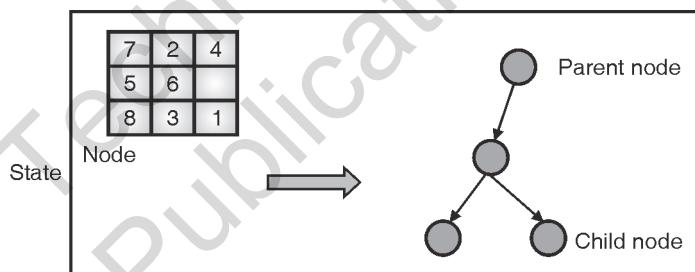


Fig. 2.4.1 : Node representation of state in searching

- Node is the data structure from which the search tree is constructed. Each node has a parent, a state, and children nodes directly reachable from that node.
- For each node of the tree, we can have following structure components :
 1. **State / Value** : The state in the state space to which the node corresponds or value assigned to the node.
 2. **Parent node** : The node in the search tree that generated this node.
 3. **Number of children** : Indicating number of actions that can be taken to generate next states (children nodes).
 4. **Path cost** : The cost of the path from the initial state to the node.

2.5 Uninformed Search

MU - Dec. 14

Q. Write short note on Uniform search.

(MU - Dec. 14, 2.5 Marks)

- Why is it called uninformed search? What is not been informed about the search?

- The term “uninformed” means they have only information about what is the start state and the end state along with the problem definition.
- These techniques can generate successor states and can distinguish a goal state from a non-goal state.
- All these search techniques are distinguished by the order in which nodes are expanded.
- The uninformed search techniques also called as “blind search”.

2.6 Depth First Search (DFS)

MU - Dec. 12

Q. Explain Depth-first search using suitable example.

(Dec. 12, 4 Marks)

2.6.1 Concept

- In depth-first search, the search tree is expanded depth wise; i.e. the deepest node in the current branch of the search tree as expanded. As the leaf node is reached, the search backtracks to previous node. The progress of the search is illustrated in Fig.2.6.1.
- The explored nodes are shown in light gray. Explored nodes with no descendants in the fringe are removed from memory. Nodes at depth three have no successors and M is the only goal node.

Process

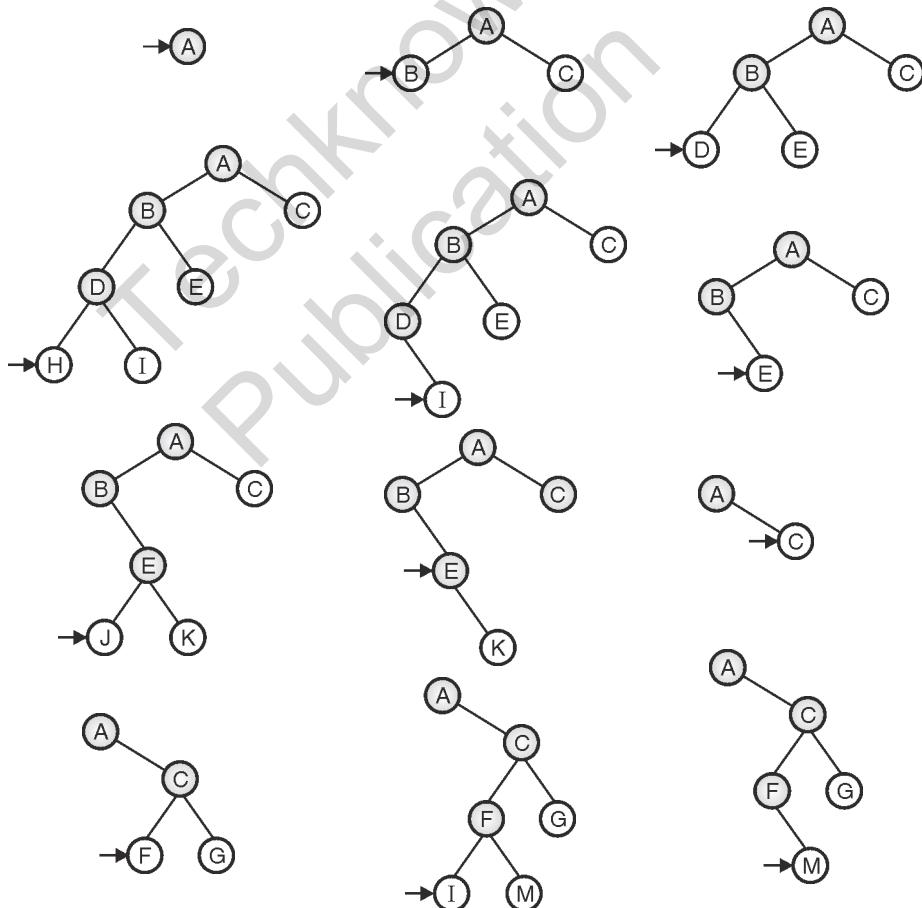


Fig. 2.6.1 : Working of Depth first search on a binary tree

2.6.2 Implementation

DFS uses a LIFO fringe i.e. stack. The most recently generated node, which is on the top in the fringe, is chosen first for expansion. As the node is expanded, it is dropped from the fringe and its successors are added. So when there are no more successors to add to the fringe, the search “back tracks” to the next deepest node that is still unexplored. DFS can be implemented in two ways, recursive and non-recursive. Following is the algorithm for the same.

2.6.3 Algorithm

(a) Non recursive implementation of DFS

1. Push the root node on a stack
2. while (stack is not empty)
 - (a) Pop a node from the stack;
 - (i) if node is a goal node then return success;
 - (ii) push all children of node onto the stack;
3. return failure

(b) Recursive implementation of DFS

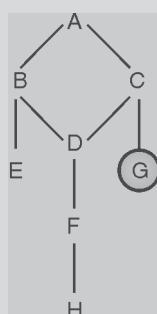
DFS(c) :

1. If node is a goal, return success;
2. for each child c of node
 - (a) if DFS(c) is successful,
 - (i) return success
3. return failure;

2.6.4 Performance Evaluation

- **Completeness** : Complete, if m is finite.
- **Optimality** : No, as it cannot guarantee the shallowest solution.
- **Time Complexity** : A depth first search, may generate all of the $O(b^m)$ nodes in the search tree, where m is the maximum depth of any node; this can be much greater than the size of the state space.
- **Space Complexity** : For a search tree with branching factor b and maximum depth m, depth first search requires storage of only $O(b^m)$ nodes, as at a time only the branch, which is getting explored, will reside in memory.

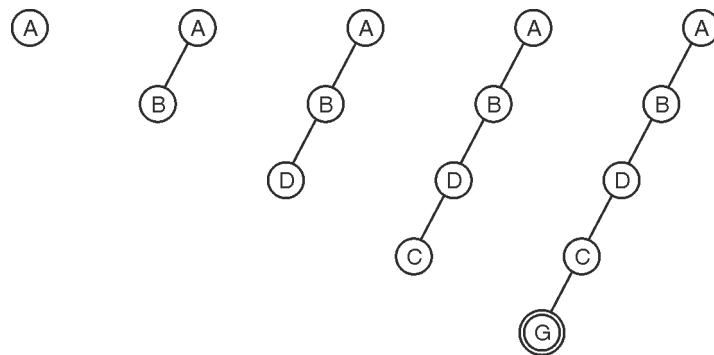
Ex. 2.6.1 : Consider following graph.



Starting from state A execute DFS. The goal node is G. Show the order in which the nodes are expanded.

Assume that the alphabetically smaller node is expanded first to break ties.

MU - May 16, 10 Marks

Soln. :**Fig. P. 2.6.1**

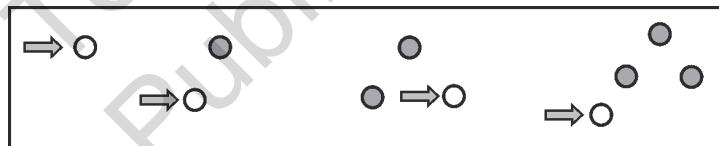
2.7 Breadth First Search (BFS)

MU - May 14**Q.** Explain breadth first algorithm.**(May 14, 10 Marks)**

2.7.1 Concept

- As the name suggests, in breadth-first search technique, the tree is expanded breadth wise.
- The root node is expanded first, then all the successors of the root node are expanded, then their successors, and so on.
- In turn, all the nodes at a particular depth in the search tree are expanded first and then the search will proceed for the next level node expansion.
- Thus, the shallowest unexpanded node will be chosen for expansion. The search process of BFS is illustrated in Fig. 2.7.1.

2.7.2 Process

**Fig. 2.7.1 : Working of BFS on binary tree**

2.7.3 Implementation

- In BFS we use a FIFO queue for the fringe. Because of which the newly inserted nodes in the fringe will automatically be placed after their parents.
- Thus, the children nodes, which are deeper than their parents, go to the back of the queue, and old nodes, which are shallower, get expanded first. Following is the algorithm for the same.

2.7.4 Algorithm

1. Put the root node on a queue
2. while (queue is not empty)
 - (a) remove a node from the queue
 - (i) if (node is a goal node) return success;
 - (ii) put all children of node onto the queue;
3. return failure;



2.7.5 Performance Evaluation

- **Completeness :** It is complete, provided the shallowest goal node is at some finite depth.
- **Optimality :** It is optimal, as it always finds the shallowest solution.
- **Time complexity :** $O(b^d)$, number of nodes in the fringe.
- **Space complexity :** $O(b^d)$, total number of nodes explored.

2.8 Uniform Cost Search (UCS)

2.8.1 Concept

- Uniform cost search is a breadth first search with all paths having same cost. To make it work in real time conditions we can have a simple extension to the basic implementation of BFS. This results in an algorithm that is optimal with any path cost.
- In BFS as we always expand the shallowest node first; but in uniform cost search, instead of expanding the shallowest node, the node with the lowest path cost will be expanded first. The implementation details are as follow.

2.8.2 Implementation

- Uniform cost search can be achieved by implementing the fringe as a priority queue ordered by path cost. The algorithm shown below is almost same as BFS; except for the use of a priority queue and the addition of an extra check in case a shorter path to any node is discovered.
- The algorithm takes care of nodes which are inserted in the fringe for exploration, by using a data structure having priority queue and hash table.
- The priority queue used here contains total cost from root to the node. Uniform cost search gives the minimum path cost the maximum priority. The algorithm using this priority queue is the following.

2.8.3 Algorithm

- Insert the root node into the queue.
- While the queue is not empty :
 - (i) Dequeue the maximum priority node from the queue.
(If priorities are same, alphabetically smaller node is chosen)
 - (ii) If the node is the goal node, print the path and exit.
Else
- Insert all the children of the dequeued node, with their total costs as priority.
- The algorithm returns the best cost path which is encountered first and will never go for other possible paths. The solution path is optimal in terms of cost.
- As the priority queue is maintained on the basis of the total path cost of node, the algorithm never expands a node which has a cost greater than the cost of the shortest path in the tree.
- The nodes in the priority queue have almost the same costs at a given time, and thus the name “Uniform Cost Search”.

2.8.4 Performance Evaluation

- **Completeness** : Completeness is guaranteed provided the cost of every step exceeds some small positive constant.
- **Optimality** : It produces optimal solution as nodes are expanded in order of their path cost.
- **Time complexity** : Uniform-cost search considers path costs rather than depths; so its complexity is does not merely depends on b and d. Hence we consider C^* be the cost of the optimal solution, and assume that every action costs at least ϵ . Then the algorithm's worst-case time and space complexity is $O(b^{C^*/\epsilon})$, which can be much greater than bd.
- **Space complexity** : $O(b^{C^*/\epsilon})$, indicating number of node in memory at execution time.

2.9 Depth Limited Search (DLS)

2.9.1 Concept

- In order to avoid the infinite loop condition arising in DFS, in depth limited search technique, depth-first search is carried out with a predetermined depth limit.
- The nodes with the specified depth limit are treated as if they don't have any successors. The depth limit solves the infinite-path problem.
- But as the search is carried out only till certain depth in the search tree, it introduces problem of incompleteness.
- Depth-first search can be viewed as a special case of depth-limited search with depth limit equal to the depth of the tree. The process of DLS is depicted in Fig. 2.9.1.

2.9.2 Process

If depth limit is fixed to 2, DLS carries out depth first search till second level in the search tree.

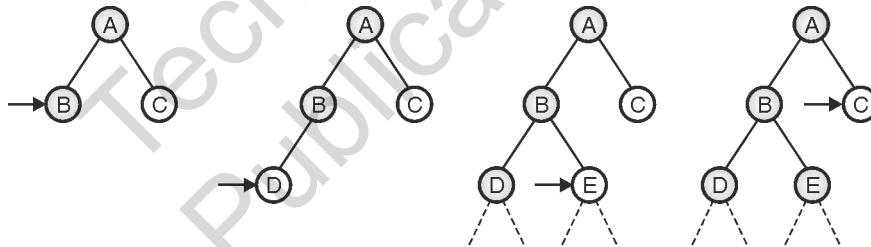


Fig. 2.9.1 : DIS working with depth limit

2.9.3 Implementation

- As in case of DFS in DLS we can use the same fringe implemented as queue.
- Additionally the level of each node needs to be calculated to check whether it is within the specified depth limit. Depth-limited search can terminate with two conditions :
 1. If the solution is found.
 2. If there is no solution within given depth limit.

2.9.4 Algorithm

- Determine the start node and the search depth.
- Check if the current node is the goal node



- If not : Do nothing
- If yes : return
- Check if the current node is within the specified search depth
 - If not : Do nothing
 - If yes : Expand the node and save all of its successors in a stack.
- Call DLS recursively for all nodes of the stack and go back to Step 2.

2.9.5 Pseudo Code

```
booleanDLS(Node node, int limit, intdepth)
{
    if (depth > limit) return failure;
    if (node is a goal node) return success;
    for each child of node
    {
        if (DLS(child, limit, depth + 1))
            return success;
    }
    return failure;
}
```

2.9.6 Performance Evaluation

- **Completeness** : Its incomplete if shallowest goal is beyond the depth limit.
- **Optimality** : Non optimal, as the depth chosen can be greater than d.
- **Time complexity** : Same as DFS, $O(b^l)$, where l is the specified depth limit.
- **Space complexity** : Same as DFS, $O(b^l)$, where l is the specified depth limit.

2.10 Depth First Iterative Deepening (DFID)

2.10.1 Concept

- Iterative deepening depth first search is a combination of BFS and DFS. In DFID search happens depth wise but, at a time the depth limit will be incremented by one. Hence iteratively it deepens down in the search tree.
- It eventually turns out to be the breadth-first search as it explores a complete layer of new nodes at each iteration before going on to the next layer.
- It does this by gradually increasing the depth limit-first 0, then 1, then 2, and so on-until a goal is found; and thus guarantees the optimal solution. Iterative deepening combines the benefits of depth-first and breadth-first search.
- The search process is depicted in Fig. 2.10.1.

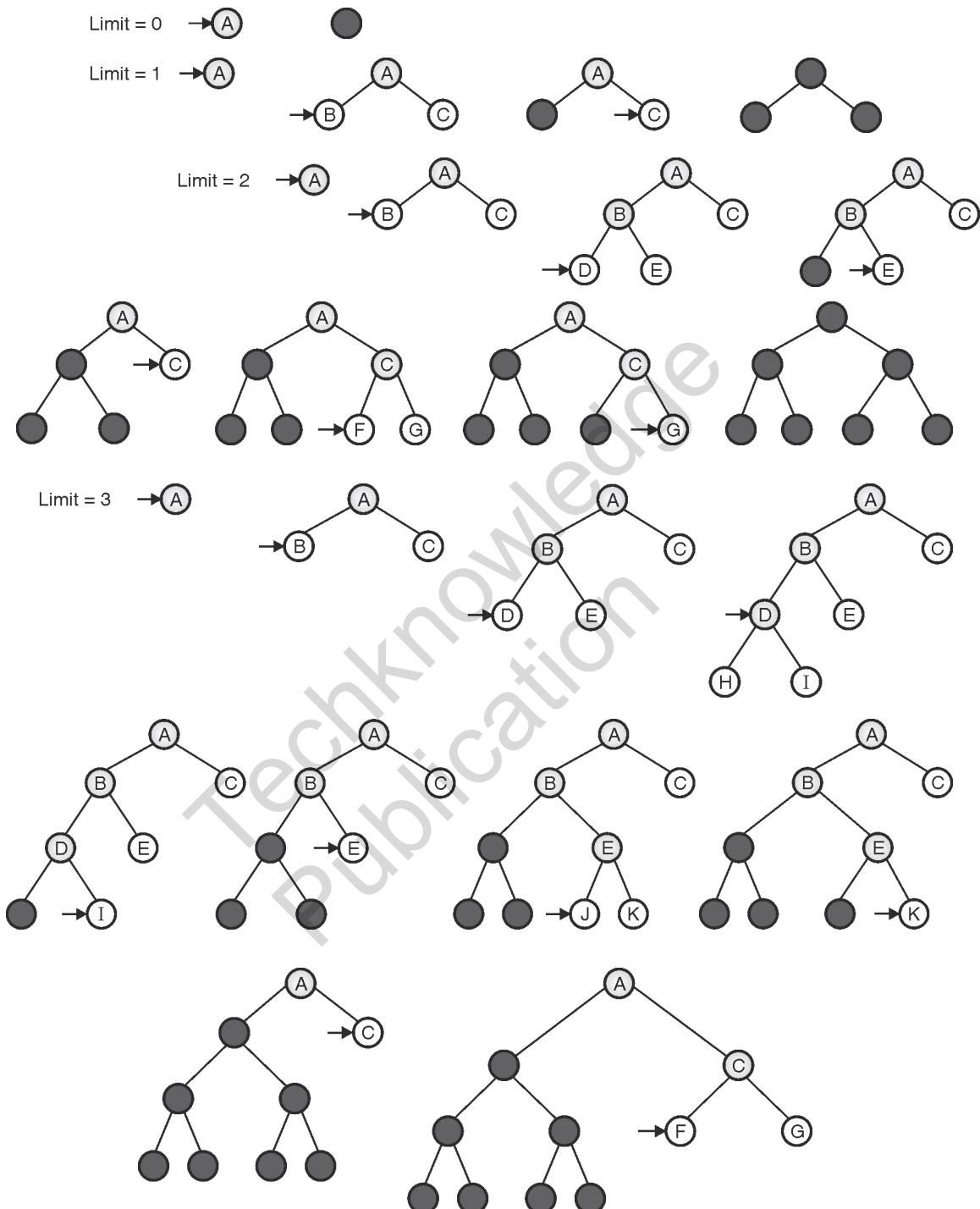


Fig.2.10.1 : Search process in DFID (contd...)

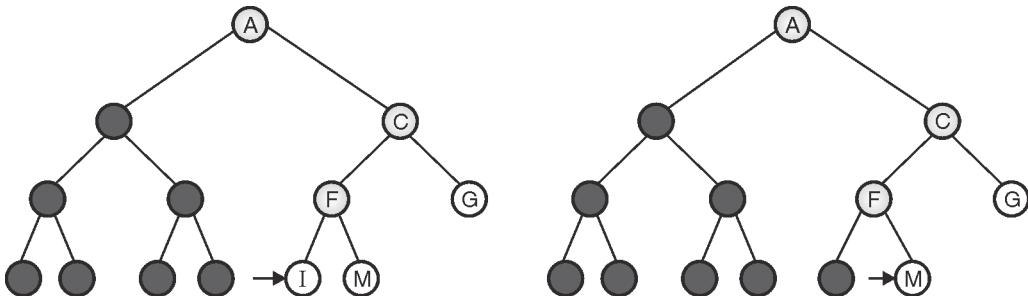


Fig. 2.10.1 : Search process in DFID

- Fig. 2.10.1 shows four iterations of on a binary search tree, where the solution is found on the fourth iteration.

2.10.2 Process

```

function 1iterative : Depending search (problem) returns a solution, or failure
for depth = 0 to  $\infty$  do
    result  $\leftarrow$  Depth – Limited – Search (problem, depth)
    if result  $\neq$  cutoff then return result
  
```

Fig. 2.10.1 the iterative depending search algorithm, which repeatedly applies depth limited search with increasing limits. It terminates when a solution is found or if the depth limited search returns failure, meaning that no solution exists.

2.10.3 Implementation

It has exactly the same implementation as that of DLS. Additionally, iterations are required to increment the depth limit by one in every recursive call of DLS.

2.10.4 Algorithm

- Initialize depth limit to zero.
- Repeat Until the goal node is found.
 - (a) Call Depth limited search with new depth limit.
 - (b) Increment depth limit to next level.

2.10.5 Pseudo Code

```

DFID()
{
    limit = 0;
    found = false;
    while (not found)
    {
        found = DLS(root, limit, 0);
        limit = limit + 1;
    }
}
  
```

2.10.6 Performance Evaluation

- **Completeness :** DFID is complete when the branching factor b is finite.
- **Optimality :** It is optimal when the path cost is a non-decreasing function of the depth of the node.
- **Time complexity :**
 - o Do you think in DFID there is a lot of wastage of time and memory in regenerating the same set of nodes again and again ?
 - o It may appear to be waste of memory and time, but it's not so. The reason is that, in a search tree with almost same branching factor at each level, most of the nodes are in the bottom level which are explores very few times as compared to those on upper level.
 - o The nodes on the bottom level that is level ' d ' are generated only once, those on the next to bottom level are generated twice, and so on, up to the children of the root, which are generated d times. Hence the time complexity is $O(b^d)$.
- **Space complexity :** Memory requirements of DFID are modest, i.e. $O(b^d)$.

Note : As the performance evaluation is quite satisfactory on all the four parameters, DFID is the preferred uninformed search method when the search space is large and the depth of the solution is not known.

2.11 Bidirectional Search

2.11.1 Concept

In bidirectional search, two simultaneous searches are run. One search starts from the initial state, called forward search and the other starts from the goal state, called backward search. The search process terminates when the searches meet at a common node of the search tree. Fig. 2.11.1 shows the general search process in bidirectional search.

2.11.2 Process

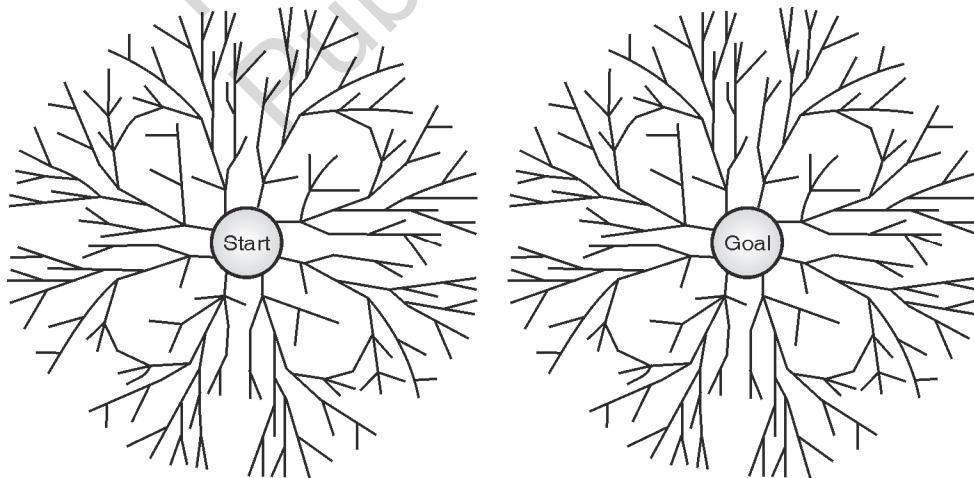


Fig. 2.11.1 : Search process in bidirectional search

2.11.3 Implementation

- In Bidirectional search instead of checking for goal node, one need to check whether the fringes of the two searches intersect; as they do, a solution has been found.



- When each node is generated or selected for expansion, the check can be done. It can be implemented with a hash table, to guarantee constant time.
- For example, consider a problem which has solution at depth $d= 6$. If we run breadth first search in each direction, then in the worst case the two searches meet when they have generated all of the nodes at depth 3. If $b= 10$.
- This requires a total of 2,220 node generations, as compared with 1,111,110 for a standard breadth-first search.

2.11.4 Performance Evaluation

- **Completeness** : Yes, if branching factor b is finite and both directions use breadth first search.
- **Optimality** : Yes, if all costs are identical and both directions use breadth first search.
- **Time complexity** : Time complexity of bidirectional search using breadth-first searches in both directions is $O(b^{d/2})$.
- **Space complexity** : As at least one of the two fringes need to kept in memory to check for the common node, the space complexity is $O(b^{d/2})$.

2.11.5 Pros of Bidirectional Search

- It is much more efficient.
- Reduces space and time requirements as, we perform two $b^{d/2}$ searches, instead of one b^d search.
- **Example :**
 - o Suppose $b = 10$, $d = 6$. Breadth first search will examine $10^6 = 1,000,000$ nodes.
 - o Bidirectional search will examine $2 \times 10^3 = 2,000$ nodes.
- One can combine different search strategies in different directions to avail better performance.

2.11.6 Cons of Bidirectional Search

- The search requires generating predecessors of states.
- Overhead of checking whether each new node appears in the other search is involved.
- For large d , is still impractical!
- For two bi-directional breadth-first searches, with branching factor b and depth of the solution d we have memory requirement of $b^{d/2}$ for each search.

2.12 Comparing Different Techniques

MU - May 13, Dec. 14

Q. Compare different uninformed search strategies.

(May 13, Dec. 14, 10 Marks)

Table 2.12.1 depicts the comparison of all uninformed search techniques basis on their performance evaluation. As we know, the algorithms are evaluated on four criteria viz. completeness, optimality, time complexity and space complexity.

The notations used are as follows :

- b : Branching factor
- d : Depth of the shallowest solution
- m : Maximum depth of the search tree
- l : Depth limit

**Table 2.12.1 : Comparison of tree-search strategies basis on performance Evaluation**

Parameters	BFS	Uniform Cost	DFS	DLS	DFID	Bidirectional
Completeness	Yes	Yes	No	No	Yes	Yes
Optimality	Yes	Yes	No	No	Yes	Yes
Time Complexity	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space Complexity	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$

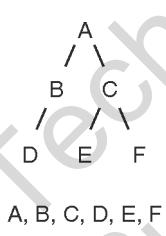
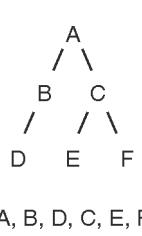
2.12.1 Difference between Unidirectional and Bidirectional Search

Sr. No.	Unidirectional search method	Bidirectional search method
1.	These methods use search tree, start node and goal node as input for starting search.	These methods have additional information about the search tree nodes, along with the start and goal node.
2.	They use only the information from the problem definition.	They incorporate additional measure of a potential of a specific state to reach the goal.
3.	Sometimes these methods use past explorations, e.g. cost of the path generated so far.	All these methods use a potential of a state (node) to reach a goal is measured through heuristic function.
4.	All unidirectional techniques are based on the pattern of exploration of nodes in the search tree.	All bidirectional search techniques totally depend on the evaluated value of each node generated by heuristic function.
5.	In real time problems uninformed search techniques can be costly with respect to time and space.	In real time problems informed search techniques are cost effective with respect to time and space.
6.	Comparatively more number of nodes will be explored in these methods.	As compared to uninformed techniques less number of nodes are explored in this case.
7.	Example : Breadth First Search Depth First search Uniform Cost search, Depth Limited search, Iterative Deepening DFS	Example : Hill Climbing search Best First search, A* Search, IDA* search, SMA* search

2.12.2 Difference between BFS and DFS

Sr. No.	BFS	DFS
1.	BFS Stands for “Breadth First Search”.	DFS stands for “Depth First Search”.
2.	BFS traverses the tree level wise.i.e. each node near to root will be visited first. The nodes are explored left to right.	DFS traverses tree depth wise. i.e. nodes in particular branch are visited till the leaf node and then search continues branch by branch from left to right in the tree.



Sr. No.	BFS	DFS
3.	Breadth First Search is implemented using queue which is FIFO list.	Depth First Search is implemented using Stack which is LIFO list.
4.	This is a single step algorithm, wherein the visited vertices are removed from the queue and then displayed at once.	This is two step algorithm. In first stage, the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped out.
5.	BFS requires more memory compare to DFS.	DFS require less memory compare to BFS.
6.	Applications of BFS : To find Shortest path Single Source & All pairs shortest paths In Spanning tree In Connectivity	Applications of DFS : Useful in Cycle detection In Connectivity testing Finding a path between V and W in the graph. Useful in finding spanning trees and forest.
7.	BFS always provides the shallowest path solution.	DFS does not guarantee the shallowest path solution.
8.	No backtracking is required in BFS.	Backtracking is implemented in DFS.
9.	BFS is optimal and complete if branching factor is finite.	DFS is neither complete nor optimal even in case of finite branching factor.
10.	BFS can never get trapped into infinite loops.	DFS generally gets trapped into infinite loops, as search trees are dense.
11.	Example :  A, B, C, D, E, F	Example :  A, B, D, C, E, F

2.13 Informed Search Techniques

MU - Dec. 14

Q. Write short note on Inform search.

(Dec. 14, 2.5 Marks)

- Informed searching techniques is a further extension of basic un-informed search techniques. The main idea is to generate additional information about the search state space using the knowledge of problem domain, so that the search becomes more intelligent and efficient. The evaluation function is developed for each state, which quantifies the desirability of expanding that state in order to reach the goal.
- All the strategies use this evaluation function in order to select the next state under consideration, hence the name "Informed Search". These techniques are very much efficient with respect to time and space requirements as compared to uninformed search techniques.

2.14 Heuristic Function

MU - Dec. 12, Dec. 13, Dec. 14, May 15, Dec. 15

- | | |
|--|--|
| Q. Explain heuristic function with example. | (Dec. 12, Dec. 14, May 15, 5 Marks) |
| Q. What is heuristics function ? How will you find suitable heuristic function ? Give suitable example. (Dec. 13, 10 Marks) | |
| Q. Define heuristic function. | (Dec. 15, 2 Marks) |

- A heuristic function is an **evaluation function**, to which the search state is given as input and it generates the tangible representation of the state as output.
- It maps the problem state description to measures of desirability, usually represented as number weights. The value of a heuristic function at a given node in the search process gives a good estimate of that node being on the desired path to solution.
- It evaluates individual problem state and determines how much promising the state is. Heuristic functions are the most common way of imparting additional knowledge of the problem states to the search algorithm. Fig. 2.14.1 shows the general representation of heuristic function.



Fig. 2.14.1 : General representation of Heuristic function

- The representation may be the approximate cost of the path from the goal node or number of hopes required to reach to the goal node, etc.
- The heuristic function that we are considering in this syllabus, for a node n is, $h(n) = \text{estimated cost of the cheapest path from the state at node } n \text{ to a goal state.}$
- **Example :** For the Travelling Salesman Problem, the sum of the distances traveled so far can be a simple heuristic function.
- Heuristic function can be of two types depending on the problem domain. It can be a **Maximization Function** or **Minimization function** of the path cost.
- In maximization types of heuristic, greater the cost of the node, better is the node while; in case of minimization heuristic, lower is the cost, better is the node. There are heuristics of every general applicability as well as domain specific. The search strategies are general purpose heuristics.
- It is believed that in general, a heuristic will always lead to faster and better solution, even though there is no guarantee that it will never lead in the wrong direction in the search tree.
- Design of heuristic plays a vital role in performance of search.
- As the purpose of a heuristic function is to guide the search process in the most profitable path among all that are available; a well designed heuristic functions can provides a fairly good estimate of whether a path is good or bad.
- However in many problems, the cost of computing the value of a heuristic function would be more than the effort saved in the search process. Hence generally there is a trade-off between the cost of evaluating a heuristic function and the savings in search that the function provides.
- So, are you ready to think of your own heuristic function definitions? Here is the word of caution. See how the function definition impact.
- Following are the examples demonstrate how design of heuristic function completely alters the scenario of searching process.

2.14.1 Example of 8-puzzle Problem

- Remember 8-puzzle problem? Can we estimate the number of steps required to solve an 8-puzzle from a given state?? What about designing a heuristic function for it?



Fig. 2.14.2 : A scenario of 8-puzzle problem

- Two simple heuristic functions are :
 - h_1 = the number of misplaced tiles. This is also known as the Hamming Distance. In the Fig. 2.14.2 example, the start state has $h_1 = 8$. Clearly, h_1 is an acceptable heuristic because any tile that is out of place will have to be moved at least once, quite logical. Isn't it?
 - h_2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot be moved diagonally, the distance counted is the sum of horizontal and vertical distances. This is also known as the Manhattan Distance. In the Fig. 2.14.2, the start state has $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$. Clearly, h_2 is also an admissible heuristic because any move can, at best, move one tile one step closer to the goal.
- As expected, neither heuristic overestimates the true number of moves required to solve the puzzle, which is 26 ($h_1 + h_2$). Additionally, it is easy to see from the definitions of the heuristic functions that for any given state, h_2 will always be greater than or equal to h_1 . Thus, we can say that h_2 dominates h_1 .

2.14.2 Example of Block World Problem

MU - Dec. 15

Q. Give an example heuristics function for block world problem.

(Dec. 15, 3 Marks)

Q. Find the heuristics value for a particular state of the blocks world problem.

(Dec. 15, 5 Marks)

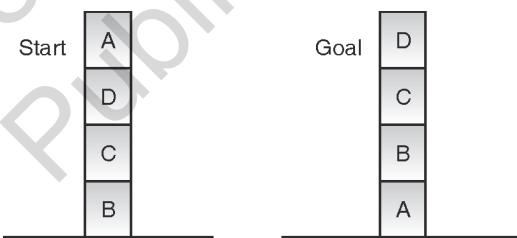


Fig. 2.14.3 : Block problem

- Fig. 2.14.3 depicts a block problem world, where the A, B, C,D letter bricks are piled up on one another and required to be arranged as shown in goal state, by moving one brick at a time. As shown, the goal state with the particular arrangement of blocks need to be attain from the given start state. Now it's time to scratch your head and define a heuristic function that will distinguish start state from goal state. Confused??
- Let's design a function which assigns + 1 for the brick at right position and – 1 for the one which is at wrong position. Consider Fig. 2.14.4

(a) Local heuristic :

- + 1 for each block that is resting on the thing it is supposed to be resting on.
- - 1 for each block that is resting on a wrong thing.

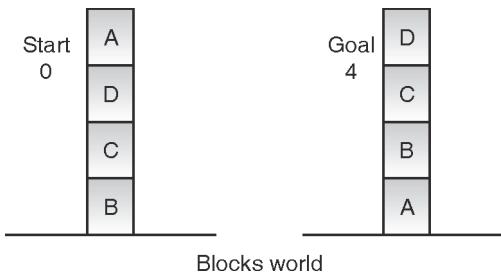


Fig. 2.14.4 : Definition of heuristic function “ h_1 ”

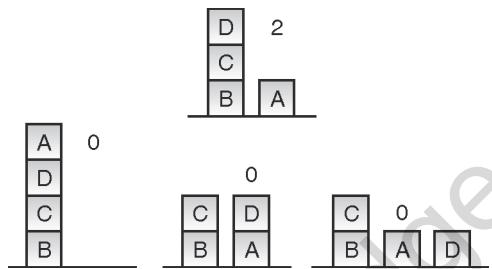


Fig. 2.14.5 : State evaluations using Heuristic function “ h_1 ”

- Fig. 2.14.5 shows the heuristic values generated by heuristic function “ h_1 ” for various different states in the state space. Please observe that, this heuristic is generating same value for different states.
- Due to this kind of heuristic the search may end up in limitless iterations as the state showing most promising heuristic value may not hold true or search may end up in finding an undesirable goal state as the state evaluation may lead to wrong direction in the search tree.
- Let's have another heuristic design for the same problem. Fig. 2.14.6 is depicting a new heuristic function “ h_2 ” definition, in which the correct support structure of each brick is given +1 for each brick in the support structure. And the one not having correct support structure, -1 for each brick in the wrong support structure.

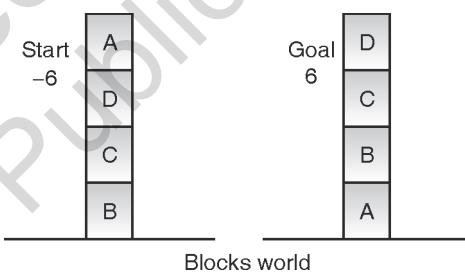


Fig. 2.14.6 : Definition of heuristic function “ h_2 ”

- As we observe in Fig. 2.14.7, the same states are considered again as that of Fig. 2.14.5, but this time using h_2 , each one of the state is assigned a unique value generate according to heuristic function h_2 .
- Observing this example one can easily understand that, in the second part of the example, search will be carried out smoothly as each unique state is getting a unique value assigned to it.
- This example makes it clear that, the design of heuristic plays a vital role in search process, as the whole search is carried out by considering the heuristic values as basis for selecting the next state to be explored.
- The state having the most promising value to reach to the goal state will be the first prior candidate for exploration, this continues till we find the goal state.

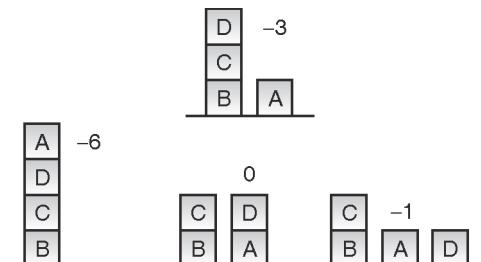
(b) Global heuristic :

Fig. 2.14.7 : State evaluations using heuristic function “ h_2 ”

- For each block that has the correct support structure : + 1 to every block in the support structure.
- For each block that has the wrong support structure : – 1 to every block in the support structure.
- This leads to a discussion of a better heuristic function definition.
- Is there any particular way of defining a heuristic function that will guarantee a better performance in search process??

2.14.3 Properties of Good Heuristic Function

1. It should generate a unique value for each unique state in search space.
 2. The values should be a logical indicator of the profitability of the state in order to reach the goal state.
 3. It may not guarantee to find the best solution, but almost always should find a very good solution.
 4. It should reduce the search time; specifically for hard problems like travelling salesman problem where the time required is exponential.
- The main objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem, as it's an extra task added to the basic search process.
 - The solution produced by using heuristic may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding the solution does not require a prohibitively long time. So we are investing some amount of time in generating heuristic values for each state in search space but reducing the total time involved in actual searching process.
 - Do we require to design heuristic for every problem in real world? There is a trade-off criterion for deciding whether to use a heuristic for solving a given problem. It is as follows.
 - o **Optimality** : Does the problem require to find the optimal solution, if there exist multiple solutions for the same?
 - o **Completeness** : In case of multiple existing solution of a problem, is there a need to find all of them? As many heuristics are only meant to find one solution.
 - o **Accuracy and precision** : Can the heuristic guarantee to find the solution within the precision limits? Is the error bar on the solution unreasonably large?
 - o **Execution time** : Is it going to affect the time required to find the solution? Some heuristics converge faster than others. Whereas, some are only marginally quicker than classic methods.
 - In many AI problems, it is often hard to measure precisely the goodness of a particular solution. But still it is important to keep performance question in mind while designing algorithm. For real world problems, it is often useful to introduce heuristics based on relatively unstructured knowledge. It is impossible to define this knowledge in such a way that mathematical analysis can be performed.

2.15 Best First Search

2.15.1 Concept

- In depth first search all competing branches are not getting expanded. And breadth first search never gets trapped on dead end paths. If we combine these properties of both DFS and BFS, it would be “follow a single path at a time, but switch paths whenever some competing path look more promising than the current one”. This is what the Best First search is..!!
- Best-first search is a search algorithm which explores the search tree by expanding the most promising node chosen according to the heuristic value of nodes. Judea Pearl described best-first search as estimating the promise of node n by a “heuristic evaluation function $f(n)$ which, in general, may depend on the description of n , the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain”.
- Efficient selection of the current best candidate for extension is typically implemented using a priority queue. Fig. 2.15.1 depicts the search process of Best first search on an example search tree. The values noted below the nodes are the estimated heuristic values of nodes.

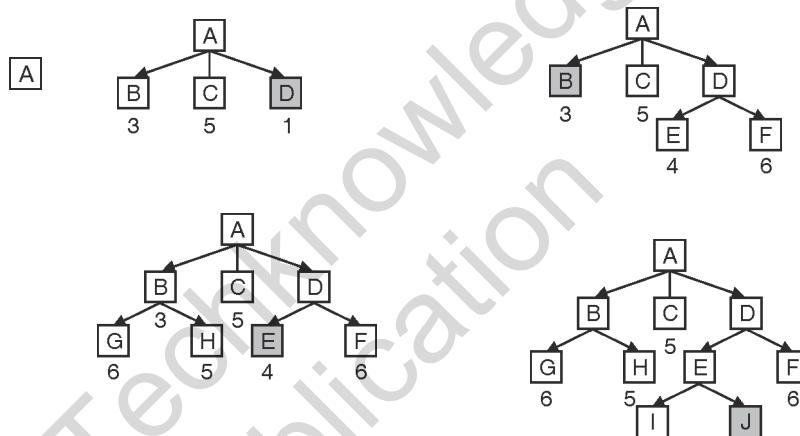


Fig. 2.15.1 : Best first search tree expansion scenario

2.15.2 Implementation

- Best first search uses two lists in order to record the path. These are namely OPEN list and CLOSED list for implementation purpose.
- OPEN list stores nodes that have been generated, but have not examined. This is organized as a priority queue, in which nodes are stored with the increasing order of their heuristic value, assuming we are implementing maximization heuristic. It provides efficient selection of the current best candidate for extension.
- CLOSED list stores nodes that have already been examined. This CLOSED list contains all nodes that have been evaluated and will **not be looked at again**. Whenever a new node is generated, check whether it has been generated before. If it is already visited before, check its recorded value and change the parent if this new value is better than previous one. This will avoid any node being evaluated twice, and will never get stuck into an infinite loops.

2.15.3 Algorithm : Best First Search

OPEN = [initial state]

CLOSED = []

while OPEN is not empty

do



1. Remove the best node from OPEN, call it n, add it to CLOSED.
2. If n is the goal state, backtrack path to n through recorded parents and return path.
3. Create n's successors.
4. For each successor do :
 - a. If it is not in CLOSED and it is not in OPEN : evaluate it, add it to OPEN, and record its parent.
 - b. Otherwise, if it is already present in OPEN with different parent node and this new path is better than previous one, change its recorded parent.
 - i. If it is not in OPEN add it to OPEN.
 - ii. Otherwise, adjust its priority in OPEN using this new evaluation.

done

This algorithm of Best First Search algorithm just terminates when no path is found. An actual implementation would of course require special handling of this case.

2.15.4 Performance Measures for Best first search

- **Completeness** : Not complete, may follow infinite path if heuristic rates each state on such a path as the best option. Most reasonable heuristics will not cause this problem however.
- **Optimality** : Not optimal; may not produce optimal solution always.
- **Time Complexity** : Worst case time complexity is still $O(b^m)$ where m is the maximum depth.
- **Space Complexity** : Since must maintain a queue of all unexpanded states, space-complexity is also $O(b^m)$.

2.15.5 Greedy Best First Search

- A greedy algorithm is an algorithm that follows the heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.
- When Best First Search uses a heuristic that leads to goal node, so that nodes which seems to be more promising are expanded first. This particular type of search is called greedy best-first search.
- In greedy best first search algorithm, first successor of the parent is expanded. For the successor node, check the following :
 1. If the successor node's heuristic is better than its parent, the successor is set at the front of the queue, with the parent reinserted directly behind it, and the loop restarts.
 2. Else, the successor is inserted into the queue, in a location determined by its heuristic value. The procedure will evaluate the remaining successors, if any of the parent.
- In many cases, greedy best first search may not always produce an optimal solution, but the solution will be locally optimal, as it will be generated in comparatively less amount of time. In mathematical optimization, greedy algorithms solve combinatorial problems.
- For example, consider the traveling salesman problem, which is of a high computational complexity, works well with greedy strategy as follows. Refer to Fig. 2.15.2. The values written on the links are the straight line distances from the nodes. Aim is to visit all the cities A through F with the shortest distance travelled.
- Let us apply a greedy strategy for this problem with a heuristic as, “At each stage visit an unvisited city nearest to the current city”. Simple logic... isn't it? This heuristic need not find a best solution, but terminates in a reasonable number of steps by finding an optimal solution which typically requires unreasonably many steps. Let's verify.

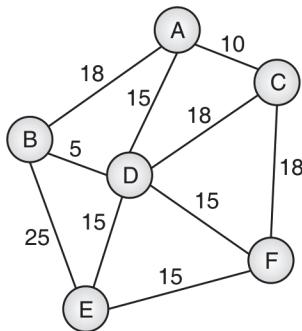


Fig. 2.15.2 : Travelling Salesmen Problem example

- As greedy algorithm, it will always make a local optimal choice. Hence it will select node C first as it found to be the one with less distance from the next non-visited node from node A, and then the path generated will be A→C→D→B→E→F with the total cost = $10 + 18 + 5 + 25 + 15 = 73$. While by observing the graph one can find the optimal path and optimal distance the salesman needs to travel. It turns out to be, A→B→D→E→F→C where the cost comes out to be $18 + 5 + 15 + 15 + 18 = 68$.

2.15.6 Properties of Greedy Best-First Search

- Completeness :** It's not complete as, it can get stuck in loops, also is susceptible to wrong start and quality of heuristic function.
- Optimality :** It's not optimal; as it goes on selecting a single path and never checks for other possibilities.
- Time Complexity :** $O(b^m)$, but a good heuristic can give dramatic improvement.
- Space Complexity :** $O(b^m)$, It needs to keep all nodes in memory.

2.16 A* Search

MU - May 13, Dec. 13, May 14, Dec. 14, May 15

- | | |
|--|-----------------------------|
| Q. Explain A* Algorithm. What is the drawback of A* ? Also shows that A* is optimally efficient. | (May 13, 10 Marks) |
| Q. Describe A* algorithm with merits and demerits. | (Dec. 13, 10 Marks) |
| Q. Explain A* algorithm with example. | (May 14, Dec. 14, 10 Marks) |
| Q. Explain A* search with example. | (May 15, 10 Marks) |

2.16.1 Concept

- A* pronounced as "Aystar" (Hart, 1972) search method is a combination of branch and bound and best first search, combined with the dynamic programming principle.
- It's a variation of Best First search where the evaluation of a state or a node not only depends on the heuristic value of the node but also considers its distance from the start state. It's the most widely known form of best-first search. A* algorithm is also called as OR graph / tree search algorithm.
- In A* search, the value of a node n, represented as $f(n)$ is a combination of $g(n)$, which is the cost of cheapest path to reach to the node from the root node, and $h(n)$, which is the cost of cheapest path to reach from the node to the goal node. Hence $f(n) = g(n) + h(n)$.
- As the heuristic can provide only the estimated cost from the node to the goal we can represent $h(n)$ as $h^*(n)$; similarly $g^*(n)$ can represent approximation of $g(n)$ which is the distance from the root node observed by A* and the algorithm A* will have,

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



- As we observe the difference between the A* and Best first search is that; in Best first search only the heuristic estimation of $h(n)$ is considered while A* counts for both, the distance travelled till a particular node and the estimation of distance need to travel more to reach to the goal node, it always finds the cheapest solution.
- A reasonable thing to try first is the node with the lowest value of $g^*(n) + h^*(n)$. It turns out that this strategy is more than just reasonable, provided that the heuristic function $h^*(n)$ satisfies certain conditions which are discussed further in the chapter. A* search is both complete and optimal.

2.16.2 Implementation

A* does also use both OPEN and CLOSED list.

2.16.3 Algorithm (A*)

1. Initialization OPEN list with initial node; CLOSED= \emptyset ; $g = 0$, $f = h$, **Found = false**;
2. While ($OPEN \neq \emptyset$ and **Found = false**)
 - {¹
 - i. Remove the node with the lowest value of f from OPEN to CLOSED and call it as a **Best_Node**.
 - ii. If **Best_Node** = Goal state then **Found = true**
 - iii. else
 - {²
 - a. Generate the **SuccofBest_Node**
 - b. For each **Succ** do
 - {³
 - i. Compute $g(Succ) = g(Best_Node) + \text{cost of getting from Best_Node to Succ.}$
 - ii. If $Succ \in OPEN$ then /* already being generated but not processed */
 - {⁴
 - a. Call the matched node as OLD and add it in the list of Best_Node successors.
 - b. Ignore the **Succ** node and change the parent of OLD, if required.
 - If $g(Succ) < g(OLD)$ then make parent of OLD to be **Best_Node** and change the values of g and f for OLD
 - If $g(Succ) \geq g(OLD)$ then ignore
 - a. If $Succ \in CLOSED$ then /* already processed */
 - {⁵
 - i. Call the matched node as OLD and add it in the list of Best_Node successors.
 - ii. Ignore the **Succ** node and change the parent of OLD, if required
 - If $g(Succ) < g(OLD)$ then make parent of OLD to be **Best_Node** and change the values of g and f for OLD.
 - Propogate the change to OLD's children using depth first search
 - If $g(Succ) \geq g(OLD)$ then do nothing
 - a. If $Succ \notin OPEN$ or $CLOSED$
 - {⁶

- i. Add it to the list of Best_Node's successors
 - ii. Compute $f(\text{Succ}) = g(\text{Succ}) + h(\text{Succ})$
 - iii. Put **Succ** on OPEN list with its f value
- }⁶
- }³ /* for loop*/
- }² /* else if */
- }¹ /* End while */.
3. If **Found = true** then report the best path else report failure
 4. Stop

2.16.4 Behaviour of A* Algorithm

As stated already the success of A* totally depends upon the design of heuristic function and how well it is able to evaluate each node by estimating its distance from the goal node. Let us understand the effect of heuristic function on the execution of the algorithm and how the optimality gets affected by it.

A. Underestimation

- If we can guarantee that heuristic function 'h' never over estimates actual value from current to goal that is, the value generated by h is always lesser than the actual cost or actual number of steps required to reach to the goal state. In this case, A* algorithm is guaranteed to find an optimal path to a goal, if one exists.

Example :

$$f = g + h, \text{ Here } h \text{ is underestimated.}$$

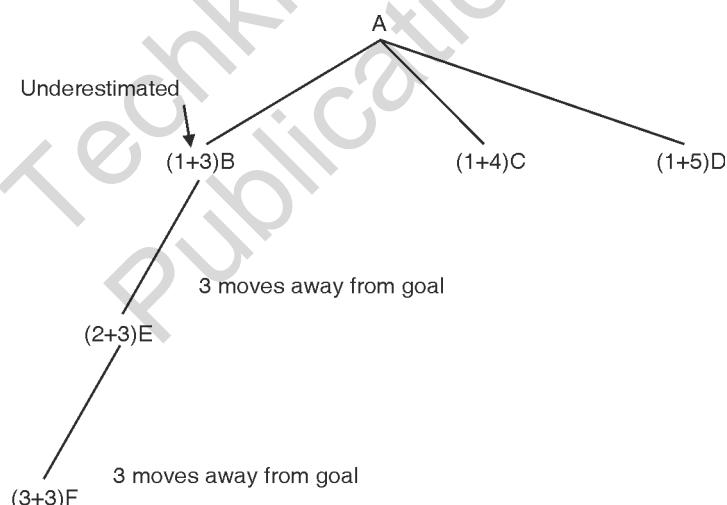


Fig. 2.16.1

- If we consider cost of all arcs to be 1. A is expanded to B, C and D. 'f' values for each node is computed. B is chosen to be expanded to E. We notice that $f(E) = f(C) = 5$. Suppose we resolve in favor of E, the path currently we are expanding. E is expanded to F. Expansion of a node F is stopped as $f(F)=6$ so we will now expand node C.
- Hence by underestimating $h(B)$, we have wasted some effort but eventually discovered that B was farther away than we thought. Then we go back and try another path, and will find optimal path.

B. Overestimation

- Here h is overestimated that is, the value generated for each node is greater than the actual number of steps required to reach to the goal node.

Example :

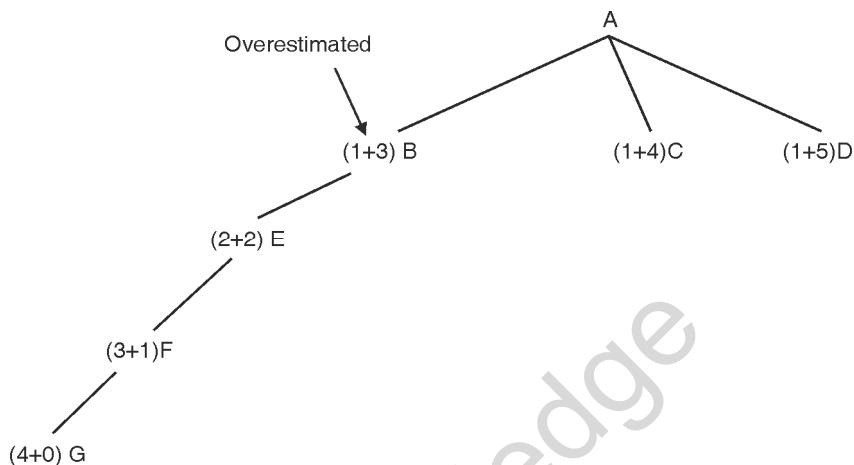


Fig. 2.16.2

- As shown in the example, A is expanded to B, C and D. Now B is expanded to E, E to F and F to G for a solution path of length 4. Consider a scenario when there a direct path from D to G with a solution giving a path of length 2. This path will never be found because of overestimating $h(D)$.
- Thus, some other worse solution might be found without ever expanding D. So by overestimating h , one cannot guarantee to find the cheaper path solution.

2.16.5 Admissibility of A*

MU - Dec. 12, May 14

Q. What do you mean an admissible heuristics function ? Discuss with suitable example.	(Dec. 12, 5 Marks)
Q. Write short note on admissibility of A*.	(May 14, 5 Marks)

- A search algorithm is admissible, if for any graph, it always terminates in an optimal path from initial state to goal state, if path exists. A heuristic is admissible if it never overestimates the actual cost from current state to goal state. Alternatively, we can say that A* always terminates with the optimal path in case $h(n)$ is an admissible heuristic function.
- A heuristic $h(n)$ is admissible if for every node n , if $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n . An admissible heuristic never overestimates the cost to reach the goal. Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.
- An obvious example of an admissible heuristic is the straight line distance. Straight line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot overestimate the actual road distance.
 - Theorem :** If $h(n)$ is admissible, tree search using A* is optimal.
 - Proof :** Optimality of A* with admissible heuristic.
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G .

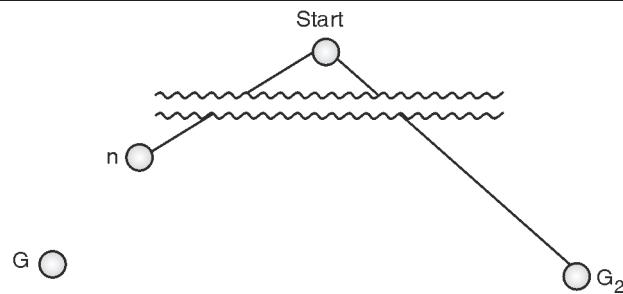


Fig. 2.16.3 : Optimality of A*

$$f(G_2) = g(G_2)$$

since $h(G_2) = 0$

$$g(G_2) > g(G)$$

since G_2 is suboptimal

$$f(G) = g(G)$$

since $h(G) = 0$

$$f(G_2) > f(G)$$

from above

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

since h is admissible

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

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

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

2.16.6 Monotonicity

MU -May 16

Q. Prove that A^* is admissible if it uses Monotone heuristic.

(May 16, 5 Marks)

- A heuristic function h is monotone or consistent if,
- \forall states X_i and X_j such that X_j is successor of X_i ,

$$h(X_i) - h(X_j) \leq \text{cost}(X_i, X_j)$$

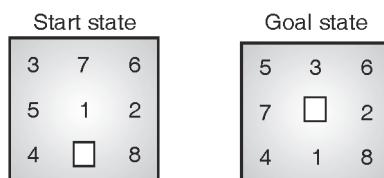
where, cost (X_i, X_j) actual cost of going from X_i to X_j and $h(\text{goal}) = 0$

- In this case, heuristic is locally admissible i.e., consistently finds the minimal path to each state they encounter in the search. The monotone property in other words is that search space which is everywhere locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors. With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter. Each monotonic heuristic is admissible.
- A cost function $f(n)$ is monotone if $f(n) \leq f(\text{succ}(n))$, $\forall n$.
- For any admissible cost function f , we can construct a monotone admissible function.
- Alternatively, the monotone property : that search space which is everywhere locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors.
- With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter.

2.16.7 Properties of A^*

1. **Completeness** : It is complete, as it will always find solution if one exist.
2. **Optimality** : Yes, it is Optimal.
3. **Time Complexity** : $O(b^m)$, as the number of nodes grows exponentially with solution cost.
4. **Space Complexity** : $O(b^m)$, as it keeps all nodes in memory.

2.16.8 Example : 8 Puzzle Problem using A* Algorithm



Evaluation function - f for EPP

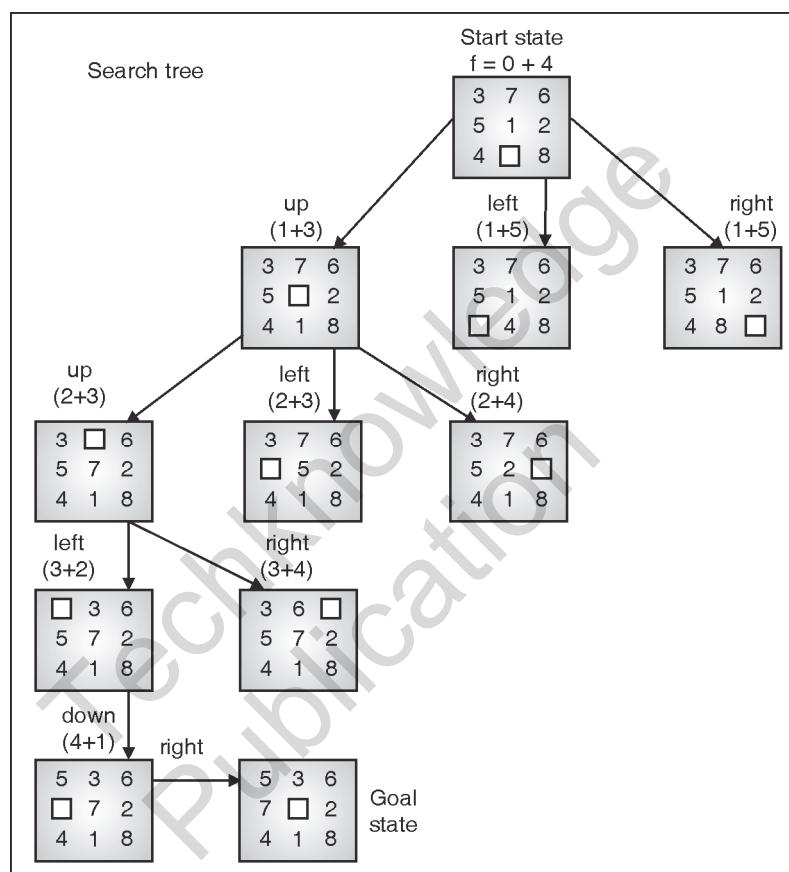


Fig. 2.16.4 : Solution of 8-puzzle using A*

- The choice of evaluation function critically determines search results.
- Consider Evaluation function

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

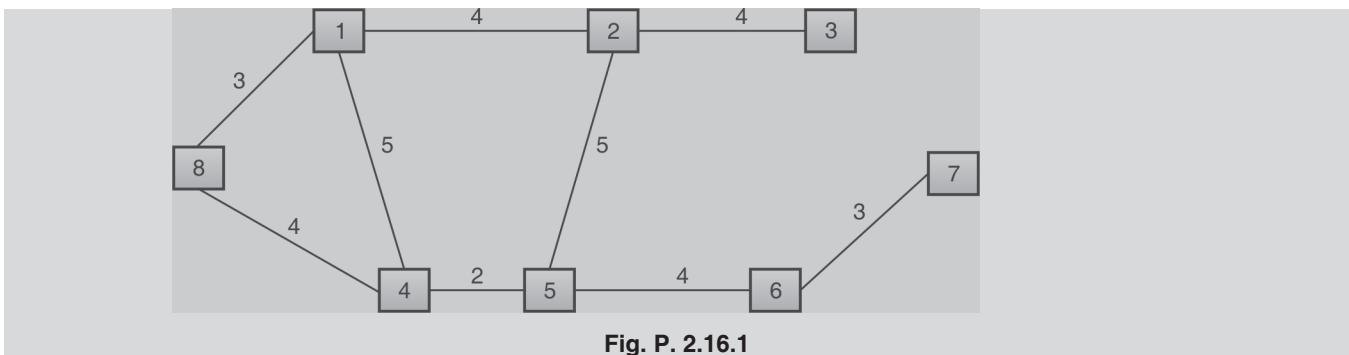
$h(X)$ = the number of tiles not in their goal position in a given state X

$g(X)$ = depth of node X in the search tree

- For Initial node $f(\text{initial_node}) = 4$

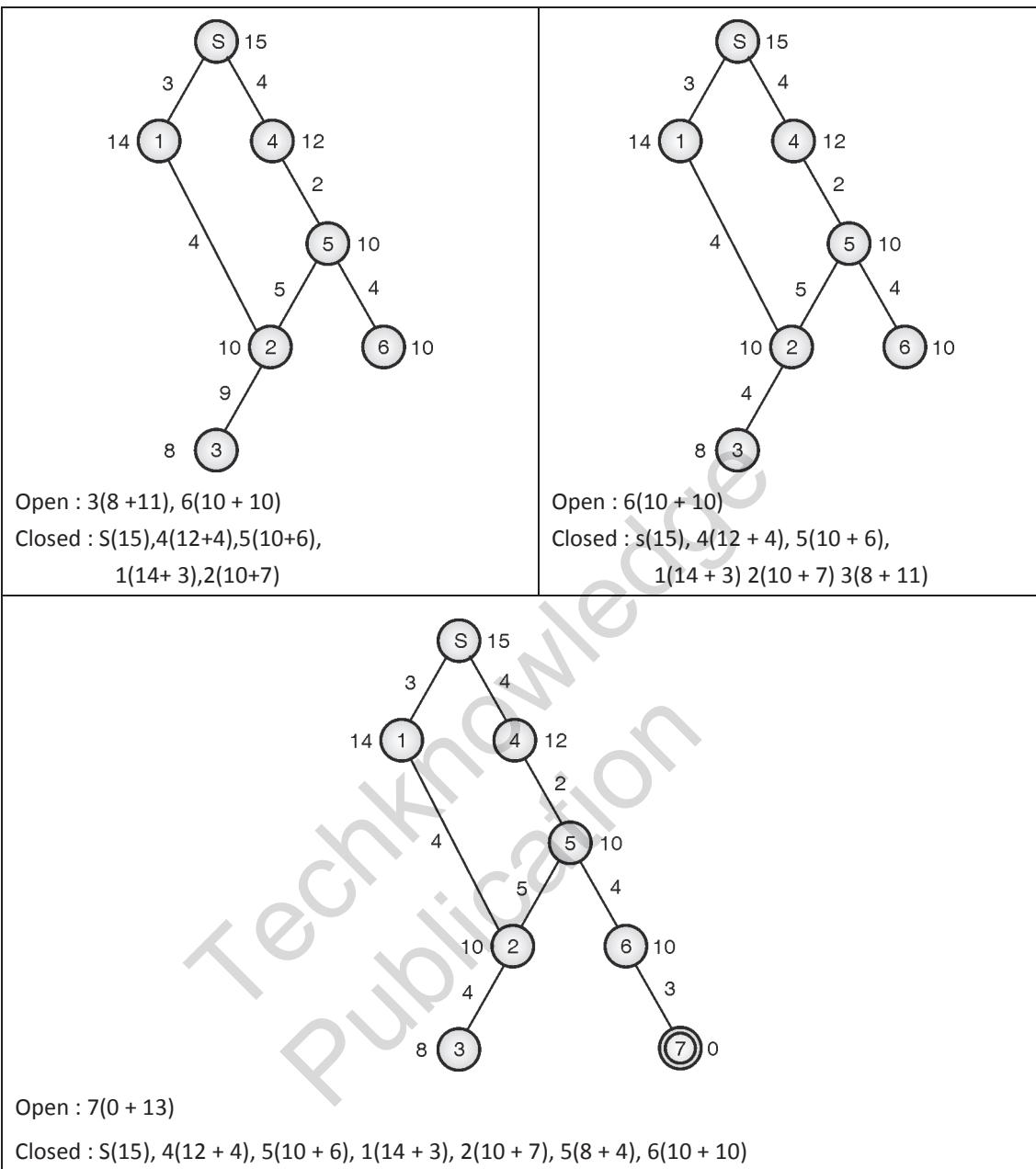
Ex. 2.16.1 : Consider the graph given in Fig. P.2.16.1 below. Assume that the initial state is S and the goal state is 7. Find a path from the initial state to the goal state using A* Search. Also report the solution cost. The straight line distance heuristic estimates for the nodes are as follows : $h(1) = 14$, $h(2) = 10$, $h(3) = 8$, $h(4) = 12$, $h(5) = 10$, $h(6) = 10$, $h(S) = 15$.

MU - Dec. 15, 5 Marks



Soln. :

<p>Open : 4(12 + 4), 1(14 + 3) Closed : S(15)</p>	<p>Open : 5(10 + 6), 1(14 + 3) Closed : S(15), 4 (12 + 4)</p>
<p>Open : 1(14 + 3) 6(10 + 10) 2(10 + 11) Closed : S(15) 4(12 + 4) 5(10 + 6)</p>	<p>Open : 2(10 + 7) 6(10+10) Closed : S(5) 4(12 + 4) 5(10 + 6) 1(14 + 3)</p>



2.16.9 Comparison among Best First Search, A* search and Greedy Best First Search

MU - May 16

- Q.** Compare following informed searching algorithms based on performance measure with justification : Complete, Optimal, Time complexity and space complexity.

(a)Greedy best first (b) A* (c) recursive best-first (RBFS)

(May 16, 10 Marks)

Algorithm	Greedy Best First Search	A* search	Best First Search
Completeness	Not complete	complete	Not complete
Optimality	Not optimal	optimal	Not optimal
Time Complexity	$O(b^m)$	$O(b^m)$	$O(bm)$
Space Complexity	$O(b^m)$	$O(b^m)$	$O(bm)$

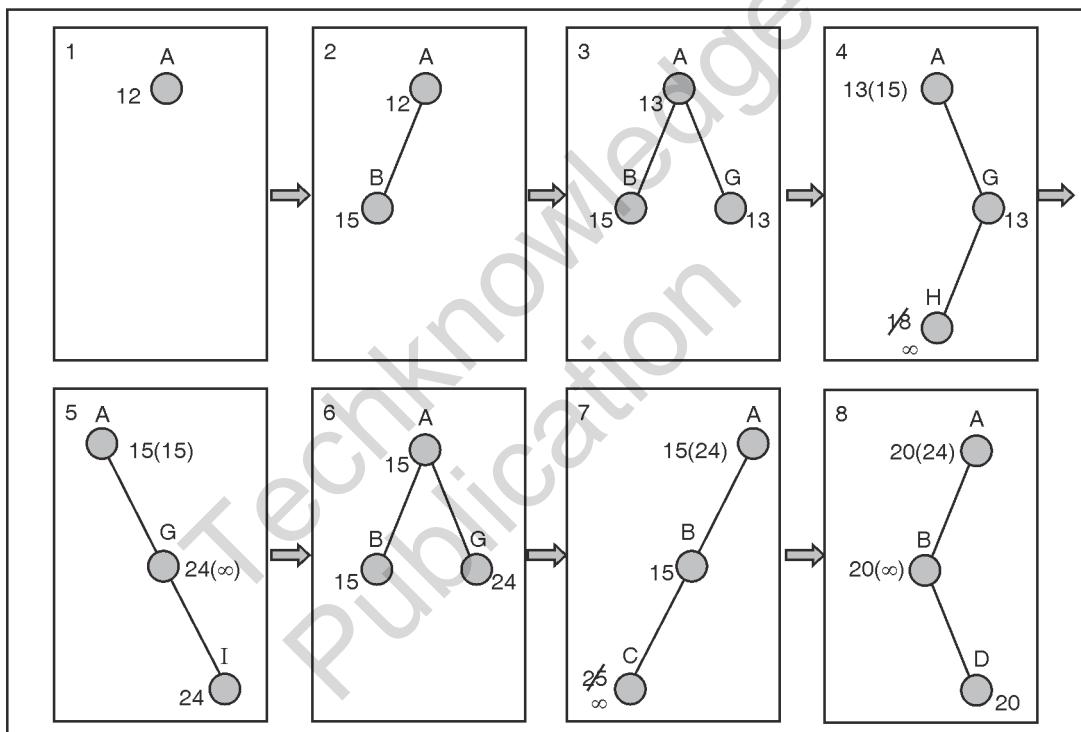
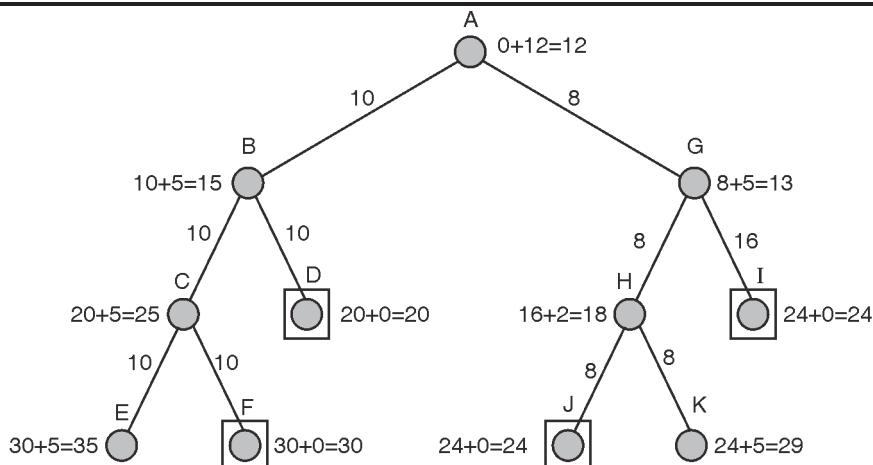


Fig. 2.16.5 : Process of SMA* with memory size of 3 nodes

2.17 Local Search Algorithms and Optimization Problems

2.17.1 Hill Climbing

MU -May 13, Dec. 13, Dec. 14, May 16

Q. Write short note on Hill Climbing algorithm.	(Dec. 13, 5 Marks)
Q. Describe Hill Climbing algorithm.	(May 13, Dec. 14, 5 Marks)
Q. Explain Hill- Climbing algorithm with example.	(May 16, 5 Marks)

- Hill climbing is simply a combination of depth first with generate and test where a feedback is used here to decide on the direction of motion in the search space.
- Hill climbing technique is used widely in artificial intelligence, to solve solving computationally hard problems, which has multiple possible solutions.

- In the depth-first search, the test function will merely accept or reject a solution. But in hill climbing the test function is provided with a heuristic function which provides an estimate of how close a given state is to goal state.
- In Hill climbing, each state is provided with the additional information needed to find the solution, i.e. the heuristic value. The algorithm is memory efficient since it does not maintain the complete search tree. Rather, it looks only at the current state and immediate level states.
- For example, if you want to find a mall from your current location. There are n possible paths with different directions to reach to the mall. The heuristic function will just give you the distance of each path which is reaching to the mall, so that it becomes very simple and time efficient for you to reach to the mall.
- Hill climbing attempts to iteratively improve the current state by means of an evaluation function. “Consider all the possible states laid out on the surface of a landscape. The height of any point on the landscape corresponds to the evaluation function of the state at that point” (Russell and Norvig, 2003). Fig. 2.17.1 depicts the typical hill climbing scenario, where multiple paths are available to reach to the hill top from ground level.

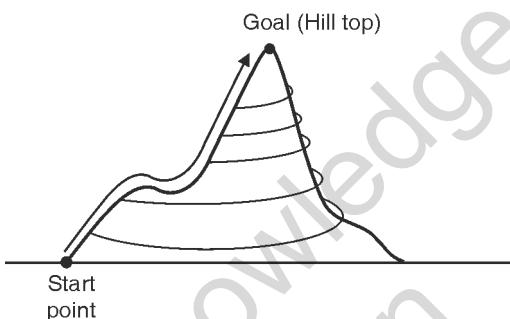


Fig. 2.17.1 : Hill Climbing Scenario

- Hill climbing always attempts to make changes that improve the current state. In other words, hill climbing can only advance if there is a higher point in the adjacent landscape.
- Hill climbing is a type of local search technique. It is relatively simple to implement. In many cases where state space is of moderate size, hill climbing works even better than many advanced techniques.
- For example, hill climbing when applied to travelling salesman problem; initially it produces random combinations of solutions having all the cities visited. Then it selects the better rout by switching the order, which visits all the cities in minimum cost.
- There are two variations of hill climbing as discussed follow.

2.17.1(A) Simple Hill Climbing

It is the simplest way to implement hill climbing. Following is the algorithm for simple hill climbing technique. Overall the procedure looks similar to that of generate and test but, the main difference between the two is use of heuristic function for state evaluation which is used in hill climbing. The goodness of any state is decided by the heuristic value of that state. It can be either incremental heuristic or detrimental one.

Algorithm

1. Evaluate the initial state. If it is a goal state, then return and quit; otherwise make it a current state and go to Step 2.
2. Loop until a solution is found or there are no new operators left to be applied (i.e. no new children nodes left to be explored).
 - a. Select and apply a new operator (i.e. generate new child node)
 - b. Evaluate the new state :
 - (i) If it is a goal state, then return and quit.



- (ii) If it is better than current state then make it a new current state.
- (iii) If it is not better than the current state then continue the loop, go to Step 2.

As we study the algorithm, we observe that in every pass the first node / state that is better than the current state is considered for further exploration. This strategy may not guarantee that most optimal solution to the problem, but may save upon the execution time.

2.17.1(B) Steepest Ascent Hill Climbing

As the name suggests, steepest hill climbing always finds the steepest path to hill top. It does so by selecting the best node among all children of the current node / state. All the states are evaluated using heuristic function. Obviously, the time requirement of this strategy is more as compared to the previous one. The algorithm for steepest ascent hill climbing is as follows.

Algorithm

1. Evaluate the initial state, if it is a goal state, return and quit; otherwise make it as a current state.
2. Loop until a solution is found or a complete iteration produces no change to current state :
 - a. SUCC = a state such that any possible successor of the current state will be better than SUCC.
 - b. For each operator that applies to the current state, evaluate the new state :
 - (i) If it is goal; then return and quit
 - (ii) If it is better than SUCC then set SUCC to this state.
 - c. SUCC is better than the current state → set the current state to SUCC.
- As we compare simple hill climbing with steepest ascent, we find that there is a tradeoff for the time requirement and the accuracy or optimality of the solution.
- In case of simple hill climbing technique as we go for first better successor, the time is saved as all the successors are not evaluated but it may lead to more number of nodes and branches getting explored, in turn the solution found may not be the optimal one.
- While in case of steepest ascent hill climbing technique, as every time the best among all the successors is selected for further expansion, it involves more time in evaluating all the successors at earlier stages, but the solution found will be always the optimal solution, as only the states leading to hill top are explored. This also makes it clear that the evaluation function i.e. the heuristic function definition plays a vital role in deciding the performance of the algorithm.

2.17.1(C) Limitations of Hill Climbing

MU -May 13, May 14, Dec. 14, May 15

Q. What are the limitations of Hill Climbing ?	(May 13, Dec.14, 5 Marks)
Q. Write short note on Limitations of Hill-climbing algorithm.	(May 14, May 15, 5 Marks)

- Now let's see what can be the impact of incorrect design of heuristic function on the hill climbing techniques.
- Following are the problems that may arise in hill climbing strategy. Sometimes the algorithms may lead to a position, which is not a solution, but from which there is no move possible which will lead to a better place on hill i.e. no further state that is going closer to the solution. This will happen if we have reached one of the following three states.
 1. **Local Maximum :** A “local maximum” is a location in hill which is at height from other parts of the hill but is not the actual hill top. In the search tree, it is a state better than all its neighbors, but there is not next better state which can be chosen for further expansion. Local maximum sometimes occur within sight of a solution. In such cases they are called “Foothills”.

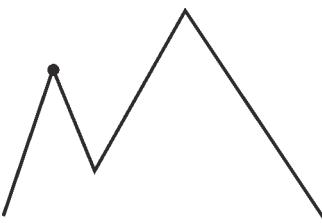


Fig. 2.17.2 : Local Maximum

- In the search tree local maximum can be seen as follows :

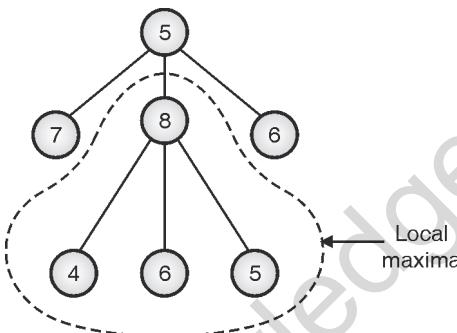


Fig. 2.17.3 : Local maxima in search tree

2. **Plateau** : A "plateau" is a flat area at some height in hilly region. There is a large area of same height in plateau. In the search space, plateau situation occurs when all the neighbouring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.



Fig. 2.17.4 : Plateau in hill climbing

- In the search tree plateau can be identified as follows :

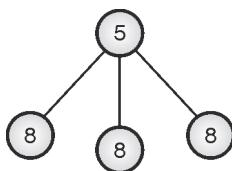


Fig. 2.17.5

3. **Ridge** : A "ridge" is an area in the hill such that, it is higher than the surrounding areas, but there is no further uphill path from ridge. In the search tree it is the situation, where all successors are either of same value or lesser, it's a ridge condition. The suitable successor cannot be searched in a simple move.

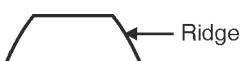


Fig. 2.17.6 : Ridge in hill climbing

- In the search tree ridge can be identified as follows :

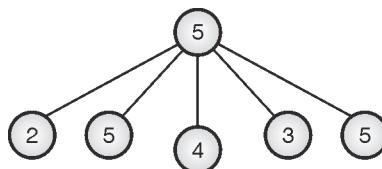


Fig. 2.17.7

- Fig. 2.17.8 depicts all the different situations together in hill climbing.

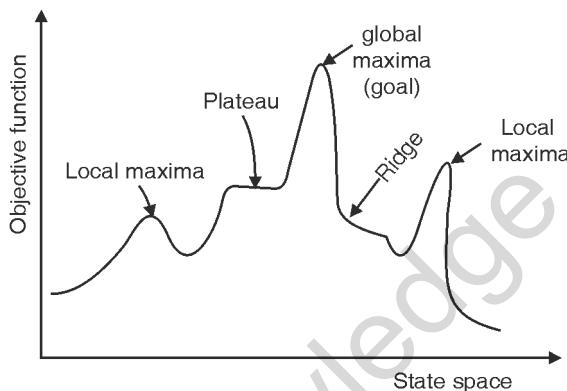


Fig. 2.17.8 : Hill climbing problems scenario

2.17.1(D) Solutions on Problems in Hill Climbing

- In order to overcome these problems we can try following techniques. At times combination of two techniques will provide a better solution.
 1. A good way to deal with local maximum, we can back track to some earlier nodes and try a different direction.
 2. In case of plateau and ridges, make a big jump in some direction to a new area in the search. This can be done by applying two more rules of the same rule several times, before testing. This is a good strategy is dealing with plateau and ridges.
- Hill climbing is a local method. It decides what to do next by looking only at the "immediate" consequences of its choices. Global information might be encoded in heuristic functions. Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it uses very little memory, usually a constant amount, as it doesn't retain the path.
- It is a useful when combined with other methods. The success of hill climbing depends very much on the shape of the state space. If there are few local maxima and plateau, random-restart hill climbing will find a good solution very quickly.

2.17.2 Simulated Annealing

- Simulated annealing is a variation of hill climbing. Simulated annealing technique can be explained by an analogy to annealing in solids. In the annealing process in case of solids, a solid is heated past melting point and then cooled.
- With the changing rate of cooling, the solid changes its properties. If the liquid is cooled slowly, it gets transformed in steady frozen state and forms crystals. While, if it is cooled quickly, the crystal formation will not get enough time and it produces imperfect crystals.
- The aim of physical annealing process is to produce a minimal energy final state after raising the substance to high energy level. Hence in simulated annealing we are actually going downhill and the heuristic function is a minimal heuristic. The final state is the one with minimum value, and rather than climbing up in this case we are descending the valley.



- The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution. In order to achieve that, at the beginning of the process, some downhill moves may be made. These downhill moves are made purposely, to do enough exploration of the whole space early on, so that the final solution is relatively insensitive to the starting state. It reduces the chances of getting caught at a local maximum, or plateau, or a ridge.

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied :

 Set T according to an annealing schedule

 Select and apply a new operator

 Evaluate the new state :

 goal → quit

$\Delta E = \text{Val}(\text{current state}) - \text{Val}(\text{new state})$

$\Delta E < 0 \rightarrow \text{new current state}$

 else → new current state with probability $e^{-\Delta E/kT}$.

- We observe in the algorithm that, if the next state is better than the current, it readily accepts it as a new current state. But in case when the next state is not having the desirable value even then it accepts that state with some probability, $e^{-\Delta E/kT}$ where ΔE is the positive change in the energy level, T is temperature and k is Boltzmann's constant.
- Thus, in the simulated annealing there are very less chances of large uphill moves than the small one. Also, the probability of uphill moves decreases with the temperature decrease. Hence uphill moves are more likely in the beginning of the annealing process, when the temperature is high. As the cooling process starts, temperature comes down, in turn the uphill moves. Downhill moves are allowed any time in the whole process. In this way, comparatively very small upward moves are allowed till finally, the process converges to a local minimum configuration, i.e. the desired low point destination in the valley.

2.17.2(A) Comparing Simulated Annealing with Hill Climbing

- A hill climbing algorithm never makes “downhill” moves toward states with lower value and it can be incomplete, because it can get stuck on a local maximum.
- In contrast, a purely **random walk**, i.e. moving to a successor chosen at random from the set of success or s independent of whether it is better than the current state, is complete but extremely inefficient. Therefore, it is reasonable to try a combination of hill climbing with a random walk in some way that yields both efficiency and completeness. Simulated annealing is the answer...!!
- As we know that, hill climbing can get stuck at local minima or maxima, thereby halting the algorithm abruptly, it may not guarantee optimal solution. Few attempts were made to solve this problem by trying hill climbing considering multiple start points or by increasing the size of neighbourhood, but none worked out to produce satisfactory results. Simulated annealing has solved the problem by performing some downhill moves at the beginning of search so that, local maximum can be avoided at later stage.
- Hill climbing procedure chooses the best state from those available or at least better than the current state for further expansion. Unlike hill climbing, simulated annealing chooses a random move from the neighbourhood. If the successor state turned out to be better than its current state then simulated annealing will accept it for further expansion. If the successor state is worse, then it will be accepted based on some probability.

2.17.3 Local Beam Search

- In all the variations of hill climbing till now, we have considered only one node getting selected at a time for further search process. These algorithms are memory efficient in that sense. But when an unfruitful branch gets explored even for some amount of time it is a complete waste of time and memory. Also the solution produced may not be the optimal one.
- The local beam search algorithm keeps track of k best states by performing parallel k searches. At each step it generates successor nodes and selects k best nodes for next level of search. Thus rather than focusing on only one branch it concentrates on k paths which seems to be promising. If any of the successors found to be the goal, search process stops.
- In parallel local beam search, the parallel threads communicate to each other, hence useful information is passed among the parallel search threads.
- In turn, the states that generate the best successors say to the others, "Come over here, the grass is greener!" The algorithm quickly terminates unfruitful branches exploration and moves its resources to where the path seems most promising. In stochastic beam search the maintained successor states are chosen with a probability based on their goodness.

Algorithm : Local Beam search

Step 1 : Found = false;

Step 2 : NODE = Root_node;

Step 3 : If NODE is the goal node, then *Found* = true else find SUCCs of NODE, if any with its estimated cost and store in OPEN list;

Step 4 : While (Found = false and not able to proceed further)

{

Sort OPEN list;

Select top W elements from OPEN list and put it in W_OPEN list and empty OPEN list;

While (W_OPEN ≠ φ and Found = false)

{

Get NODE from W_OPEN;

if NODE = Goal state then Found = true else

{

Find SUCCs of NODE, if any with its estimated cost

store in OPEN list;

}

} // end inner while

} // end outer while

Step 5 : If *Found* = true then return Yes otherwise return No and Stop

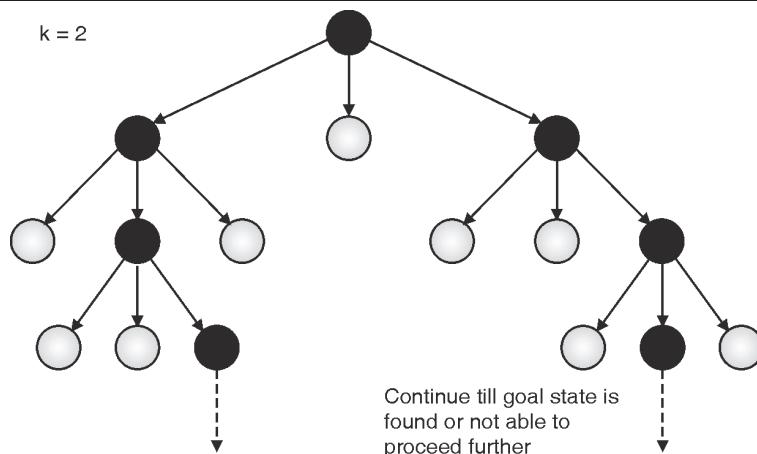


Fig. 2.17.9 : Process of local Beam Search

- As shown in Fig. 2.17.9, here $k = 2$, hence two better successors are selected for expansion at first level of search and at each next level, two better successors will be selected by both searches. They do exchange their information with each other throughout the search process. The search will continue till goal state is found or no further search is possible.
- It may seem to be that local beam search is same as running k searches in parallel. But it is not so. In case of parallel searches, all search run independent of each other. While in case of local beam search, the parallel running threads continuously coordinate with one another to decide the fruitful region of the search tree.
- Local beam search can suffer from a lack of diversity among the k states by quickly concentrating to small region of the state space.
- While it is possible to get excellent fits to training data, the application of back propagation is fraught with difficulties and pitfalls for the prediction of the performance on independent test data. Unlike most other learning systems that have been previously discussed, there are far more choices to be made in applying the gradient descent method.
- The key variations of these choices are : The learning rate and local minima - the selection of a learning rate is of critical importance in finding the true global minimum of the error distance.
- Back propagation training with too small a learning rate will make agonizingly slow progress. Too large a learning rate will proceed much faster, but may simply produce oscillations between relatively poor solutions.
- Both of these conditions are generally detectable through experimentation and sampling of results after a fixed number of training epochs.
- Typical values for the learning rate parameter are numbers between 0 and 1 : $0.05 < h < 0.75$.
- One would like to use the largest learning rate that still converges to the minimum solution momentum - empirical evidence shows that the use of a term called momentum in the back propagation algorithm can be helpful in speeding the convergence and avoiding local minima.
- The idea about using a momentum is to stabilize the weight change by making non radical revisions using a combination of the gradient decreasing term with a fraction of the previous weight change :

$$\Delta w(t) = -\partial E/\partial w(t) + \alpha \Delta w(t-1)$$

where α is taken $0 \leq \alpha \leq 0.9$, and t is the index of the current weight change.

- This gives the system a certain amount of inertia since the weight vector will tend to continue moving in the same direction unless opposed by the gradient term.
- The momentum has the following effects :



- It smooths the weight changes and suppresses cross-stitching, that is cancels side-to-side oscillations across the error valey;
 - When all weight changes are all in the same direction the momentum amplifies the learning rate causing a faster convergence;
 - Enables to escape from small local minima on the error surface.
- The hope is that the momentum will allow a larger learning rate and that this will speed convergence and avoid local minima. On the other hand, a learning rate of 1 with no momentum will be much faster when no problem with local minima or non-convergence is encountered ; sequential or random presentation - the epoch is the fundamental unit for training, and the length of training often is measured in terms of epochs. During a training epoch with revision after a particular example, the examples can be presented in the same sequential order, or the examples could be presented in a different random order for each epoch. The random representation usually yields better results.
- The randomness has advantages and disadvantages :
- **Advantages** : It gives the algorithm some stochastic search properties. The weight state tends to jitter around its equilibrium, and may visit occasionally nearby points. Thus it may escape trapping in suboptimal weight configurations. The on-line learning may have a better chance of finding a global minimum than the true gradient descent technique.
 - **Disadvantages** : The weight vector never settles to a stable configuration. Having found a good minimum it may then continue to wander around it.
- Random initial state - unlike many other learning systems, the neural network begin in a random state. The network weights are initialized to some choice of random numbers with a range typically between -0.5 and 0.5 (The inputs are usually normalized to numbers between 0 and 1). Even with identical learning conditions, the random initial weights can lead to results that differ from one training session to another.
- The training sessions may be repeated till getting the best results.

