

Soft Computing

Fundamentals and Applications

Dilip K. Pratihara



Alpha Science

Soft Computing

Fundamentals and Applications

Soft Computing

Fundamentals and Applications

Dilip K. Pratihari



Alpha Science International Ltd.

Oxford, U.K.

Soft Computing

Fundamentals and Applications

296 pgs. | 135 figs. | 29 tbls.

Dilip K. Pratihar

Department of Mechanical Engineering

Indian Institute of Technology

Kharagpur

Copyright © 2014

ALPHA SCIENCE INTERNATIONAL LTD.

7200 The Quorum, Oxford Business Park North

Garsington Road, Oxford OX4 2JZ, U.K.

www.alphasci.com

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the publisher.

Printed from the camera-ready copy provided by the Author.

ISBN 978-1-84265-863-5

E-ISBN 978-1-78332-085-1

Printed in India

Dedicated to

my wife **Arpita**
and son **Bitan**

Preface

This assignment of writing a textbook on **SOFT COMPUTING: Fundamentals and Applications** can be posed as a highly complex, non-linear, dynamic and constrained optimization problem. To solve the said problem, I started learning and collecting information in 1995, although the above optimization problem did not come to my mind, at that time. Prof. Amitabha Ghosh of IIT Kanpur, India, is the key person behind this transformation of a Metal Cutting and Machine Tools man into a Soft Computing modeler. I heard the word: **Genetic Algorithms** from Prof. Ghosh first, who then introduced me to Prof. Kalyanmoy Deb of IIT Kanpur. Prof. Deb took the responsibility of teaching me the fundamentals of Genetic Algorithms, which helped me a lot to swim across this deep sea of Soft Computing. Thus, the initial momentum required for propelling against this rough sea was provided by them. I am profoundly grateful to Prof. Ghosh and Prof. Deb, my Ph.D. supervisors. In 2000, Prof. Hideyuki Takagi of Kyushu Institute of Design, Fukuoka, Japan, gave me a shelter in his Laboratory for six months, where I learnt a lot of **Fuzzy Logic Techniques** (although I started learning the fundamentals of this field of research at IIT Kanpur) and **Neural Networks**. In 2001, I received the Alexander von Humboldt Fellowship, Germany and Prof. Wolfgang Bibel of Darmstadt University of Technology, Darmstadt, included me in his big research group, which gave me enough opportunities to digest the food I could eat till that time and assimilate them to strengthen my knowledge of Soft Computing. I express my deep sense of gratitude towards Prof. Takagi and Prof. Bibel.

I extend my profound thanks to my Ph.D. students (namely Dr. J.P. Ganjigatti, Dr. Subhagata Chattopadhyay, Dr. Nirmal Baran Hui, Dr. B.P. Mahesh, Dr. V. Pandu Ranga, Dr. Shibendu Shekhar Ray, Dr. Vidyut Dey, Dr. Suman Ghosh, Dr. Rega Rajendra, Dr. Somak Datta, Mr. Kuntal Maji, Mr. Abhishek R. Pal, Mr. T. Ganguly, Mr. S. Chanda, Mr. Rajesh Ghosh, Mr. M.N. Jha, Mr. N.K. Samal, Mr. Abhijit Mahapatra, Mr. P. Kalyan Chakravarthy); all M. Tech. project students (such as Mr. Kishan Choudhuri, Dr. Arup Kumar Nandi, Mr. Balvinder Singh, Mr. Venkata Subba Rao Amara, Mr. Onkar Pradeep Rao Bhise, Mr. Asfak Ali Mollah, Mr. Deepanshu, and others); all B. Tech. project students of IIT Kharagpur and NIT Durgapur (namely Mr. Parikshit Datta, Mr. Rahul Kumar Jha, Mr. Tushar Sinha, and others), who showed their willingness and courage to work with me. The discussions with them were always fruitful, which helped to reinforce my knowledge of Soft Computing.