2.17.4 Genetic Algorithms

MU – Dec. 15, May 16

Q. Write a short note on genetic algorithm.	(Dec. 15, 8 Marks)
Q. Explain how genetic algorithm can be used to solve a problem by tacking a suitable example.	(May 16, 10 Marks)

- Gas are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, Gas are by no means random, instead they exploit historical information to direct the search for better performance within the search space. The basic techniques of the Gas are designed to simulate processes in natural systems necessary for evolution, especially those following the principles of “survival of the fittest” laid down by Charles Darwin.
- Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves towards better solutions. The solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individual and occurs in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified to form a new population. The new population is then used in the next iteration of the algorithm.



2.17.4(A) Terminologies of GA

– Gene

Gene is the smallest unit in genetic algorithm. The gene represents the smallest unit of information in the problem domain and can be thought of as the basic building block for a possible solution. If the problem context were, for example, the creation of a well-balanced investment portfolio, a gene might represent the number of shares of a particular security to purchase.

– Chromosome

Chromosome is a series of genes that represent the components of one possible solution to the problem. The chromosome is represented in computer memory as a bit string of binary digits that can be “decoded” by the genetic algorithm to determine how good a particular chromosome’s gene pool solution is for a given problem. The decoding process simply informs the genetic algorithm what the various genes within the chromosome represent.

– Encoding

Encoding of chromosomes is one of the problems, to start solving problem with GA. Encoding depends on the type of the problem. There are various types of encoding techniques like binary encoding, permutation encoding, value encoding, etc.

– Population

A population is a pool of individuals (chromosomes) that will be sampled for selection and evaluation. The performance of each individual will be computed and a new population will be reproduced using standard genetic operators.

– Reproduction

Reproduction is the process of creating new individuals called off-springs from the parents population. This new population will be evaluated again to select the desired results. Reproduction is done basically using two genetic operators : crossover and mutation. However, the genetic operators used can vary from model to model, there are a few standard or canonical operators : crossover and recombination of genetic material contained in different parent chromosomes, random mutation of data in individual chromosomes, and domain specific operations, such as migration of genes.

2.17.4(B) Genetic Operators

– Selection

In this process, chromosomes are selected from the population to be the parents for the crossover. The problem is how to select these chromosomes. According to Darwin’s evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection, etc.

– Crossover

Crossover involves the exchange of gene information between two selected chromosomes. The purpose of the crossover operation is to allow the genetic algorithm to create new chromosomes that share positive characteristics while simultaneously reducing the prevalence of negative characteristics in an otherwise reasonably fit solution. Types of crossover techniques include single-point crossover, two-point crossover, uniform crossover, mathematical crossover, tree crossover, etc.



- Mutation

Mutation is another refinement step that randomly changes the value of a gene from its current setting to a completely different one. The majority of the mutations formed by this process are, as is often the case in nature, less fit than more so. Occasionally, however, a highly superior and beneficial mutation will occur. Mutation provides the genetic algorithm with the opportunity to create chromosomes and information genes that can explore previously uncharted areas of the solution space, thus increasing the chances for the discovery of an optimal solution. There are various types of mutation techniques like bit inversion, order changing, value encoding.

2.17.4(C) The Basic Genetic Algorithm

1. [Start] Generate random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome x in the population
3. [New population] Create a new population by repeating following steps until the new population is complete
4. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
5. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
6. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
7. [Accepting] Place new offspring in a new population
8. [Replace] Use new generated population for a further run of algorithm
9. [Test] If the end condition is satisfied, stop, and return the best solution in current population
10. [Loop] Go to step 2

2.17.4(D) Example of Genetic Algorithm

Let's consider following pair of chromosomes encoded using permutation encoding technique and are undergoing the complete process of GA. Assuming that they are selected using rank selection method and will be applied, arithmetic crossover and value encoding mutation techniques.

a. Parent chromosome

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Child chromosome after **arithmetic crossover** : i.e adding bits of both chromosomes.

b. Child chromosome

Chromosome C	909 9 8 7837
--------------	--------------

After applying **value encoding mutation** i.e. adding or subtracting a small value to selected values. E.g. subtracting 1 to 3rd and 4th bit.

c. Child chromosome

Chromosome C	908 8 8 7837
--------------	--------------

It can be observed that the child produced is much better than both parents.

2.18 Adversarial Search

- **Adversarial Search Problem** is having competitive activity which involves ‘n’ players and it is played according to certain set of protocols.
- Game is called adversarial because there are agents with conflicting goals and the surrounding environment in a game is competitive as there are ‘n’ players or agents participating.
- We say that goals are conflicting and environment is competitive because every participant wants to win the game.
- From above explanation it is understandable that we are dealing with a competitive multi agent environment.
- As the actions of every agent are unpredictable there are many possible moves/actions.
- In order to play a game successfully every agent in environment has to first analyse the action of other agents and how other agents are contributing in its own winning or losing. After performing this analysis agent executes.

2.18.1 Environment Types

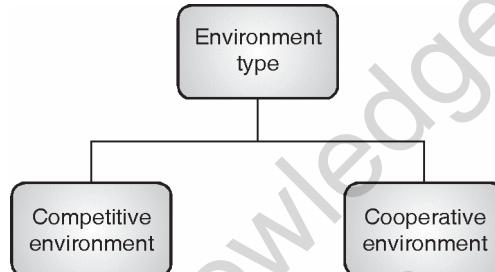


Fig. 2.18.1 : Environment types

There can be two types of environments in case of multi-agent : Competitive and cooperative.

1. Competitive environment

- In this type of environment every agent makes an effort to win the game by defeating or by creating superior it over other agents who are also trying to win the game.
- Chess is an example of a competitive environment.

2. Cooperative environment

- In this type of environment all the agents jointly perform activities in order to achieve same goal.
- Car driving agent is an example of a cooperative environment.

2.18.2 AI Game – Features

Under artificial intelligence category, there are few special features as shown in Table 2.18.1, which make the game more interesting.

Table 2.18.1 : AI game features

Features	Explanation	Example
Two player	When there are two opponents/agents playing the game it is called 2- play game. To increase difficulty of the game Intelligence is added to agents in AI games. Note that, in case of AI Games we must have at least two players. (i.e. single player games don't come under AI games category).	Chess

Features	Explanation	Example
Multi-agent	When there are two or more opponents/agents playing the game it is called multi agent environment where action of every agents affects the action of other agents.	Monopoly
Non-cooperative environment	When the surrounding environment is not helpful for winning the game it is called as non-cooperative or competitive.	Card games
Turn taking	In a multi-agent environment when the agent/ player performs a move and has to wait for the next player to make the next move.	Any board game, Chess, carom, etc.
Time limit	One more constraint can be incorporated in a game i.e. keeping a limitation on time. Every player will get a finite amount of time to take an action.	Time bound chess games
Unpredictable opponent	In AI games action of opponent agent is fuzzy which makes the game challenging and unpredictable. Players are called unpredictable when the next step depends upon an input set which is generated by the other player.	Card game with multiplayer

2.18.2(A) Zero Sum Game

- “Zero sum game” concept is associated with payoffs which are assigned to each player when the instance of the game is over. It is a mathematical representation of circumstances when the game is in a neutral state. (i.e. agents winning or losing is even handed with the winning and losing of other agents).
- For example, if player 1 wins chess game it is marked with say +1 point and at the same time the loss of player 2 is marked with -1 point, thus sum is zero. Another condition is when game is draw; in that case players 1 and 2 are marked with zero points. (Here +1, -1 and 0 are called payoffs).

2.18.2(B) Non-Zero Sum Game

Non-zero sum game's don't have algebraic sum of payoffs as zero. In this type of games one player winning the game does not necessarily mean that the other player has lost the game.

There are two types of non-zero sum games :

1. Positive sum game
2. Negative sum game

2.18.2(C) Positive Sum Game

It is also called as **cooperative game**. Here, all players have same goal and they contribute together to play the game. For example, educational games.

2.18.2(D) Negative Sum Game

It is also called as **competitive game**. Here, every player has a different goal so no one really wins the game, everybody loses. Real world example of a war suits the most.

2.19 Relevant Aspects of AI Game

To understand game playing, we will first take look at all appropriate aspects of a game which give overview of the stages in a game play. See Fig. 2.19.1.

- **Accessible environments** : Games with accessible environments have all the necessary information handy. For example : Chess.
- **Search** : Also there are games which require search functionality which illustrates how players have to search through possible game positions to play a game. For example : minesweeper, battleships.
- **Unpredictable opponent** : In AI games opponents can be unpredictable, this introduces uncertainty in game play and thus game-playing has to deal with contingency/ probability problems. For example : Scrabble.



Fig. 2.19.1 : Relevant aspects of AI game

2.20 Game Playing

- Fig. 2.20.1 shows examples of two main varieties of problems faced in artificial intelligence games. First type is “Toy Problems” and the other type is “Real World Problems”.

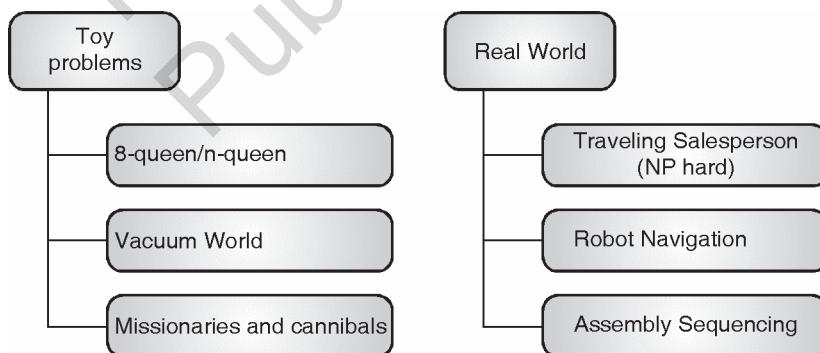


Fig. 2.20.1 : Example Problems

- Game play follows some strategies in order to mathematically analyse the game and generate possible outcomes. A two player strategy table can be seen in Table 2.20.1.

**Table 2.20.1 : Two player strategy table**

Player 1 Player 2	Strategy 1	Strategy 2	Strategy 3	...
Strategy 1	Draw	Player 1 Wins	Player 2 Wins	...
Strategy 2	Player 1 Wins	Draw	Player 2 Wins	...
Strategy 3	Player 2 Wins	Player 2 Wins	Draw	...
□□□	□□□	□□□	□□□	...

2.20.1 Type of Game Strategies

- The simplest mathematical description of a game is the strategic form, mentioned in the introduction. For a two-person zero-sum game, the payoff function of Player II is the negative of the payoff of Player I, so we may restrict attention to the single payoff function of Player I, which we call here A .
 - The strategic form, or normal form, of a two-person zero-sum game is given by a triplet (X, Y, A) , where
 - (1) X is a nonempty set, the set of strategies of Player I
 - (2) Y is a nonempty set, the set of strategies of Player II
 - (3) A is a real-valued function defined on $X \times Y$. (Thus, $A(x, y)$ is a real number for every $x \in X$ and every $y \in Y$.)
 - The interpretation is as follows. Simultaneously, Player I chooses $x \in X$ and Player II chooses $y \in Y$, each unaware of the choice of the other. Then their choices are made known and I wins the amount $A(x, y)$ from II.
 - This is a very simple definition of a game; yet it is broad enough to encompass the finite combinatorial games and games such as tic-tac-toe and chess.
 - On the basis of how many times Player I or Player II is winning the game, following strategies can be discussed.
1. **Equalizing Strategy :** A strategy that produces the same average winnings no matter what the opponent does is called an **equalizing strategy**.
 2. **Optimal Strategy :** If I has a procedure that guarantees him at least $A(x, y)$ amount on the average, and II has a procedure that keeps her average loss to at most $A(x, y)$. Then $A(x, y)$ is called the value of the game, and the procedure each uses to insure this return is called an optimal strategy or a minimax strategy.
 3. **Pure Strategies and Mixed Strategies :** It is useful to make a distinction between a pure strategy and a mixed strategy. We refer to elements of X or Y as pure strategies. The more complex entity that chooses among the pure strategies at random in various proportions is called a mixed strategy.

2.20.2 Type of Games

Game can be classified under deterministic or probabilistic category. Let's see what we mean by deterministic and Probabilistic.

(a) Deterministic

- It is a fully observable environment. When there are two agents playing the game alternatively and the final results of the game are equal and opposite then the game is called deterministic.
- Take example of tic-tac-toe where two players play a game alternatively and when one player wins a game then other player losses game.

(b) Probabilistic

- Probabilistic is also called as non-deterministic type. It is opposite of deterministic games, where you can have multiple players and you cannot determine the next action of the player.
- You can only predict the probability of the next action. To understand probabilistic type you can take example of card games.
- Another way of classification for games can be based on exact/perfect information or based on inexact / approximate information. Now, let us understand these terms.
 1. **Exact/perfect information** : Games in which all the actions are known to other player is called as game of exact or perfect information. For example tic-tac-toe or board games like chess, checkers.
 2. **Inexact / approximate information** : Game in which all the actions are not known to other players (or actions are unpredictable) is called game of inexact or approximate information. In this type of game, player's next action depends upon who played last, who won last hand, etc. For example card games like hearts.

Consider following games and see how they are classified into various types of games based on the parameters which we have learnt in above sections :

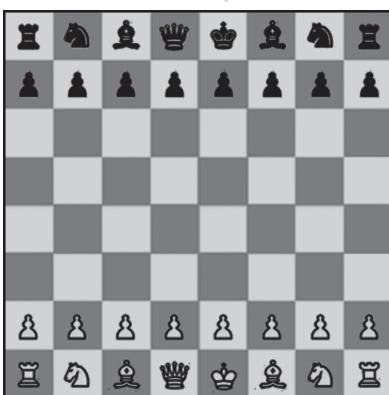
Table 2.20.2 : Types of game

	Exact/perfect information	Inexact / approximate information
Deterministic	<ul style="list-style-type: none">- Chess- Checkers	<ul style="list-style-type: none">- Battleships- Card Game (Hearts)
Probabilistic	<ul style="list-style-type: none">- Ludo- Monopoly	<ul style="list-style-type: none">- Scrabble- Poker

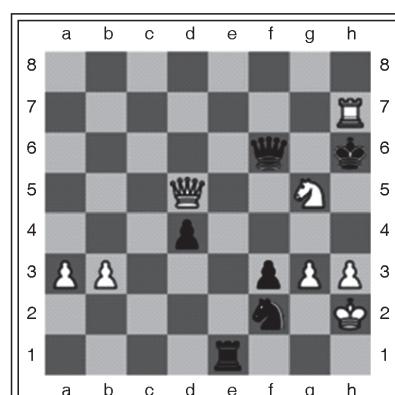
Now, let us try to learn about few games mention in Table 2.20.2.

2.20.2(A) Chess

- Chess comes under deterministic and exact/perfect information category. This game is a two person, zero-sum game.
- In chess both players can see chess board positions so there is no secrecy and players don't play at the same time they play one after the other in an order.
- Thus this game has perfect environment to test artificial intelligence techniques. In 1990's a computer Deep Blue II defeated Garry Kasparov who was world champion at that time. This example is given to understand how artificial intelligence can be used in decision making.



(a) Chess board



**(b) Deep Blue II vs Garry Kasparov
final position game 1**

Fig. 2.20.2

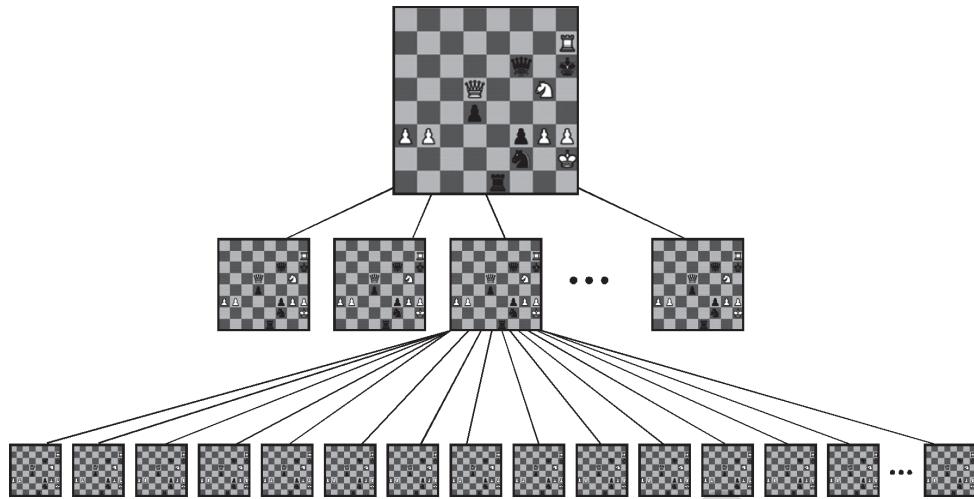


Fig. 2.20.3 : Chess game tree

2.20.2(B) Checkers

- Checkers comes under deterministic and exact/perfect information category. This game is a two person game where both players can see board positions, so there is no secrecy and players play one after the other in an order.
- In 1990's a computer program name Chinook (also called draughts) was developed which defeated human world champion Marion Tinsley.

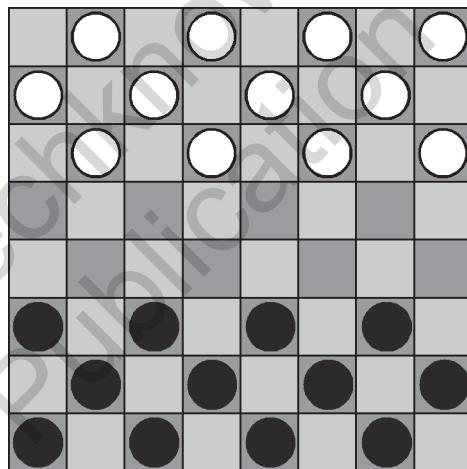


Fig. 2.20.4 : Checkers board

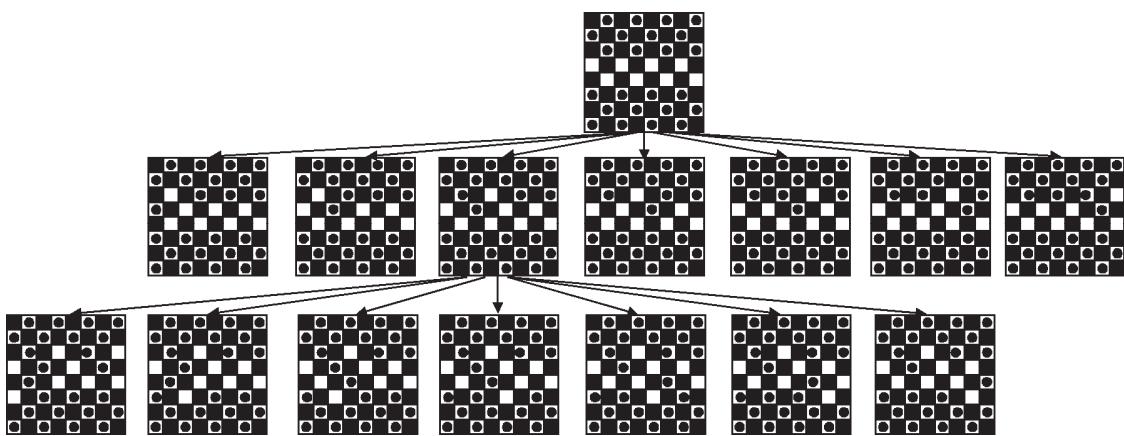
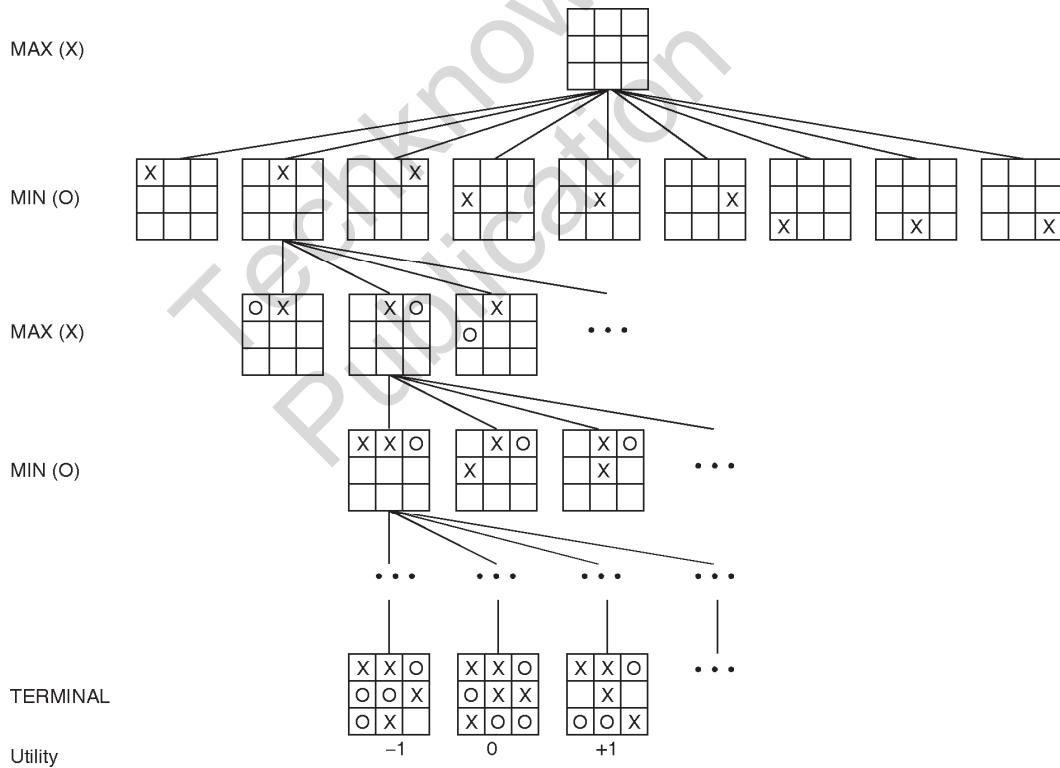


Fig. 2.20.5 : Checkers game tree

2.20.3 What is Game Tree?

MU – Dec. 15**Q.** Draw game tree of tic-tac-toe problem.**(Dec. 15, 5 Marks)**

- We saw game tree of chess and checkers in earlier section.
- Game tree is defined as a directed graph with nodes and edge. Here nodes indicate positions in a game and edges indicate next actions.
- Let us try to understand what a game tree is with the help of Tic-Tac-Toe example. Tic-Tac-Toe is a 2-player game, it is deterministic and every player plays when his/her turn come. Game tree has a **Root Node** which give the starting position of the game board, which is a blank 3×3 grid.
- Say in given example player 1 takes 'X' sign. Then MAX(X) indicates the board for the best single next move. Also it indicates that it is player 1's turn. (Remember that initial move is always indicated with a MAX).
- If the node is labelled as MIN then it means that it is opponents turn (i.e. player 2's turn) and the board for the best single next move is shown.
- Possible moves are represented with the help of lines. Last level shows terminal board position, which illustrates end of the game instance, Here, we get zero as sum of all the play offs. Terminal state gives winning board position. Utility indicates the play off points gained by the player (-1, 0, +1).
- In a similar way we can draw a game tree for any artificial intelligence based games.

**Fig. 2.20.3 : Tic-tac-toe game tree**

We can formulate a game as a search problem as follows :

Initial state	It gives the starting position of the game board.
Operators	They define the next possible actions.

Terminal state	Which indicates that all instance of game are over.
Utility	It displays a number which indicates if the game was won or lost or it was draw.

- From Tic-Tac-Toe game's example you can understand that for a 3×3 grid, two player game, where the game tree is relatively small (It has 9! terminal nodes), still we cannot draw it completely on one single page.
- Imagine how difficult it would be to create a game tree for multi-player games or for the games with bigger grid size.
- Many games have huge search space complexity. Games have limitation over the time and the amount of memory space it can consume. Finding an optimal solution is not feasible most of the times, so there is a need for approximation.
- Therefore there is a need for an algorithm which will reduce the tree size and eventually will help in reducing the processing time and in saving memory space of the machine.
- One method is **pruning** where, only the required parts, which improve quality of output, of the tree are kept and remaining parts are removed.
- Another method is **heuristic method** (it makes use of an **evaluation function**) it does not require exhaustive research. This method depends upon readily available information which can be used to control problem solving.

2.21 MiniMax Algorithm

- Minimax algorithm evaluates decision based on the present status of the game. This algorithm needs deterministic environment with perfect/exact information.
- Minimax algorithm directly implements the defining equation. Every time based on the **successor state minimax value** is calculated with the help of simple **recursive computation**.
- In case of minimax algorithm the selected action with **highest minimax value** should be equal to the best possible payoff (outcome) against best play.

2.21.1 Minimax Algorithm

Take example of tic-tac toe game to understand minimax algorithm. We will take a random stage.

Step 1 : Create an entire game tree including all the terminal states.

Start action : 'O'

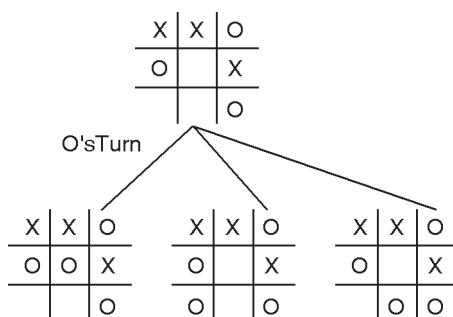


Fig. 2.21.1

Next action : 'X'

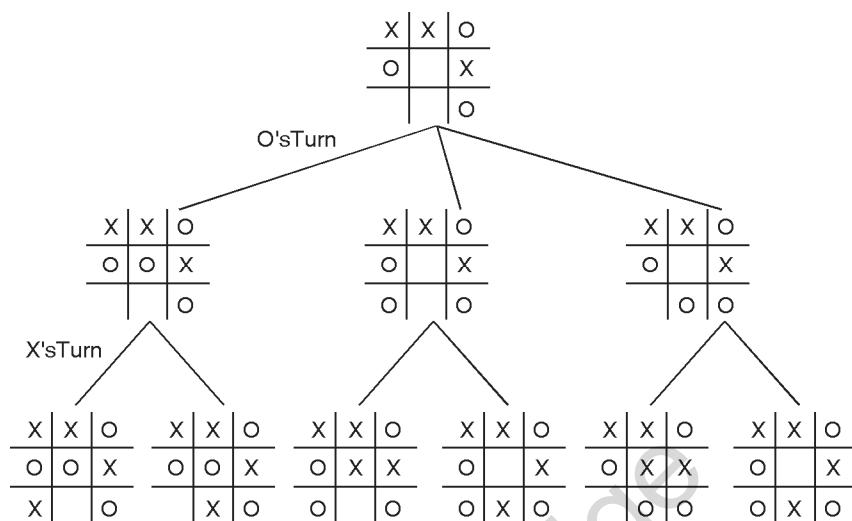


Fig. 2.21.2

Next action : 'O'

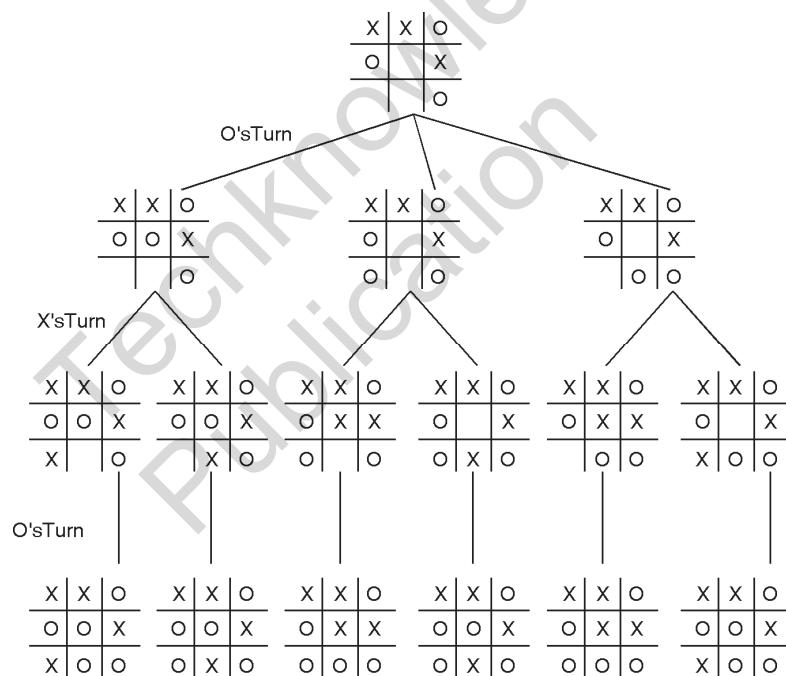


Fig. 2.21.3

Step 2 : For every terminal state find out utility (playoff points gained by every terminal state). Terminal position where 1 means win and 0 means draw.

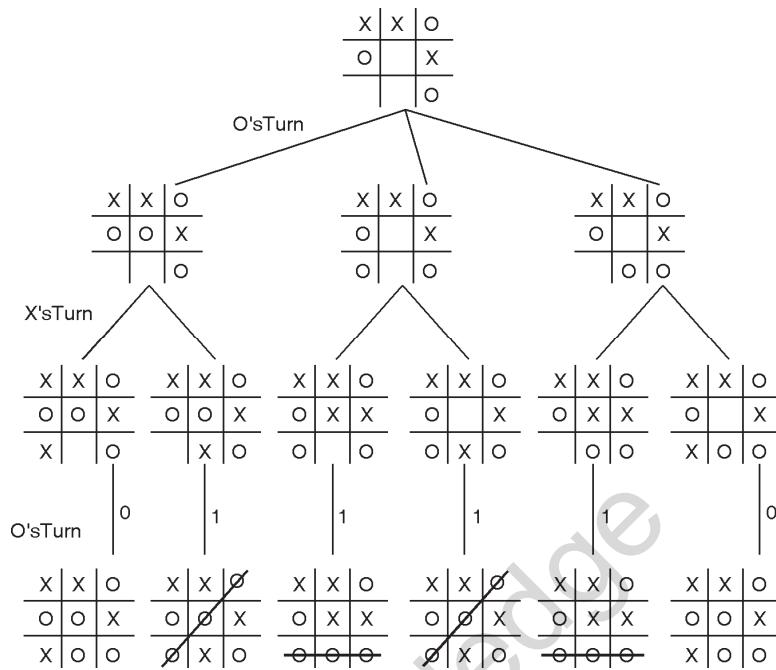


Fig. 2.21.4

Step 3 : Apply MIN and MAX operators on the nodes of the present stage and propagate the utility values upward in the three.

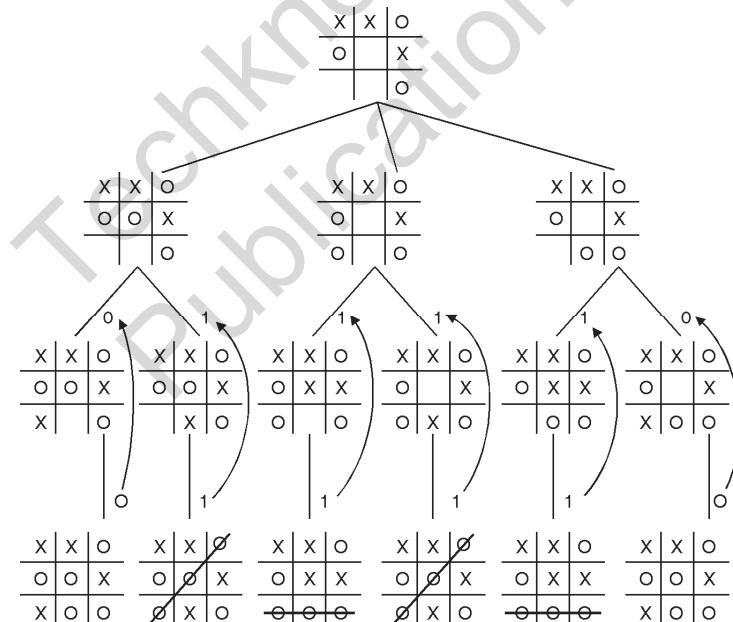


Fig. 2.21.5

Step 4 : With the max (of the min) utility value (payoff value) select the action at the root node using minimax decision.

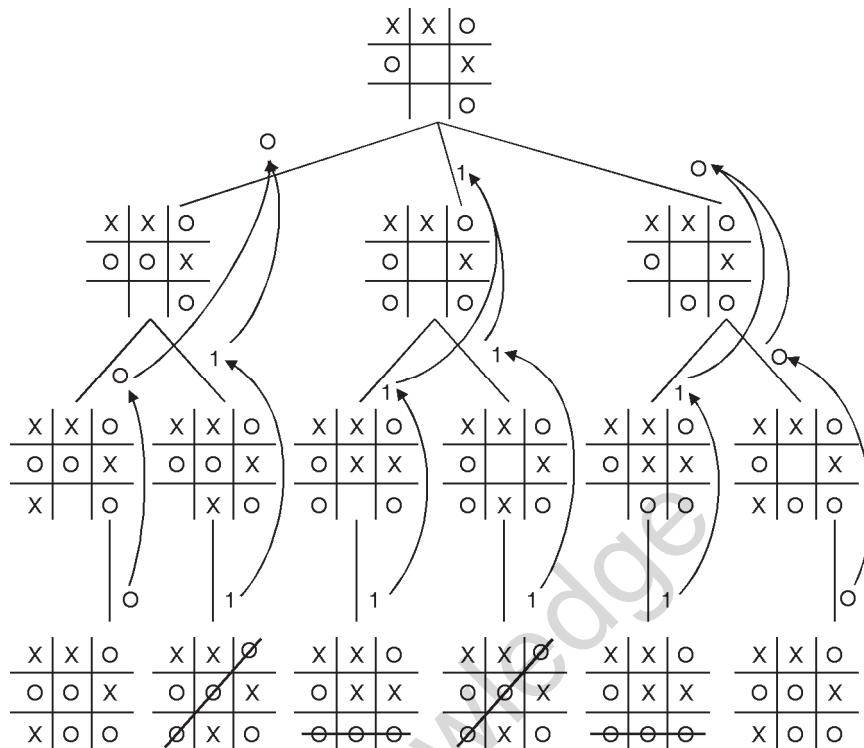


Fig. 2.21.6

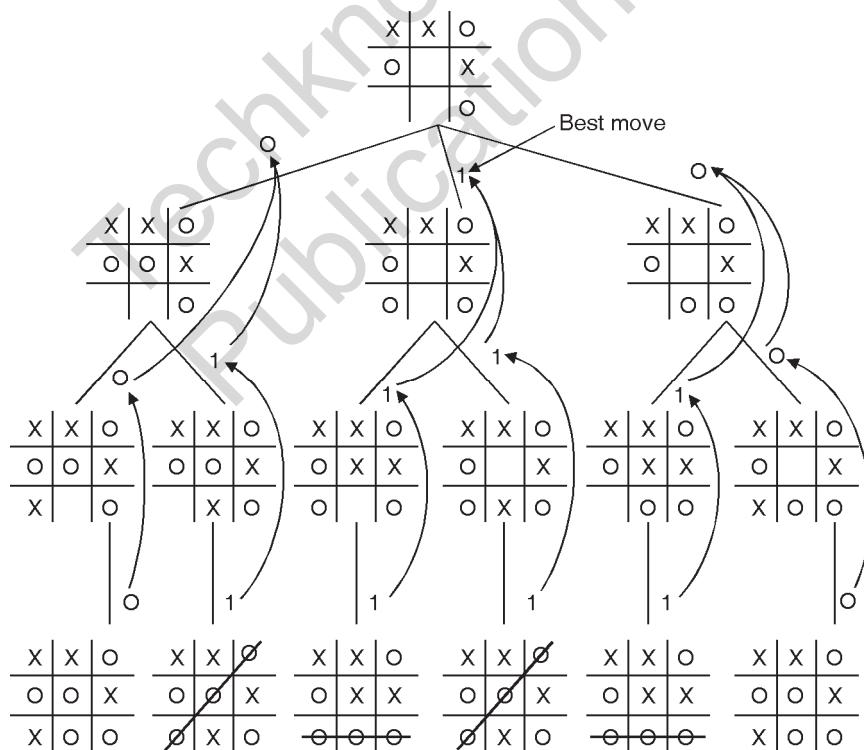


Fig. 2.21.7

(In case of Steps 2 and 3 we are assuming that the opponent will play perfectly as per our expectation)

2.21.2 Properties of Minimax Algorithm

- It is considered as Complete if the game tree size is finite.
- It is considered Optimal when it is played against an optimal number of opponents.
- Time complexity of minimax algorithm is indicated as $O(b^m)$.
- Space complexity of minimax algorithm is indicated by $O(b^m)$ (using depth-first exploration approach).
- For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games.
- Exact solution is completely infeasible in most of the games.

2.22 Alpha Beta Pruning

- Pruning means cutting off. In game search it resembles to clipping a branch in the search tree, probably which is not so fruitful.
- At any choice point along the path for max, α is considered as the value of the best possible choice found i.e., highest-value. For each “X”, if “X” is worse i.e. lesser value than α value then, MAX will avoid it. Similarly we can define β value for MIN.
- α - β pruning is an extension to minimax algorithm where, decision making process need not consider each and every node of the game tree.
- Only the important nodes for quality output are considered in decision making. Pruning helps in making the search more efficient.
- **Pruning** keeps only those parts of the tree which contribute in improving the quality of the result remaining parts of the tree are removed.
- Consider the following game tree :

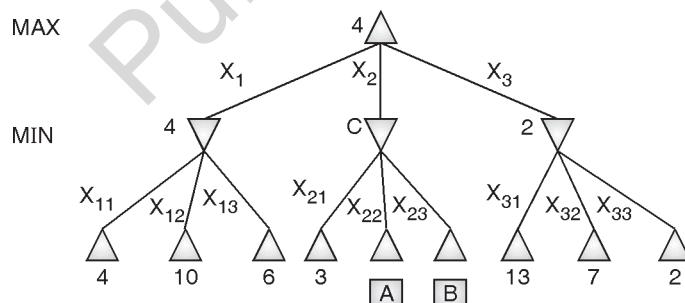
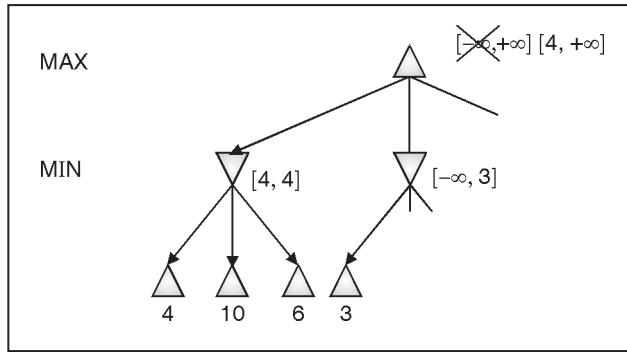
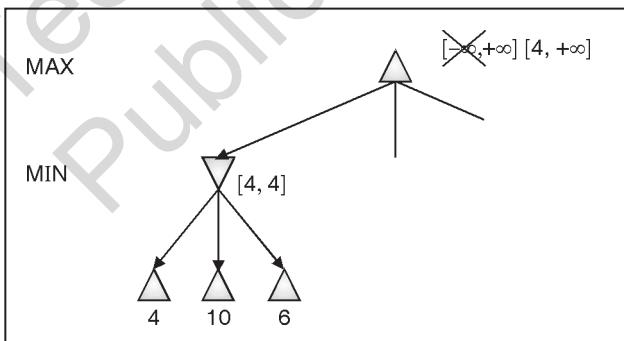
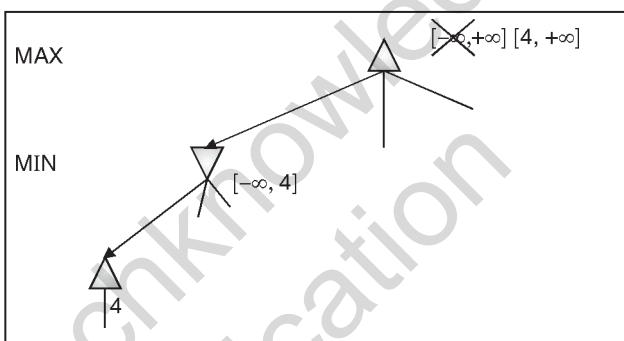
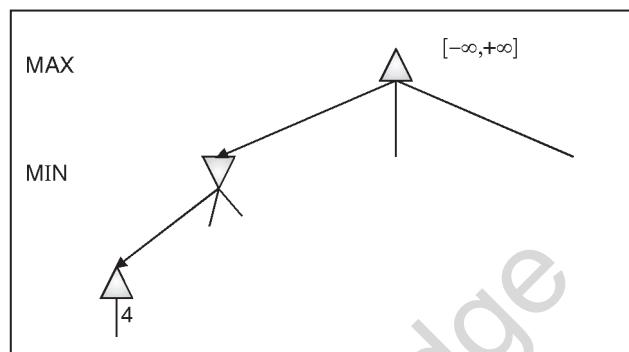
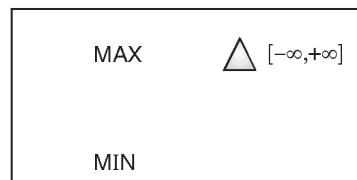


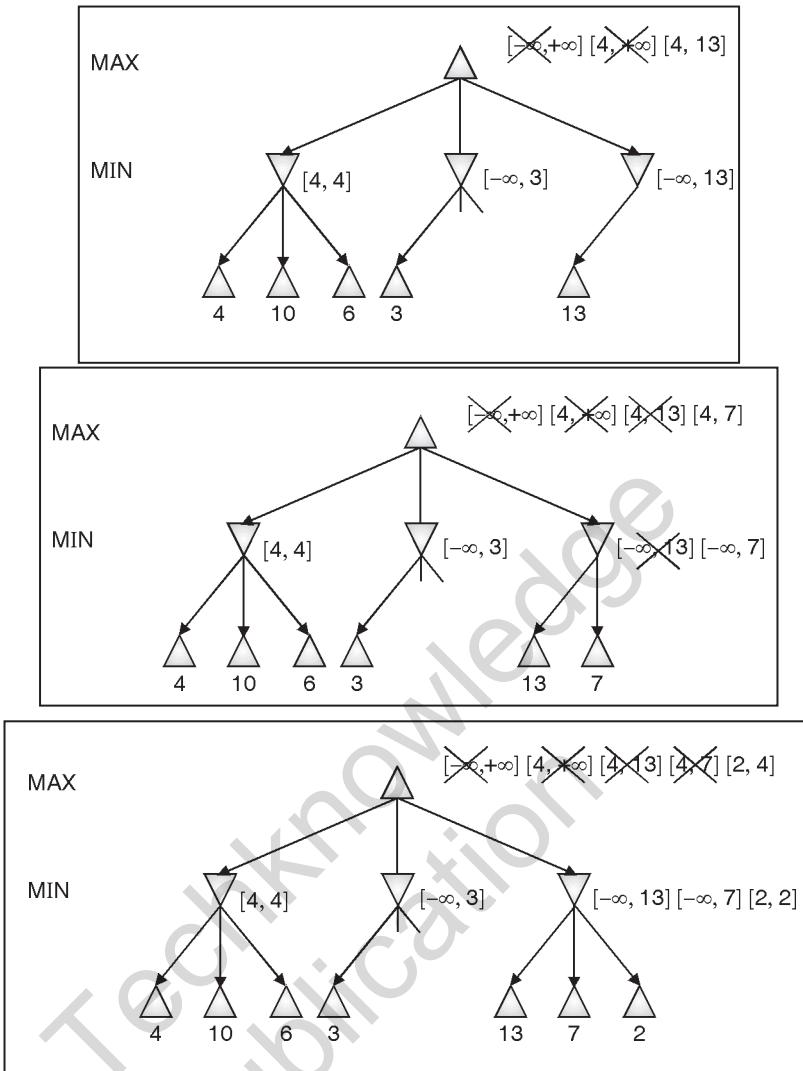
Fig. 2.22.1

- For which we have to calculate minimax values for root node.

$$\begin{aligned}\text{Minimax value of root node} &= \text{Max}(\min(4, 10, 6), \min(3, A, B), \min(13, 7, 2)) \\ &= \text{Max}(4, C, 2) \quad (C \leq 3) = 4\end{aligned}$$

- Let us see how to check this step by step.





- So in this example we have pruned 2 β and 0 α branches. As the tree is very small, you may not appreciate the effect of branch pruning; but as we consider any real game tree, pruning creates a significant impact on search as far as the time and space is concern.

2.22.1 Example of α - β Pruning

Ex. 2.22.1 : Explain Min-Max and Alpha Beta pruning algorithm with following example.

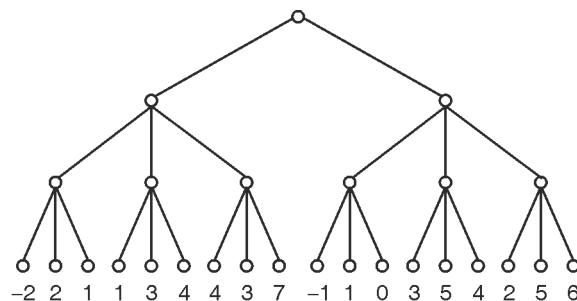
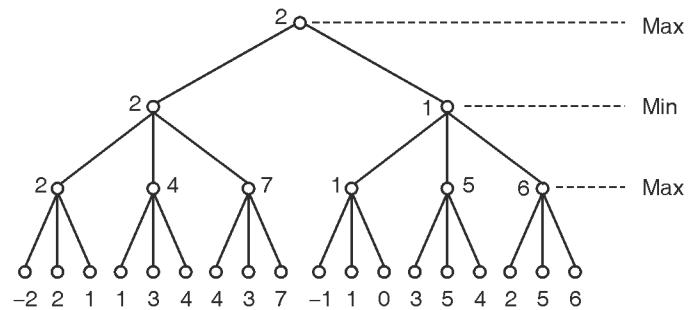


Fig. P. 2.22.1 : Game Tree

Soln. : Using min-max algorithm



(a) Min-max solution with optimal path shown in bold

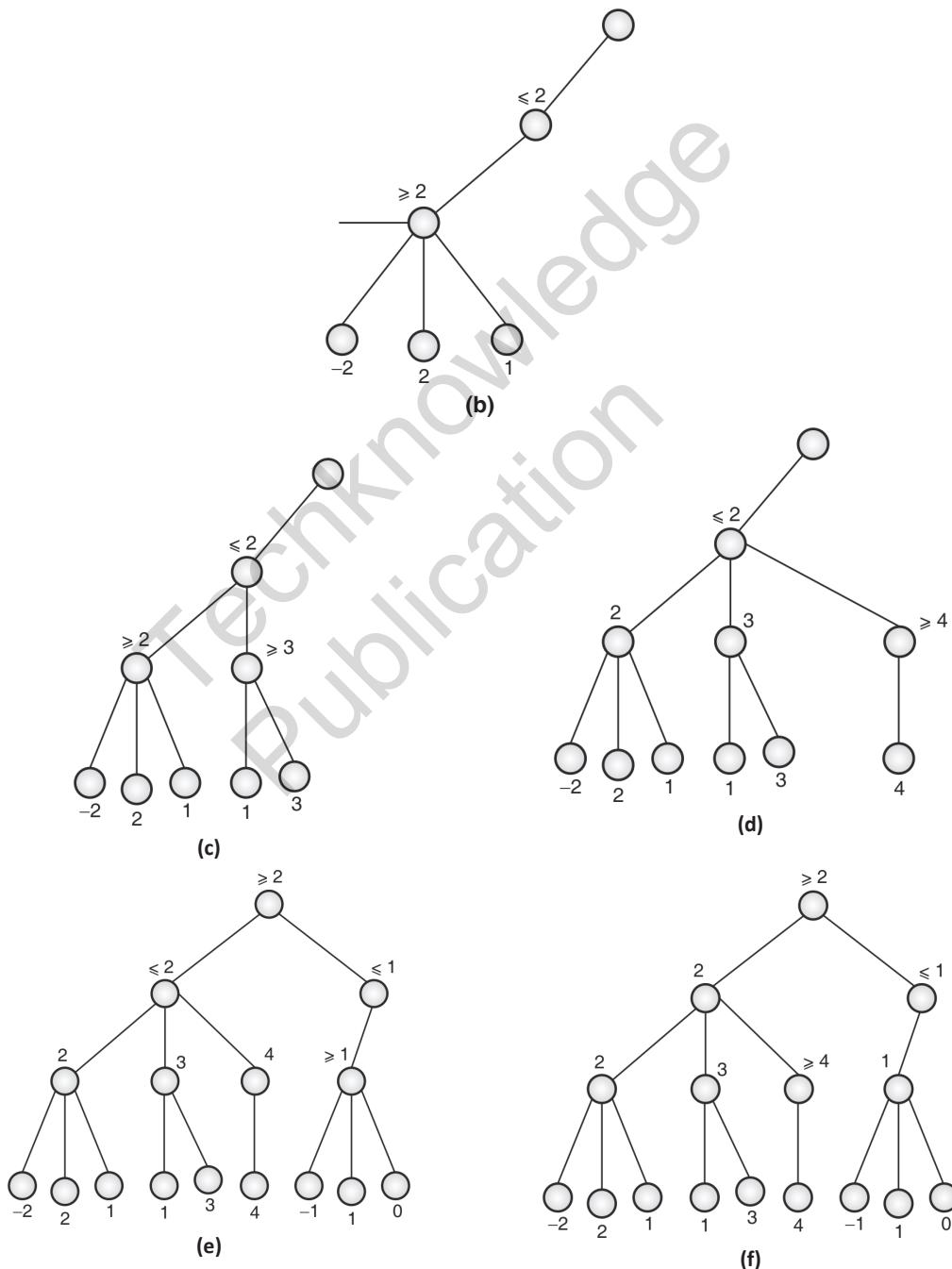


Fig. P. 2.22.1

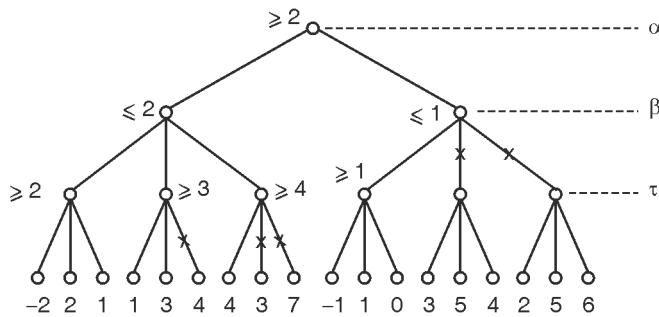
(g) Solution by α - β pruning

Fig. P. 2.22.1

Total pruned branches

$$\alpha - \text{cuts} = 2$$

$$\beta - \text{cuts} = 3$$

Ex. 2.22.2 : Perform α - β cutoff on the following.

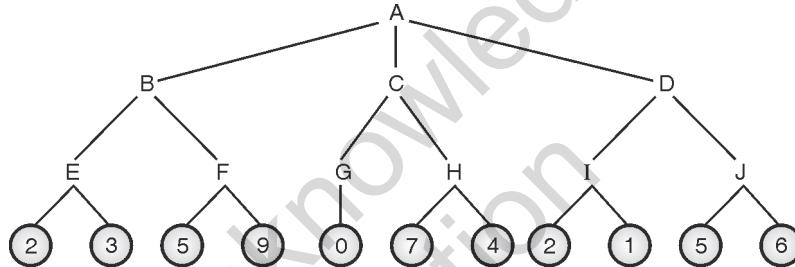


Fig. P. 2.22.2

Soln. :

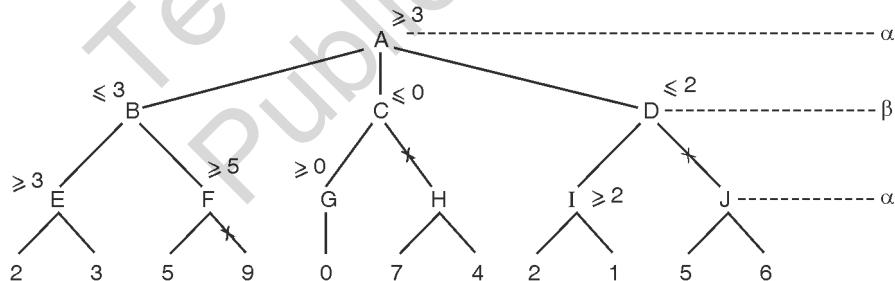


Fig. P. 2.22.2(a)

$$\text{No. of } \alpha - \text{cuts} = 1$$

$$\text{No. of } \beta - \text{cuts} = 2$$

Ex. 2.22.3 : Apply alpha-beta pruning on example given in Fig. P.2.22.3 considering first node as max.

MU - May 16, 10 Marks

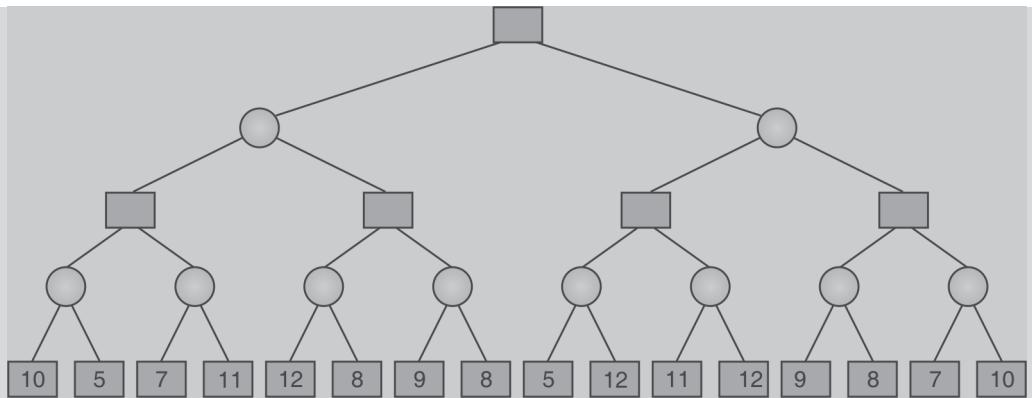
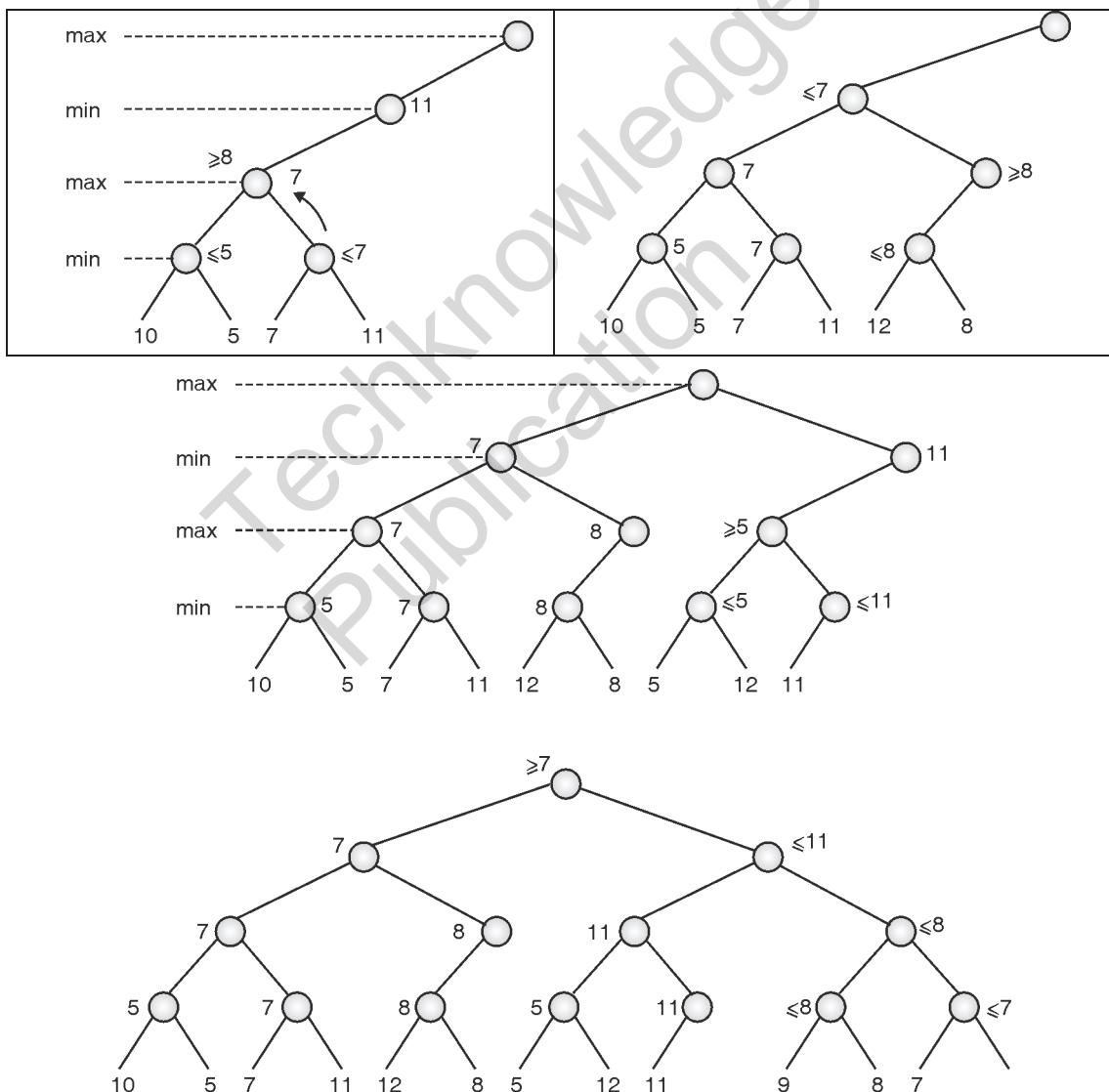
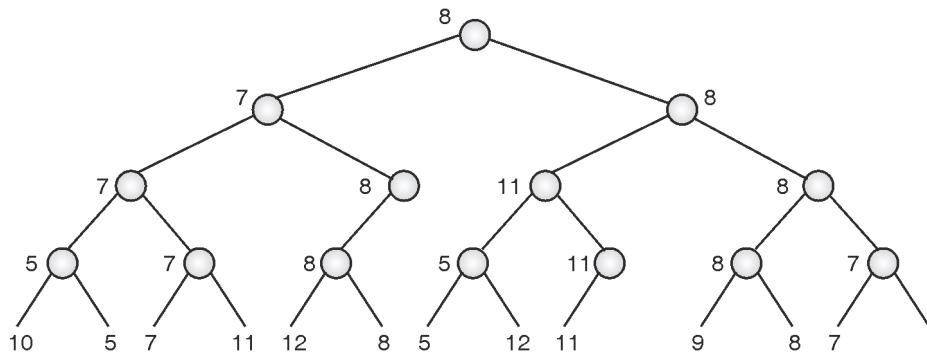


Fig. P.2.22.3

Soln. :



2.22.3 Properties of α - β

- Final results are not affected by pruning.
- Ordering of Good actions helps in improving effectiveness of pruning technique.
- If there is exact/perfect ordering then we can get time complexity as $O(b^{m/2})$.
- Depth of search is doubled with pruning.

Review Questions

- Q. 1 Differentiate between BFS and DFS.
- Q. 2 How the drawbacks of DFS are overcome by DLS and DFID?
- Q. 3 Compare and contrast all the un-informed searching techniques.
- Q. 4 Write short note on bidirectional search.
- Q. 5 Write a note on BFA and Uniform cost search.
- Q. 6 Compare and contrast DFS, FLS and DFID.
- Q. 7 What are various informed search techniques? Explain A* with example.
- Q. 8 Compare Best First Search and A* with an example.
- Q. 9 Write algorithm of steepest ascent hill climbing. And compare it with simple hill climbing.
- Q. 10 What are the limitations of hill climbing? How can we solve them?
- Q. 11 Write algorithm for Best first search and specify its properties.
- Q. 12 What is the difference between best first and greedy best first search? Explain with example.
- Q. 13 What is heuristic function? What are the qualities of a good heuristic?
- Q. 14 Write short note on simulated annealing and local beam search.
- Q. 15 Compare and contrast Simulated annealing with Hill climbing.
- Q. 16 How the definition of heuristic affects the search process? explain with suitable example.
- Q. 17 Write short note on behavior of A* in case of underestimating and overestimating Heuristic.
- Q. 18 Discuss admissibility of A* in case of optimality.
- Q. 19 Explain SMA* algorithm with example. When should we choose SMA* given options?

Q. 20 Write a short notes on :

- (a) Game types
- (b) Zero-sum game
- (c) Relevant aspects of AI games
- (d) Features of AI game

Q. 21 Explain minimax algorithm with an example and give its properties.

Q. 22 Give α - β pruning algorithm with an example and it's properties, also explain why is it called α - β pruning.

Q. 23 What is adversarial search?

Q. 24 Apply alpha-beta pruning on example given in Fig. Q. 24 considering first node as max.

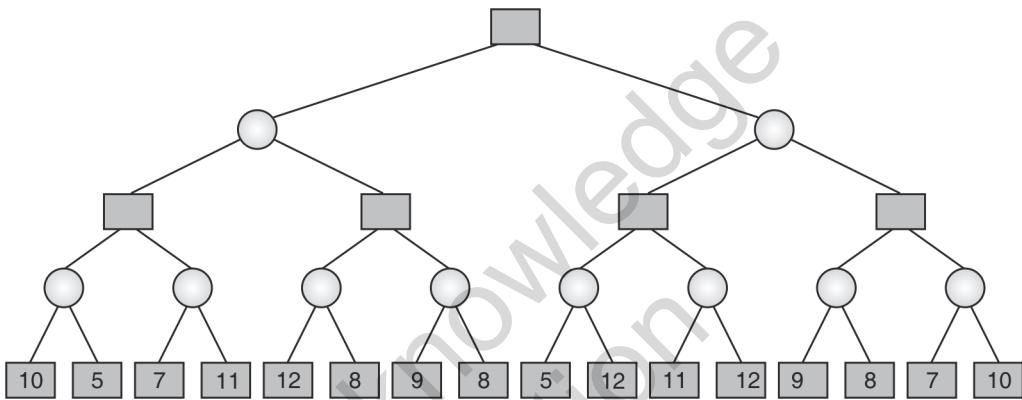


Fig. Q. 24





Knowledge, Reasoning and Planning

Syllabus

- 3.1 Knowledge based agents
- 3.2 First order logic: syntax and Semantic, Knowledge Engineering in FOL Inference in FOL : Unification, Forward Chaining, Backward Chaining and Resolution
- 3.3 Planning Agent, Types of Planning: Partial Order, Hierarchical Order, Conditional Order

- Understanding theoretical or practical aspects of a subject is called as knowledge. We can gain knowledge through experience acquired based on the facts, information, etc. about the subject.
- After gaining knowledge about some subject we can apply that knowledge to derive conclusions about various problems related to that subject based on some reasoning.
- We have studied various types of agents in chapter 1. In this chapter we are going to see what is “knowledge based agent”, with a very interesting game example.
- We are also going to study how do they store knowledge, how do they infer next level of knowledge from the existing set. In turn, we are studying various knowledge representation and inference methods in this chapter.

3.1 A Knowledge Based Agent

- As shown Fig. 3.1.1, a knowledge based agents can be described at different levels : **Knowledge Base (KB)** and an **Inference Engine**.

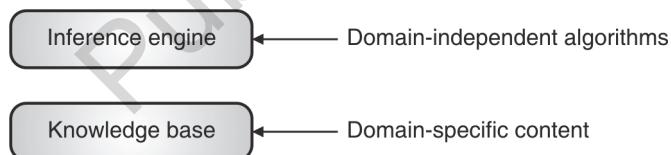


Fig. 3.1.1 : Levels of knowledge base

1. Knowledge level:

- Knowledge level is a base level of an agent, which consists of domain-specific content.
- In this level agent has facts/information about the surrounding environment in which they are working, it does not consider the actual implementation.

2. Implementation level:

- Implementation level consists of domain independent algorithms. At this level, agents can recognize the data structures used in knowledge base and algorithms which use them. For example, propositional logic and resolution. (We will be learning about logic and resolution in this chapter)
- Knowledge based agents are crucial to use in partially observable environments. Before choosing any action, knowledge based agents make use of the existing knowledge along with the current inputs from the environment in order to infer hidden aspects of the current state.



- As we have learnt that knowledge base is a set of representations of facts/information about the surrounding environment (real world). Every single representation in the set is called as a **sentence** and sentences are expressed with the help of formal representation language. We can say that sentence is a statement which is a set of words that express some truth about the real world with the help of knowledge representation language.
- Declarative approach of building an agent makes use of TELL and ASK mechanism.
 - TELL the agent, about surrounding environment (what it needs to know in order to perform some action). TELL mechanism is similar to taking input for a system.
 - Then the agent can ASK itself what action should be carried out to get desired output. ASK mechanism is similar to producing output for a system. However, ASK mechanism makes use of the knowledge base to decide what it should do.
- TELL and ASK mechanism involve inference. When you run ASK function, the answer is generated with the help of knowledge base, based on the knowledge which was added with TELL function previously.
 - **TELL(K)** :Is a function that adds knowledge K to the knowledge base.
 - **ASK(K)** :Is a function that queries the agent about the truth of K.
- An agent carries out following operations: First, it TELLS the knowledge base about facts/information it perceives with the help of sensors. Then, it ASKS the knowledge base what action should be carried out based on the input it has received. Lastly, it performs the selected action with the help of effectors.

3.1.1 Architecture of a KB Agent

Knowledge based agents can be implemented at three levels namely, knowledge level, logical level and implementation level.

- | | |
|-------------------------|------------------|
| 1. Knowledge level | 2. Logical level |
| 3. Implementation level | |

1. Knowledge level :

- It is the most abstract level of agent implementation. The knowledge level describes agent by saying what it knows. That is what knowledge the agent has as the initial knowledge.
- Basic data structures and procedures to access that knowledge are defined in his level. Initial knowledge of knowledge base is called as background knowledge.
- Agents at the knowledge level can be viewed as an agent for which one only need to specify what the agent knows and what its goals are in order to specify its behaviour, regardless of how it is to be implemented.
- **For example :**A taxi driving agent might know that the Golden Gate Bridge connects San Francisco with the Marin county.

2. Logical level :

- At the logical level, the knowledge is encoded into sentences. This level uses some formal language to represent the knowledge the agent has. The two types of representations we have are propositional logic and first order or predicate logic.
- Both these representation techniques are discussed in detail in the further sections.
- **For example:** Links(Golden Gate Bridge, San Francisco, Marin County)

3. Implementation level :

- In implementation level, the physical representation of logical level sentences is done. This level also describes data structures used in knowledge base and algorithms that used for data manipulation.

- **For example :** Links(Golden Gate Bridge, San Francisco, Marin County)

```
function KB – Agent (percept) returns an action
    static: KB, a knowledge base
        t, a counter, initially 0, indicating time
        TELL (KB, MAKE – PERCEPT-SENTENCE(percept, t))
        action← ASK (KB, MAKE-ACTION-QUERY(t))
        TELL(KB, MAKE-ACTION-SENTENCE(action,t))
        t←t + 1
    returns action
```

Fig. 3.1.2 : General function of knowledge based agent

- Fig. 3.1.2 is the general implementation of knowledge based agent. TELL and ASK are the sub procedures implemented to perform the respective actions.
- **The knowledge base agent must be able to perform following tasks :**
 - Represent states, actions, etc.
 - Incorporate new precepts.
 - Update internal representations of the world.
 - Deduce hidden properties of the world.
 - Deduce appropriate actions.

3.2 The WUMPUS World Environment

- You have learnt about vacuum world problem, block world problem so far. Similarly we have WUMPUS world problem. Fig.3.2.1 shows the WUMPUS world.
- WUMPUS is an early computer game also known as “Hunt the Wumpus”. WUMPUS was developed by Gregory Yob in 1972/1973. It was originally written in **BASIC (Beginner's All-purpose Symbolic Instruction Code)**.
- WUMPUS is a map-based game. Let's understand the game :
 - WUMPUS world is like a cave, which represents number of rooms, rooms, which are connected by passage ways. We will take a 4×4 grid to understand the game.
 - WUMPUS is a monster who lives in one of the rooms of the cave. WUMPUS eats the player (agent) if player (agent) comes in the same room. Fig. 3.2.1 shows that room (3, 1) where WUMPUS is staying.
 - Player (agent) starts from any random position in cave and has to explore the cave. We are starting from (1, 1) position.
- There are various sprites in the game like pit, stench, breeze, gold, and arrow. Every sprite has some feature. Let's understand this one-by-one :
 - Few rooms have bottomless pits, which trap the player (agent) if he comes to that room. You can see in the Fig. 3.2.1 that room (1,3), (3,3) and (4,4) have bottomless pit. Note that even WUMPUS can fall into a pit.
 - Stench experienced in a room, which has a WUMPUS in its neighbourhood room. See the Fig. 3.2.1, here room (2,1), (3,2) and (4,1) have Stench.
 - Breeze is experienced in a room, which has a pit in its neighbourhood room. Fig. 3.2.1 shows that room (1,2), (1,4), (2,3), (3,2), (3,4) and (4,3) consists of Breeze.
 - Player (Agent) has arrows and he can shoot these arrows in straight line to kill WUMPUS.

- One of the rooms consists of gold, this room glitters. Fig.3.2.1 shows that room (3,2) has Gold.
- Apart from above features player (agent) can accept two types of percepts which are: Bump and scream. A bump is generated if player (agent) walks into a wall. While a sad scream created everywhere in the cave when the WUMPUS is killed.

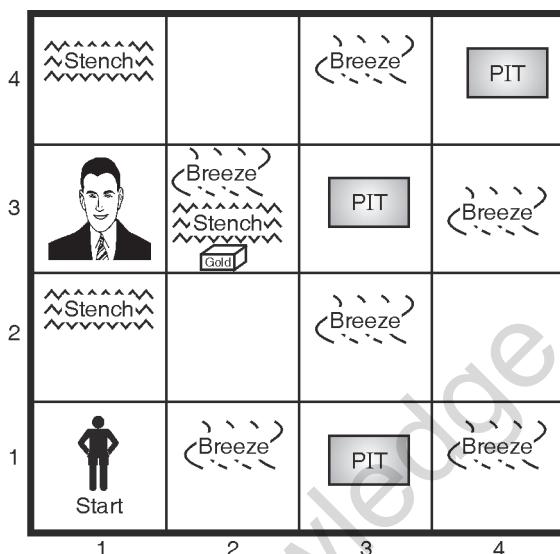


Fig. 3.2.1 : The WUMPUS World

3.2.1 Description of the WUMPUS World

- An agent receives percepts while exploring the rooms of cave. Every percepts can be represented with the help of five element list, which is [stench, breeze, glitter, bump, scream]. Here player (agent) cannot perceive its own location.
- If the player (agent) gets percept as [Stench, Breeze, None, None, None]. Then it means that there is a stench and a breeze, but no glitter, no bump, and no scream in the WUMPUS world at that position in the game.
- Let's take a look at the actions which can be performed by the player(agent) in WUMPUS World :
 - **Move** : To move in forward direction,
 - **Turn** : To turn right by 90 degrees or left by 90 degrees,
 - **Grab** : To pick up gold if it is in the same room as the player(agent),
 - **Shoot** : To Shoot an arrow in a straight line in the direction faced by the player (agent).
- These actions are repeated till the player (agent) kills the WUMPUS or if the player (agent) is killed. If the WUMPUS is killed then it is a winning condition, else if the player(agent) is killed then it is a losing condition and the game is over.
- Game developer can keep a restriction on the number of arrows which can be used by the player(agent). So if we allow agent to have only one arrow, then only the first shoot action will have some effect. If this shoot action kills the WUMPUS then you win the game, otherwise it reduces the probability of winning the game.
- Lastly there is a die action : It takes places automatically if the agent enters in a room with a bottomless pit or in a room with WUMPUS. Die action is irreversible.

Goal of the game

- Main aim of the game is that player (agent) should grab the gold and return to starting room (here its (1,1)) without being killed by the monster (WUMPUS).



- Award and punishment *points* are assigned to a player (Agent) based on the actions it performs. **Points can be given as follows:**
 - 100 points are awarded if player (agent) comes out of the cave with the gold.
 - 1 point is taken away for every action taken.
 - 10 points are taken away if the arrow is used.
 - 200 points are taken away if the player (agent) gets killed.

3.2.2 PEAS Properties of WUMPUS World

MU - May 13, Dec. 14

Q. Give PEAS descriptors for WUMPUS world

(May 13, Dec. 14, 3 Marks)

1. Performance measure

- +100 for grabbing the gold and coming back to the starting position,
- – 200 if the player(agent) is killed.
- – 1 per action,
- – 10 for using the arrow.

2. Environment

- Empty Rooms.
- Room with WUMPUS.
- Rooms neighbouring to WUMPUS which are smelly.
- Rooms with bottomless pits
- Rooms neighbouring to bottomless pits which are breezy.
- Room with gold which is glittery.
- Arrow to shoot the WUMPUS.

3. Sensors (assuming a robotic agent)

- Camera to get the view
- Odour sensor to smell the stench
- Audio sensor to listen to the scream and bump

4. Effectors (assuming a robotic agent)

- Motor to move left, right
- Robot arm to grab the gold
- Robot mechanism to shoot the arrow

The WUMPUS world agent has following characteristics :

- | | | |
|---------------------|------------------|-----------------|
| 1. Fully observable | 2. Deterministic | 3. Episodic |
| 4. Static | 5. Discrete | 6. Single agent |

3.2.3 Exploring a WUMPUS World

- Let's try to understand the WUMPUS world problem in step by step manner. Keep Fig. 3.2.2 as a reference figure.

4.1	4.2	4.3	4.4
3.1	3.2	3.3	3.4
2.1	2.2	2.3	2.4
1.1 [A] OK	1.2 OK	1.3	1.4

[A] - Agent
B – Breeze
G - Glitter, Gold
OK – Safe square
P - Pit
S – Stench
V – Visited
W –Wumpus

Fig. 3.2.2(a) : WUMPUS world with player in room (1,1)

- The knowledge base initially contains only the rules (facts) of the WUMPUS world environment.

Step 1 : Initially the player(agent) is in the room (1,1). See Fig. 3.2.2(a).

The first percept received by the player is [none, none, none, none, none]. (remember percept consists of [stench, breeze, glitter, bump, scream])

Player can move to room(1,2) or (2,1) as they are safe cells.

Step 2 : Let us move to room (1,2). See Fig.3.2.2(b).

4.1	4.2	4.3	4.4
3.1	3.2	3.3	3.4
2.1	2.2	2.3	2.4
1.1 V OK	1.2 [A] B OK	1.3 P?	1.4

[A] - Agent
B – Breeze
G - Glitter, Gold
OK – Safe square
P - Pit
S – Stench
V – Visited
W –Wumpus

Fig. 3.2.2(b) : WUMPUS world with player in room (1,2)

- As room (1,1) is visited you can see "V" mark in that room. The player receives following percept :[none, breeze, none, none, none].
- As breeze percept is received room (1,2) is marked with "B" and it can be predicted that there is a bottomless pit in the neighboring room.
- You can see that room (1,3) and room (2,2) is marked with "P?". So room (1,3) and (2,2) is not safe to move in. Thus player should return to room (1,1)and try to find other, safe room to move to.

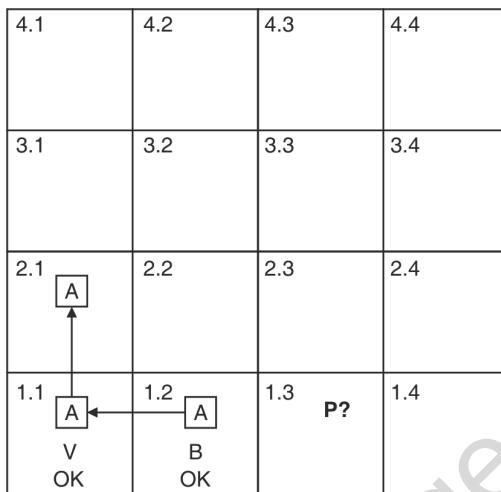
Step 3 :

Fig. 3.2.2(c) WUMPUS world with player moving back to room (1,1) and then moves to other safe room (2,1).

- As seen in Fig.3.2.2(c). Player is now in room (2,1), where it receives a percept as follows : [stench, none, none, none, none] which means that there is a WUMPUS in neighboring room (i.e. either room (2,2) or (3,1) has WUMPUS).
- As we did not get breeze percept in this room, we can understand that room (2,2) cannot have any pit and from step 2 we can understand that room (2,2) cannot have WUMPUS because room (1,2) did not show stench percept.
- Thus room(2,2) is safe to move in.

Step 4 : Player receives [none, none, none, none, none] percept when it comes to room (2,2). From Fig. 3.2.2(d) you can understand that room (2,3) and room (3,2) are safe to move in.

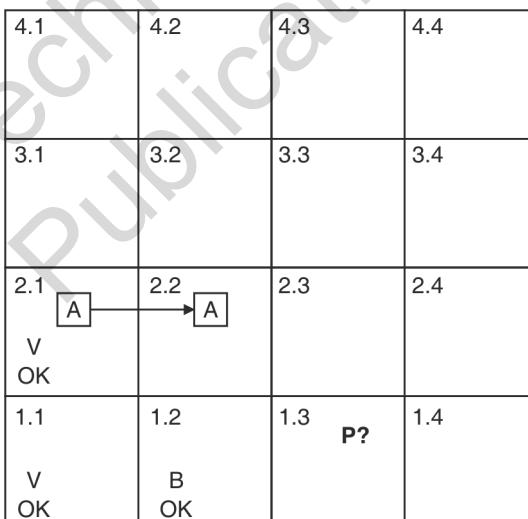


Fig. 3.2.2(d) : WUMPUS world with player moving to room (2,2)

Step 5: Let's move to room (3,2). Here, player receives [stench, breeze, glitter, none, none] percept. See Fig. 3.2.2(e). Field 1 of the percept shows that room (3,1), (3,3) and (4,2) can have WUMPUS. Field 2 of the percept shows that room (3,1), (3,3), (2,2) and (4,2) can have bottomless pit. Field 3 of the percept shows that room (3,2) has gold. So, the player grabs the gold first. As the aim of this game is to grab the gold and go back to the starting position, without being killed by the WUMPUS.

	4.1 P?	4.2 P?	4.3	4.4
3.1 P?	3.2 A OK ↑ A	3.3 P?	3.4	
2.1	2.2 A	2.3	2.4	
1.1 V OK	1.2 B OK	1.3 P?	1.4	

Fig. 3.2.2(e) : WUMPUS world with player moving to room (3,2)

Now, we have to go back to the starting position i.e. room (1,1) without getting killed by WUMPUS. From steps 1, 2, 3 and 4 We know that room (1,1), (1,2), (2,1) and (2,2) are safe rooms. so, we can go back to room (1,1) by following any of the two paths: i.e. (2,2), (2,1), (1,1) or (2,2), (1,2), (1,1).

Step 6: As can be seen in Fig. 3.2.2(f). We will go from room (2,2) to room (2,1) and from room (2,1) to room (1,1). Thus we won the WUMPUS World game!!!

	4.1 W? P?	4.2 W? P?	4.3	4.4
3.1 W? P?	3.2 A OK ↑ A	3.3 W? P?	3.4	
2.1 V OK	2.2 A OK V → A	2.3	2.4	
1.1 V OK	1.2 B OK V	1.3 P?	1.4	

Fig.3.2.2(f) : WUMPUS world with player moving back to room (1,1) with gold

3.3 Logic

- Logic can be called as **reasoning** which is carried out or it is a review based on strict rule of validity to perform a specified task.
- In case of intelligent systems we say that any of logic's particular form cannot bind logical representation and reasoning, they are independent of any particular form of logic.
- Make a note that logic is beneficial only if the knowledge is represented in small extent and when knowledge is represented in large quantity the logic is not considered valuable.
- Fig. 3.3.1 depicts that sentences are physical configurations of an agent, also it shows that sentences need sentence. This means that reasoning is a process of forming new physical configurations from old ones.

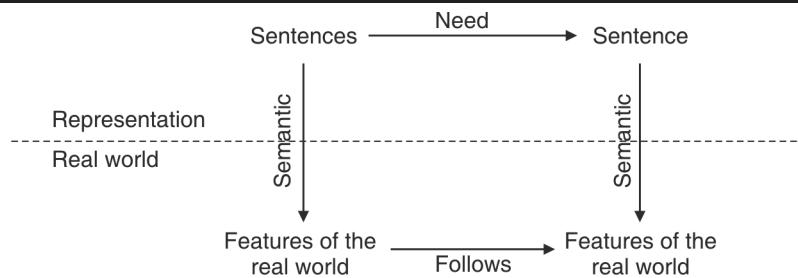


Fig. 3.3.1 : Correspondence between real world and its representation

- Logical reasoning should make sure that the new configurations represent features of the world that actually follow the features of the world that the old configurations represented

3.3.1 Role of Reasoning in AI

- Fig.3.3.2 shows how logic can be seen as a knowledge representation language. There are various levels to the logic and most fundamental type of logic is propositional logic.

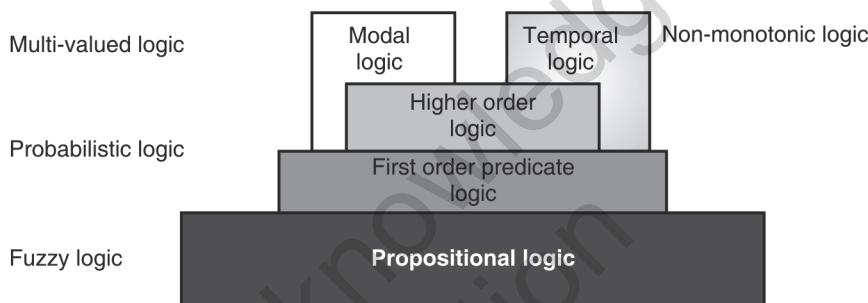


Fig. 3.3.2 : Logic as Knowledge Representation language

- Propositional logic can be considered at fuzzy logic level, where rules are values between range of 0 and 1. Next level is also called as probabilistic logic level using which first order predicate logic is implemented.
- In this Fig. 3.3.2, there are two more levels above higher order logic which are multi-valued and non-monotonic logic levels and they consist of modal logic and temporal logic respectively. All these types of logic are basic building blocks of intelligent systems and they all use reasoning in order to represent sentences. Hence reasoning plays a very important role in AI.

3.4 Representation of Knowledge using Rules

MU – May 13, Dec. 14, May 15

Q. Explain various methods of knowledge representation with example.

(May 13, Dec. 14, May 15, 10 Marks)

- Knowledge can be considered to be represented at generally two levels :
 - Knowledge level :**This level describes the facts.
 - Symbol level :**This level deals with using the symbols for representing the objects, which can be manipulated in programs.
- **Knowledge can be represented using the following rules :**
 - Logical representations
 - Production rule representations
 - Semantic networks
 - Frame representations



(a) Logical representation

- The logical representations are mostly concerned with truth of statements regarding the world. These statements are most generally represented using statements like TRUE or FALSE.
- Logic is successfully used to define ways to infer new sentences from the existing ones. There are certain logics that are used for the representation of information, and range in terms of their expressiveness. There are logic that are more expressive and are more useful in translation of sentences from natural languages into the logical ones. There are several logics that are widely used :

1. **Propositional logic** : These are restricted kinds that make use of propositions (sentences that are either true or false but not both) which can be either true or false. Proposition logic is also known as propositional calculus, sentential calculus or boolean algebra.

All propositions are either true or false, **For example** : .

- (i) Leaves are green
- (ii) Violets are blue.

Sentence	Truth Value	Proposition
Sky is blue	true	yes
Roses are red	true	yes
2+2=5	false	yes

2. **First Order Predicate Logic** : These are much more expressive and make use of variables, constants, predicates, functions and quantifiers along with the connectives explained already in previous section.
3. **Higher Order Predicate Logic** : Higher order predicate logic is distinguished from first order predicate logic by using additional quantifiers and stronger semantics.
4. **Fuzzy Logic** : These indicate the existence of in between TRUE and FALSE or fuzziness in all logics.
5. **Other Logic** : These include multiple valued logic, modal logics and temporal logics.

(b) Production rule representation

One of the widest used methods to represent knowledge is to use production rules, it is also known as IF-THEN rules.

Syntax :

```
IF condition THEN action  
IF premise THEN conclusion  
IF proposition p1 and proposition p2 are TRUE  
THEN proposition p3 is TRUE
```

Example :

IF pressure is high, THEN volume is small.

IF the road is slippery, THEN driving is dangerous.

- Some of the benefits of IF-THEN rules are that they are modular, each defining a relatively small and, at least in principle, independent piece of knowledge. New rules may be added and old ones deleted usually independently of other rules.
- Production rules are simple but powerful forms of representing knowledge, they provide flexibility for combining procedural and declarative representations in a unified manner. The major advantage of production rules are that they are modular, independent of other rules with the provision for addition new rules and deleting older ones.

(c) Semantic networks

- These represent knowledge in the form of graphical networks, since graphs are easy to be stored inside programs as they are concisely represented by nodes and edges.
- A semantic network basically comprises of *nodes* that are named and represent concepts, and labelled *links* representing relations between concepts. Nodes represent both types and tokens.
- For example, the semantic network in Fig. 3.4.1 expresses the knowledge to represent the following data :
 - o Tom is a cat.
 - o Tom caught a fish.
 - o Tom is grey in color.
 - o Tom is owned by Sam.
 - o Tom is a Mammal.
 - o Fish is an Animal.
 - o Cats love Milk.
 - o All mammals are animals.

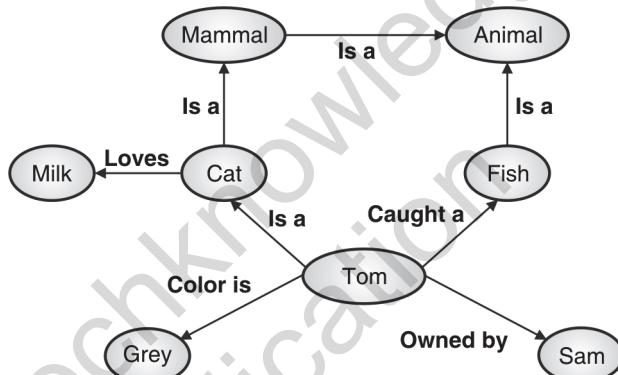


Fig. 3.4.1

- **Conceptual Graph :** It is a recent scheme used for semantic network, introduced by John Sowa, has a finite, connected, bipartite graph. The nodes represent either concepts or conceptual relations. It differs from the previous method that it does not use labelled arcs. **For example:** Ram, Laxman and Bharat are Brothers or cat color is grey can be represented as shown.

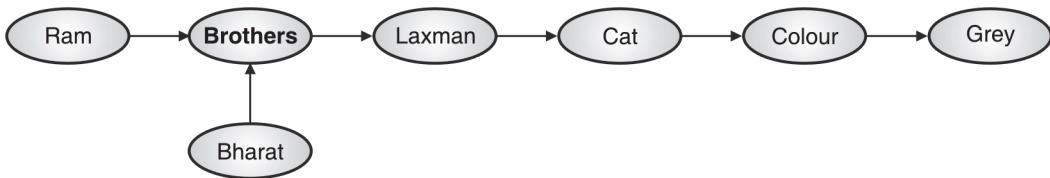


Fig. 3.4.2

(d) Frame representation

- This concept was introduced by Marvin Minsky in 1975. They are mostly used when the task becomes quite complex and needs more structured representation. More structured the system becomes more would be the requirement of using frames which would prove beneficial. Generally frames are record like structures that consists of a collection of slots or attributes and the corresponding slot values.
- Slots can be of any size and type. The slots have names and values (subfields) called as facets. Facets can have names or numbers too. A simple frame is shown in the Fig. 3.4.2 for a person Ram,



- o (Ram)
- o (PROFESSION (VALUE professor))
- o (AGE(VALUE 50))
- o (WIFE(VALUE sita))
- o (CHILDREN(VALUE luv kush))
- o (ADDRESS (STREET(VALUE 4C gb road)))
- o CITY(VALUE banaras))
- o (STATE(VALUE mh))
- o (ZIP(VALUE400615))

3.4.1 Ontology

- Ontology is study about what kind of things or entities exist in the universe. In AI, ontology is the specification of conceptualizations, used to help programs and humans to share knowledge about a particular domain. In turn, ontology is a set of concepts, like entity, relationships among entities, events that are expressed in a uniform way in order to create a vocabulary for information exchange.
- An ontology should also enable a person to verify what a symbol means. That is, given a concept, they want to be able to find the symbol, and, given the symbol, they want to be able to determine what it means.
- Typically, it specifies what types of individuals will be modelled, specifies what properties will be used, and gives some axioms that restrict the use of that vocabulary. Ontologies are usually written independently of a particular application and often involve a community to agree on the meanings of symbols
- For example: Consider a map showing hotels, railway station, buildings, schools, hospitals in a particular locality. In this map the symbols used to indicate these entities are enough to describe them. Hence the community who knows the meaning of these symbols can easily recognize it. Hence that becomes ontology of that map. In this ontology, it may define a building as human-constructed artifacts.
- It may give some restriction on the size of buildings so that shoeboxes cannot be buildings or that cities cannot be buildings. It may also state that a building cannot be at two geographically dispersed locations at the same time.