I am grateful to the Head of the Department of Mechanical Engineering, IIT Kharagpur, India, who encouraged me to write this book. I am thankful to all my present colleagues of Mechanical Engineering Department at IIT Kharagpur and previous colleagues of NIT Durgapur, India, for their help and cooperation.

A large number of my teachers, friends, ex-colleagues and present colleagues of other Departments or Institutes, namely Prof. S.C. De Sarkar, Prof. P.P. Chakraborty, Prof. G.D. Ray, Prof. A.K. Ray, Prof. O.P. Shah, Prof. S.K. Barai, Prof. Sanat Kumar Roy, Prof. N.C. Chakraborty, Prof. M.K. Tiwari, Prof. D. Samanta, Prof. S. Sural of IIT Kharagpur; Prof. A. K. Mallik, Prof. K. Muralidhar, Prof. V.K. Jain, Prof. P.K. Kalra of IIT Kanpur; Prof. S.K. Maiti of IIT Bombay; Late Prof. A.K. Sen, Prof. D.K. Pal of NIT Durgapur; Prof. S.R. Deb, Ex-Prof. of Jadavpur University; Prof. S.K. Pal, Prof. A. Ghosh of ISI, Kolkata; Prof. A. Deshpande of University of Pune; Prof. G.C. Nandi of IIIT Allahabad, and many others, were the sources of my inspiration. I remember with gratitude the help and cooperation of my International friends working in this field of research, namely Prof. L.C. Jain of University of South Australia; Prof. R. Langari of Texas A&M University, USA; Prof. F. Klawonn of University of Wolfenbuettel, Germany; Prof. R. Kruse and Prof. habil Frank Palis of University of Magdeburg, Germany; Prof. G. Chakraborty of Iwate Prefectural University, Japan; Prof. G. Parker of Connecticut College, USA; Prof. A. Abraham of Chung-Ang University, Korea; Prof. M. Ali of Texas State University, USA; Prof. S.B. Cho of Yonsei University, Korea; Prof. D. Mitra of Florida Institute of Technology, USA, and others.

The financial help of the Continuing Education Centre, IIT Kharagpur, for preparing the manuscript of this book, is gratefully acknowledged.

I am thankful to Mr. N.K. Mehra of Narosa Publishing House, New Delhi, who took the decision of publishing this book. Special thanks are due to the people of production department of Narosa Publishing House for their assistance and cooperation.

I dedicate this book to my wife **Arpita** and son **Bitan**. Arpita took responsibility on her shoulder to make me free and gave me a constant encouragement. I get shelter under her big umbrella during both the rainy and sunny days. This project would not have been a possibility without her help and cooperation. Our son, Bitan is also neglected during this period of writing this book. I could not spend enough time with him and we used to miss each other. I express my deepest gratitude to my parents, parents-in-laws and other family members for their invaluable love, affection and support. Last but not the least, I am indebted to all of them, who directly and indirectly helped me for the successful completion of my book on Soft Computing: Fundamentals and Applications.

The contents of this book have been designed based on the experience I gained while teaching two subjects, namely **Evolutionary Computing** (renamed as **Soft Computing** later on) and **Knowledge-based Systems in Engineering** at the UG and PG levels, respectively, of IIT Kharagpur.

This book contains fourteen chapters. The first chapter introduces various terms, namely hard computing, soft computing and hybrid computing with some suitable examples. The second chapter gives introduction to optimization and explains a few traditional

methods of optimization, such as exhaustive search, random walk method, steepest descent method. An introduction is given to Genetic Algorithm (GA), one of the most popular non-traditional tools of optimization, in Chapter 3 and the principle of a binary-coded GA has been explained in detail. Chapter 4 includes discussion on some of the specialized genetic algorithms, like real-coded GA, visualized interactive GA, micro-GA and scheduling GA. Chapter 5 presents an overview of other non-traditional optimization tools like Particle Swarm Optimization (PSO) algorithm, Simulated Annealing (SA), and others. The concept of multi-objective optimization has been introduced and a few related algorithms have been explained in Chapter 6. Chapter 7 is devoted to the elements of classical and fuzzy sets. Some of the approaches of fuzzy logic controller, such as Mamdani Approach, Takagi and Sugeno's Approach are explained in Chapter 8. Moreover, the principles of fuzzy clustering algorithms have been discussed in this chapter. Chapter 9 concentrates on the fundamentals of Neural Networks (NNs). Some of the popular networks, such as Feed-Forward NN (FFNN), Recurrent NN (RNN), Self-Organizing Map (SOM), Counter-Propagation NN (CPNN), Radial Basis Function Network (RBFN) are explained in Chapter 10. Realizing the fact that each tool has its inherent advantages and disadvantages, combined techniques are developed to get advantages of the constituent tools and remove their disadvantages. Chapter 11 concentrates on the combined Genetic Algorithm-Fuzzy Logic (GA-FL) approaches. Each approach has been explained with a suitable example. Similarly, the combined GA-NN and NN-FL techniques are discussed with the help of some appropriate examples in Chapters 12 and 13, respectively. Chapter 14 deals with the applications of soft computing.

Dilip K. Pratihari

Contents

Dedication	iv
Preface	vii
Nomenclature	xvii
Greek Symbols	xix
Abbreviations	xxi
1 Introduction	1
1.1 Hard Computing	1
1.1.1 Features of Hard Computing	1
1.1.2 Examples of Hard Computing	2
1.2 Soft Computing	2
1.2.1 Features of Soft Computing	3
1.2.2 Examples of Soft Computing	3
1.3 Hybrid Computing	4
1.3.1 Examples of Hybrid Computing	5
1.4 Summary	5
1.5 Exercise	6
2 Optimization and Some Traditional Methods	7
2.1 Introduction to Optimization	7
2.1.1 A Practical Example	9
2.1.2 Classification of Optimization Problems	10
2.1.3 Principle of Optimization	12
2.1.4 Duality Principle	13
2.2 Traditional Methods of Optimization	14
2.2.1 Exhaustive Search Method	16
2.2.2 Random Walk Method	21
2.2.3 Steepest Descent Method	23
2.2.4 Drawbacks of Traditional Optimization Methods	27
2.3 Summary	28
2.4 Exercise	28
3 Introduction to Genetic Algorithms	31
3.1 Working Cycle of a Genetic Algorithm	31
3.2 Binary-Coded GA	33

3.2.1	Crossover or Mutation ?	42
3.2.2	A Hand Calculation	42
3.2.3	Fundamental Theorem of GA/Schema Theorem	44
3.2.4	Limitations of a Binary-Coded GA	46
3.3	GA-parameters Setting	46
3.4	Constraints Handling in GA	48
3.4.1	Penalty Function Approach	49
3.5	Advantages and Disadvantages of Genetic Algorithms	51
3.6	Combination of Local and Global Optimum Search Algorithms	52
3.7	Summary	52
3.8	Exercise	53
4	Some Specialized Genetic Algorithms	59
4.1	Real-Coded GA	59
4.1.1	Crossover Operators	59
4.1.2	Mutation Operators	63
4.2	Micro-GA	65
4.3	Visualized Interactive GA	65
4.3.1	Mapping Methods	66
4.3.2	Simulation Results	69
4.3.3	Working Principle of the VIGA	71
4.4	Scheduling GA	72
4.4.1	Edge Recombination	73
4.4.2	Order Crossover #1	75
4.4.3	Order Crossover #2	76
4.4.4	Cycle Crossover	76
4.4.5	Position-Based Crossover	77
4.4.6	Partially Mapped Crossover (PMX)	78
4.5	Summary	80
4.6	Exercise	80
5	Overview of Other Non-Traditional Optimization Methods	83
5.1	Simulated Annealing	84
5.1.1	Working Principle	84
5.2	Particle Swarm Optimization	87
5.2.1	Comparisons Between PSO and GA	89
5.3	Summary	89
5.4	Exercise	89
6	Multi-Objective Optimization	91
6.1	Introduction	91
6.2	Some Approaches to Solve Multi-Objective Optimization Problems	92
6.2.1	Weighted Sum Approach	92
6.2.2	Goal/Target Programming	93
6.2.3	Vector Evaluated Genetic Algorithm (VEGA)	93
6.2.4	Distance-based Pareto-GA (DPGA)	94
6.2.5	Non-dominated Sorting Genetic Algorithms (NSGA)	95
6.3	Summary	98
6.4	Exercise	98
7	Introduction to Fuzzy Sets	101
7.1	Crisp Sets	101