3.5 Propositional Logic (PL)

- **Propositional Logic(PL)** is simple but powerful for some artificial intelligence problems. You have learnt simple mathematical logic in which uses atomic formulas are used. (Atomic formula is a formula that has no strict sub-formulas). Atomic logic formulas are called propositions.
- In case of artificial intelligence propositional logic is not categorized as the study of truth values, but it is based on relativity of truth values. (i.e. The relationship between the truth value of one statement to that of the truth value of other statement)

3.5.1 Syntax

Basic syntax followed by the propositional logic can be given as follows:

- Propositional symbols are denoted with capital letters like: A, B, C, etc.
- Propositional logic constants have a truth value generally truth values have a crisp nature (i.e. 0 (false) and 1 (true)). But for fuzzy logic truth values can vary in the range of 0 and 1.
- Propositional logic make use of wrapping parenthesis while writing atomic sentence. It is denoted as '(....)'.
- Literal is an atomic sentence or it can be negation of atomic sentence. (A, $\neg A$)



If A is a sentence, then $\neg A$ is a sentence.

- Propositional logic makes use of relationships between propositions and it is denoted by connectives, if A and B are propositions. Connectives used in proposition logic can be seen in the Table 3.5.1.

Table 3.5.1 : Connectives used in Propositional logic

Connective symbol	Name of the Connective symbol	Relationship between Propositional symbols	Name of the Relationship between Propositional symbols
\wedge	And	$A \wedge B$	Conjunction
\vee	Or	$A \vee B$	Disjunction
\neg	Not	$\neg A$	Negation
\Rightarrow	Implies	$A \Rightarrow B$	Implication/ conditional
\Leftrightarrow	is equivalent/ if and only if	$A \Leftrightarrow B$	Biconditional

- To define logical connectives truth tables are used. Truth table 3.5.2 shows five logical connectives.

Table 3.5.2

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$A \Rightarrow B$	$A \Leftrightarrow B$
False	false	false	False	true	true	true
False	true	false	True	true	true	false
True	false	false	True	false	false	false
True	true	true	True	false	true	true

- Take an example, where $A \wedge B$, i.e. Find the value of $A \wedge B$ where A is true and B is false. Third row of the Table 3.5.2 shows this condition, now see third row of the third column where, $A \wedge B$ shows result as false. Similarly other logical connectives can be mapped in the truth table.

3.5.2 Semantics

- World is set of facts which we want to represent to form propositional logic. In order to represent these facts propositional symbols can be used where each propositional symbol's interpretation can be mapped to the real world feature.
- Semantics of a sentence is meaning of a sentence. Semantics determine the interpretation of a sentence. **For example :** You can define semantics of each propositional symbol in following manner:
 1. A means "It is hot"
 2. B means "It is humid", etc.
- Sentence is considered true when its interpretation in the real world is true. Every sentence results from a finite number of usages of the rules. For example, if A and B are sentences then $(A \wedge B)$, $(A \vee B)$, $(B \rightarrow A)$ and $(A \Leftrightarrow B)$ are sentences. The knowledge base is a set of sentences as we have seen in previous section.
- Thus we can say that real world is a model of the knowledge base when the knowledge base is true for that world. In other words a model can be thought of as a truth assignment to the symbols.



- If truth values of all symbols in a sentence are given then it can be evaluated for determining its truth value (i.e. we can say if it is true or false).

3.5.3 What is Propositional Logic ?

- $A \wedge B$ and $B \wedge A$ should have same meaning but in natural language words and sentences may have different meanings. Say for an example,
 1. Radha started feeling feverish and Radha went to the doctor.
 2. Radha went to the doctor and Radha started feeling feverish.
- Here, sentence 1 and sentence 2 have different meanings.
- In artificial intelligence propositional logic is a relationship between the truth value of one statement to that of the truth value of other statement.

3.5.4 PL Sentence - Example

Take example of a weather problem.

- Semantics of each propositional symbol can be defined as follows:
 - Symbol A is a sentence, which means “It is hot”.
 - Symbol B is a sentence, which means “It is humid”.
 - Symbol C is a sentence, which means “It is raining”.
- We can also choose symbols which are easy to understand, like:
 - HT for “It is hot”.
 - HM for “It is humid”.
 - RN for “It is raining”.
- If you have $HM \rightarrow HT$, then that means “If it is humid, then it is hot”.
- If you have $(HT \wedge HM) \rightarrow RN$ then it means “If it is hot and humid, then it is raining” and so on.
- First we have to create the possible mode Is for a knowledge base. To do this we need to consider all the possible assignments of true or false values for Sentence A, B and C. Then verify the truth table for the validity. There can be total 8 possibilities as shown below:

Sentence HT	Sentence HM	Sentence RN	Validity
False	False	False	Valid
False	False	True	Valid
False	True	False	Not Valid
False	True	True	Not Valid
True	False	False	Valid
True	False	True	Valid
True	True	False	Not Valid
True	True	True	Valid

- Now, if the knowledge base is $[HM, HM \rightarrow HT, (HT \wedge HM) \rightarrow RN]$ (i.e. [“It is humid”, “If it is humid, then it is hot”, “If it is hot and humid, then it is raining”]), then “True - True - True” is the only possible valid model.



Tautology and Contradiction

- **Tautology** means valid sentence. It is a sentence which is true for all the interpretations. **For example :** $(A \vee \neg A)$ (“A or not A”) : “It is hot or It is not hot”
- **Contradiction** means an inconsistent sentence. It is a sentence which is false for all the interpretations. **For example :** $A \wedge \neg A$ (“A and not A”) : “It is hot and it is not hot.”
- **X entails Y**, is shown as $X \models Y$. It means that whenever sentence X is True, sentence Y will be True. **For Example :** if, X= Priya is Pooja's Mother's Sister and Y = Priya is Pooja's Aunty. Then $X \models Y$ (X entails Y).

3.5.5 Inference Rules

- New sentences are formed with the logical inference. **For example :** If $A = B$ and $B = C$ then $A = C$. You must have come across this example many times it implies that if knowledge base has “ $A = B$ ” and “ $B = C$ ” then we can infer that “ $A = C$ ”.
- In short inference rule says that new sentence can be created by logically following the set of sentences of the knowledge base.

Table 3.5.3 : Inference Rules

Inference Rules	Premise (KB)	Conclusion
Modus Ponens	$X, X \rightarrow Y$	Y
Substitution	$X \rightarrow Z \ \& \ Y \rightarrow Z$	$X = Y$
Chain rule	$X \rightarrow Y, Y \rightarrow Z$	$X \rightarrow Z$
AND introduction	X, Y	$X \wedge Y$
Transposition	$X \rightarrow Y$	$\neg X \rightarrow \neg Y$

- Entailment is represented as : $KB \models Q$ and Derivation is represented as : $KB \vdash Q$.
- There are two types of inference rules :

1. Sound inference
2. Complete inference

1. Sound inference

- Soundness property of inference says that, if “X is derived from the knowledge base” using given set of protocols of inference, then “X is entailed by knowledge base”. Soundness property can be represented as : “If $KB \vdash X$ then $KB \models X$ ”.
- For **Modus Ponens (MP)** rule we assume that knowledge base has $[A, A \rightarrow B]$, from this we can conclude that knowledge base can have B. See following truth table :

A	B	$A \rightarrow B$	Valid?
TRUE	TRUE	TRUE	Yes
TRUE	FALSE	FALSE	Yes
FALSE	TRUE	TRUE	Yes
FALSE	FALSE	TRUE	Yes



In general,

For atomic sentences p_i , p'_i , and q , where there is a substitution Θ such that

$$\text{SUBST}(\Theta, p_i) = \text{SUBST}(\Theta, p'_i) \quad \text{for all } i,$$

$$\frac{p'_1, p'_2, p'_3, \dots, p'_n, (p_1 \wedge p_2 \wedge p_3 \wedge \dots \wedge p_n \Rightarrow q)}{\text{SUBST}(\Theta, q)}$$

$N + 1$ premises = N atomic sentences + one implication.

Example :

A : It is rainy.

B : I will stay at home.

$A \rightarrow B$: If it is rainy, I will stay at home.

Modus Tollens

When B is known to be false, and if there is a rule "if A, then B," it is valid to conclude that A is also false.

2. Complete inference

- Complete inference is converse of soundness. Completeness property of inference says that, if "X is entailed by knowledge base" then "X can be derived from the knowledge base" using the inference protocols.
- Completeness property can be represented as: "If $KB \models Q$ then $KB \vdash Q$ ".

3.5.6 Horn Clause

- Clauses are generally written as sets of literals. Horn clause is also called as **horn sentence**. In a horn clause a conjunction of 0 or more symbols is to the left of " \rightarrow " and 0 or 1 symbols to the right. See following formula :

$$A_1 \wedge A_2 \wedge A_3 \dots \wedge A_n \rightarrow B_m$$
 where $n \geq 0$ and m is in range{0,1}
- There can be following special cases in horn clause in the above mentioned formula :
 - o For $n=0$ and $m=1$: A (This condition shows that assert A is true)
 - o For $n>0$ and $m=0$: $A \wedge B \rightarrow$ (This constraint shows that both A and B cannot be true)
 - o For $n=0$ and $m=0$: (This condition shows empty clause)
- Conjunctive normal form is a conjunction of clauses and by its set of clauses it is determined up to equivalence. For a horn clause conjunctive normal form can be used where, each sentence is a disjunction of literals with at most one non-negative literal as shown in the following formula : $\neg A_1 \vee \neg A_2 \vee \neg A_3 \dots \vee \neg A_n \vee B$
- This can also be represented as : $(A \rightarrow B) = (\neg A \vee B)$

Significance of horn logic

- Horn sentences can be used in first order logic. Reasoning processes is simpler with horn clauses. Satisfiability of a propositional knowledge base is NP complete. (Satisfiability means the process of finding values for symbols which will make it true).
- For restricting knowledge base to horn sentences, satisfiability is in A. Due to this reason, first order logic horn sentences are the basis for prolog and data log languages.
- Let's take one example which gives entailment for horn formulas.
- Find out if following horn formula is satisfiable?

$$(true \rightarrow X) \wedge (X \wedge Y \rightarrow Z) \wedge (Z \wedge \neg W \rightarrow \text{false}) \wedge (true \rightarrow Y)$$

- From the above equation, we entail if the query atom is false. Equation shows that there are clauses which state that true $\rightarrow X$ and true $\rightarrow Y$, so we can assign X and Y to true value (i.e. true $\rightarrow X \wedge Y$).
- Then we can say that all premises of $X \wedge Y \rightarrow Z$ are true, based on this information we can assign Z to true. After that we can see all premises of $Z \rightarrow W$ are true, so we can assign W to true.
- As now all premises of $Z \wedge W \rightarrow \text{false}$ are true, from this we can entail that the query atom is false. Therefore, the horn formula is not satisfiable.

3.5.7 Propositional Theorem Proving

- Sequence of sentences form a “Proof”. A sentence can be premise or it can be a sentence derived from earlier sentences in the proof based on the inference rule. Whatever we want to prove is called as a query or a goal. Query/goal is the last sentence of the theorem in the proof.
- Take Example of the “weather problem” which we have seen above.
 - HT for “It is hot”.
 - HM for “It is humid”.
 - RN for “It is raining”.

1.	HM	Premise (initial sentence)	“It’s humid”
2.	HM \rightarrow HT	Premise(initial sentence)	“If it’s humid, it’s hot”
3.	HT	Modus ponens(1,2) (sentence derived from 1 and 2)	“It’s hot”
4.	(HT \wedge HM) \rightarrow RN	Premise(initial sentence)	“If it’s hot and humid, it’s raining”
5.	HT \wedge HM	And introduction(1,3)	“It’s hot and humid”
6.	RN	Modusponens(4,5)(sentence derived from 4 and 5)	“It’s raining”

3.5.8 Advantages of Propositional Logic

- Propositional logic is a simple knowledge representation language.
- It is sufficient and efficient technique for solving some artificial intelligence based problems.
- Propositional logic forms the foundation for higher logics like **First Order Logic (FOL)**, etc.
- Propositional logic is NP complete and reasoning is decidable.
- The process of inference can be illustrated by PL.

3.5.9 Disadvantages of Propositional Logic

- Propositional logic is cannot express complex artificial intelligence problems.
- Propositional logic can be impractical for even small worlds, think about WUMPUS hunter problem.
- Even if we try to make use of propositional logic to express complex artificial intelligence problems, it can be very wordy and lengthy.
- PL is a weak knowledge representation language because :
 - With PL it is hard to identify if the used entity is “individual”. For example : If there are entities like : Priya, Mumbai, 123, etc.



- PL cannot directly represent properties of individual entities or relations between individual entities. For example, Pooja is tall.
- PL cannot express specialization, generalizations, or patterns, etc. **For example:** All rectangles have 4 sides.

3.6 First Order Predicate Logic

MU - May 14

Q. Write a short note on predicate logic.

(May 14, 5 Marks)

- Because of the inadequacy of PL discussed above there was a need for more expressive type of logic. Thus **First-Order Logic (FOL)** was developed. FOL is more expressive than PL, it can represent information using relations, variables and quantifiers, e.g., which was not possible with propositional logic.
 - “Gorilla is Black” can be represented as :
 $\text{Gorilla}(x) \rightarrow \text{Black}(x)$
 - “It is Sunday today” can be represented as :
 $\text{today}(\text{Sunday})$
- First Order Logic(FOL)is also called as **First Order Predicate Logic (FOPL)**. Since FOPL is much more expressive as a knowledge representation language than PL it is more commonly used in artificial intelligence.

3.6.1 Syntactic Elements, Semantic and Syntax

- FOL symbol can be a constant term, a variable term or a function.
- Assuming that “X” is a domain of values, we can define a term with following rules :
 1. **Constant term:** It is a term with fixed value which belongs to the domain.
 2. **Variable term:** It is a term, which can be assigned values in the domain.
 3. **Function :** Say “f” is a function of “n” arguments. If we assume that t_1, t_2, \dots, t_n are terms then $f(t_1, t_2, \dots, t_n)$ is also called as a term.
- All the terms are generated by applying the above three protocols.
- First order predicate logic makes use of propositional logic as a base logic, so the connectives used in PL and FOPL are common. Hence, it also supports \wedge conjunction, \vee disjunction, negation, \Rightarrow implication and \Leftrightarrow double implication.
- **Ground Term:** If a term does not have any variables it is called as a **ground term**. A sentence in which all the variables are quantified is called as a “well-formed formula”.
 - Every ground term is mapped with an object.
 - Every condition (predicate) is mapped to a relation.
 - A ground atom is considered as true if the predicate’s relation holds between the terms’ objects.
 - **Rules in FOL:**In predicate logic rule has two parts predecessor and successor. If the predecessor is evaluated to TRUE successor will be true. It uses the implication \rightarrow symbol. Rule represents If-then types of sentences.
- Example: The sentence “If the bag is of blue colour, I will buy it.” Will be represented as colour (bag, blue) \rightarrow buy(bag).

Quantifiers

Apart from these connectives FOPL makes use of quantifiers. As the name suggests they quantify the number of variables taking part in the relation or obeying the rule.



1. Universal Quantifier ‘ \forall ’

- Pronounced as “for all” and it is applicable to all the variables in the predicate
- “ $\forall x A$ ” means A is true for every replacement of x.
- Example: “Every Gorilla is Black” can be represented as :
“ $\forall x (\text{Gorilla}(x) \rightarrow \text{Black}(x))$ ”

2. Existential Quantifier ‘ \exists ’

- Pronounced as “there exists”
- “ $\exists x A$ ” means A is true for at least one replacement of x.
- Example: “There is a white dog” can be represented as,
 $\exists x (\text{Dog}(X) \wedge \text{white}(X))$

Note :

1. Typically, \Rightarrow is the main connective with \forall
Example: “Everyone at MU is smart” is represented as
 $\forall x \text{At}(x, \text{MU}) \rightarrow \text{smart}(x)$
2. Typically, \wedge is the main connective with \exists
Example: Someone killed the cat and is guilty.
 $\exists x \text{killed}(x, \text{cat}) \wedge \text{guilty}(x)$
 - **Equality** : term₁ = term₂ is true under a given interpretation if and only if term₁ and term₂ refer to the same object
 - **Example** : Richard has at least two brothers
 $\exists x \exists y \text{Brother}(x, \text{Richard}) \wedge \text{Brother}(y, \text{Richard}) \wedge \neg(x = y)$

3.7 Comparison between Propositional Logic and First Order Logic

Sr. No.	Propositional logic (PL)	Predicate logic (FOL)
1.	PL cannot represent small worlds like vacuum cleaner world.	FOL can very well represent small worlds' problems.
2.	PL is a weak knowledge representation language	FOL is a strong way of representing language.
3.	Propositional Language uses propositions in which the complete sentence is denoted by a symbol.	FOL uses predicates which involve constants, variables, functions, relations.
4.	PL cannot directly represent properties of individual entities or relations between individual entities. e.g. Meera is short.	FOL can directly represent properties of individual entities or relations between individual entities using individual predicates using functions. E.g. Short(Meera)
5.	PL cannot express specialization, generalizations, or patterns, etc. e.g. All rectangles have 4 sides.	FOL can express specialization, generalizations, or patterns, etc. Using relations. E.g. no_of_sides(rectangle, 4)
6.	PL is a foundation level logic.	FOL is a higher level logic.
7.	PL is not sufficiently expressive to represent complex statements.	FOL can represent complex statements.

Sr. No.	Propositional logic (PL)	Predicate logic (FOL)
8.	PL assumes the world contains facts	FOL assumes the world contains objects, relations, functions like natural language.
9.	In PL Meaning of the facts is context-independent unlike natural language.	In FOL Meaning of the sentences is context dependent like natural language.
10.	PL is declarative in nature.	FOL is derivative in nature.

3.8 Inference in FOL

3.8.1 Forward Chaining

- For any type of inference there should be a path from start to goal. When based on the available data a decision is taken, then the process is called as the forward chaining. Forward chaining or data-driven inference works from an initial state, and by looking at the premises of the rules (IF-part), perform the actions (THEN-part), possibly updating the knowledge base or working memory. This continues until no more rules can be applied or some cycle limit is met.
- For example, “If it is raining then, we will take umbrella”. Here, “it is raining” is the data and “we will take umbrella” is a decision. This means it was already known that it’s raining that’s why it was decided to take umbrella. This process is **forward chaining**.

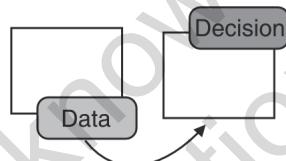


Fig. 3.8.1 : Forward chaining

- “Forward chaining” is called as a data-driven inference technique.

Example

Given :

- Rule : $\text{human}(A) \rightarrow \text{mortal}(A)$
- Data : $\text{human}(\text{Mandela})$
- To prove : $\text{mortal}(\text{Mandela})$

Forward Chaining Solution

- Human (Mandela) matches Left Hand Side of the Rule. So, we can get $A = \text{Mandela}$
- based on the rule statement we can get : $\text{mortal}(\text{Mandela})$
- Forward chaining is used by the “design expert systems”, as it performs operation in a forward direction (i.e. from start to the end).

Example

- Consider following example. Let us understand how the same example can be solved using both forward.
- Given facts are as follows:
 - It is a crime for an American to sell weapons to the enemy of America.
 - Country Nono is an enemy of America.
 - Nono has some missiles.

4. All the missiles were sold to Nono by Colonel West.
 5. Missile is a weapon.
 6. Colonel West is American.
- We have to prove that West is a criminal.
- Let's see how to represent these facts by FOL.
1. It is a crime for an American to sell weapons to the enemy nations.
 $\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{sell}(x, y, z) \wedge \text{enemy}(z, \text{America}) \Rightarrow \text{Criminal}(x)$
 2. Country Nono is an enemy of America.
 $\text{Enemy}(\text{Nono}, \text{America})$
 3. Nono has some missiles.
 - o Owns (Nono, x)
 - o Missile(x)
 4. All the missiles were sold to Nono by Colonel West.
 $\text{Missile}(x) \wedge \text{owns}(\text{Nono}, x) \Rightarrow \text{Sell}(\text{West}, x, \text{Nono})$
 5. Missile is a weapon.
 $\text{Missile}(x) \Rightarrow \text{weapon}(x)$
 6. Colonel West is American.
 $\text{American}(\text{West})$

Proof by forward chaining

The proof will start from the given facts. And as we can derive other facts from those, it will lead us to the solution. Please refer to Fig. 3.8.2 As we observe from the given facts we can reach to the predicate Criminal (West).

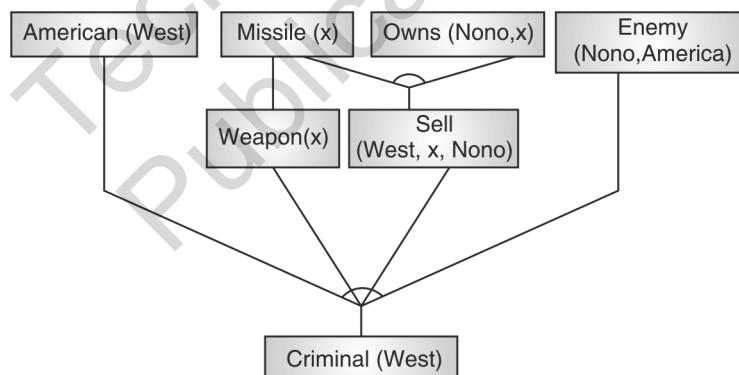


Fig. 3.8.2 : Proof by forward chaining

3.8.2 Backward Chaining

MU – Dec. 12, May 14

Q. Describe backward chaining algorithm with an example.	(Dec. 12, 10 Marks)
Q. Explain backward chaining giving suitable example.	(May 14, 10 Marks)

- If based on the decision the initial data is fetched, then it is called as **backward chaining**. Backward chaining or goal-driven inference works towards a final state, and by looking at the working memory to see if goal already there. If not look at the actions (THEN-parts) of rules that will establish goal, and set up sub-goals for achieving premises of the rules (IF-part). This continues until some rule can be applied, apply to achieve goal state.

- For example, If while going out one has taken umbrella. Then based on this decision it can be guessed that it is raining. Here, “taking umbrella” is a decision based on which the data is generated that “it's raining”. This process is **backward chaining**. “Backward chaining” is called as a decision-driven or goal-driven inference technique.

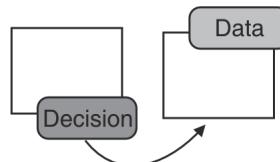


Fig. 3.8.3 : Backward chaining

- Given :
 - Rule : $\text{human}(A) \rightarrow \text{mortal}(A)$
 - Data : $\text{human}(\text{Mandela})$
- To prove : $\text{mortal}(\text{Mandela})$

Backward Chaining Solution

- $\text{mortal}(\text{Mandela})$ will be matched with $\text{mortal}(A)$ which gives $\text{human}(A)$ i.e. $\text{human}(\text{Mandela})$ which is also a given fact. Hence proved.
- It makes use of right hand side matching. backward chaining is used by the “diagnostic expert systems”, because it performs operations in a backward direction (i.e. from end to start).

Example

Let us understand how the same example used in forward chaining can be solved using backward chaining.

Proof by backward Chaining

The proof will start from the fact to be proved. And as we can map it with given facts, it will lead us to the solution. Please refer to Fig. 3.8.4. As we observe, all leaf nodes of the proof are given facts that means “West is Criminal”.

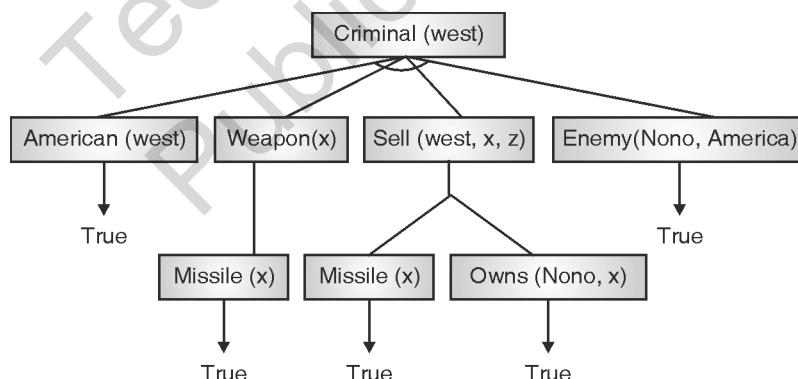


Fig. 3.8.4 : Proof by backward chaining

3.8.3 Differentiate between Forward Chaining and Backward Chaining

Attribute	Backward Chaining	Forward Chaining
Also known as	Goal-driven	Data-driven
Starts from	Possible conclusion	New data
Processing	Efficient	Somewhat wasteful
Aims for	Necessary data	Any Conclusion(s)

Attribute	Backward Chaining	Forward Chaining
Approach	Conservative/Cautious	Opportunistic
Practical if	Number of possible final answers is reasonable or a set of known alternatives is available.	Combinatorial explosion creates an infinite number of possible right answers.
Appropriate for	Diagnostic, prescription and debugging application	Planning, monitoring, control and interpretation application
Reasoning	Top-down reasoning	Bottom-up reasoning
Type of Search	Depth-first search	Breadth-first search
Who determine search	Consequents determine search	Antecedents determine search
Flow	Consequent to antecedent	Antecedent to consequent

Ex. 3.8.1 : Using predicate logic find the course of Anish's liking for the following :

- (i) Anish only likes easy courses.
- (ii) Computer courses are hard.
- (iii) All electronics courses are easy
- (iv) DSP is an electronics course.

Soln. :

Step 1 : Converting given facts to FOL

- (i) $\forall x : \text{course}(x) \wedge \text{easy}(x) \rightarrow \text{likes}(\text{Anish}, x)$
- (ii) $\forall x : \text{course}(x) \wedge \text{computes}(x) \rightarrow \text{hard}(x)$
- (iii) $\forall x : \text{course}(x) \wedge \text{electronics}(x) \rightarrow \text{easy}(x)$
- (iv) Electronics(DSP)
- (v) course(DSP)

Step 2 : Proof by backward chaining :

As we have to find out which course Anish likes. So we will start the proof from the same fact.

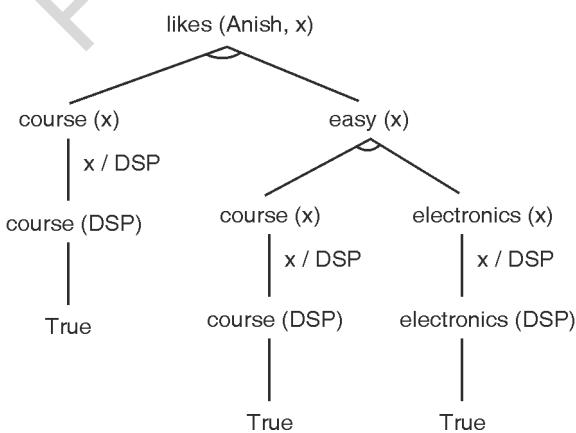


Fig. P. 3.8.1

Hence proved that Anish likes DSP course.



3.9 Knowledge Engineering in First Order Logic

Knowledge engineering is a process of knowledgebase construction. It requires a knowledge engineer to investigate a particular domain, learn the important concepts in that domain, and create a formal representation and logical relations among objects in that domain.

3.9.1 Knowledge Engineering Process

Following is the general knowledge engineering process which can be applied to problem of any domain.

1. **Identify the task:** This step is analogues to PEAS process while designing an agent. While identifying task, the knowledge engineer must define the scope of knowledgebase and the range of questions that can be answered through the database. He also need to specify the type of facts that will be available for each specific problem instance.
2. **Assemble the relevant knowledge:** Assembling the relevant knowledge of that particular domain is called the process of knowledge acquisition. In this the knowledge engineer needs to extract the domain knowledge either by himself provided he is the domain expert or needs to work with the real experts of the domain. In this process knowledge engineer learns how the domain actually works and can determine the scope of the knowledgebase as per the identified tasks.
3. **Defining vocabulary:** Defining a complete vocabulary including predicates, functions and constants is a very important step of knowledge engineering. This process transforms the domain level concepts to logic level symbols. It should be exhaustive and precise. This vocabulary is called as ontology of the domain. And once the ontology is defined, it means, the existence of the domain is defined. That is, what kind of things exist in the domain is been decided.
4. **Encoding of general knowledge about the domain:** In this step the knowledge engineer defines axioms for all the vocabulary terms by define meaning of each term. This enables expert to cross check the vocabulary and the contents. If he finds any misinterpretations or gaps, it can be fixed at this point by redoing step 3.
5. **Encode the problem:** In this step, the specific problem instance is encoded using the defined ontology. This step will be very easy if the ontology is defined properly. Encoding means writing atomic sentences about problem instances which are already part of ontology. It can be analogues to input data for a computer program.
6. **Query the Knowledgebase:** Once all the above steps are done, all input for the system is set and now is a time to generate some output from the system. So, in order to get some interested facts inferred from the provided knowledge, we can query the knowledgebase. The inference procedure will operate on the axioms and facts to derive the new inferences. This lessens the task of a programmer to write application specific programs.
7. **Debug the knowledgebase:** This is the step in which one can prove or check the toughness of the knowledgebase. In the sense, if the inference procedure is able to give appropriate answers to all the queries asked or it stops in between because of the incomplete axioms; will be easily identified by debugging process. If one observes the reasoning chain stopping in between or some of the queries could not be answered then, it is an indication of a missing or a weak axioms. Then the corrective measures can be taken by repeating the required steps and system can be claimed to have a complete and precise knowledgebase.

3.10 Unification and Lifting

3.10.1 Unification

The processes of finding legal substitutions that make different logical expressions look identical. The unification algorithm is a recursive algorithm; the problem of **unification** is: given two atoms, to find if they unify, and, if they do, return an MGU (Most General Unifier) of them.

**Procedure** Unify(t_1, t_2)**Inputs** t_1, t_2 : atoms **Output**most general unifier of t_1 and t_2 if it exists or \perp otherwise**Local** E : a set of equality statements S : substitution $E \leftarrow \{t_1 = t_2\}$ $S = \{\}$ **while** ($E \neq \{\}$)select and remove $x = y$ from E **if** (y is not identical to x) **then****if** (x is a variable) **then**replace x with y everywhere in E and S $S \leftarrow \{x/y\} \cup S$ **else if** (y is a variable) **then**replace y with x everywhere in E and S $S \leftarrow \{y/x\} \cup S$ **else if** (x is $f(x_1, \dots, x_n)$ and y is $f(y_1, \dots, y_n)$) **then** $E \leftarrow E \cup \{x_1 = y_1, \dots, x_n = y_n\}$ **else****return** \perp **return** S

- Unification algorithm for Data log

Example: “ $x \text{ King}(x) \wedge \text{Brave}(x) \Rightarrow \text{Noble}(x)$ ”

King(Ram)

Brave(Ram)

- We get an ‘ x ’ where, ‘ x ’ is a king and ‘ x ’ is brave (Then x is noble) then ideally what we want is Θ = {substitution set}

i.e. $\Theta = \{x/ \text{Ram}\}$ Hence, Ram Unifies x .

3.10.2 Lifting

The process of encapsulating inference rule is called as **Generalized Modus Ponens**.

Generalized Modus Ponens

- For atomic sentences p_i, p'_i , and q , where there is a substitution Θ such that

$$\text{SUBST } (\Theta, p_i) = \text{SUBST } (\Theta, p'_i) \quad \text{for all } i,$$

$$\frac{p'_1, p'_2, p'_3, \dots, p'_{n'} (p_1 \wedge p_2 \wedge p_3 \wedge \dots \dots \wedge p_n \Rightarrow q)}{\text{SUBST } (\Theta, q)}$$

- $N + 1$ premises = N atomic sentences + one implication.

- Applying SUBST(θ , q) produces the conclusion we seek.

$$p'_1 = \text{King}(\text{Ram})$$

$$p'_2 = \text{Brave}(y)$$



$$p_1 = \text{King}(x) \quad p_2 = \text{Brave}(x)$$

$$\Theta = \{x / \text{Ram}, y / \text{Ram}\} \quad q = \text{Noble}(x)$$

SUBST(Θ, q) is $\text{Noble}(\text{Ram})$

- Generalized Modus Ponens is a **lifted** version of Modus Ponens. It raises Modus Ponens from ground (variable-free) propositional logic to first-order logic. Hence it is called as **lifting**.
- Here “lifted” indicates transformed from.
- The major advantage of lifted inference rules over propositional logic is that only those substitutions are made that are required so as particular inferences are allowed to proceed.

Ex. 3.10.1 : Represent following sentences in FOL using a consistence vocabulary.

- (i) Every person who buys a policy is smart.
- (ii) No person buys an expensive policy.
- (iii) There is an agent who sells policies only to people who are not insured.
- (iv) There is a barber who shaves all men in town who do not save themselves.

Soln. :

- (i) $\forall x \forall y : \text{person}(x) \wedge \text{policy}(y) \wedge \text{buys}(x, y) \rightarrow \text{smart}(x)$
- (ii) $\forall x, \forall y : \text{person}(x) \wedge \text{policy}(y) \wedge \text{expensive}(y) \rightarrow \neg \text{buys}(x, y)$
- (iii) $\forall x : \text{person}(x) \wedge \neg \text{insured}(x)$
 $\forall y : \exists x \wedge \text{policy}(y) \wedge \neg \text{agent}(x) \rightarrow \text{sells}(x, y, x)$
- (iv) $\exists x \forall y : \text{barber}(x) \wedge \text{person}(y) \wedge \neg \text{shaves}(y, y) \rightarrow \text{shaves}(x, y)$

Ex. 3.10.2 : Represent following statements in FOPL.

1. Anyone who kills an animal is loved by no one.
2. A square is breezy if there is a pit in the neighbouring squares.

Soln. :

- (i) $\forall x \forall y : \text{kills}(x, \text{animal}) \rightarrow \neg \text{loves}(y, x)$
- (ii) $\forall x \forall y : \text{pit}(x, y) \rightarrow \text{breez}(x, y - 1) \wedge \text{breez}(x, y + 1) \wedge \text{breez}(x - 1, y) \wedge \text{breez}(x + 1, y)$

Ex. 3.10.3 : Write first order logic statements for following statements :

- (i) If a perfect square is divisible by a prime p then it is also divisible by square of p.
- (ii) Every perfect square is divisible by some prime.
- (iii) Alice does not likechemistry and history.
- (iv) If it is Saturday and warm, then sam is in the park.
- (v) Anything anyone eats and is not killed by is food.

Soln. :

- (i) $\forall x : \text{square}(x) \wedge \text{prime}(y) \wedge \text{divides}(p, x) \rightarrow [\exists z : \text{square_of}(z, p) \wedge \text{divides}(z, x)]$
- (ii) $\forall x \exists y : \text{square}(x) \wedge \text{divides}(p, x)$
- (iii) $\neg \text{likes}(\text{Alice}, \text{History}) \wedge \neg \text{likes}(\text{Alice}, \text{Chemistry})$
- (iv) $\text{day}(\text{Saturday}) \wedge \text{weather}(\text{warm}) \rightarrow \text{in_park}(\text{Sam})$
- (v) $\forall x : \forall y : \text{person}(x) \wedge \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$



3.11 Resolution

- Resolution is a valid inference rule. Resolution produces a new clause which is implied by two clauses containing complementary literals. This resolution rule was discovered by Alan Robinson in the mid 1960's.
- We have seen that a literal is an atomic symbol or a negation of the atomic symbol (i.e. $A, \neg A$).
- Resolution is the only inference rule you need, in order to build a sound (soundness means that every sentence produced by a procedure will be "true") and complete (completeness means every "true" sentence can be produced by a procedure) theorem proof maker.
- Take an example where we are given that :
 - o A clause X containing the literal : Z
 - o A clause Y containing the literal : $\neg Z$
- Based on resolution and the information given above we can conclude :
 $(X - \{Z\}) \cup (Y - \{\neg Z\})$
- Take a generalized version of the above problem :

Given:

- A clause X containing the literal: Z
 - A clause Y containing the literal: $\neg Y$
 - A most general unifier G of Z and $\neg Y$
- We can conclude that : $((X - \{Z\}) \cup (Y - \{\neg Y\})) \mid G$

3.11.1 The Resolution Procedure

- Let knowledge base be a set of true sentences which do not have any contradictions, and Z be a sentence that we want to prove.
- The Idea is based on the proof by negation. So, we should assume $\neg Z$ and then try to find a contradiction (You must have followed such methods while solving geometry proofs). Then based on the Intuition that, if all the knowledge base sentences are true, and assuming $\neg Z$ creates a contradiction then Z must be inferred from knowledge base. Then we need to convert knowledge base $\cup \{\neg Z\}$ to clause form.
- If there is a contradiction in knowledge base, that means Z is proved. Terminate the process after that.
- Otherwise select two clauses and add their resolvents to the current knowledge base. If we do not find any resolvable clauses then the procedure fails and then we terminate. Else, we have to start finding if there is a contradiction in knowledge base, and so on.

3.11.2 Conversion from FOL Clausal Normal Form (CNF)

MU - May 16

Q. Explain the steps involved in converting the propositional logic statement into CNF with a suitable example.

(May 16, 10 Marks)

1. Elimination of implication i.e. Eliminate all ' \rightarrow ' : Replace $P \rightarrow Q$ with $\neg P \vee Q$
2. Distribute negations: Replace $\neg\neg P$ with P , $\neg(P \vee Q)$ with $\neg P \wedge \neg Q$ and so on.



3. Eliminate existential quantifiers by replacing with Skolem constants or Skolemfunctions:

e.g. $\forall X \exists Y (P_1(X,Y) \vee (P_2(X,Y)) \equiv \forall X (P_1(X,f(X)) \vee (P_2(X,f(X)))$

4. Rename variables to avoid duplicate quantifiers.
5. Drop all universal quantifiers
6. Place expression into Cconjunctive NormalForm.
7. Convert to clauses i.e. separates all conjunctions as separate clause.
8. Rename variables to avoid duplicate clauses.

Ex. 3.11.1 : Convert following propositional logic statement into CNF.

$A \rightarrow (B \rightarrow C)$

MU - Dec. 15, 4 Marks

Soln. :

FOL : $A \rightarrow (B \leftrightarrow C)$

Normalizing the given statement.

- (i) $A \rightarrow (B \rightarrow C \wedge C \rightarrow B)$
- (ii) $(A \rightarrow (B \rightarrow C)) \wedge (A \rightarrow (C \rightarrow B))$

Converting to CNF.

Applying Rule, $\alpha \rightarrow \beta \equiv \neg \alpha \vee \beta$

$$\neg A \vee (\neg B \vee C) \wedge \neg A \vee (\neg C \vee B)$$

$$\text{i.e. } \neg A \vee ((\neg B \vee C) \wedge (\neg C \vee B))$$

3.11.3 Facts Representation

- To show how facts can be represented let's take a simple problem:
 - o "Heads X wins, Tails Y loses."
 - o Our goal is to show that X always wins with the help of resolution.
- Solution can be given as follows:

1. $H \Rightarrow \text{Win}(X)$
2. $T \Rightarrow \text{Loose}(Y)$
3. $\neg H \Rightarrow T$
4. $\text{Loose}(Y) \Rightarrow \text{Win}(X)$

Thus we have : $\text{Win}(X)$

- We can write a proof for this problem as follows:

1. $\{\neg H, \text{Win}(X)\}$
2. $\{\neg T, \text{Loose}(Y)\}$
3. $\{H, T\}$
4. $\{\neg \text{Loose}(Y), \text{Win}(X)\}$



5. $\{\neg \text{Win}(X)\}$
6. $\{\neg T, \text{Win}(X)\}$ (From 2 and 4)
7. $\{T, \text{Win}(X)\}$ (From 1 and 3)
8. $\{\text{Win}(X)\}$ (From 6 and 7)
9. $\{\}$ (From 5 and 8)

3.11.4 Example

Let's take the same example of forward and backward chaining to learn how to write proofs for resolution.

Step 1 :

The given facts are :

1. It is a crime for an American to sell weapons to the enemy nations.

$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{sell}(x, y, z) \wedge \text{enemy}(z, \text{America}) \Rightarrow \text{Criminal}(x)$

2. Country Nono is an enemy of America.

$\text{Enemy}(\text{Nono}, \text{America})$

3. Nono has some missiles.

- o $\text{Owns}(\text{Nono}, x)$
- o $\text{Missile}(x)$

4. All the missiles were sold to Nono by Colonel West.

$\text{Missile}(x) \wedge \text{owns}(\text{Nono}, x) \Rightarrow \text{Sell}(\text{West}, x, \text{Nono})$

5. Missile is a weapon.

$\text{Missile}(x) \Rightarrow \text{weapon}(x)$

6. Colonel West is American.

$\text{American}(\text{West})$

Step 2 :

Lets convert them to CNF

1. $\sim \text{American}(x) \vee \sim \text{Weapon}(y) \vee \sim \text{sell}(x, y, z) \vee \sim \text{enemy}(z, \text{America})$

$\vee \text{Criminal}(x)$

2. $\text{Enemy}(\text{Nono}, \text{America})$

3. $\text{Owns}(\text{Nono}, x)$

4. $\text{Missile}(x)$

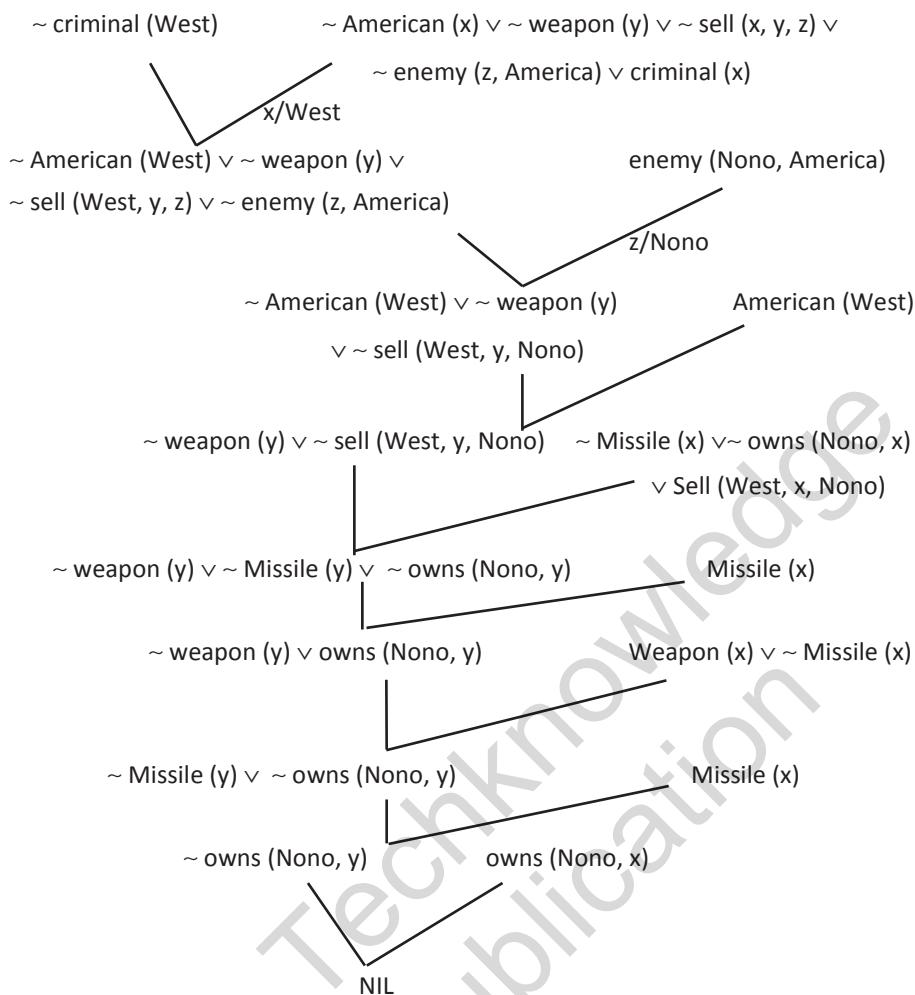
5. $\sim \text{Missile}(x) \vee \sim \text{owns}(\text{Nono}, x) \vee \text{Sell}(\text{West}, x, \text{Nono})$

6. $\sim \text{Missile}(x) \vee \text{weapon}(x)$

7. $\text{American}(\text{West})$

Step 3 :

To prove that West is criminal using resolution.



Hence our assumption was wrong. Hence proved that West is criminal.

Ex. 3.11.2 : Consider following statements :

- (a) Ravi Likes all kind of food.
 - (b) Apple and Chicken are food
 - (c) Anything anyone eats and is not killed is food.
 - (d) Ajay eats peanuts and still alive.
 - (e) Rita eats everything that Ajay eats.

Prove that Ravi Likes Peanuts using resolution. What food does Rita eat?

Soln. :

(A) Proof by Resolution

Step 1 : Negate the statement to be proved.

~likes (Ravi, Peanuts)

Step 2 : Convert given facts to FOL

- (a) $\forall x, \text{food}(x) \rightarrow \text{likes}(\text{Ravi}, x)$
 - (b) $\text{food}(\text{Apple})$

- (c) food (Chicken)
- (d) $\forall x \forall y : \text{eats}(x, y) \wedge \sim \text{killed}(x) \rightarrow \text{food}(y)$
- (e) eats (Ajay, Peanuts) \wedge alive (Ajay)
- (f) $\forall x : \text{eats}(\text{Ajay}, x) \rightarrow \text{eats}(\text{Rita}, x)$

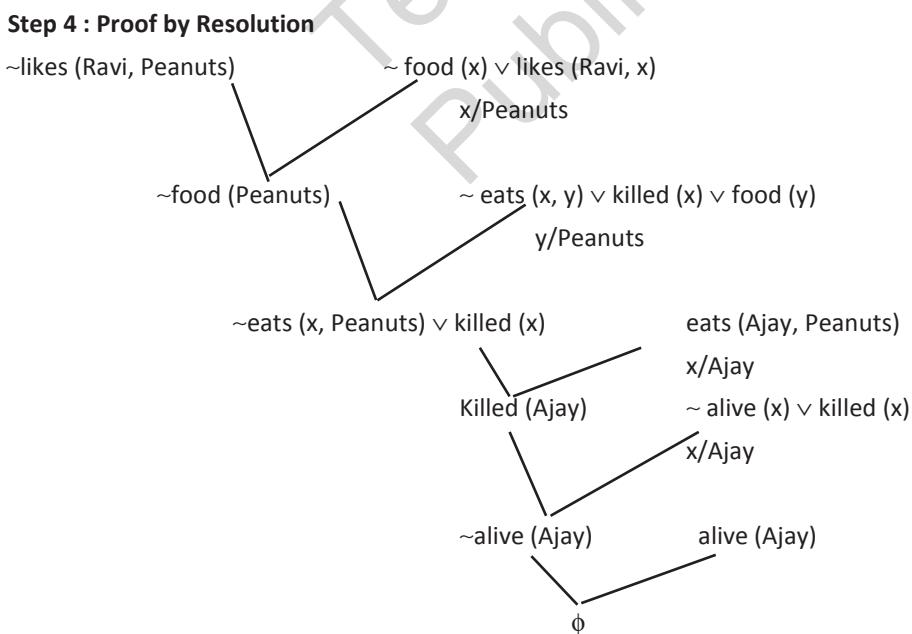
In this case we have to add few common sense predicate which are always true.

- (g) $\forall x : \sim \text{killed}(x) \rightarrow \text{alive}(x)$
- (h) $\forall x : \text{alive}(x) \rightarrow \sim \text{killed}(x)$

Step 3 : Converting FOLs to CNF

- (a) $\sim \text{food}(x) \vee \text{likes}(\text{Ravi}, x)$
- (b) $\text{food}(\text{Apple})$
- (c) $\text{food}(\text{Chicken})$
- (d) $\sim \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$
- (e) $\text{eats}(\text{Ajay}, \text{Peanuts})$
- (f) $\text{alive}(\text{Ajay})$
- (g) $\sim \text{eats}(\text{Ajay}, x) \vee \text{eats}(\text{Rita}, x)$
- (h) $\text{killed}(x) \vee \text{alive}(x)$

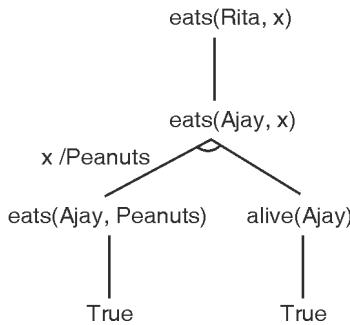
Step 4 : Proof by Resolution



As the result of this resolution is NIL, it means our assumption is wrong. Hence proved that "Ravi likes Peanuts".

To answer : What food Rita eats ?

(B) Proof by backward chaining : (Referring to FOLs of Step 2)



Hence the answer is Rita eats peanuts.

Ex. 3.11.3 : Using a predicate logic convert the following sentences to predicates and prove that the statement "Ram did not jump" is false.

- (a) Ram went to temple.
- (b) The way to temple is, walk till post box and take left or right road.
- (c) The left road has a ditch.
- (d) Way to cross the ditch is to jump
- (e) A log is across the right road.
- (f) One needs to jump across the log to go ahead.

Soln.:

Step 1 : Negate the statement to be proved.

$\sim \text{jump}(\text{Ram})$

Step 2 : Converting given statement to FOL

- (a) $\text{At}(\text{Ram}, \text{temple})$
- (b₁) $\forall x : \text{At}(x, \text{temple}) \rightarrow \text{At}(x, \text{PostBox}) \wedge \text{take left}(x)$
- (b₂) $\forall x : \text{At}(x, \text{temple}) \rightarrow \text{At}(x, \text{PostBox}) \wedge \text{take right}(x)$
- (c) $\forall x : \text{take left}(x) \rightarrow \text{cross}(x, \text{ditch})$
- (d) $\forall x : \text{cross}(x, \text{ditch}) \rightarrow \text{jump}(x)$
- (e) $\forall x : \text{take right}(x) \rightarrow \text{at}(x, \text{log})$
- (f) $\forall x : \text{at}(x, \text{log}) \rightarrow \text{jump}(x)$

Step 3 : Converting FOLs to CNF

- (a) $\text{At}(\text{Ram}, \text{temple})$
- (b₁₁) $\sim \text{At}(x, \text{temple}) \vee \text{At}(x, \text{PostBox})$
- (b₁₂) $\sim \text{At}(x, \text{temple}) \vee \text{take left}(x)$
- (b₂₁) $\sim \text{At}(x, \text{temple}) \vee \text{At}(x, \text{PostBox})$
- (b₂₂) $\sim \text{At}(x, \text{temple}) \vee \text{take right}(x)$

(c) $\sim \text{take left}(x) \vee \text{cross}(x, \text{ditch})$

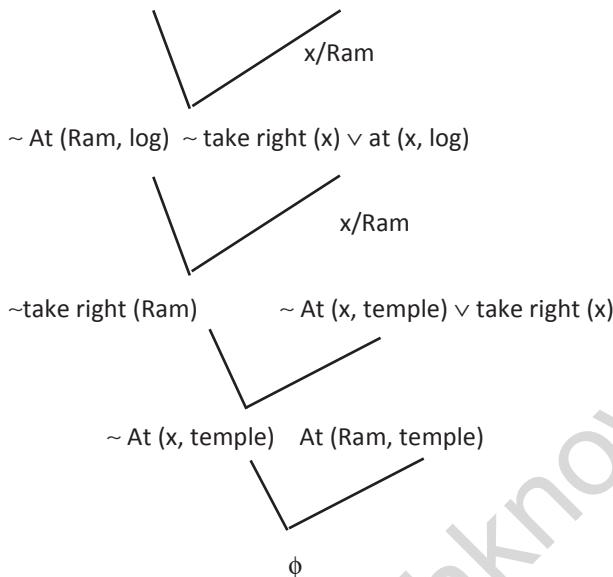
(d) $\sim \text{cross}(x, \text{ditch}) \vee \text{jump}(x)$

(e) $\sim \text{take right}(x) \vee \text{at}(x, \text{log})$

(f) $\sim \text{at}(x, \text{log}) \vee \text{jump}(x)$

Step 4 : Proof by Resolution

$\sim \text{jump}(\text{Ram}) \quad \text{at}(x, \text{log}) \vee \text{jump}(x)$



Hence proved.

Ex. 3.11.4: Consider following statements.

1. Rimi is hungry.
2. If Rimi is hungry she barks.
3. If Rimi is barking then Raja is angry.

Explain statements in predicate logic. Convert them into CNF form.

Prove that Raja is angry using resolution.

Soln. :

Step 1 : Converting given facts to FOL.

1. $\text{Hungry}(\text{Rimi})$
2. $\text{Hungry}(\text{Rimi}) \rightarrow \text{barks}(\text{Rimi})$
3. $\text{Barks}(\text{Rimi}) \rightarrow \text{angry}(\text{Raja})$

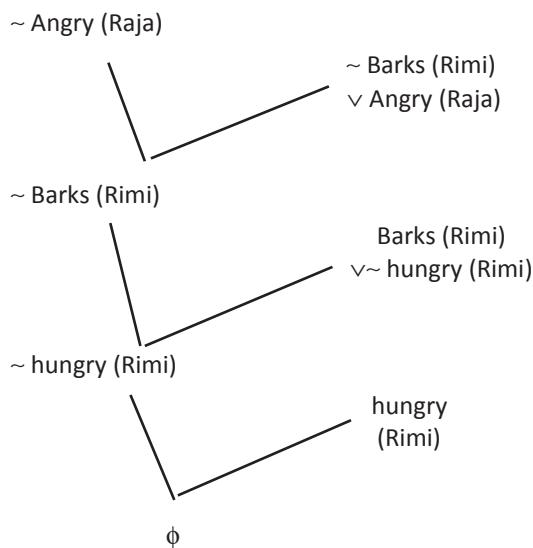
Step 2 : Converting FOL statements to CNF.

1. $\text{Hungry}(\text{Rimi})$
2. $\sim \text{hungry}(\text{Rimi}) \vee \text{barks}(\text{Rimi})$
3. $\sim \text{barks}(\text{Rimi}) \vee \text{angry}(\text{Raja})$

Step 3 : Negate the stmt to be proved

T.P.T. $\text{Angry}(\text{Raja})$

Negation : $\sim \text{Angry}(\text{Raja})$

Step 4 : Proof by resolution

This shows that our assumption is Wrong. Hence proved that **Raja is Angry**.

Ex. 3.11.5 : Consider following facts.

1. If maid stole the jewellery then butler was not guilty.
2. Either maid stole the jewellery or she milked the cow.
3. If maid milked the cow then butler got the cream.
4. Therefor if butler was guilty then he got the cream.

Prove the conclusion (step 4) is valid using resolution.

Soln. :

Step 1 : Converting given facts to FOL.

1. $\text{steal}(\text{maid}, \text{jwellary}) \rightarrow \sim \text{guilty}(\text{butler})$
2. $\text{steal}(\text{maid}, \text{jwellary}) \vee \text{milk}(\text{maid}, \text{cow})$
3. $\text{milk}(\text{maid}, \text{cow}) \rightarrow \text{got_cream}(\text{butler})$

To prove that

4. $\text{guilty}(\text{butler}) \rightarrow \text{got_cream}(\text{butler})$

Step 2 : Converting FOL to CNF.

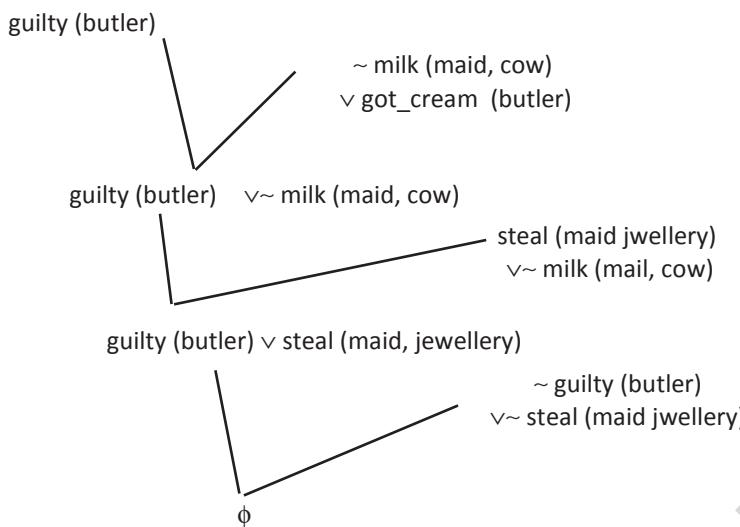
1. $\sim \text{steal}(\text{maid}, \text{jwellary}) \vee \sim \text{guilty}(\text{butler})$
2. $\text{steal}(\text{maid}, \text{jwellary}) \vee \text{milk}(\text{maid}, \text{cow})$
3. $\sim \text{milk}(\text{maid}, \text{cow}) \vee \text{got_cream}(\text{butler})$
4. $\sim \text{guilty}(\text{butler}) \vee \text{got_cream}(\text{butler})$

Step 3 : Negate the proof sentence

As sentence 4 is the one to be proved.

$\text{guilty}(\text{butler}) \wedge \sim \text{got_cream}(\text{butler})$

Step 4 : Proof by resolution $\wedge \vee \text{got_cream}(\text{butter})$



Hence proved.

Ex.3.11.6 : Consider following axioms.

All people who are graduating are happy.

All happy people smile.

Someone is graduating.

(i) Represent these axioms in FOL.

(ii) Convert each formula to CNF.

(iii) Prove that someone is smiling using resolution technique. Draw the resolution tree.

MU - Dec. 15, 12 Marks

Soln. :

Step 1 : Converting axioms to FOL.

(i) $\forall x : \text{graduating}(x) \rightarrow \text{happy}(x)$.

(ii) $\forall x : \text{happy}(x) \rightarrow \text{smile}(x)$

(iii) $\exists x : \text{graduating}(x)$

Step 2 : Converting FOL to CNF.

(i) $\sim \text{graduating}(x) \vee \text{happy}(x)$

(ii) $\sim \text{happy}(x_1) \vee \text{smile}(x_1)$

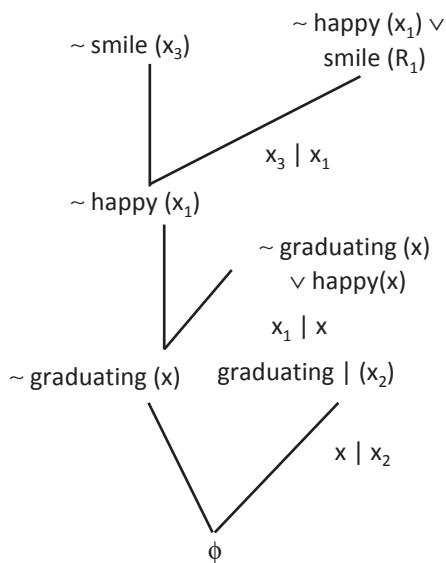
(iii) $\text{graduating}(x_2)$

Step 3 : T.P.T. $x_3 \text{ smile}(x_3)$

Negating the stmt

$\sim \text{smile}(x_3)$

Proof by resolution



Hence our assumption is wrong.

Hence proved.

3.12 Planning

- Planning various tasks is a part of day to day activities in real world. Say, you have tests of two different subjects on one day then, you will plan your study timetable as per your strengths and weaknesses in those two subjects.
- Also you must have learnt about various scheduling algorithms (e.g. first in first out) in operating systems subject and how printers plan/schedule their printing tasks based on tasks importance while printing.
- These examples illustrate how planning is important. We have seen that artificially intelligent systems are rational systems. So, devising a plan to perform actions becomes a part of creating an artificially intelligent agent. When we think about giving intelligence to a system or device, we have to make sure that it prioritizes between given activities or tasks.
- In this we are going to learn about how the machines can become more intelligent by using planning while performing various actions.

3.12.1 Introduction to Planning

MU - Dec. 13

Q. What is planning? How it differs from searching ?

(Dec. 13, 10 Marks)

- **Planning** in Artificial Intelligent can be defined as a problem that needs decision making by intelligent systems to accomplish the given target.
- The intelligent system can be a robot or a computer program.
- Take example of a driver who has to pick up and drop people from one place to another. Say he has to pick up two people from two different places then he has to follow some sequence, he cannot pick both passengers at same time.
- There is one more definition of planning which says that, **Planning** is an activity where agent has to come up with a sequence of actions to accomplish target.
- Now, let us see what information is available while formulating a planning problem and what results are expected.
- We have information about the initial status of the agent, goal conditions of agent and set of actions an agent can take.

- Aim of an agent is to find the proper sequence of actions which will lead from starting state to goal state and produce an efficient solution.

3.12.2 Simple Planning Agent

- Take example of an agent which can be a coffee maker, a printer and a mailing system, also assume that there are 3 people who have access to this agent.
- Suppose at same time if all 3 users of an agent give a command to execute 3 different tasks of coffee making, printing and sending a mail.
- Then as per definition of planning, agent has to decide the sequence of these actions.

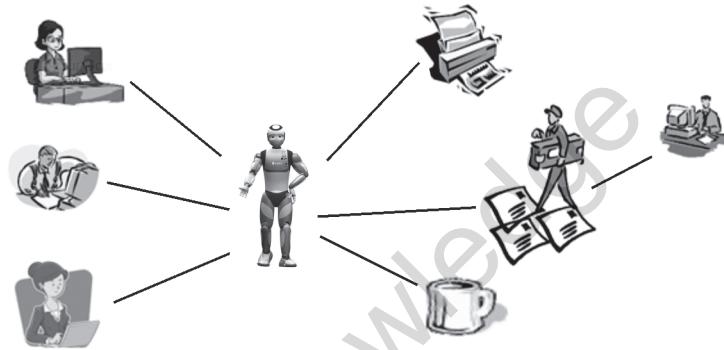


Fig. 3.12.1 : Example of a Planning Problem

- Fig. 3.12.2 depicts a general diagrammatic representation of a planning agent that interacts with environment with its sensors and effectors/actuators. When a task comes to this agent it has to decide the sequence of actions to be taken and then accordingly execute these actions.

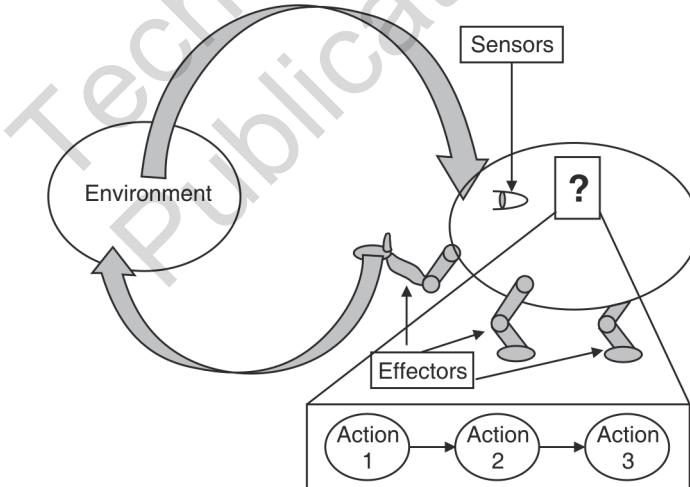


Fig. 3.12.2 : Planning agent

3.13 Planning Problem

MU - Dec. 12

Q. What is planning problem ?

(Dec. 12, 2.5 Marks)

- We have seen in above section what information is available while formulating a planning problem and what results are expected. Also it is understandable here that, states of an agent correspond to the probable surrounding environments while the actions and goals of an agent are specified based on logical formalization.



- Also we have learnt about various types of intelligent agents in chapter 1. Which shows that, to achieve any goal an agent has to answer few questions like “what will be the effect of its actions”, “how it will affect the upcoming actions”, etc. This illustrates that an agent must be able to provide a proper reasoning about its future actions, states of surrounding environments, etc.
- Consider simple Tic-Tac-Toe game. A Player cannot win a game in one step, he/she has to follow sequence of actions to win the game. While taking every next step he/she has to consider old steps and has to imagine the probable future actions of an opponent and accordingly make the next move and at the same time he/she should also consider the consequences of his/her actions.
- **A classical planning has the following assumptions about the task environment:**
 - **Fully Observable :**Agent can observe the current state of the environment.
 - **Deterministic:** Agent can determine the consequences of its actions.
 - **Finite:** There are finite set of actions which can be carried out by the agent at every state in order to achieve the goal.
 - **Static :**Events are steady. External event which cannot be handled by agent is not considered.
 - **Discrete :**Events of the agent are distinct from starting step to the ending(goal) state in terms of time.
- So, basically a planning problem finds the sequence of actions to accomplish the goal based on the above assumptions.
- Also goal can be specified as a union of sub-goals.
- Take example of ping pong game where, points are assigned to opponent player when a player fails to return the ball within the rules of ping-pong game. There can a best 3 of 5 matches where, to win a match you have to win 3 games and in every game you have to win with a minimum margin of 2 points.

3.13.1 Problem Solving and Planning

MU - Dec. 12, Dec. 13, Dec. 15

Q. How planning problem differs from search problem?	(Dec. 12, 2.5M)
Q. How planning defers from searching ?	(Dec. 13, 5 Marks)
Q. Compare and contrast problem solving agent and planning agent.	(Dec. 15, 5 Marks)

- Generally problem solving and planning methodologies can solve similar type of problems. Main difference between problem solving and planning is that planning is a more open process and agents follow logic-based representation. Planning is supposed to be more powerful than problem solving because of these two reasons.
- Planning agent has situations (i.e. states), goals (i.e. target end conditions) and operations (i.e. actions performed).All these parameters are decomposed into sets of sentences and further in sets words depending on the need of the system.
- Planning agents can deal with situations/states more efficiently because of its explicit reasoning capability also it can communicate with the world. Agents can reflect on their targets and we can minimize the complexity of the planning problem by independently planning for sub-goals of an agent. Agents have information about past actions, presents actions and the important point is that it can predict the effect of actions by inspecting the operations.
- Planning is a logical representation, based on situation, goals and operations, of problem solving.

Planning = Problem solving + Logical representation

3.14 Goal of Planning

- We plan activities in order to achieve some goals. Main goal can be divided into sub-goals to make planning more efficient.

- Take example of a grocery shopping at supermarket, suppose you want to buy milk, bread and egg from supermarket, then your initial state will be – “at home” and goal state will be – “get milk, bread and egg”.
- Now if you look at the Fig. 3.14.1 you will understand that branching factor can be enormous depending upon the set of actions, for e.g. Watch TV, read book, etc., available at that point of time.

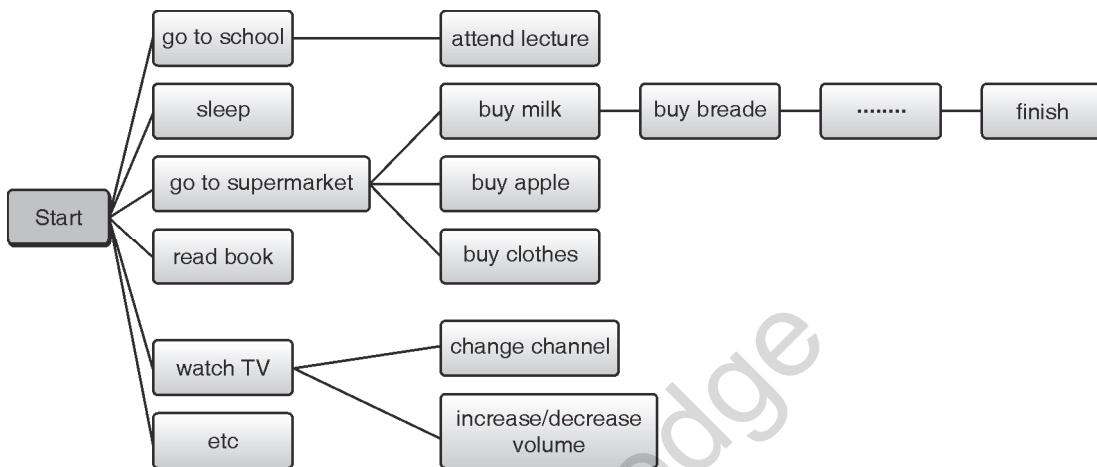


Fig. 3.14.1 : Supermarket examples to understand need of planning

- Thus **branching factor** can be defined as a set of all probable actions at any state, set can be very large, such as in the supermarket example or block problem. If the domain of probable actions increases, the branching factor will also increase as they are directly proportional to each other, this will result in the increase of search space.
- To reach the goal state you have to follow many steps, if you consider using heuristic functions, then you have to remember that it will not be able to eliminate states; these functions will be helpful only for guiding the search of states.
- So it becomes difficult to choose best actions. (i.e. even if we go to supermarket we need to make sure that all three listed items are picked, only then goal state can be achieved).
- As there are many possible actions and it is difficult to describe every state, and there can be combined goals (as seen in supermarket example) searching is inadequate to achieve goals efficiently. In order to be more efficient planning is required.
- In above sections, we discussed that planning requires explicit knowledge that means in case of planning we need to know the exact sequence of actions which will be useful in order to achieve the goal.
- Advantage of planning is that, the order of planning and the order of execution need not be same. For example, you can plan how to pay bills for grocery before planning to go to supermarket.
- Another advantage of planning is that you can make use of divide and conquer policy by dividing /decomposing the goal into sub goals.

3.14.1 Major Approaches

- There are many approaches to solve planning problems. Following are few **major approaches** used for planning :
 - o Planning with state space search.
 - o Partial ordered planning.
 - o Hierarchical planning / hierarchical decomposition (HTN planning).
 - o Planning situation calculus/ planning with operators.
 - o Conditional planning.
 - o Planning with operators.

- Planning with graphs.
- Planning with propositional logic.
- Planning reactive.
- Out of these major approaches we will be learning about following approaches in detail :
 - Planning with state space search
 - Partial ordered planning and
 - Hierarchical planning / Hierarchical decomposition (HTN planning).
 - Conditional planning.
 - Planning with operators.

3.15 Planning Graphs

- Planning graph is a special data structure which is used to get better accuracy. It is a directed graph and is useful to accomplish improved heuristic estimates. Any of the search technique can make use of planning graphs. Also GRAPHPLAN can be used to extract a solution directly.
- Planning graphs work only for propositional problems without variables. You have learnt Gantt charts. Similarly, in case of planning graphs there are series of levels which match to time ladder in the plan. Every level has set of literals and a set of actions. Level 0 is the initial state of planning graph.

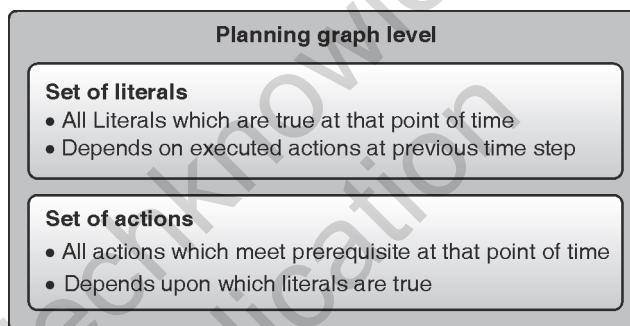


Fig. 3.15.1 : Planning graph level

Example

- Init(Have(Apple))
- Goal(Have(Apple) \wedge Ate(Apple))
- Action(Eat(Apple), PRECOND: Have(Apple))
- EFFECT: \neg Have(Apple) \wedge Ate(Apple))
- Action(Cut(Apple), PRECOND: \neg Have(Apple))
- EFFECT: Have(Apple))

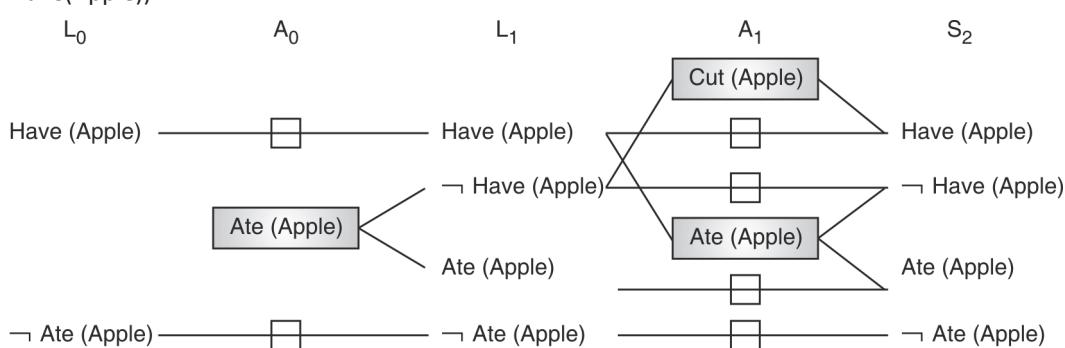


Fig. 3.15.2

- Start at level L0 and determine action level A0 and next level L1.
 - o A0 >> all actions whose prerequisite is satisfied at previous level.
 - o Connect precondition and effect of actions L0 → L1.
 - o Inaction is represented by *persistence actions*.
 - o Level A0 contains the possible actions.
 - o Conflicts between actions are shown by *mutual exclusion* links.
 - o Level L1 contains all literals that could result from picking any subset of actions in Level A0.
 - o Conflicts between literals which cannot occur together (as a effect of selection action) are represented by mutual exclusion links.
 - o L1 defines multiple states and the mutual exclusion links are the constraints that define this set of states.
 - o Continue until two consecutive levels are the same or contain the same amount of literals.

A mutual exclusion relation holds between two actions when :	A mutual exclusion relation holds between two literals when :
<ul style="list-style-type: none">- One action cancels out the effect of another action.- One of the effects of action is negation of preconditions of other action.- One of the preconditions of one action is mutually exclusion with the precondition of the other action.	<ul style="list-style-type: none">- If one literal is the negation of the other literal OR.- If each possible action pair that could achieve the literal is mutually exclusive.

GRAPH PLAN algorithm

GRAPHPLAN can directly extract solution from planning graph with the help of following algorithm :

```
function GRAPHPLAN(problem) return solution or failure
    graph ← INITIAL-PLANNING-GRAph(problem)
    goals ← GOALS[problem]
    loop do
        if goals all non-mutex in last level of graph then do
            solution ← EXTRACT-SOLUTION(graph, goals,
LENGTH(graph))
            if solution ≠ failure then return solution
            else if NO-SOLUTION-POSSIBLE(graph) then return failure
            graph ← EXPAND-GRAph(graph, problem)
```

Properties of planning graph

- If goal is absent from last level then goal cannot be achieved!.
- If there exists a path to goal then goal is present in the last level.
- If goal is present in last level then there may not exist any path.

3.16 Planning as State-Space Search

- We have seen example of an agent that can perform three tasks of printing, sending a mail and making coffee namely lets' call this agent as office agent.
- When this office agent gets order from three people at a same time to perform these three different tasks then, let us see how planning with state space search problem will look if we have a finite space.

- You can understand from Fig. 3.16.1 that the office agent is at location 250 on the state space grid. When he gets a task he has to decide which task can be performed more efficiently in lesser time.

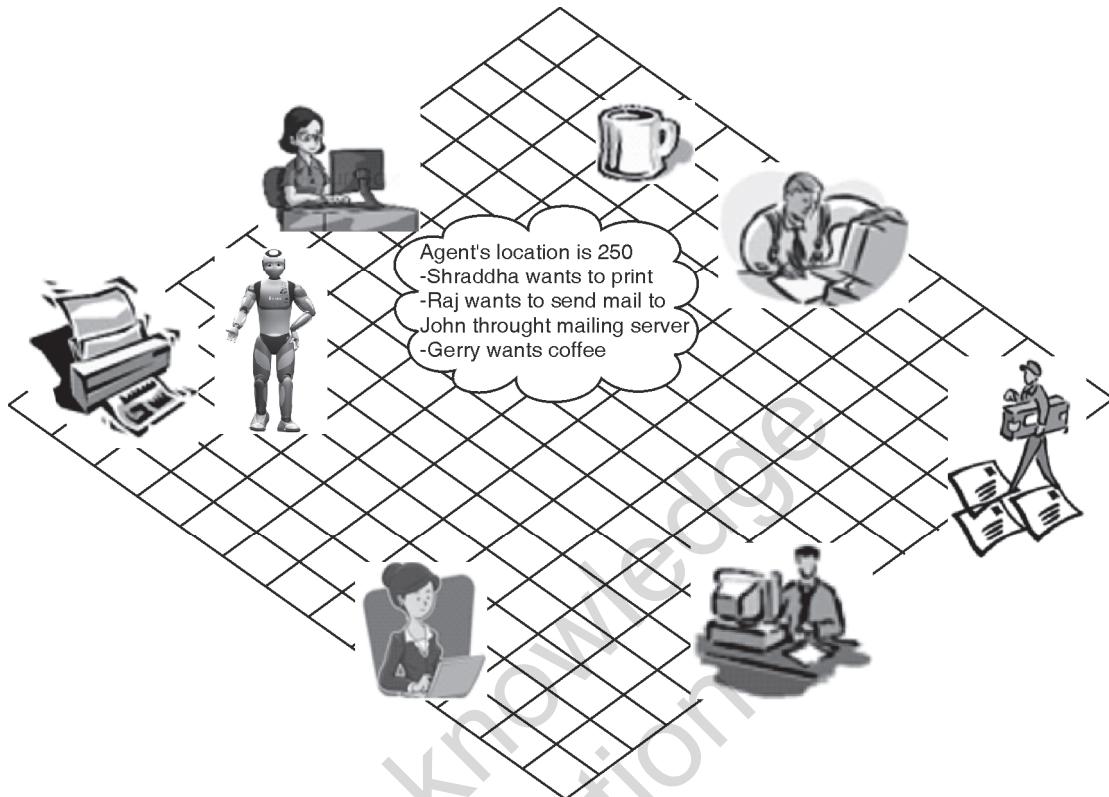


Fig. 3.16.1 : Office agent example with finite state space

- If it finds some input and output locations nearer on state space grid for example in case of printing task then the probability of performing that task will increase.
- But to do this it should be aware of its own current location, the locations of people who are assigning tasks and the locations of the required devices.
- State space search is unfavourable for solving real-world problems because, it requires complete description of every searched state, also search should be carried out locally.
- There can be two ways of representations for a state:

1. Complete world description
2. Path from an initial state

1. Complete world description

- Description is available in terms of an assignment of a value to each previous suggestion.
- Or we can say that description is available as a suggestion that defines the state.
- Drawback of this types is that it requires a large amount of space.

2. Path from an initial state

- As per the name, path from an initial state gives the sequence of actions which are used to reach a state from an initial state.
- In this case, what holds in a state can be deduced from the axiom which specifies the effects of an actions.

- Drawback of these types is that it does not explicitly specify “What holds in every state”. Because of this it can be difficult to determine whether two states are same.

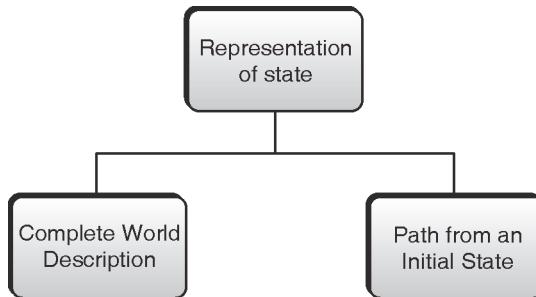


Fig. 3.16.2 : Representation of states

3.16.1 Example of State Space Search

Let us take an example of a water jug problem

- We have two jugs, a 4- gallon one and a 3- gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water how can you get exact 2 gallons of water into the 4- gallon jug?
- The state space for this problem can be described as of ordered pairs of integers (x, y) , such that $x = 0, 1, 2, 3$ or 4 , representing the number of gallons of water in the 4- gallon jug and $y = 0, 1, 2$ or 3 , representing the quantity of water in the 3- gallon jug. The start state is $(0, 0)$. The goal state is $(2, n)$ for any value of n , since the problem does not specify how many gallons need to be in the 3- gallon jug.
- The operators to be used to solve the problem can be described as shown bellow. They are represented as rules whose left side are matched against the current state and whose right sides describe the new state that results from applying the rule.

Rule set

1. $(x,y) \rightarrow (4,y)$ fill the 4- gallon jug
If $x < 4$
2. $(x,y) \rightarrow (x,3)$ fill the 3-gallon jug
If $x < 3$
3. $(x,y) \rightarrow (x-d,y)$ pour some water out of the 4- gallon jug
If $x > 0$
4. $(x,y) \rightarrow (x-d,y)$ pour some water out of the 3- gallon jug
If $y > 0$
5. $(x,y) \rightarrow (0,y)$ empty the 4- gallon jug on the ground
If $x > 0$
6. $(x,y) \rightarrow (x,0)$ empty the 3- gallon jug on the ground
If $y > 0$
7. $(x,y) \rightarrow (4,y-(4-x))$ pour water from the 3- gallon jug into the 4- gallon
If $x+y \geq 4$ and $y > 0$ jug until the 4-galoon jug is full
8. $(x,y) \rightarrow (x-(3-y),3)$ pour water from the 4- gallon jug into the 3-gallon
If $x+y \geq 3$ and $x > 0$ jug until the 3-gallon jug is full

9. $(x,y) \longrightarrow (x+y, 0)$ pour all the water from the 3-gallon jug into
If $x+y \leq 4$ and $y > 0$ the 3-gallon jug
 10. $(x,y) \longrightarrow (0, x+y)$ pour all the water from the 4-gallon jug into
If $x+y \leq 3$ and $x > 0$ the 3-gallon jug
 11. $(0,2) \longrightarrow (2,0)$ pour the 2-gallon from the 3-gallon jug into
the 4-gallon jug
 12. $(2,y) \longrightarrow (0,x)$ empty the 2 gallon in the 4 gallon on the ground

Production for the water jug problem

Gallons in the 4-gallon Jug	Gallons in the 3-gallon	Rule Applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

One solution to the water jug problem.

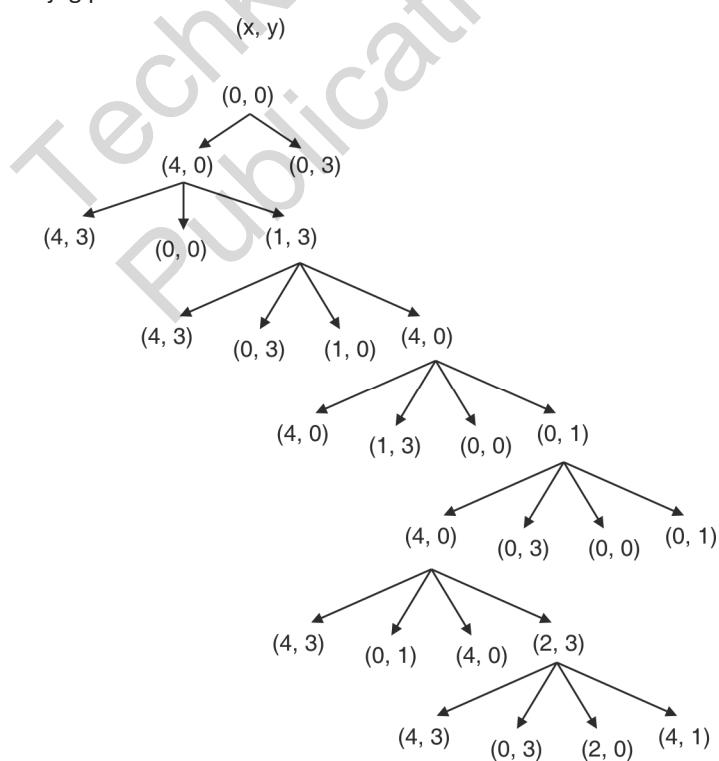


Fig. 3.16.3

3.17 Classification of Planning with State Space Search

- As the name suggests state space search planning techniques is based on the spatial searching.

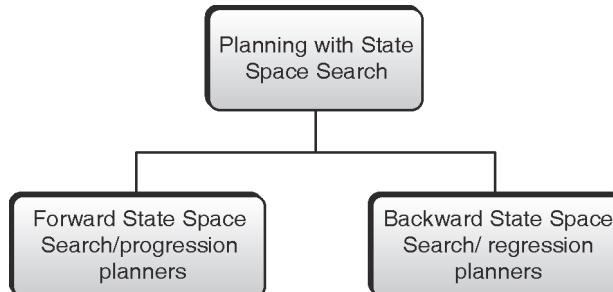


Fig. 3.17.1: Classification of Planning with State Space Search

- Planning with state space search can be done by both forward and backward state-space search techniques.

3.18 Progression Planners

- "Forward state-space search" is also called as "**progression planner**". It is a deterministic planning technique, as we plan sequence of actions starting from the initial state in order to attain the goal.
- With forward state space searching method we start with the initial state and go to final goal state. While doing this we need to consider the probable effects of the actions taken at every state.
- Thus the prerequisite for this type of planning is to have initial world state information, details of the available actions of the agent, and description of the goal state.
- Remember that details of the available actions include preconditions and effects of that action.
- See Fig. 3.18.1 it gives a state-space graph of progression planner for a simple example where flight 1 is at location A and flight 2 is also at location A. these flights are moving from location A to location B. In 1st case only flight 1 moves from location A to location B, so the resulting state shows that after performing that action flight 1 is at location B where as flight 2 is at its original location – A. similarly In 2nd case only flight 2 moves from location A to location B and the resulting state shows that after performing that action flight 2 is at location B while flight 1 is at its original location – A.

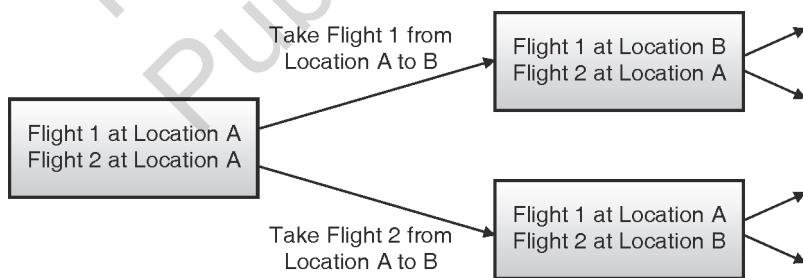


Fig. 3.18.1 : State-space graph of a Progression planners

- It can be observed from the Fig. 3.18.1 that, rectangles show state of the flights (i.e. their current location), and lines give the corresponding actions from one state to another (i.e. move from one location to other location).
- Here the lines coming out of every state matches to all of the permissible actions which can be accepted if the agent is in that state.

Progression planner algorithm

- Formulate the state space search problem :
 - Initial state is the first state of the planning problem which has a set of positive, the literals which don't appear are considered as false.

- If preconditions are satisfied then the actions are favoured i.e. if the preconditions are satisfied then positive effect literals are added for that action else the negative effect literals are deleted for that action.
 - Perform goal testing by checking if the state will satisfy the goal.
 - Lastly keep the step cost for each action as 1.
2. Consider example of A* algorithm. A complete graph search is considered as a complete planning algorithm. Functions are not used.
3. Progression planner algorithm is supposed to be inefficient because of the irrelevant action problem and requirement of good heuristics for efficient search.

3.19 Regression Planners

- “Backward state-space search” is also called as “**regression planner**” from the name of this method you can make out that the processing will start from the finishing state and then you will go backwards to the initial state.
- So basically we try to backtrack the scenario and find out the best possibility, in-order to achieve the goal to achieve this we have to see what might have been correct action at previous state.
- In forward state space search we used to need information about the successors of the current state now, for backward state-space search we will need information about the predecessors of the current state.
- Here the problem is that there can be many possible goal states which are equally acceptable. That is why this approach is not considered as a practical approach when there are large numbers of states which satisfy the goal.
- Let us see flight example, here you can see that the goal state is flight 1 is at location B and flight 2 is also at location B. We can see in Fig. 3.19.1. If this state is checked backwards we have two acceptable states in one state only flight 2 is at location B, but flight 1 is at location A and similarly in 2nd possible state flight 1 is already at location B, but flight 2 is at location A.
- As we search backwards from goal state to initial state, we have to deal with partial information about the state, since we do not yet know what actions will get us to goal. This method is complex because we have to achieve a conjunction of goals.

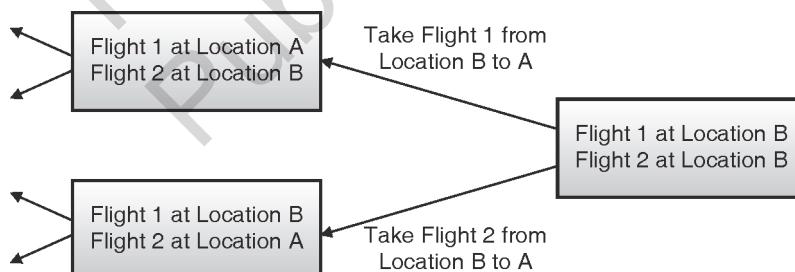


Fig. 3.19.1 : State-space graph of a regression planners

- In this Fig. 3.19.1, rectangles are goals that must be achieved and lines shows the corresponding actions.