7.1.1	Notations Used in Set Theory	102
7.1.2	Crisp Set Operations	103
7.1.3	Properties of Crisp Sets	104
7.2	Fuzzy Sets	106
7.2.1	Representation of a Fuzzy Set	107
7.2.2	Difference Between Crisp Set and Fuzzy Set	111
7.2.3	A Few Definitions in Fuzzy Sets	112
7.2.4	Some Standard Operations in Fuzzy Sets and Relations	115
7.2.5	Properties of Fuzzy Sets	121
7.3	Measures of Fuzziness and Inaccuracy of Fuzzy Sets	122
7.4	Summary	123
7.5	Exercise	123
8	Fuzzy Reasoning and Clustering	125
8.1	Introduction	125
8.2	Fuzzy Logic Controller	125
8.2.1	Two Major Forms of Fuzzy Logic Controller	126
8.2.2	Hierarchical Fuzzy Logic Controller	139
8.2.3	Sensitivity Analysis	141
8.2.4	Advantages and Disadvantages of Fuzzy Logic Controller	141
8.3	Fuzzy Clustering	142
8.3.1	Fuzzy C-Means Clustering	142
8.3.2	Entropy-based Fuzzy Clustering	147
8.4	Summary	152
8.5	Exercise	152
9	Fundamentals of Neural Networks	157
9.1	Introduction	157
9.1.1	Biological Neuron	157
9.1.2	Artificial Neuron	158
9.1.3	A Layer of Neurons	161
9.1.4	Multiple Layers of Neurons	162
9.2	Static vs. Dynamic Neural Networks	163
9.3	Training of Neural Networks	163
9.3.1	Supervised Learning	164
9.3.2	Un-supervised Learning	164
9.3.3	Incremental Training	164
9.3.4	Batch Mode of Training	164
9.4	Summary	165
9.5	Exercise	165
10	Some Examples of Neural Networks	167
10.1	Introduction	167
10.2	Multi-Layer Feed-Forward Neural Network (MLFFNN)	167
10.2.1	Forward Calculation	169
10.2.2	Training of Network Using Back-Propagation Algorithm	170
10.2.3	Steps to be Followed to Design a Suitable NN	175
10.2.4	Advantages and Disadvantages	175
10.2.5	A Numerical Example	175
10.3	Radial Basis Function Network (RBFN)	178
10.3.1	Forward Calculations	179
10.3.2	Tuning of RBFN Using Back-Propagation Algorithm	182

10.4	Self-Organizing Map (SOM)	185
10.4.1	Competition	186
10.4.2	Cooperation	186
10.4.3	Updating	187
10.4.4	Final Mapping	187
10.4.5	Simulation Results	188
10.5	Counter-Propagation Neural Network (CPNN)	188
10.5.1	Full CPNN	189
10.5.2	A Numerical Example	191
10.5.3	Forward-Only CPNN	193
10.6	Recurrent Neural Networks (RNNs)	194
10.6.1	Elman Network	194
10.6.2	Jordan Network	195
10.6.3	Combined Elman and Jordan Network	195
10.7	Summary	196
10.8	Exercise	196
11	Combined Genetic Algorithms: Fuzzy Logic	199
11.1	Introduction	199
11.2	Fuzzy-Genetic Algorithm	199
11.3	Genetic-Fuzzy System	202
11.3.1	A Brief Literature Review	202
11.3.2	Working Principle of Genetic-Fuzzy Systems	205
11.4	Summary	212
11.5	Exercise	212
12	Combined Genetic Algorithms: Neural Networks	215
12.1	Introduction	215
12.2	Working Principle of a Genetic-Neural System	217
12.2.1	Forward Calculation	218
12.2.2	A Hand Calculation	221
12.3	Summary	223
12.4	Exercise	223
13	Combined Neural Networks: Fuzzy Logic	225
13.1	Introduction	225
13.2	Neuro-Fuzzy System Working Based on Mamdani Approach	226
13.2.1	Tuning of the Neuro-Fuzzy System Using a Back-Propagation Algorithm	231
13.2.2	Tuning of the Neuro-Fuzzy System Using a Genetic Algorithm	232
13.2.3	A Numerical Example	233
13.3	Neuro-Fuzzy System Based on Takagi and Sugeno's Approach	238
13.3.1	Tuning of the ANFIS Using a Genetic Algorithm	241
13.3.2	A Numerical Example	242
13.4	Summary	246
13.5	Exercise	246
14	Applications of Soft Computing	249
14.1	Introduction	249
14.2	Applications of Soft Computing in Design and Development of Intelligent Autonomous Robots	249
14.2.1	Information Collection of the Environment	250
14.2.2	Motion Planning of Robots in Dynamic Environment	250

14.2.3	Determination of Appropriate Control Signals for the Motors	251
14.2.4	Learning from Previous Experience	251
14.3	Applications of Soft Computing in Data Analysis	252
14.3.1	Classification Tools	252
14.3.2	Predictive Models	252
14.4	Summary	254
14.5	Exercise	254

Nomenclature

$A(x)$	Fuzzy set
\overline{A}	Absolute complement of a set A
A^c	Absolute complement of a set A
$A^p(x)$	p -th power of a fuzzy set $A(x)$
$ A(x) $	Scalar cardinality of a fuzzy set A
A_i	Area corresponding to i -th fired rule
$A - B$	Difference between two sets A and B
$A \cap B$	Intersection of two sets A and B
$A \cup B$	Union of two sets A and B
$A \circ B$	Composition of two fuzzy relations A, B
$A \subset B$	A is a proper subset of B
$A \supset B$	A is a proper superset of B
$A(\alpha_j)$	Firing area of j -th rule of the FLC
Ch	Child solution
d_{ij}	Euclidean distance between the points i and j
\overline{d}	Mean distance
d	Depth of cut, mm
D	Decoded value of a binary sub-string
E	Entropy
f	Fitness of a GA-string
\overline{f}	Average fitness
g	Level of cluster fuzziness
G_{max}	Maximum number of generations
$h(A)$	Height of a fuzzy set A
$H(A)$	Entropy of a fuzzy set A
I	Input
$I(A; B)$	Inaccuracy of fuzzy set B w.r.t. fuzzy set A
$[I_{ex}]$	External inputs
$[I_{in}]$	Internal inputs
K_P	Gain value of Proportional controller
K_I	Gain value of Integral controller
K_D	Gain value of Derivative controller
l	Length of a sub-string
L	Length of a GA-string

m	Mean
N	GA-population size
O	Output
$O(H)$	Order of a schema H
p_c	Probability of crossover
p_m	Probability of mutation
p_s	Crossover survival probability
P	Load
P_i	Penalty used for i -th infeasible solution
Pr	Parent solution
q	Exponent of the polynomial function
r	Random number
R	Rank
R_{Ij}	Input of j -th neuron lying on R -th layer
R_{Oj}	Output of j -th neuron lying on R -th layer
s	Sensitivity of the controller
S	Surface roughness, <i>micro</i> – m
S_i	Search direction at i -th iteration
S_{ij}	Similarity between two data points i and j
$Sh(d_{ij})$	Sharing between two points i and j
t	Feed rate, mm/rev
T	Torque
$[T]$	Data set
u_i	Unit random vector
$U'_{f'}$	Crisp output of the controller
v	Cutting speed, m/min
$[V]$	Connecting weights between the input and hidden layers of neural network
w	Firing strength of a rule in Takagi and Sugeno's approach
\bar{w}	Normalized firing strength of a rule in Takagi and Sugeno's approach
$[W]$	Connecting weights between the hidden and output layers of neural network
X	Universal set
X_i	Design/decision variables at i -th iteration
y^i	Output of i -th rule