Regression algorithm

1. Firstly predecessors should be determined :
 - To do this we need to find out which states will lead to the goal state after applying some actions on it.
 - We take conjunction of all such states and choose one action to achieve the goal state.
 - If we say that “X” action is relevant action for first conjunct then, only if pre-conditions are satisfied it works.
 - Previous state is checked to see if the sub-goals are achieved.

2. Actions must be consistent it should not undo preferred literals. If there are positive effects of actions which appear in goal then they are deleted. Otherwise Each precondition literal of action is added, except it already appears.
3. Main advantage of this method is only relevant actions are taken into consideration. Compared to forward search, backward search method has much lower branching factor.

3.19.1 Heuristics for State-Space Search

- Progression or Regression is not very efficient with complex problems. They need good heuristic to achieve better efficiency. Best solution is NP Hard (NP stands for Non-deterministic and Polynomial-time).
- There are two ways to make state space search efficient:
 - **Use linear method :**Add the steps which build on their immediate successors or predecessors.
 - **Use partial planning method :**As per the requirement at execution time ordering constraints are imposed on agent.

3.20 Total Order Planning (TOP)

- We have seen in above section that forward and regression planners impose a total ordering on actions at all stages of the planning process.
- In case of Total Order Planning (TOP), we have to follow sequence of actions for the entire task at once and to do this we can have multiple combinations of required actions. Here, we need to remember one most important thing which should be taken care of, is that TOP should take care of preconditions while creating the sequence of actions.
- For example, we cannot wear left shoe without wearing the left sock and we cannot wear the right shoe without wearing the right sock. So while creating the sequence of actions in total ordered planning, wearing left sock action should be executed before wearing the left shoe and wearing the right sock action should be executed before wearing the right shoe. As you can see in Fig.3.20.1.

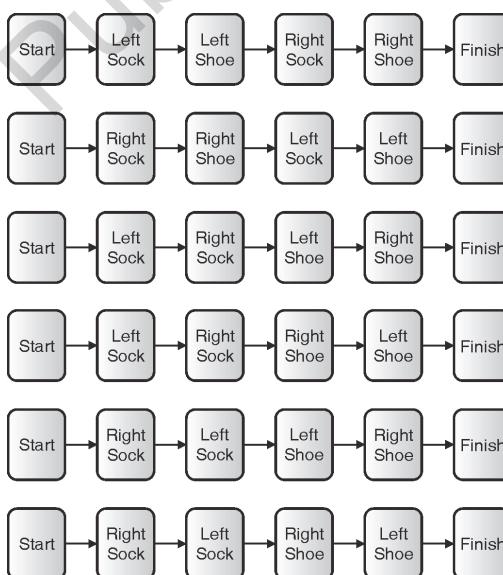


Fig. 3.20.1 :Total order planning of Wearing Shoe

- If there is a cycle of constraints then, total ordered planning cannot give good results. TOP can fail in non-cooperative environments. So we have Partial Ordered Planning method.

3.21 Partial Order Planning

MU – May 13, May 14, Dec. 14, May 15, Dec. 15

Q. Explain partial order planner with an example.	(Dec. 15, 6 Marks)
Q. Define partial order planner.	(May 13, Dec. 14, 5 Marks)
Q. Discuss partial order planning giving suitable example.	(May 14, May 15, 10 Marks)

- In case of **Partial Ordered Planning (POP)**, ordering of the actions is partial. Also partial ordered planning does not specify which action will come first out of the two actions which are placed in plan.
- With partial ordered planning, problem can be decomposed, so it can work well in case the environment is non-cooperative.
- Take same example of wearing shoe to understand partial ordered planning.

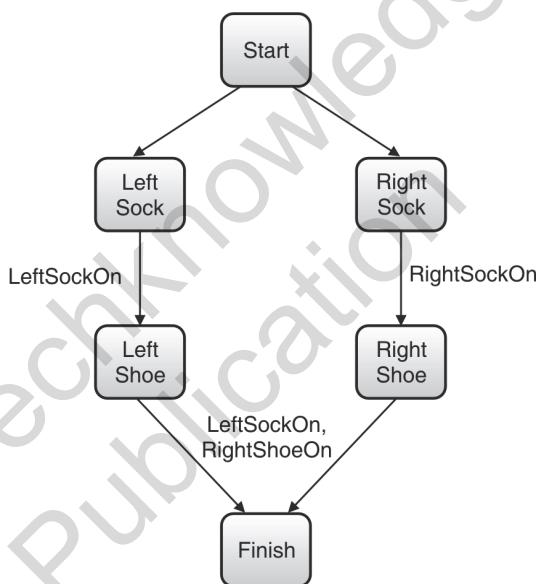


Fig. 3.21.1 : Partial order planning of Wearing Shoe

- A partial order planning combines two action sequences
 - First branch covers left-sock and left-shoe.
 - In this case to wear a left shoe, wearing left sock is the precondition, similarly.
 - Second branch covers right-sock and right-shoe.
 - Here, wearing a right sock is the precondition for wearing the right shoe.
- Once these actions are taken we achieve our goal and reach the finish state.

3.21.1 POP as a Search Problem

MU - Dec. 14, May 15, Dec. 15

Q. Define partial order planner.	(Dec. 14, May 15, Dec. 15, 5/6 Marks)
----------------------------------	---------------------------------------

- If we considered POP as a search problem, then we say that states are small plans.
- States are generally unfinished actions. If we take an empty plan then, it will consist of only starting and finishing actions.
- Every plan has four main components, which can be given as follows:

1. Set of actions

- These are the steps of a plan. Actions which can be performed in order to achieve goal are stored in set of actions component.
- **For example :** Set of Actions={ Start, Rightsock, Rightshoe, Leftsock, Leftshoe, Finish}
- Here, wearing left sock, wearing left shoe, wearing right sock, wearing right shoe are set of actions.

2. Set of ordering constraints/ preconditions

- Preconditions are considered as ordering constraints. (i.e. without performing action "x" we cannot perform action "y")
- **For example :** Set of Ordering ={Right-sock < Right-shoe; Left-sock < Left-shoe} that is In order to wear shoe, first we should wear a sock.
- So the ordering constraint can be Wear Left-sock < wear Left-shoe (Wearing Left-sock action should be taken before wearing Left-shoe) Or Wear right-sock < wear right-shoe (Wearing right-sock action should be taken before wearing right-shoe).
- If constraints are cyclic then it represents inconsistency.
- If we want to have a consistent plan then there should not be any cycle of preconditions.

3. Set of causal links

- Action A achieves effect "E" for action B

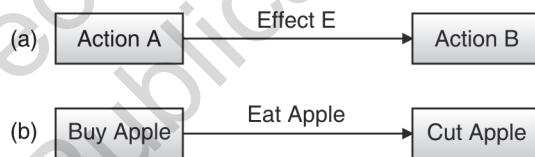


Fig.3.21.2 : (a) Causal Link Partial Order Planning (b) Causal Link Example

- From Fig.3.21.2(b) you can understand that if you buy an apple its effect can be eating an apple and the precondition of eating an apple is cutting apple.
- There can be conflict if there is an action C that has an effect $\neg E$ and, according to the ordering constraints it comes after action A and before action B.
- Say we don't want to eat an apple instead of that we want to make a decorative apple swan. This action can be between A and B and It does not have effect "E".
 - o **For example:** Set of Causal Links = {Right-sock->Right-sock-on → Right-shoe, Leftsock → Leftsockon → Leftshoe, Rightshoe → Rightshoeon → Finish, leftshoe → leftshoeon → Finish }.
 - o To have a consistent plan there should not be any conflicts with the causal links.

4. Set of open preconditions

- Preconditions are called open if it cannot be achieved by some actions in the plan. Least commitment strategy can be used by delaying the choice during search.
- To have a consistent plan there should not be any open precondition

3.21.2 Consistent Plan is a Solution for POP Problem

- As consistent plan does not have cycle of constraints; it does not have conflicts in the causal links and does not have open preconditions so it can provide a solution for POP problem.
- While solving POP problem operators can add links and steps from existing plans to open preconditions in order to fulfil the open preconditions and then steps can be ordered with respect to the other steps for removing the potential conflicts. If the open precondition is unattainable, then backtrack the steps and try solving the problem with POP.
- Partial ordered planning is a more efficient method, because with the help of POP we can progress from vague plan to complete and correct solution in a faster way. Also we can solve a huge state space plan in less number of steps, this is because search takes place only when sub-plans interact.

3.22 Hierarchical Planning

- Hierarchical planning is also called as plan decomposition. Generally plans are organized in a hierarchical format.
- Complex actions can be decomposed into more primitive actions and it can be denoted with the help of links between various states at different levels of the hierarchy. This is called as operator expansion.

For example :

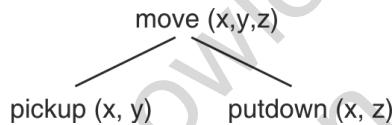


Fig. 3.22.1 : Operator expansion

- Fig. 3.22.2 shows, how to create a hierarchical plan to travel from some source to a destination. Also you can observe, at every level how we follow some sequence of actions.

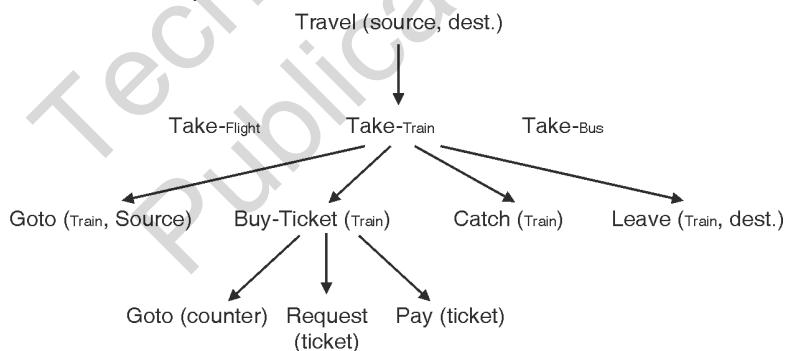


Fig. 3.22.2 : Hierarchical planning example

3.22.1 POP One Level Planner

- If you are planning to take a trip, then first you have to decide the location. To decide location we can search for various good locations from internet based on, weather conditions, travelling expenses, etc.
- Say we select Rajasthan, with one level planner, first we switch on PC, then we open browser, after that we open Indian Railways website booking site for ticket booking, then we enter the date, time, etc details to book the railway ticket. After that we will have to do hotel's ticket booking and so on.
- This type of planning is called **one level planning**. If the type of problem is simple then we can make use of one level planner. For complex problems, one level planner cannot provide good solution.

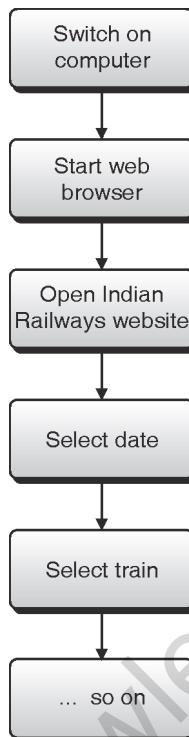


Fig. 3.22.3 : One Level planning example

3.22.2 Hierarchy of Actions

- In terms of major and minor actions, hierarchy of actions can be decided. Minor activities would cover more precise activities to accomplish the major activities. In case of above example, we can have railway Ticket Booking, Hotel Booking, Reaching Rajasthan, Staying and enjoying there, coming back are the major activities.
- While take a taxi to reach railway station, Have candle light dinner in palace, Take photos, etc are the Minor activities.
- In real world there can be complex problems.
- **For example :** A captain of a cricket team plans the order of 4 bowlers in 2 days of a test match(180 overs).Number of probabilities : $4^{180} = 16^{90}$.
- Motivation behind this planning is to reduce the size of search space. For plan ordering we have to try out a large number of possible plans. With plan hierarchies we have limited ways in which we can select and order primitive operators.
- In hierarchical planning major steps are given more importance. Once the major steps are decided we attempt to solve the minor detailed actions.
- It is possible that major steps of plan may run into difficulties at a minor step of plan. In such case we need to return to the major step again to produce appropriately ordered sequence to devise the plan.

3.22.3 Planner

1. First identify a hierarchy of major conditions.
2. Construct a plan in levels (Major steps then minor steps), so we postpone the details to next level.
3. Patch major levels as detail actions become visible.
4. Finally demonstrate.

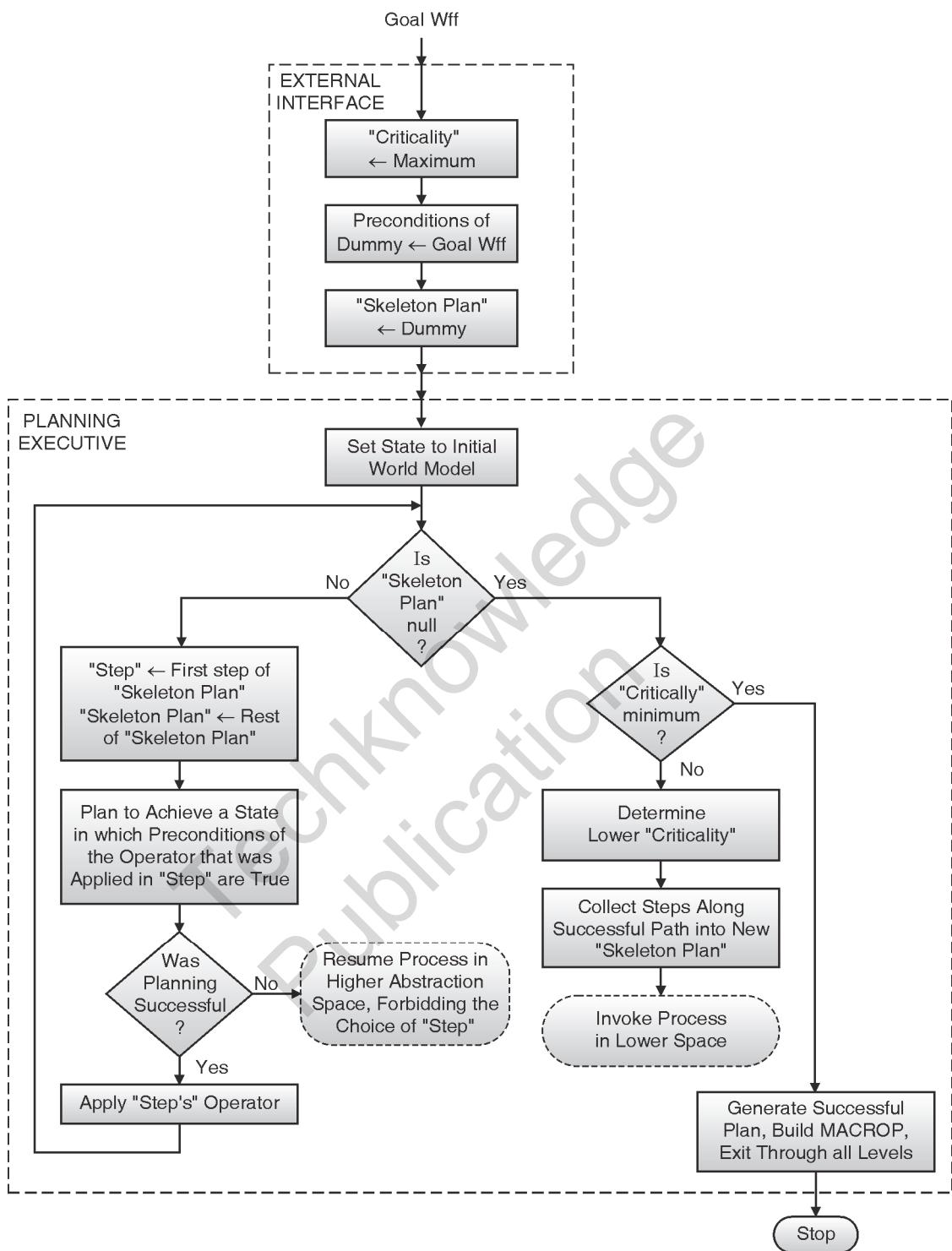


Fig. 3.22.4 : Planner

Example :

Actions required for “Travelling to Rajasthan” can be given as follows :

- Opening yatra.com (1)
- Finding train (2)
- Buy Ticket (3)



- Get taxi(2)
- Reach railway station(3)
- Pay-driver(1)
- Check in(1)
- Boarding train(2)
- Reach Rajasthan (3)

1st level plan

Buy Ticket (3), Reach Railway Station(3), Reach Rajasthan (3)

2nd level plan

Finding train (2), Buy ticket (3), Get taxi(2), Reach railway station (3), Boarding train(2), Reach Rajasthan (3).

3rd level plan (final)

Opening yatra.com (1), Finding train (2), Buy ticket (3), Get taxi(2), Reach Railway station (3), Pay-driver(1), Check in(1), Boarding train(2), Reach Rajasthan (3).

3.23 Planning Languages

MU - May 13, Dec. 14

Q. Explain STRIPS representation of planning problem.	(May 13, Dec. 14, 5 Marks)
--	-----------------------------------

- Language should be expressive enough to explain a wide variety of problems and restrictive enough to allow efficient algorithms to operate on it.
- Planning languages are known as action languages.

Stanford Research Institute Problem Solver (STRIPS)

- Richard Fikes and Nils Nilsson developed an automated planner called STRIPS (Stanford Research Institute Problem Solver) in 1971.
- Later on this name was given to a formal planning language. STRIPS is foundation for most of the languages in order to express automated planning problem instances in current use.

Action Description Language (ADL)

ADL is an advancement of STRIPS. Pednault proposed ADL in 1987.

Comparison between STRIPS and ADL

Sr. No.	STRIPS language	ADL
1.	Only allows positive literals in the states, For example : A valid sentence in STRIPS is expressed as $\Rightarrow \text{Intelligent} \wedge \text{Beautiful}$.	Can support both positive and negative literals. For example : Same sentence is expressed as $\Rightarrow \neg \text{Stupid} \wedge \neg \text{Ugly}$
2.	Makes use of closed-world assumption (i.e. Unmentioned literals are false)	Makes use of Open World Assumption (i.e. unmentioned literals are unknown)
3.	We only can find ground literals in goals. For example : Intelligent \wedge Beautiful.	We can find quantified variables in goals. For example : $\exists x \text{ At}(P1, x) \wedge \text{At}(P2, x)$ is the goal of having P1 and P2 in the same place in the example of the blocks

Sr. No.	STRIPS language	ADL
4.	Goals are conjunctions For example : (Intelligent \wedge Beautiful).	Goals may involve conjunctions and disjunctions For example : (Intelligent \wedge (Beautiful \vee Rich)).
5.	Effects are conjunctions	Conditional effects are allowed: when P:E means E is an effect only if P is satisfied
6.	Does not support equality.	Equality predicate ($x = y$) is built in.
7.	Does not have support for types	Supported for types For example : The variable p : Person

3.23.1 Example of Block World Puzzle



Fig. 3.23.1

Standard sequence of actions is

1. Grab Z and Pickup Z
 2. Then Place Z on the table
 3. Grab Y and Pickup Y
 4. Then Stack Y on Z
 5. Grab X and Pickup X
 6. Stack X on Y
- Elementary problem is that framing problem in AI is concerned with the question of what piece of knowledge or information is pertinent to the situation.
 - To solve this problem we have to make an Elementary Assumption which is a Closed world assumption. (i.e. If something is not asserted in the knowledge base then it is assumed to be false, this is also called as "Negation by failure")
 - Standard sequence of actions can be given as for the block world problem :

on(Y, table)	on(Z, table)
on(X, table)	on(Y, Z)
on(Z, X)	on(X, Y)
hand empty	hand empty
clear(Z)	clear(X)
clear(Y)	



Fig. 3.23.2

- We can write 4 main rules for the block world problem as follows :

	Rule	Precondition and Deletion List	Add List
Rule 1	pickup(X)	hand empty, on(X,table), clear(X)	holding(X)
Rule 2	putdown(X)	holding(X)	hand empty, on(X,table), clear(X)
Rule 3	stack(X,Y)	holding(X), clear(Y)	on(X,Y), clear(X)
Rule 4	unstack(X,Y)	on(X,Y), clear(X)	holding(X), clear(Y)

- Based on the above rules, plan for the block world problem: Start \rightarrow goal can be specified as follows:

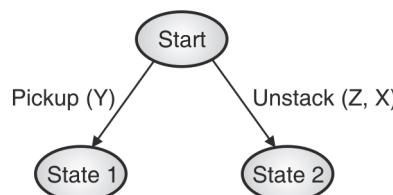
1. unstack(Z,X)
2. putdown(Z)
3. pickup(Y)
4. stack(Y,Z)
5. pickup(X)
6. stack(X,Y)

- Execution of this plan can be done by making use of a data structure called "Triangular Table".

1	on (C, A) clear (C) handempty	unstuck (C, A)					
2		holding (C)	putdown (C)				
3	on (B, table)		hand empty	pickup (B)			
4			clear (C)	Holding (B)	stack (B,C)		
5	on (A, table)	clear (A)			Handem pty	pickup (A)	
6					clear (B)	holding (A)	stack (A, B)
7			on (C, table)		on (B, C)		on (A, B) clear (A)
	0	1	2	3	4	5	6

Fig. 3.23.3

- In a triangular table there are $N + 1$ rows and columns. It can be seen from the Fig. 3.23.4 that rows have $1 \rightarrow n+1$ condition and for columns $0 \rightarrow n$ condition is followed. The first column of the triangular table indicates the starting state and the last row of the triangular table indicates the goal state.
- With the help of triangular table a tree is formed as shown below to achieve the goal state :

**Fig. 3.23.4**

- An agent (in this case robotic arm) can have some amount of fault tolerance. Fig. 3.23.5 shows one such example.

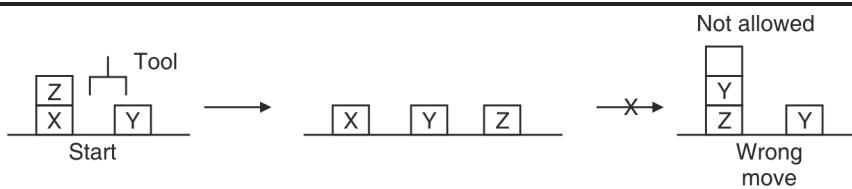


Fig. 3.23.5

3.23.2 Example of the Spare Tire Problem

- Consider the problem of changing a flat tire. More precisely, the goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. To keep it simple, our version of the problem is a very abstract one, with no sticky lug nuts or other complications.
- There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight. We assume that the car is in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear.
- The ADL description of the problem is shown. Notice that it is purely propositional. It goes beyond STRIPS in that it uses a negated precondition, $\neg \text{At}(\text{Flat}, \text{Axle})$, for the $\text{PutOn}(\text{Spare}, \text{Axle})$ action. This could be avoided by using $\text{Clear}(\text{Axle})$ instead, as we will see in the next example.

Solution using STRIPS

- $\text{Init}(\text{At}(\text{Flat}, \text{Axle}) \wedge \text{At}(\text{Spare}, \text{Trunk}))$
- $\text{Goal}(\text{At}(\text{Spare}, \text{Axle}))$ Action($\text{Remove}(\text{Spare}, \text{Trunk})$,
- PRECOND: $\text{At}(\text{Spare}, \text{Trunk})$
- EFFECT: $\neg \text{At}(\text{Spare}, \text{Trunk}) \wedge \text{At}(\text{Spare}, \text{Ground})$
- Action($\text{Remove}(\text{Flat}, \text{Axle})$,
- PRECOND: $\text{At}(\text{Flat}, \text{Axle})$
- EFFECT: $\neg \text{At}(\text{Flat}, \text{Axle}) \wedge \text{At}(\text{Flat}, \text{Ground})$
- Action($\text{PutOn}(\text{Spare}, \text{Axle})$,
- PRECOND: $\text{At}(\text{Spare}, \text{Ground}) \wedge \neg \text{At}(\text{Flat}, \text{Axle})$
- EFFECT: $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \text{At}(\text{Spare}, \text{Axle})$
- Action(LeaveOvernight)
- PRECOND
- EFFECT: $\neg \text{At}(\text{Spare}, \text{Ground}) \wedge \neg \text{At}(\text{Spare}, \text{Axle}) \wedge \neg \text{At}(\text{Spare}, \text{Trunk}) \wedge \neg \text{At}(\text{Flat}, \text{Ground}) \wedge \neg \text{At}(\text{Flat}, \text{Axle})$

3.24 Planning and Acting in the Real World

- In earlier sections, we have discussed planning that is observable and deterministic. Real world problems however are not predictable and completely unobservable. The planning and acting on real world problems require more sophisticated approach.
- One of the most important characteristic of real world is uncertainty. In an uncertain environment, it is very important for agent to rely upon its percepts (series of past experience)
- Whenever some unexpected condition encountered, agent should refer its percept and should change course of action accordingly. In other words agent should be able to cancel or replace the currently executing plan with some other more suitable and reliable plan if something unexpected happens.

- It should be noted that real world itself is not uncertain but human perception related to world is uncertain. In artificial intelligence, we try to give human perception ability to machine, and hence machine also receives the perception of uncertainty about the real world. So machine has to deal with incomplete and incorrect information like human does.
- Determining the condition of state depends on available knowledge. In real world, knowledge availability is always limited, so most of the time, conditions are non deterministic.
- The amount or degree of indeterminacy depends upon the knowledge available. The inter determinacy is called “bounded indeterminacy” when actions can have unpredictable effects.
- Four planning strategies are there for handling indeterminacy :

- (i) Sensorless planning
- (ii) Conditional planning
- (iii) Execution monitoring and replanning
- (iv) Continuous planning

(i) Sensorless planning

Sensorless planning is also known as conformant planning. These kinds of planning are not based on any perception. The algorithm ensures that plan should reach its goal at any cost.

(ii) Conditional planning

Conditional plannings are sometimes termed as contingency planning and deals with bounded indeterminacy discussed earlier. Agent makes a plan, evaluate the plan and then execute it fully or partly depending on the condition.

(iii) Execution monitoring and replanning

In this kind of planning agent can employ any of the strategy of planning discussed earlier. Additionally it observes the plan execution and if needed, replan it and again executes and observes.

(iv) Continuous planning

Continuous planning does not stop after performing action. It persist over time and keeps on planning on some predefined events. These events include any type unexpected circumstance in environment.

3.25 Multi-Agent Planning

- Whatever planning we have discussed so far, belongs to single user environment. Agent acts alone in a single user environment.
- When the environment consists of multiple agent to, then the way a single agent plan its action get changed.
- We have a glimpse of environment where multiple agents have to take actions based on current state. The environment could be co-operative or competitive. In both the cases agent's action influences each other.
- Few of the multi agent planning strategies are listed below :

- (i) Co-operation
- (ii) Multi body planning
- (iii) Co-ordination mechanisms
- (iv) Competition

(i) Co-operation

In co-operation strategy agents have joint goals and plans. Goals can be divided into sub goals but ultimately combined to achieve ultimate goal.

(ii) Multibody planning

Multi body planning is the strategy of implementing correct joint plan.

(iii) Co-ordination mechanisms

These strategies specify the co-ordination between co-operating agents. Co-ordination mechanism is used in several co-operating plannings.

(iv) Competition

Competition strategies are used when agents are not co-operating but competing with each other. Every agent wants to achieve the goal first.

3.26 Conditional Planning

- Conditional planning has to work regardless of the outcome of an action.
- Conditional Planning can take place in Fully Observable Environments (FOE) where the current state of the agent is known environment is fully observable. The outcome of actions cannot be determined so the environment is said to be nondeterministic.

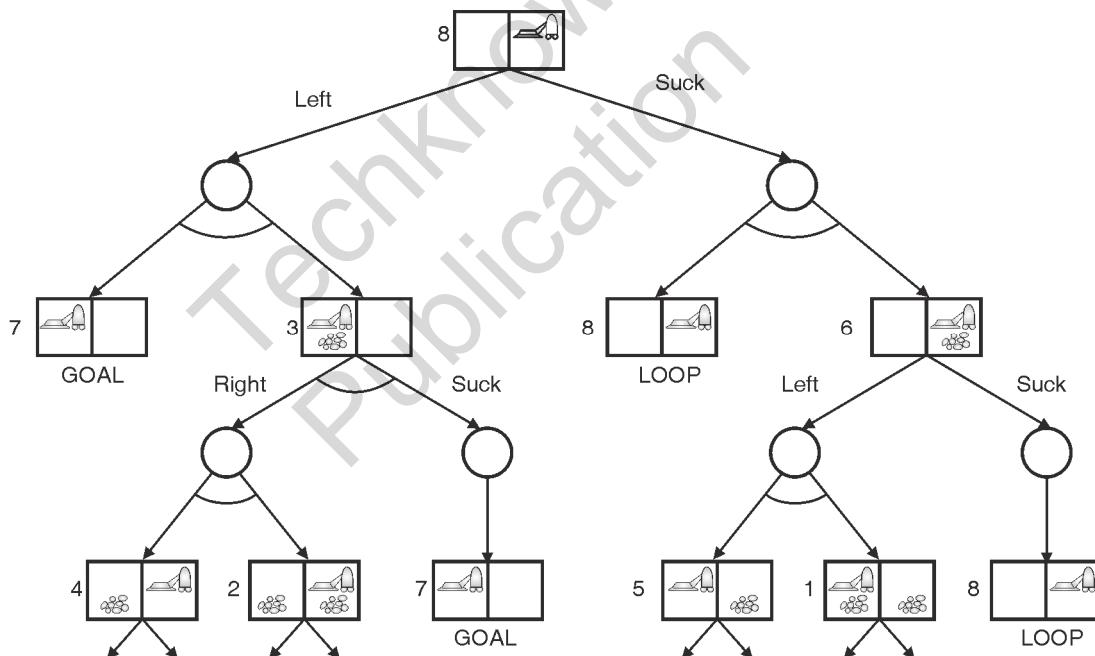


Fig. 3.26.1 : Conditional planning - vacuum world example

- In conditional planning we can check what is happening in the environment at predetermined points of the plan to deal with ambiguous actions.
- It can be observed from vacuum world example ,Conditional Planning needs to take some actions at every state and must be able to handle every outcome for the action it takes. A State node is represented with a square and chance node is represented with a circles.
- For a state node we have an option of choosing some actions. For a chance node agent has to handle every outcome.
- Conditional Planning can also take place in the Partially Observable Environments (POE) where, we cannot keep a track on every state. Actions can be Uncertain because of the imperfect sensors.

- In vacuum agent example if the dirt is at Right and agent knows about Right, but not about Left. Then, in such cases Dirt might be left behind when the agent, leaves a clean square. Initial state is also called as a state set or a belief state.
- Sensors play important role in Conditional Planning for partially observable environments. Automatic sensing can be useful; with automatic sensing an agent gets all the available percepts at every step. Another method is Active sensing, with which percepts are obtained only by executing specific sensory actions.

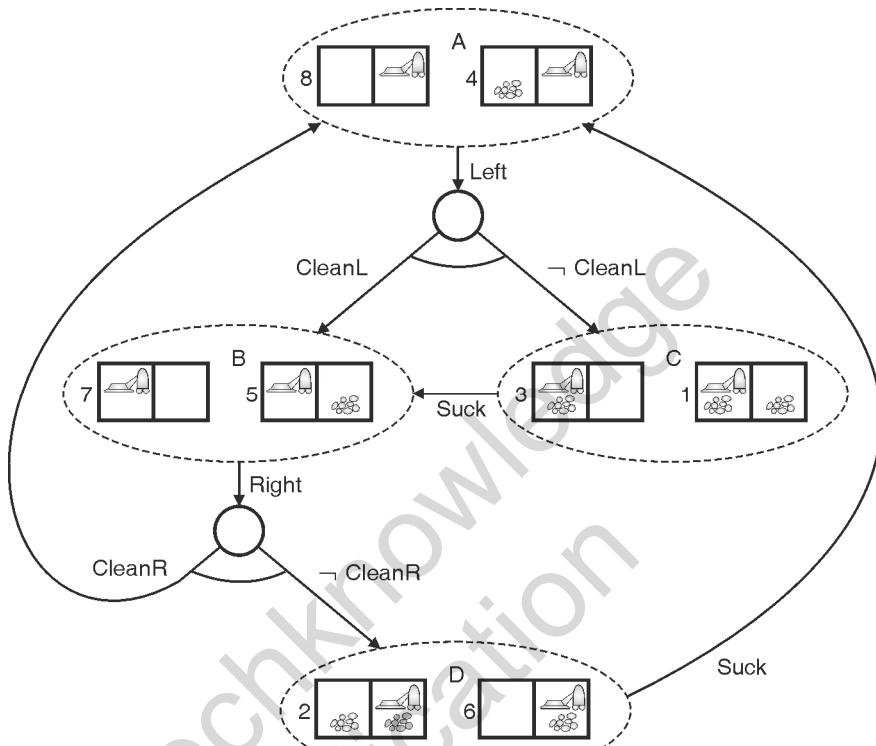


Fig. 3.26.2 :Conditional Planning - vacuum world example (condition 2)

Review Questions

- Q. 1 What is Knowledge Based Agent?
- Q. 2 Describe WUMPUS WORLD Environment. Specify PEAS properties and type of environment for the same.
- Q. 3 What is Logic? Explain various knowledge representation techniques.
- Q. 4 What is propositional logic? Write syntax and semantics and example sentences for propositional logic.
- Q. 5 Explain the inference process in case of propositional logic with suitable examples.
- Q. 6 Explain Horn Clause with example.
- Q. 7 What is first order logic? Write syntax and semantics of FOL with example.
- Q. 8 What is unification and lifting?
- Q. 9 Explain inference process in FOL using Forward Chaining and Backward Chaining.
- Q. 10 Write a short note on : Resolution.



- Q. 11 What is planning in AI?
- Q. 12 Explain planning problem.
- Q. 13 Explain goal of planning with supermarket example.
- Q. 14 What are the major approaches of planning? Explain conditional planning with example.
- Q. 15 Write short note on planning graphs.
- Q. 16 How planning strategies are classified? Explain regression planning.
- Q. 17 Explain progression planners with example.
- Q. 18 Explain regression planners with example.
- Q. 19 Explain total order planning with example.
- Q. 20 Explain partial order planning with example.
- Q. 21 Explain process of generating solution to partial order planning problem.
- Q. 22 Write short note hierarchical planning.
- Q. 23 What is one level planner?
- Q. 24 What are different languages used for implementation of planning?
- Q. 25 Compare and contrast planning languages.
- Q. 26 What are various planning strategies used to handle indeterminacy?
- Q. 27 Write a short note on multi-agent planning.
- Q. 28 Explain conditional planning with example.





Fuzzy Logic

Syllabus

- 4.1 Introduction to Fuzzy Set: Fuzzy set theory, Fuzzy set versus crisp set, Crisp relation & fuzzy relations, membership functions,
- 4.2 Fuzzy Logic: Fuzzy Logic basics, Fuzzy Rules and Fuzzy Reasoning
- 4.3 Fuzzy inference systems: Fuzzification of input variables, defuzzification and fuzzy controllers.

4.1 Introduction to Fuzzy Set

- Fuzzy logic was introduced by Prof. Lofti A. Zadeh in 1965.
- The word fuzzy means “Vagueness”.
- Fuzziness occurs when a boundary of a piece of information is not clear.
- He proposed a mathematical way of looking at the vagueness of the human natural language.

Why fuzzy set is required?

- Most of our traditional tools for formal modelling, reasoning and computing are crisp, deterministic and precise.
- While designing the system using classical set, we assume that the structures and parameters of the model are definitely known and there are no doubts about their values or their occurrence.
- But in real world there exists much fuzzy knowledge; knowledge that is vague, imprecise, uncertain, ambiguous, inexact or probabilistic in nature.
- There are two facts ;
 1. Real situations are very often not crisp and deterministic and they cannot be described precisely.
 2. The complete description of a real system often would require more detailed data than a human being could ever recognize simultaneously, process and understand.
- Because of these facts, modelling the real system using classical sets often do not reflect the nature of human concepts and thoughts which are abstract, imprecise and ambiguous.
- The classical (crisp) sets are unable to cope with such unreliable and incomplete information.
- We want our systems should also be able to cope with unreliable and incomplete information and give expert opinion.
- Fuzzy set theory has been introduced to deal with such unreliable, incomplete, vague and imprecise information.
- Fuzzy set theory is an extension to classical set theory where element have degree of membership.
- Fuzzy logic uses the whole interval between 0 (false) and 1 (true) to describe human reasoning.

4.2 Fuzzy Set vs. Crisp Set

- A classical set (or conventional or crisp set) is a set with a crisp boundary.

- For example, a classical set A of real numbers greater than 6 can be expressed as

$$A = \{x \mid x > 6\}$$

Where there is a clear, unambiguous boundary '6' such that if x is greater than this number, then x belongs to the set A, otherwise x does not belong to the set.

- Although classical sets are suitable for various approximations and have proven to be an important tool for mathematics and computer science, they do not reflect the nature of human concepts and thoughts, which are abstract, imprecise and ambiguous.
- For example, mathematically we can express the set of all tall persons as a collection of persons whose height is more than 6 ft.

$$A = \{x \mid x > 6\}$$

Where A = "tall person" and x = "height".

- The problem with the classical set is that it would classify a person 6.001 ft. tall as a tall person, but a person 5.999 ft. tall as "not tall". This distinction is intuitively unreasonable.

The flaw comes from the sharp transition between inclusion and exclusion in a set.(Fig. 4.2.1)

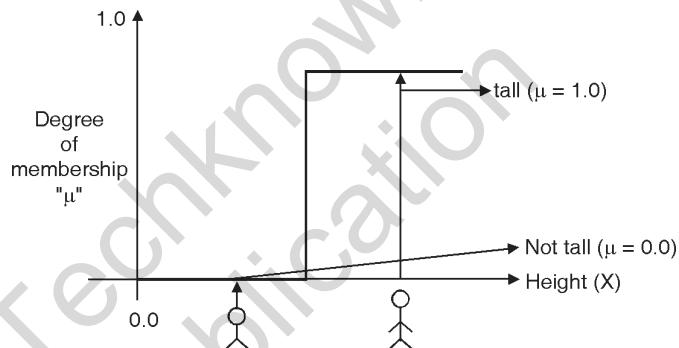


Fig. 4.2.1 : Sharp edged membership function for TALL

- Fuzzy logic uses the "degrees of truth" rather than the usual "true or false" (1 or 0) Boolean logic.
- Fuzzy logic includes 0 and 1 as extreme cases of truth but also includes the various states of truth in between so that, for example, the result of a comparison between two things could be not "tall" or "short" but "0.38 of tallness."

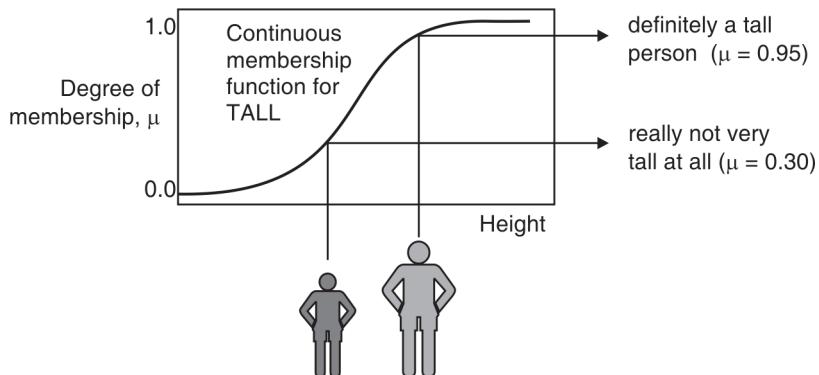


Fig 4.2.2 : Fuzzy membership function TALL



- As shown in Fig 4.2.2 the fuzzy logic defines smooth transition from ‘not tall’ to ‘tall’. A person’s height may now belong to both the groups “tall” and ‘Not all’ but now it will have the degree of membership associated with it for each group.
- A person has 0.30 membership in ‘Not tall’ group and ‘0.95’ membership in ‘tall’ group, so definitely the person is categorized as a tall person.

Key differences between Fuzzy Set and Crisp Set

Sr. No.	Fuzzy Set	Crisp Set
1.	A fuzzy set follows the infinite-valued logic	A crisp set is based on bi-valued logic.
2.	A fuzzy set is determined by its indeterminate boundaries, there exists an uncertainty about the set boundaries	A crisp set is defined by crisp boundaries, and contain the precise location of the set boundaries.
3.	Fuzzy set elements are permitted to be partly accommodated by the set (exhibiting gradual membership degrees).	crisp set elements can have a total membership or non-membership.
4.	Fuzzy sets are capable of handling uncertainty and vagueness present in the data	Crisp set requires precise , complete and finite data.

4.3 Fuzzy Set Theory

4.3.1 Fuzzy Set : Definition

- If X is a collection of objects denoted generally by x , then a fuzzy set \tilde{A} in X is defined as a set of ordered pairs :

$$\tilde{A} = \{ (x, \mu_{\tilde{A}}(x)) \mid x \in X \}$$

Where $\mu_{\tilde{A}}(x)$ is called the Membership Function (MF) for the fuzzy set \tilde{A} .

- The MF maps each element of X to a membership grade between 0 and 1.
- If the value of membership function $\mu_{\tilde{A}}(x)$ is restricted to either 0 or 1, then \tilde{A} is reduced to a classical set.

Note : Classical sets are also called ordinary sets, crisp sets, non fuzzy sets or just sets.

- Here, X is referred to as the **Universe of discourse** or simply the **Universe** and it may consist of discrete objects or continuous space.

4.3.2 Types of Universe of Discourse

1. Fuzzy sets with a discrete non ordered universe

- Universe of discourse may contain discrete non-ordered objects.

For example,

Let $X = \{\text{San Francisco, Boston, Los Angeles}\}$ be the set of cities one may choose to live in.

The fuzzy set "desirable city to live in" may be described as follows:

$$A = \{(\text{San Francisco}, 0.9), (\text{Boston}, 0.8), (\text{Los Angeles}, 0.6)\}$$

- Here, the universe of discourse X is discrete and it contains non-ordered objects, in this case three big cities in United States.

2. Fuzzy sets with a discrete ordered universe

- Let $X = \{0, 1, 2, 3, 4, 5, 6\}$ be the set of number of children a family may choose to have.
- Then, a fuzzy set "Sensible number of children in family" may be described as

$$\tilde{A} = \{(0, 0.1), (1, 0.3), (2, 0.7), (3, 1), (4, 0.7), (5, 0.3), (6, 0.1)\}$$

Here we have a discrete ordered universe X .

The MF is shown in Fig. 4.3.1.

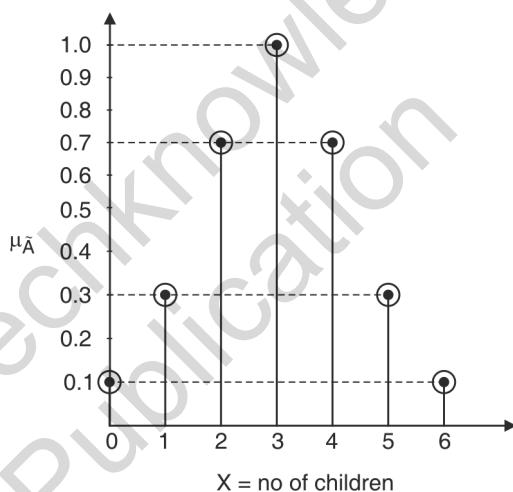


Fig. 4.3.1 : MF on a discrete universe

Note : Membership grades of the fuzzy set are subjective measures. (For example, the height 5'5" may be considered tall in Japan, but in Australia, it may be considered medium).

3. Fuzzy sets with a continuous universe

Let $X = R^+$ be the set of possible ages for human beings (Real numbers - continuous). Then the fuzzy set B = "about 50 years old" may be expressed as,

$$\tilde{B} = \{ (x, \mu_{\tilde{B}}(x)) \mid x \in X \}$$

$$\text{Where, } \mu_{\tilde{B}}(x) = \frac{1}{1 + \left(\frac{x-50}{10}\right)^4}$$

This is illustrated in Fig. 4.3.2.

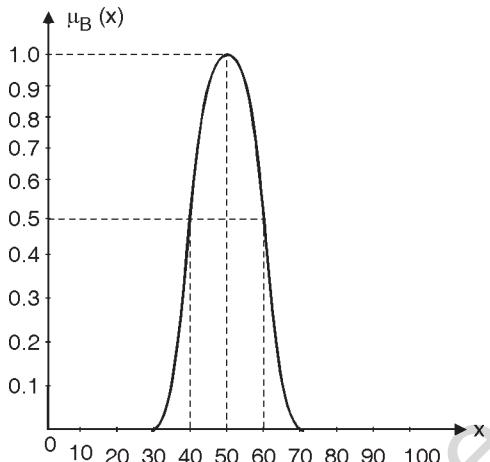


Fig. 4.3.2 : MF for “about 50 years old”

Table 4.3.1 shows $\mu_{\tilde{B}}(x)$ for some value of x .

Table 4.3.1: x and corresponding $\mu_{\tilde{B}}(x)$ for “about 50 years old”

X	40	42	45	48	50	52	53	55	56	58	60
$\mu_{\tilde{B}}(x)$	0.5	0.71	0.94	0.99	1	0.99	0.99	0.94	0.89	0.71	0.5

Note: Construction of a fuzzy set depends on two things:

- 1) The identification of a suitable universe of discourse and
- 2) Specification of an appropriate membership function.

The specification of membership function is **subjective**, which means that the membership functions specified for the same concept by different persons may vary considerably. Therefore, the subjectivity and non randomness of fuzzy sets is the primary difference between the fuzzy sets and probability theory.

4.3.3 Different Notations for Representing Fuzzy Sets

1) Using ordered pairs :

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X\}$$

E.g. Let $X = \{1, 2, 3, \dots, 6\}$

X is available types of houses described by X = “Number of bedrooms in a house” then comfortable house for four persons family is described using ordered pair as,

$$\tilde{A} = \{(1, 0.2), (2, 0.5), (3, 0.8), (4, 1), (5, 0.7), (6, 0.3)\}$$

2) Using membership function :

A fuzzy set can be represented by stating its membership function.

E.g. To represent “real numbers considerably larger than 10”.

We define,

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X\},$$

$$\mu_{\tilde{A}}(x) = \begin{cases} 0 & x \leq 10 \\ \frac{1}{1 + \frac{1}{(x - 10)^2}}; & x > 10 \end{cases}$$

3) Using + Notation :

Fuzzy set for “comfortable type of house for a four person family” may be described as,

$$\tilde{A} = \left\{ \frac{0.2}{1} + \frac{0.5}{2} + \frac{0.8}{3} + \frac{1}{4} + \frac{0.7}{5} + \frac{0.3}{6} \right\}$$

i.e. we define A as

$$\begin{aligned} \tilde{A} &= \mu_{\tilde{A}}(x_1)/x_1 + \mu_{\tilde{A}}(x_2)/x_2 + \dots \\ &= \sum_{i=1}^n \mu_{\tilde{A}}(x_i)/x_i \end{aligned}$$

4) Using Venn diagrams :

Sometimes it is more convenient to give the graph that represents membership function.

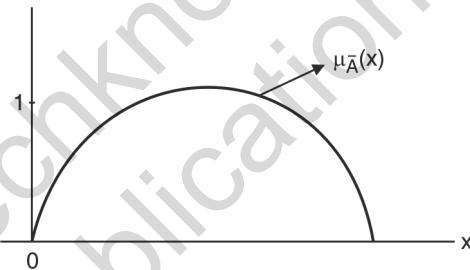


Fig. 4.3.3 : Representation of fuzzy set using Venn diagram

5) Other notations :

$$\tilde{A} = \{(3, 0.1) + (4, 0.3)\} + \dots$$

or

$$\tilde{A} = \left\{ \frac{0.1}{3}, \frac{0.3}{4}, \frac{0.6}{5}, \dots \right\}$$

4.3.4 Linguistic Variables and Linguistic Values

- Suppose that $X = \text{"age"}$. Then, we can define fuzzy sets “young”, “middle aged” and “old” that are characterized by MFs $\mu_{\text{young}}(x)$, $\mu_{\text{middle aged}}(x)$ and $\mu_{\text{old}}(x)$.
- A linguistic variable (“age”) can assume different linguistic values such as “young”, “middle aged” and “old” in this case.
- Note that, the universe of discourse is totally covered by these MFs (MFs for young, middle aged and old) and transition from one MF to another is smooth and gradual.

4.3.5 Important Terminologies related to Fuzzy Sets

MU – Dec. 11

Q. Define supports, core, normality, crossover points and α – cut for fuzzy set.

(Dec. 11, 5 Marks)

1. Support :

A support of a fuzzy set \tilde{A} is the set of all points x in X such that, $\mu_{\tilde{A}}(x) > 0$.

$$\text{Support } (\tilde{A}) = \{x \mid \mu_{\tilde{A}}(x) > 0\}$$

2. Core / Nucleus :

The core of a fuzzy set \tilde{A} is the set of all points x in X such that $\mu_{\tilde{A}}(x) = 1$.

$$\text{Core } \tilde{A} = \{x \mid \mu_{\tilde{A}}(x) = 1\}$$

3. Normality :

A fuzzy set \tilde{A} is normal if its core is non-empty. In other words there must be at least one point $x \in X$ such that $\mu_{\tilde{A}}(x) = 1$.

4. Crossover points :

A cross over point of a fuzzy set \tilde{A} is a point $x \in X$ at which $\mu_{\tilde{A}}(x) = 0.5$.

$$\text{Crossover } (\tilde{A}) = \{x \mid \mu_{\tilde{A}}(x) = 0.5\}$$

5. Fuzzy singleton :

A fuzzy set whose support is a single point in X with $\mu_{\tilde{A}}(x) = 1$ is called a fuzzy singleton.

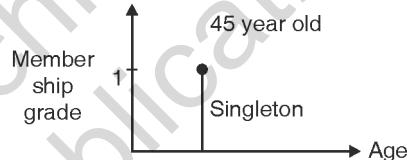


Fig.4.3.4: A fuzzy singleton

Fig. 4.3.5 shows three parameters (core, support and crossover points) of a fuzzy set.

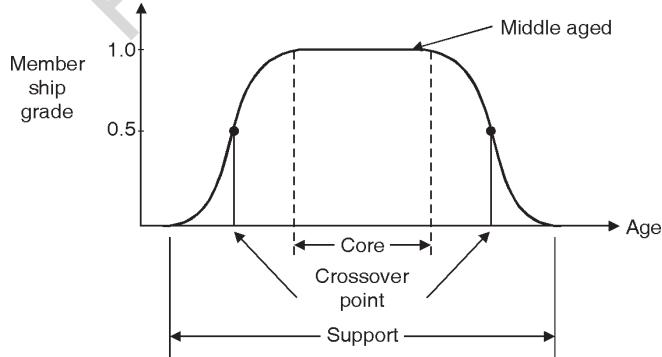


Fig. 4.3.5 : Core, Support and Crossover points of a fuzzy set

6. α -cut :

The α -cut or α -level set of a fuzzy set \tilde{A} is a crisp set defined by,

$$A_\alpha = \{x \mid \mu_{\tilde{A}}(x) \geq \alpha\}$$

7. Strong α -cut / strong α -level set :

Strong α -cut is defined by

$$A'_\alpha = \{x \mid \mu_{\tilde{A}}(x) > \alpha\}$$

Using the above notations, we can express support and core of a fuzzy set A as,

$$\text{Support } (\tilde{A}) = A'_0 \quad \text{Here } \alpha = 0$$

$$\text{Core } (\tilde{A}) = A'_1 \quad \text{Here } \alpha = 1$$

8. Convexity :

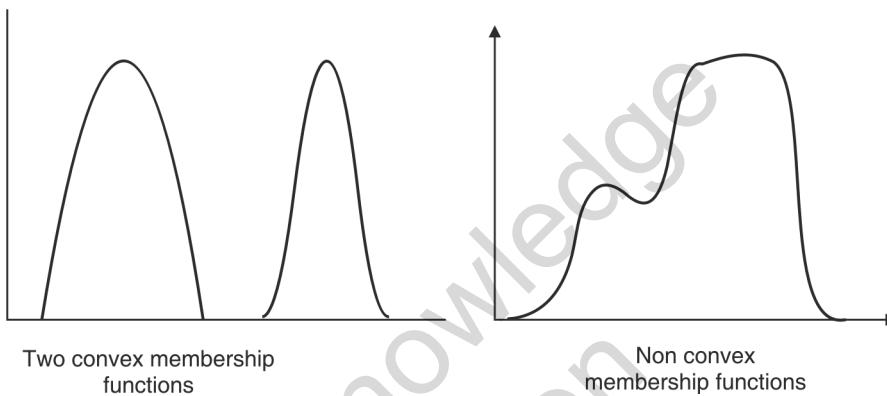


Fig. 4.3.6 : Convex and Non Convex MFs

A fuzzy set \tilde{A} is convex if and only if for any x_1 and $x_2 \in X$ and any $\lambda \in [0, 1]$.

$$\mu_{\tilde{A}}(\lambda x_1 + (1 - \lambda) x_2) \geq \min \{\mu_{\tilde{A}}(x_1), \mu_{\tilde{A}}(x_2)\}$$

or

\tilde{A} is convex if all its α -level sets are convex.

9. Fuzzy numbers :

A fuzzy number \tilde{A} is a fuzzy set in the real line (R) that satisfies the conditions for normality and convexity.

10. Bandwidth of normal and convex fuzzy set :

For a normal and convex fuzzy set, the bandwidth or width is defined as the distance between two unique crossover points.

$$\text{Width } (\tilde{A}) = |x_2 - x_1|$$

$$\text{Where } \mu_{\tilde{A}}(x_1) = \mu_{\tilde{A}}(x_2) = 0.5$$

11. Symmetry :

A fuzzy set \tilde{A} is symmetric if its MF is symmetric around a certain point $x = c$,

$$\mu_{\tilde{A}}(c + x) = \mu_{\tilde{A}}(c - x) \quad \text{for all } x \in X$$

12. Open left, Open right and closed MFs :

A fuzzy set \tilde{A} is open left if,

$$\lim_{x \rightarrow -\infty} \mu_{\tilde{A}}(x) = 1 \text{ and}$$

$$\lim_{x \rightarrow +\infty} \mu_{\tilde{A}}(x) = 0$$

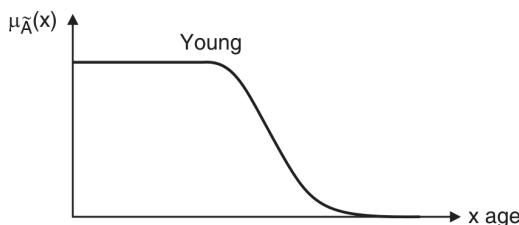


Fig. 4.3.7 : Open Left MF

A fuzzy set \tilde{A} is open right if,

$$\lim_{x \rightarrow -\infty} \mu_{\tilde{A}}(x) = 0 \quad \text{and}$$

$$\lim_{x \rightarrow +\infty} \mu_{\tilde{A}}(x) = 1$$

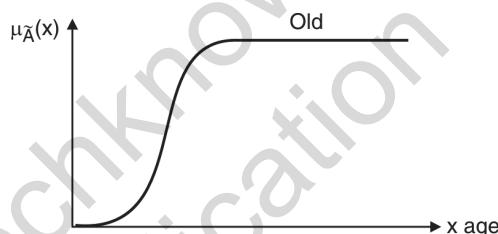


Fig. 4.3.8: Open right MF

A fuzzy set \tilde{A} is closed if,

$$\lim_{x \rightarrow +\infty} \mu_{\tilde{A}}(x) = 0 \quad \text{and}$$

$$\lim_{x \rightarrow -\infty} \mu_{\tilde{A}}(x) = 0$$

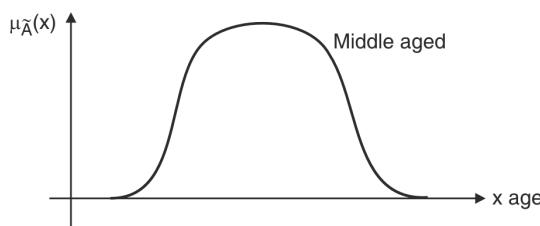


Fig. 4.3.9 : Closed MF

13. Cardinality :

Cardinality of a fuzzy set \tilde{A} is defined as

$$|\tilde{A}| = \sum_{x \in X} \mu_{\tilde{A}}(x)$$

14. Relative cardinality :

Relative cardinality of a fuzzy set \tilde{A} is defined as,

$$\|\tilde{A}\| = \frac{|\tilde{A}|}{|X|}$$

15. Height of a fuzzy set :

The height of a fuzzy set \tilde{A} in X , is equal to the largest membership degree μ_m

$$\text{hgt}(\tilde{A}) = \sup_{x \in X} \mu_{\tilde{A}}(x)$$

If $\text{hgt}(\tilde{A}) = 1$ then, \tilde{A} is normal.

If $\text{hgt}(\tilde{A}) < 1$ then, \tilde{A} is subnormal.

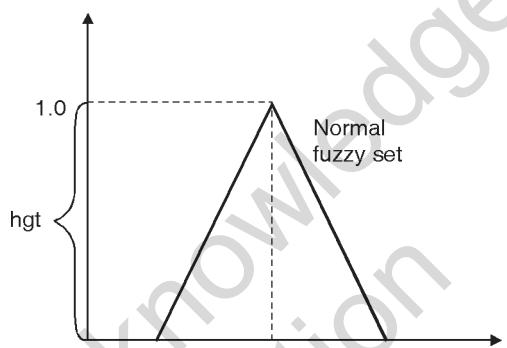


Fig. 4.3.10: Height of a fuzzy set

4.3.6 Properties of Fuzzy Sets

MU – Dec. 12

Q. State the different properties of fuzzy set.

(Dec. 12, 8 Marks)

Fuzzy sets follow the same properties as crisp set except for the law of excluded middle and law of contradiction.

That is, for fuzzy set \tilde{A}

$$\tilde{A} \cup \tilde{\bar{A}} \neq U ; \quad \tilde{A} \cap \tilde{\bar{A}} \neq \emptyset$$

The following are the properties of fuzzy sets,

1. Commutativity :

$$\tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A} \quad \text{and}$$

$$\tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A}$$

2. Associativity :

$$\tilde{A} \cup (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C}$$

$$\tilde{A} \cap (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cap \tilde{C}$$

3. Distributivity :

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C})$$

$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C})$$

4. Identity :

$$\tilde{A} \cup \phi = \tilde{A} ; \quad \tilde{A} \cup U = U$$

$$\tilde{A} \cap \phi = \phi ; \quad \tilde{A} \cap U = \tilde{A}$$

5. Involution :

$$\tilde{\tilde{A}} = \tilde{A}$$

6. Transitivity :

If $\tilde{A} \subset \tilde{B} \subset \tilde{C}$, then $\tilde{A} \subset \tilde{C}$

7. De Morgan's law :

$$\overline{\tilde{A} \cup \tilde{B}} = \tilde{\tilde{A}} \cap \tilde{\tilde{B}}$$

$$\overline{\tilde{A} \cap \tilde{B}} = \tilde{\tilde{A}} \cup \tilde{\tilde{B}}$$

4.3.7 Operations on Fuzzy Sets

1. Containment or Subset :

Fuzzy set \tilde{A} is contained in fuzzy set \tilde{B} if and only if $\mu_{\tilde{A}}(x) \leq \mu_{\tilde{B}}(x)$ for all x .

$$\tilde{A} \subseteq \tilde{B} \Leftrightarrow \mu_{\tilde{A}}(x) \leq \mu_{\tilde{B}}(x)$$

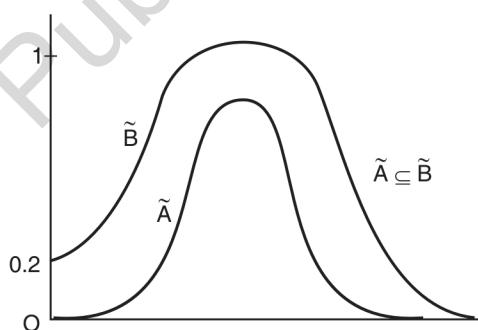


Fig. 4.3.11 : Containment or subset

2. Union (Disjunction) :

A union of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set \tilde{C} , such that whose MF is,

$$\mu_{\tilde{C}}(x) = \max(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

Union of \tilde{A} and \tilde{B} is denoted by $(\tilde{A} \cup \tilde{B})$ or $(\tilde{A} \text{ or } \tilde{B})$

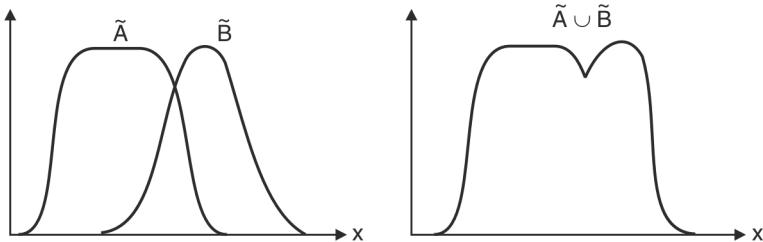


Fig. 4.3.12 : Union of two fuzzy sets

3. Intersection (Conjunction) :

The intersection of two fuzzy sets \tilde{A} and \tilde{B} is a fuzzy set \tilde{C} , such that whose MF is defined as

$$\mu_{\tilde{C}}(x) = \min(\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x))$$

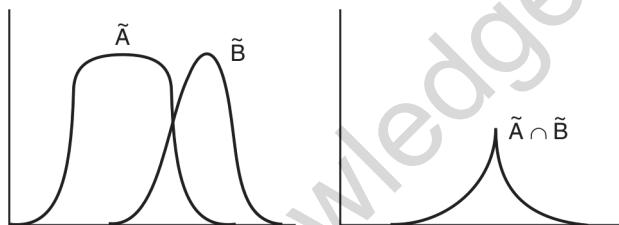


Fig. 4.3.13 : Intersection of two fuzzy sets

4. Complement (Negation) :

The complement of a fuzzy set \tilde{A} , denoted by $\tilde{\tilde{A}}$ is defined as,

$$\mu_{\tilde{\tilde{A}}}(x) = 1 - \mu_{\tilde{A}}(x)$$

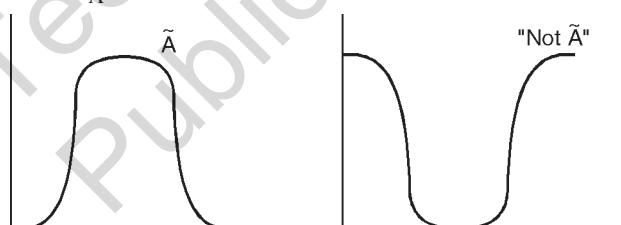


Fig. 4.3.14 : Complement of a fuzzy set

More operations of fuzzy sets

1. Algebraic sum :

$$\mu_{\tilde{A} + \tilde{B}}(x) = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

2. Algebraic product :

$$\mu_{\tilde{A} \cdot \tilde{B}}(x) = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x)$$

3. Bounded sum :

$$\mu_{\tilde{A} \oplus \tilde{B}}(x) = \min[1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)]$$

4. Bounded difference :

$$\mu_{\tilde{A} \ominus \tilde{B}}(x) = \max[0, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)]$$

4.4 Crisp Relation and Fuzzy Relations

4.4.1 Crisp Relation

- An n-ary relation over $M_1, M_2, M_3, \dots, M_n$ is a subset of the Cartesian product $M_1 \times M_2 \times \dots \times M_n$, where $n = 2$, the relation is a subset of the Cartesian product $M_1 \times M_2$. This is called a binary relation from M_1 to M_2 .

Let X and Y be two universes and $X \times Y$ be their Cartesian product.

Then $X \times Y$ can be defined as,

$$X \times Y = \{(x, y) | x \in X, y \in Y\}$$

Every element in X is related to every element in Y .

- We can define characteristic function f that gives the strength of the relationship between each element of X and Y .

$$f_{X \times Y}(x, y) = \begin{cases} 1 & , (x, y) \in X \times Y \\ 0 & , (x, y) \notin X \times Y \end{cases}$$

- We can represent the relation in the form of matrix.

- An n-dimensional relation matrix represents an n-ary relation.

- So, binary relation is represented by 2 dimensional matrices.

Ex : Consider the following two universes,

$$X = \{a, b, c\}, Y = \{1, 2, 3\}$$

- The Cartesian product - $X \times Y$ is,

$$X \times Y = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3), (c, 1), (c, 2), (c, 3)\}$$

From the above set, we may select a subset R , such that

$$R = \{(a, 1), (b, 2), (b, 3), (c, 1), (c, 3)\}$$

Then R can be represented in matrix form as,

R	1	2	3
a	1	0	0
b	0	1	1
c	1	0	1

- The relation between sets X and Y can also be represented as coordinate diagram as shown in Fig. 4.4.1.

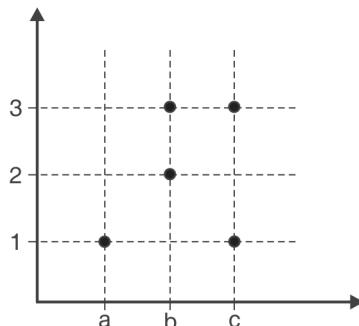


Fig. 4.4.1 : Co-ordinate diagram of a relation

- The relation R can also be expressed by mapping representation as shown in Fig. 4.4.2.

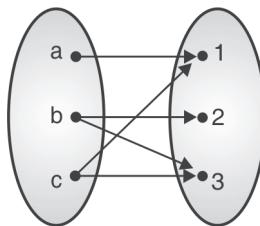


Fig. 4.4.2 : Mapping representation of a relation

- A characteristic function is used to assign values of relationship in the mapping of $X \times Y$ to the binary values and is given by,

$$f_{R(x,y)} = \begin{cases} 1 & , (x,y) \in R \\ 0 & , (x,y) \notin R \end{cases}$$

4.4.1(A) Cardinality of Classical Relation

- Let X and Y be two universe and n elements of X are related to m elements of Y.
- Let the Cardinality of X is η_X and cardinality of Y is η_Y , then the cardinality of relation R between X and Y is,

$$\eta_{X \times Y} = \eta_X \times \eta_Y$$

- The Cardinality of the power set P (X × Y) is given as,

$$\eta_{P(X \times Y)} = 2^{(\eta_X \eta_Y)}$$

4.4.1(B) Operations on Classical Relations

- Let A and B be two separate relations defined on the Cartesian universe $X \times Y$.
- Then the null relation defined as,

$$\phi_A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

And complete relation is defined as,

$$E_A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- The following operations can be performed on two relations A and B.

1. Union

$$A \cup B \rightarrow f_{A \cup B}(x,y) : f_{A \cup B}(x,y) = \max [f_A(x,y), f_B(x,y)]$$

2. Intersection

$$A \cap B \rightarrow f_{A \cap B}(x,y) : f_{A \cap B}(x,y) = \min [f_A(x,y), f_B(x,y)]$$

3. Complement

$$\bar{A} \rightarrow f_{\bar{A}}(x,y) : f_{\bar{A}}(x,y) = 1 - f_A(x,y)$$

4. Containment

$$A \subset B \rightarrow f_A(x,y) : f_B(x,y) \leq f_A(x,y)$$

5. Identity

$\phi \rightarrow \phi_A$ and $X \rightarrow E_A$

4.4.1(C) Properties of Crisp (Classical) Relations

- The properties of classical set such as commutativity, associativity, involution, distributivity and idempotency hold good for classical relation also.
- Also De Morgan's law and excluded middle laws hold good for crisp relations.

4.4.1(D) Composition of Crisp Relations

- Composition is a process of combining two compatible binary relations to get a single relation.
- Let A be a relation that maps elements from universe X to universe Y.

Let B be a relation that maps elements from universe Y to universe Z.

- The two binary relations A and B are said to be compatible if,
 $A \subseteq X \times Y$ and $B \subseteq Y \times Z$
- The composition between the two relations A and B can be denoted as $A \circ B$.

Ex. :

$$\text{Let } X = \{a_1, a_2, a_3\}$$

$$Y = \{b_1, b_2, b_3\}$$

$$Z = \{c_1, c_2, c_3\}$$

Let the relation A and B as,

$$A = X \times Y = \{(a_1, b_1), (a_1, b_2), (a_2, b_2)\}$$

$$B = Y \times Z = \{(b_1, c_1), (b_2, c_3), (b_3, c_2)\}$$

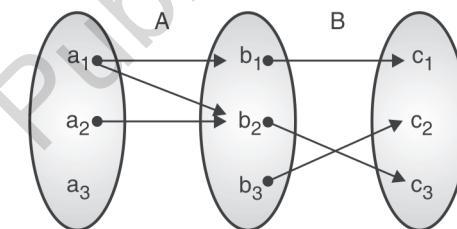


Fig. 4.4.3 : Illustration of relations A and B

Then $A \circ B$ can be written as,

$$A \circ B = \{(a_1, c_1), (a_2, c_3), (a_2, c_3)\}$$

The representation of A and B in matrix form is given as,

$$A = \begin{matrix} & b_1 & b_2 & b_3 \\ a_1 & 1 & 1 & 0 \\ a_2 & 0 & 1 & 0 \\ a_3 & 0 & 0 & 0 \end{matrix}; \quad B = \begin{matrix} & c_1 & c_2 & c_3 \\ b_1 & 1 & 0 & 0 \\ b_2 & 0 & 0 & 1 \\ b_3 & 0 & 1 & 0 \end{matrix}$$



Then, composition $A \circ B$ is represented as,

$$A \circ B = \begin{matrix} & c_1 & c_2 & c_3 \\ a_1 & \left[\begin{array}{ccc} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array} \right] \\ a_2 & \\ a_3 & \end{matrix}$$

There are two types of composition operations:

- 1. Max-min composition
- 2. Max-product composition

1. The max-min composition is defined as,

$$\begin{aligned} T &= A \circ B \\ f_T(x, z) &= \vee_{y \in Y} [f_A(x, y) \wedge f_B(y, z)] \end{aligned}$$

2. The max-product composition is defined as,

$$\begin{aligned} T &= A \circ B \\ f_T(x, z) &= \vee_{y \in Y} [f_A(x, y) \cdot f_B(y, z)] \end{aligned}$$

Note : In the above equations \vee represents max operation, \wedge represents min operation and \cdot represents product operation.

Properties of composition operation

1. Associative : $(A \circ B) \circ C = A \circ (B \circ C)$
2. Commutative : $A \circ B \neq B \circ A$
3. Inverse : $(A \circ B)^{-1} = A^{-1} \circ B^{-1}$

4.4.2 Fuzzy Relations

In general, a relation can be considered as a set of tuples, where a tuple is an ordered pair. Similarly, a fuzzy relation is a fuzzy set of tuples i.e. each tuple has a membership degree between 0 and 1.

Definition:

Let U and V be continuous universe, and $\mu_R : U \times V \rightarrow [0, 1]$ then,

$$R = \int_{U \times V} \mu_R(u, v) / (u, v)$$

Is a binary fuzzy relation on $U \times V$.

If U and V are discrete universe, then

$$R = \sum_{U \times V} \mu_R(u, v) / (u, v)$$



We can express fuzzy relation $R = U \times V$ in matrix form as,

$$R = \begin{bmatrix} \mu_R(u_1, v_1) & \mu_R(u_1, v_2) & \dots & \mu_R(u_1, v_n) \\ \mu_R(u_2, v_1) & \mu_R(u_2, v_2) & \dots & \mu_R(u_2, v_n) \\ \vdots & & & \\ \mu_R(u_m, v_1) & \mu_R(u_m, v_2) & \dots & \mu_R(u_m, v_n) \end{bmatrix}$$

Where $U = \{u_1, u_2, u_3, \dots, u_m\}$ and

$V = \{v_1, v_2, v_3, \dots, v_n\}$ are universe of discourse.

Ex. 4.4.1 : Given universe of discourse

$U = \{1, 2, 3\}$ form a relation R where "x is approximately equal to y"

Then Relation R can be defined as,

$$R = \left\{ \frac{1}{(1,1)} + \frac{1}{(2,2)} + \frac{1}{(3,3)} + \frac{0.8}{(1,2)} + \frac{0.3}{(1,3)} + \frac{0.8}{(2,1)} + \frac{0.8}{(2,3)} + \frac{0.3}{(3,1)} + \frac{0.8}{(3,2)} \right\}$$

Soln. : The membership function μ_R of this relation can be described as,

$$\begin{aligned} 1 &\text{ when } x = y \\ &\mu_R(x, y) = 0.8 \\ 0.3 &\text{ when } |x - y| = 2 \end{aligned} \quad \left\{ \begin{array}{l} \text{when } |x - y| = 1 \\ \dots \end{array} \right.$$

The matrix notation is,

		Y		
		1	2	3
X	1	1	0.8	0.3
	2	0.8	1	0.8
	3	0.3	0.8	1

N - ary fuzzy relation

It is possible to define n-ary fuzzy relations as fuzzy set of n-tuples. In general it is a relation of pairs.

$\mu_R(x_1, \dots, x_n)/(x_1, \dots, x_n);$

4.4.2(A) Operations on Fuzzy Relation

MU - May 13

Q. Explain cylindrical extension and projection operations on fuzzy relation with example.

(May 13, 5 Marks)

Fuzzy relations are very important in fuzzy controller because they can describe interaction between variables.

Four types of operations can be performed on fuzzy relation.

- | | |
|------------------|---------------------------|
| (1) Intersection | (2) Union |
| (3) Projection | (4) Cylindrical extension |

1. Intersection

– Let R and S be binary relations defined on $X \times Y$. The intersection of R and S is defined by,

$$\forall (x, y) \in X \times Y : \mu_{R \cap S}(x, y) = \min(\mu_R(x, y), \mu_S(x, y))$$

– Instead of the minimum, any T - Norm can be used.



2. Union

- The union of R and S is defined as,

$$\forall (x, y) \in X \times Y : \mu_{R \cup S}(x, y) = \max(\mu_R(x, y), \mu_S(x, y)).$$

Instead of maximum, any S – norm can be used.

- Given two relations R and S

	y_1	y_2	y_3
x_1	0.3	0.2	0.1
x_2	0.4	0.6	0.1
x_3	0.2	0.3	0.5

	y_1	y_2	y_3
x_1	0.4	0	0.1
x_2	1	0.2	0.8
x_3	0.3	0.2	0.4

- Then, using max operation.

$$R \cup S =$$

0.4	0.2	0.1
1	0.6	0.8
0.3	0.3	0.5

- Suppose a simple s - norm

$$S(a, b) = a + b - a \cdot b \text{ is used then,}$$

$$R \cup S =$$

0.58	0.2	0.19
1	0.68	0.84
0.44	0.44	0.7

- This operation is more optimistic than the max operation. All the membership degrees are at least as high as in the max operation.

Now, using min operation

$$R \cap S =$$

0.3	0	0.1
0.4	0.2	0.1
0.2	0.2	0.4

- Suppose a simple T-norm $T(a, b) = \frac{a \cdot b}{a + b - a \cdot b}$ is used then,

$$R \cap S =$$

0.20	0	0.1
0.4	0.17	0.10
0.13	0.13	0.28

- The above operation is more optimistic than the min operation. All the membership degrees are less than in the min operation.

3. Projection

- The projection relation brings a ternary relation back to a binary relation, or a binary relation to a fuzzy set, or a fuzzy set to a single crisp value.

Ex. Consider the relation R as given below.

	y_1	y_2	y_3	y_4
x_1	0.8	1	0.1	0.7
x_2	0	0.8	0	0
x_3	0.9	1	0.7	0.8

Then the projection on X means that

- x_1 is assigned the maximum of the first row.
- x_2 is assigned the maximum of the second row.
- x_3 is assigned the maximum of the third row.

Thus,

$$\text{Proj. } R \text{ on } X = \frac{1}{x_1} + \frac{0.8}{x_2} + \frac{1}{x_3}$$

Similarly

$$\text{Proj. } R \text{ on } Y = \frac{0.9}{y_1} + \frac{1}{y_2} + \frac{0.7}{y_3} + \frac{0.8}{y_4}$$

4. Cylindrical Extension

- The projection operation is almost always used in combination with cylindrical extension.

Cylindrical extension is more or less opposite of projection. It converts fuzzy set to a relation.

Ex : Consider a fuzzy set,

$$A = \text{proj. of } R \text{ on } X = 1/x_1 + 0.8/x_2 + 1/x_3.$$

- Its cylindrical extension on the domain $X \times Y$ is

$$\text{ce}(A) =$$

	y_1	y_2	y_3	y_4
x_1	1	1	1	1
x_2	0.8	0.8	0.8	0.8
x_3	1	1	1	1

Consider the fuzzy set

$$B = \text{proj of } R \text{ on } X = \frac{0.9}{y_1} + \frac{0.8}{y_2} + \frac{0.7}{y_3} + \frac{0.8}{y_4}$$

$$\text{ce}(B) =$$

	y_1	y_2	y_3	y_4
x_1	0.9	0.8	0.7	0.8
x_2	0.9	0.8	0.7	0.8
x_3	0.9	0.8	0.7	0.8



4.4.2(B) Properties of Fuzzy Relations

Let R , S and T be fuzzy relations defined on the universe $X \times Y$. Then, the properties of fuzzy relations are stated below :

1. Commutativity

$$R \cup S = S \cup R$$

$$R \cap S = S \cap R$$

2. Associativity

$$R \cup (S \cup T) = (R \cup S) \cup T$$

$$R \cap (S \cap T) = (R \cap S) \cap T$$

3. Distributivity

$$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$$

$$R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$$

4. Idempotency

$$R \cup R = R$$

$$R \cap R = R$$

5. Identity

$$R \cup \phi_R = R, \quad R \cap \phi_R = \phi_R$$

$$R \cup E_R = E_R, \quad R \cap E_R = R$$

Where ϕ_R and E_R are null relation (null matrix) and complete relation (unit matrix of all 1s) respectively.

6. Involution

$$\bar{\bar{R}} = R$$

7. De-Morgan's law

$$\overline{R \cap S} = \bar{R} \cup \bar{S}$$

$$\overline{R \cup S} = \bar{R} \cap \bar{S}$$

8. Law of excluded middle and law of contradiction are not satisfied.

$$\text{i.e. } R \cup \bar{R} \neq E_R$$

$$\text{and } R \cap \bar{R} \neq \phi_R$$

4.4.2(C) Fuzzy Composition

Composition operation can be used to combine two fuzzy relations in different product spaces.

There are two compositions that are used commonly.

- 1. Max - min composition
- 2. Max - product composition

1. Max - Min Composition

- Let R_1 be a fuzzy relation defined on $X \times Y$.
And R_2 be a fuzzy relation defined on $Y \times Z$.
- Then the max - min composition of two fuzzy relations R_1 and R_2 is denoted by $R_1 \circ R_2$ and defined as,

$$R_1 \circ R_2 = \{[(x, z), \max_{y \in Y} (\min(\mu_{R1}(x, y), \mu_{R2}(y, z)))] | x \in X, y \in Y, z \in Z\}$$

OR $\mu_{R1 \circ R2}(x, z) = \max_{y \in Y} \{\min(\mu_{R1}(x, y), \mu_{R2}(y, z))\}$

2. Max - Product Composition

The max - product composition is defined as,

$$R_1 \circ R_2 = \{[(x, z), \max_{y \in Y} (\mu_{R1}(x, y) \cdot \mu_{R2}(y, z))] | x \in X, y \in Y, z \in Z\}$$

OR $\mu_{R1 \circ R2}(x, z) = \max_{y \in Y} \{\mu_{R1}(x, y) \cdot \mu_{R2}(y, z)\}$

The following are the properties of fuzzy composition. Assuming R, S and T are binary relations defined on $X \times Y, Y \times Z$ and $Z \times W$ respectively.

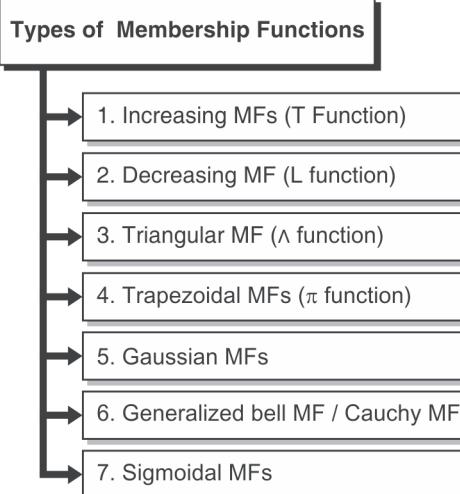
1. Associativity $\rightarrow R \circ (S \circ T) \Rightarrow (R \circ S) \circ T$
2. Monotonicity $\rightarrow S \subseteq T \Rightarrow R \circ S \subseteq R \circ T$
3. Distributivity $\rightarrow R \circ (S \cup T) \Rightarrow (R \circ S) \cup (R \circ T)$
4. Inverse $\rightarrow (R \circ S)^{-1} = S^{-1} \circ R^{-1}$

4.5 Membership Functions

MU - May 12, Dec. 12, Dec.13, Dec.14

Q. Explain the different Fuzzy membership function.	(Dec. 12, Dec. 14, 5/8 Marks)
Q. Explain standard fuzzy membership functions.	(May 12, Dec. 13, 8 Marks)

- One way to represent a fuzzy set is by stating its Membership Function (MF). MFs can be represented using any mathematical equation as per requirement or using one of the standard MFs available.
- There are several different standard MFs available.



1. Increasing MFs (T Function)

An increasing MF is specified by two parameters (a, b) as follows:

$$T(x ; a, b) = \begin{cases} 0 & ; x \leq a \\ (x - a)/(b - a) & ; a \leq x \leq b \\ 1 & ; x \geq b \end{cases}$$

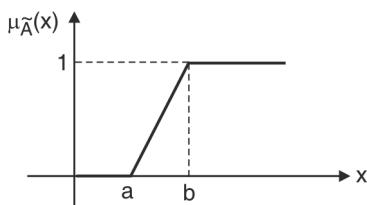


Fig. 4.5.1: Increasing MF

2. Decreasing MF (L function)

A decreasing MF is specified by two parameters (a, b) as follows :

$$L(x ; a, b) = \begin{cases} 1 & ; x \leq a \\ (b - x)/(b - a) & ; a \leq x \leq b \\ 0 & ; x \geq b \end{cases}$$

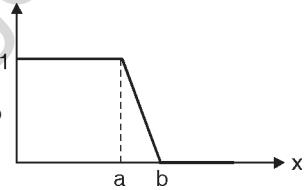


Fig. 4.5.2 : Decreasing MF

3. Triangular MF (\wedge function)

- A triangular MF is specified by three parameters (a, b, c) as follows:

$$\wedge(x ; a, b, c) = \begin{cases} 0 & ; x \leq a \\ (x - a)/(b - a) & ; a \leq x \leq b \\ (c - x)/(c - b) & ; b \leq x \leq c \\ 0 & ; x \geq c \end{cases}$$

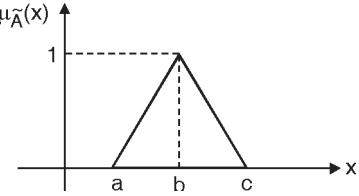


Fig. 4.5.3 : Triangular MF

4. Trapezoidal MFs (π function)

- A trapezoidal MF is specified by four parameters (a, b, c, d) as follows :

$$\text{Trapezoid}(x ; a, b, c, d) = \begin{cases} 0 & ; x \leq a \\ (x - a)/(b - a) & ; a \leq x \leq b \\ 1 & ; b \leq x \leq c \\ (d - x)/(d - c) & ; c \leq x \leq d \\ 0 & ; x \geq d \end{cases}$$

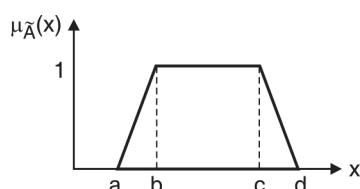


Fig. 4.5.4 : Trapezoid MF

- An alternative expression using min and max can be given as,

$$\text{trapezoid}(x ; a, b, c, d) = \max \left(\min \left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c} \right), 0 \right)$$

- The parameters $\{a, b, c, d\}$ (with $a < b < c < d$) determine the x coordinates of the four corners of the trapezoidal MF.
- Here, if $b = c$ then trapezoidal MF reduces to triangle MF.
- Since two MFs triangular and trapezoidal are composed of straight line segment, they are not smooth at the corner points specified by the parameters. However due to the simple formulae and computational efficiency, they are used extensively.
- Some **smooth** and **non-linear** MFs (Gaussian and Generalized Bell) are discussed below :

5. Gaussian MFs

A Gaussian MF is specified by two parameters $\{c, \sigma\}$.

$$\text{Gaussian}(x ; c, \sigma) = e^{-\frac{1}{2} \left(\frac{x-c}{\sigma} \right)^2}$$

c represents MFs center and

σ determines MFs width.

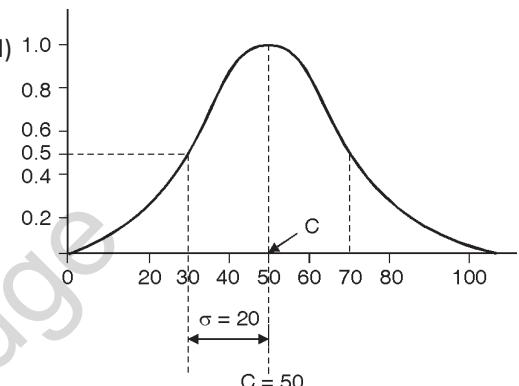


Fig. 4.5.5 : Gaussian $(x ; 50, 20)$ MF

6. Generalized bell MF / Cauchy MF

- A generalized bell MF (or bell MF) is specified by three parameters $\{a, b, c\}$.

$$\text{bell}(x ; a, b, c) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}}$$

- A desired generalized bell MF can be obtained by a proper selection of the parameters a, b, c .

c specifies the center of a bell MF

a specifies the width of a bell MF

and b determines the slope at the crossover points.

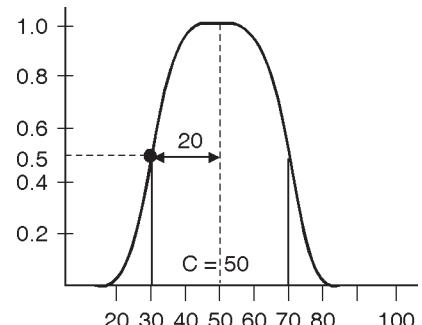
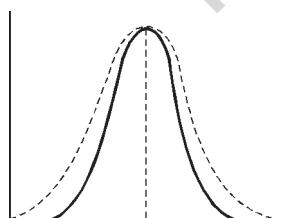
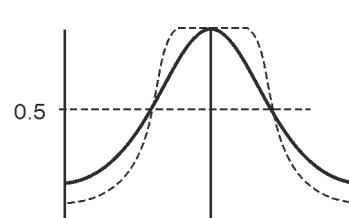


Fig. 4.5.6: Bell $(x ; 20, 4, 50)$

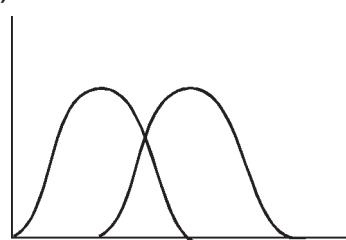
- Fig. 4.5.7 illustrates the effect of changing these parameters on the shape of the curve.



Changing 'a' (width)



Changing 'b' (slope)



Changing 'c' (centre)

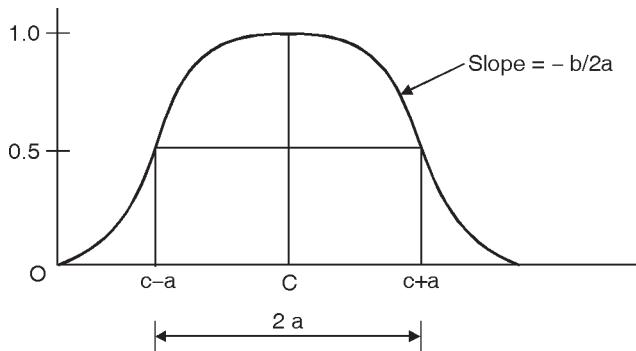


Fig. 4.5.7 : Effect of change of different parameters in Bell MF

- The bell MF is direct generalization of Cauchy distribution used in probability theory; so it is also referred to as the Cauchy MF. The bell MF has more parameter than Gaussian MF, so it has more degree of freedom to adjust the steepness at the crossover point.
- Although Gaussian and bell MFs achieve smoothness, they are unable to specify asymmetric MFs.

Asymmetric MFs

Asymmetric and close MFs can be achieved by using either the absolute difference or the product of two sigmoidal functions.

Sigmoidal MFs

- A sigmoidal MF is defined by,

$$\text{sig}(x ; a, c) = \frac{1}{1 + \exp[-a(x - c)]}$$

Where, a controls the slope at the crossover point $x = c$.

- Depending on the sign of the parameter a , a sigmoidal MF is open right or open left and thus is appropriate for representing concepts such as “very large” or “very negative”.
- They are widely used as the activation function in artificial neural networks.