Greek Symbols

α	Spread factor
α'	momentum constant
β	Threshold value of similarity
$\delta(H)$	Defining length of a schema
$\bar{\delta}$	Perturbation factor
η	Learning rate
ϵ	Termination criterion
γ	Threshold used for declaring a valid cluster
λ	Step length
μ	Membership value
ϕ	Penalty term
ρ	Density
σ	Standard deviation
τ	Shear stress
∇	Gradient
Δ	Maximum value of perturbation

Abbreviations

<i>A</i>	Ahead
<i>ABC</i>	Artificial Bee Colony
<i>ACO</i>	Ant Colony Optimization
<i>AIS</i>	Artificial Immune System
<i>AL</i>	Ahead Left
<i>ANFIS</i>	Adaptive Neuro-Fuzzy Inference System
<i>ANN</i>	Artificial Neural Network
<i>AR</i>	Almost Red
<i>ART</i>	Ahead Right
<i>AVA</i>	Average Variance of the Alleles
<i>BP</i>	Back-Propagation
<i>BPNN</i>	Back-Propagation Neural Network
<i>CA</i>	Cultural Algorithm
<i>CLARANS</i>	Clustering Large Applications based on RANdomized Search
<i>CPNN</i>	Counter-Propagation Neural Network
<i>CSS</i>	Charged System Search
<i>DB</i>	Data Base
<i>DBSCAN</i>	Density-Based Spatial Clustering of Applications with Noise
<i>DE</i>	Differential Evolution
<i>DPGA</i>	Distance-based Pareto Genetic Algorithm
<i>DV</i>	Decoded Value
<i>EP</i>	Evolutionary Programming
<i>ES</i>	Evolution Strategies
<i>F</i>	Fast
<i>FCM</i>	Fuzzy C-Means
<i>FEM</i>	Finite Element Method
<i>FL</i>	Fuzzy Logic
<i>FLC</i>	Fuzzy Logic Controller
<i>FLS</i>	Fuzzy Logic System
<i>FFNN</i>	Feed-Forward Neural Network
<i>FNN</i>	Fuzzy Neural Network
<i>FR</i>	Far
<i>FGA</i>	Fuzzy-Genetic Algorithm
<i>GA</i>	Genetic Algorithm
<i>GDM</i>	Genotypic Diversity Measure
<i>GFS</i>	Genetic-Fuzzy System

<i>GNS</i>	Genetic-Neural System
<i>GP</i>	Genetic Programming
<i>H</i>	High
<i>ICA</i>	Imperialist Competitive Algorithm
<i>KB</i>	Knowledge Base
<i>LW</i>	Low
<i>LR</i>	Large
<i>LT</i>	Left
<i>M</i>	Medium
<i>MAS</i>	Multi-Agent System
<i>MF</i>	Magic Factor
<i>MNN</i>	Modular Neural Network
<i>MOGUL</i>	Methodology to Obtain GFRBSs Under the IRL approach
<i>MPI</i>	Message Passing Interface
<i>MSD</i>	Mean Square Deviation
<i>MSE</i>	Mean Square Error
<i>NPGA</i>	Niched Pareto Genetic Algorithm
<i>NR</i>	Near
<i>NSGA</i>	Non-dominated Sorting Genetic Algorithm
<i>NFS</i>	Neuro-Fuzzy System
<i>NLM</i>	Non-Linear Mapping
<i>NN</i>	Neural Network
<i>NRD</i>	Not Red
<i>PAES</i>	Pareto-Archived Evolution Strategy
<i>PBGA</i>	Pseudo-Bacterial Genetic Algorithm
<i>PCA</i>	Principal Component Analysis
<i>PDM</i>	Phenotypic Diversity Measures
<i>PID</i>	Proportional Integral Derivative
<i>PMX</i>	Partially Mapped Crossover
<i>PR</i>	Perfectly Red
<i>PSO</i>	Particle Swarm Optimization
<i>RB</i>	Rule Base
<i>RBF</i>	Radial Basis Function
<i>RBFN</i>	Radial Basis Function Network
<i>RNN</i>	Recurrent Neural Network
<i>RT</i>	Right

Chapter 1

Introduction

Before we introduce an emerging field, namely **soft computing**, let us examine the meaning of the term: **hard computing**. This chapter defines this term. Another term called **hybrid computing** has also been introduced in this chapter.