4.6 Fuzzy Logic

4.6.1 Fuzzy Logic Basics

- Fuzzy logic is an extension of Boolean logic.
- In Boolean logic we express everything in the form of 1 or 0 i.e. true or false respectively.
- Fuzzy logic handles the concept of partial truth, where the range of truth value is in between completely true and completely false, that is in between 0 and 1. In other words , Fuzzy logic can be considered as multi-valued logic.
- In other words, fuzzy logic replaces Boolean truth-values with some degree of truth.
- This degree of truth is used to capture the imprecise modes of reasoning.
- The basic elements of fuzzy logic are fuzzy sets, linguistic variables and fuzzy rules.
- Usually in mathematics, variables take numerical values whereas fuzzy logic allows the non-numeric linguistic variables to be used to form the expression of rules and facts.

- The linguistic variables are words, specifically adjectives like “small,” “little,” “medium,” “high,” and so on. A fuzzy set is a collection of couples of elements.

Linguistic Variables and Linguistic Values

- A **linguistic variable** is a **variable** whose values are words or sentences in a natural or artificial language.
- Consider the variable $X = \text{“age”}$.
- A linguistic variable (“age”) can assume different linguistic values such as “young”, “Middle aged”, “Mature” and “old” in this case.
- Then, ‘age’ can be considered as a linguistic variable whose values can be “young”, “Middle aged” “Mature” and “old” and these values can be characterized by MFs $\mu_{\text{young}}(x)$, $\mu_{\text{middle aged}}(x)$, $\mu_{\text{mature}}(x)$ and $\mu_{\text{old}}(x)$.
- The universe of discourse is totally covered by these MFs (MFs for young, middle aged and old) and transition from one MF to another is smooth and gradual.

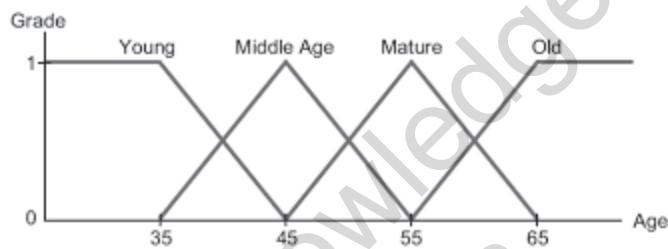


Fig 4.6.1 : Linguistic variable “age” as membership functions

4.6.2 Fuzzy Rules and Fuzzy Reasoning

- Fuzzy inference is the process of obtaining a new knowledge from an existing knowledge.
- To perform inference, knowledge must be represented in some form.
- Fuzzy logic uses IF –THEN rules to represent its knowledgebase.
- The basic rule of inference in traditional two-valued logic is **modus ponens**, according to which, we can infer the truth of a proposition B from the truth of A and the implication $A \rightarrow B$.

Ex. : If A is identified with - “the tomato is red” and B with “the tomato is ripe” then if it is true that “the tomato is red” it is also true that “the tomato is ripe”.

i.e.	Premise 1 (fact)	X is A
	Premise 2 (rule)	if X is A then Y is B
Consequence (conclusion) : Y is B		

- However, in most of the human reasoning, modus ponens is employed in an approximate manner.

For e.g. : if we have the same implication rule, “if the tomato is red, then it is ripe” and we know that,

“the tomato is more or less red” then we may infer that

“the tomato is more or less ripe”

.	Premise 1 (fact)	X is A'
	Premise 2 (rule)	if X is A then Y is B
	Conclusion	Y is B'

Where A' is close to A and

B' is close to B

- When A, B, A' and B' are fuzzy sets of appropriate universes, the inference procedure is called "**approximate reasoning**" or **fuzzy reasoning**, it is also called Generalized Modus Ponens (GMP).

Definition : Approximate reasoning / fuzzy reasoning

Let A, A' and B be fuzzy sets of X, X and Y respectively. Assume that the fuzzy implication $A \rightarrow B$ is expressed as a fuzzy relation R on $X \times Y$. Then the fuzzy set B induced by "x is A' " (fact) and the fuzzy rule "if x is A then y is B " is defined by,

$$\mu_{B'}(y) = \max \min [\mu_{A'}(x), \mu_R(x, y)] = V_x [\mu_{A'}(x) \wedge \mu_R(x, y)]$$

or $\beta' = A' \circ (A \rightarrow B)$

if x is A , y is B

a. Single rule with single antecedent

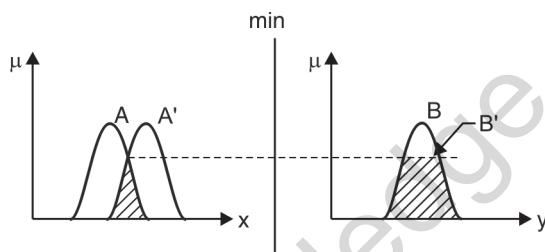


Fig. 4.6.2 : Graphic interpretation of GMP using Mamdani's fuzzy implication and max-min composition

Here $\mu_{B'}$ can be defined as ;

$$\begin{aligned} \mu_{B'}(y) &= [V_x (\mu_{A'}(x) \wedge \mu_A(x))] \wedge \mu_B(y) \\ &= w \wedge \mu_B(y) \end{aligned}$$

Thus, we first find the degree of match w which is the maximum of $\mu_{A'}(x) \wedge \mu_A(x)$.

Then the MF of resulting B' is equal to the MF of B clipped by w .

b. Single rule with multiple antecedent

A fuzzy if then rule with two antecedents can be written as,

"if x is A and y is B then z is C "

Premise 1 (fact)	: x is A' and y is B'
Premise 2 (rule)	: If x is A and y is B then z is C
Consequence	: z is C'

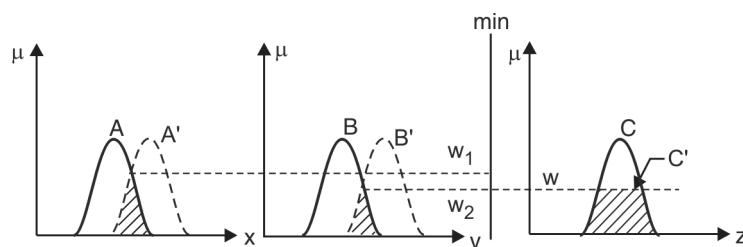


Fig. 4.6.3 : Approximate reasoning for multiple antecedents

$$\mu_{C'}(z) = \underbrace{\{V_x [\mu_{A'}(x) \wedge \mu_A(x)]\}}_{w_1} \wedge$$

$$\{ \underbrace{V_y [\mu_{B'}(y) \wedge \mu_B(y)]}_{w_2} \} \wedge \mu_C(z)$$

$$= (\underbrace{w_1 \wedge w_2}_{\text{firing strength}}) \wedge \mu_C(z)$$

- When w_1 and w_2 are the maxima of the MFs of $A \cap A'$ and $B \cap B'$ respectively.
 - Thus, w_1 denotes the degree of compatibility between A and A' , similarly for w_2 .
 - Since the antecedent parts of the fuzzy rule is constructed using **and** connective, $w_1 \wedge w_2$ is called **firing strength or degree of fulfilment of the fuzzy rule**.
 - The firing strength represents the degree to which the antecedent part of the rule is satisfied.
 - The MF of the resulting C' is equal to the MF of clipped by the firing strength w (when $w = w_1 \wedge w_2$)

c. Multiple rules with multiple antecedents

- The GMP problem for multiple rules with multiple antecedents can be written as,

Premise 1 (fact) : x is A' and y is B'

Premise 2 (rule 1) : If x is A_1 and y is B_1 then z is C_1

Premise 3 (rule 2) : If x is A₂ and y is B₂ then z is C₂

Consequence : z is C'

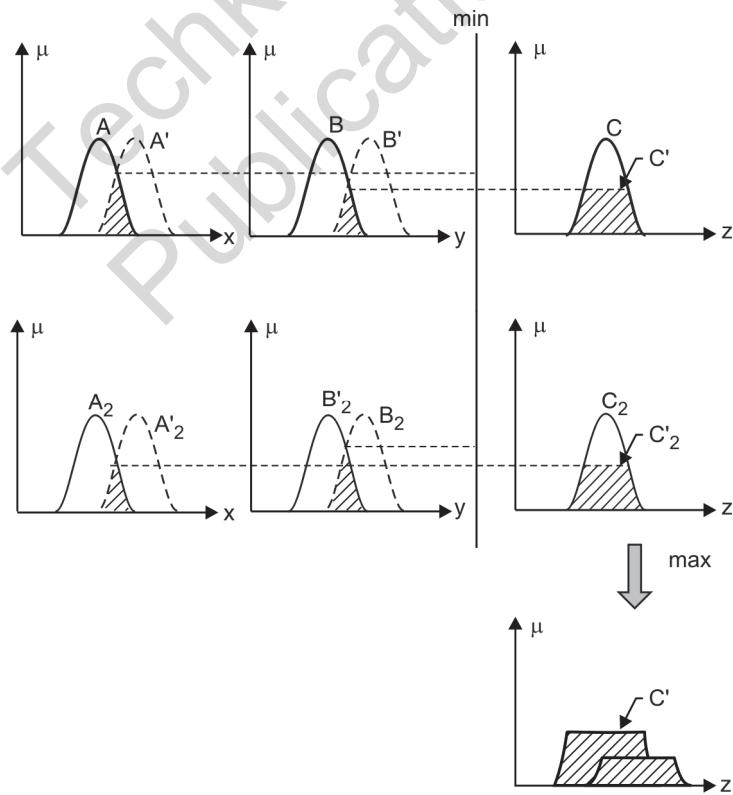


Fig. 4.6.4 : Fuzzy reasoning for multiple rules with multiple antecedents

- Here C_1' and C_2' are the inferred fuzzy sets for rule 1 and rule 2 respectively.
- When a given fuzzy rule assumes the form “if x is A or y is B ” then firing strength is given as the maximum of degree of match on the antecedent part for a given condition.

Ex.

If x is A_1 or y is B_1 then z is C_1 .

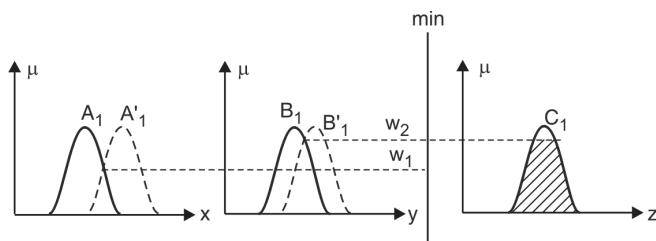


Fig. 4.6.5

- In the above example, because two antecedents are connected using **or**, we take maximum of w_1 and w_2 as a firing strength.
- Since $w_2 > w_1$, we take w_2 as a firing strength and then we apply **min** implication operator on the output MF C_1 .

4.7 Fuzzy Inference Systems

MU - May 12, May 13, Dec. 15

Q. Explain the three types of Fuzzy Inference Systems in detail.	(May 12, 10 Marks)
Q. Compare Mamdani and Sugeno fuzzy models.	(May 13, 10 Marks)
Q. Write short note on Fuzzy inference system.	(Dec. 15, 10 Marks)

- Fuzzy Inference System is the key unit of a fuzzy logic system. Fuzzy inference (reasoning) is the actual process of mapping from a given input to an output using fuzzy logic.
- It uses the “IF...THEN” rules along with connectors “OR” or “AND” for drawing essential decision rules.

4.7.1 Construction and Working Principle of FIS

- Fig. 4.7.1(a) shows the block diagram of general fuzzy inference system.

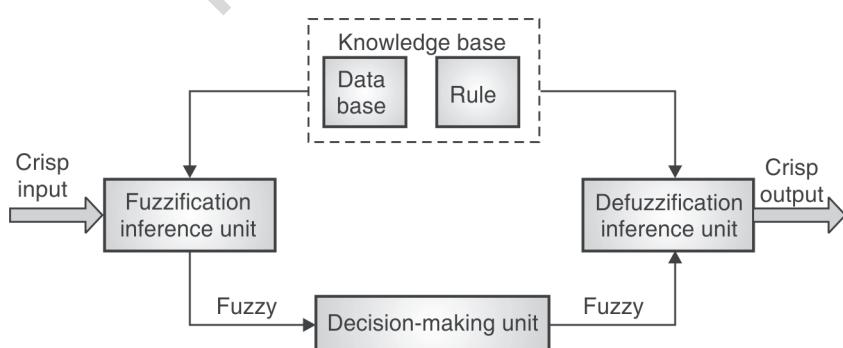


Fig. 4.7.1 (a) : Block diagram : Fuzzy inference system

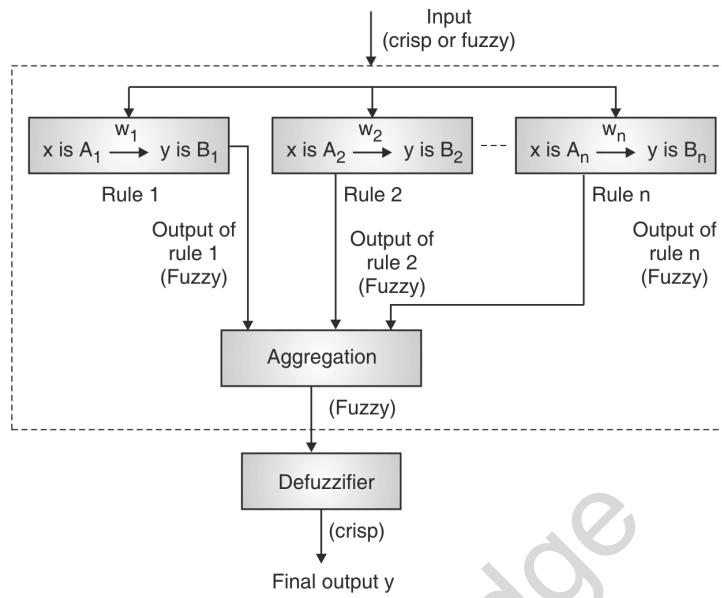


Fig 4.7.1(b) : Fuzzy Inference using If-Then rules

- As shown in Fig. 4.7.1(a), FIS involves five important modules.

1. Fuzzification Inference Unit (FU)
2. Decision making / Inferencing Unit
3. Rule Base
4. Data Base
5. Defuzzification Inference Unit (DU)

1. Fuzzification Inference Unit

This block performs a fuzzification which converts a crisp input in to a fuzzy set. Here we need to decide the proper fuzzification strategy.

2. Decision making/Inferencing Unit

- The basic function of the inference unit is to compute the overall value of the control output variable based on the individual contribution of each rule in the rule base.
- The output of the fuzzification module representing the crisp input is matched to each rule-antecedent.
- The degree of match of each rule is established. Based on this degree of match, the value of the control output variable in the rule-consequent is modified. The result is, we get the “clipped” fuzzy set representing the control output variable.
- The set of all clipped control output values of the matched rules represent the overall fuzzy value of control output.

3. A rule base

It contains a number of fuzzy IF-THEN rules.

4. A database

- Data Base defines the membership functions of the fuzzy sets used in the fuzzy rules.



- The information in the database includes:
 - Fuzzy Membership Functions for the input and output control variables
 - The physical domains of the actual problems and their normalized values along with the scaling factors.

5. Defuzzification Unit

- It performs defuzzification which converts the overall control output into a single crisp value.
- The **rule base** and the **database** are jointly referred to as the **knowledge base**.

Working

The input to the FIS may be a Fuzzy or crisp value.

1. Fuzzification Unit converts the crisp input into fuzzy input by using any of the fuzzification methods.
2. The next, rule base is formed. Database and rule base are collectively called knowledgebase.
3. Finally, defuzzification process is carried out to produce crisp output.

Methods of FIS

- The most important two types of fuzzy inference method are :

- 1) Mamdani FIS
- 2) Sugeno FIS

- Mamdani fuzzy inference is the most commonly seen inference method. This method was introduced by Mamdani and Assilian (1975).
- Another well-known inference method is the so- called Sugeno or Takagi–Sugeno–Kang method of fuzzy inference process. This method was introduced by Sugeno (1985). This method is also called as TS method.
- The main difference between the two methods lies in the consequent of fuzzy rules.

1. Mamdani FIS

- Mamdani FIS was proposed by Ebahim Mamdani in the year 1975 to control a steam engine and boiler combination.
- To compute the output of this FIS given the inputs, six steps has to be followed.
 1. Determining a set of fuzzy rules.
 2. Fuzzifying the inputs using the input membership functions.
 3. Combining the fuzzified inputs according to the fuzzy rules to establish a rule strength (Fuzzy Operations).
 4. Finding the consequence of the rule by combining the rule strength and the output membership function (implication).
 5. Combining the consequences to get an output distribution (aggregation).
 6. Defuzzifying the output distribution (this step is only if a crisp output (class) is needed).

Fuzzy Rule Composition in Mamdani Model

- In Mamdani FIS, The fuzzy rules are formed using IF-THEN statements and AND/OR connectives.
- The consequent of the rule can be obtained in two steps.
 - By computing the strength of each rule
 - By clipping the output membership function at the rule strength.

- The outputs of all the fuzzy rules are then combined to obtain the aggregated fuzzy output. Finally, defuzzification is applied on to the aggregated fuzzy output to obtain a crisp output value.
- Consider two inputs, two rule Mamdani fuzzy inference system.
- Assume two inputs are crisp value x and y .
- Assume the following two rules :
 - Rule 1 :** if x is A_1 and y is B_1 then z is C_1
 - Rule 2 :** if x is A_2 and y is B_2 then z is C_2
- Fig. 4.7.2 (a) shows Mamdani fuzzy inference system using **min – max** decomposition.
- Fig. 4.7.2 (a) illustrates a procedure of deriving overall output z when presented with two crisp inputs x and y . In the above Mamdani inference system, we have used **min** as T - norm and **max** as T - conorm operators.
- The T-norm operator is used for inferencing antecedent part of the rule. And co-norm operator used to aggregate outputs resulting form of each rule.
- Mamdani model also supports **max - product** composition to derive overall output z . Here the **algebraic product** is used as T-norm operator and **max** is used as T-conorm operator.

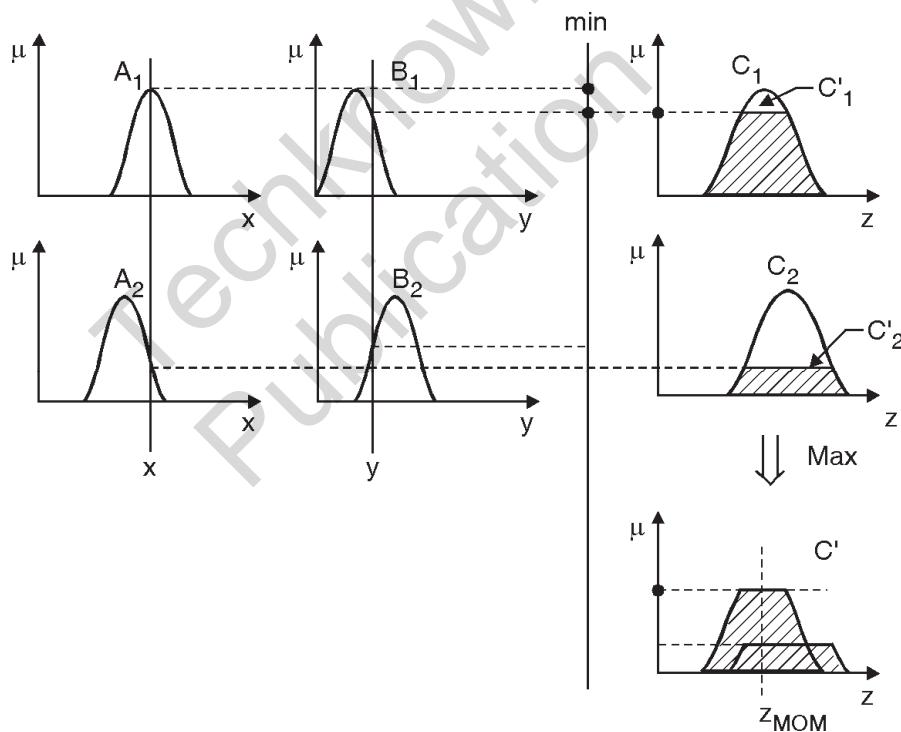


Fig. 4.7.2 (a) : Mamdani fuzzy inference systems using max - min decomposition

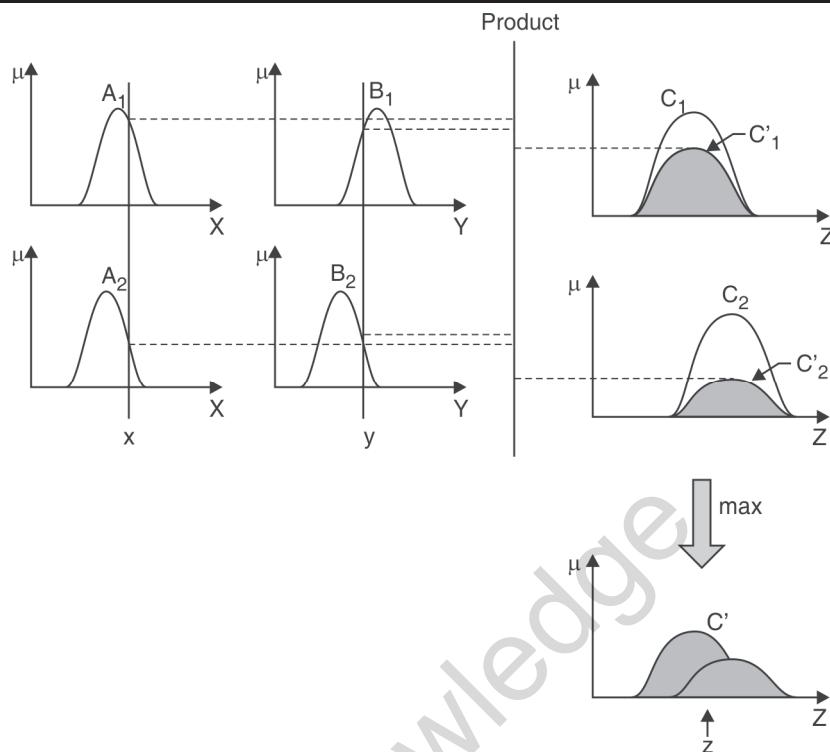


Fig. 4.7.2 (b): Mamdani fuzzy inference systems using max - product decomposition

2. Takagi-Sugeno-Kang (TSK) FIS

– Takagi - Sugeno FIS was proposed by Takagi, Sugeno and Kang in the year 1985.

– A typical fuzzy rule in TSK model has the form ,

IF x is A and y is B then z = f(x,y)

Where,

x, y and z are linguistic variables.

A and B are fuzzy sets in the antecedent part of the rule.

Z = f(x, y) is a crisp function in the consequent part of the rule.

Usually f(x, y) is a polynomial in the input variables x and y.

Fuzzy Inference Process

The fuzzy inference process under Takagi-Sugeno Fuzzy Model (TS Method) works in the following way :

Step 1 : Fuzzifying the inputs

Here, the inputs of the system are made fuzzy.

Step 2 : Applying the fuzzy operator

In this step, the fuzzy operators must be applied to get the output.

First order Sugeno fuzzy model

When f(x,y) is a first order polynomial (e.g. z = ax + by + c) the resulting FIS is called , first order Sugeno fuzzy model.

Zero Order fuzzy model

– In zero order fuzzy model, the output z is a constant (i.e. a = b = 0).

– The typical form of the rule in zero order FIS is

IF x is A and y is B then z = c

Where c is a constant

- In this case the output of each fuzzy rules is a constant and hence the overall output is obtained via weighted average method.
- The output level z_i of each rule is weighted by the firing strength w_i of the rule.

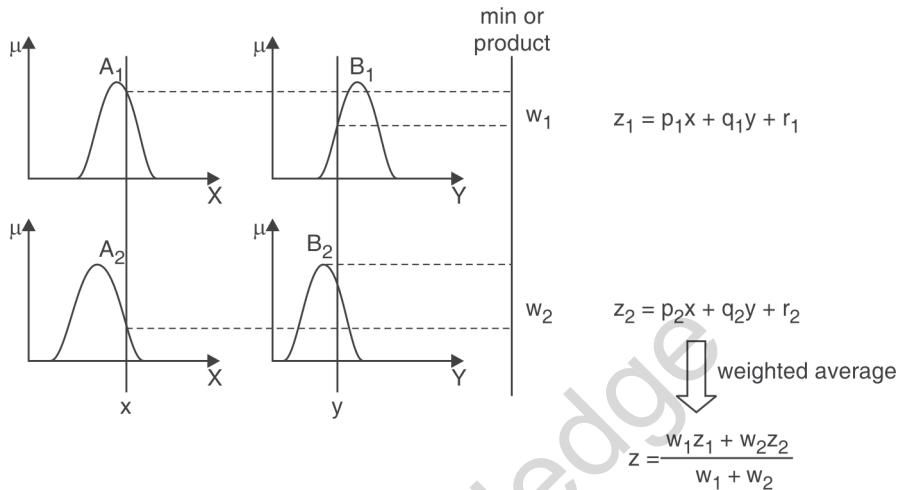


Fig. 4.7.3 : Reasoning in Sugeno FIS

Comparison between the Mamdani System and the Sugeno Model

- **Output Membership Function :** The main difference between them is on the basis of output membership function. The Sugeno output membership functions are either linear or constant.
- **Aggregation and Defuzzification Procedure :** The difference between them also lies in the consequence of fuzzy rules and due to the same their aggregation and defuzzification procedure also differs.
- **Mathematical Rules :** More mathematical rules exist for the Sugeno rule than the Mamdani rule.
- **Adjustable Parameters :** The Sugeno controller has more adjustable parameters than the Mamdani controller.

4.7.2 Fuzzification of Input Variables

- Fuzzification is the process of converting a crisp set into a fuzzy set.
- Here the crisp value is transformed into linguistic variables.
- In real word problems, many a times the input values are not very precise and accurate rather they are uncertain, imprecise and unknown.
- The uncertainty may arise due to the vagueness and incompleteness of data. In such cases, variable may be represented as fuzzy and can be represented as fuzzy membership function.

Methods of membership value of assignment

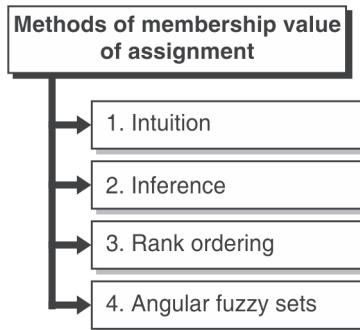


Fig. 4.7.4

1. Intuition

- As the name suggest, this method is based upon the common intelligence of human. The human develops membership functions based on their own understanding capability.

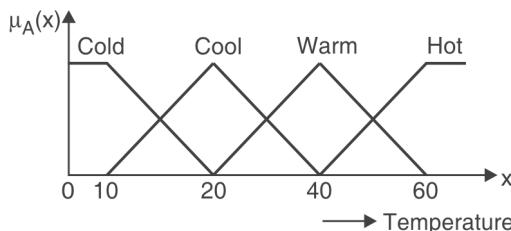


Fig. 4.7.5 : Membership functions for fuzzy variable “temperature”

- As shown in Fig. 4.7.5 each triangular curve is a membership function corresponding to various fuzzy (linguistic) variables such as cold, cool, warm etc.

2. Inference

- In inference method we use knowledge to perform deductive reasoning. To deduce or infer a conclusion, we use the facts and knowledge on that particular problem. Let us consider the example of Geometric shapes for the identification of a triangle.

Let A, B, C be the interior angles of a triangle such that,

$$A \geq B \geq C > 0^\circ \quad \text{and} \quad A + B + C = 180^\circ$$

- For this purpose, we define five types of triangles.
 1. R = Approximately Right-angle triangle
 2. I = Approximately Isosceles triangle
 3. E = Approximately Equilateral triangle
 4. I ∙ R = Isosceles Right-angle triangle
 5. T = Other type of triangle
- Now, we can infer membership values for all those types of triangles through the method of inference because we possess the knowledge about the geometry of their shapes.

The membership values for five types of triangle can be defined as,

$$\mu_R(A, B, C) = 1 - \frac{1}{90^\circ} |A - 90^\circ|$$

$$\mu_I(A, B, C) = 1 - \frac{1}{60^\circ} \min \{(A - B), (B - C)\}$$

$$\mu_E(A, B, C) = 1 - \frac{1}{80^\circ} |A - C|$$

$$\begin{aligned}\mu_{I \cdot R}(A, B, C) &= \mu_I \cap R(A, B, C) \\ &= \min \{\mu_I(A, B, C), \mu_R(A, B, C)\}\end{aligned}$$

$$\mu_T(A, B, C) = (R \cup I \cup E) = R \cap \bar{I} \cap \bar{E}$$



Ex :

$$\begin{aligned}
 \mu(A, B, C) &= \{80, 65, 35\} \\
 \mu_R(A, B, C) &= 1 - \frac{1}{90} |80 - 90| = \frac{8}{9} \\
 \mu_I(A, B, C) &= 1 - \frac{1}{60} \min \{15, 45\} = \frac{3}{4} \\
 \mu_E(A, B, C) &= 1 - \frac{1}{180} |45| = \frac{3}{4} \\
 \mu_{IR}(A, B, C) &= \min \{\mu_I, \mu_R\} \frac{3}{4} \\
 \mu_T &= R^C \cap I^C \cap E^E \\
 &= \min \left\{ \frac{1}{9}, \frac{1}{4}, \frac{1}{4} \right\} \\
 &= \frac{1}{4}
 \end{aligned}$$

3. Rank ordering

- In rank ordering method, preferences are assigned by a single individual, committee, a poll and other opinion methods can be used to assign membership values to fuzzy variables.
- Here the preferences are determined by pair wise comparisons and they are used to determine ordering of the membership.

Example :

Let's suppose 1000 people respond to a questionnaire and their pair wise preferences among the colors red, orange, yellow and blue is given as below.

	Red	Orange	Yellow	Green	Blue
Red	-	517	525	545	661
Orange	483	-	891	477	576
Yellow	474	159	-	534	614
Green	455	523	466	-	643
Blue	339	424	386	357	-

4. Angular fuzzy sets

- Angular fuzzy sets differ from normal fuzzy sets only in their coordinate description.
- Angular fuzzy sets are defined on a universe of angles; hence they are of repeating shapes for every 2π cycles.
- Angular fuzzy sets are used in the quantitative description of the linguistic variables, which are known as "truth values".

Example :

Let's consider that pH values of water samples are taken from a contaminated pond. We know that,

- If P_n value is 7 means it's a neutral solution.

- Levels of P_n between 14 and 7 are labelled as Absolute Basic (AB), Very Basic (VB), Basic (B), Fairly Basic (FB), Neutral (N) drawn from $\theta = \frac{\pi}{2}$ to $\theta = -\frac{\pi}{2}$
- Levels of P_n between 7 to 0 are called neutral, Fairly Acidic (FA), Acidic (A), Very Acidic (VA), Absolutely Acidic (AA), are drawn from $\theta = 0$ to $\theta = -\frac{\pi}{2}$.
- Linguistic values vary with θ and their membership values are given by equation,

$$E_\theta = t \tan \theta$$

Here 't' is the horizontal projection of the radial vector.

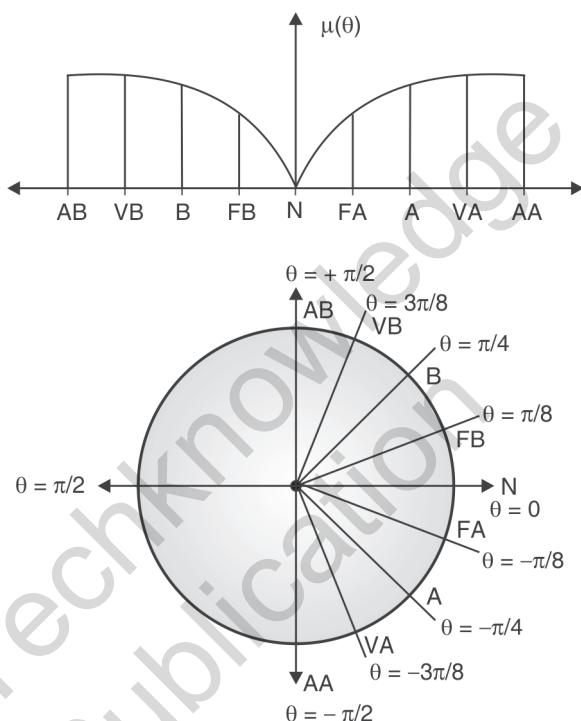


Fig. 4.7.6 : Model of angular fuzzy set

4.7.3 Defuzzification

MU - Dec. 12, Dec. 14

Q. Explain any four defuzzification methods with suitable example.

(Dec. 12, 10 Marks)

Q. Explain different methods of defuzzification.

(Dec. 14, 10 Marks)

- Defuzzification is the process of converting a fuzzy set into a crisp value.
- The output of a fuzzy process may be union of two or more fuzzy membership functions. In that case we need to find crisp value as a representative of the entire fuzzy MF.
- Different methods of defuzzification are listed below :

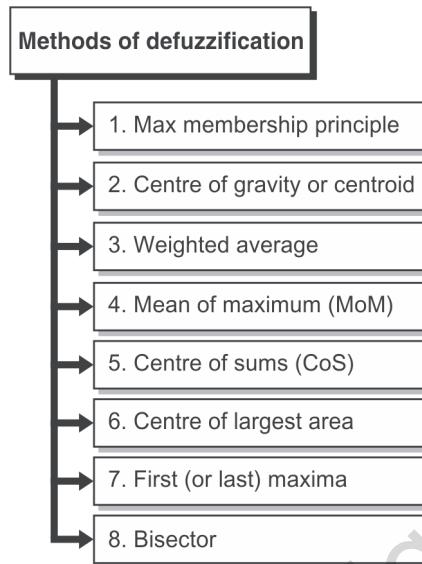


Fig. 4.7.7 : Defuzzification methods

1. Max-membership principle / Height method

- This method is limited to peak output functions. It uses the individual clipped or scaled central outputs.

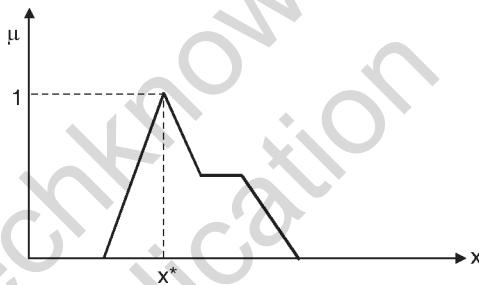


Fig. 4.7.8 : Max-membership

- The algebraic expression is:

$$\mu_c(x^*) \geq \mu_c(x) \text{ for all } x \in X$$

2. Centre of Area / Gravity (Centroid) Method

- This method is the most preferred and physically appealing of all the defuzzification methods.
- This method determines the centre of the area below the **combined membership function**. (i.e. it takes union of all output fuzzy sets).
- So, if there exist an overlapping area, will be considered only once. Thus overlapping areas are not reflected.
- This operation is computationally complex and therefore results in slow inference cycle.
- Algebraic expression is

For Continuous	For Discrete
$x^* = \frac{\int \mu_c(x) \cdot x \, dx}{\int \mu_c(x) \, dx}$	$x^* = \frac{\sum_{i=1}^n \mu_c(x_i) \cdot x_i}{\sum_{i=1}^n \mu_c(x_i)}$

- It is basically used for non-convex membership functions.

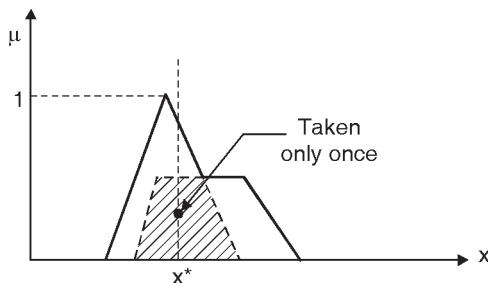


Fig. 4.7.9 : Centroid method

3. Centre of Sum (COS)

- This is faster than many defuzzification methods that are presently in use.
- This method involves the algebraic sum of individual output fuzzy sets, instead of their union.
- The idea is to consider the contribution of the area of each output membership curve.
- In contrast, the centre of area/gravity method considers the union of all output fuzzy sets.
- In COS method, we take overlapping areas. If such overlapping areas exist, they are reflected more than once.

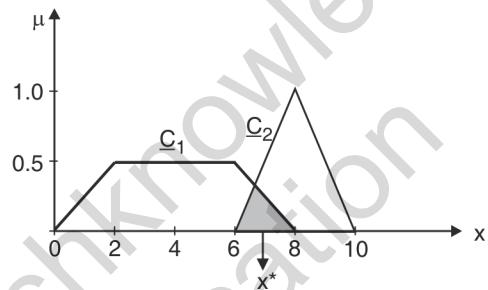


Fig. 4.7.10 : Centre of Sum method

For Continuous	For Discrete
$x^* = \frac{\int x \sum_{k=1}^N \mu_{C_k}(x) dx}{\int x \sum_{k=1}^N \mu_{C_k}(x) dx}$	$x^* = \frac{\sum_{i=1}^n x_i \sum_{k=1}^n \mu_{C_k}(x_i)}{\sum_{i=1}^n \sum_{k=1}^n \mu_{C_k}(x_i)}$

Advantage : It can be implemented easily and leads to a faster computation.

4. Weighted average method

- This method is only valid for **symmetrical** output membership functions.

Algebraic expression is :

$$x^* = \frac{\sum_{i=1}^n \mu_{C_i}(x_i) \cdot x_i}{\sum_{i=1}^n \mu_{C_i}(x_i)}$$

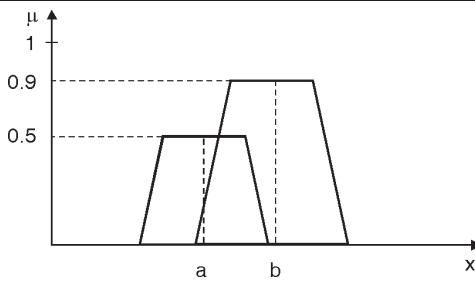


Fig. 4.7.11 : Weighted average method

- The weighted average method is formed by weighting each membership function in the output by its respective maximum membership value.
- The two functions shown in Fig. 4.7.11 would result in the following general form of defuzzification.

$$x^* = \frac{(a \times 0.5) + (b \times 0.9)}{0.5 + 0.9}$$

5. Mean-max membership (Middle of maxima)

- This method is closely related to the max-membership principle (height defuzzification) method; except that the locations of the maximum membership can be non-unique (can be more than one).
- In that case we take the average of the elements having maximum membership value of Maximizing MF.

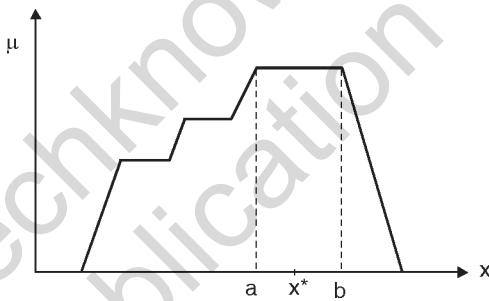


Fig. 4.7.12 : Mean of maximum method

- Algebraic expression is,

$$x^* = \frac{a + b}{2}$$

This method only works for convex.

6. Centre of largest area

- The centre of largest area method is used when the combined output fuzzy set is **non-convex** i.e. it consists of at least two convex fuzzy subsets.
- Then the method determines the convex fuzzy subset with the largest area and defines crisp output value x^* to be the centre of area of the largest fuzzy subset.

$$x^* = \frac{\int \mu_{\tilde{c}_m}(x) \cdot x \, dx}{\int \mu_{\tilde{c}_m}(x) \, dx}$$

Where, \tilde{c}_m is the convex fuzzy subset that has the largest area.

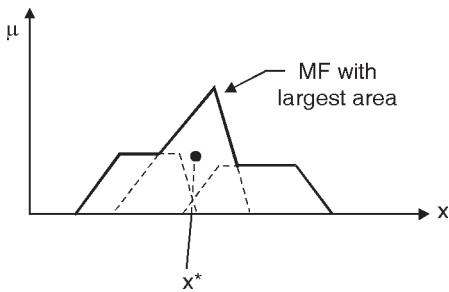


Fig. 4.7.13 : Centre of largest area

7. First (or last) of maxima

- This method uses the overall output (i.e. union of all individual output MF).
- First of maxima is determined by taking the smallest value of the domain with maximized membership degree.
- Last of maxima is determined by taking the greatest value of the domain with maximized membership degree.

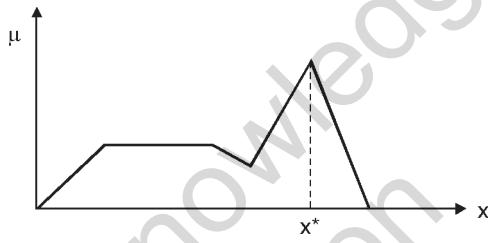


Fig. 4.7.14: First (or last) of maxima

8. Bisector method

This method uses the vertical line that divides the region into two equal areas as shown in Fig. 4.7.15. This line is called bisector.

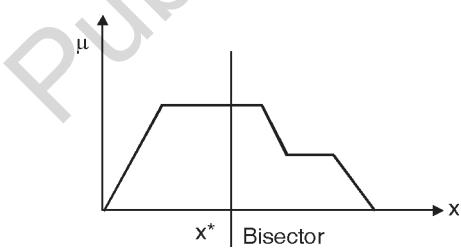


Fig. 4.7.15 : Bisector method of defuzzification

4.8 Fuzzy Controllers

- Most commercial fuzzy products use Fuzzy Knowledge-Based Controllers (FKBC).
- The principal structure of a FKBC is shown in Fig. 4.8.1.

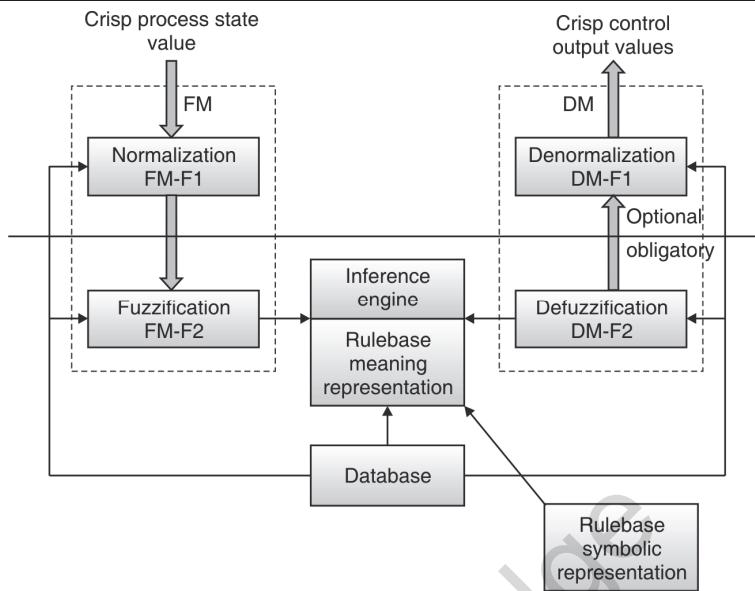


Fig. 4.8.1 : The structure of FKBC

- As shown in Fig.4.8.1, FKBC involves three important modules.

1. Fuzzification module
2. Decision making or Inferencing module
3. Defuzzification module

- In addition to this, it uses two more components
 - Data base and
 - Rule base
- } Knowledge base

1. Fuzzification module :

Fuzzification module performs the following two functions.

(a) Normalization

This block performs a scale transformation which maps the physical values of input variables in to a normalized universe of discourse. This block is optional. If a non-normalized domain is used then this block is not required.

(b) Fuzzification

This block performs a fuzzification which converts a crisp input in to a fuzzy set. Here we need to decide the proper fuzzification strategy.

2. Decision making/Inferencing module :

- The basic function of the inference engine is to compute the overall value of the control output variable based on the individual contribution of each rule in the rule base.
- The output of the fuzzification module representing the crisp input is matched to each rule-antecedent.
- The degree of match of each rule is established. Based on this degree of match, the value of the control output variable in the rule-consequent is modified. The result is, we get the “clipped” fuzzy set representing the control output variable.
- The set of all clipped control output values of the matched rules represent the overall fuzzy value of control output.

3. Defuzzification module :

Defuzzification module performs two tasks :

(a) Defuzzification

It performs defuzzification which converts the overall control output into a single crisp value.

(b) Denormalization module

- This block maps the crisp value of the control output into the physical domain. This block is optional. It is used only if normalization is performed during the fuzzification phase.

The knowledge base basically consists of a database and a rule base.

- The **database** provides the necessary information for proper functioning of the fuzzification module, the rule base and the defuzzification module.
- The information in the database includes :
 - Fuzzy MFs for the input and output control variables
 - The physical domains of the actual problems and their normalized values along with the scaling factors.

4.8.1 Steps in Designing FLC

Following are the steps involved in designing FLC

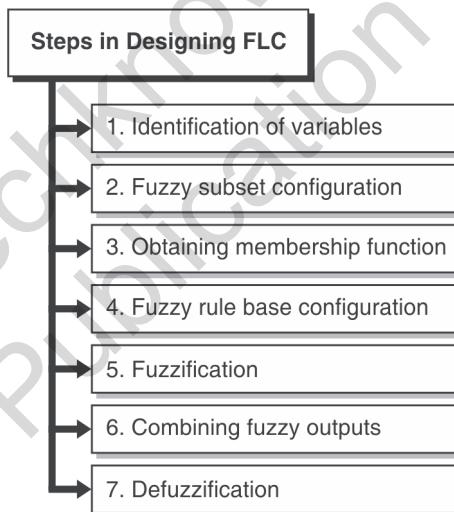


Fig. 4.8.2 Steps in designing FLC

1. **Identification of variables** : Here, the input, output and state variables must be identified of the plant which is under consideration.
2. **Fuzzy subset configuration** : The universe of information is divided into number of fuzzy subsets and each subset is assigned a linguistic label. Always make sure that these fuzzy subsets include all the elements of universe.
3. **Obtaining membership function** : Now obtain the membership function for each fuzzy subset that we get in the above step.
4. **Fuzzy rule base configuration** : Now formulate the fuzzy rule base by assigning relationship between fuzzy input and output.
5. **Fuzzification** :The fuzzification process is initiated in this step.

6. **Combining fuzzy outputs** : By applying fuzzy approximate reasoning, locate the fuzzy output and merge them.

7. **Defuzzification** : Finally, initiate defuzzification process to form a crisp output

4.8.2 Advantages of FLSs

- It uses very simple Mathematical concepts for reasoning.
- An FLS can be modified by just adding or deleting rules due to flexibility of fuzzy logic.
- Fuzzy logic Systems can take imprecise, distorted, noisy input information.
- FLSs are easy to construct and understand.
- Fuzzy logic is a solution to complex problems in all fields of life, including medicine, as it resembles human reasoning and decision making.

4.8.3 Disadvantages of FLSs

- There is no systematic approach to fuzzy system designing.
- They are understandable only when simple.
- They are suitable for the problems which do not need high accuracy.

4.9 Solved Problems

Ex. 4.9.1 : Model the following as fuzzy set using suitable membership function. "Numbers close to 6".

MU - Dec. 12, Dec. 13, Dec. 14, 6 Marks

Soln. : Let universe of discourse be the set of all integer numbers.

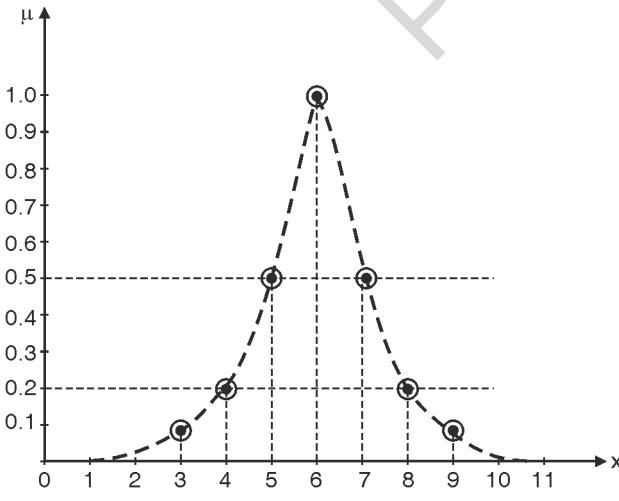
$$X = \text{Integers}$$

Then fuzzy set "Numbers close to 6" can be defined as

$$\mu_A(x) = \frac{1}{1 + (x - 6)^2}$$

Fig. P. 4.9.1 shows the plot of degree of membership of each element.

Table P. 4.9.1: x and corresponding $\mu_A(x)$



x	$\mu_A(x)$
2	0.05
3	0.1
4	0.2
5	0.5
6	1
7	0.5
8	0.2
9	0.1
10	0.05

Fig. P. 4.9.1 : Plot of $x \rightarrow \mu_A(x)$

Ex. 4.9.2 : Model the following fuzzy set using the suitable fuzzy membership function “Number close to 10”.

Soln. :

Let X be the universe of discourse.

$$X = \text{Integers}$$

Then fuzzy set “Number close to 10” can be defined as,

$$\mu_A(x) = \frac{1}{1 + (x - 10)^2}$$

Fig. P. 4.9.2 shows the plot of degree of membership for each element.

Table P. 4.9.2 : x and corresponding $\mu_A(x)$

x	$\mu_A(x)$
5	0.03
6	0.05
7	0.1
8	0.2
9	0.5
10	1
11	0.5
12	0.2
13	0.1
14	0.05
15	0.03

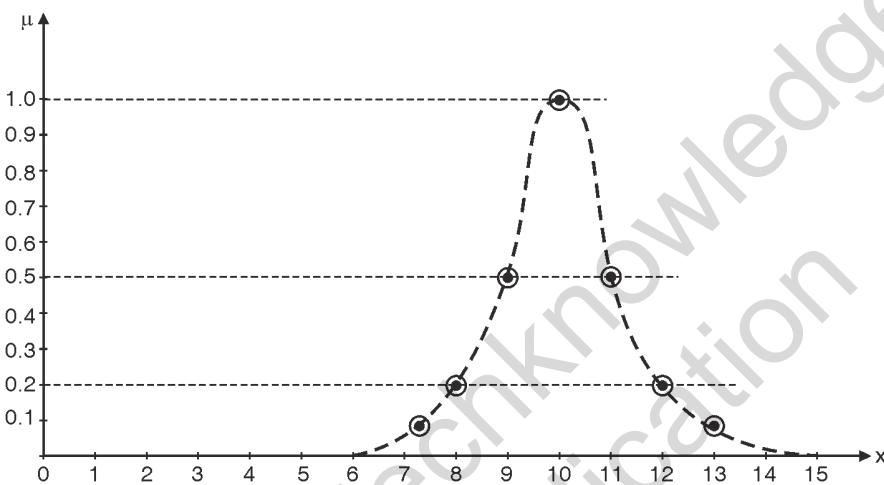


Fig. P. 4.9.2 : Plot of $x \rightarrow \mu_A(x)$

Ex. 4.9.3 : Model the following fuzzy set using suitable membership function. “Integer number considerably larger than 6”.

Soln. :

Here universe of discourse is set of all integer numbers.

$$X = \text{Integers}$$

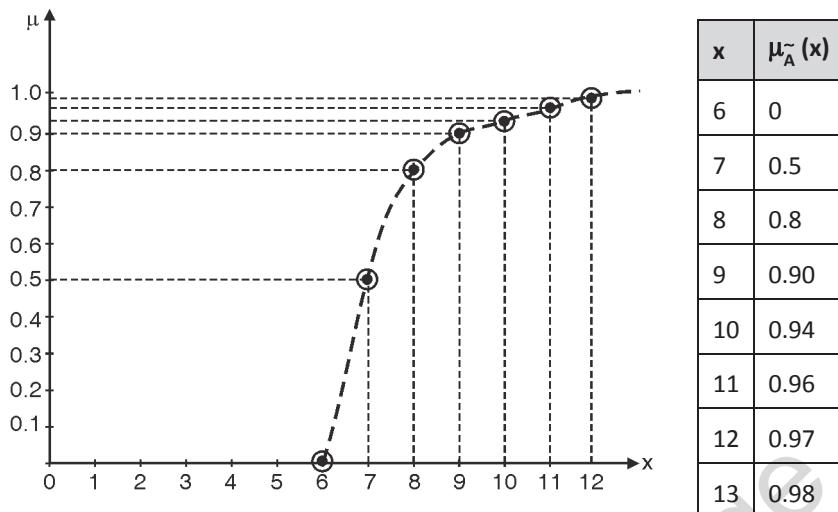
Then fuzzy set for “Number considerably larger than 6” can be defined as.

$$\mu_A(x) = \frac{1}{1 + \frac{1}{(x - 6)^2}}$$

Fig. P.4.9.3 shows the plot of $x \rightarrow \mu_A(x)$

So membership function for “Number considerably larger than 6” is defined as

$$\mu_A(x) = \begin{cases} 0 & , x \leq 6 \\ \frac{1}{1 + \frac{1}{(x - 6)^2}} & , x > 6 \end{cases}$$

Table P. 4.9.3 : x and corresponding $\mu_A^-(x)$ Fig. P. 4.9.3 : Plot of $x \rightarrow \mu_A^-(x)$

Ex. 4.9.4 : Determine all α -level sets and strong α -level sets for the following fuzzy set

$$A = \{(1, 0.2), (2, 0.5), (3, 0.8), (4, 1), (5, 0.7), (6, 0.3)\}$$

MU - Dec. 13, Dec. 15, 5/6 Marks

Soln. :

The following are α -level sets

$$\begin{aligned} A_{0.2} &= \{1, 2, 3, 4, 5, 6\} \\ A_{0.3} &= \{2, 3, 4, 5, 6\} \\ A_{0.5} &= \{2, 3, 4, 5\} \\ A_{0.7} &= \{3, 4, 5\} \\ A_{0.8} &= \{3, 4\} \\ A_1 &= \{4\} \end{aligned}$$

Following are strong α -level sets.

$$\begin{aligned} A_{0.2'} &= \{2, 3, 4, 5, 6\} \\ A_{0.3'} &= \{2, 3, 4, 5\} \\ A_{0.5'} &= \{3, 4, 5\} \\ A_{0.7'} &= \{3, 4\} \\ A_{0.8'} &= \{4\} \\ A_1' &= \emptyset \end{aligned}$$

Ex. 4.9.5 : Find out all α -level sets and strong α -level sets for the following fuzzy set

$$\tilde{A} = \{(3, 0.1), (4, 0.2), (5, 0.3), (6, 0.4), (7, 0.6), (8, 0.8), (10, 1), (12, 0.8), (14, 0.6)\}$$

Soln. :

α -level sets

$$\begin{aligned} A_{0.1} &= \{3, 4, 5, 6, 7, 8, 10, 12, 14\} \\ A_{0.2} &= \{4, 5, 6, 7, 8, 10, 12, 14\} \end{aligned}$$

$$A_{0.3} = \{5, 6, 7, 8, 10, 12, 14\}$$

$$A_{0.4} = \{6, 7, 8, 10, 12, 14\}$$

$$A_{0.6} = \{7, 8, 10, 12, 14\}$$

$$A_{0.8} = \{8, 10, 12\}$$

$$A_1 = \{10\}$$

Strong α -level sets

$$A_{0.1'} = \{4, 5, 6, 7, 8, 10, 12, 14\}$$

$$A_{0.2'} = \{5, 6, 7, 8, 10, 12, 14\}$$

$$A_{0.3'} = \{6, 7, 8, 10, 12, 14\}$$

$$A_{0.4'} = \{7, 8, 10, 12, 14\}$$

$$A_{0.6'} = \{8, 10, 12\}$$

$$A_{0.8'} = \{10\}$$

$$A_{1'} = \emptyset$$

Ex. 4.9.6 : A realtor wants to classify the houses he offers to his clients. One indicator of comfort of these houses is the number of bedrooms in them. Let the available types of houses be represented by the following set.

$$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

The houses in this set are specified by the number of bedrooms in a house. Describe comfortable house for 4-person family" using a fuzzy set.

Soln. :

The fuzzy set for "comfortable type of house for a 4-person family" may be described as,

$$\tilde{A} = \{(1, 0.2), (2, 0.5), (3, 0.8), (4, 1), (5, 0.7), (6, 0.3)\}$$

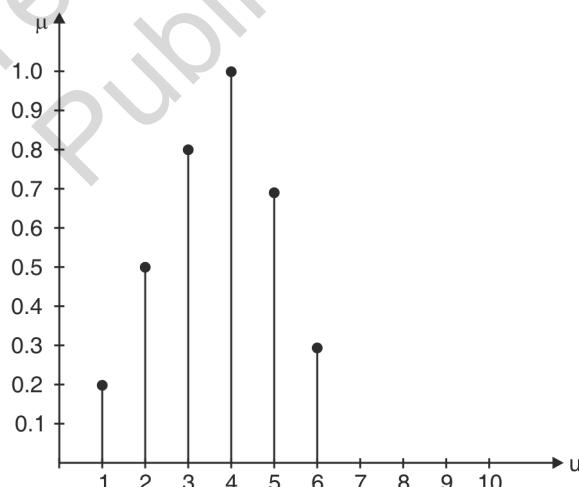


Fig. P. 4.9.6: Plot of $u \rightarrow \mu_A(u)$

Ex. 4.9.7 : Assume \tilde{A} = "x considerably larger than 10" and \tilde{B} = "x approximately 11" characterized by

$\tilde{A} = \{x, \mu_{\tilde{A}}(x) \mid x \in X\}$ Draw the plot for both the sets and show $\tilde{A} \cup \tilde{B}$ and $\tilde{A} \cap \tilde{B}$ in a plot.

Soln. :

Fuzzy set \tilde{A} can be defined as,

$$\mu_{\tilde{A}}(x) = \begin{cases} 0 & , x \leq 10 \\ \frac{1}{1 + \frac{1}{(x - 10)^2}} & , x > 10 \end{cases}$$

Set \tilde{B} can be defined as,

$$\mu_{\tilde{B}}(x) = \frac{1}{1 + (x - 11)^2}$$

Then,

$$\mu_{\tilde{A} \cap \tilde{B}}(x) = \begin{cases} \min [(1 + (x - 10)^{-2})^{-1}, (1 + (x - 11)^2)^{-1}] & , x > 10 \\ 0 & , x \leq 10 \end{cases}$$

That is, intersection operation on fuzzy set \tilde{A} and \tilde{B} represents a new fuzzy set "x considerably larger than 10 and approximately 11".

and

$$\mu_{\tilde{A} \cup \tilde{B}}(x) = \max [(1 + (x - 10)^{-2})^{-1}, (1 + (x - 11)^2)^{-1}], x \in X$$

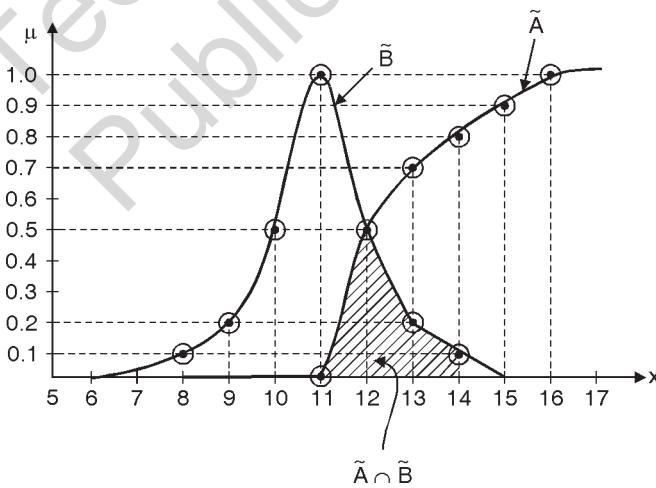
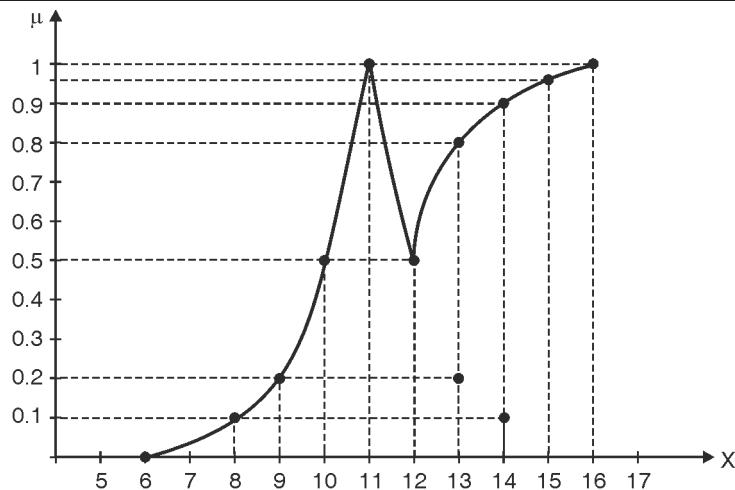


Fig. P. 4.9.7 : Plot of $\tilde{A} \cap \tilde{B}$

Fig. P. 4.9.7(a) : Plot of $\tilde{A} \cup \tilde{B}$

Ex. 4.9.8 : Model the following as fuzzy set using trapezoidal membership function “Number close to 10”.

Soln. :

“Number close to 10” can be represented by,

$$\text{Trapezoid}(x ; a, b, c, d) = \begin{cases} 0 & , \quad x \leq a \\ (x - a) / (b - a) & , \quad a \leq x \leq b \\ 1 & , \quad b \leq x \leq c \\ (d - x) / (d - c) & , \quad c \leq x \leq d \\ 0 & , \quad x > d \end{cases}$$

We have selected $a = 5$, $b = 8$, $c = 12$ and $d = 15$

$$\text{Trapezoid}(x ; 5, 8, 12, 15) = \begin{cases} 0 & , \quad x \leq 5 \\ (x - 5) / 3 & , \quad 5 \leq x \leq 8 \\ 1 & , \quad 8 \leq x \leq 12 \\ (15 - x) / 3 & , \quad 12 \leq x \leq 15 \\ 0 & , \quad x > 15 \end{cases}$$

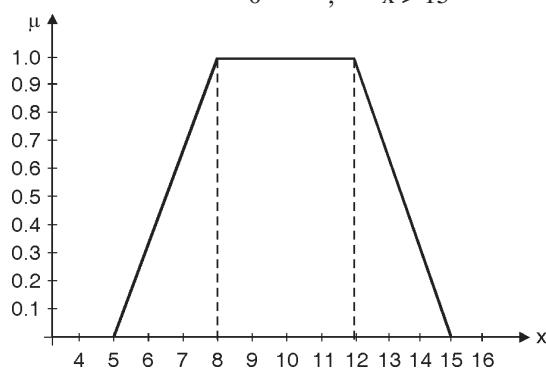


Fig. P. 4.9.8 : Trapezoidal MF for “Number close to 10”



Ex. 4.9.9 : Let $A = \{a_1, a_2\}$, $B = \{b_1, b_2, b_3\}$, $C = \{c_1, c_2\}$. Let R be a relation from A to B defined by matrix.

	b_1	b_2	b_3
a_1	0.4	0.5	0
a_2	0.2	0.8	0.2

Let S be a relation from B to C defined by matrix.

	c_1	c_2
b_1	0.2	0.7
b_2	0.3	0.8
b_3	1	0

Find : (1) Max-min composition of R and S . (2) Max-product composition of R and S .

Soln. :

(1) Max-min composition

T	c_1	c_2
a_1	0.3	0.5
a_2	0.3	0.8

$$\begin{aligned}
 T(a_1, c_1) &= \max(\min(0.4, 0.2), \min(0.5, 0.3), \min(0, 1)) \\
 &= \max(0.2, 0.3, 0) = 0.3 \\
 T(a_1, c_2) &= \max(\min(0.4, 0.7), \min(0.5, 0.8), \min(0, 0)) \\
 &= \max(0.4, 0.5, 0) = 0.5 \\
 T(a_2, c_1) &= \max(\min(0.2, 0.2), \min(0.8, 0.3), \min(0.2, 1)) \\
 &= \max(0.2, 0.3, 0.2) = 0.3 \\
 T(a_2, c_2) &= \max(\min(0.2, 0.7), \min(0.8, 0.8), \min(0.2, 0)) \\
 &= \max(0.2, 0.8, 0) = 0.8
 \end{aligned}$$

(2) Max-product composition

T	c_1	c_2
a_1	0.15	0.4
a_2	0.24	0.64

$$\begin{aligned}
 T(a_1, c_1) &= \max(0.4 \times 0.2, 0.5 \times 0.3, 0 \times 1) \\
 &= \max(0.08, 0.15, 0) = 0.15 \\
 T(a_1, c_2) &= \max(0.4 \times 0.7, 0.5 \times 0.8, 0 \times 0) \\
 &= \max(0.28, 0.40, 0) = 0.4 \\
 T(a_2, c_1) &= \max(0.2 \times 0.2, 0.8 \times 0.3, 0.2 \times 1)
 \end{aligned}$$



$$= \max(0.04, 0.24, 0.2) = 0.24$$

$$T(a_2, c_2) = \max(0.2 \times 0.7, 0.8 \times 0.8, 0.2 \times 0)$$

$$= \max(0.14, 0.64, 0) = 0.64$$

Ex. 4.9.10 : High speed rail monitoring devices sometimes make use of sensitive sensors to measure the deflection of the earth when a rail car passes. These deflections are measured with respect to some distance from the rail car and, hence are actually very small angles measured in micro-radians. Let a universe of deflection be $A = [1, 2, 3, 4]$ where A is the angle in micro-radians, and let a universe of distance be $D = [1, 2, 5, 7]$ where D is distance in feet, suppose a relation between these two parameters has been determined as follows :

R		D_1	D_2	D_3	D_4
A_1	1	0.3	0.1	0	
A_2	0.2	1	0.3	0.1	
A_3	0	0.7	1	0.2	
A_4	0	0.1	0.4	1	

Now let a universe of rail car weights be $W = [1, 2]$, where W is the weight in units of 100,000 pounds. Suppose the fuzzy relation of W to A is given by,

S		W_1	W_2
A_1	1	0.4	
A_2	0.5	1	
A_3	0.3	0.1	
A_4	0	0	

Using these two relations, find the relation $R^T \circ S = T$.

- (a) Using max-min composition.
- (b) Using max-product composition.

Soln. :

First find R^T .

	A_1	A_2	A_3	A_4		W_1	W_2
D_1	1	0.2	0	0	A_1	1	0.4
D_2	0.3	1	0.7	0.1	A_2	0.5	1
D_3	0.1	0.3	1	0.4	A_3	0.3	0.1
D_4	0	0.1	0.2	1	A_4	0	0

and $S =$

	W_1	W_2
A_1	1	0.4
A_2	0.5	1
A_3	0.3	0.1
A_4	0	0

(a) Using max-min composition

	W_1	W_2
D ₁	1	0.4
D ₂	0.5	1
D ₃	0.3	0.3
D ₄	0.2	0.1

$$T(D_1, W_1) = \max(1, 0.2, 0, 0) = 1$$

$$T(D_1, W_2) = \max(0.4, 0.2, 0, 0) = 0.4$$

$$T(D_2, W_1) = \max(0.3, 0.5, 0.3, 0) = 0.5$$

$$T(D_2, W_2) = \max(0.3, 1, 0.1, 0) = 1$$

$$T(D_3, W_1) = \max(0.1, 0.3, 0.3, 0) = 0.3$$

$$T(D_3, W_2) = \max(0.1, 0.3, 0.1, 0) = 0.3$$

$$T(D_4, W_1) = \max(0, 0.1, 0.2, 0) = 0.2$$

$$T(D_4, W_2) = \max(0, 0.1, 0.1, 0) = 0.1$$

(a) Using max product composition

	W_1	W_2
D ₁	1	0.4
D ₂	0.5	1
D ₃	0.3	0.3
D ₄	0.06	0.1

$$T(D_1, W_1) = \max(1 \times 1, 0.2 \times 0.5, 0 \times 0.3, 0 \times 0) = \max(1, 0.10, 0, 0) = 1$$

$$\begin{aligned} T(D_1, W_2) &= \max(1 \times 0.4, 0.2 \times 1, 0 \times 0.1, 0 \times 0) \\ &= \max(0.4, 0.2, 0, 0) = 0.4 \end{aligned}$$

$$\begin{aligned} T(D_2, W_1) &= \max(0.3 \times 1, 1 \times 0.5, 0.7 \times 0.3, 0.1 \times 0) \\ &= \max(0.3, 0.5, 0.21, 0) = 0.5 \end{aligned}$$

$$\begin{aligned} T(D_2, W_2) &= \max(0.3 \times 0.4, 1 \times 1, 0.7 \times 0.1, 0.1 \times 0) \\ &= \max(0.12, 1, 0.07, 0) = 1 \end{aligned}$$

$$\begin{aligned} T(D_3, W_1) &= \max(0.1 \times 1, 0.3 \times 0.5, 1 \times 0.3, 0.4 \times 0) \\ &= \max(0.1, 0.15, 0.3, 0) = 0.3 \end{aligned}$$

$$\begin{aligned} T(D_3, W_2) &= \max(0.1 \times 0.4, 0.3 \times 1, 1 \times 0.1, 0.4 \times 0) \\ &= \max(0.04, 0.3, 0.1, 0) = 0.3 \end{aligned}$$

$$\begin{aligned} T(D_4, W_1) &= \max(0 \times 1, 0.1 \times 0.5, 0.2 \times 0.3, 1 \times 0) \\ &= \max(0, 0.05, 0.06, 0) = 0.06 \end{aligned}$$

$$\begin{aligned} T(D_4, W_2) &= \max(0 \times 0.4, 0.1 \times 1, 0.2 \times 0.1, 1 \times 0) \\ &= \max(0, 0.1, 0.02, 0) = 0.1 \end{aligned}$$

Ex. 4.9.11 : Model the following fuzzy set using trapezoidal membership function, “Middle age”.

MU - May 13, 5 Marks

Soln. :

Let X be a reasonable age interval of human being.

$$X = \{0, 1, 2, 3, \dots, 100\}$$

Then a fuzzy set “Middle age” can be represented using Trapezoidal MF as follows.

$$\text{Trapezoid } (\mu_x; 30, 40, 60, 70) = \begin{cases} 0 & , \quad x \leq 30 \\ (x - 30) / 10 & , \quad 30 \leq x \leq 40 \\ 1 & , \quad 40 \leq x \leq 60 \\ (70 - x) / 10 & , \quad 60 \leq x \leq 70 \\ 0 & , \quad x > 70 \end{cases}$$

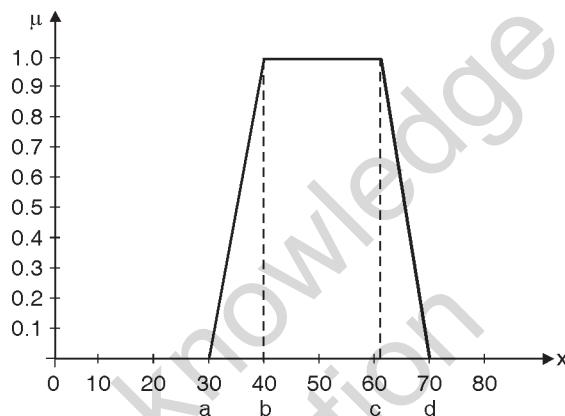


Fig. P. 4.9.11 : Trapezoidal MF for “Middle age”

Ex. 4.9.12 : Represent the set of old people as a fuzzy set using appropriate membership function.

Soln. :

Let $X = (0, 120)$ set of all possible ages.

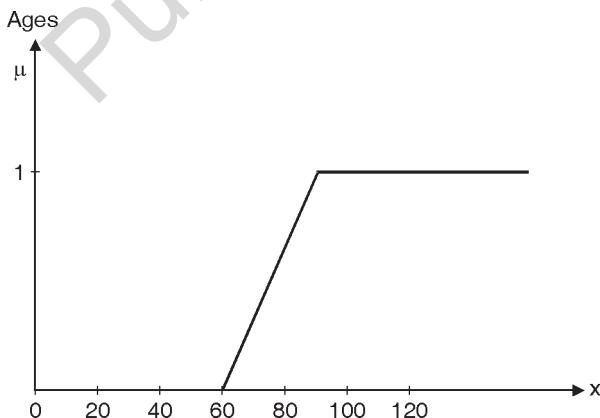


Fig. P. 4.9.12 : Membership function for “old people”

$$\mu_{old}(x) = \begin{cases} 0, & 0 \leq x \leq 60 \\ (x - 60)/20, & 60 \leq x \leq 80 \\ 1, & x \geq 80 \end{cases}$$

Ex. 4.9.13 : Develop graphical representation of membership function to describe linguistic variables “cold”, “warm” and “hot”. The temperature ranges from 0°C to 100°C . Also show plot for “cold and warm” and “warm or hot” temperature.

Soln. :

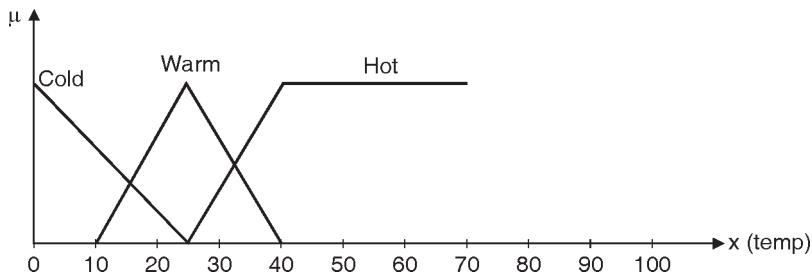


Fig. P. 4.9.13 : MF for cold, warm and hot temp.

a. Plot for “cold and warm”

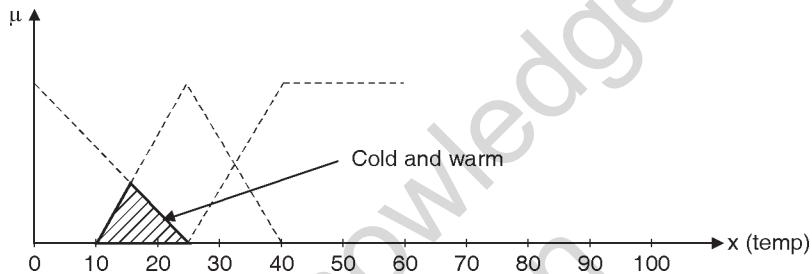


Fig. P. 4.9.13 (a) : MF for “cold and warm”

b. Plot for “warm or hot”

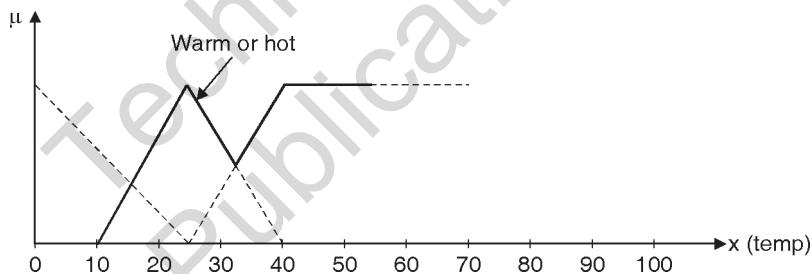


Fig. P. 4.9.13(b) : MF for “warm or hot”

Ex. 4.9.14 : Given two fuzzy sets.

$$\tilde{A} = \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} \right\}$$

$$\tilde{B} = \left\{ \frac{0.6}{1} + \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.5}{4} \right\}$$

Perform following operations on \tilde{A} and \tilde{B} .

- (1) Union
- (2) Intersection
- (3) Set difference
- (4) Verify Demorgan's law.



Soln. :

1. Union

$$\tilde{A} \cup \tilde{B} = \left\{ \frac{0.6}{1} + \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.5}{4} \right\}$$

2. Intersection

$$\tilde{A} \cap \tilde{B} = \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} \right\}$$

3. Set difference

$$\begin{aligned}\tilde{A} \setminus \tilde{B} &= \tilde{A} \cap \tilde{\bar{B}} \\ \tilde{A} &= \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} \right\} \\ \tilde{\bar{B}} &= \left\{ \frac{0.4}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{0.5}{4} \right\} \\ \tilde{A} \cap \tilde{\bar{B}} &= \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} \right\} \\ \tilde{B} \setminus \tilde{A} &= \tilde{B} \cap \tilde{\bar{A}} \\ \tilde{B} &= \left\{ \frac{0.6}{1} + \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.5}{4} \right\} \\ \tilde{\bar{A}} &= \left\{ \frac{0.9}{1} + \frac{0.8}{2} + \frac{0.7}{3} \right\} \\ \tilde{B} \cap \tilde{\bar{A}} &= \left\{ \frac{0.6}{1} + \frac{0.5}{2} + \frac{0.4}{3} \right\}\end{aligned}$$

4. Verification of Demorgan's law

To verify Demorgan's law first normalize both the sets \tilde{A} and \tilde{B} .

$$\tilde{A} = \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} + \frac{0}{4} \right\}$$

$$\tilde{B} = \left\{ \frac{0.6}{1} + \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.5}{4} \right\}$$

a. $\overline{\tilde{A} \cup \tilde{B}} = \tilde{\bar{A}} \cap \tilde{\bar{B}}$

L.H.S : $\overline{\tilde{A} \cup \tilde{B}}$

$$\tilde{A} \cup \tilde{B} = \left\{ \frac{0.6}{1} + \frac{0.5}{2} + \frac{0.4}{3} + \frac{0.5}{4} \right\}$$

$$\overline{\tilde{A} \cup \tilde{B}} = \left\{ \frac{0.4}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{0.5}{4} \right\} \quad \dots(1)$$

R.H.S : $\tilde{\bar{A}} \cap \tilde{\bar{B}}$

$$\tilde{\bar{A}} = \left\{ \frac{0.9}{1} + \frac{0.8}{2} + \frac{0.7}{3} + \frac{1}{4} \right\}$$

$$\begin{aligned}\tilde{\tilde{B}} &= \left\{ \frac{0.4}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{0.5}{4} \right\} \\ \tilde{\tilde{A}} \cap \tilde{\tilde{B}} &= \left\{ \frac{0.4}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{0.5}{4} \right\} \quad \dots(2)\end{aligned}$$

Since L.H.S. = R.H.S. hence proved.

b. $\tilde{\tilde{A}} \cap \tilde{\tilde{B}} = \tilde{\tilde{A}} \cup \tilde{\tilde{B}}$

L.H.S. : $\tilde{\tilde{A}} \cap \tilde{\tilde{B}}$

$$\begin{aligned}\tilde{\tilde{A}} \cap \tilde{\tilde{B}} &= \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.3}{3} + \frac{0}{4} \right\} \\ \tilde{\tilde{A}} \cap \tilde{\tilde{B}} &= \left\{ \frac{0.9}{1} + \frac{0.8}{2} + \frac{0.7}{3} + \frac{1}{4} \right\} \quad \dots(1)\end{aligned}$$

R.H.S. : $\tilde{\tilde{A}} \cup \tilde{\tilde{B}}$

$$\tilde{\tilde{A}} \cup \tilde{\tilde{B}} = \left\{ \frac{0.9}{1} + \frac{0.8}{2} + \frac{0.7}{3} + \frac{1}{4} \right\} \quad \dots(2)$$

Since L.H.S. = R.H.S. hence proved.

Ex. 4.9.15 : For the given membership functions shown in Fig. P.4.9.15, determine the defuzzified output value by any two methods.

MU - May 13, 10 Marks

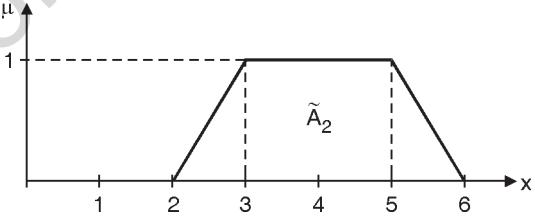
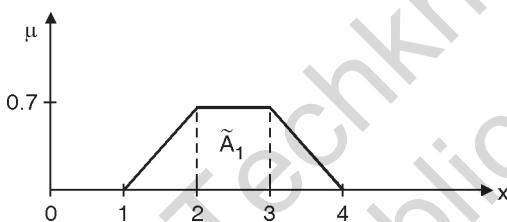


Fig. P. 4.9.15

Soln. :

1. Weighted average method

In weighted average method we first find the centre of each individual fuzzy set

Centre of $\tilde{A}_1 = 2.5$ Centre of $\tilde{A}_2 = 4.0$

Next we find the membership value at the centre

Membership value of centre of $\tilde{A}_1 = 0.7$

Membership value of centre of $\tilde{A}_2 = 1$

$$x^* = \frac{(0.7 * 2.5) + (1 * 4.0)}{0.7 + 1} = \frac{1.75 + 4}{1.7} = 3.38$$

2. Centre of sum method

First find the area of each individual curve.

We know that,

$$\text{Area of Trapezoid} = \frac{[(\text{Sum of length of parallel lines}) \times (\text{distance between parallel lines})]}{2}$$

$$\text{Therefore, Area of } \tilde{A}_1 = \frac{(1+3)*0.7}{2} = 1.4$$

$$\text{Similarly, Area of } \tilde{A}_2 = \frac{(2+4)*1}{2} = \frac{6}{2} = 3$$

Next, find center of each curve.

$$\text{Centre of } \tilde{A}_1 = 2.5$$

$$\text{Centre of } \tilde{A}_2 = 4$$

$$X^* = \frac{(2.5 \times 1.4) + (4 \times 3)}{1.4 + 3} = \frac{15.5}{4.4} = 3.52$$

Ex. 4.9.16 : Consider three fuzzy sets \tilde{C}_1 , \tilde{C}_2 and \tilde{C}_3 given below. Find defuzzified value using :

- | | |
|-----------------------|---------------------------|
| (1) mean of max | (2) centroid |
| (3) centre of sum and | (4) weighted avg. method. |

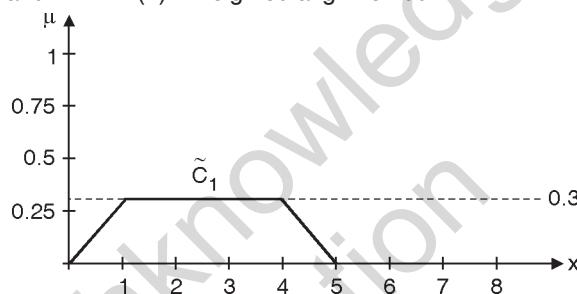


Fig. P. 4.9.16 : Fuzzy set \tilde{C}_1

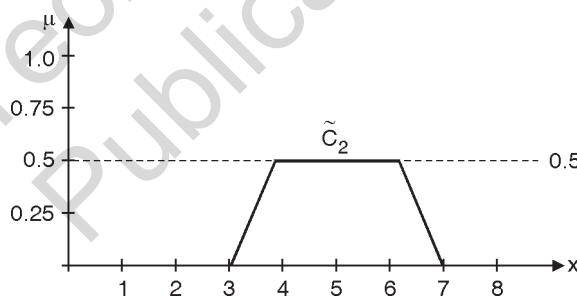


Fig. P. 4.9.16 (a) : Fuzzy set \tilde{C}_2

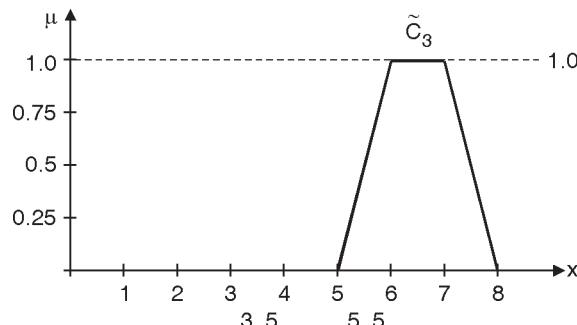


Fig. P. 4.9.16(b) : Fuzzy set \tilde{C}_3

Soln. :

First find aggregation of all MFs (union).

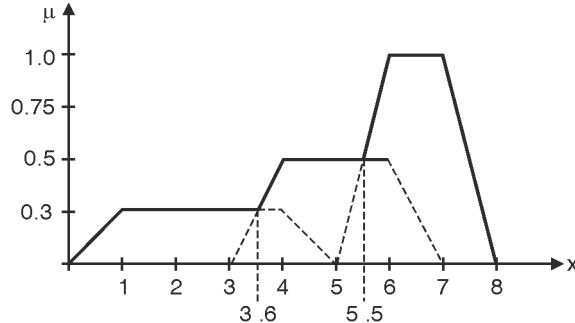


Fig. P. 4.9.16(c) : Aggregated fuzzy set of \tilde{C}_1 , \tilde{C}_2 and \tilde{C}_3

1) Using Mean of max

Since \tilde{C}_3 is the maximizing MF, we take the mean (average) of all the elements having maximum membership value in \tilde{C}_3

$$x^* = \frac{6+7}{2} = \frac{13}{2} = 6.5$$

2) Using Centroid method

$$\begin{aligned} z^* &= \frac{\int \mu_{\beta}(z) \cdot z dz}{\int \mu_{\beta}(z) dz} \\ &= \frac{\left[\int_0^1 (0.3z) dz + \int_1^{3.6} (0.3z) dz + \int_{3.6}^4 \left(\frac{z-3}{2}\right) z dz + \int_4^{5.5} (0.5) z dz + \int_{5.5}^6 (z-5) z dz + \int_6^7 z dz + \int_7^8 (8-z) z dz \right]}{\left[\int_0^1 (0.3z) dz + \int_1^{3.6} (0.3) dz + \int_{3.6}^4 \left(\frac{z-3}{2}\right) dz + \int_4^{5.5} (0.5) dz + \int_{5.5}^6 (z-5) dz + \int_6^7 dz + \int_7^8 (8-z) dz \right]} \\ &\div \left[\int_0^1 (0.3) dz + \int_1^{3.6} (0.3) dz + \int_{3.6}^4 \left(\frac{z-3}{2}\right) dz + \int_4^{5.5} (0.5) dz + \int_{5.5}^6 (z-5) dz + \int_6^7 dz + \int_7^8 (8-z) dz \right] \\ &= 4.9 \end{aligned}$$

3) Using centre of sum method

First find area of each individual fuzzy set

$$\text{Area of } \tilde{C}_1 = 1.2$$

$$\text{Area of } \tilde{C}_2 = 1.5$$

$$\text{Area of } \tilde{C}_3 = 2$$

Then find centre of each individual fuzzy set.

$$\text{Centre of } \tilde{C}_1 = 2.5$$

$$\text{Centre of } \tilde{C}_2 = 5$$

$$\text{Centre of } \tilde{C}_3 = 6.5$$



$$x^* = \frac{(2.5 \times 1.2) + (1.5 \times 5) + (2 \times 6.5)}{1.2 + 1.5 + 2}$$

$$\therefore x^* = 5$$

Note: Area of Trapezoid = $\frac{[(\text{Sum of length of parallel lines}) \times (\text{distance between parallel lines})]}{2}$

4) Using weighted average method

Find centre of each individual fuzzy set.

$$\text{Centre of } \tilde{C}_1 = 2.5$$

$$\text{Centre of } \tilde{C}_2 = 5$$

$$\text{Centre of } \tilde{C}_3 = 6.5$$

Find membership values of these centres.

$$\text{Membership value of centre of } \tilde{C}_1 = 0.3$$

$$\text{Membership value of centre of } \tilde{C}_2 = 0.5$$

$$\text{Membership value of centre of } \tilde{C}_3 = 1$$

$$x^* = \frac{(2.5 \times 0.3) + (5 \times 0.5) + (6.5 \times 1)}{0.3 + 0.5 + 1} = \frac{9.75}{1.8}$$

$$\therefore x^* = 5.146$$

Ex. 4.9.17 : Given fuzzy set.

$$\tilde{A} = \left\{ \frac{0.1}{1} + \frac{0.3}{2} + \frac{0.8}{3} + \frac{1}{4} + \frac{1}{5} + \frac{0.8}{6} \right\}$$

Find core and support of fuzzy set \tilde{A}

Soln. :

Core of $\tilde{A} = \{4, 5\} \rightarrow$ Membership value equal to 1

Support of $\tilde{A} = \{1, 2, 3, 4, 5, 6\} \rightarrow$ Membership value > 0

Ex. 4.9.18 : Consider following two fuzzy sets

$$\tilde{A} = \left\{ \frac{0.2}{1} + \frac{0.3}{2} + \frac{0.4}{3} + \frac{0.5}{4} \right\} \quad \tilde{B} = \left\{ \frac{0.1}{1} + \frac{0.2}{2} + \frac{0.2}{3} + \frac{1}{4} \right\}$$

Find : 1) Algebraic sum 2) Algebraic product

3) Bounded sum 4) Bounded difference

Soln. :

1) Algebraic sum

$$\mu_{\tilde{A} + \tilde{B}}(x) = [\mu_A(x) + \mu_B(x)] - [\mu_A(x) \cdot \mu_B(x)]$$

$$\begin{aligned}
 &= \left\{ \frac{0.3}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{1.5}{4} \right\} - \left\{ \frac{0.02}{1} + \frac{0.06}{2} + \frac{0.08}{3} + \frac{0.5}{4} \right\} \\
 &= \left\{ \frac{0.28}{1} + \frac{0.44}{2} + \frac{0.52}{3} + \frac{1}{4} \right\}
 \end{aligned}$$

2) Algebraic product

$$\begin{aligned}
 \mu_{\tilde{A} \cdot \tilde{B}}(x) &= \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x) \\
 &= \left[\frac{0.02}{1} + \frac{0.06}{2} + \frac{0.08}{3} + \frac{0.5}{4} \right]
 \end{aligned}$$

3) Bounded sum

$$\begin{aligned}
 \mu_{\tilde{A} \oplus \tilde{B}}(x) &= \min [1, \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x)] \\
 &= \min \left\{ 1, \left\{ \frac{0.3}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{1.5}{4} \right\} \right\} = \left\{ \frac{0.3}{1} + \frac{0.5}{2} + \frac{0.6}{3} + \frac{1}{4} \right\}
 \end{aligned}$$

4) Bounded difference

$$\begin{aligned}
 \mu_{\tilde{A} \ominus \tilde{B}}(x) &= \max [0, \mu_{\tilde{A}}(x) - \mu_{\tilde{B}}(x)] \\
 &= \max \left\{ 0, \left\{ \frac{0.1}{1} + \frac{0.1}{2} + \frac{0.2}{3} + \frac{-0.5}{4} \right\} \right\} = \left\{ \frac{0.1}{1} + \frac{0.1}{2} + \frac{0.2}{3} + \frac{0}{4} \right\}
 \end{aligned}$$

Ex. 4.9.19 : For given two fuzzy sets

$$\tilde{A} = \left\{ \frac{0.3}{x_1} + \frac{0.7}{x_2} + \frac{1}{x_3} \right\} \text{ and } \tilde{B} = \left\{ \frac{0.4}{y_1} + \frac{0.9}{y_2} \right\}$$

Perform Cartesian product.