1.1 Hard Computing

The term: **hard computing** was first coined by Prof. L.A. Zadeh of the University of California, USA, in 1996 [1], although it had been used to solve different problems for a long time.

Let us see the steps to be followed to solve an engineering problem, which are:

- The variables related to an engineering problem are identified first and then classified into two groups, namely input or condition variables (also known as **antecedents**) and output or action variables (also called **consequents**).
- The input-output relationships are expressed in terms of mathematical (say differential) equations.
- Differential equations are then solved analytically or using numerical methods (if required).
- Control action is decided based on the solutions of these mathematical equations.

The procedure stated above is nothing but the principle of hard computing.

1.1.1 Features of Hard Computing

Human beings have their natural quest for precision and as a result of which, they try to model a problem using the principle of mathematics. Thus, hard computing could establish itself long back, as a conventional method for solving engineering problems. It has the following features:

- As it works based on the principle of mathematics, it may yield precise solutions. Thus, control actions will be accurate.
- It may be suitable for the problems, which are easy to model mathematically and whose stability is highly predictable.

1.1.2 Examples of Hard Computing

Hard computing has been used to solve a variety of problems, two such problems are stated below.

1. Stress analysis of a mechanical member (say beam) subjected to some forms of loading pattern
If the beam is assumed to have constant cross-section along its length, the problem of stress analysis can be easily formulated mathematically. On the other hand, if the beam is assumed to have varying cross-sections along its length, the principle of Finite Element Method (FEM) can be used to determine the developed stress.
2. Determination of gain values of a Proportional Integral Derivative (PID) controller to be used to control a process
Let us assume that a PID controller is to be used to control a motor mounted at a robotic joint. A set of gain values (that is, K_P , K_I and K_D) can be obtained mathematically for this controller, which are generally kept constant.

It is important to note that hard computing is applicable, if and only if the problem can be formulated mathematically. Unfortunately, most of the real-world problems are so complex and ill-defined that they cannot be modelled mathematically or the mathematical modelling becomes highly non-linear. Moreover, some sort of imprecisions and uncertainties are inherent to these problems. Thus, it may not always be possible to solve the complex real-world problems using the principle of hard computing.

1.2 Soft Computing

The term: **soft computing** was also introduced by Prof. Zadeh, in 1992 [2]. It is a collection of some biologically-inspired methodologies, such as Fuzzy Logic (FL), Neural Network (NN), Genetic Algorithm (GA) and others, and their different combined forms, namely GA-FL, GA-NN, NN-FL, GA-FL-NN, in which precision is traded for tractability, robustness, ease of implementation and a low cost solution. Fig. 1.1 shows a schematic view indicating different members of soft computing family and their possible interactions. Control algorithms developed based on soft computing may be computationally tractable, robust and adaptive in nature.

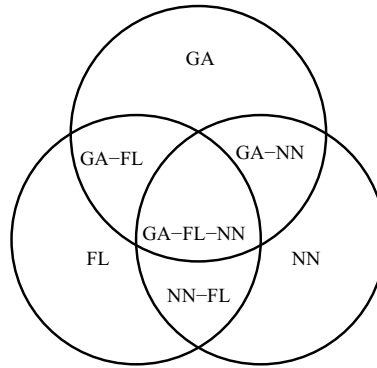


Figure 1.1: A schematic view showing different members of soft computing and their possible interactions.