Soln. :

For Cartesian product we consider min operator (denoted by \wedge)

$$\begin{aligned}
 \tilde{A} \times \tilde{B} &= \left\{ \frac{0.3 \wedge 0.4}{(x_1, y_1)} + \frac{0.3 \wedge 0.9}{(x_1, y_2)} + \frac{0.7 \wedge 0.4}{(x_2, y_1)} + \frac{0.7 \wedge 0.9}{(x_2, y_2)} + \frac{1 \wedge 0.4}{(x_3, y_1)} + \frac{1 \wedge 0.9}{(x_3, y_2)} \right\} \\
 &= \left\{ \frac{0.3}{(x_1, y_1)} + \frac{0.3}{(x_1, y_2)} + \frac{0.4}{(x_2, y_1)} + \frac{0.7}{(x_2, y_2)} + \frac{0.4}{(x_3, y_1)} + \frac{0.9}{(x_3, y_2)} \right\}
 \end{aligned}$$

It can be represented in a matrix form

		y_1	y_2
$\tilde{A} \times \tilde{B} =$	x_1	0.3	0.3
	x_2	0.4	0.7
	x_3	0.4	0.9

Ex. 4.9.20 : Consider the following fuzzy sets

$$\text{Low temperature} = \left\{ \frac{1}{131} + \frac{0.8}{132} + \frac{0.6}{133} + \frac{0.4}{134} + \frac{0.2}{135} + \frac{0}{136} \right\}$$

$$\text{High temperature} = \left\{ \frac{0}{134} + \frac{0.2}{135} + \frac{0.4}{136} + \frac{0.6}{137} + \frac{0.8}{138} + \frac{1}{139} \right\}$$



$$\text{High pressure} = \left\{ \frac{0.1}{400} + \frac{0.2}{600} + \frac{0.4}{700} + \frac{0.6}{800} + \frac{0.8}{900} + \frac{1}{1000} \right\}$$

Temperature ranges are 130° F to 140° F and pressure limit is 400 psi to 1000 psi. Find the following membership functions :

- | | |
|------------------------------|------------------------------|
| 1) Temperature not very low. | 2) Temperature not very high |
| 3) Pressure slightly high. | 4) Pressure very very high. |

Soln. :

1. Temperature not very low

$$\begin{aligned} \text{Very low} &= \text{low}^2 \\ &= \left\{ \frac{1}{131} + \frac{0.64}{132} + \frac{0.36}{133} + \frac{0.16}{134} + \frac{0.04}{135} + \frac{0}{136} \right\} \end{aligned}$$

$$\begin{aligned} \text{Not very low} &= 1 - \text{very low} \\ &= \left\{ \frac{0}{131} + \frac{0.36}{132} + \frac{0.64}{133} + \frac{0.84}{134} + \frac{0.96}{135} + \frac{1}{136} \right\} \end{aligned}$$

2. Temperature not very high

$$\begin{aligned} \text{SVery high} &= \text{high}^2 \\ \therefore \text{Temp very high} &= \left\{ \frac{0}{134} + \frac{0.04}{135} + \frac{0.16}{136} + \frac{0.36}{137} + \frac{0.64}{138} + \frac{1}{139} \right\} \end{aligned}$$

$$\text{Temp not very high} = 1 - \text{very high}$$

$$= \left\{ \frac{1}{134} + \frac{0.96}{135} + \frac{0.84}{136} + \frac{0.64}{137} + \frac{0.36}{138} + \frac{0}{139} \right\}$$

3. Pressure slightly high

$$\text{Slightly high} = \text{dilation (high)} = \sqrt{\text{high}}$$

$$\therefore \text{Pressure slightly high} = (\text{high pressure})^{1/2}$$

$$\begin{aligned} &= \left\{ \frac{\sqrt{0.1}}{400} + \frac{\sqrt{0.2}}{600} + \frac{\sqrt{0.4}}{700} + \frac{\sqrt{0.6}}{800} + \frac{\sqrt{0.8}}{900} + \frac{\sqrt{1}}{1000} \right\} \\ &= \left\{ \frac{0.31}{400} + \frac{0.44}{600} + \frac{0.63}{700} + \frac{0.77}{800} + \frac{0.89}{900} + \frac{1}{1000} \right\} \end{aligned}$$

4. Pressure very very high

$$\text{Very very high} = (\text{high})^4$$

$$\therefore \text{Pressure very very high} = (\text{high pressure})^4$$

$$= \left\{ \frac{0.0001}{400} + \frac{0.0016}{600} + \frac{0.025}{700} + \frac{0.12}{800} + \frac{0.40}{900} + \frac{1}{1000} \right\}$$

Ex. 4.9.21 : Two fuzzy relations are given by

$$y_1 \quad y_2 \quad R = \begin{bmatrix} x_1 & 0.6 & 0.3 \\ x_2 & 0.2 & 0.9 \end{bmatrix}$$

$$z_1 \quad z_2 \quad z_3 \quad S = \begin{bmatrix} y_1 & 1 & 0.5 & 0.3 \\ y_2 & 0.8 & 0.4 & 0.7 \end{bmatrix}$$

Obtain fuzzy relation T as a max-min composition and max-product composition between the fuzzy relations.

MU-May 16, 10 Marks



Soln. :

1. Using max-min composition

$$\begin{aligned} T(x_1, z_1) &= \max(\min(0.6, 1), \min(0.3, 0.8)) \\ &= \max(0.6, 0.3) \\ &= 0.6 \end{aligned}$$

$$\begin{aligned} T(x_1, z_2) &= \max(\min(0.6, 0.5), \min(0.3, 0.4)) \\ &= \max(0.5, 0.3) \\ &= 0.5 \end{aligned}$$

$$\begin{aligned} T(x_2, z_3) &= \max(\min(0.6, 0.3), \min(0.3, 0.7)) \\ &= \max(0.3, 0.3) \\ &= 0.3 \end{aligned}$$

$$\begin{aligned} T(x_2, z_1) &= \max(\min(0.2, 1), \min(0.9, 0.8)) \\ &= \max(0.2, 0.8) \\ &= 0.8 \end{aligned}$$

$$\begin{aligned} T(x_2, z_2) &= \max(\min(0.2, 0.5), \min(0.9, 0.4)) \\ &= \max(0.2, 0.4) \\ &= 0.4 \end{aligned}$$

$$\begin{aligned} T(x_2, z_3) &= \max(\min(0.2, 0.3), \min(0.9, 0.7)) \\ &= \max(0.2, 0.7) \\ &= 0.7 \end{aligned}$$

$$T = R \otimes S \quad \begin{matrix} X_1 \\ X_2 \end{matrix} \quad \left[\begin{array}{ccc} 0.6 & 0.5 & 0.3 \\ 0.8 & 0.4 & 0.7 \end{array} \right]$$

2. Using max-product

$$\begin{aligned} T(X_1, Z_1) &= \max(0.6 \times 1, 0.3 \times 0.8) \\ &= \max(0.6, 0.24) \\ &= 0.6 \end{aligned}$$

$$\begin{aligned} T(X_1, Z_2) &= \max(0.6 \times 0.5, 0.3 \times 0.4) \\ &= \max(0.30, 0.12) \\ &= 0.3 \end{aligned}$$

$$\begin{aligned} T(X_1, Z_3) &= \max(0.6 \times 0.3, 0.3 \times 0.7) \\ &= \max(0.18, 0.21) \\ &= 0.21 \end{aligned}$$

$$\begin{aligned} T(X_2, Z_1) &= \max(0.2 \times 1, 0.9 \times 0.8) \\ &= \max(0.2, 0.72) \\ &= 0.72 \end{aligned}$$

$$T(X_2, Z_2) = \max(0.2 \times 0.5, 0.9 \times 0.4)$$



$$= \max (0.1, 0.36)$$

$$= 0.36$$

$$T(X_2, Z_3) = \max (0.2 \times 0.3, 0.9 \times 0.7)$$

$$= \max (0.06, 0.63)$$

$$= 0.63$$

$$T = R \circ S = \begin{matrix} & Z_1 & Z_2 & Z_3 \\ X_1 & 0.6 & 0.3 & 0.21 \\ X_2 & 0.72 & 0.36 & 0.63 \end{matrix}$$

Ex. 4.9.22 : Given two fuzzy relations R_1 and R_2 defined on $X \times Y$ and $Y \times Z$ respectively, where

$X = \{1, 2, 3\}$, $Y = \{\alpha, \beta, \gamma, \delta\}$ and $Z = \{a, b\}$. Find :

1. Max - min composition.
2. Max - product composition.

$$R_1 = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.5 \\ 0.4 & 0.3 & 0.7 & 0.9 \\ 0.6 & 0.1 & 0.8 & 0.2 \end{bmatrix} \quad R_2 = \begin{bmatrix} 0.1 & 0.9 \\ 0.2 & 0.3 \\ 0.5 & 0.6 \\ 0.7 & 0.3 \end{bmatrix}$$

Soln. :

1. Max - min composition

Here R_1 is defined on $X \times Y$ where $X = \{1, 2, 3\}$ and $Y = \{\alpha, \beta, \gamma, \delta\}$

$$R_1 = \begin{matrix} & \alpha & \beta & \gamma & \delta \\ 1 & 0.1 & 0.2 & 0.3 & 0.5 \\ 2 & 0.4 & 0.3 & 0.7 & 0.9 \\ 3 & 0.6 & 0.1 & 0.8 & 0.2 \end{matrix}$$

Similarly R_2 is defined on $Y \times Z$ where $Y = \{\alpha, \beta, \gamma, \delta\}$ and $Z = \{a, b\}$

So,

$$R_2 = \begin{matrix} & a & b \\ \alpha & 0.1 & 0.9 \\ \beta & 0.2 & 0.3 \\ \gamma & 0.5 & 0.6 \\ \delta & 0.7 & 0.3 \end{matrix}$$

So composition of R_1 and R_2 will be defined on $X \times Z$ where $X = \{1, 2, 3\}$ and $Z = \{a, b\}$.

Here $R_1 \circ R_2$ is 3×2 matrix

$$R_1 \circ R_2 = \begin{matrix} & a & b \\ 1 & 0.5 & 0.3 \\ 2 & 0.7 & 0.6 \\ 3 & 0.5 & 0.6 \end{matrix}$$

We compute each element of $R_1 \circ R_2$ as follows :

$$\mu_{R_1 \circ R_2}(1, a) = \max (\min (0.1, 0.1), \min (0.2, 0.2), \min (0.3, 0.5), \min (0.5, 0.7))$$

$$= \max(0.1, 0.2, 0.3, 0.5)$$

$$= 0.5$$

$$\mu_{R_1 \circ R_2}(1, b) = \max(\min(0.1, 0.9), \min(0.2, 0.3), \min(0.3, 0.6), \min(0.5, 0.3))$$

$$= \max(0.1, 0.2, 0.3, 0.3)$$

$$= 0.3$$

$$\mu_{R_1 \circ R_2}(2, a) = \max(\min(0.4, 0.1), \min(0.3, 0.2), \min(0.7, 0.5), \min(0.9, 0.7))$$

$$= \max(0.1, 0.2, 0.5, 0.7)$$

$$= 0.7$$

$$\mu_{R_1 \circ R_2}(2, b) = \max(\min(0.4, 0.9), \min(0.3, 0.3), \min(0.7, 0.6), \min(0.9, 0.3))$$

$$= \max(0.4, 0.3, 0.6, 0.3)$$

$$= 0.6$$

$$\mu_{R_1 \circ R_2}(3, a) = \max(\min(0.6, 0.1), \min(0.1, 0.2), \min(0.8, 0.5), \min(0.2, 0.7))$$

$$= \max(0.1, 0.1, 0.5, 0.2)$$

$$= 0.5$$

$$\mu_{R_1 \circ R_2}(3, b) = \max(\min(0.6, 0.9), \min(0.1, 0.3), \min(0.8, 0.6), \min(0.2, 0.3))$$

$$= \max(0.6, 0.1, 0.6, 0.2)$$

$$= 0.6$$

2. Max - product composition

$$R_1 \circ R_2 = \begin{bmatrix} & a & b \\ 1 & 0.35 & 0.18 \\ 2 & 0.63 & 0.42 \\ 3 & 0.40 & 0.54 \end{bmatrix}$$

$$\mu_{R_1 \circ R_2}(1, a) = \max(0.1 \times 0.1, 0.2 \times 0.2, 0.3 \times 0.5, 0.5 \times 0.7)$$

$$= \max(0.01, 0.04, 0.15, 0.35)$$

$$= 0.35$$

$$\mu_{R_1 \circ R_2}(1, b) = \max(0.1 \times 0.9, 0.2 \times 0.3, 0.3 \times 0.6, 0.5 \times 0.3)$$

$$= \max(0.09, 0.06, 0.18, 0.15)$$

$$= 0.18$$

$$\mu_{R_1 \circ R_2}(2, a) = \max(0.4 \times 0.1, 0.3 \times 0.2, 0.7 \times 0.5, 0.9 \times 0.7)$$

$$= \max(0.04, 0.06, 0.35, 0.63)$$

$$= 0.63$$

$$\mu_{R_1 \circ R_2}(2, b) = \max(0.4 \times 0.9, 0.3 \times 0.3, 0.7 \times 0.6, 0.9 \times 0.3)$$

$$= \max(0.36, 0.09, 0.42, 0.27)$$

$$= 0.42$$

$$\mu_{R_1 \circ R_2}(3, a) = \max(0.6 \times 0.1, 0.1 \times 0.2, 0.8 \times 0.5, 0.2 \times 0.7)$$

$$= \max(0.06, 0.02, 0.40, 0.14)$$



$$= 0.40$$

$$\mu_{R_1 \circ R_2}(3, b) = \max(0.6 \times 0.9, 0.1 \times 0.3, 0.8 \times 0.6 \times 0.2 \times 0.3)$$

$$= \max(0.54, 0.03, 0.48, 0.06) = 0.54$$

Ex. 4.9.23 : Let R be the relation that specifies the relationship between the 'color of a fruit' and 'grade of maturity'. Relation S specifies the relationship between 'grade of maturity' and 'taste of a fruit', where color, grade and taste of a fruit are characterized by crisp sets x, y, z respectively as follows.

$$X = \{\text{green, yellow, red}\}$$

$$Y = \{\text{verdant, half mature, mature}\}$$

$$Z = \{\text{sour, tasteless, sweet}\}$$

Consider following relations R and S and find the relationship between 'color and taste' of a fruit using

1. Max - min composition

R	Verdant	Half mature	Mature
Green	1	0.5	0
Yellow	0.3	1	0.4
Red	0	0.2	1

2. Max - product composition

S	Sour	Tasteless	Sweet
Verdant	1	0.2	0
Half mature	0.7	1	0.3
Mature	0	0.7	1

Soln. :

1. Max - min composition

T	Sour	Tasteless	Sweet
Green	1	0.5	0.3
Yellow	0.7	1	0.4
Red	0.2	0.7	1

$$\begin{aligned} T(\text{green, sour}) &= \max(\min(1, 1), \min(0.5, 0.7), \min(0, 0)) \\ &= \max(1, 0.5, 0) \\ &= 1 \end{aligned}$$

$$\begin{aligned} T(\text{green, tasteless}) &= \max(\min(1, 0.2), \min(0.5, 1), \min(0, 0.7)) \\ &= \max(0.2, 0.5, 0) \\ &= 0.5 \end{aligned}$$

$$\begin{aligned} T(\text{green, sweet}) &= \max(\min(1, 0), \min(0.5, 0.3), \min(0, 1)) \\ &= \max(0, 0.3, 0) \\ &= 0.3 \end{aligned}$$

$$\begin{aligned} T(\text{yellow, sour}) &= \max(\min(0.3, 1), \min(1, 0.7), \min(0.4, 0.7)) \\ &= \max(0.3, 0.7, 0.4) \\ &= 0.7 \end{aligned}$$

$$T(\text{yellow, tasteless}) = \max(\min(0.3, 0.2), \min(1, 1), \min(0.4, 0.7))$$



$$= \max(0.2, 1, 0.4)$$

$$= 1$$

$$T(\text{yellow, sweet}) = \max(\min(0.3, 0), \min(1, 0.3), \min(0.4, 1))$$

$$= \max(0, 0.3, 0.4)$$

$$= 0.4$$

$$T(\text{red, sour}) = \max(\min(0, 1), \min(0.2, 0.7), \min(1, 0))$$

$$= \max(0, 0.2, 0) = 0.2$$

$$T(\text{red, tasteless}) = \max(\min(0, 0.2), \min(0.2, 1), \min(1, 0.7))$$

$$= \max(0, 0.2, 0.7)$$

$$= 0.7$$

$$T(\text{red, sweet}) = \max(\min(0, 0), \min(0.2, 0.3), \min(1, 1))$$

$$= \max(0, 0.2, 1)$$

$$= 1$$

2. Max - product composition

T	Sour	Tasteless	Sweet
Green	1	0.5	0.15
Yellow	0.7	1	0.4
Red	0.14	0.7	1

$$T(\text{green, sour}) = \max(1 \times 1, 0.5 \times 0.7, 0 \times 0)$$

$$= \max(1, 0.35, 0)$$

$$= 1$$

$$T(\text{green, tasteless}) = \max(1 \times 0.2, 0.5 \times 1, 0 \times 0.7)$$

$$= \max(0.2, 0.5, 0)$$

$$= 0.5$$

$$T(\text{green, sweet}) = \max(1 \times 0, 0.5 \times 0.3, 0 \times 1)$$

$$= \max(0, 0.15, 0)$$

$$= 0.15$$

$$T(\text{yellow, sour}) = \max(0.3 \times 1, 1 \times 0.7, 0.4 \times 0.7)$$

$$= \max(0.3, 0.7, 0.28)$$

$$= 0.7$$

$$T(\text{yellow, tasteless}) = \max(0.3 \times 0.2, 1 \times 1, 0.4 \times 0.7)$$

$$= \max(0.06, 1, 0.28)$$

$$= 1$$

$$T(\text{yellow, sweet}) = \max(0.3 \times 0, 1 \times 0.3, 0.4 \times 1)$$

$$= \max(0, 0.3, 0.4)$$



$$= 0.4$$

$$T(\text{red, sour}) = \max(0 \times 1, 0.2 \times 0.7, 1 \times 0)$$

$$= \max(0, 0.14, 0)$$

$$= 0.14$$

$$T(\text{red, tasteless}) = \max(0 \times 0.2, 0.2 \times 1, 1 \times 0.7)$$

$$= \max(0, 0.2, 0.7)$$

$$= 0.7$$

$$T(\text{red, sweet}) = \max(0 \times 0, 0.2 \times 0.3, 1 \times 1)$$

$$= \max(0, 0.06, 1)$$

$$= 1$$

4.10 Design of Controllers (Solved Problems)

Note : All the problems based on controller design have been solved using Mamdani Fuzzy Inference model and mean of max defuzzification method.

1. Domestic Shower Controller

Ex. 4.10.1 : Design a fuzzy controller to regulate the temperature of a domestic shower. Assume that:

- The temperature is adjusted by single mixer tap.
- The flow of water is constant.
- Control variable is the ratio of the hot to the cold water input.

The design should clearly mention the descriptors used for fuzzy sets and control variables, set of rules to generate control action and defuzzification. The design should be supported by figures where ever possible.

Soln. :

Step 1 : Identify input and output variables and decide descriptors for the same.

- Here **input** is the position of mixer tap. Assume that position of mixer tap is measured in degrees (0° to 180°). It represents opening of the mixer tap in degrees. 0° indicates tap is closed and 180° indicates tap is fully opened.
- Output** is temperature of water according to the position of mixer tap. It is measured in $^\circ\text{C}$. We take **five descriptors** for each input and output variables.
- Descriptors for input variable (position of mixer tap) are given below.

EL - Extreme Left

L - Left

C - Centre

R - Right

ER - Extreme Right

{ EL, L, C, R, ER }

- Descriptors for output variable (Temperature of water) are given below :

VCT - Very Cold Temperature

CT - Cold Temperature

WT - Warm Temperature

HT - Hot Temperature

VHT - Very Hot Temperature

{ VCT, CT, WT, HT, VHT }

Step 2 : Define membership functions for input and output variables.

We use triangular MFs because of its simplicity.

1. Membership functions for input variable – position of mixer tap.

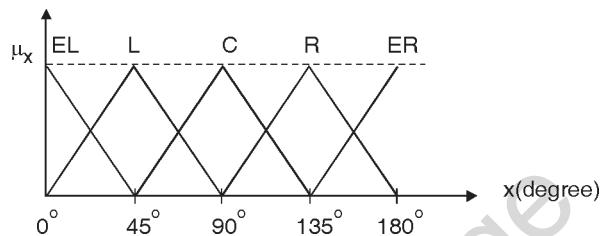


Fig. P. 4.10.1 : Membership function for position of mixer tap

$$\mu_{EL}(x) = \frac{45-x}{45}, 0 \leq x \leq 45$$

$$\mu_L(x) = \begin{cases} \frac{x}{45}, & 0 \leq x \leq 45 \\ \frac{90-x}{45}, & 45 < x \leq 90 \end{cases}$$

$$\mu_C(x) = \begin{cases} \frac{x-45}{45}, & 45 \leq x \leq 90 \\ \frac{135-x}{45}, & 90 < x \leq 135 \end{cases}$$

$$\mu_R(x) = \begin{cases} \frac{x-90}{45}, & 90 \leq x \leq 135 \\ \frac{180-x}{45}, & 135 < x \leq 180 \end{cases}$$

$$\mu_{ER}(x) = \frac{x-135}{45}, 135 \leq x \leq 180$$

2. Membership functions for output variable - temperature of water.

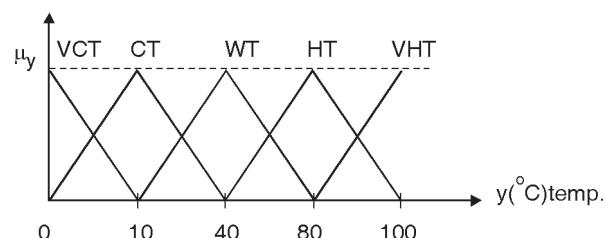


Fig. P. 4.10.1(a) : Membership functions for water temperature

$$\mu_{VCT}(y) = \frac{10-y}{10}, 0 \leq y \leq 10$$

$$\mu_{CT}(y) = \begin{cases} \frac{y}{10}, & 0 \leq y \leq 10 \\ \frac{40-y}{30}, & 10 < y \leq 40 \end{cases}$$

$$\mu_{WT}(y) = \begin{cases} \frac{y-10}{30}, & 10 \leq y \leq 40 \\ \frac{80-y}{40}, & 40 < y \leq 80 \end{cases}$$

$$\mu_{HT}(y) = \begin{cases} \frac{y-40}{40}, & 40 \leq y \leq 80 \\ \frac{100-y}{20}, & 80 < y \leq 100 \end{cases}$$

$$\mu_{VHT}(y) = \frac{y-80}{20}, 80 \leq y \leq 100$$

Step 3 : Form a rule base.

Table P. 4.10.1

Input (Mixer tap position)	Output (Temperature of water)
EL	VCT
L	CT
C	WT
R	HT
ER	VHT

We can read the rule base shown in Table P. 4.10.1 in terms of If-then rules.

Rule 1 : If mixer tap position is EL (Extreme Left) then temperature of water is VCT (Very Cold Temperature).

Rule 2 : If mixer tap position is L (Left) then temperature of water is CT (Cold).

Rule 3 : If mixer tap position is C (Centre) then temperature of water is WT (Warm)

Rule 4 : If mixer tap position is R (Right) then temperature of water is HT (Hot)

Rule 5 : If mixer tap position is ER (Extreme Right) then temperature of water is VHT (Very Hot).

Thus, we have five rules.

Step 4 : Rule Evaluation.

Assume that mixer tap position is 75° . This value $x = 75^\circ$ maps to following two MFs of Rule 2 and Rule 3 respectively.

$$\text{Rule 2 : } \mu_L(x) = \frac{90-x}{45}$$

$$\text{Rule 3 : } \mu_C(x) = \frac{x-45}{45}$$

Now, substitute the value of $x = 75$ in above two equations, we get strength of each rule

$$\text{Strength of Rule 2} \Rightarrow \mu_L(75) = \frac{90-75}{45} = \frac{1}{3}$$

$$\text{Strength of Rule 3} \Rightarrow \mu_C(75) = \frac{75-45}{45} = \frac{2}{3}$$

Step 5 : Defuzzification

We apply **mean of maximum** defuzzification technique.



We find the rule with the maximum strength

$$\begin{aligned}
 &= \max (\text{Strength of Rule 1, strength of Rule 2}) \\
 &= \max (\mu_L(x), \mu_c(x)) \\
 &= \max \left(\frac{1}{3}, \frac{2}{3} \right) = \frac{2}{3}
 \end{aligned}$$

Thus, Rule 3 has the maximum strength.

According to Rule 3, If mixer tap position is C (center) then water temperature is **Warm**. So, we use Output MFs of **warm water temperature** for defuzzification. We have following two equations for warm water temperature.

$$\mu_{WT}(y) = \frac{y-10}{30} \text{ and}$$

$$\mu_{WT}(y) = \frac{80-y}{40}$$

Since, the strength of rule 3 is $\frac{2}{3}$, substitute $\mu_{WT}(y) = \frac{2}{3}$ in the above two equations.

$$\frac{y-10}{30} = \frac{2}{3} \Rightarrow y = 30$$

$$\frac{80-y}{40} = \frac{2}{3} \Rightarrow y = 53$$

Now take the average (mean) of these values.

$$y^* = \frac{30+53}{2} = 41.5^\circ\text{C}$$

2. Washing Machine Controller

Ex. 4.10.2 : Design a controller to determine the wash time of a domestic washing machine. Assume that input is dirt and grease on cloths. Use three descriptors for input variables and five descriptors for output variables. Derive set of rules for controller action and defuzzification. The design should be supported by figures wherever possible. Show that if the cloths are soiled to a larger degree the wash time will be more and vice-versa.

MU - May 12, Dec. 12, Dec. 13, Dec. 14, 20 Marks

Soln. :

Step 1 : Identify input and output variables and decide descriptors for the same.

- Here inputs are ‘dirt’ and ‘grease’. Assume that they are measured in percentage (%). That is amount of dirt and grease is measured in percentage.
- Output is ‘wash time’ measured in minutes.
- We use three descriptors for each of the input variables.

Descriptors for dirt are as follows :

SD - Small Dirt

MD - Medium Dirt

LD - Large Dirt

{ SD, MD, LD }

Descriptors for grease are { NG, MG, LG }

NG - No Grease

MG - Medium Grease

LG - Large Grease

We use five descriptors for output variable.

So, descriptors for wash time are {VS, S, M, L, VL}

VS - Very Short

S - Short

M - Medium

L - Large

VL - Very Large

Step 2 : Define membership functions for each of the input and output variables.

We use triangular MFs because of their simplicity.

(1) Membership functions for dirt

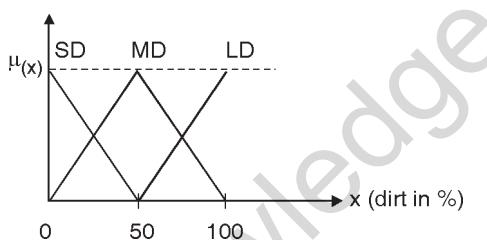


Fig. P. 4.10.2 : Membership functions for dirt

$$\mu_{SD}(x) = \frac{50-x}{50}, \quad 0 \leq x \leq 50$$

$$\mu_{MD}(x) = \begin{cases} \frac{x}{50}, & 0 \leq x \leq 50 \\ \frac{100-x}{50}, & 50 < x \leq 100 \end{cases}$$

$$\mu_{LD}(x) = \frac{x-50}{50}, \quad 50 \leq x \leq 100$$

(2) Membership functions for grease

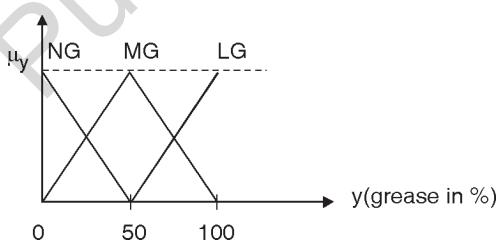


Fig. P. 4.10.2(a) : Membership functions of grease

$$\mu_{NG}(y) = \frac{50-y}{50}, \quad 0 \leq y \leq 50$$

$$\mu_{MG}(y) = \begin{cases} \frac{y}{50}, & 0 \leq y \leq 50 \\ \frac{100-y}{50}, & 50 < y \leq 100 \end{cases}$$

$$\mu_{LG}(y) = \frac{y-50}{50}, \quad 50 \leq y \leq 100$$

(3) Membership functions for wash time

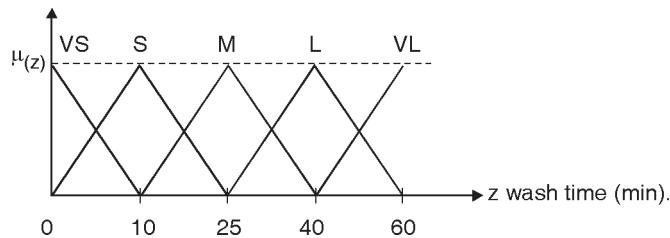


Fig. P. 4.10.2(b) : Membership functions for wash time

$$\mu_{VS}(z) = \frac{10-z}{10}, \quad 0 \leq z \leq 10$$

$$\mu_S(z) = \begin{cases} \frac{z}{10}, & 0 \leq z \leq 10 \\ \frac{25-z}{15}, & 10 < z \leq 25 \end{cases}$$

$$\mu_M(z) = \begin{cases} \frac{z-10}{15}, & 10 \leq z \leq 25 \\ \frac{40-z}{15}, & 25 < z \leq 40 \end{cases}$$

$$\mu_L(z) = \begin{cases} \frac{z-25}{15}, & 25 \leq z \leq 40 \\ \frac{60-z}{20}, & 40 < z \leq 60 \end{cases}$$

$$\mu_{VL}(z) = \frac{z-40}{20}, \quad 40 \leq z \leq 60$$

Step 3 : Form a Rule base

x	y	NG	MG	LG
SD		VS	M	L
MD		S	M	L
LD		M	L	VL

The above matrix represents in all nine rules. For example, first rule can be “If dirt is small and no grease then wash time is very short” similarly all nine rules can be defined using if --- then.

Step 4 : Rule Evaluation

Assume that dirt = 60 % and grease = 70%

dirt = 60 % maps to the following two MFs of “dirt” variable

$$\mu_{MD}(x) = \frac{100-x}{50} \text{ and } \mu_{LD}(x) = \frac{x-50}{50}$$

Similarly grease = 70 % maps to the following two MFs of “grease” variable.

$$\mu_{MG}(y) = \frac{100-y}{50} \text{ and } \mu_{LG}(y) = \frac{y-50}{50}$$

Evaluate $\mu_{MD}(x)$ and $\mu_{LD}(x)$ for $x = 60$, we get

$$\mu_{MD}(60) = \frac{100-60}{50} = \frac{4}{5} \quad \dots (1)$$

$$\mu_{LD}(60) = \frac{60 - 50}{50} = \frac{1}{5} \quad \dots(2)$$

Similarly evaluate $\mu_{MG}(y)$ and $\mu_{LG}(y)$ for $y = 70$, we get

$$\mu_{MG}(70) = \frac{100 - 70}{50} = \frac{3}{5} \quad \dots(3)$$

$$\mu_{LG}(70) = \frac{70 - 50}{50} = \frac{2}{5} \quad \dots(4)$$

The above four equation leads to the following four rules that we are suppose to evaluate.

- (1) Dirt is medium and grease is medium.
- (2) Dirt is medium and grease is large.
- (3) Dirt is large and grease is medium.
- (4) Dirt is large and grease is **large**.

Since the antecedent part of each of the above rule is connected by **and** operator we use **min** operator to evaluate strength of each rule.

$$\text{Strength of rule 1: } S_1 = \min(\mu_{MD}(60), \mu_{MG}(70)) = \min(4/5, 3/5) = 3/5$$

$$\text{Strength of rule 2: } S_2 = \min(\mu_{MD}(60), \mu_{LG}(70)) = \min(4/5, 2/5) = 2/5$$

$$\text{Strength of rule 3: } S_3 = \min(\mu_{LD}(60), \mu_{MG}(70)) = \min(1/5, 3/5) = 1/5$$

$$\text{Strength of rule 4: } S_4 = \min(\mu_{LD}(60), \mu_{LG}(70)) = \min(1/5, 2/5) = 1/5$$

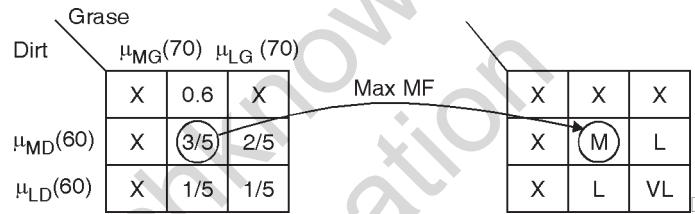


Fig. P.4.10.2(c): Rule strength and its mapping to corresponding output MF

Step 5 : Defuzzification

- Since, we use "**Mean of max**" defuzzification technique, we first find the rule with the maximum strength.

$$\begin{aligned}
 &= \text{Max } (S_1, S_2, S_3, S_4) \\
 &= \text{Max } (3/5, 2/5, 1/5, 1/5) \\
 &= 3/5
 \end{aligned}$$
- This corresponds to rule 1.
- This rule 1 – "dirt is **medium** and grease is **medium**" has maximum strength 3/5.
- The above rule corresponds to the output MF $\mu_M(z)$. This mapping is shown in Fig. P. 4.10.2(c).
- To find out the final defuzzified value, we now take average (mean) of $\mu_M(z)$.

$$\begin{aligned}
 \mu_M(z) &= \frac{z - 10}{15} \quad \text{and} \quad \mu_M(z) = \frac{40 - z}{15} \\
 \therefore 3/5 &= \frac{z - 10}{15} \quad 3/5 = \frac{40 - z}{15} \\
 \therefore z &= 19 \quad z = 31 \\
 \therefore z^* &= \frac{19 + 31}{2} \\
 &= 25 \text{ min}
 \end{aligned}$$

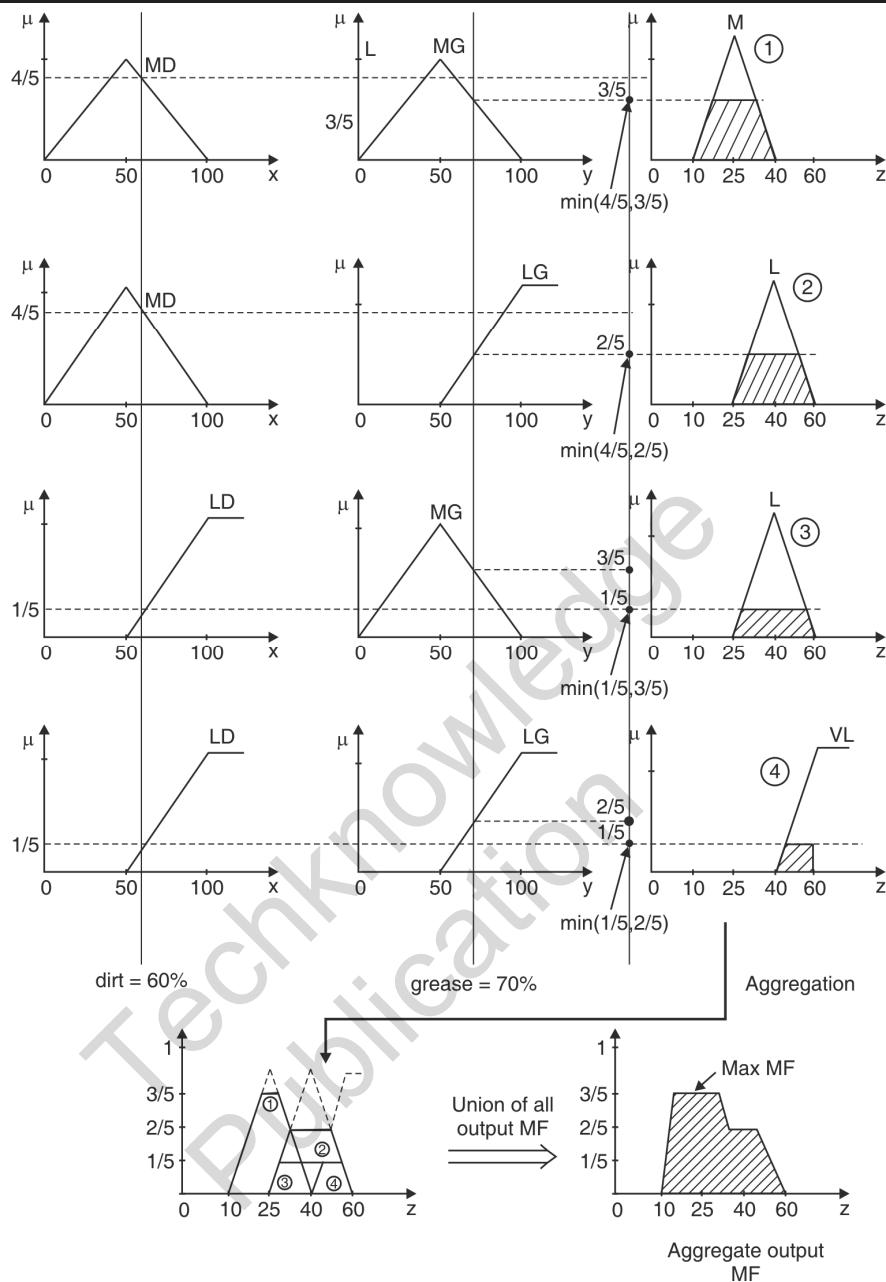


Fig. P. 4.10.2(d) : Process of rule evaluation and defuzzification

3. Water Purifier Controller

Ex. 4.10.3 : Design a fuzzy controller to control the feed amount of purifier for the water purification plant. Raw water is purified by injecting chemicals. Assume input as water temperature and grade of water. Output as amount of purifier. Use three descriptors for input and output variables. Design rules to control action and defuzzification. Design should be supported by figures whenever necessary. Clearly indicate that when temperature is low, grade is low then chemical used is in large amount.

MU - May 13, 20 Marks

Soln. :

Step 1 : Identify input and output variables and decide descriptors for the same.

- Here input variables are water temperature and grade of water.

- Water temperature is measured in °C. grade of water is measured in percentage.
- Descriptors for water temperature are

{C, M, H}	C - Cold
	M - Medium
	H - High

- Descriptors for grade are

{L, M, H}	L - Low
	M - Medium
	H - High

- Amount of purifier is measured in grams. Descriptors for amount of purifier are

{S, M, L}.	S - Small
	M - Medium
	L - Large

Step 2 : Fuzzification

Define membership functions for each of the input and output variables.

We use triangular MFs because of its simplicity.

(1) Membership functions for water temperature

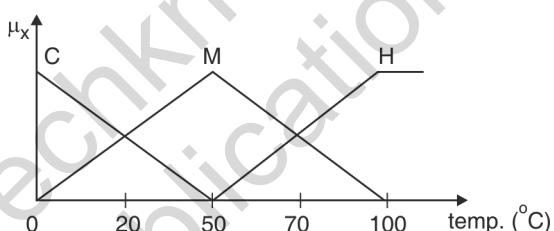


Fig. P. 4.10.3 : Membership functions for water temp.

$$\mu_C(x) = \frac{50-x}{50}, \quad 0 \leq x \leq 50$$

$$\mu_M(x) = \begin{cases} \frac{x}{50}, & 0 \leq x \leq 50 \\ \frac{100-x}{50}, & 50 < x \leq 100 \end{cases}$$

$$\mu_H(x) = \frac{x-50}{50}, \quad 50 \leq x \leq 100$$

(2) Membership functions for grade of water

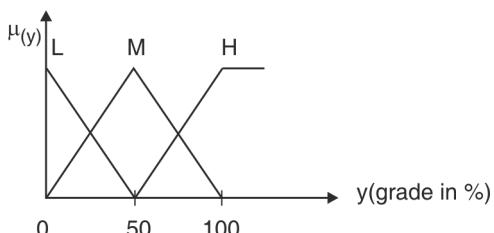


Fig. P. 4.10.3(a) : Membership functions for grade of water

$$\mu_L(y) = \frac{50-y}{50}, \quad 0 \leq y \leq 50$$

$$\mu_M(y) = \begin{cases} \frac{y}{50}, & 0 \leq y \leq 50 \\ \frac{100-y}{50}, & 50 < y \leq 100 \end{cases}$$

$$\mu_H(y) = \frac{y-50}{50}, \quad 50 \leq y \leq 100$$

(3) Membership functions for amount of purifier

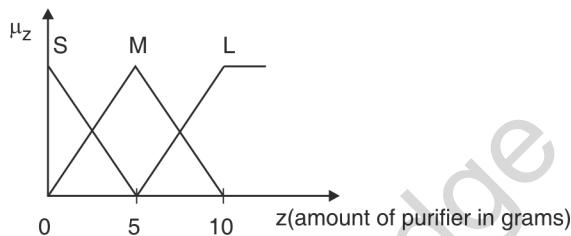


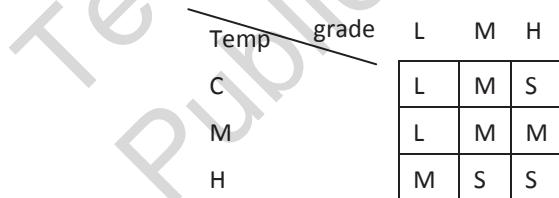
Fig. P. 4.10.3(b) : Membership functions for purifier

$$\mu_S(z) = \frac{5-z}{5}, \quad 0 \leq z \leq 5$$

$$\mu_M(z) = \begin{cases} \frac{z}{5}, & 0 \leq z \leq 5, \\ \frac{10-z}{5}, & 5 < z \leq 10 \end{cases}$$

$$\mu_L(z) = \frac{z-5}{5}, \quad 5 < z \leq 10$$

Step 3 : Form a rule base



- The above matrix represents in all nine rules. For example,
- First rule can be, “If temperature is cold and grade is low then amount of purifier required is large.”
- Similarly all nine rules can be defined using if-then rules.

Step 4 : Rule Evaluation

- Assume water temperature = 5° and grade = 30
- Water temperature = 5° maps to the following two MFs of “temperature” variable.

$$\mu_C(x) = \frac{50-x}{50} \text{ and } \mu_M(x) = \frac{x}{50}$$

- Similarly, grade = 30 maps to the following two MFs of “grade” variable.

$$\mu_L(y) = \frac{50-y}{50} \text{ and } \mu_M(y) = \frac{y}{50}$$

- Evaluate $\mu_c(x)$ and $\mu_M(x)$ for $x = 5^\circ$

We get,

$$\mu_c(5) = \frac{50-5}{50} = \frac{9}{10} = 0.9 \quad \dots(1)$$

$$\mu_M(5) = \frac{5}{50} = \frac{1}{5} = 0.1 \quad \dots(2)$$

- Evaluate $\mu_L(y)$ and $\mu_M(y)$ for $y = 30$

$$\mu_L(30) = \frac{50-30}{50} = \frac{2}{5} = 0.4 \quad \dots(3)$$

$$\mu_M(30) = \frac{30}{50} = \frac{3}{5} = 0.6 \quad \dots(4)$$

- The above four equation represents following for rules that we need to evaluate.
 1. If temperature is cold and grade is low.
 2. If temperature is cold and grade is medium.
 3. If temperature is medium and grade is low.
 4. If temperature is medium and grade is **medium**.
- Since the antecedent part of each rule is connected by **and** operator we use **min** operator to evaluate strength of each rule

$$\text{Strength of rule1 : } S_1 = \min(\mu_c(5), \mu_L(30)) = \min(0.9, 0.4) = 0.4$$

$$\text{Strength of rule2 : } S_2 = \min(\mu_c(5), \mu_M(30)) = \min(0.9, 0.6) = 0.6$$

$$\text{Strength of rule3 : } S_3 = \min(\mu_M(5), \mu_L(30)) = \min(0.1, 0.4) = 0.1$$

$$\text{Strength of rule4 : } S_4 = \min(\mu_M(5), \mu_M(30)) = \min(0.1, 0.6) = 0.1$$

Temp	Grade		Temp	Grade	
	$\mu_L(30)$	$\mu_M(30)$		$\mu_c(5)$	$\mu_M(5)$
$\mu_c(5)$	0.4	(0.6)	X	L	M
$\mu_M(5)$	0.1	0.1	X	L	M
	X	X	X	M	S

(a) Rule strength table

(b) Rule base table

Fig. P. 4.10.3(c) :Rule strength and its mapping to corresponding output MF

Step 5 : Defuzzification

Since, we use “mean of max” defuzzification technique, we first find the rule with maximum strength.

$$= \max(S_1, S_2, S_3, S_4) = \max(0.4, 0.6, 0.1, 0.1) = 0.6$$

- This corresponds to rule 2.

Thus rule 2 : “Temperature is cold and grade is medium” has maximum strength 0.6.

- The above rule corresponds to the output MF $\mu_M(z)$. This is show is shown in Fig. P.4.10.3(c).
- To find out final defuzzified value, we now take average (i.e. mean) of $\mu_M(z)$.

$$\begin{aligned}\mu_M(z) &= \frac{10-z}{5} \quad \text{and} \quad \mu_M(z) = \frac{z}{5} \\ 0.6 &= \frac{10-z}{5} \quad \therefore 0.6 = \frac{z}{5} \\ \therefore z &= 13 \quad \therefore z = 3 \\ \therefore z^* &= \frac{13+3}{2} \quad = 8 \text{ gms}\end{aligned}$$

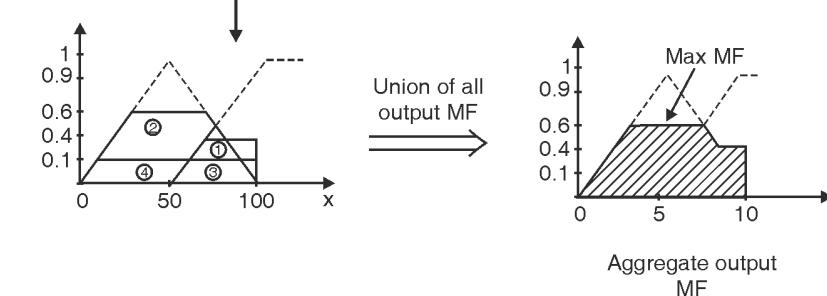
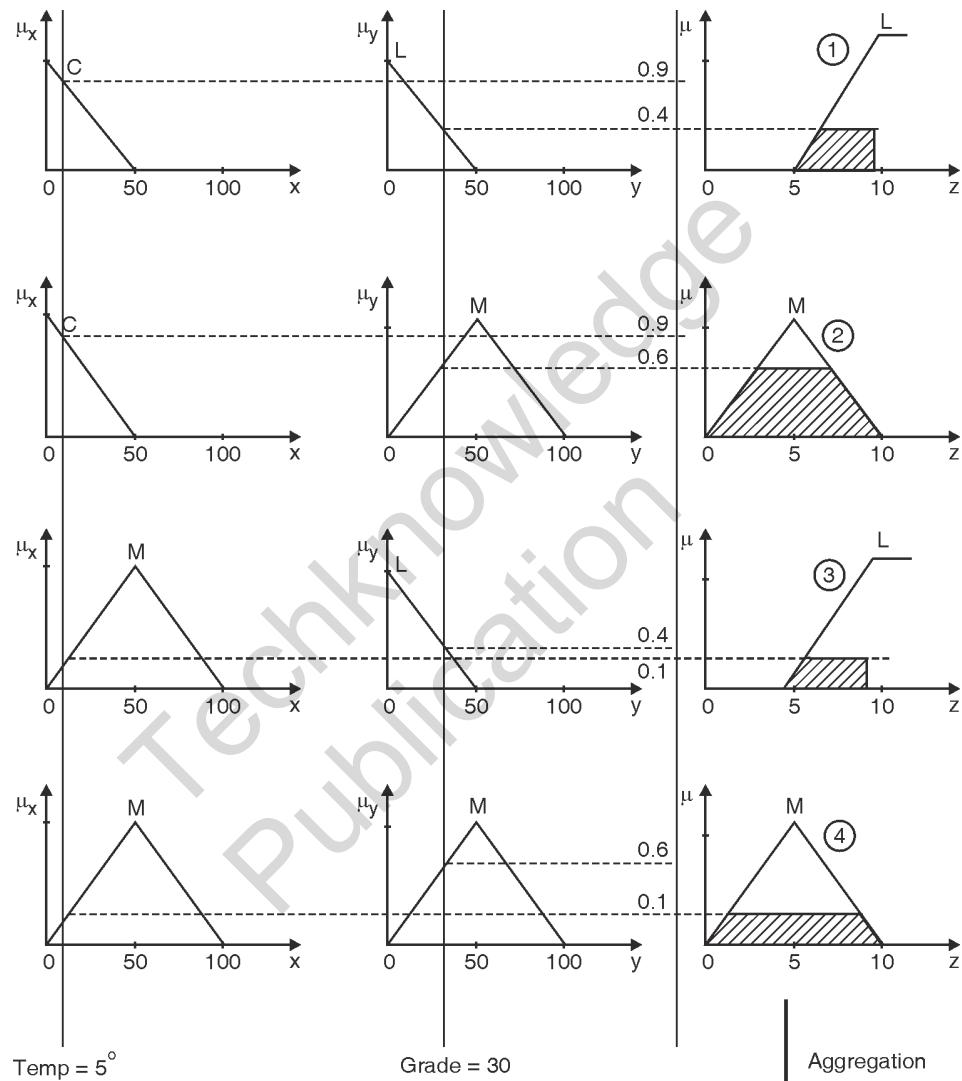


Fig. P. 4.10.3(d) : Process of rule evaluation and defuzzification

4. Train Brake Power Controller

- Ex. 4.10.4 :** Design a fuzzy controller for a train approaching or leaving a station. The inputs are distance from a station and speed of the train. The output is brake power used. Use,
- Triangular membership functions
 - Four descriptors for each variables
 - Five to six rules.
 - Appropriate defuzzification method.

Soln. :

Step 1 : Identify input and output variables and decide descriptors for the same.

- Here inputs are
 - Distance of a train from the station, measured in meters and
 - Speed of train measured in km/hr.
- Output variable is brake power measured in %.

As mentioned, we take four descriptors for each of the input and output variables.

For distance $\Rightarrow \{VSD, SD, LD, VLD\}$

- VSD : Very Short Distance
- SD : Short Distance
- LD : Large Distance
- VLD : Very Large Distance

- For speed $\Rightarrow \{VLS, LS, HS, VHS\}$

- VLS : Very Low Speed
- LS : Low Speed
- HS : High Speed
- VHS : Very High Speed

- For brake power $\Rightarrow \{VLP, LP, HP, VHP\}$

- VLP : Very Low Power
- LP : Low Power
- HP : High Power
- VHP : Very High Power

Step 2 : Define membership functions for each of the input and output variables.

- As mentioned, we use triangular membership functions.

1. Membership functions for distance

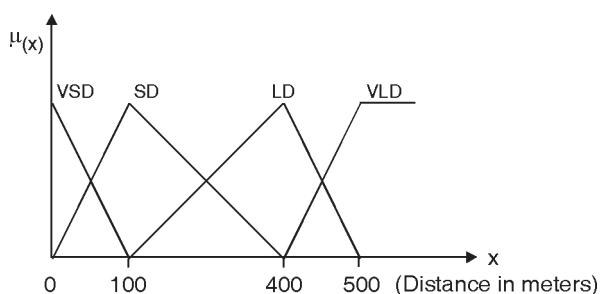


Fig. P. 4.10.4 : Membership functions for distance (distance in meters)

$$\begin{aligned}\mu_{VSD}(x) &= \frac{100-x}{100}, & 0 \leq x \leq 100 \\ \mu_{SD}(x) &= \left\{ \begin{array}{ll} \frac{x}{100}, & 0 \leq x \leq 100 \\ \frac{400-x}{300}, & 100 < x \leq 400 \end{array} \right. \\ \mu_{LD}(x) &= \left\{ \begin{array}{ll} \frac{x-100}{300}, & 100 \leq x \leq 400 \\ \frac{500-x}{100}, & 400 < x \leq 500 \end{array} \right. \\ \mu_{VLD}(x) &= \frac{x-400}{100}, & 400 \leq x \leq 500\end{aligned}$$

2. Membership functions for speed

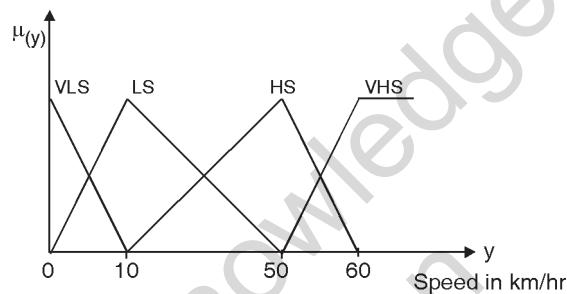


Fig. P. 4.10.4(a) : Membership functions for speed

$$\begin{aligned}\mu_{VLS}(y) &= \frac{10-y}{10}, & 0 \leq y \leq 10 \\ \mu_{LS} &= \left\{ \begin{array}{ll} \frac{y}{10}, & 0 \leq y \leq 10 \\ \frac{50-y}{40}, & 10 < y \leq 50 \end{array} \right. \\ \mu_{HS} &= \left\{ \begin{array}{ll} \frac{y-10}{40}, & 10 \leq y \leq 50 \\ \frac{60-y}{10}, & 50 < y \leq 60 \end{array} \right. \\ \mu_{VHS}(y) &= \frac{y-50}{10}, & 50 \leq y \leq 60\end{aligned}$$

3. Membership functions for brake power

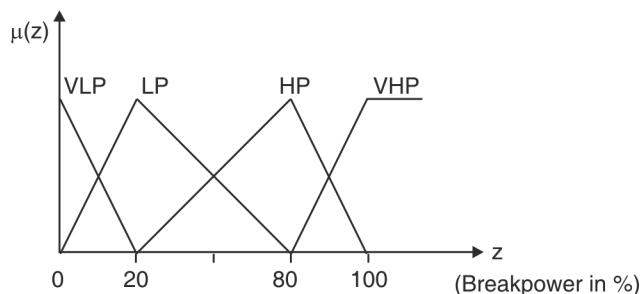


Fig. P. 4.10.4(b) : Membership functions for brake power



$$\begin{aligned}\mu_{VLP}(z) &= \frac{20-z}{20}, & 0 \leq z \leq 20 \\ \mu_{LP}(z) &= \left\{ \begin{array}{l} \frac{z}{20}, \\ \frac{80-z}{60}, \end{array} \right. & 0 \leq z \leq 20 \\ \mu_{HP}(z) &= \left\{ \begin{array}{l} \frac{z-20}{60}, \\ \frac{100-z}{20}, \end{array} \right. & 20 \leq z \leq 80 \\ \mu_{VHP}(z) &= \frac{z-80}{20}, & 80 \leq z \leq 100\end{aligned}$$

Step 3 : Form a rule base.

Dist	Speed	VLS	LS	HS	VHS
VSD		HP	HP	VHP	VHP
SD		LP	LP	HP	VHP
LD		VLP	VLP	LP	HP
VLD		VLP	VLP	LP	LP

The above matrix represents in all 16 rules.

For example, First rule can be “If distance of a train is Very Short (VSD) and speed is Very Low (VLS) then required brake power is High (HP)”.

Similarly all 16 rules can be defined using If then rules.

Step 4 : Rule Evaluation

Assume distance = 110 meters and speed = 52 km/hr

Distance = 110 maps to the following two MFs of “distance” variable.

$$\mu_{SD}(x) = \frac{400-x}{300} \text{ and } \mu_{LD}(x) = \frac{x-100}{300}$$

– Similarly speed = 52 maps to the following two MFs of “speed” variable.

$$\mu_{HS}(y) = \frac{60-y}{10} \text{ and } \mu_{VHS}(y) = \frac{y-50}{10}$$

– Evaluate $\mu_{SD}(x)$ and $\mu_{LD}(x)$ for $x = 110$, we get,

$$\mu_{SD}(110) = \frac{400-110}{300} = 0.96 \quad \dots(1)$$

$$\mu_{LD}(110) = \frac{110-100}{300} = 0.033 \quad \dots(2)$$

– Similarly evaluate $\mu_{HS}(y)$ and $\mu_{VHS}(y)$ for $y = 52$, we get,

$$\mu_{HS}(52) = \frac{60-52}{10} = 0.8 \quad \dots(3)$$

$$\mu_{VHS}(52) = \frac{52-50}{10} = 0.2 \quad \dots(4)$$

– The above four equations leads to the following for rules that we needs to evaluate.

1. Distance is short and speed is high
 2. Distance is short and speed is very high
 3. Distance is large and speed is high
 4. Distance is large and speed is very high
- Since the antecedent part of each rule is connected by **and** operator we use **min**.
- Operator to evaluate strength of each rule

$$\text{Strength of rule1: } S_1 = \min(\mu_{SD}(110), \mu_{HS}(52)) = \min(0.96, 0.8) = 0.8$$

$$\begin{aligned} \text{Strength of rule2 : } S_2 &= \min(\mu_{SD}(110), \mu_{VHS}(52)) \\ &= \min(0.96, 0.2) = 0.2 \end{aligned}$$

$$\begin{aligned} \text{Strength of rule3 : } S_3 &= \min(\mu_{LD}(110), \mu_{HS}(52)) \\ &= \min(0.033, 0.8) = 0.033 \end{aligned}$$

$$\begin{aligned} \text{Strength of rule: } S_4 &= \min(\mu_{LD}(110), \mu_{VHS}(52)) \\ &= \min(0.033, 0.2) = 0.033 \end{aligned}$$

		Speed			
		VLS	LS	HS	VHS
Distance	VSD	X	X	X	X
	SD	X	X	(0.8)	0.2
	LD	X	X	0.03	0.03
	VLD	X	X	X	X

		Speed			
		VLS	LS	HS	VHS
Distance	VSD				
	SD			(HP)	VHP
	LD			LP	HP
	VLD				

(a) Rule strength table

(b) Rule base table

Fig. P. 4.10.4 (c) : Rule strength table and its mapping to corresponding output MF

Step 5 : Defuzzification

- We use “mean of max” defuzzification technique.
 - We first find the rule with maximum strength
- $$\begin{aligned} &= \max(S_1, S_2, S_3, S_4) \\ &= \max(0.8, 0.2, 0.33, 0.2) = 0.8 \end{aligned}$$
- This corresponds to rule 1.
 - Thus rule 1 – “If dist is short and speed is high” has maximum strength 0.8.
 - The above rule corresponds to the output MF $\mu_{HP}(z)$. This mapping is shown in Fig. P. 4.10.4(c).
 - To compute the final defuzzified value, we take average of $\mu_{HP}(z)$.

$$\mu_{HP}(z) = \frac{z - 20}{60} \quad \mu_{HP}(z) = \frac{100 - z}{20}$$

$$\therefore 0.8 = \frac{z - 20}{60} \quad 0.8 = \frac{100 - z}{20}$$

$$\therefore z = 68 \quad \therefore z = 84$$

$$\therefore z^* = \frac{68 + 84}{2} = 76\%$$

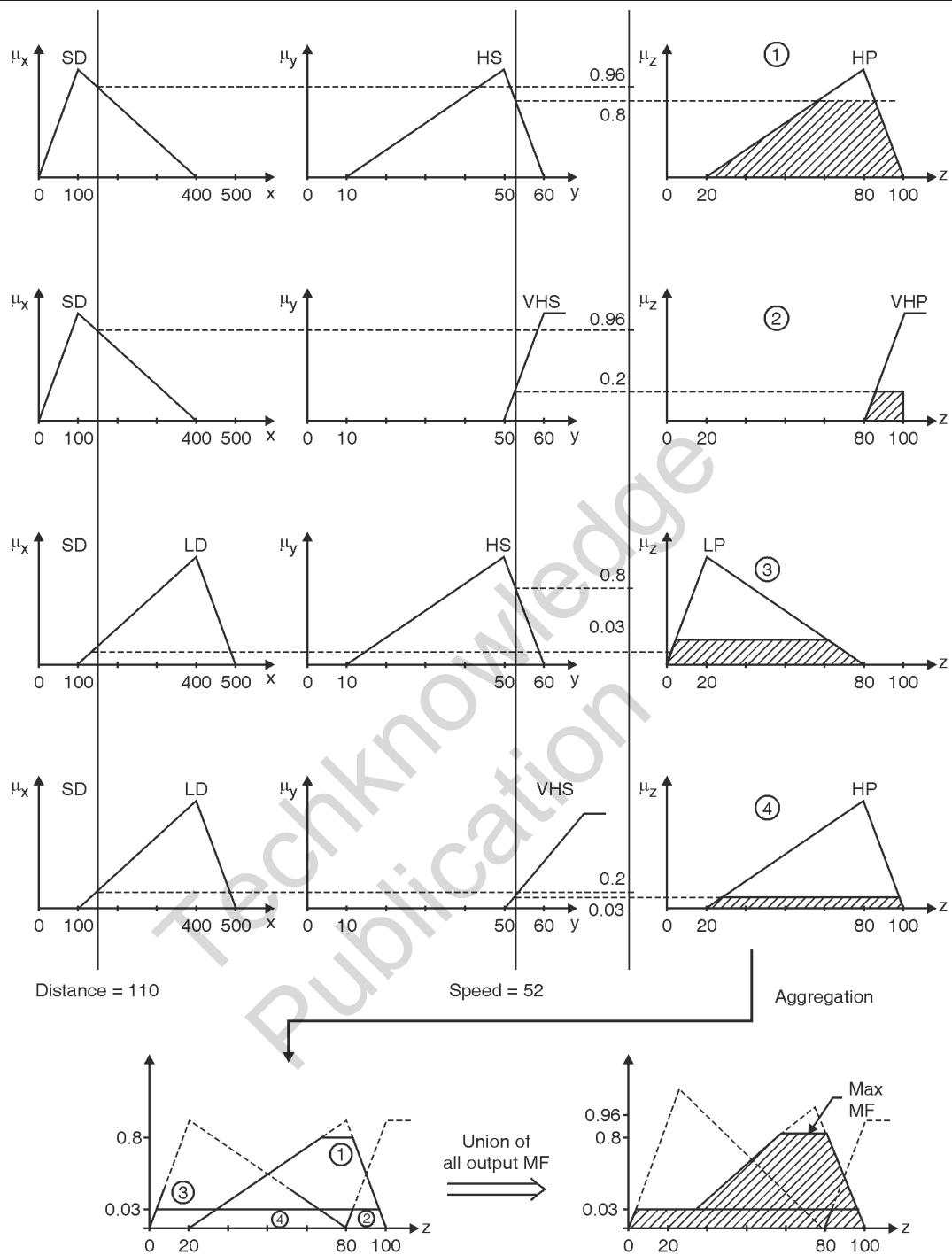


Fig. P. 4.10.4(d) : Process of rule evaluation and defuzzification

5. Water Tank Temperature Controller

Ex. 4.10.5 : Design a fuzzy controller for maintaining the temperature of water in the tank at a fixed level. Input variables are the cold water flow into the tank and steam flow into the tank. For cooling, cold water flow is regulated and for raising the temperature steam flow is regulated. Define the fuzzification scheme for input variables. Device a set of rules for control action and defuzzification. Formulate the control problem in terms of fuzzy inference rules incorporating the degree of relevance for each rule. Design a scheme which shall regulate the water and steam flows properly.

Soln. :

Step 1 : Identify input and out variables and decide descriptors for each variables.

Here inputs are,

1. Amount of valve opening for cold water.
 2. Amount of valve opening for steam.
- Valve opening is measured in degree from 0° to 180° . we take five descriptors for each of the input variables.
 - Descriptors for value opening for cold water flow are
{ELCV, LCV, CCV, RCV, ERCV}

ELCV	:	Extreme Left Cold Valve
LCV	:	Left Cold Valve
CCV	:	Centre Cold Valve
RCV	:	Right Cold Valve
ERCV	:	Extreme Right Cold Valve

Descriptors for valve opening of steam flow are {ELSV, LSV, CSV, RSV, ERSV}

ELSV	:	Extreme Left Steam Valve
LSV	:	Left Steam Value
CSV	:	Centre Steam Value
RSV	:	Right Steam Value
ERSV	:	Extreme Right Steam Value

- Output is temperature of water in the tank measured in $^\circ\text{C}$.

We use seven descriptors for temperature of water {VVCT, VCT, CT, WT, HT, VHT, VVHT}

VVCT	:	Very Very Cold Temperature
VCT	:	Very Cold Temperature
CT	:	Cold Temperature
WT	:	Warm Temperature
HT	:	Hot Temperature
VHT	:	Very Hot Temperature
VVHT	:	Very Very Hot Temperature

Step 2 : Define membership functions for each of the input and output variables.

1. Membership functions for valve opening for cold water flow

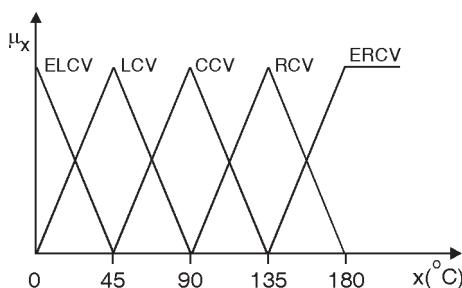


Fig. P. 4.10.5 : Membership functions for valve opening of cold water

$$\begin{aligned}
 \mu_{ELCV}(x) &= \frac{45-x}{45}, & 0 \leq x \leq 45 \\
 \mu_{LCV}(x) &= \frac{x}{45}, & 0 \leq x \leq 45 \\
 &\quad \left. \frac{90-x}{45} \right\}, & 45 < x \leq 90 \\
 \mu_{CCV}(x) &= \frac{x-45}{45}, & 45 \leq x \leq 90 \\
 &\quad \left. \frac{135-x}{45} \right\}, & 90 < x \leq 135 \\
 \mu_{RCV}(x) &= \frac{x-90}{45}, & 90 \leq x \leq 135 \\
 &\quad \left. \frac{180-x}{45} \right\}, & 135 < x \leq 180 \\
 \mu_{ERCV}(x) &= \frac{x-135}{45}, & 135 \leq x \leq 180
 \end{aligned}$$

2. Membership functions for valve opening for steam flow

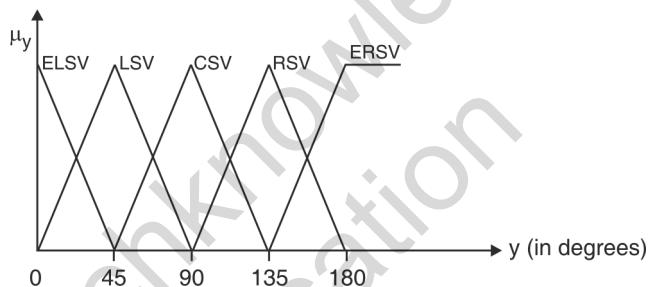


Fig. P.4.10.5(a) : Membership functions for value opening for steam flow

$$\begin{aligned}
 \mu_{ELSV}(y) &= \frac{45-y}{45}, & 0 \leq y \leq 45 \\
 \mu_{LSV}(y) &= \frac{y}{45}, & 0 \leq y \leq 45 \\
 &\quad \left. \frac{90-y}{45} \right\}, & 45 < y \leq 90 \\
 \mu_{ESV}(y) &= \frac{y-45}{45}, & 45 \leq y \leq 90 \\
 &\quad \left. \frac{135-y}{45} \right\}, & 90 < y \leq 135 \\
 \mu_{RSV}(y) &= \frac{y-90}{45}, & 90 \leq y \leq 135 \\
 &\quad \left. \frac{180-y}{45} \right\}, & 135 < y \leq 180 \\
 \mu_{ERSV}(y) &= \frac{y-135}{45}, & 135 \leq y \leq 180
 \end{aligned}$$

3. Membership functions for temperature of a water in tank

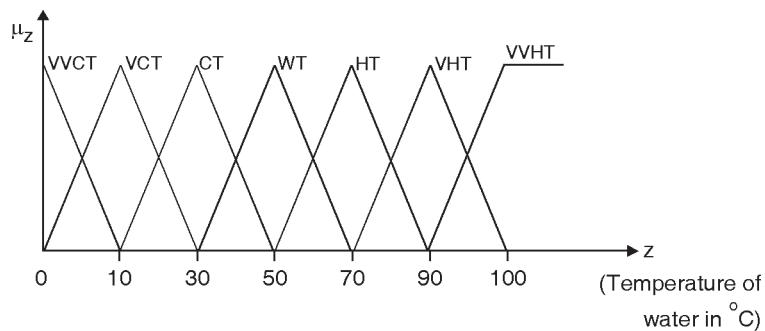


Fig. P.4.10.5(b): Membership functions for temperature of water in tank

$$\begin{aligned}
 \mu_{VVCT}(z) &= \frac{10-z}{10}, & 0 \leq z \leq 10 \\
 \mu_{VCT}(z) &= \frac{z}{10}, & 0 \leq z \leq 10 \\
 &\quad \left. \frac{30-z}{20} \right\}, & 10 < z \leq 30 \\
 \mu_{CT}(z) &= \frac{z-10}{20}, & 10 \leq z \leq 30 \\
 &\quad \left. \frac{50-z}{20} \right\}, & 30 < z \leq 50 \\
 \mu_{WT}(z) &= \frac{z-30}{20}, & 30 \leq z \leq 50 \\
 &\quad \left. \frac{70-z}{20} \right\}, & 50 < z \leq 70 \\
 \mu_{HT}(z) &= \frac{z-50}{20}, & 50 \leq z \leq 70 \\
 &\quad \left. \frac{90-z}{20} \right\}, & 70 < z \leq 90 \\
 \mu_{VHT}(z) &= \frac{z-70}{20}, & 70 \leq z \leq 90 \\
 &\quad \left. \frac{100-z}{10} \right\}, & 90 < z \leq 100 \\
 \mu_{VVHT}(z) &= \frac{z-90}{10}, & 90 \leq z \leq 100
 \end{aligned}$$

Step 3 : Form a rule base.

x	Y	ELSV	LSV	CSV	RSV	ERSV
ELCV		WT	WT	HT	VHT	VVHT
LCV		CT	WT	HT	VHT	VVHT
CCV		VCT	CT	WT	HT	VHT
RCV		VCT	VCT	CT	WT	HT
ERCV		VVCT	VVCT	VCT	CT	WT

Step 4 : Rule evaluation

Assume that $x = 95^\circ$ (valve opening of cold water)

$y = 50^\circ$ (valve opening for steam flow)

- Here $x = 95^\circ$ maps to the following two MFs of variable x :

$$\mu_{CCV}(x) = \frac{135-x}{45} \text{ and } \mu_{RCV}(x) = \frac{x-90}{45}$$

- Similarly $y = 50^\circ$ maps to the following two MFs of variable y :

$$\mu_{LSV}(y) = \frac{90-y}{45} \text{ and } \mu_{CSV}(y) = \frac{y-45}{45}$$

- Evaluate $\mu_{CCV}(x)$ and $\mu_{RCV}(x)$ for $x = 95^\circ$ we get,

$$\mu_{CCV}(95) = \frac{135-95}{45} = 0.88 \quad \dots(1)$$

$$\mu_{RCV}(95) = \frac{95-90}{45} = 0.11 \quad \dots(2)$$

- Evaluate $\mu_{LSV}(y)$ and $\mu_{CSV}(y)$ for $y = 50^\circ$, we get

$$\mu_{LSV}(50) = \frac{90-50}{45} = 0.88 \quad \dots(3)$$

$$\mu_{CSV}(50) = \frac{50-45}{45} = 0.11 \quad \dots(4)$$

Above four equations lead to the following four rules that we need to evaluate.

- Cold water valve is in **center** and steam flow valve is in **left**.
- Cold water valve is in **center** and steam flow valve is in **center**.
- Cold water valve is in **right** and steam flow valve is in **left**.
- Cold water valve is in **right** and steam flow valve is in **center**.

Table P.4.10.5 shows rule strength table.

Table P. 4.10.5 : Rule strength table

	μ_{ELSV}	μ_{LSV}	μ_{CSV}	μ_{RSV}	μ_{ERSV}
μ_{ELSV}	X	X	X	X	X
μ_{LSV}	X	X	X	X	X
μ_{CSV}	X	(0.88)	0.11	X	X
μ_{RSV}	X	0.11	0.11	X	X
μ_{ERSV}	X	X	X	X	X

Step 5 : Defuzzification

- We find the rule with maximum strength

$$= \max (0.88, 0.11, 0.11, 0.11)$$

$$= 0.88$$

Which corresponds to rule 1.

- The rule 1 corresponds to the output MF $\mu_{CT}(z)$. To compute final defuzzified value we take average of $\mu_{CT}(z)$.



$$\begin{aligned}
 \mu_{CT}(z) &= \frac{z-10}{20} \Rightarrow 0.88 \\
 &= \frac{z-10}{20} \Rightarrow z = 27.7 \\
 \mu_{CT}(z) &= \frac{50-z}{20} \Rightarrow 0.88 \\
 &= \frac{50-z}{20} \Rightarrow z = 32.3 \\
 \therefore z^* &= \frac{27.7 + 32.3}{2} \\
 &= 30^\circ\text{C}
 \end{aligned}$$

Review Questions

- Q. 1** Explain support and core of a fuzzy set with examples.
- Q. 2** Model the following as fuzzy set using trapezoidal membership function : "Numbers close to 10".
- Q. 3** Using Mamdani fuzzy model, Design a fuzzy logic controller to determine the wash time of a domestic washing machine. Assume that the inputs are dirt and grease on cloths. Use three descriptors for each input variables and five descriptor for the output variable. Derive a set of rules for control action and defuzzification. The design should be supported by figures wherever possible.
- Q. 4** Let $A = \{a_1, a_2\}$, $B = \{b_1, b_2, b_3\}$, $C = \{c_1, c_2\}$
Let R be a relation from A to B defined by matrix :

	b₁	b₂	b₃
a₁	0.4	0.5	0
a₂	0.2	0.8	0.2

Let S be a relation from B to C defined by matrix :

	c₁	c₂
b₁	0.2	0.7
b₂	0.3	0.8
b₃	1	0

Find (i) Max-min composition of R and S

(ii) Max-products composition of R and S.

- Q. 5** Define Supports, Core, Normality, Crossover points and α – cut for fuzzy set.
- Q. 6** High speed rail monitoring devices sometimes make use of sensitive sensors to measure the deflection of the earth when a rail car passes. These deflections are measured with respect to some distance from the rail car and, hence are actually very small angles measured in micro-radians. Let a universe of deflection be $A = [1, 2, 3, 4]$ where A is the angle in micro-radians, and let a universe of distance be $D = [1, 2, 5, 7]$ where D is distance in feet, suppose a relation between these two parameters has been determined as follows :



R =		D ₁	D ₂	D ₃	D ₄
A ₁	1	0.3	0.1	0	
A ₂	0.2	1	0.3	0.1	
A ₃	0	0.7	1	0.2	
A ₄	0	0.1	0.4	1	

Now let a universe of rail car weights be $W = [1, 2]$, where W is the weight in units of 100,000 pounds. Suppose the fuzzy relation of W to A is given by,

S =		W ₁	W ₂
A ₁	1	0.4	
A ₂	0.5	1	
A ₃	0.3	0.1	
A ₄	0	0	

Using these two relations, find the relation $R^T \circ S = T$.

- (a) Using max-min composition. (b) Using max-product composition.

- Q. 7** Design a fuzzy logic controller for a train approaching or leaving a station. The inputs are the distance from the station and speed of the train. The output is the amount of brake power used. Use four descriptors for each variable use Mamdani Fuzzy model.
- Q. 8** Explain the Fuzzy Inference System in detail.
- Q. 9** Explain the different Fuzzy membership function.
- Q. 10** Explain any four defuzzification methods with suitable Example.
- Q. 11** State the different properties of Fuzzy set.
- Q. 12** Determine all α - level sets and strong α - level sets for the following fuzzy set.
 $A = \{(1, 0.2), (2, 0.5), (3, 0.8), (4, 1), (5, 0.7), (6, 0.3)\}$.
- Q. 13** Design a fuzzy controller to determine the wash time of a domestic washing machine. Assume that the inputs are dirt and grease on clothes. Use three descriptors for each I/P variable. Device a set of rules for control action and defuzzification. The design should be supported by figures wherever possible. Clearly indicate that if the clothes are soiled to a larger degree the wash time required will be more.
- Q. 14** Explain different methods of defuzzification.
- Q. 15** Explain cylindrical extension and projection operations on fuzzy relation with example.
- Q. 16** Model the following as fuzzy set using trapezoidal membership function “Middle age”.

Q. 17 For the given membership function as shown in Fig. Q. 17, determine the defuzzified output value by any 2 methods.

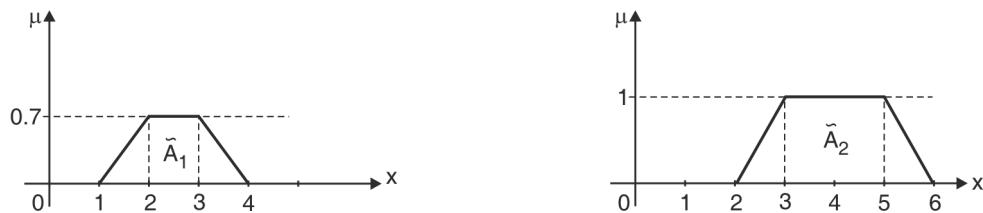


Fig. Q. 17

Q. 18 Compare Mamdani and Sugeno fuzzy models.

Q. 19 Design fuzzy logic controller for water purification plant. Assume the grade of water and temperature of water as the inputs and the required amount of purifier as the output. Use three descriptors for input and output variables. Derive set of rules for control the action and deffuzzification. The design should be supported by figures. Clearly indicate that if water temperature is low and grade of water is low, then amount of purifier required is large.

Q. 20 Discuss fuzzy composition techniques with suitable example.

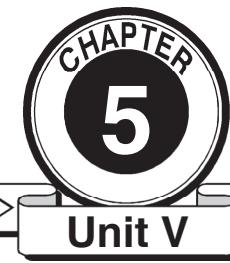
Q. 21 Two fuzzy relations are given by

$$R = \begin{matrix} & y_1 & y_2 \\ \begin{matrix} x_1 \\ x_2 \end{matrix} & \left[\begin{matrix} 0.6 & 0.3 \\ 0.2 & 0.9 \end{matrix} \right] \end{matrix}$$
$$S = \begin{matrix} & z_1 & z_2 & z_3 \\ \begin{matrix} y_1 \\ y_2 \end{matrix} & \left[\begin{matrix} 1 & 0.5 & 0.3 \\ 0.8 & 0.4 & 0.7 \end{matrix} \right] \end{matrix}$$

Obtain fuzzy relation T as a max-min composition and max-product composition between the fuzzy relations.

Q. 22 Describe in detail the formation of inference rules in a Mamdani Fuzzy inference system.





Artificial Neural Network

Syllabus

- 5.1 Introduction : Fundamental concept : Basic Models of Artificial Neural Networks : Important Terminologies of ANNs : McCulloch-Pitts Neuron
- 5.2 Neural Network Architecture: Perceptron, Single layer Feed Forward ANN, Multilayer Feed Forward ANN, Activation functions, Supervised Learning : Delta learning rule, Back Propagation algorithm.
- 5.3 Un-Supervised Learning algorithm : Self Organizing Maps

5.1 Fundamental Concept - Artificial Neural Networks

5.1.1 Artificial Neural Networks

- Artificial Neural Networks (ANNs) are simplified models of the biological nervous system. They basically mimic the working of a human brain.
- An ANN, in general, is a highly interconnected network of a large number of processing elements called neurons.
- An ANN can be considered as a highly parallel network. Distributed processing is a typical feature of a neural network.
- Neural networks work on the principle of learning by examples. They are presented with known examples of a problem called ‘training set’, to acquire knowledge about the problem. After training, the network can be effectively employed in solving instances of the problem previously unknown to the network.

Architecture of a simple artificial neural net

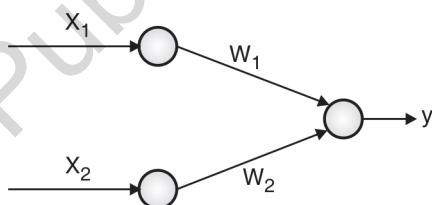


Fig: 5.1.1 Architecture of a simple artificial neural net

- As shown in Fig 5.1.1, There are two input neurons and one output neuron.
- Each neuron has an internal state of its own called ‘activation of a neuron’.
- The activation signal of one neuron is transmitted to another neuron.
- This Activation of a neuron can be considered as a function of the inputs a neuron receives.
- Consider a set of neurons; say X_1 and X_2 , transmitting signals to another neuron, Y .
- As shown in Fig 5.1.1 there are two input neurons X_1 and X_2 . They transmit signals to output neuron Y .
- Input neurons X_1 and X_2 are connected to the output neuron Y , over a weighted interconnection links (W_1 and W_2).
- For the above simple neural net architecture, the net input is calculated as :

$$Y_{in} = W_1 X_1 + W_2 X_2$$

Where x_1 and x_2 are activations of the input neurons x_1 and x_2 respectively.

- The output y of the output neuron Y can be obtained by applying activations over the net input, i.e.,

$$y = f(y_{in})$$

- The function that we apply over the net input is called activation function.

5.1.2 Biological Neural Networks

- The fundamental computing element in the human brain is called a neuron. There are approximately 10^{11} neurons in a human brain.
- Communication between neurons happens through a connection network of **axons** and **synapses**. There are approximately 10^4 synapses per neuron.
- Neurons communicate with each other by means of electrical impulses.
- A chemical environment surrounds the network of neurons.

Information flow in nervous system

- The huge neural network in the brain is made up of extremely complex interconnections and has a very complicated structure.
- Fig. 5.1.2 shows information flow in the human nervous system.
- As shown in Fig. 5.1.2, sensory receptors provide input to the network. Receptors deliver stimuli both from within the body and from sensory organs, when the stimuli originate in the external world.
- The stimuli are in the form of electrical impulses that convey information into the network of neurons.

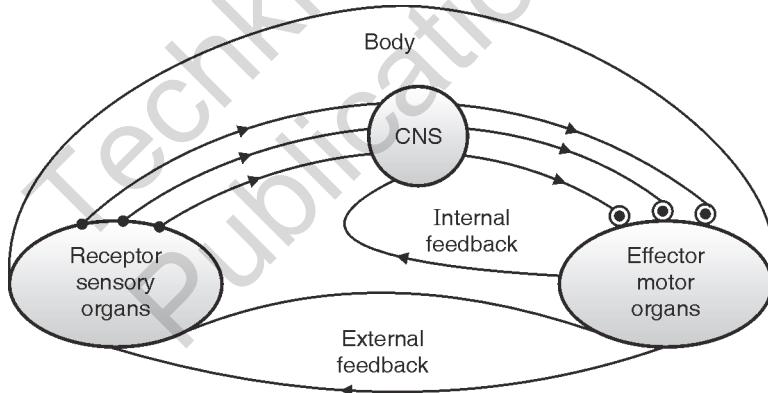


Fig. 5.1.2: Flow of information in the nervous system

- As shown in Fig. 5.1.2, effectors are controlled on account of information processing in the Central Nervous System (CNS). These effectors then produce human responses in the form of various actions.
- When necessary, commands are generated and transmitted to the motor organs.
- Motor organs are monitored in the CNS by both internal and external feedback.
- The overall nervous system structure has the characteristics of a **closed loop system**.

Biological Neuron

- A typical cell has 3 major regions: the cell body, called the soma, the axon and the dendrites. A schematic of a typical biological neuron is shown in Fig.5.1.3.
- Dendrites form a dendrite tree, which is a very fine bush of thin fibres around the neuron's body.

- Dendrites receive information from neurons through axons.
- An axon is a long cylindrical fibre that carries impulses from the neuron.
- Towards the end, an axon splits into fine branches. Each branch of an axon terminates in a small end bulb, almost touching the dendrites of neighbouring neurons.
- The contact between an axon and a dendrite is called a **synapse**. Via the synapse, a neuron introduces its signal to a neighbouring neuron. The receiving neuron either generates an impulse to its axon or produces no responses. The neuron is able to respond to the total of its inputs aggregated within a short time interval called the **period of latent summation**.
- The neuron generates a response and sends it to its axon only if the conditions necessary for firing are fulfilled.

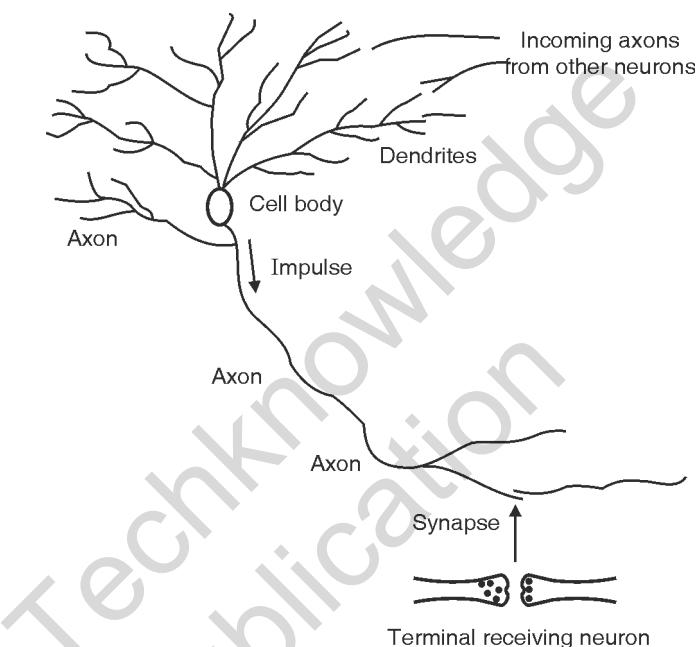


Fig. 5.1.3 : Neuron-schematic diagram

- Incoming impulses can be **excitatory** if they cause the firing, or **inhibitory** if they hinder the firing of the response.
- The condition for firing is that the excitation should exceed the inhibition by the amount called the **threshold** of the neuron, typically a value of about 40 mV.
- The incoming impulse to a neuron can only be generated by neighbouring neurons and by the neuron itself.
- Hence, impulses that are closely spaced in and arrive synchronously are more likely to cause the neuron to fire.
- The characteristic feature of the biological neuron is that the signals generated do not differ significantly in magnitude; the signal in the nerve fibre is either absent or has the maximum value.
- The information is transmitted between the nerve cells by means of binary signals.
- After an axon fibre finishes generating a pulse, it goes into a state of total non-excitability for a certain amount of time, called the **refractory period**. The nerve does not conduct any signals during the refractory period, even if the excitation intensity is very high.
- The time for modelling biological neurons is of the order of milliseconds. However, the refractory period is not uniform over the cells.

5.1.3 Brain vs. Computer

Human brain and computer can be compared based on following parameters.

	Biological Neuron	Artificial Neuron
Speed	Low compared to ANN. The speed is in milliseconds. Human Brain may fatigue if too much information and stress is presented.	Speed is High (in nanoseconds). They do not experience 'Fatigue'.
Size	The no. of neurons in Brains is approximately 10^{11} and total no. of connections are 10^{15} . Thus the complexity of human brain is very high .	The size of ANN depends on the application (usually hundreds or thousands) chosen and the developer who develops the application.
Complexity	Can perform Massive parallel computation simultaneously.	Can also perform Massive parallel computation simultaneously but much more faster than Human brain.
Storage capacity	Stores the information in the synapse.	Stores the information in continuous memory locations.
Tolerance	Biological neuron networks due to their topology are fault-tolerant. Information is stored redundantly so minor failures will not result in memory loss.	Artificial neural networks are not modelled for fault tolerance or self-regeneration.
Control Mechanism	No specific control mechanism external to the computing task.	There is a control unit for controlling computing activities
Power consumption	The brain consumes about 20% of all the human body's energy. An adult brain operates on about 20 watts.	A single Nvidia GPU runs on 250 watts alone, and requires a power supply. Machines are way less efficient than biological systems.
Learning	In human brain, Brain fibres grow and reach out to connect to other neurons; neuroplasticity allows new connections to be created and synapses may strengthen or weaken based on their importance	Artificial neural networks in the other hand, have a predefined model, where no further neurons or connections can be added or removed

Comparison between Biological Neuron and ANN

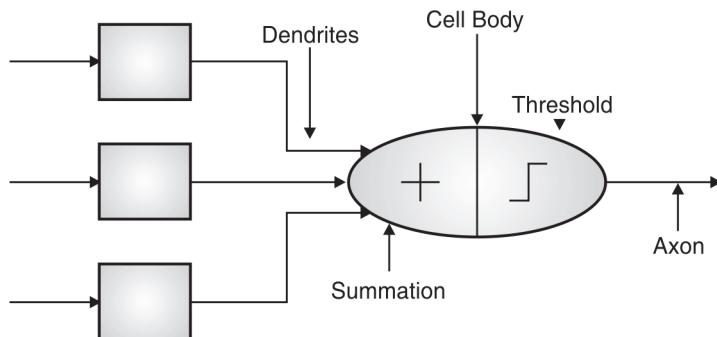


Fig. 5.1.4 : Analogy between biological neuron and artificial neuron

- Fig. 5.1.4 shows analogy between Biological and artificial Neuron.

- The dendrites in the Biological Neural Network are analogous to the weighted inputs based on their synaptic interconnection in the Artificial Neural Network.
- The cell body is comparable to the artificial neuron unit in the ANN which also comprises of summation and threshold unit.
- Axon carries output that is analogous to the output unit in case of Artificial Neural Network. So, ANN is modeled using the working of basic biological neurons.

5.1.4 Features of ANN

1. **Adaptive learning** : An ANN possess the ability to learn from the existing environment and also adapt to the new environment.
2. **Self-organization** : An ANN can create its own organization or representation of the information it receives during learning time.
3. **Real-time operation** : Since ANN computations may be carried out in parallel, they can be used for real time applications. Special hardware devices are being designed and manufactured to take advantage of this capability of ANNs and to reduce the response time.
4. **Fault tolerance** : Neural networks are fault tolerant in the sense even if some portions of the neural net is removed (For example some connections are removed), there will be only a small degradation in the neural network performance.
5. **Generalization** : After learning from the available inputs and their relationships, ANN has capability to infer unseen relationships on unseen data, thus making the model generalize.
6. **Non-linearity** : ANNs have the ability to learn and model non-linear and complex relationships. In most of the real-life problems, many of the relationships between inputs and outputs are non-linear as well as complex.
7. **Parallel distributed processing** : ANNs have massive parallelism which makes them very efficient.

5.1.5 Disadvantages of ANN

1. The best-known disadvantage of Neural Networks is their “black box” nature. This means that you don’t know how and why your NN came up with a certain output. For example, when you put in an image of a cat into a neural network and it predicts it to be a car, it is very hard to understand what caused it to come up with this prediction.
2. Neural Networks usually require much more data than traditional Machine Learning algorithms, as in at least thousands if not millions of labeled samples.
3. Computationally Expensive : Usually, Neural Networks are also more computationally expensive than traditional algorithms. State of the art deep learning algorithms can take several weeks to train completely from scratch. Most traditional Machine Learning Algorithms take much less time to train.

5.1.6 Applications of ANN

Neural networks have been successfully applied to a broad spectrum of data-intensive applications. Few of them are listed below.

1. **Forecasting**

Neural network can be used very effectively in forecasting exchange rates, predicting stock values, inflation and cash forecasting, forecasting weather conditions etc. Researchers have proved that the forecasting accuracy of NN systems tend to excel over that of the linear regression model.

2. Image compression

- Digital images require a large amount of memory for storage. As a result, the transmission of image from one computer to another can be very expensive in terms of time and bandwidth required.
- With the explosion of Internet, more sites are using images. Image compression is a technique that removes some of the redundant information present in the image without affecting its perceptibility, thus, reducing the storage size required to store the image.
- NN can be effectively used to compress the image. Several NN techniques such as Kohonen's self-organizing maps, Back propagation algorithm, Cellular neural network etc. can be used for image compression.

3. Industrial process control

- Neural networks have been applied successfully in industrial process control of dynamic systems.
- Neural networks (especially multi-layer perceptrons) have been proved to be the best choice for modelling non-linear systems and implementing general-purpose non-linear controllers, due to their universal approximation capabilities. For example control and management of agriculture machinery.

4. Optical Character Recognition

- Well known application using image recognition is the Optical Character Recognition (OCR) tools that are available with the standard scanning software for the home computer.
- **Scansoft** has had great success in combining NN with a rule based system for correctly recognizing both characters and words, to get a high level of accuracy.

5. Customer Relationship Management

- Another popular application for NN is Customer Relationship Management (CRM).
- Customer Relationship Management requires key information to be derived from raw data collected for each individual customer. This can be achieved by building models using historical data information.
- Many companies are now using neural technology to help in their day to day business processes. They are doing this to achieve better performance, greater insight, faster development and increased productivity.
- By using Neural Networks for data mining in the databases, patterns can be identified for the different types of customers, thus giving valuable customer information to the company.
- Also, NN could be useful for important tasks related to CRM, such as forecasting call centre loading, demand and sales levels, monitoring and analysing the market, validating, completing and enhancing databases, clustering and profiling client base etc.
- One example is the airline reservation system AMT, which could predict sales of tickets in relation to destination, time of year and ticket price.

6. Medical science

- Medicine is the field that has always taken benefits from the latest and advanced technologies.
- Artificial Neural Networks (ANN) is currently the next promising area of interest in medical science.
- It is believed that neural networks (also Deep networks) will have extensive application to biomedical problems in the next few years.
- ANN has already been successfully applied in medical applications such as diagnostic systems, bio chemical analysis, disease detection, image analysis and drug development.



5.2 Basic Models of Artificial Neural Networks

The basic model of ANN is specified by three basic entities.

1. The Model's synaptic connections.
2. The training or learning rules for updating weights.
3. The activation functions.

5.2.1 Connections

The arrangement of neurons in layers and connection between them defines the neural network architecture. There are four basic types of neuron connection architectures. They are:

1. Single Layer feed forward network
2. Multi-layer feed forward network
3. Single layer Recurrent network
4. Multi-layer Recurrent network

[We will discuss above points in Section 5.5]

5.2.2 Learning

MU - Dec. 11, May 12, Dec. 13, Dec. 14

Q. What is learning ? Compare different learning rules.	(Dec. 11, 10 Marks)
Q. What is learning in Neural Networks ? Compare different learning rules.	(May 12, 10 Marks)
Q. What is learning in neural networks ? Differentiate between supervised and unsupervised learning.	(Dec. 13, Dec. 14, 10 Marks)

- Learning is a relatively permanent change in behaviour brought about by experience.
- Learning in a network is a process that forces the network to yield a particular response to a specific input.
- There are two types of learning used to train the neural network: **batch learning** and **incremental learning**.
- Batch learning takes place when the network weights are adjusted in a single training step.
- In this mode, the complete set of input training data is needed to determine weights. Feedback information produced by the network is not involved in developing the network. This learning technique is called **recording**.
- **Incremental learning** is most commonly used and can be broadly classified into three basic types.

5.2.2(A) Supervised Learning

MU - May 13, Dec. 13

Q. Explain with neat diagram supervised learning.	(Dec. 13, 3 Marks)
Q. What is supervised learning ?	(May 13, 3 Marks)

- In this type of learning, each **input pattern has a corresponding output pattern associated with it**, which is the target or the desired pattern.
- Here, a comparison is made between the actual output of the network and the desired output to **find out the 'error'**.
- The computed error can be used to adjust network parameters (like connection weights, threshold etc.). As the network parameters are modified, the performance of the network improves.
- The training could continue until the **network is able to produce the expected or desired response**.

- As shown in Fig. 5.2.1, the distance function $\rho[d, o]$ takes two values as an input. Actual network output o and desired output d for an input X and then computes error measure.
- Since we have assumed adjustable weights, the weights can be adjusted to improve network performance that is to reduce the error.

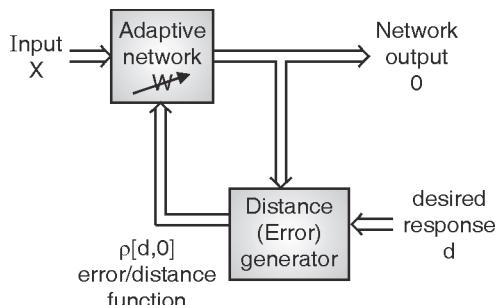


Fig. 5.2.1 : Block diagram of supervised learning

- This is analogous to classroom learning with the teacher's questions answered by students and corrected, if needed by the teacher.

5.2.2(B) Unsupervised Learning

MU - Dec. 12, May 13, Dec. 15

- | | |
|--|--------------------|
| Q. Explain with neat diagram unsupervised learning. | (Dec. 12, 3 Marks) |
| Q. What is unsupervised learning ? Compare different learning rules. | (May 13, 6 Marks) |
| Q. Distinguish between supervised and un-supervised learning. | (Dec. 15, 5 Marks) |

- In this learning method, the **desired output** is not presented to the network. It is as if there is **no teacher** to present the desired or expected output. Hence the system learns on its own by **discovering** and adapting to structural features present in the input pattern.

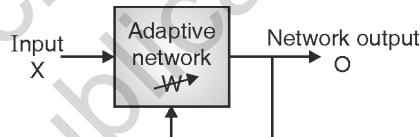
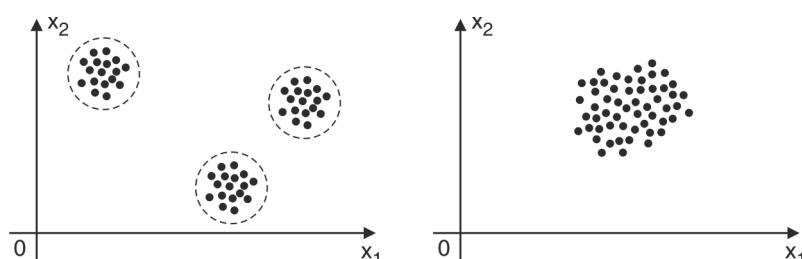


Fig. 5.2.2 : Block diagram of unsupervised learning

- Since no information is available to correct the network, learning must somehow be accomplished based on the observations of responses to inputs.
- For example, unsupervised learning can be used for finding the boundary between classes of input patterns distributed as shown in Fig. 5.2.3.
- In favourable case, as shown in Fig. 5.2.3 (a), cluster boundaries can be found based on the large and representative sample of inputs.



(a) Clustered

(b) Unclustered

Fig. 5.2.3 : Two dimensional patterns



- Since no external instructions regarding potential clusters is available, suitable weights self-adaption mechanism needs to be embedded in the trained network. One possible network adaptation rule that can be applied is “A pattern added to a cluster has to be closer to the cluster centre than to the other clusters’ centres”.
- Thus in unsupervised learning, the network must itself discover possibly existing patterns, regularities or separating properties.
- This type of learning is analogous to learning the subject from a videotape lecture covering the material but not including any other teacher’s involvement. Here the teacher delivers the lecture but is not available for clarification of unclear questions or to check answers.

Table 5.2.1 : Difference between supervised and unsupervised learning

Sr. No.	Supervised Learning	Unsupervised Learning
1.	Supervised Learning is that type of learning where the desired output is known	Unsupervised Learning is a type of learning where the desired output is unknown
2.	A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples.	Unsupervised learning uses procedures that attempt to find natural partitions of patterns given to the neural network.
3.	Supervised learning is achieved by means of classification or regression	Unsupervised learning is achieved by means of clustering
4.	Supervised learning is known as learning with a teacher’s presence as the desired output is known beforehand	Unsupervised learning is known as learning without a teacher’s presence as desired output is unknown
5.	Supervised learning is limited to learning small and simple models	With unsupervised learning it is possible to learn larger and more complex models than with supervised learning
6.	Perceptron learning, Delta learning, Widrow-Hoff learning, are examples of supervised learning algorithms	Hebbian learning, Winner-take-all learning are examples of unsupervised learning algorithms
7.	Example: Given several images of faces and non-faces (i.e. labelled data), a supervised algorithm will eventually learn and be able to predict whether or not an unseen image is a face.	Example: Given several images and there is no labelling on them, an unsupervised algorithm clusters the data into different groups, e.g. it can distinguish that faces are very different from landscapes, which are very different from horses.

5.2.2(C) Reinforced Learning

- In this method, a teacher though available, does not present the expected answer but only indicates if the computed output is correct or incorrect.
- The information provided helps the network in its learning process.
- A reward is given for a correct answer computed and a penalty for a wrong answer. Reinforced learning is a very general approach to learning that can be applied when the knowledge required to apply supervised learning is not available.
- However, it is usually better to use other methods such as supervised or unsupervised learning, because they are more direct.
- Reinforcement training is related to supervised learning. The output in this case may not be indicated as the desired output, but the condition whether it is ‘success’(1) or ‘failure’ (0) may be indicated. Based on this, error may be calculated. The error signal produced here is binary.

- Basically, this learning attempts to learn the input-output mapping through trial and error. Here the system knows whether the output is correct or not, but does not know the correct output.
- Table 5.2.1 shows the difference between supervised and unsupervised learning.

5.2.3 Activation Functions

- The output response of a neuron is calculated using an **activation function** (also called transfer function).
- The sum of weighted inputs (net input to the neuron) is applied with an activation to obtain the neuron's response.
- Neurons placed in the same layer use same activation.
- There are basically two types of activation functions: linear and non-linear.
- The non-linear activation functions are used in a multi-layer net.

The most commonly used activation functions are :

1. Unipolar Binary

- It's unipolar in nature meaning it generates only two values 0 or 1.
- It is used only at output layer.
- It is computed as,

$$f(\text{net}) = \begin{cases} 1 & \text{net} > 0 \\ 0 & \text{net} < 0 \end{cases}$$

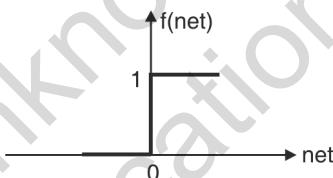


Fig. 5.2.3 : Unipolar binary activation function

2. Bipolar Binary

- It's bipolar in nature meaning it generates two values +1 or -1.
- It is used only at output layer.
- Its computed as,

$$f(\text{net}) = \text{sgn}(\text{net}) = \begin{cases} +1 & \text{net} > 0 \\ -1 & \text{net} < 0 \end{cases}$$

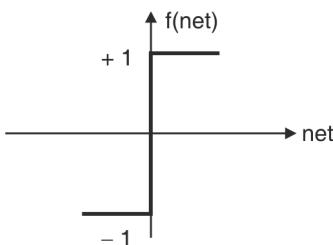


Fig. 5.2.4: Bipolar binary activation function

- The two functions bipolar binary and unipolar binary are called **hard limiting** activation functions.

3. Sigmoidal Function / Unipolar Sigmoidal

- Also known as unipolar Continuous or (Binary Sigmoidal)
- It's range is 0 to 1.
- It's non-linear in nature.
- Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only.
- It is computed as,

$$f(\text{net}) = \frac{1}{1 + e^{(-\lambda \text{net})}}$$

Where, $\text{net} = \sum_{i=1}^n w_i x_i$

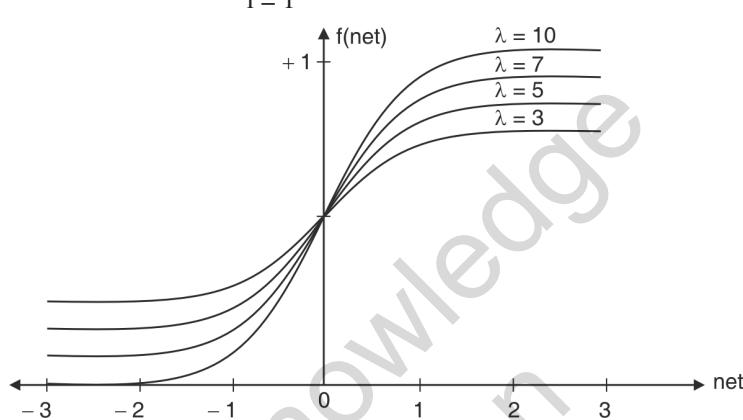


Fig. 5.2.5 : Sigmoidal or Unipolar continuous activation function

4. Tanh Function / Bipolar Sigmoidal

- Also called Bipolar Continuous (or Bipolar Sigmoidal).
 - The range is between +1 and -1.
 - It is computed as,
- $$f(\text{net}) = \frac{2}{1 + e^{(-\lambda \text{net})}} - 1 \quad \text{Where, } \text{net} = \sum_{i=1}^n w_i x_i$$

- This function is related to the hyperbolic tangent function. λ is called the **steepness parameter**.
- Usually used in hidden layers of a neural network as it's values lies between **-1 to 1**.
- It's output is zero centered because its range in between -1 to 1. Hence in practice it is always preferred over Sigmoid function .

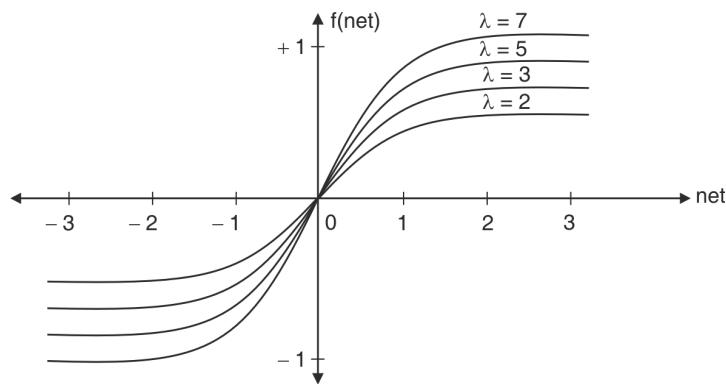


Fig. 5.2.6 : Tanh or bipolar continuous activation function

- We can easily observe that at $\lambda \rightarrow \infty$ in bipolar continuous (sigmoidal) function it becomes the sgn (net) function.
- The two activation functions bipolar continuous and unipolar continuous have sigmoidal characteristics and hence called **soft limiting** activation functions.

Note : Both sigmoidal and tanh functions suffer from Vanishing gradient problem.

5. ReLU- Rectified Linear units

- It is mathematically represented as

$$R(x) = \max(0, x)$$

i.e if $x < 0$, $R(x) = 0$ and if $x \geq 0$, $R(x) = x$.

- It is simple and efficient.
- It was recently proved that it had 6 times improvement in convergence from Tanh function.
- it avoids and rectifies **vanishing gradient** problem . Almost all deep learning Models use **ReLU** nowadays. But its limitation is that it should only be used within Hidden layers of a Neural Network Model.

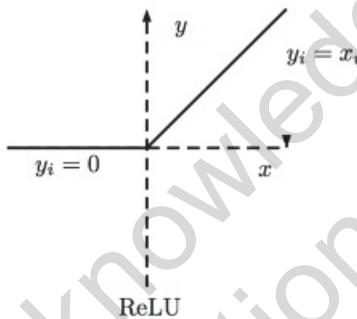


Fig. 5.2.7 : ReLU

6. Softmax Function

- The softmax function is also a type of sigmoid function.
- The sigmoid function can handle only two classes. What if there are multiple classes.
- Softmax is commonly used for multi - classification problems.
- It is non-linear in nature.
- The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide it by the sum of the outputs. This gives the probability of the input being in a particular class.
- The softmax function is ideally used in the output layer of the classifier where we actually try to attain the probabilities to define the class of each input.

Other activation functions:

Type	Equation	Functional form
Linear	$O = gI$ $g = (\tan \phi)$	

Type	Equation	Functional form
Piecewise linear	$O = \begin{cases} 1 & \text{if } mI > 1 \\ gI & \text{if } mI < 1 \\ -1 & \text{if } mI > -1 \end{cases}$ $g = \tan \phi$	
Step function (unipolar binary)	$O = \begin{cases} 1 & \text{if } I \geq \theta \\ 0 & \text{if } I < \theta \end{cases}$	

Note :

Sigmoidal functions (unipolar continuous and bipolar continuous) are used in multilayer nets like back-propagation networks, Radial Basis Function Networks (RBFN) etc.

The reason is, in multilayer network, the actual inputs are effectively masked off from the output units by the intermediate layer.

The hard-limiting threshold functions (unipolar binary and bipolar binary) remove the information that is needed if the network is to successfully learn.

Hence the network is unable to determine which of the input weights should be increased and which ones should not.

On the other hand, soft-limiting activation functions (sigmoidal functions) give us some information on the inputs, so that the network will be able to determine when to strengthen or weaken the relevant weights.

5.3 Important Terminologies of ANN

5.3.1 Weights

In the architecture of neural networks, each neuron is connected to other neurons by means of directed communication links. Each of these links are associated with weights. These weights carry the information about the input signals. The weights can be presented in terms of a matrix called the **weight matrix**. The weight matrix is also called **connection matrix**, and is defined as,

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1^t \\ \mathbf{W}_2^t \\ \vdots \\ \mathbf{W}_n^t \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{bmatrix}$$

Where $\mathbf{W}_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T$ for $i = 1, 2, 3, \dots, n$ is the weight vector for processing element.

w_{ij} is the individual element of weight vector that represents the weight on the communication link from i^{th} neuron to j^{th} neuron.

5.3.2 Bias

- A bias is a weight on a connection from an additional input unit whose activation is always 1.
- Increasing the bias increases the net input to the unit.

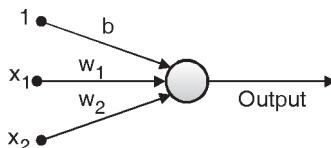


Fig. 5.3.1 : A single neuron with a bias

- If the bias is present, the net value is calculated as,

$$\text{net} = b + \sum_{i=1}^n x_i w_i; i = 1, 2, \dots, n$$

- For the above example, net is computed as,

$$\text{net} = b + w_1 x_1 + w_2 x_2$$

- Similar to initialization of weights, bias should also be initialized to some specific value.

Why do we need to use bias?

- Using bias in a neural network helps to improve performance of the net.
- For example, the bias value allows us to shift the activation function to the left or right which may be critical for successful learning.
- Usually, when a bias is not used, the separating line (or plane) passes through the origin. This may not be appropriate for solving a particular problem. Adding an adjustable bias helps us to shift the separating line (or plane) either to the right or to the left of the origin.
- Consider the following network shown in Fig. 5.3.2.

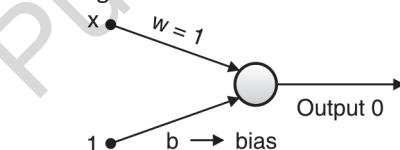


Fig. 5.3.2: Network with a bias weight

- If we use sigmoidal activation function, then output o is calculated as,

$$\begin{aligned} o &= \text{sigmoid}(1 \times x + b) \\ &= \text{sigmoid}(x + b) \end{aligned}$$

where b is the bias.

- Using different values of bias, we can shift the entire curve to the left or to the right as shown in the Fig. 5.3.4.

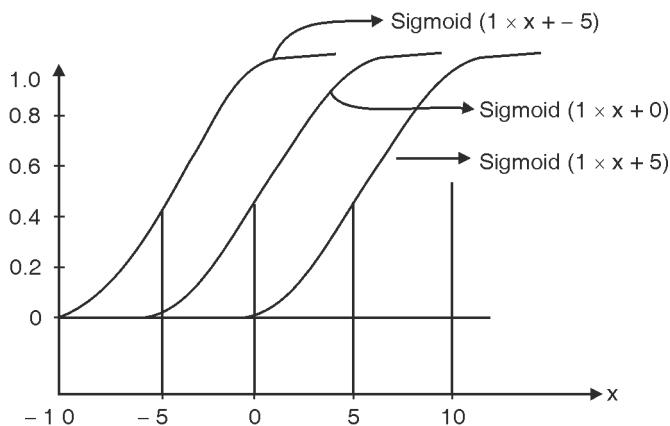


Fig. 5.3.4: Sigmoid functions with different bias values

5.3.3 Threshold

- Threshold is a set value based upon which the final output of the network is calculated.
- Usually a net input to the neuron is calculated and then compared with the threshold value.
- If the net value is greater than the threshold, then the neuron fires, otherwise, it does not fire.

5.3.4 Learning Constant

- The learning constant is used to control the amount of weight adjustment at each step of training. It controls the rate of learning.
- The effectiveness and convergence of learning algorithm depends significantly on the value of the learning constant. However, the optimum value of learning rate depends on the problem being solved and therefore there is no single learning constant that can be used for different training cases.
- For example, to solve a problem with broad minima, a large value of learning constant will result in a more rapid convergence. However, for problems with steep and narrow minima, a small value of learning constant must be chosen to avoid overshooting the solution. But choosing a small value of learning constant also increases the total number of steps in training.

5.3.5 Momentum Factor

- Convergence is made faster if a momentum factor is added to the weight update process. This is generally done in back propagation algorithm.
- The method involves supplementing the current weight adjustment with a fraction of the most recent weight adjustment.

$$\Delta W(t) = -\eta \nabla E(t) + \alpha \Delta W(t-1)$$

where, argument t and $t-1$ indicate the current and the most recent training step respectively.

- η is a learning constant and α is the user defined selected positive momentum constant.
- In the above equation, the term $\alpha \Delta W(t-1)$ indicates a scaled most recent adjustment of weights and is called the **momentum term**.
- Typically α is chosen between 0.1 and 0.8.

5.3.6 Vigilance Parameter

The vigilance parameter denoted by ρ is used in ART networks. It is used to control the degree of similarity required for pattern to be assigned to the same cluster unit. The range of vigilance parameter is usually 0.7 to 1.

5.4 McCulloch-Pitts Neuron Model

MU – Dec. 11, May 12, Dec. 12

- | | |
|---|------------------------------|
| Q. Explain Mc-Culloch Pitts Neuron Model with help of an example. | (Dec. 11, May 12, 5/6 Marks) |
| Q. Explain with an example McCulloch-Pitts neuron model. | (Dec. 12, 6 Marks) |

- The first formal definition of a synthetic neuron model based on highly simplified considerations of the biological model was formulated by McCulloch and Pitts in 1943.
- The model is shown in the Fig. 5.4.1.
- The inputs x_i for $i = 1, 2, 3, \dots, n$ are either 0 or 1, depending on the absence or presence of the input impulse at instant k .

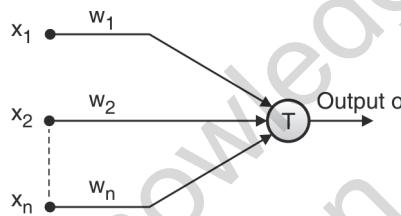


Fig. 5.4.1 : McCulloch Pitts Neuron model

- The neuron's output signal is denoted by o .
- The inputs are connected by direct weighted paths to neuron.
- The weights associated with the communication links may be excitatory (weight is positive) or inhibitory (weight is negative).
- There is a fixed threshold for each neuron, and if the net input to the neuron is greater than the threshold, then the neuron fires. If the net input to the neuron is less than the threshold, then the neuron does not fire i.e., It goes into an inhibition state.
- Accordingly, the firing rule for the neuron is defined as follows,

$$o^{k+1} = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i^k \geq T \\ 0 & \text{if } \sum_{i=1}^n w_i x_i^k < T \end{cases}$$

- In the above equation, superscript k denotes discrete time instances, $k = 1, 2, 3, \dots$.
- By analogy with the biological neuron, an effective synapse which transmits a stronger signal will have correspondingly larger weights, while a weak synapse will have smaller weights.
- In McCulloch Pitts' model,

$w_i = +1$ for excitatory synapses,

$w_i = -1$ for inhibitory synapses.

And T is the neuron's threshold value, which needs to be exceeded by the weighted sum of input signals for the neuron to fire.

- Although, this neuron model is very simplistic, it has substantial computing potential. It can perform basic logical operations such as AND, OR, NOT etc. provided its weights and threshold are appropriately selected.
- The McCulloch Pitts neuron does not have any particular training algorithm. An analysis has to be performed to determine the values of weights and the threshold.

Problems based on McCulloch Pitts Model :

Ex. 5.4.1 : Design two input AND logic using McCulloch Pitts Neuron model.

Soln.: Consider the truth table for the AND logic function shown in Table P. 5.4.1.

Here we need to find the appropriate values of w_1 , w_2 and threshold T such that, they satisfy the truth table of AND logic.

Here, the firing rule is,

$$w_1 x_1 + w_2 x_2 \geq T \quad \dots(1)$$

And inhibition rule is,

$$w_1 x_1 + w_2 x_2 < T \quad \dots(2)$$

From the truth table of AND logic, we get following four inequalities.

$$0 < T$$

$$w_2 < T$$

$$w_1 < T$$

$$w_1 + w_2 \geq T$$

Table P.5.4.1 : Truth table of AND

x_1	x_2	Output y
0	0	0
0	1	0
1	0	0
1	1	1

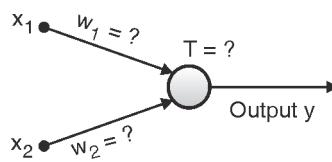


Fig. P.5.4.1: MCP Neuron Model for Logical AND

In order to get above four inequalities, use Equation (1) when neuron must fire and use Equation (2) when neuron must be inhibitory.

For $x_1 = 0, x_2 = 0 \Rightarrow y = 0$ (i.e. inhibitory)

So substitute $x_1 = 0, x_2 = 0$ in Equation (2), we get,

$$0 < T$$

For $x_1 = 0, x_2 = 1 \Rightarrow y = 0$ (i.e. inhibitory)

So substitute $x_1 = 0, x_2 = 1$ in Equation (2), we get,

$$w_2 < T$$

For $x_1 = 1, x_2 = 0 \Rightarrow y = 0$ (i.e. inhibitory)

So, substitute $x_1 = 1$ and $x_2 = 0$ in Equation (2), we get,

$$w_1 < T$$

For $x_1 = 1, x_2 = 1 \Rightarrow y = 1$ (i.e. Firing)

So, substitute $x_1 = 1$ and $x_2 = 1$ in Equation (1), we get,

$$w_1 + w_2 \geq T$$

Here w_1 and w_2 can only take either +1 or -1 values.

So, for $w_1 = 1, w_2 = 1$ and $T = 2$, all of the four inequalities are satisfied. Hence, solution is,

$$w_1 = 1$$

$$w_2 = 1$$

$$T = 2$$

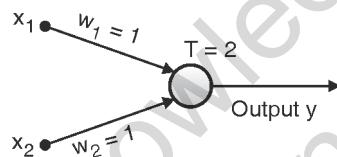


Fig. P.5.4.1(a) : Final weights and threshold for AND logic

Ex. 5.4.2 : Design two inputs OR logic using McCulloch Pitts Neuron model.

Soln.:

Consider the truth table for OR logic function shown in the Table P.5.4.2.

Table P.5.4.2 : Truth Table of OR

x₁	x₂	Output y
0	0	0
0	1	1
1	0	1
1	1	1

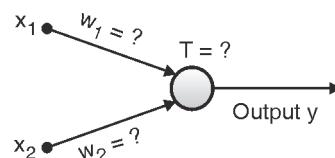


Fig. P.5.4.2 :MCP Neuron Model for Logical OR

We need to find the appropriate values of w_1, w_2 and T such that they satisfy the truth table of OR logic.

Here, the firing rule is,

$$w_1 x_1 + w_2 x_2 \geq T \quad \dots(1)$$

and the inhibitory rule is,

$$w_1 x_1 + w_2 x_2 < T \quad \dots(2)$$

From the truth table of OR logic, we get the following four inequalities.

$$0 < T$$

$$w_2 \geq T$$

$$w_1 \geq T$$

$$w_1 + w_2 \geq T$$

To derive the above four inequalities, use Equation (1) when the neuron must fire and use Equation (2) when the neuron must be inhibitory.

For $x_1 = 0, x_2 = 0 \Rightarrow y = 0$ (i.e. inhibitory), so substitute $x_1 = 0, x_2 = 0$ in Equation (2), we get,

$$0 < T$$

For $x_1 = 0, x_2 = 1 \Rightarrow y = 1$ (i.e. firing), so substitute $x_1 = 0, x_2 = 1$ in Equation (1), we get,

$$w_2 \geq T$$

For $x_1 = 1, x_2 = 0 \Rightarrow y = 1$ (i.e. firing) so substitute $x_1 = 1, x_2 = 0$ in Equation (1) we get,

$$w_1 \geq T$$

For $x_1 = 1, x_2 = 1 \Rightarrow y = 1$ (i.e. firing).

So, substitute $x_1 = 1, x_2 = 1$ in Equation (1) we get,

$$w_1 + w_2 \geq T$$

For $w_1 = 1, w_2 = 1$ and $T = 1$, all of the above inequalities are satisfied. Hence, the solution is,

$$w_1 = 1$$

$$w_2 = 1$$

$$T = 1$$

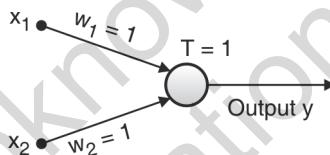


Fig. P.5.4.2(a) : Final weights and threshold for OR logic

Ex. 5.4.3 : Implement the logic function shown in Table P.5.4.3 using McCulloch-Pitts neuron.

Soln. :

Table P.5.4.3 : Truth table

x_1	x_2	Output y
0	0	0
0	1	0
1	0	1
1	1	0

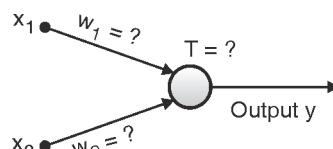


Fig. P.5.4.3: MCP Neuron Model for given Logical function

We need to find the appropriate values of w_1, w_2 and T such that they satisfy the truth table.

The firing rule is,

$$w_1 x_1 + w_2 x_2 \geq T \quad \dots(1)$$

And the inhibitory rule is,

$$w_1 x_1 + w_2 x_2 < T \quad \dots(2)$$

From the truth table, we get the following four inequalities,

$$0 < T$$

$$w_2 < T$$

$$w_1 \geq T$$

$$w_1 + w_2 < T$$

So for $w_1 = 1$, $w_2 = -1$ and $T = 1$, all of the above inequalities are satisfied. So, the solution is,

$$w_1 = 1$$

$$w_2 = -1$$

$$T = 1$$

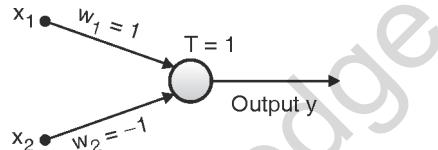


Fig. P.5.4.3(a) : Final weights and threshold for given logic

Ex. 5.4.4 : Design NOT logic using McCulloch Pitts neuron model.

Soln.:

Consider the truth table of NOT logic function.

Table P. 5.4.4 : Truth table of NOT

x	y
0	1
1	0

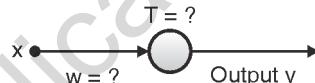


Fig. P.5.4.4 : MCP Neuron Model for Logical NOT

Here, we need to find out the appropriate values of w and T, such that, they satisfy the truth table of NOT logic.

The firing rule can be,

$$w_x \geq T \quad \dots(1)$$

And inhibitory rule can be,

$$w_x < T \quad \dots(2)$$

From the truth table of NOT logic, we get following two inequalities.

$$0 \geq T$$

$$w < T$$

So, for $T = 0$ and $w = -1$, the above mentioned two inequalities are satisfied.

So, solution is,

$$w = -1$$

$$T = 0$$

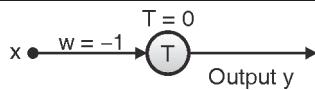


Fig. P.5.4.4(a) : Final weights and threshold for NOT

5.5 Neural Network Architecture

- As we know, an artificial neuron is defined as a data processing system consisting of a large number of simple highly interconnected processing elements inspired by the biological neuron.
- Generally, an ANN structure is represented using a directed graph, where neuron outputs are connected, through weights, to all other neurons including themselves; Both lag-free and delay connections are allowed.
- Based on these connection types, there are four fundamental architectures of neural networks.

5.5.1 Single Layer Feed Forward Networks

- Fig 5.5.1 shows the architecture of a single layer feed forward network.
- A single layer feed forward network consists of neurons arranged in two layers.
- The first layer is called the **input layer** and the second layer is called the **output layer**.
- The input signals are fed to the neurons of the input layer and neurons of the output layer produce output signals.
- Every input neuron is connected to every output neuron via synaptic links or weights.

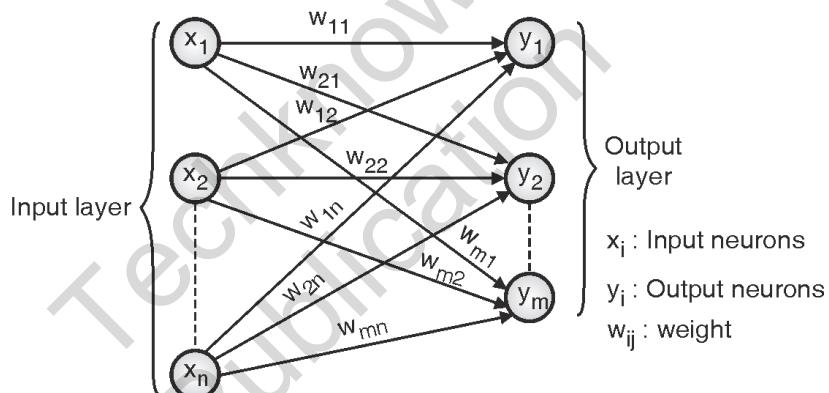


Fig. 5.5.1 : Single Layer feed forward network

- The output vector is given as,

$$\mathbf{O} = [o_1 \ o_2 \ o_3 \dots \ o_m]^t$$

And input vector is,

$$\mathbf{X} = [x_1 \ x_2 \ x_3 \dots \ x_n]^t$$

- Weight W_{ij} connects the i^{th} output neuron with the j^{th} input neuron.
- Then the activation value of the i^{th} neuron is given as,

$$\text{net}_i = \sum_{j=0}^n w_{ij} x_j, \text{ For } i = 1, 2, 3, \dots, m$$

- The transformation performed by each of the m neurons in the network, is non-linear mapping and expressed as,

$$O_i = f(W_i^t X), \text{ for } i = 1, 2, 3, \dots, m$$

Where weight vector W_i contains weights leading toward the i^{th} output node and is defined as,

$$W_i = [w_{i1} \ w_{i2} \ w_{i3} \dots \ w_{in}]^t$$

- In spite of having two layers, the network is still named as ‘single layer’ because out of the two layers, only the output layer performs computations. The input signal is simply transmitted to the output layer by the input layer.

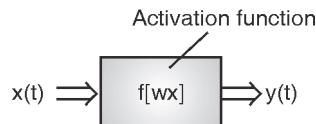


Fig. 5.5.2 : Feed forward network block diagram

- Thus, feed forward networks are characterised by the lack of feedback. That is, the output of any given neuron is not fed back to itself directly or indirectly or through other neurons. Thus, present output does not influence future output.

5.5.2 Multilayer Feed Forward Networks

- In a multilayer feed forward network, there are multiple layers. Thus, besides having input layer and output layer, this network has one or more intermediary layers called hidden layers.

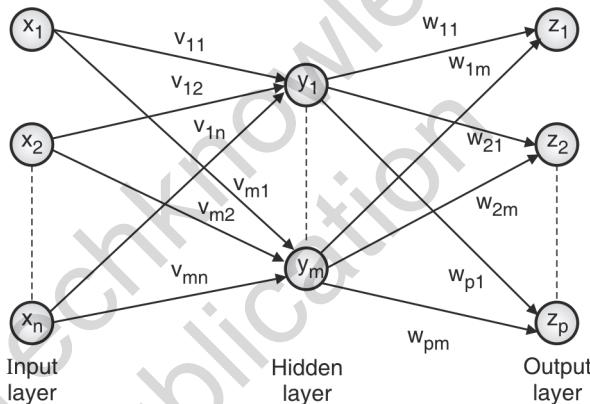


Fig. 5.5.3 : Multilayer feed forward network

Input layer	Hidden layer	Output layer
x_i	Input neurons	$i = 1, 2, 3, \dots, n$
y_j	Hidden neurons	$j = 1, 2, 3, \dots, m$
z_k	Output neurons	$k = 1, 2, 3, \dots, p$
v_{ij}	Input- hidden layer weight	
w_{jk}	Hidden-output layer weight	

- Fig. 5.5.3 shows a multilayer feed forward network.
- The computational units of the hidden layer are known as hidden neurons or hidden units. Before directing the input to the output layer the hidden layer performs useful intermediary computations.
- Neurons in input layer are connected to the neurons in hidden layer and the weights on these connections are referred to as “Input-hidden layer weights”.

- Similarly, the hidden layer neurons are connected to the output layer neurons and the corresponding weights are referred to as "Hidden-output layer weights".
- A multi-layer feed forward network with m input neurons, n_1 neurons in the first hidden layer, n_2 neurons in the second hidden layer and k output neurons is written as $m - n_1 - n_2 - k$ network.

5.5.3 Single Layer Feedback Network

- The feedback networks differ from the feed forward network in the sense that there exists at least one feedback loop.
- When the output of the output neurons is fed back as an input to the same or preceding layer nodes, then this type of network is called feedback network.
- Fig. 5.5.4 shows one such type of feedback network.

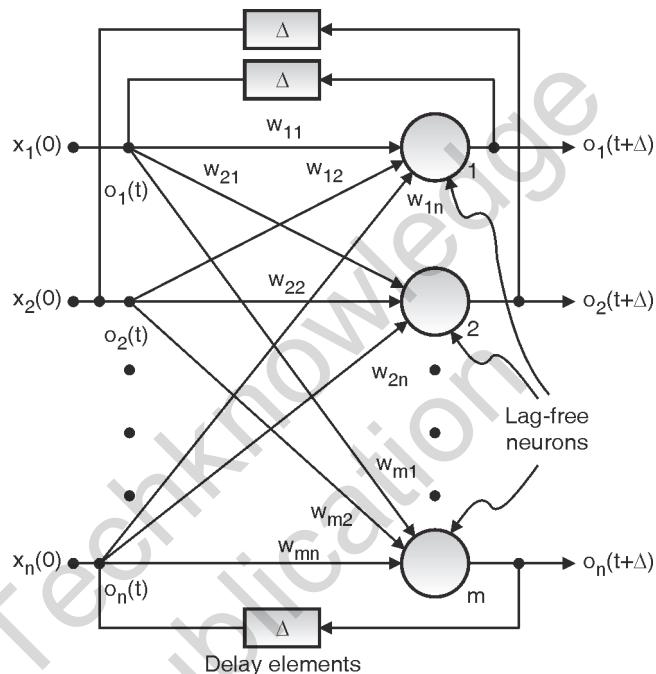


Fig. 5.5.4 : Single layer discrete time feedback network

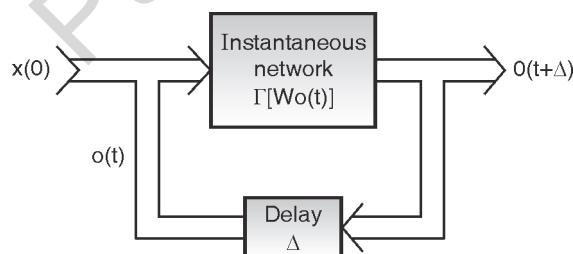


Fig. 5.5.5 : Block diagram of feedback network

- The present output, say $o(t)$, controls the output at the following instant, $o(t + \Delta)$.
- Δ indicates the time elapsed between t and $t + \Delta$. Here the time delay Δ is in an analogy to the refractory period of a basic biological neuron model.
- Thus, the mapping of $o(t)$ into $o(t + \Delta)$ can be written as,

$$o(t + \Delta) = \Gamma [W_o(t)]$$

5.5.4 Multilayer Feedback Networks

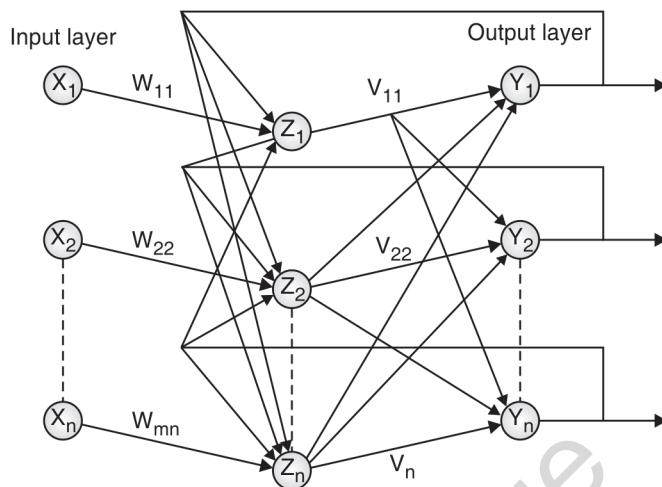


Fig. 5.5.6 : Multiplayer recurrent network (Redraw Figure)

- Fig 5.5.6 shows architecture of multilayer feedback network.
- It can be noted that a processing element output can be directed back to the nodes in a preceding layer, forming a multilayer recurrent network.
- Also, in these networks, a processing element output can be directed back to the processing element itself and to other processing elements in the same layer.

5.6 Supervised Learning

As discussed earlier, supervised learning works on labeled data meaning, for every input, the corresponding output (or class) is known.

Following section discusses three main supervised algorithms.

1. Perceptron learning rule
2. Delta learning rule
3. Back propagation algorithm

5.6.1 Perceptron Learning Rule

MU – Dec. 11, May 12

- | | |
|---|---|
| <p>Q. Explain with example perceptron learning rule.</p> <p>Q. Explain perceptron learning with the help of an example.</p> | <p>(Dec. 11, 10 Marks)</p> <p>(May 12, 8 Marks)</p> |
|---|---|

5.6.1(A) Rosenblatt's Perceptron

As shown in Fig.5.6.1, the network consists of 3 units, the Sensory unit S, Association unit A and Response unit R.

- The S unit receives input images and generates 0 or 1 electrical signal as output .It contains 400 photo-detectors.
- These input signals are then compared with threshold. If the input signals exceed a threshold, then the photo-detector outputs 1 otherwise 0.
- The photo-detectors are randomly connected to the Association unit A. The association unit A comprises of **feature demons or predicates**.

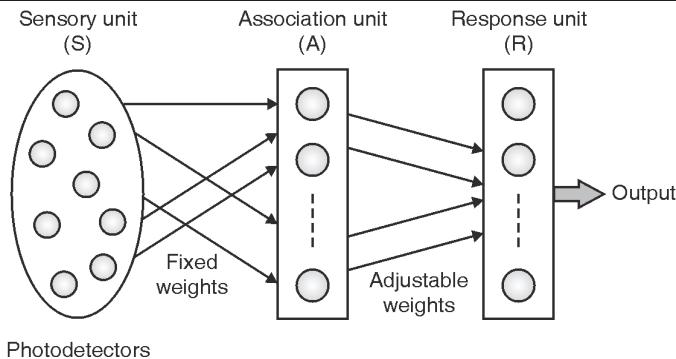


Fig. 5.6.1 : Rosenblatt's original perceptron model

- The 3rd unit **R** contains **pattern recognizers** (also called **perceptrons**) which receive the results of the predicate.
- The weights of S and A units are fixed while those of R are adjustable.
- The output of R is 1 if the weighted sum of its inputs is greater than 0, otherwise it is 0.

5.6.1(B) The Perceptron Model

- The perceptron model is based on the perceptron learning rule in which the learning signal is the difference between the desired output and the neuron's actual output.
- Here learning is obtained in **supervised** environment and the learning signal is equal to:

$$r = d_i - o_i$$

Where $o_i = \text{sgn}(W_i^t X)$

and d_i = desired (or target) output

- Here the weights are adjusted using following weight update formula.

$$\Delta W_i = c [d_i - \text{sgn}(W_i^t X)] X$$

$$\Delta W_{ij} = c [d_i - \text{sgn}(W_i^t X)] X_j ; \text{ for } j = 1, 2, \dots, n$$

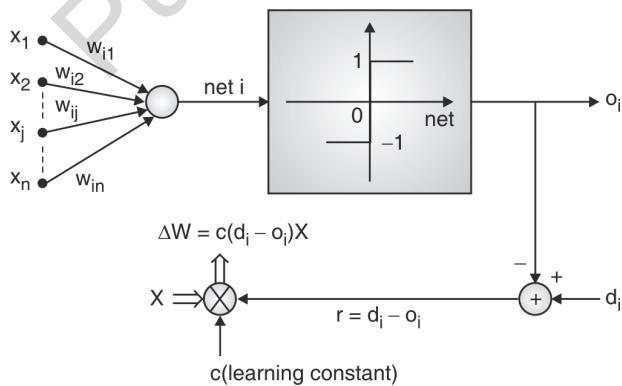


Fig. 5.6.2 : The perceptron model

- This rule is applicable for **binary neuron** response only.
- From the above formula of ΔW_i it is clear that ,weights are adjusted if and only if o_i is incorrect (i.e. if $d_i \neq o_i$)
- Since the desired output/response is either 1 or – 1, the weight adjustment reduces to

$$\Delta W_i = \pm 2cX$$

- Where $\Delta W_i = +2cX$ when $d_i = 1$ and $\text{Sgn}(W_i^t X) = o_i = -1$
and $\Delta W_i = -2cX$ when $d_i = -1$ and $\text{Sgn}(W_i^t X) = o_i = 1$
- The weight adjustment is inherently zero, when $d_i = o_i$ (i.e. $d_i = \text{sgn}(W_i^t X)$). Thus, when the desired output is the same as the actual output, there will not be any change in weights.

5.6.1(C) Algorithm for Training a Single Perceptron Network

- A single perceptron can solve a classification problem with 'n' input features and two output classes (0/1).
- Given below is the algorithm for training such a network and is called SDPTA (Single Discrete Perceptron Training Algorithm)

SDPTA (Single Discrete Perceptron Training Algorithm)

- Given are P training pairs $\{X_1, d_1, X_2, d_2, X_3, d_3, \dots, X_p, d_p\}$ where X_i is $(n \times 1)$, d_i is (1×1) , $i = 1, 2, \dots, P$
- Here augmented input vectors are used.

$$Y_i = \begin{bmatrix} X_i \\ 1 \end{bmatrix} \quad \text{for } i = 1, 2, \dots, P$$

i.e. there is an additional input for the bias and it is 1.

In the following, k denotes the training steps and p denotes the step counter within a cycle.

Step 1 : $c > 0$ is chosen

Step 2 : Weights are initialized as W at small random values.

W is $(n + 1) \times 1$

Counters are initialized

$K \leftarrow 1, p \leftarrow 1, E \leftarrow 0$

Step 3 : The training cycle begins here. Input is presented and output is computed

$Y \leftarrow Y_p, d \leftarrow d_p$

$O \leftarrow \text{sgn}(W^t Y)$

Step 4 : Weights are updated

$W \leftarrow W + \frac{1}{2} c (d - o) Y$

Step 5 : Cycle error is computed

$E \leftarrow \frac{1}{2} (d - o)^2 + E$

Step 6 : If $p < P$ then

$p \leftarrow p + 1 \quad k \leftarrow k + 1$

and go to step 3 ; otherwise go to step 7

Step 7 : The training cycle is completed.

For $E = 0$, terminate the training session. Display weights and k .

If $E > 0$, then $E \leftarrow 0, p \leftarrow 1$

and enter the new training cycle by going to step 3

5.6.1(D) Model for Multilayer Perceptron

- In multilayer perceptron networks, the perceptrons are arranged in layers.
- This model has three (or more) layers:
 - a. An input layer
 - b. An output layer and
 - c. One (or more) layers between input layer and output layer. These layers are called hidden layers.
- For perceptrons in the input layer, we use linear transfer function and for the perceptrons in the hidden and output layers, we use **sigmoid (continuous)** function.
- Here, since the single layer perceptron's model is modified by adding a hidden layer and changing the transfer (activation) function from linear function to a nonlinear function, we need to alter the learning rule as well. So now the multilayer perceptron network should be able to recognize more complex things.
- The input-output mapping of multilayer perceptron is shown in Fig. 5.6.3 and is represented by,

$O = f_2 [W [f_1[XV]]]$ Where, f_1 and f_2 represent non-linear mapping.

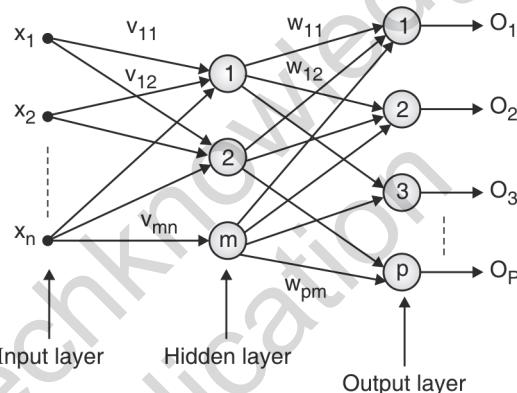


Fig. 5.6.3 : Multilayer perceptron

5.6.1(E) The EX-OR Problem and Need for Multi-layer Perceptron

- The simplest and most well-known pattern recognition problem in neural network literature is the Exclusive-OR problem.
- The truth table of Ex-OR function is given below.

Table 5.6.1: EX-OR Truth Table

x_1	x_2	Output (y)
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

- The task of the network is to classify a binary input vector to output class = -1, if the vector has similar inputs (Both the inputs +1 Or both the inputs -1) or assign it to output class = 1 otherwise.
- Let us try to solve this problem using a single-layer perceptron model. Consider the following single-layer perceptron model shown in Fig. 5.6.4.

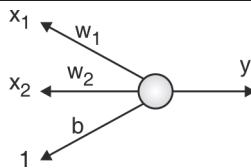


Fig. 5.6.4 : Single layer perceptron model

- For the above model, the decision boundary is

$$b + w_1 x_1 + w_2 x_2 = 0$$

$$\text{i.e. net} = 0$$

If net > 0, y (the output) = 1, else y = -1

- So from the truth-table and above conditions, we have the following four inequalities.

$$b + w_1 + w_2 \leq 0$$

$$b + w_1 - w_2 > 0$$

$$b - w_1 + w_2 > 0$$

$$b - w_1 - w_2 \leq 0$$

which must be satisfied.

- However, the set of inequalities is clearly self-contradictory when considered as a whole.
- This is because the EX-OR problem is not linearly separable. This can easily be observed from the plot given in Fig. 5.6.5.
- In other words, we cannot use a single layer perceptron to construct a straight line to partition the 2D input space into two regions (class 1 and class 2).

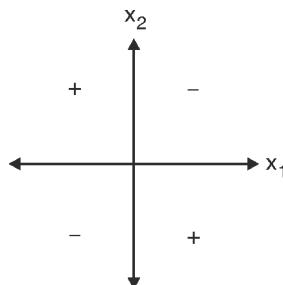


Fig. 5.6.5 : EX-OR function plot

Solving EX-OR problem using Multilayer perceptron

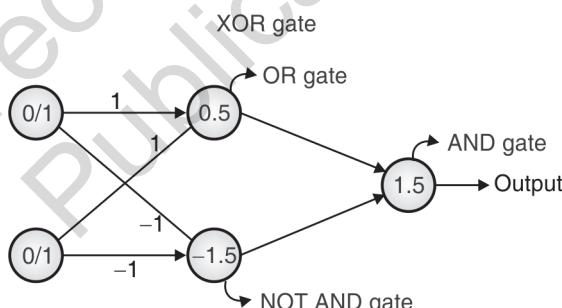


Fig. 5.6.6 : EX-OR using multi-layer perceptron

5.6.1(F) Linearly and Non-Linearly Separable Patterns

MU – Dec. 10, May 12, Dec. 13, Dec. 15

Q. Explain linearly separable and non-linearly separable patterns with examples.

(Dec. 10, 5 Marks)

Q. Explain with examples linearly separable and non-linearly separable pattern classification.

(May 12, Dec. 13, Dec. 15, 10 Marks)

a. Linearly Separable Patterns

- Consider a particular classification problem, where if the input pattern is a member of its class, then the desired response (output) is "yes", otherwise it is "no".
- A "yes" is represented by an output signal of 1, a "no" by an output signal of -1 (considering bipolar signals).

- To achieve this, a step function can be used as an activation function. That is, the output is + 1, if the net input to the neuron is positive, and -1 if the net input to the neuron is negative. Since the net input to the neuron is computed as

$$\text{net} = b + \sum_{i=1}^n x_i w_i$$

- It is clear that the boundary between the region where $\text{net} > 0$ and the region where $\text{net} < 0$ is determined by the relation.

$$b + \sum_{i=1}^n x_i w_i = 0$$

- This boundary is also called **decision boundary**.
- Depending on the number of input units in the network, this equation represents a line, a plane or a hyperplane.
- If there exist weights (and a bias) such that all of the training input vectors for which correct response is + 1 lie on one side of the decision boundary and all of the training input vectors for which the correct response is – 1 lie on the other side of the boundary then we say that the problem is “**linearly separable**”.

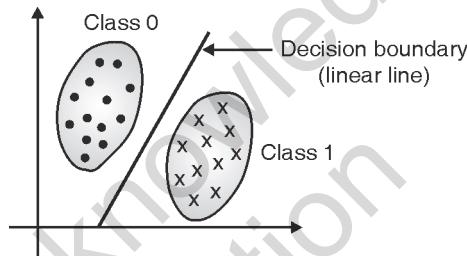


Fig. 5.6.7 (a) : Linearly separable pattern

- The problem shown in Fig. 5.6.7 (a) is a linearly separable problem and can be solved using the network shown in Fig. 5.6.7 (b).

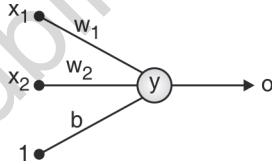


Fig. 5.6.7 (b) : Network to solve linearly separable problem in Fig. 5.6.7 (a)

- The region where the output is positive is separated from the region where it is negative by the line,

$$x_2 = \frac{-w_1}{w_2} x_1 - \frac{b}{w_2}$$

- Here,

∴ net is computed as

$$\text{net} = b + \sum_{i=1}^n w_i x_i$$

- Here,

$$\text{net} = b + w_1 x_1 + w_2 x_2$$

- So, the decision boundary is

$$b + w_1 x_1 + w_2 x_2 = 0$$

$$\therefore x_2 = \frac{-w_1}{w_2}x_1 - \frac{b}{w_2}$$

- In the above example, there are many different lines that will serve to separate the input vectors into two classes. There could also be many choices of w_1 , w_2 and b that give exactly the same line.
- To understand the concept of linear separability further, consider simple logic gates AND and OR.
- The AND gate can be represented by the truth-table shown in Table 5.6.2.

Table 5.6.2 : AND Truth Table

x_1	x_2	y(output)
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Then, the desired response for the AND function can be represented as shown in Fig. 5.6.8(a).

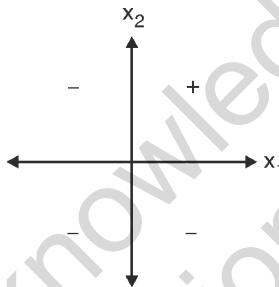


Fig. 5.6.8(a) : AND function plot

- The possible decision boundary for AND function could be as shown in Fig. 5.6.8 (b).

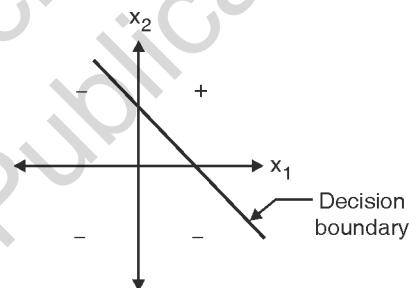


Fig. 5.6.8 (b) : AND function decision boundary

- Hence it is linearly separable.
- Similarly, the OR function is also linearly separable which is clear from the Fig. 5.6.9.

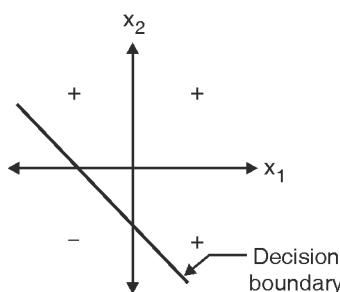


Fig. 5.6.9: OR Function solution

- The examples that we have seen are in 2D space.
- When we say that a set of points in two dimensional space are linearly separable, we mean that they can be separated by a straight line.
- More generally, a set of points in n-dimensional space are said to be linearly separable if they can be separated by a hyper plane.
- A linearly separable problem can be solved by a perceptron network.
- However, problems which are non-linearly separable cannot be solved by a single layered perceptron network.

b. Non-Linearly Separable Patterns

- To understand non-linearly separable patterns, consider the Exclusive-OR problem.
- Here, the task of the network is to classify a binary input vector to class 1 (output = -1), if the vector has even number of 1s, or assign it to class 2 (output = 1) otherwise.

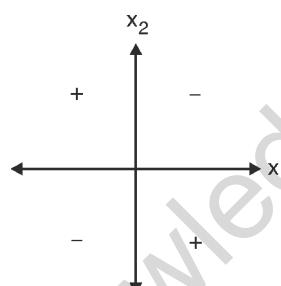


Fig. 5.6.10 : EX-OR function plot

- Fig. 5.6.10 shows EX-OR function plot. The EX-OR is non-linearly separable since there is no line that can partition the 2D input space into two regions (class1 and class2). This can easily be observed from the plot given in Fig. 5.6.10.
- Fig. 5.6.11 shows examples of linearly separable and non-linearly separable patterns.

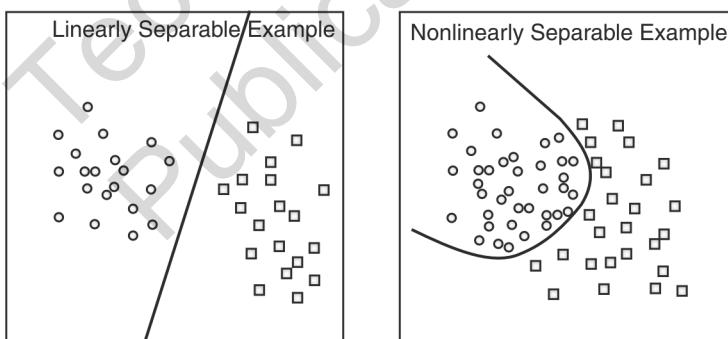


Fig. 5.6.11 : Linearly separable and non-linearly separable patterns

5.6.2 Delta Learning Rule

- The problem with perceptron learning is that many times it is unable to solve non-linearly separable problems. The initial solution to such problems was to use multiple layer perceptron.
- In such a network, for the perceptron in the first layer, the input comes from the actual inputs of the problem, while for the perceptron in the second layer, the inputs are outputs of the first layer.
- So, the perceptrons of the second layer do not know which of the real inputs of the first layer were on or off. The actual inputs are effectively masked off from the output units by the intermediate layer.
- The problem here is, the hard limiting thresholding activation function removes the information that is needed to train the network.

- The solution to this problem is to use non-linear (continuous) activation function such as sigmoidal or radial basis function.
- Hence Widrow-Hoff introduced the Delta learning rule that uses non-linear differentiable activation function for training the multilayer perceptron networks.

5.6.2(A) Delta Learning Model

- Delta learning is a supervised form of learning which uses ***continuous activation functions***.
- The learning signal for this rule is called delta and is defined as

$$r = [d_i - f(W_i^T X)] f'(W_i^T X)$$

- The term $f'(W_i^T X)$ is the derivative of the activation function $f(\text{net})$ computed for,

$$\text{net} = W_i^T X$$

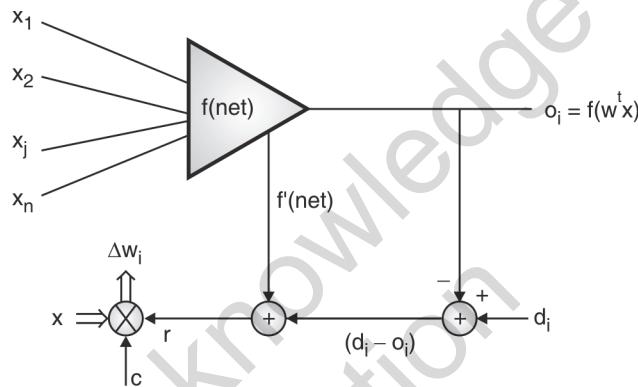


Fig. 5.6.12: Delta learning model

- The learning rule can be readily derived from the condition of least squared error between o_i and d_i .

Calculating the gradient vector with respect to W_i of the squared error defined as

$$E = \frac{1}{2} (d_i - o_i)^2 \quad \dots(5.6.1)$$

which is equivalent to

$$E = \frac{1}{2} [d_i - f(W_i^T X)]^2 \quad \dots(5.6.2)$$

- We obtain the error gradient vector value which is obtained by taking partial derivative of E with respect to each of the weights.

$$\nabla E = -(d_i - o_i) f'(W_i^T X) X \quad \dots(5.6.3)$$

The components of the gradient vector are

$$\frac{\partial E}{\partial W_{ij}} = -(d_i - o_i) f'(W_i^T X) x_j ; j = 1, 2, \dots, n \quad \dots(5.6.4)$$

Since the minimization of the error requires the weight changes to be in the negative gradient direction, we take

$$\therefore \Delta W_i = -\eta \nabla E \quad \dots(5.6.5)$$

Where η is a positive constant

$$\therefore \Delta W_i = \eta (d_i - o_i) f'(W_i^T X) X \quad \dots(5.6.6)$$



For a single weight, the adjustment becomes.

$$\Delta w_{ij} = \eta (d_i - o_i) f'(net_i) x_j ; j = 1, 2, 3, \dots n \quad \dots(5.6.7)$$

Thus,

$$\Delta W_i = c (d_i - o_i) f'(net_i) X$$

where $c = \eta$ (learning constant)

- The initial weights can be any random values.
- This rule is sometimes also called **continuous perceptron learning** rule.

5.6.2(B) Proofs

MU – Dec. 12, Dec. 13, Dec. 15

Q. Prove the following identities :

- (i) For unipolar continuous activation function $f'(net) = o(1-o)$
- (ii) For bipolar continuous activation function $f'(net) = o(1-o^2)/2$

Where o is output.

(Dec. 12, Dec. 13 10 Marks)

Q. Prove that the first order derivative of a unipolar continuous activation function is

$$f'(net) = O(1-O)$$

(Dec. 15, 5 Marks)

- Depending on whether the activation function is bipolar continuous or unipolar continuous, and further assuming that $\lambda = 1$, we can express $f'(net)$ in terms of the output $f(net)$ or o .
- For bipolar continuous activation function, we have,

$$f'(net) = \frac{1}{2} (1 - o^2) \quad \dots(5.6.8)$$

- Similarly, for unipolar continuous activation function.

$$f'(net) = o(1 - o) \quad \dots(5.6.9)$$

- Now, we will prove results (5.6.8) and (5.6.9)

Proofs

(I) $f'(net) = \frac{1}{2} (1 - o^2)$...[Bipolar continuous]

Proof :

$$o = f(net) = \frac{2}{1 + \exp(-\lambda net)} - 1$$

Assume $\lambda = 1$

$$\therefore o = f(net) = \frac{2}{1 + \exp(-net)} - 1$$

$$\text{L.H.S} = f'(net)$$

$$= \frac{d}{d(net)} \left\{ \frac{2}{1 + \exp(-net)} - 1 \right\} = \frac{d}{d(net)} \left\{ \frac{2}{1 + \exp(-net)} \right\} - (1)$$

$$= 2 \frac{d}{d(net)} \left\{ \frac{1}{1 + \exp(-net)} \right\}$$



$$\begin{aligned}
 &= 2 \left[\frac{-\frac{d}{d(\text{net})} (1 + \exp(-\text{net}))}{(1 + \exp(-\text{net}))^2} \right] \\
 &= \frac{-2(-\exp(-\text{net}))}{(1 + \exp(-\text{net}))^2} \\
 \text{L. H. S.} &= \frac{2 \exp(-\text{net})}{(1 + \exp(-\text{net}))^2} \quad \dots(5.6.10)
 \end{aligned}$$

$$\begin{aligned}
 \text{R. H. S.} &= \frac{1}{2} (1 - o^2) \\
 &= \frac{1}{2} \left[1 - \left\{ \frac{2}{1 + \exp(-\text{net})} - 1 \right\}^2 \right] \\
 &= \frac{1}{2} - \frac{1}{2} \left[\frac{2}{1 + \exp(-\text{net})} - 1 \right]^2 \\
 &= \frac{1}{2} - \frac{1}{2} \left[\frac{4}{(1 + \exp(-\text{net}))^2} - (2) \frac{2}{(1 + \exp(-\text{net}))} + 1 \right] \\
 &= \frac{-2}{(1 + \exp(-\text{net}))^2} + \frac{2}{1 + \exp(-\text{net})} \\
 &= 2 \left[\frac{-1 + (1 + \exp(-\text{net}))}{(1 + \exp(-\text{net}))^2} \right] \\
 &= \frac{2 \exp(-\text{net})}{(1 + \exp(-\text{net}))^2} \quad \dots(5.6.11)
 \end{aligned}$$

From Equations (5.6.10) and (5.6.11)

$$\text{L.H.S.} = \text{R.H.S.}$$

$$\text{Hence proved } f'(\text{net}) = \frac{1}{2} (1 - o^2)$$

(II) $f'(\text{net}) = o(1 - o)$... (Unipolar continuous)

Proof

$$\begin{aligned}
 o &= f(\text{net}) \\
 &= \frac{1}{1 + \exp(-\lambda\text{net})}
 \end{aligned}$$

$$\text{Assume } \lambda = 1$$

$$\begin{aligned}
 o &= f(\text{net}) \\
 &= \frac{1}{1 + \exp(-\text{net})}
 \end{aligned}$$

$$\begin{aligned}
 \text{L. H. S.} &= f'(\text{net}) \\
 &= \frac{d}{d(\text{net})} \left[\frac{1}{1 + \exp(-\text{net})} \right] \\
 &= \frac{-1 \frac{d}{d(\text{net})} (1 + \exp(-\text{net}))}{(1 + \exp(-\text{net}))^2} \\
 &= \frac{-(-\exp(-\text{net}))}{(1 + \exp(-\text{net}))^2}
 \end{aligned}$$



$$\text{L.H.S.} = \frac{\exp(-\text{net})}{(1 + \exp(-\text{net}))^2} \quad \dots(5.6.12)$$

$$\begin{aligned}\text{R.H.S.} &= o(1 - o) \\ &= \frac{1}{1 + \exp(-\text{net})} \left[1 - \frac{1}{1 + \exp(-\text{net})} \right] \\ &= \frac{1 + \exp(-\text{net}) - 1}{(1 + \exp(-\text{net}))^2} \\ &= \frac{\exp(-\text{net})}{(1 + \exp(-\text{net}))^2} \quad \dots(5.6.13)\end{aligned}$$

From Equations (5.6.12) and (5.6.13),

$$\text{L.H.S.} = \text{R.H.S.}$$

$$\text{Hence proved, } f'(\text{net}) = o(1 - o)$$

5.6.2(C) Algorithm-Delta Learning

MU – Dec. 15

Q. Explain Single Continuous Perceptron Training Algorithm (SCPTA).

(Dec. 15, 5 Marks)

SCPTA-Single Continuous Perception Training Algorithm

Given are P training pairs $\{X_1, d_1, X_2, d_2, \dots, X_p, d_p\}$

Where X_i is $(n \times 1)$ and d_i is (1×1)

$$i = 1, 2, 3, \dots, P$$

Here, augmented input is used,

$$Y_i = \begin{bmatrix} X_i \\ 1 \end{bmatrix}_{(n+1) \times 1} \text{ for } i = 1, 2, \dots, P$$

In the following, k denotes the no. of training steps and p denotes the no. of steps within a training cycle.

Step 1 : $\eta > 1$, $\lambda = 1$, $E_{\max} > 0$ chosen

Step 2 : Weights W are initialized at some small random values.

W is $(n + 1) \times 1$

Counters an initialized

$$p \leftarrow 1, \quad k \leftarrow 1, \quad E \leftarrow 0$$

Step 3 : The training cycle begins here.

Input is presented and output is computed.

$$Y \leftarrow Y_p, \quad d \leftarrow d_p$$

$$o \leftarrow f(W^T Y)$$

$$\text{With } f(\text{net}) = \frac{2}{1 + e^{-\lambda \text{net}}} - 1$$



Step 4 : Weights are updated

$$\mathbf{W} \leftarrow \mathbf{W} + 1/2 \eta (d - o) (1 - o^2) Y$$

Step 5 : Cycle error is computed

$$E \leftarrow E + 1/2 (d - o)^2$$

Step 6 : If $p < P$ then

$$k \leftarrow k + 1$$

$p \leftarrow p + 1$ and go to Step 3

Otherwise, go to Step 7.

Step 7 : The training cycle is completed. For $E < E_{\max}$ terminate the training session. Output weights, k and E .

If $E \geq E_{\max}$ then,

$$E \leftarrow 0$$

$p \leftarrow 1$ and enter the new training cycle by going to Step 3.

5.6.3 Back Propagation Algorithm

MU – Dec. 10, Dec. 11, May 12, Dec. 12, May 13, Dec. 13, Dec. 14

Q. Explain Error back Propagation Algorithm with the help of flowchart.	(Dec. 10, 12 Marks)
Q. Explain Error back propagation training algorithm with the help of a flowchart.	(Dec. 11, May 12, Dec. 12, Dec. 13, Dec. 14, 10 Marks)
Q. Explain Error Back Propagation Training Algorithm (EBPTA).	(May 13, 10 Marks)

- The back propagation algorithm is one of the most important developments in neural networks. This learning algorithm is applied to multilayer feed-forward networks. Multi layer feed forward networks consist of processing elements with continuous differentiable activation functions.
- The back propagation algorithm is different from the other networks in respect to the process by which the weights are calculated during learning process.
- For the multilayer perception calculating the weights of the hidden layer in an efficient way that would result in a very small or zero output error is a challenging task.
- We can easily measure the error between the actual and desired output at the output layer. But at the hidden layer there is no direct information of the error available. Therefore, adjusting the weights of hidden layer perception is difficult.
- The back propagation algorithm solves this problem by providing the equation for updating the weights at hidden layer along with weight update formula at output layers such that the overall output error is minimized.
- The back-propagation algorithm contains two phases:
 1. Feedforward recall and
 2. Feedback phase that calculates the back-propagation of the error.

5.6.3(A) Architecture of Back Propagation Network

Consider the layered feed forward neural network with two continuous perceptron layers shown in Fig. 5.6.13.

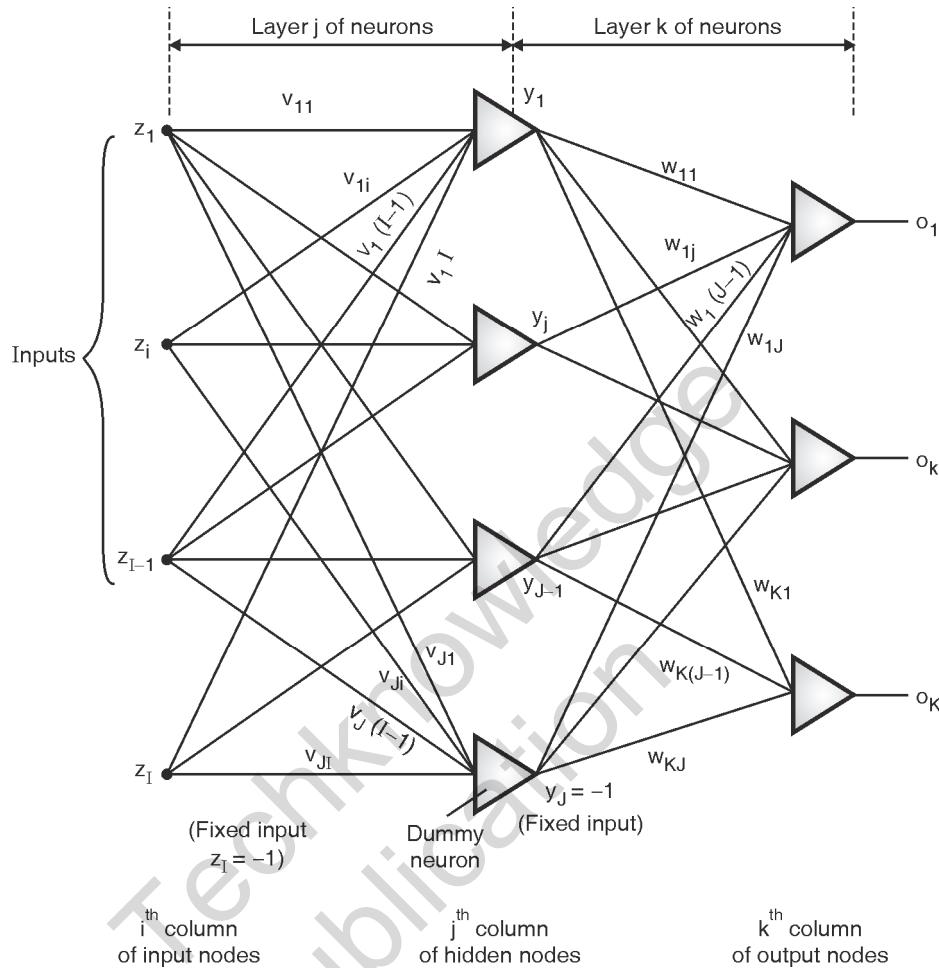


Fig. 5.6.13 : Back propagation network architecture

- In the above network, I input neurons are connected to each of the J^{th} hidden layer neuron.
- Weights connecting the input layer to the hidden layer are represented by matrix \mathbf{V} . Similarly, outputs of J hidden neurons are connected to each of the K output neurons, with weight matrix \mathbf{W} .
- EBPTA uses supervised learning mode, therefore the training patterns Z should be arranged in pairs with desired response d provide by the teacher.

Feed forward recall phase

- In this phase, the input pattern vectors are presented and mapped to the output vector O as follows :

$$O = \Gamma [\mathbf{W} \cdot \Gamma [\mathbf{V}Z]]$$

Where, $\Gamma [\mathbf{V}Z]$ is the internal mapping and relates to the hidden layer mapping $Z \rightarrow Y$

- The operator Γ is a non-linear diagonal operator.
- \mathbf{V} and \mathbf{W} are adjusted so that error value proportional to $\|d - o\|^2$ is minimized.

Error back propagation training

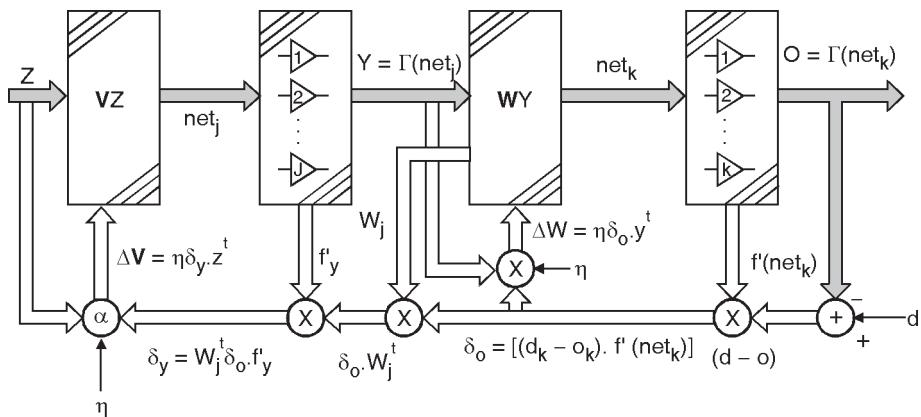


Fig. 5.6.14 : Block diagram : EBPT (Error back propagation training)

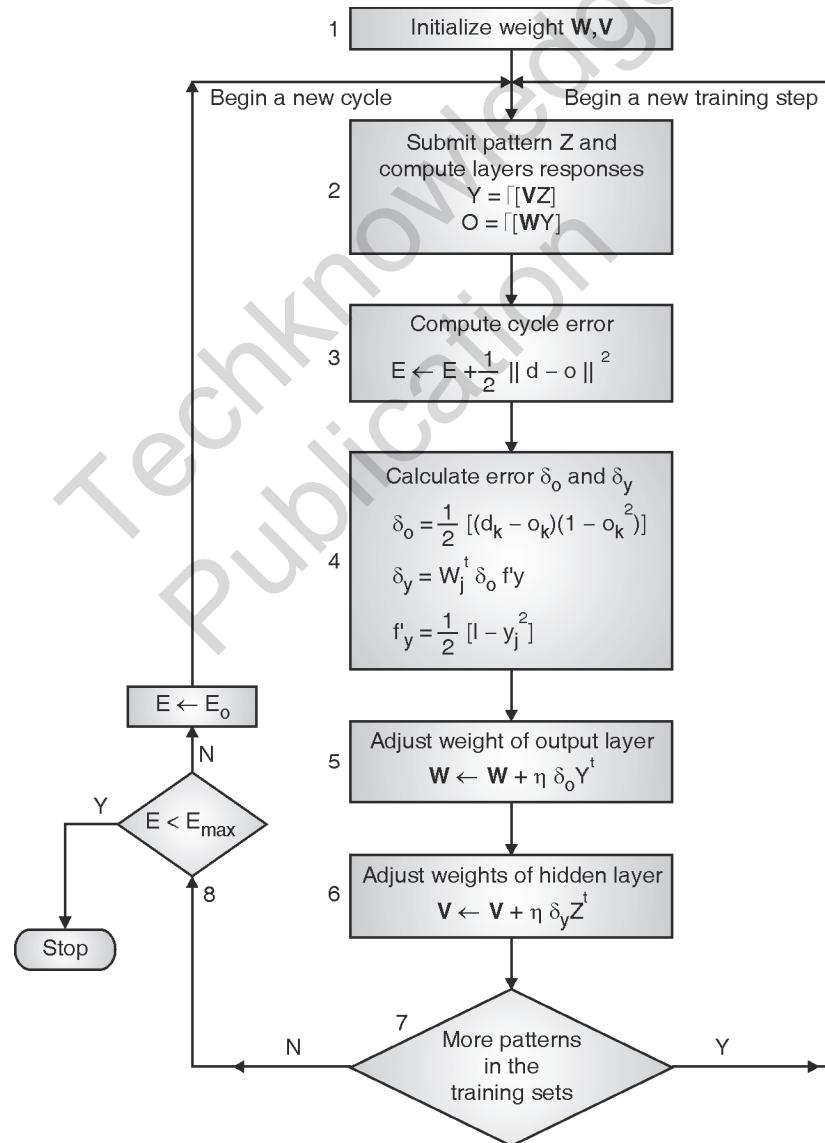


Fig. 5.6.15 : Flowchart Error Back propagation



- Consider the flowchart for EBPTA shown in Fig. 5.6.15.
- The training begins with the feed forward recall phase (Step 2). After a single pattern vector Z is submitted at the input, the layers response Y and O are computed.
- Then the error signal computation phase (Step 4) follows. The error signal vector must be determined in the output layer first and then it is propagated back towards the network input nodes.
- Next, $K \times J$ weights are subsequently adjusted within the matrix W (Step 5).
- Finally $J \times I$ weights are adjusted within the matrix V (Step 6).
- The cumulative cycle error of input to output mapping is computed (in Step 3)
- The final error value for the entire training cycle is calculated after each completed pass through the training set $\{Z_1, Z_2, Z_3, \dots, Z_p\}$
- The learning stops when the final error value less than E_{max} is obtained (Step 8)

5.6.3(B) Algorithm (Error Back Propagation Training)

- Given are P training pairs

$$\{Z_1, d_1, Z_2, d_2 \dots Z_p, d_p\}$$

Where Z_i is $(I \times 1)$, d_i is (1×1) and $i = 1, 2, 3, \dots, P$. The i^{th} component of each Z_i is of value (-1) since input vector have been augmented.

- There are total $J - 1$ neurons in hidden layer having output Y. Note that the J^{th} component of Y is also of value (-1) , since hidden layer outputs have also been augmented.
- Thus Y is $(J \times 1)$ and O is $(K \times 1)$

Step 1 : $\eta > 0$ and E_{max} chosen.

Weights W and V are initialized at small random values.

W is $(K \times J)$, **V** is $(J \times I)$

$q \leftarrow 1, p \leftarrow 1, E \leftarrow 0$

Step 2 : Training step starts here

Input is presented and layer's outputs computed.

$$Z \leftarrow Z_p, d \leftarrow d_p$$

$$y_j \leftarrow f(V_j^T Z), \text{ for } j = 1, 2, \dots, J$$

where V_j , a column vector, is the j^{th} row of vector V and

$$o_k \leftarrow f(W_k^T Y), \text{ for } k = 1, 2, \dots, K$$

Where W_k is a column vector, and is the k^{th} row of W.

Step 3 : Error value is computed

$$E \leftarrow \frac{1}{2} (d_k - o_k)^2 + E, \text{ for } k = 1, 2, 3, \dots, K$$

Step 4 : Error signal δ_o and δ_y of both the layers are computed.



Vector δ_o , is $(K \times 1)$, δ_y is $(J \times 1)$

$$\delta_{ok} = \frac{1}{2}(d_k - o_k) \left(1 - o_k^2 \right), \text{ for } k = 1, 2, \dots, K$$

$$\text{and } \delta_{yj} = \frac{1}{2} \left(1 - y_j^2 \right) \sum_{k=1}^K \delta_{ok} w_{kj}, \text{ for } j = 1, 2, \dots, J$$

Step 5 : Output layer weights are adjusted.

$$w_{kj} \leftarrow w_{kj} + \eta \delta_{ok} y_j, \quad \text{for } k = 1, 2, 3, \dots, K \\ j = 1, 2, 3, \dots, J$$

Step 6 : Hidden layer weights are updated

$$v_{ji} \leftarrow v_{ji} + \eta \delta_{yj} z_i, \quad \text{for } j = 1, 2, 3, \dots, J \\ i = 1, 2, 3, \dots, l$$

Step 7 : If $p < P$ then $p \leftarrow p + 1$, $q \leftarrow q + 1$ and go to step 2, otherwise go to step 8.

Step 8 : The training cycle is completed.

For $E < E_{\max}$ terminate the training and

Output weights W, V, q and E .

If $E > E_{\max}$, then $E \leftarrow 0$, $p \leftarrow 1$, and initiate the new training cycle by going to step 2.

5.7 Un-Supervised Learning Algorithm: Self Organizing Maps

MU – Dec. 10, Dec. 11, May 12, May 13, Dec. 14, Dec. 15

Q. Write a short note on Kohonen's self organizing network.

(Dec. 10, Dec. 11, May 12, May 13, Dec. 14, 5/10 Marks)

Q. What is self-organizing map? Draw and explain architecture of Kohonen's Self Organization Feature Map KSOFM.

(Dec. 15, 10 Marks)

Self-Organizing Maps:

- A Self-Organizing Map (SOM) uses unsupervised learning to build a two-dimensional map of a problem space.
- It uses competitive learning as opposed to error-correction learning (which is used in backpropagation with gradient descent)
- A self-organizing map can generate a visual representation of data on a hexagonal or rectangular grid.
- SOM also represents clustering concepts by grouping similar data together.
- Applications include meteorology, oceanography, project prioritization, and oil and gas exploration.
- A self-organizing map is also known as a Self-Organizing Feature Map (SOFM) or a Kohonen map.

Kohonen's Self Organizing maps:

- Kohonens's self organizing networks, also called Kohonen's feature maps or topology preserving maps, are used for data clustering.
- Networks of this type impose a neighbourhood constraint on the output unit such that a certain topological property in the input data is reflected in the output unit's weight.

- Fig. 5.7.1 presents relatively simple Kohonen's self organizing network with two inputs and 49 outputs.

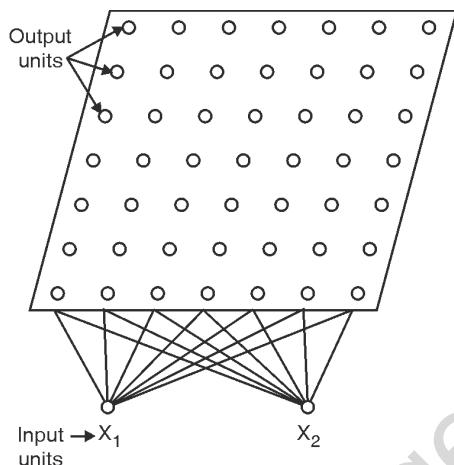


Fig. 5.7.1 : Kohonen's self organizing network with two inputs and 49 outputs

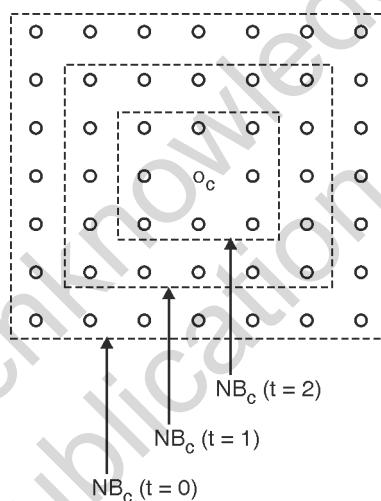


Fig. 5.7.2 : Neighbourhood of unit C

- The learning process here is similar to that of competitive learning networks. That is, a similarity (or dissimilarity) measure is selected and the winning unit is considered to be the one with the largest (or smallest) activation function.
- However, for Kohonen's feature maps, we update not only the winning unit's weight but also all of the weights in a neighbourhood around the winning unit.
- The neighbourhood size generally decreases slowly with each iteration, as shown in Fig. 5.7.2.
- Instead of defining a neighbourhood of a winning neuron, we can also use a neighbourhood function $\Omega_c(i)$ around a winning unit c .

Where,

$$\Omega_c(i) = \exp\left(\frac{-(P_i - P_c)^2}{2\sigma^2}\right)$$

- Here, σ reflects the scope of the neighbourhood and P_i and P_c are the positions of the output unit i and c respectively.

Algorithm (Kohonen's self organization map)

- Step 1 :** Initialize the weights at some random values → set topological neighborhood parameters. As clustering progresses, the radius of the neighborhood decreases. Initialize the learning rate α . It should be slowly decreasing function of time.

Step 2 : Perform step 3-9 till stopping condition is true.

Step 3 : Perform steps 4-6 for each input vector X.

Step 4 : Compute the square of the Euclidean distance i.e. for each $j = 1$ to m (there are m no. of output neurons)

$$D(j) = \sum_{i=1}^n \sum_{j=1}^m (x_i - w_{ij})^2$$

Step 5 : Find the winning unit index J, so that $D(J)$ is minimum.

Note : Instead of Euclidean distance, dot product method can also be used to finalize the winner, in which case the winner will be the one with the largest dot product.

Step 6 : For all units j within a specific neighborhood of J and for all i , calculate new weights.

$$w_{ij} = w_{ij}(\text{old}) + \alpha [x_i - w_{ij}(\text{old})]$$

Step 7 : Update the learning rate α using formula $\alpha(t+1) = 0.5 \alpha(t)$

Step 8 : Reduce radius of topological neighborhood at specified time intervals.

Step 9 : Test for stopping condition.

5.8 Solved Problems

Ex. 5.8.1 : A neuron with 4 inputs has the weight vector $w = [1 2 3 4]^t$. The activation function is linear that is, the activation function is given by $f(\text{net}) = 2 * \text{net}$. If the input vector is $X = [5 6 7 8]^t$, then find the output of the neuron.

MU - Dec. 11, 5 Marks

Soln. :

Fig.P. 5.8.1 shows the architecture of the given network.

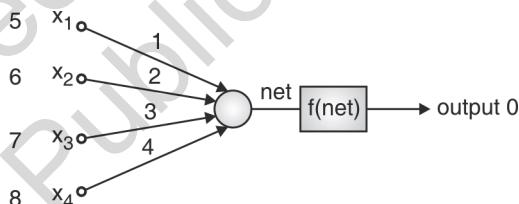


Fig. P. 5.8.1 : Architecture of given network

$$\text{net} = W^t \cdot X$$

$$= [1 2 3 4] \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

$$= 5 + 12 + 21 + 32$$

$$= 70$$

$$O = f(\text{net}) = 2 * \text{net} = 2 \times 70 = 140$$



Ex. 5.8.2 : Use perceptron learning rule to train the network. The set of input training vectors are as follows.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, X_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, X_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

and the initial weight vector W_1 is,

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

The learning constant $c = 0.1$. The teacher's desired response for X_1, X_2 and X_3 are $d_1 = -1$, $d_2 = -1$ and $d_3 = 1$ respectively. Calculate the weights after one complete cycle.

Soln. :

For perceptron learning rule,

$$\text{net}_i = W_i^T X$$

$$o_i = \text{sign}(\text{net}_i)$$

$$\Delta W_i = c(d - o_i) X$$

Step 1: Take first training pair $X = X_1$, $d = d_1$

$$X = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, d = -1$$

Compute :

$$\text{net}_1 = W_1^T X = [1 \ -1 \ 0 \ 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$o_1 = \text{sign}(\text{net}_1) = \text{sign}(2.5) = 1$$

$$\Delta W_1 = c(d - o_1) X$$

$$= 0.1(-1 - 1) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.2 \\ 0.4 \\ 0 \\ 0.2 \end{bmatrix}$$

$$W_2 = W_1 + \Delta W_1$$

$$= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -0.2 \\ 0.4 \\ 0 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 2 : Take second training pair,

$$\text{Set } X = X_2, d = d_2$$

$$X = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, d = -1$$

$$\text{Compute} \rightarrow \text{net}_2 = \mathbf{W}_2^T X = [0.8 \ -0.6 \ 0 \ 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = -1.6$$

$$O_2 = \text{Sign}(\text{net}_2) = \text{Sign}(-1.6) = -1$$

Here $d = -1$ and $O_2 = -1$

So, $(d - O_2) = 0$

Hence, $\Delta W_2 = 0$. Thus there is no change in weight

$$\therefore \mathbf{W}_3 = \mathbf{W}_2 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

Step 3 : Take third training pair,

$$\text{Set } X = X_3, d = d_3$$

$$X = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}, d = 1$$

$$\text{Compute} \rightarrow \text{net}_3 = \mathbf{W}_3^T X$$

$$= [0.8 \ -0.6 \ 0 \ 0.7] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = -2.1$$

$$O_3 = \text{Sign}(\text{net}_3) = \text{sign}(-2.1) = -1$$

Here

$$d = 1 \text{ and } O_3 = -1$$

$$\therefore \Delta W_3 = c(d - O_3) X$$

$$= (0.1)(1 - (-1)) \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 0.2 \\ 0.1 \\ -0.2 \end{bmatrix}$$

$$W_4 = W_3 + \Delta W_3$$

$$= \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + \begin{bmatrix} -0.2 \\ 0.2 \\ 0.1 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

Ex. 5.8.3 : Determine the weights after four steps of training for perceptron learning rule of a single neuron network starting with initial weights :

$$W = [0 \ 0]^T, \text{ inputs as } X_1 = [2 \ 2]^T,$$

$$X_2 = [1 \ -2]^T, X_3 = [-2 \ 2]^T, X_4 = [-1 \ 1]^T,$$

$$d_1 = 0, d_2 = 1, d_3 = 0, d_4 = 1 \text{ and } c = 1.$$

MU - Dec. 13, 10 Marks

**Soln. :**

Here, we use **unipolar binary** activation function because desired values given are 0's and 1's for perceptron learning,

$$\begin{aligned} \text{net}_i &= \mathbf{W}_i^t \mathbf{X} \\ o_i &= f(\text{net}_i) \\ \Delta \mathbf{W}_i &= c(d - o_i) \mathbf{X} \\ \text{Here } o_i &= \begin{cases} 1, & \text{net}_i \geq 0 \\ 0, & \text{net}_i < 0 \end{cases} \end{aligned}$$

Step 1 : Take first training pair,

$$\begin{aligned} \text{set } \mathbf{X} &= \mathbf{X}_1 \text{ and } d = d_1 \Rightarrow \mathbf{X} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \text{ and } d = 0 \\ \text{net}_1 &= \mathbf{W}_1^t \mathbf{X} \end{aligned}$$

Here \mathbf{W}_1 is the initial weight, $\mathbf{W}_1^t = [0 \ 0]$

$$\begin{aligned} \therefore \text{net}_1 &= [0 \ 0] \begin{bmatrix} 2 \\ 2 \end{bmatrix} \\ &= 0 \\ o_1 &= f(0) = +1 \\ \Delta \mathbf{W}_1 &= c(d - o_1) \mathbf{X} = (1)(0 - 1) \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \\ \therefore \mathbf{W}_2 &= \mathbf{W}_1 + \Delta \mathbf{W}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -2 \\ -2 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix} \end{aligned}$$

Step 2 : Take second training pair,

$$\begin{aligned} \text{set } \mathbf{X} &= \mathbf{X}_2 \text{ and } d = d_2 \\ \mathbf{X} &= \begin{bmatrix} 1 \\ -2 \end{bmatrix} \text{ and } d = 1 \\ \text{net}_2 &= \mathbf{W}_2^t \mathbf{X} = [-2 \ -2] \begin{bmatrix} 1 \\ -2 \end{bmatrix} \\ &= -2 + 4 = 2 \\ o_2 &= f(2) = +1 \\ \Delta \mathbf{W}_2 &= c \cdot (d - o_2) \cdot \mathbf{X} \\ &= (1)(1 - 1) \begin{bmatrix} 1 \\ -2 \end{bmatrix} = 0 \end{aligned}$$

Hence there is no change in weights.

$$\therefore \mathbf{W}_3 = \mathbf{W}_2 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

Step 3 : Take third training pair

$$\begin{aligned} \text{set } \mathbf{X} &= \mathbf{X}_3 \text{ and } d = d_3 \\ \mathbf{X} &= \begin{bmatrix} -2 \\ 2 \end{bmatrix} \text{ and } d = 0 \\ \text{net}_3 &= \mathbf{W}_3^t \mathbf{X} \end{aligned}$$

$$\begin{aligned}
 &= [-2 -2] \begin{bmatrix} -2 \\ 2 \end{bmatrix} \\
 &= 4 - 4 = 0 \\
 o_3 &= f(0) = +1 \\
 \Delta W_3 &= c(d - o_3) \cdot X \\
 &= (1)(0 - 1) \begin{bmatrix} -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \end{bmatrix} \\
 W_4 &= W_3 + \Delta W_3 = \begin{bmatrix} -2 \\ -2 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ -4 \end{bmatrix}
 \end{aligned}$$

Step 4 : Take fourth training pair

Set

$$\begin{aligned}
 X &= X_4 \quad \text{and} \quad d = d_4 \\
 X &= \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \text{and} \quad d = 1 \\
 \text{net}_4 &= W_4^T \cdot X = [0 \ -4] \begin{bmatrix} -1 \\ 1 \end{bmatrix} = 0 - 4 = -4 \\
 o_4 &= f(-4) = 0 \\
 \Delta W_4 &= c(d - o_4) X \\
 &= (1)(1 - 0) \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\
 W_5 &= W_4 + \Delta W_4 \\
 &= \begin{bmatrix} 0 \\ -4 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -3 \end{bmatrix}
 \end{aligned}$$

Ex. 5.8.4 : A neuron with 3 inputs has the weight vector $w = [0.1 \ 0.3 \ -0.2]$. The activation function is binary sigmoidal activation function. If input vector is $[0.8 \ 0.6 \ 0.4]$ then find the output of neuron. MU - May 13, 5 Marks

Soln. :

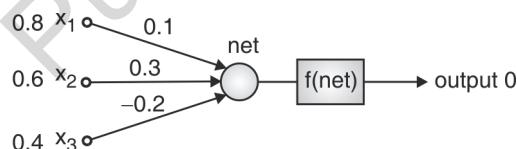


Fig. P. 5.8.4

Here $f(\text{net}) = \left[\frac{1}{1 + e^{(-\lambda \text{net})}} \right]$ [Binary sigmoid means the unipolar continuous]

Assume $\lambda = 1$

$$\text{Compute net} = W^T \cdot X = [0.1 \ 0.3 \ -0.2] \begin{bmatrix} 0.8 \\ 0.6 \\ 0.4 \end{bmatrix} = 0.08 + 0.18 - 0.08 = 0.18$$

Compute $o = f(\text{net})$

$$= \left[\frac{1}{1 + e^{(-0.18)}} \right] = \frac{1}{1 + 0.8352} = 0.5448$$



Ex. 5.8.5 : Implement OR function using perceptron networks for bipolar inputs and targets.

MU - Dec. 11, 10 Marks

Soln. :

The Truth table for a 2 input OR function is (assuming bipolar inputs target).

x_1	x_2	d
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1

Where x_1, x_2 are the 2 inputs and 'd' is the desired output (target)

Perceptron N/W for OR

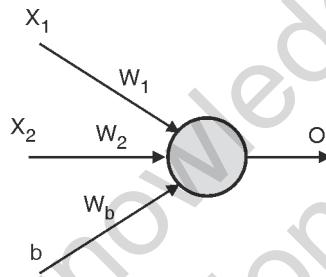


Fig. P. 5.8.5

Shown in Fig. P. 5.8.5 is a perceptron to represent OR function.

The output 'O' of the network is given by

$$O = \text{sign}(x_1w_1 + x_2w_2 + bw_b)$$

Now, let us train the n/w to obtain the OR function.

Assume that the initial weights are $w_1 = 0.5$ and $w_2 = -0.5$

Assume bias weight $w_b = 0.5$

Further assume bias input, $b = 1$

\therefore Our initial weight vector

$$w_1 = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \end{bmatrix}$$

And, our training pairs would be :

$$\left(x_1 = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}, d_1 = -1 \right), \left(x_2 = \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix}, d_2 = +1 \right)$$

$$\left(x_3 = \begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix}, d_3 = +1 \right), \left(x_4 = \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix}, d_4 = +1 \right)$$

Assume $c = 1$ (learning constant)

**Step 1 :**

$$\begin{aligned}
 \text{Compute } \text{net}_1 &= w_1^T x_1 \\
 w_1^T x_1 &= [0.5 \ -0.5 \ 0.5] \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} \\
 &= -0.5 + 0.5 + 0.5 = 0.5 \\
 O_1 &= \text{sgn}(\text{net}_1) = \text{sgn}(0.5) = +1 \\
 \text{Now, } d_1 &= -1 \\
 \therefore \Delta w_1 &= -2c x_1 \\
 &= -2(1) \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ -2 \end{bmatrix} \\
 \therefore \Delta w_2 &= w_1 + \Delta w_1 \\
 &= \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \\ -2 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1.5 \\ -1.5 \end{bmatrix}
 \end{aligned}$$

Step 2 :

$$\begin{aligned}
 \text{net}_2 &= w_2^T x_2 \\
 &= [2.5 \ 1.5 \ -1.5] \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix} \\
 &= -2.5 + 1.5 - 1.5 \\
 &= -2.5 \\
 O_2 &= \text{sgn}(\text{net}_2) = -1 \\
 d_2 &= +1 \\
 \therefore \Delta w_2 &= 2c x_2 = 2(1) \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 2 \end{bmatrix} \\
 \therefore w_3 &= w_2 + \Delta w_2 \\
 &= \begin{bmatrix} 2.5 \\ 1.5 \\ -1.5 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 3.5 \\ 0.5 \end{bmatrix}
 \end{aligned}$$

Step 3 :

$$\begin{aligned}
 \text{net}_3 &= w_3^T x_3 \\
 &= [0.5 \ 3.5 \ 0.5] \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \\
 &= 0.5 - 3.5 + 0.5 = -2.5 \\
 O_3 &= f(\text{net}_3) = -1 \\
 d_3 &= +1
 \end{aligned}$$

$$\therefore \Delta w_3 = 2 c x_3 = 2(1) \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ 2 \end{bmatrix}$$

$$\begin{aligned} w_4 &= w_3 + \Delta w_3 \\ &= \begin{bmatrix} 0.5 \\ 3.5 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 1.5 \\ 2.5 \end{bmatrix} \end{aligned}$$

Step 4 :

$$\begin{aligned} \text{net}_4 &= w_4^T x_4 \\ &= [2.5 \ 1.5 \ 2.5] \begin{bmatrix} +1 \\ +1 \\ +1 \end{bmatrix} \\ &= 2.5 + 1.5 + 2.5 = 6.5 \\ O_4 &= \text{sgn}(\text{net}_4) = \text{sgn}(6.5) = +1 \\ d_4 &= +1 \end{aligned}$$

\therefore No change in weights

$$\begin{aligned} \text{i.e. } \Delta w_4 &= 0 \\ w_5 &= w_4 = \begin{bmatrix} 2.5 \\ 1.5 \\ 2.5 \end{bmatrix} \end{aligned}$$

Step 5 :

$$\begin{aligned} \text{net}_5 &= w_5^T x_1 \\ &= [2.5 \ 1.5 \ 2.5] \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} \\ &= -2.5 - 1.5 + 2.5 = -1.5 \\ O_5 &= \text{sgn}(\text{net}_5) = \text{sgn}(-1.5) = -1 \\ d_1 &= -1 \end{aligned}$$

\therefore No change in weights

$$\Delta w_5 = 0 \qquad \qquad \qquad w_6 = w_5 = \begin{bmatrix} 2.5 \\ 1.5 \\ 2.5 \end{bmatrix}$$

Now, after Step 5, we have,

$$w_6 = w_5 = w_4$$

i.e. Weight vector is not changing after 2 consecutive steps. Hence, we stop training and this means that the training in complete final weight vector after training is

$$w = \begin{bmatrix} 2.5 \\ 1.5 \\ 2.5 \end{bmatrix}$$

∴ The perceptron network for OR function is as shown in Fig. P. 5.8.5(a).

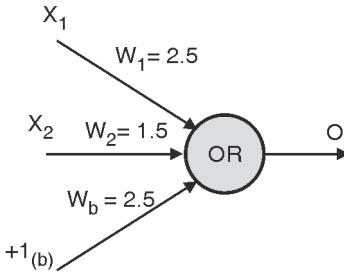


Fig. P. 5.8.5 (a)

$$\text{Where } O = \text{sgn}(2.5x_1 + 1.5x_2 + 2.5)$$

Ex. 5.8.6 : Consider the following set of input training vectors.

$$X_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, X_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, X_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$d_1 = -1, d_2 = -1$ and $d_3 = +1$ are desired responses for X_1, X_2 and X_3 respectively. Initial weight vector

$$W_1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}$$

The learning constant $c = 0.1$ and $\lambda = 1$. Use delta learning rule to calculate the final weights. Use bipolar continuous activation function.

Soln. :

For delta learning with bipolar continuous activation function.

$$\text{net}_i = W_i^T X \quad \dots(1)$$

$$o_i = f(\text{net}_i) = \frac{2}{1 + \exp(-\lambda \text{net}_i)} - 1 \quad \dots(2)$$

$$f'(\text{net}_i) = \frac{1}{2} (1 - o_i^2) \quad \dots(3)$$

$$\Delta W_i = c(d - o_i)f'(\text{net}_i)X \quad \dots(4)$$

Step 1 : Take 1st training pair, $X = X_1$ and $d = d_1$

$$X = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}, d = -1$$

$$\text{Compute } \rightarrow \text{net}_1 = W_1^T X = [1 \ -1 \ 0 \ 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = [1 + 2 - 0.5] = 2.5$$

$$o_1 = f(\text{net}_1) = \frac{2}{1 + \exp(-2.5)} - 1 = \frac{2}{1 + 0.0820} - 1 = 0.848$$

$$f'(\text{net}_1) = \frac{1}{2} (1 - o_1^2) = \frac{1}{2} (1 - 0.848^2) = 0.140$$

$$\Delta W_1 = c(d - o_1) f'(net_1) X = (0.1)(-1 - 0.848)(0.14) \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}$$

$$= -0.026 \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.026 \\ 0.052 \\ 0 \\ 0.026 \end{bmatrix}$$

$$\begin{aligned} W_2 &= W_1 + \Delta W_1 \\ &= \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + \begin{bmatrix} -0.026 \\ 0.052 \\ 0 \\ 0.026 \end{bmatrix} = \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix} \end{aligned}$$

Step 2: Take 2nd training pair, $X = X_2$ and $d = d_2$

$$X = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}, d = -1$$

$$\text{Compute } \rightarrow net_2 = W_2^T X = [0.974 \ -0.948 \ 0 \ 0.526] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= -1.422 - 0.526$$

$$= -1.948$$

$$O_2 = f(net_2) = \frac{2}{1 + \exp(-1.948)} - 1 = \frac{2}{8.0146} - 1 = -0.75$$

$$f'(net_2) = \frac{1}{2}(1 - O_2^2) = \frac{1}{2}(1 - (-0.75)^2) = 0.2187$$

$$\Delta W_2 = c(d - o_2) f'(net_2) X = (0.1)(-1 - (-0.75)) \times (0.2187) \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}$$

$$= (-0.00546) \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.00819 \\ 0.00273 \\ 0.00546 \end{bmatrix}$$

$$W_3 = W_2 + \Delta W_2 = \begin{bmatrix} 0.974 \\ -0.948 \\ 0 \\ 0.526 \end{bmatrix} + \begin{bmatrix} 0 \\ -0.00819 \\ 0.00273 \\ 0.00546 \end{bmatrix} = \begin{bmatrix} 0.974 \\ -0.956 \\ 0.0027 \\ 0.5315 \end{bmatrix}$$

Step 3 : Take 3rd training pair, $X = X_3$ and $d = d_3$

$$X = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}, d = 1$$

$$\text{Compute } \rightarrow net_3 = W_3^T X = [0.974 \ -0.956 \ 0.0027 \ 0.531] \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$



$$= -0.974 - 0.956 + 0.00135 - 0.5315 = -2.462$$

$$o_3 = f(\text{net}_3) = \frac{2}{1 + \exp(2.462)} - 1 = \frac{2}{12.728} - 1 = -0.843$$

$$f'(\text{net}_3) = \frac{1}{2}(1 - o_3^2) = \frac{1}{2}(1 - (-0.843)^2) = 0.145$$

$$\Delta W_3 = (0.1)(d - o_3)f'(\text{net}_3)X = (0.1)(1 - (-0.843))(0.145) \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

$$\Delta W_3 = (0.0267) \begin{bmatrix} -1 \\ +1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.0267 \\ 0.0267 \\ 0.0133 \\ -0.0267 \end{bmatrix}$$

$$W_4 = W_3 + \Delta W_3 = \begin{bmatrix} 0.974 \\ -0.956 \\ 0.0027 \\ 0.5315 \end{bmatrix} + \begin{bmatrix} -0.0267 \\ 0.0267 \\ 0.0133 \\ -0.0267 \end{bmatrix} = \begin{bmatrix} 0.947 \\ -0.929 \\ 0.016 \\ 0.504 \end{bmatrix}$$

Ex. 5.8.7 : Perform two training steps of the network using delta learning rule for $\lambda = 1$, $c = 0.25$. Train the network using the following data pairs.

$$\left(X_1 = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, d_1 = -1 \right) \left(X_2 = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}, d_2 = +1 \right). \text{ The initial weights are } W_1 = [1 \ 0 \ 1]^t$$

Soln. :

For delta learning rule

$$\text{net}_i = W_i^t X \quad \dots(1)$$

$$o_i = f(\text{net}_i) = \frac{2}{1 + \exp(-\lambda \text{net}_i)} - 1 \quad \dots(2)$$

$$f'(\text{net}_i) = \frac{1}{2}(1 - o_i^2) \quad \dots(3)$$

$$\Delta W_i = c(d - o_i) \left[\frac{1}{2}(1 - o_i^2) \right] \quad \dots(4)$$

Step 1 : Take 1st training pair $X = X_1$ and $d = d_1$

$$X = \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}, d = -1$$

$$\text{Compute} \rightarrow \text{net}_1 = W_1^t X = [1 \ 0 \ 1] \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = 2 - 1 = 1$$

$$o_1 = \frac{2}{1 + \exp(-1)} - 1 = 0.463$$

$$f'(\text{net}_1) = \frac{1}{2}(1 - (0.463)^2) = 0.393$$

$$\Delta W_1 = (0.25)(-1 - 0.463)(0.393) \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix}$$



$$= (-0.1437) \begin{bmatrix} 2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.287 \\ 0 \\ 0.1437 \end{bmatrix}$$

$$W_2 = W_1 + \Delta W_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.287 \\ 0 \\ 0.1437 \end{bmatrix} = \begin{bmatrix} 0.713 \\ 0 \\ 1.1437 \end{bmatrix}$$

Step 2 : Take 2nd training pair, $X = X_2$ and $d = d_2$

$$X = \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}, d = 1$$

Compute $\rightarrow \text{net}_2 = W_2^T X = [0.713 \quad 0 \quad 1.1437] \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix}$

$$= [0.713 \quad -1.1437] = -0.4307$$

$$o_2 = -0.2119$$

$$f'(\text{net}_2) = \frac{1}{2} (1 - (-0.2119)^2) = \frac{1}{2} (1 - 0.0449) = 0.477$$

$$\Delta W_2 = (0.25) (1 - (-0.2119)) \times (0.477) X_2$$

$$= (0.145) \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.145 \\ -0.29 \\ -0.145 \end{bmatrix}$$

$$W_3 = W_2 + \Delta W_2 = \begin{bmatrix} 0.713 \\ 0 \\ 1.143 \end{bmatrix} + \begin{bmatrix} 0.145 \\ -0.29 \\ -0.145 \end{bmatrix} = \begin{bmatrix} 0.858 \\ -0.29 \\ 0.998 \end{bmatrix}$$

Ex. 5.8.8 : A single neuron network using $f(\text{net}) = \text{sgn}(\text{net})$ has been trained using the pairs of (x_i, d_i) as given below :
 $X_1 = [1 \quad -2 \quad 3 \quad -1]^T, d_1 = -1$

$$X_2 = [0 \quad -1 \quad 2 \quad -1]^T, d_2 = 1$$

$$X_3 = [-2 \quad 0 \quad -3 \quad -1]^T, d_3 = -1$$

The final weights obtained using perception rule are :

$$W_4 = [3 \quad 2 \quad 6 \quad 1]^T$$

Knowing that correction has been performed at each step for $c = 1$, determine the following weights :

- (a) W_3, W_2, W_1 by backtracking the training
- (b) W_5, W_6, W_7 obtained for step 4, 5, 6 of training by reusing the sequence $(X_1, d_1), (X_2, d_2), (X_3, d_3)$.

MU - Dec. 11, 10 marks

Soln. :

(a) Backtracking

Step 3 : $W_4 = \begin{bmatrix} 3 \\ 2 \\ 6 \\ 1 \end{bmatrix}$

$$W_4 = W_3 + \Delta W_3$$

$$W = c \cdot (d_3 - o_3) \cdot X_3$$



We know that the correction has been performed at every step this means, actual output and desired outputs were not same.

This means they are exactly opposite of each other.

$$\text{Since } d_1 = -1 \Rightarrow o_1 = +1$$

$$d_2 = +1 \Rightarrow o_2 = -1$$

$$d_3 = -1 \Rightarrow o_3 = +1$$

$$\therefore \Delta W_3 = (1)(-1-1) \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 6 \\ 2 \end{bmatrix}$$

$$\therefore W_4 = W_3 + \Delta W_3$$

$$\therefore W_3 = W_4 - \Delta W_3$$

$$= \begin{bmatrix} 3 \\ 2 \\ 6 \\ 1 \end{bmatrix} - \begin{bmatrix} 4 \\ 0 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

$$\text{Step 2 : } W_3 = W_2 + \Delta W_2$$

$$\therefore \Delta W_2 = c(d_2 - o_2) X_2$$

$$= (1)(1 - (-1)) \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix}$$

$$\text{Now } W_2 = W_3 - \Delta W_2$$

$$= \begin{bmatrix} -1 \\ 2 \\ 0 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} -1 \\ 4 \\ -4 \\ 1 \end{bmatrix}$$

$$\text{Step 1 : } W_2 = W_1 + \Delta W_1 \quad \text{and } \Delta W_1 = c(d_1 - o_1) X_1$$

$$= (1)(1 - (-1)) \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = (-2) \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix}$$

$$\therefore W_1 = W_2 - \Delta W_1$$

$$= \begin{bmatrix} -1 \\ 4 \\ -4 \\ 1 \end{bmatrix} - \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \end{bmatrix}$$



Thus,

$$W_1 = \begin{bmatrix} 1 \\ 0 \\ 2 \\ -1 \end{bmatrix}, W_2 = \begin{bmatrix} -1 \\ 4 \\ -4 \\ 1 \end{bmatrix}, W_3 = \begin{bmatrix} -1 \\ 2 \\ 0 \\ -1 \end{bmatrix}$$

(b) Find W_5 , W_6 and W_7

Step 4 : Take first training pair

set $X = X_1$ and $d = d_1$

$$X = \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} \text{ and } d = -1$$

$$\text{net}_4 = W_4^T X$$

$$= [3 \ 2 \ 6 \ 1] \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix}$$

$$= 3 - 4 + 18 - 1 = 16$$

$$o_4 = \text{sgn}(\text{net}_4) = +1$$

$$\Delta W_4 = c(d - o_4) \cdot X$$

$$= (1)(-1 - 1) \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = (1)(-2) \begin{bmatrix} 1 \\ -2 \\ 3 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix}$$

$$\therefore W_5 = W_4 + \Delta W_4 = \begin{bmatrix} 3 \\ 2 \\ 6 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ 4 \\ -6 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 3 \end{bmatrix}$$

Step 5 : Take second training pair,

set $X = X_2$ and $d = d_2$

$$X = \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} \quad d = 1$$

$$\text{net}_5 = W_5^T X = [1 \ 6 \ 0 \ 3] \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix}$$

$$= 0 - 6 + 0 - 3 = -9$$



$$o_5 = \text{sgn}(-9) = -1$$

$$\Delta W_5 = c(d - o_5) \cdot X$$

$$= (1)(1 - (-1)) \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} = 2 \begin{bmatrix} 0 \\ -1 \\ 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix}$$

$$\therefore W_6 = W_5 + \Delta W_5$$

$$= \begin{bmatrix} 1 \\ 6 \\ 0 \\ 3 \end{bmatrix} + \begin{bmatrix} 0 \\ -2 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

Step 6 : Take third training pair,

$$\text{set } X = X_3 \quad \text{and} \quad d = d_3$$

$$X = \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} \quad d = -1$$

$$\text{net}_6 = W_6^T X$$

$$= [1 \ 4 \ 4 \ 1] \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} = -2 + 0 - 12 - 1 = -14$$

$$o_6 = \text{sgn}(-14) = -1$$

$$\Delta W_6 = c(d - o_6) \cdot X = (1)(-1 - (-1)) \begin{bmatrix} -2 \\ 0 \\ -3 \\ -1 \end{bmatrix} = 0$$

$$\therefore W_7 = W_6 = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

$$\text{Hence } W_5 = \begin{bmatrix} 1 \\ 6 \\ 0 \\ 3 \end{bmatrix} \quad W_6 = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix} \quad W_7 = \begin{bmatrix} 1 \\ 4 \\ 4 \\ 1 \end{bmatrix}$$

Ex. 5.8.9 : Apply Back propagation Algorithm to find the final weights for the following net. Inputs : $x = [0.0, 1.0]$, Weights between hidden and output layers : $w = [0.4, 0.2]$, Bias on the Output Node O is $W_O = [-0.4]$. Weights between input and hidden layer : $v = [2, 1 ; 1.2]$, Bias on Hidden Unit nodes are $V_O = [0.1 \ 0.3]$, Desired output : $d = 1.0$.

MU - Dec. 15, 10 Marks

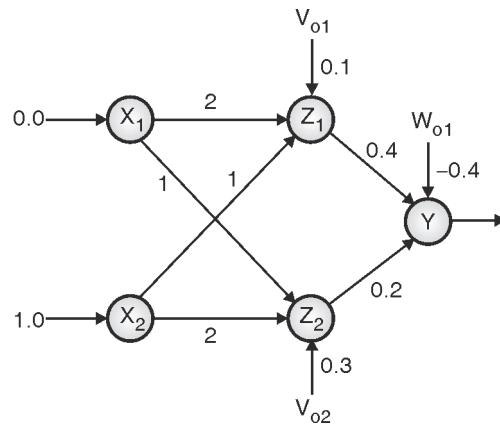


Fig. P.5.8.9

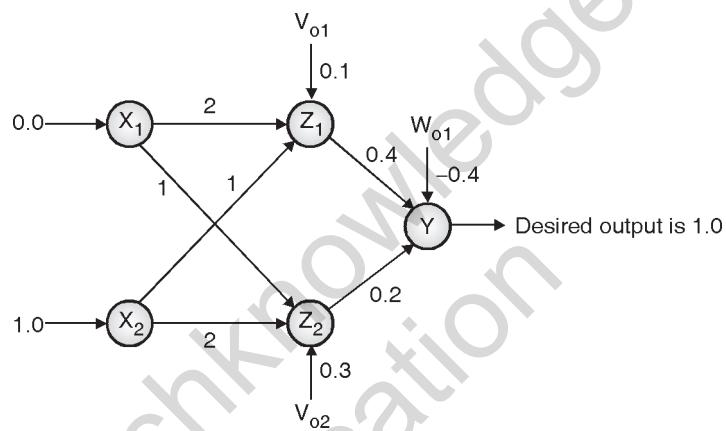
Soln. :

Fig. P. 5.8.9(a)

(I) Forward Pass :

Computing forward activations.

- (i) Computing hidden layer activations (
- Z_1
- and
- Z_2
-)

$$\begin{aligned} \text{net } z_1 &= V_{11}X_1 + V_{12}x_2 + V_{01} \\ &= 2(0) + 1(1) + 0.1 \\ &= 0 + 1 + 0.1 \\ &= 1.1 \\ Z_1 &= f(\text{net}_z_1) \end{aligned}$$

Using bipolar continuous activation function, we have

$$\begin{aligned} f(\text{net}) &= \frac{2}{1 + e^{-\lambda \text{net}}} - 1 && [\text{Assume } \lambda = 1] \\ &= \frac{2}{1 + e^{-\text{net}}} - 1 \\ Z_1 = f(\text{net}_z_1) &= \frac{2}{1 + e^{-\text{net}_z_1}} - 1 = \frac{2}{1 + e^{-1.1}} - 1 \\ z_1 &= 0.5005 \end{aligned}$$

$$\text{Net } z_2 = v_{21}x_1 + v_{22}x_2 + V_{o2}$$

$$= 1(0) + 2(1) + 0.3$$

$$= 2.3$$

$$z_2 = f(\text{net } z_2) = \frac{2}{1+e^{-2.3}} - 1$$

$$z_2 = 0.8178$$

(ii) Computing output layer activations (y)

$$\text{net}_y = w_1 z_1 + w_2 z_2 + w_{01}$$

$$= 0.4(0.5005) + 0.2(0.8178) + (-0.4)$$

$$\text{net}_y = -0.03624$$

$$y = f(\text{net}_y) = \frac{2}{1+e^{-0.03624}} - 1$$

$$y = -0.0181$$

(II) Backward Pass :

Computing error signal vectors for both layers.

(i) Computing error signal terms δ_{ok} for output layer

$$\delta_{ok} = \frac{1}{2} (d_k - o_k) (1 - o_k^2), \text{ for } k = 1, 2, \dots, K$$

$$\delta_y = \frac{1}{2} (d - y) (1 - y^2) \quad \text{Since, there is only one neuron 'y'}$$

$$\delta_y = \frac{1}{2} (1 - (-0.0181)) (1 - (-0.0181)^2)$$

$$\delta_y = 0.5089$$

(ii) Computing error signal terms of the hidden layer (δz_j)

$$\delta_{zj} = \frac{1}{2} (1 - z_j^2) \sum_{k=1}^K \delta_{ok} w_{kj}, j = 1, 2, \dots, J$$

$$\begin{aligned} \delta_{z1} &= \frac{1}{2} (1 - z_1^2) \delta_y w_1 \\ &= \frac{1}{2} (1 - 0.5005^2) (0.5089) (0.4) \end{aligned}$$

$$\delta_{z1} = 0.0763$$

$$\begin{aligned} \delta_{z2} &= \frac{1}{2} (1 - z_2^2) \delta_y w_2 \\ &= \frac{1}{2} (1 - 0.8178^2) (0.5089) (0.2) \end{aligned}$$

$$\delta_{z2} = 0.0169$$

(iii) Adjusting output layer weights (w_1 and w_2)

$$W_{kj} = W_{kj} + \eta \delta_y z_j \quad (\text{Assume } \eta = 1)$$

$$W_{1\text{new}} = W_{1\text{old}} + \delta_y z_1$$



$$= 0.4 + (0.5089) (0.5005)$$

$$\mathbf{w}_1 = \mathbf{0.6547}$$

$$\mathbf{w}_2 = \mathbf{w}_2 + \delta_y z_2$$

$$= 0.2 + (0.5089) (0.8178)$$

$$\mathbf{w}_2 = \mathbf{0.6162}$$

$$\text{Bias } w_{01} = w_{01} + \delta_y (1)$$

$$= -0.4 + 0.5089$$

$$W_{01} = 0.1089$$

(iv) Adjusting hidden layer weights ($v_{11}, v_{12}, v_{21}, v_{22}$)

$$v_{ji} = v_{ji} + \eta \delta_{zj} x_i \text{ for } j = 1, 2 \text{ and } i = 1, 2, \dots, I$$

$$v_{11} = v_{11} + \delta_{z1} x_1 = 2 + 0.0763 (0) = 2$$

$$v_{12} = v_{12} + \delta_{z1} x_2 = 1 + 0.0763(1) = 1.0763$$

$$v_{21} = v_{21} + \delta_{z2} x_1 = 1 + 0.0169 (0) = 1$$

$$v_{22} = v_{22} + \delta_{z2} x_2 = 2 + 0.0169(1) = 2.0169$$

$$v_{o1} = v_{o1} + \delta_{z1} (1) = 0.1 + 0.0763(1) = 0.1763$$

$$v_{o2} = v_{o2} + \delta_{z2} (1) = 0.3 + 0.0169(1) = 0.3169$$

$$\text{Cycle error } E = \frac{1}{2} (d - y)^2 + E = \frac{1}{2} (1 - (-0.0181))^2 = 0.5183$$

Final weights:

$$w_1 = 0.6547, w_2 = 0.6162, w_{01} = 0.1089, v_{11} = 2, v_{12} = 1.0763$$

$$v_{21} = 1, v_{22} = 2.0169, v_{o1} = 0.1763, v_{o2} = 0.3169$$

Review Questions

- Q. 1 Explain linearly separable and non-linearly separable patterns with examples.
- Q. 2 Explain Error back Propagation Algorithm with the help of flowchart.
- Q. 3 Write a short note on Kohonen self organizing network.
- Q. 4 Explain McCulloch Pitts Neuron Model with help of an example.
- Q. 5 A neuron with 4 inputs has the weight vector $w = [1, 2, 3, 4]^t$. The activation function is linear, that is, the activation function is given by $f(\text{net}) = 2 * \text{net}$. If the input vector is $X = [5, 6, 7, 8]^t$ then find the output of the neuron.
- Q. 6 Explain Error back propagation training algorithm with the help of a flowchart.
- Q. 7 A single neuron network using $f(\text{net}) = \text{sgn}(\text{net})$ has been trained using the pairs of (X_i, d_i) as given below :

$$X_1 = [1 - 2 3 - 1]^t, d_1 = -1$$

$$X_2 = [0 - 1 2 - 1]^t, d_2 = 1$$

$$X_3 = [-2 0 -3 -1]^t, d_3 = -1$$

The final weights obtained using the perceptron rule are :

$$W_4 = [3 \ 2 \ 6 \ 1]^t$$

Knowing that correction has been performed in each step for $c = 1$, determine the following weights :

- (a) W_3, W_2, W_1 by backtracking the training.
- (b) W_5, W_6, W_7 obtained for steps 4, 5, 6 of training by reusing the sequence $(X_1, d_1), (X_2, d_2), (X_3, d_3)$,

Q. 8 Explain with example perceptron learning rule.

Q. 9 Determine the weights after one iteration for hebbian learning of a single neuron network starting with initial weights $w = [1, -1]$, inputs as $X_1 = [1, -2]$, $X_2 = [2, 3]$, $X_3 = [1, -1]$ and $c = 1$.

- Use
- (i) Bipolar binary activation function
 - (ii) Bipolar continuous activation function.

Q. 10 Explain Single Continuous Perceptron Training Algorithm (SCPTA).

Q. 11 Explain with neat diagram supervised and unsupervised learning.

Q. 12 A neuron with 3 inputs has the weight vector $w = [0.1 \ 0.3 \ -0.2]$. The activation function is binary sigmoidal activation function. If input vector is $[0.8 \ 0.6 \ 0.4]$ then find the output of neuron.

Q. 13 Differentiate between supervised and unsupervised learning.

Q. 14 Explain Kohonen's self organizing neural network.

Q. 15 Determine the weights after four steps of training for perceptron learning rule of a single neuron network starting with initial weights :

$$W = [0 \ 0]^t, \text{ inputs as } X_1 = [2 \ 2]^t,$$
$$X_2 = [1 \ -2]^t, X_3 = [-2 \ 2]^t, X_4 = [-1 \ 1]^t,$$
$$d_1 = 0, d_2 = 1, d_3 = 0, d_4 = 1 \text{ and } c = 1.$$

Q. 16 Prove the following identities :

- (i) For unipolar continuous activation function

$$f^1(\text{net}) = 0(1 - 0)$$

- (ii) For bipolar continuous activation function :

$$f^1(\text{net}) = \frac{1}{2}(1 - 0^2)$$

Q. 17 What is learning in neural networks ? Differentiate between supervised and unsupervised learning.





Expert System

Syllabus

- 6.1 Hybrid Approach - Fuzzy Neural Systems
- 6.2 Expert system : Introduction, Characteristics, Architecture, Stages in the development of expert system,

6.1 Hybrid Approach

6.1.1 Introduction to Hybrid Systems

- Hybrid systems are those for which more than one soft computing technique is integrated to solve a real-world problem.
- Neural networks, fuzzy logic and genetic algorithms are soft computing techniques which have been inspired by biological computational processes.
- Each of these technologies has its own advantages and limitations. For example, while neural networks are good at recognizing patterns, they are not good at explaining how they reach their decisions.
- Fuzzy logic systems, which can reason with imprecise information, are good at explaining their decisions but they cannot automatically acquire the rules they use to make those decisions.
- These limitations have been a central driving force behind the creation of intelligent hybrid systems where two or more techniques are combined in a manner that overcomes the limitations of individual techniques.

6.1.2 Types of Hybrid Systems

Hybrid systems can be classified as:

1. Sequential hybrids
2. Auxiliary hybrids
3. Embedded hybrids

1. Sequential hybrid systems

- In sequential hybrid systems, all the technologies are used in a pipe-line fashion. The output of one technology becomes the input to another technology (Fig. 6.1.1).
- This kind of hybridization form is one of its weakest, because it does not integrate different technologies into a single unit.

- An example is a GA pre-processor which obtains the optimal parameters such as initial weights, threshold, learning rate etc. and hands over these parameters to a neural network.

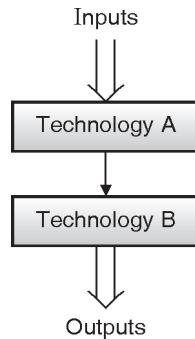


Fig.6.1.1 : A sequential hybrid system

2. Auxiliary hybrid systems

- In this type, a technology treats another technology as a “subroutine” and calls it to process or generate whatever information is needed by it. Fig. 6.1.2 illustrates the auxiliary hybrid system.

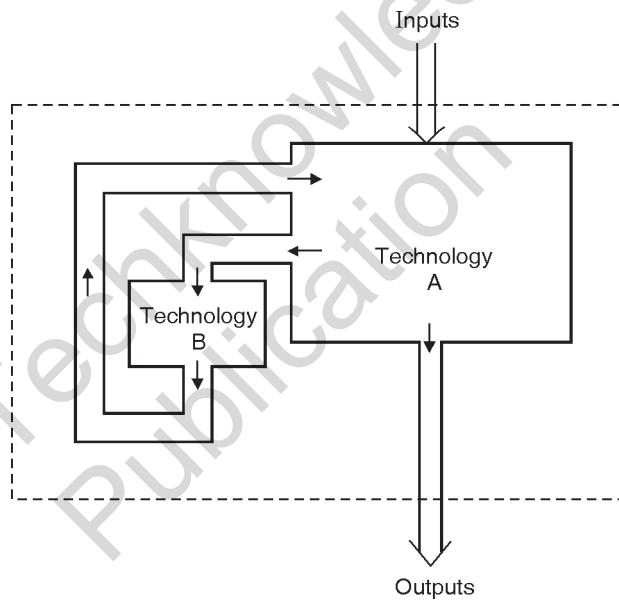


Fig. 6.1.2 : An auxiliary system

- An example is a neuro - genetic system in which an NN employs a GA to optimize its structural parameters, i.e. parameters which define Neural Network's architecture.

3. Embedded hybrid systems

- In embedded hybrid systems, the technologies are integrated in such a manner that they appear to be intertwined.
- The fusion is so complete that it would appear that no technology can be used without the other for solving the problem. Fig. 6.1.3 depicts the schema for an embedded hybrid system.
- Example of this system is a NN-FL (Neural Network Fuzzy Logic) hybrid system that has NN which receives information, processes it and generates fuzzy outputs as well.

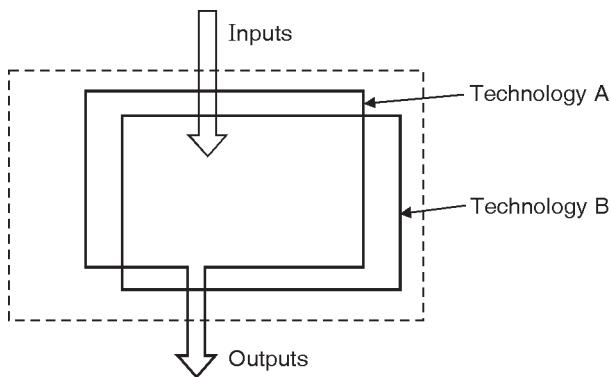


Fig. 6.1.3 : An embedded system

6.1.3 Fuzzy Neural System

- Fuzzy neural system is a system with seamless integration of fuzzy logic and neural networks.
- While fuzzy logic provides an inference mechanism under cognitive uncertainty, Neural networks offer advantages, such as learning, adaptation, fault-tolerance, parallelism and generalization.
- To enable a system to deal with cognitive uncertainties like humans, we need to incorporate the concept of fuzzy logic into the neural networks.
- The computational process for fuzzy neural systems is as follows.
- The first step is to develop a “fuzzy neuron” based on the understanding of biological neuronal morphologies.
- This leads to the following three steps in a fuzzy neural computational process.
 1. Development of fuzzy neural models.
 2. Models of synaptic connections which incorporates fuzziness into neural network
 3. Development of learning algorithms, which helps in adjusting the synaptic weights.
- There are two possible models of fuzzy neural systems:

(a) Model 1 :

- In this model the output of fuzzy system is fed as an input to the neural networks.
- The input to the system is linguistic statements. The fuzzy interface block converts these linguistic statements into an input vector which is then fed to a multi-layer neural network. The neural network can be adapted (trained) to yield desired command outputs or decisions.

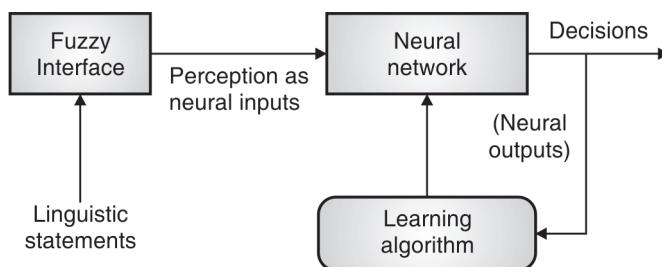


Fig 6.1.4 Fuzzy Neural System : Model 1

(b) Model 2 :

- In second model, a multi-layered neural network drives the fuzzy inference mechanism. That is the output of neural networks is fed as an input to the fuzzy system.

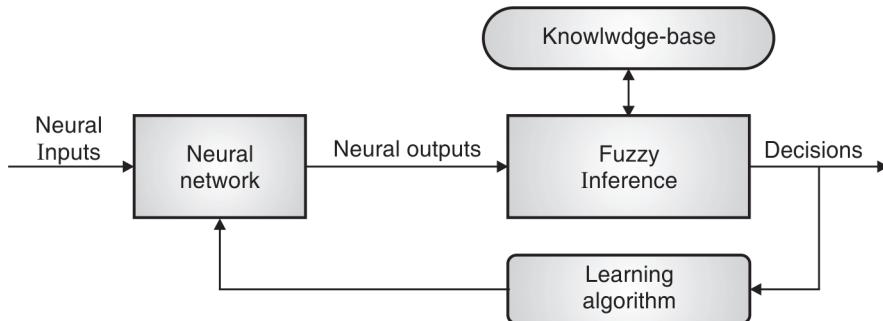


Fig 6.1.5 Fuzzy Neural System : Model 2

- Here, neural networks are used to tune membership functions of fuzzy systems that are employed as decision-making systems for controlling equipment. Neural network learning techniques can automate this process and substantially reduce development time and cost while improving performance.

6.1.4 Adaptive Neuro - Fuzzy Inference System (ANFIS)

MU – Dec. 14, Dec. 15

- | | |
|---|---------------------|
| Q. Explain the architecture of ANFIS with the help of a diagram. | (Dec. 14, 10 Marks) |
| Q. Draw the five layer architecture of ANFIS and explain each layer in brief. | (Dec. 15, 5 Marks) |

- A well-known and practically used Neuro-Fuzzy system is ANFIS (Adaptive Neuro –fuzzy Inference system) explained below. It is based on Takagi–Sugeno fuzzy inference system developed in 1990s.
- ANFIS is a neural network which is functionally equivalent to a fuzzy inference model.
- Since it integrates both neural networks and fuzzy logic, it has the potential to capture the benefits of both in a single framework.

ANFIS Architecture:

- Fig. 6.1.6 shows the first-order Sugeno model and its equivalent ANFIS architecture is shown in Fig. 6.1.7.

$$\begin{aligned}
 f_1 &= p_1x + q_1y + r_1 \\
 \Rightarrow f &= \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \bar{w}_1 f_1 + \bar{w}_2 f_2 \\
 f_2 &= p_2x + q_2y + r_2
 \end{aligned}$$

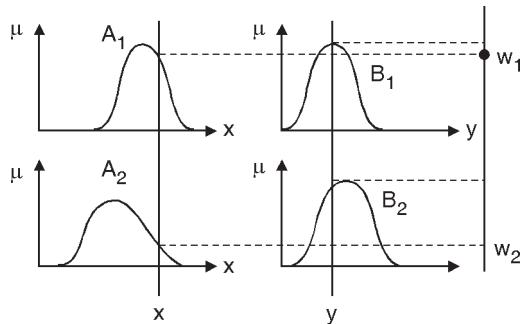


Fig. 6.1.6 : A two-input first-order Sugeno model

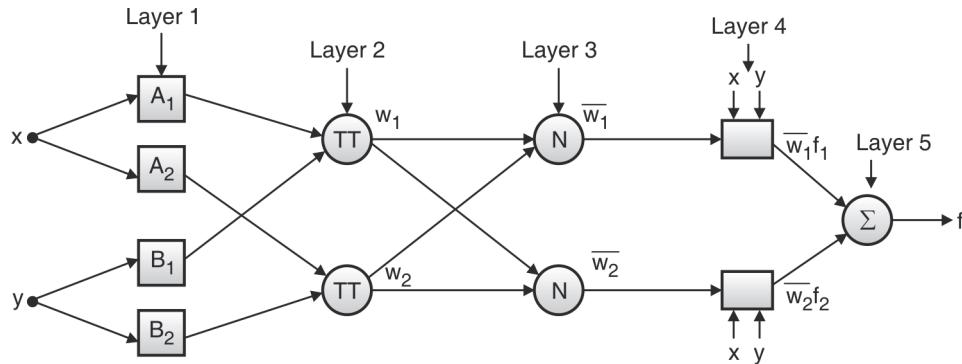


Fig. 6.1.7 : Equivalent ANFIS architecture

- Assume that the fuzzy inference system under consideration has two inputs x and y and one output.
- For a first-order Sugeno fuzzy model, a common rule set with two fuzzy rules is,
 - Rule 1:** If x is A_1 and y is B_1 then $f_1 = p_1x + q_1y + r_1$
 - Rule 2:** If x is A_2 and y is B_2 then $f_2 = p_2x + q_2y + r_2$
- The architecture contains five layers.

Layer 1 : Fuzzification layer

- Neurons in this layer represent fuzzy sets used in the antecedents of fuzzy rules. A neuron in the fuzzification layer receives a crisp input and determines the degree to which this input belongs to the neuron's fuzzy set.
- Every node in this layer is an **adaptive node** and a node function can be represented as,

$$\begin{aligned} O_{1,i} &= \mu_{A_i}(x), \text{ for } i = 1, 2 \text{ or} \\ O_{1,j} &= \mu_{B_{i-2}}(y), \text{ for } i = 3, 4 \end{aligned}$$

Where x (or y) are the input to the node i and A_i (or B_{i-2}) is a linguistic label associated with this node.

- Any appropriate membership function can be used such as generalized bell function

$$\mu_A(x) = \frac{1}{1 + \left| \frac{x - c_i}{a_i} \right|^{2b}}$$

where, $\{a_i, b_i, c_i\}$ is the parameter set. Parameters in this layer are called **premise parameters**.

Layer 2 : Fuzzy rule layer

- Every node in this layer is the **fixed node**.
- Each node (neuron) in this layer corresponds to a single fuzzy rule. A fuzzy rule neuron receives inputs from the fuzzification layer neurons that represent fuzzy sets in the rule antecedents.
- The output of i^{th} node is the product of all incoming signals.

$$O_{2i} = W_i = \mu_{A_i}(x) \mu_{B_i}(y), i = 1, 2$$

- Each node output represents the firing strength (W_i) of a rule.
- The number of neurons (or nodes) in this layer will be equal to the number of rules.

Layer 3 : Normalization layer

- Every node in this layer is a **fixed node**. The i^{th} node of the layer calculates the ratio of the i^{th} rule's firing strength to the sum of all rules' firing strengths.

$$O_{3,i} = \bar{w}_i = \frac{w_i}{w_1 + w_2}, i = 1, 2$$

- The output of this layer is the normalized firing strength.

Layer 4 : Output membership layer

- Nodes in this layer represent fuzzy sets used in the consequent of fuzzy rules.
- Every node in this layer is an **adaptive node** with a node function represented as,

$$O_{4,i} = \bar{w}_i f_i = \bar{w}_i (p_i x + q_i y + r_i), i=1, 2$$

where w_i is the normalized firing strength from layer 3 and $\{p_i, q_i, r_i\}$ is the parameter set of this node.

- Parameters in this layer are referred to as **consequent parameters**.

Layer 5 : Defuzzification layer

- Node in this layer is a **fixed node**.
- Every node in this layer represents a single output of the neuro-fuzzy system. It takes the output fuzzy sets clipped by the respective integrated firing strengths and combines them into a single fuzzy set.
- Output of this node can be represented as,

$$O_{5,1} = \sum \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}, i = 1, 2$$

- The structure of this adaptive network is not unique. The alternative structure is shown Fig. 6.1.8.

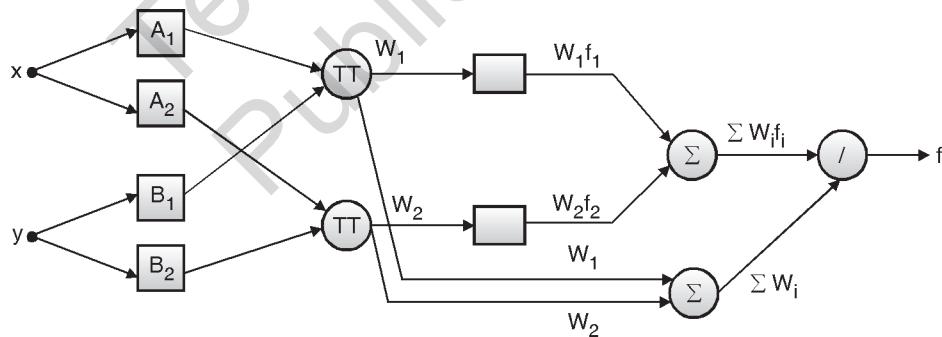


Fig. 6.1.8 : Alternative ANFIS architecture for Sugeno fuzzy model

- Here weight normalization is performed in the last layer.

6.2 Expert System

6.2.1 Introduction

- Jackson (1999) has defined Expert Systems as follows:
- “An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view to solve problems or give advice.”



- Artificial intelligent aims to implement intelligence in machines by developing computer programs that exhibit intelligent behavior. To develop systems which are capable of solving complex problems, it needs efficient access to substantial domain knowledge, a good reasoning mechanism and an effective and efficient way of representing knowledge and inferences; to apply the knowledge to the problems they are supposed to solve. Expert systems also need to explain to the users how they have reached their decisions.
- Expert systems are generally based on the way of knowledge representation, production rule formation, searching mechanism and so on. Usually, to build an expert system, system's shell is used, which is an existing knowledge independent framework, into which domain knowledge can be added to produce a working expert system. This avoids programming for each new system from scratch.
- Often, the two terms, Expert Systems (ES) and Intelligent Knowledge Based Systems (IKBS), are used synonymously. Expert systems are programs whose knowledge base contains the knowledge used by human experts, in place of knowledge gathered from textbooks or non-experts. Expert systems have the most widespread areas of artificial intelligence application.

6.2.2 Characteristics of Expert Systems

Following characteristics distinguish expert systems from conventional computer applications.

1. Simulation of human reasoning

Expert systems simulate human reasoning process in the given problem domain, whereas computer applications try to simulate the domain itself.

2. Representation of human knowledge

Expert systems use various methods to represent the domain knowledge gathered from human expert. It performs reasoning over representations of human knowledge, in addition to numerical calculations or data retrieval. In order to do this, expert systems have corresponding distinct modules referred to as the **inference engine** and the **knowledge base**. Whereas in case of computer applications it might be just the calculations performed on available data, without inference knowledge.

3. Use approximations

Expert systems tend to solve the problems using heuristics or approximate methods or probabilistic methods which are very much like how human do in general. While in case of computer applications strict algorithms are followed to produce solutions, most of the times which do not guarantee the result to be correct or optimal.

4. Provide explanations

Expert systems usually need to provide **explanations** and **justifications** of their solutions or recommendations in order to make user understand their reasoning process to produce a particular solution. This type of behavior is hardly observed in case of computer applications.

Some typical tasks performed by existing expert system are as follows:

- **Data interpretation** : There are different types of data to be interpreted by expert system, which have various formats and features. Example: sonar data, geophysical measurements.
- **Diagnosis of malfunctions** : While collecting data from machines or from experts there can be shortfall of accuracy or mistakes in readings. Example equipment faults or human diseases.



- **Structural analysis :** If system is build for a domain where complex objects like, chemical compounds, computer systems are used; configuration of these complex objects must be studied by the expert system.
- **Planning :** Expert systems are required to plan sequences of actions in order to perform some task. Example: actions that might be performed by robots.
- **Prediction :** Expert systems need to predict the future basis on past knowledge and current information available. Example: weather forecast, exchange rates, share prices, etc.
- **High Performance :** Expert systems are generally preferred because of their high performance. In the sense, they can process huge amount of data to produce results considering several details, in acceptable amount of time. The response time is very less.
- **Highly responsive :** Expert system are required to be highly responsive and user friendly. User can ask any query system should be able to produce the appropriate reply to it. Even if the query asked by user is not answerable with the existing knowledgebase, expert system should give some informative reply about the question.
- **Reliable :** Expert systems are highly reliable as they process huge amount of database. Hence the results produce by the system are always close to exact.

6.2.3 Real Time Expert Systems

There are no fundamental limits on what problem domains an expert system can be built to deal with. Expert systems can be developed almost for every domain for which requires human expert. However, the domain should ideally be one in which an expert requires a few hours to accomplish the task. This section explains some of the expert systems which have been created for such domains.

- (i) **Dendral** : This system is considered to be the first expert system in existence. Dendral identifies the molecular structure of unknown compounds. It was developed by Stanford University.
- (ii) **Mycin** : This is a milestone in expert system development which, made significant contributions to the medical field; but was not used in actual practice. It provides assistance to physicians in the diagnosis and treatment of meningitis and bacterial infections. It was developed by Stanford University.
- (iii) **Altrex** : It helps to diagnose engine troubles of certain models of Toyota cars, used in a central service department which can be called up by those actually servicing the cars for assistance, if required. It was developed by their research lab.
- (iv) **Prospector** : This expert system is successful to locate deposits of several minerals, including copper and uranium. It was developed by SRI International.
- (v) **Predicate** : This expert system provides estimate of the time required to construct high rise buildings. It was developed by Digital Equipment Corporation for use by Land Lease, Australia.

6.3 Building Blocks of Expert Systems

MU -Dec. 15, May 16

Q. Draw and describe the architecture of expert system.	(Dec.15, 6 Marks)
Q. Draw and explain architecture of expert system	(May 16, 5 Marks)

- Let's first understand the components of expert systems.

- Following are the two principal components of every expert system.

1. Knowledge base
2. The reasoning or inference engine

- Every expert system is developed for a specific task domain. It is the area where human intelligence is required. Task refers to some goal oriented problem solving activity and Domain refers to the area within which the task is being performed.
- Consider the expert system architecture in Fig. 6.3.1.

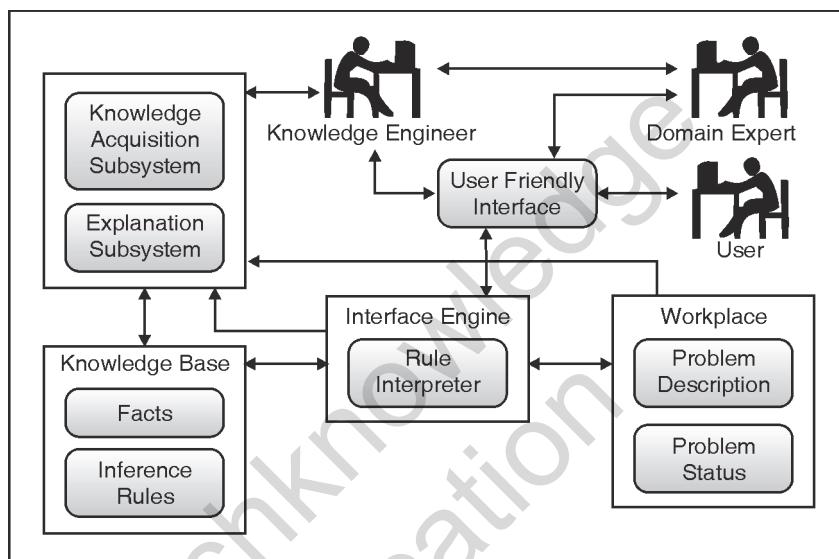


Fig. 6.3.1 : Expert System Architecture

- If we consider inference engine as the brain of the expert systems, then knowledge base is the heart. As the heart is more powerful, the brain can function faster and efficient way. Hence the success of any expert system is more or less depends on the quality of knowledgebase it works on.

1. Knowledge base

There are two types of knowledge expert systems can have about the task domain.

- (i) Factual knowledge
- (ii) Heuristic knowledge

- (i) **Factual knowledge :** It is the knowledge which is accepted widely as a standard knowledge. It is available in text books, research journals and internet. It is generally accepted and verified by domain experts or researchers of that particular field.
- (ii) **Heuristic knowledge :** It is experiential, judgmental and may not be approved or acknowledged publically. This type of knowledge is rarely discussed and is largely individualistic. It doesn't have standards for evaluation of its correctness. It is the knowledge of good practice, good judgment and probable reasoning in the domain. It is the knowledge that is based on the "art of good guessing." It is very subjective to the practitioner's knowledge, and experience in the respective problem domain.

The knowledge base an expert uses is based on his learning from various sources over a time period. Hence, the knowledge store of an expert increases with number of years of experience in the given field, which allows him to interpret the information in his databases to advantage in diagnosis, analysis and design.

2. Inference engine

The inference engine has a problem solving module which organizes and controls the steps required to solve the problem. A common but powerful inference engine involves chaining of IF...THEN rules to form a line of reasoning. There are two types of chaining practices to solve a problem.

1. **Forward Chaining:** This type of reasoning strategy starts from a set of conditions and moves toward some conclusion, also called as data driven approach.
2. **Backward Chaining :** Backward chaining is a goal driven approach. In this type of reasoning, the conclusion is known and the path to the conclusion needs to be found out. For example, a goal state is given, but the path to that state from start state is not known, then backward reasoning is used.

Inference engine is nothing but these methods implemented as program modules. Inference engine manipulates and uses knowledge in the knowledge base to generate a line of reasoning.

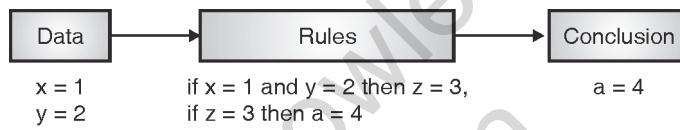


Fig. 6.3.2 : Forward Chaining

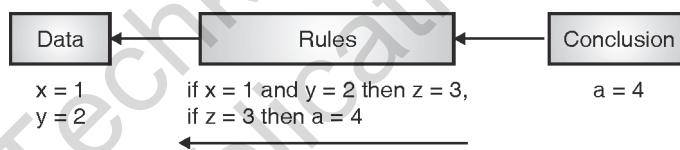


Fig. 6.3.3 : Backward Chaining

Knowledge is most of the times incomplete and uncertain. There are various ways to deal with uncertainty in knowledge. One of the simplest methods is to associate a weight or a confidence factor with each rule. The set of methods used to represent uncertain knowledge in combination with uncertain data in the reasoning process is called reasoning with uncertainty. “Fuzzy Logic” is an important area of artificial intelligence which provides methods for reasoning with uncertainty and the systems that use it are called as “fuzzy systems”.

3. **Explanation subsystem :** Most of the times, human experts can guess or use their gut feeling to approximate or estimate the results. As an expert system try to mimic human thinking, it uses uncertain or heuristic knowledge as we humans do. As a result, its credibility of the system is often in question, as is the case with humans. As an answer to a problem itself is questionable, one urges to know the rationale or the reasoning behind the answer. If the rationale seems to be acceptable, generally the answer is considered to be correct. So is the case with expert systems!!

Most of the expert systems have the ability to answer questions of the form: "Why is the answer X?" Inference engine can generate explanations by tracing the line of reasoning used by it.

4. **Knowledge engineer :** Building an expert system is also known as “knowledge engineering” and its practitioners are called knowledge engineers. The primary job of knowledge engineer is to make sure that the expert system has all the knowledge needed to solve a problem. He does a vital task of gathering knowledge from domain experts.

Knowledge engineer needs to learn how the domain expert reasons with their knowledge by interviewing them. Then he translates his knowledge into programs using which he designs the interface engine. There might be some uncertain knowledge involved in the knowledgebase; knowledge engineer needs to decide how to integrate this with the available knowledgebase. Lastly, he needs to decide upon what kind of explanations will be required by the user, and according to that he designs the inference levels.

6.4 Development Phases of Expert Systems

To develop an expert system following are the general steps followed. It is an iterative process. Steps in developing an expert system include:

1. Identify problem domain

- The problem must be suitable for an expert system to solve it.
- Find the experts in task domain for the ES project.
- Establish cost-effectiveness of the system.

2. Design the system

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.
- Realize how the concepts can represent the domain knowledge best.

3. Develop the prototype

- From Knowledge Base: The knowledge engineer works to –
- Acquire domain knowledge from the expert.
- Represent it in the form of If-THEN-ELSE rules.

4. Test and refine the prototype

- The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of the ES.

5. Develop and complete the ES

- Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
- Document the ES project well.
- Train the user to use ES.

6. Maintain the ES

- Keep the knowledge base up-to-date by regular review and update.
- Cater for new interfaces with other information systems, as those systems evolve.

6.5 Representing and Using Domain Knowledge

- Knowledge affects the development, efficiency, speed, and maintenance of the system. Knowledge representation is a way to transform human knowledge to machine understandable format. It is a very challenging task in expert systems, as the knowledge is very vast, unformatted and most of the times it is uncertain. Knowledge representation formalizes and organizes the knowledge required to build an expert system.



- The knowledge engineer must identify one or more forms in which the required knowledge should be represented in the system. He must also ensure that the computer can use the knowledge efficiently with the selected reasoning methods. As the quality of knowledge matters, the representation used also matters a lot as that will best allow a programmer to write the code for the system.
- A number of knowledge-representation techniques have been devised till date to represent the knowledge efficiently, but finally it depends on the application, the design of system. Few of the knowledge representation techniques are mentioned below.
 - Production rules
 - Decision trees
 - Semantic nets
 - Factor tables
 - Attribute-value pairs
 - Frames
 - Scripts
 - Logic
 - Conceptual graphs
- Out of these the most commonly used methods for representing domain knowledge are Production Rules, Semantic Nets and Frames. Let's have detail study of these methods.

a. Production Rules

- One widely used representation of knowledge is the set of production rules, or simply rule tree. A rule has a condition and an action associated with it. The condition part is identified by keyword "IF". It lists a set of conditions in some logical combination. Actions are specified in "THEN" part.
- As the IF part of the rule is satisfied; consequently, the THEN part actions can be taken. The piece of knowledge represented by the production rule is used to produce the line of reasoning. Expert systems whose knowledge is represented in rule form are called rule-based systems. We have studied rule-based agents named as simple reflex agents in chapter 1.
- As human thinking is evolved on the basis of situation→ conclusion or condition→action basis, this model is predominantly used representing knowledge in ES.
- IF-THEN rules are the simplest and efficient way to represent expert's knowledge. It takes following form.

IF $a_1, a_2, a_3, \dots, a_n$ THEN $b_1, b_2, b_3, \dots, b_n$

Where a_1, a_2, \dots are the situation or conditions and

b_1, b_2, \dots are the conclusions or actions.

Consider the following example :

Design advisor : [Steele et al., 1989] is a system that critiques chip designs. Its rules look like :

If : the sequential level count of ELEMENT is greater than 2,

UNLESS the signal of ELEMENT is resetable

then : critique for poor resetability

DEFEAT : poor resetability of ELEMENT

due to : sequential level count of ELEMENT greater than 2

by : ELEMENT is directly resettable

b. Semantic Nets

- A semantic net or semantic network is a knowledge representation technique used for propositional information, so it is also called a propositional net. In semantic networks the knowledge is represented as objects and relationships between objects. They are two dimensional representations of knowledge. It conveys meaning. Relationships provide the basic structure for organizing knowledge. It uses graphical notations to draw the networks. Mathematically a semantic net can be defined as a labelled directed graph. As nodes are associated with other nodes semantic nets are also referred to as associative nets.
- Semantic nets consist of nodes, links and link labels. Objects are denoted by nodes of the graph while the links indicate relations among the objects. Nodes can appear as circles or ellipses or rectangles to represent objects such as physical objects, concepts or situations. Links are drawn as arrows to express the relationships between objects, and link labels specify specifications of relationships.
- The two nodes connected to each other via a link are related to each other. The relationships can be of two types: "IS-A" relationship or "HAS" relationship. IS-A relationship stands for one object being "part of" the other related object. And "HAS" relationship indicates one object "consists of" the other related object. These relationships are nothing but the super class subclass relationships. It is assumed that all members of a subclass will inherit all the properties of their super classes. That's how semantic network allows efficient representation of inheritance reasoning.

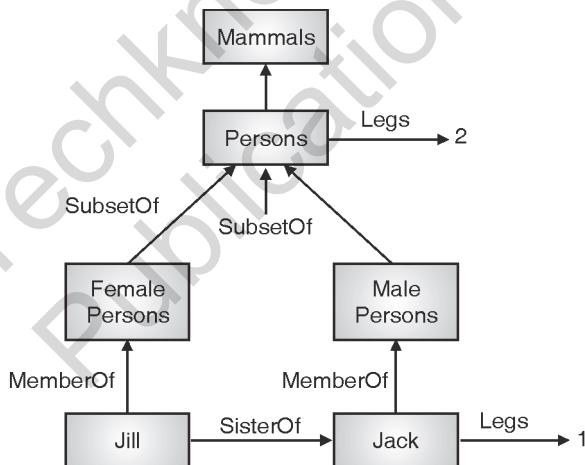


Fig. 6.5.1:Semantic net example

- For example Fig. 6.5.1 is showing an instance of a semantic net. In the Fig. 6.5.1 all the objects are within rectangles and connected using labeled arcs. The links are given labels according to the relationships. This makes the network more readable and conveys more information about all the related objects. For example, the "Member Of" link between Jill and Female Persons indicates that Jill belongs to the category of Female Persons.
- It does also indicate the inheritance among the related objects. Like, Jill inherits the property of having two legs as she belongs to the category of Female Persons which in turn belongs to the category of Persons which has a boxed Legs link with value 2.

- Semantic nets also can represent multiple inheritance through which, an object can belong to more than one objects and an object can be a subset of more than one another objects. It does also allows a common form of inference known as inverse links. The inverse links make the job of inference algorithms much easier to answer reverse queries.
- For example, in Fig. 6.5.1 there IS-A HAS Sister link which is the inverse of Sister Of link. If there is a query “such as who the sister of Jack ?”The inference system will discover that Has Sister is the inverse of Sister of, to make the inference algorithm follow the link HAS Sister from Jack to Jill and answer the query.

Advantages of Semantic Nets

1. Semantic nets are very expressive and minute details can be represented using semantic nets. For example, it can represent default values for categories. In Fig. 6.5.1, Jack has one leg can be represented by explicitly mentioning as shown. This specified value replaces the default value 2 for number of legs for person object.
2. Semantic nets can represent semantics of relationships in a transparent manner.
3. Semantic nets are simple and easy to understand.
4. Semantic nets are easy to implement using PROLOG.

Disadvantage of Semantic Nets

1. The links between the objects represent only binary relations. In relations where more than two objects are involved cannot be represented using semantic nets. For example, the sentence Run(Chennai Express, Chennai, Bangalore, Today) cannot be represented directly.
2. There is no standard definition of link names.

c. Frames

- Frames provide a convenient structure for representing objects that are typical to stereotypical situations. For example, visual scenes, structure of complex physical objects etc. In this technique knowledge is decomposed into highly modular format.
- Frame is a type of schema used in many Artificial Intelligence applications including vision and natural language processing. Frames are also useful for representing commonsense knowledge.
- As frames can represent concepts, situations, attributes of concepts, relationships between concepts, and also procedures to explain their relationships. It allows nodes to have structures and hence is regarded as 3-D representations of knowledge.
- A frame is also known as unit, schema, or list. Typically, a frame consists of a list of properties of the object and associated values for the properties ; similar to the fields and values; also called as slots and slot fillers. The contents of slot can be a string, numbers, functions, procedures, etc.
- A frame is a group of slots and fillers that defines a stereotypical object. Rather than a single frame, frame systems usually have collection of frames connected to each other. One of the attribute values can be another frame. For example, Fig. 6.5.2 shows a frame for a book object.

Slots	Fillers
Publisher	Techmax
Title	Intelligent Systems
Author	PurvaRaut
Edition	First
Year	15
Pages	275

Fig. 6.5.2 : A simple Frame for Book Object

- This is one of the simplest frames, but frames can have more complex structure in real world. A powerful knowledge system can be built with filler slots and inheritance provided by frames. Following Fig. 6.5.3 is the example for generic frame.

Slot	Fillers
Name	Laptop
specialization_of	a_kind_of machine
Types	(desktop, laptop, mainframe, super) if-added : Procedure ADD_LAPTOP
Speed	default: faster if-needed : Procedure FIND_SPEED
Location	(home, office, mobile)
under_warranty	(yes, no)

Fig.6.5.3 : Generic Frame Example

- From Fig.6.5.3 we can conclude that fillers are of various types and it may also include procedural attachments. Let's see what those types are.
 1. **Value** : "laptop" in the "name" slot.
 2. **Range** : "types" slot has (desktop, laptop, mainframe, super) as fillers.
 3. **If-needed** : It's a procedural attachment. It will be executed when a filler value is needed.
 4. **Default** : "Default" value is taken if no other value exists. It represents common sense knowledge, when no specific value is available.
 5. **If-added** : It's a procedural attachment. It is required if any value is to be added to a slot. In the above example, on arrival of a new type of laptop, ADD_LAPTOP procedure should be executed to add that information.
 6. **If-removed** : It's a procedural attachment. It is used to remove a value from the slot.
- Finally, the domain knowledge in an expert system will reside in a Knowledge Representation Language (KRL), such as an expert system shell. Let's understand what it is.

6.6 Expert System-Shell

- In early days each of the expert system was built from scratch. As numbers of expert systems are developed, researchers found that there are few parts of the systems which were common.
- As in each of those systems, there was an interpreter, which was developed separately for each system to interpret the rules. It was vividly noticed that, there can be a common generic interpreter developed independent of the domain.
- Also, there are only a handful of ways to represent knowledge, or to make inferences, or to generate explanations; it is better to have a generic system without any domain specific knowledge.
- Such systems are known as skeletal systems, shells, or simply AI tools. A new ES can be developed by adding domain knowledge to the shell.

- Fig. 6.6.1 depicts generic components of expert system shell. It includes Knowledge acquisition system, Knowledgebase, Inference engine, Explanation subsystem, and user interface. Knowledgebase and inference mechanism are the core components of shell.

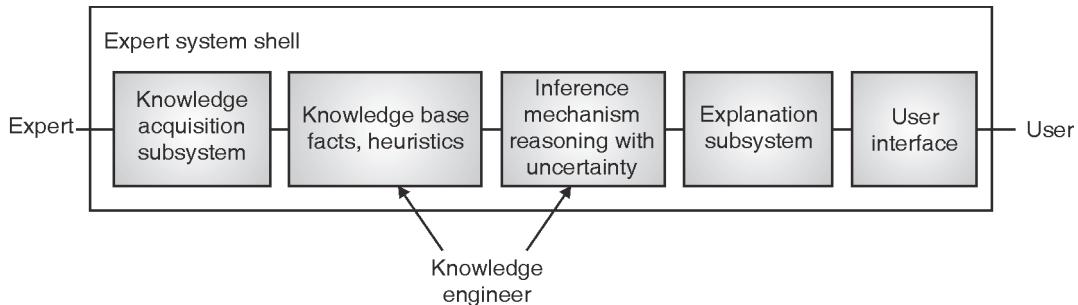


Fig. 6.6.1 : Components of Expert System Shell

- Let us understand in short what each component is, and what it is used for.
 1. **Knowledge acquisition system** : Knowledge acquisition is the first and the fundamental step in building an expert system. It helps to collect the experts knowledge required to solve the problems and build the knowledgebase. But this is also the biggest bottleneck of building ES.
 2. **Knowledge base** : This component is the heart of expert system. It stores all factual and heuristic knowledge about the application domain. It provides with the various representation techniques for all the data, as the programmers are required to program the system accordingly.
 3. **Inference mechanism** : Inference engine is the brain of expert system. This component is mainly responsible for generating inference from the given knowledge from the knowledgebase and produce line of reasoning in turn the result of the user's query.
 4. **Explanation subsystem** : This part of shell is responsible for explaining or justifying the final or intermediate result of user query. It is also responsible to justify need of additional knowledge.
 5. **User interface** : It is the means of communication with the user. Earlier it was not use to be a part of expert systems, as there was no significance associated with user interface. But later on it was also recognized as an important component of the system, as it decides the utility of expert system.
- Building expert systems by using shells has significant advantages. It is always advisable to use shell to develop expert system as it avoids building the system from scratch. To build an expert system using system shell, one needs to enter all the necessary knowledge about a task domain into a shell. The expert can himself create the knowledgebase by undergoing some training on how to use the shell. The inference engine that applies the knowledge to the given task is built into the shell.
- There are many commercial shells available today, “EMYCIN” derived from MYCIN ; “Inter modeller” used to develop educational expert systems, to name a few. There are variety of sizes for shells, starting from shells on PCs, to shells on large mainframe computers. They range in complexity from simple, forward-chained, rule-based systems. Accordingly to the size and complexity, shell price range from hundreds to tens of thousands of dollars. Application wise, they range from general-purpose shells to customized ones, such as financial planning or real-time process control.



6.7 Explanations

- Expert systems are developed with the aim of efficient and maximum utilization of technology in place of human expertise. To achieve this aim, along with the accuracy in the working, also the user interface must be good.
- User must be able to interact with system easily. To facilitate user interactions, system must possess following two properties:
 1. Expert systems must be able to explain its reasoning. In many cases user are interested in not only knowing the answers to their queries but also how the system has generated that answer. That will ensure the accuracy of the reasoning process that has produced those answers. Such kind of reasoning will be required typically in medical applications, where doctor needs to know why a particular medicine is advised for a particular patient, as he owns the ultimate responsibility for the medicines he subscribe. Hence, it is important that the system must store enough meta-knowledge about the reasoning process and should be able to explain it to the user in an understandable way.
 2. The system should be able update its old knowledge by acquiring new knowledge. As the knowledgebase is where the system's power resides, expert system should be able to maintain the complete, accurate and up to date knowledge about the domain. It is easy said than done!! As the system is programmed based on the available knowledgebase, it is very difficult to adapt to the changes in knowledgebase. It must have some mechanism through which the programs will learn its expert behavior from raw data. Another comparatively simple way is to keep on interacting with human experts and update the system.
- TEIRESIES was the first expert system with both these properties implemented in it. MYCIN used TEIRESIES as its user interface.
- As the TEIRESIAS-MYCIN system answers the user questions, he might be satisfied or might want to know the reasoning behind the answers. User can very well find it out by asking "HOW" question.
- The system will interpret it as "How do you know that?" and answers it by using backward chining starting from the answer to one of the given fact or rule. TEIRESIAS-MYCIN can do fairly good job in satisfying the user's query and providing proper reasoning for it.

Types of Explanations

There are four types of explanations generally the expert system is asked for there are:

1. Report on rule trace on progress of consultation.
2. Explanation on how the system has reached to a particular conclusion.
3. Explanation of why the system is asking question.
4. Explanation of why the system did not give any conclusion.

6.8 Knowledge Acquisition

- Knowledge is the most important ingredient in any expert system. The power of expert systems resides in the specific, high quality knowledge they contain about task domains. The more quality knowledge a system is given, the more competent it becomes.
- Even if expert systems shells simplify programming by providing a general skeleton; knowledge acquisition has to be done separately for each system. The choice of reasoning method, or a shell, is important, but it's equally important to accumulate high quality knowledge for developing any expert system. Knowledge engineers perform the task of knowledge acquisition.

- The knowledge acquisition component allows the expert to enter their knowledge or expertise into the expert system. It can be refined as and when required. Nowadays automated systems that allow the expert to interact directly with the system are becoming increasingly common, thereby reducing the work pressure of knowledge engineer.
- The knowledge acquisition process has three principal steps. Fig. 6.8.1 depicts them in a diagrammatic form. They are as follow :
 1. **Knowledge elicitation** : Knowledge engineer needs to interact with the domain expert and get all the knowledge. He also needs to format it in a systematic way so that it can be used while developing the expert system shell.
 2. **Intermediate knowledge representation** : The knowledge obtained from the domain expert needs to be stored in some intermediate representation, such that, it can be worked upon to produce the final refined version.
 3. **Knowledgebase representation** : The intermediate representation of the knowledge needs to be compiled and transformed into an executable format. This version of knowledge is ready to get uploaded to system shell as it is. e.g. production rules, that the inference engine can process.
- In the process of expert system development, numbers of iterations through these three stages are required in order to equip the system with good quality knowledge.

- The iterative nature of the knowledge acquisition process can be represented in Fig. 6.8.1.

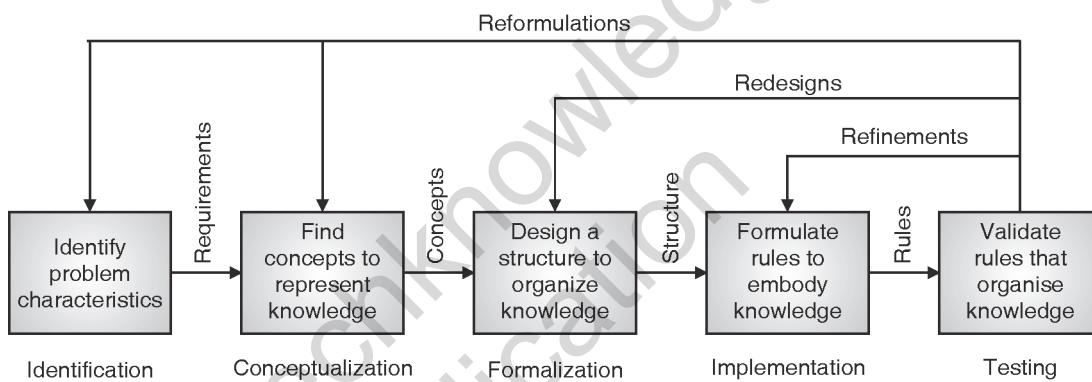


Fig. 6.8.1 : Stages of knowledge acquisition

- As quality of knowledge base determines the success of an expert system, AI researchers are continually exploring and adding new methods of knowledge representation and reasoning. Future of expert systems depends on breaking the knowledge acquisition bottleneck and codifying and representing a large knowledge infrastructure.

6.8.1 Knowledge Elicitation

- The knowledge elicitation is the first step of knowledge acquisition. In this process itself there are several stages. Generally knowledge engineer performs these steps.
- These steps need to be carried out before meeting the domain expert to collect the quality knowledge. They are as follows.
 1. Gather maximum possible data about the problem domain from books, manuals, internet, etc., in order to become familiar with specialist terminology and jargons of the problem domain.
 2. Identify the types of reasoning and problem solving tasks that the system will be required to perform.
 3. Find domain expert or team of experts who are willing to work on the project. Sometimes experts are frightened of being replaced by a computer system!
 4. Interview the domain experts multiple times during the course of building the system. Find out how they solve the problems that, the system is expected to solve. Have them check and refine the intermediate knowledge representation.



- Knowledge elicitation is a time consuming process. There exists automated knowledge elicitation and machine learning techniques which are increasingly being used as common modern alternatives.

6.9 Expert System vs. Traditional System

Sr. No.	Expert System (ES)	Traditional System (TS)
1.	Expert system Is knowledgebase + inference Engine	Traditional systems are Algorithms + data structures
2.	Expert systems can predict future events based on the current data input patterns using their inference process.	TS cannot do prediction tasks so efficiently as they do not have a strong inference engine.
3.	ES have very strong inference system to deduce knowledge from given facts	TS does not have a strong inference system to deduce knowledge.
4.	ES have explanation subsystem which can explain and justify the results at any intermediate stage.	TS do not have any mechanism to justify the results. Manual debugging is required to be done
5.	ES can do tasks like planning, scheduling, prediction, diagnosis; which require to deal with current data input and knowledge from past experiences, which generally handled by human experts.	TS cannot do expert tasks without human intervention.
6.	Expert systems are able to match human expertise in a particular domain provided with a complete knowledgebase and a powerful inference engine.	TS systems can only provide data based on available data, It cannot provide user with knowledge about the domain. Hence human expertise are required to analyse the data further to deduce knowledge from it. Hence TS cannot eliminate Human experts it can only assist them.

6.10 The Applications of Expert Systems

- The applications of expert systems find their way into most areas of knowledge work. Expert systems are used in the place of human interactions to industrial and commercial problems.
- There exists a wide spectrum of applications for expert systems. These applications are so big in number that, they cannot be categorized easily. All these applications are clustered into seven major classes.

1. Diagnosis and troubleshooting of devices and systems of all kinds

This category of applications comprises of expert systems that detect faults and suggest corrective actions for a malfunctioning device or process. Medical diagnosis was one of the first knowledge areas to which expert system technology was applied.

2. Planning and scheduling

These expert systems analyze set of goals and provides with the plan of action in order to achieve the predetermined goal. These applications has great commercial potential. For example, airline flight scheduling, manufacturing job-shop scheduling; and manufacturing process planning.

**3. Configuration of manufactured objects from subassemblies**

Configuration is historically one of the most important of expert system applications. In this category of applications a solution to a problem is synthesized from a given set of elements related by a set of constraints. These applications are used in many different industries. Examples include modular home building, manufacturing, complex engineering design and manufacturing.

4. Financial decision making

Expert systems are used vigorously in financial services, typically in foreign exchange trading. Advisory programs are widely used to assist bankers to take decisions about loans. Insurance companies used expert systems to assess the risk associated with the customer in order to determine price for the insurance. A typical application in the financial markets is

5. Knowledge publishing

Usage of expert systems in this area is relatively new, but has good potential for further exploration. There are many applications based on user information access preferences.

6. Process monitoring and control

In this category systems performing analysis of real time data are designed. These systems obtain data from physical devices and produce results specifying anomalies, predicting trends, etc. We have existing expert systems to monitor the manufacturing processes in the steel making and oil refining industries.

7. Design and manufacturing

These expert systems assist in the design of physical devices and processes. It is ranging from high level conceptual design of abstract entities all the way to factory floor configuration of manufacturing processes.

Review Questions

- Q. 1 Explain the need of hybrid approach.
- Q. 2 Explain Fuzzy neural system.
- Q. 3 Explain the architecture of ANFIS with the help of block diagram.
- Q. 4 What is expert system? Give examples of real time expert systems.
- Q. 5 What are the characteristics of expert system?
- Q. 6 Explain in building blocks of expert system.
- Q. 7 What are the various methods to represent domain knowledge in case of expert system?
- Q. 8 Write a short note on expert system shell.
- Q. 9 How reasoning is performed in case of expert systems?
- Q. 10 What is knowledge acquisition? What are the steps of knowledge acquisition?
- Q. 11 Write a short note on knowledge elicitation.



Note