1.2.1 Features of Soft Computing

Soft computing is an emerging field, which has the following features:

- It does not require an extensive mathematical formulation of the problem.
- It may not be able to yield so much precise solution as that obtained by the hard computing technique.
- Different members of this family are able to perform various types of tasks. For example, Fuzzy Logic (FL) is a powerful tool for dealing with imprecision and uncertainty, Neural Network (NN) is a potential tool for learning and adaptation and Genetic Algorithm (GA) is an important tool for search and optimization. Each of these tools has its inherent merits and demerits (which are discussed in detail, in the subsequent chapters). In combined techniques (such as GA-FL, GA-NN, NN-FL, GA-FL-NN), either two or three constituent tools are coupled to get the advantages from both of them and remove their inherent limitations. Thus, in soft computing, the functions of the constituent members are complementary in nature and there is no competition among themselves.
- Algorithm developed based on soft computing is generally found to be adaptive in nature. Thus, it can accommodate to the changes of a dynamic environment.

1.2.2 Examples of Soft Computing

Soft computing is becoming more and more popular, nowadays. It has been used by various researchers to solve a variety of problems, two of them are discussed below.

1. Design and development of adaptive motion planners for intelligent and autonomous robots

An intelligent and autonomous robot should be able to plan its collision-free, time-optimal paths, while navigating in the presence of some moving obstacles in a dynamic

environment. Both FL- and NN-based adaptive motion planners have been designed and developed for the intelligent and autonomous robots [3].

2. Soft computing-based expert systems to carry out input-output modeling of engineering systems or processes

To automate any engineering process or system, it may be required to know its input-output relationships in both forward and reverse directions. Forward mapping problems (where the responses are expressed as the functions of input process parameters) can be solved mathematically. However, it may not be always possible to solve the problem of reverse mapping using the obtained mathematical equations. The principle of soft computing has been successfully used to solve the problems of both forward and reverse mappings for a number of engineering systems or processes [4, 5].

Most of the real-world problems are too complex to model mathematically. In such cases, hard computing will fail to provide any solution, and we will have to switch over to soft computing, in which precision is considered to be secondary and we are primarily interested in acceptable solutions.

1.3 Hybrid Computing

Hybrid computing is a combination of the conventional hard computing and emerging soft computing. Fig. 1.2 shows the schematic view of a hybrid computing scheme. Both

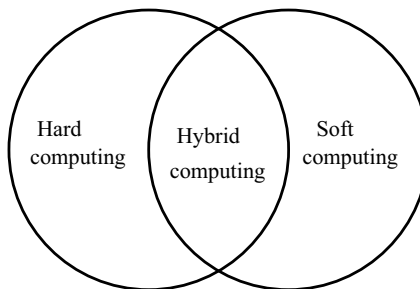


Figure 1.2: A schematic view showing the scheme of hybrid computing.

hard computing as well as soft computing have their inherent advantages and disadvantages. To get the advantages of both these techniques and eliminate their individual limitations, one may switch over to hybrid computing to solve a problem more efficiently. Thus, a part of the problem will be solved using hard computing and the remaining part can be tackled utilizing soft computing, if it so demands. Moreover, these two techniques may be complementary to each other, while solving some complex real-world problems. However, the users of hard computing often do not like soft computing people, due to their inertia and they fight among themselves.

1.3.1 Examples of Hybrid Computing

Nowadays, hybrid computing has been utilized by various investigators to solve different engineering problems in a more efficient way. Ovaska et al. [6] provides a survey on various applications of hybrid computing. Some of the possible applications of hybrid computing are:

1. Optimal design of machine elements using Finite Element Method (FEM) and soft computing

Several attempts have been made to design different machine elements using the FEM. However, the quality of these solutions depends on the selection of the elements, their size and connectivity. Moreover, material properties of these members may not remain unchanged during their applications. Thus, there exists fuzziness in both the FEM analysis as well as material properties. This fuzziness can be modelled using soft computing, before the FEM analysis is finally carried out to obtain the optimal design of machine elements [7].

2. PID controller trained by soft computing

A PID controller is one of the most widely-used controllers, nowadays. Its gain values (K_P , K_I and K_D) are determined mathematically based on a fixed condition of the environment. Thus, if there is a sudden change in the environment, the conventional PID controller may not be able to yield an optimal control action. To overcome this difficulty, the gain values can be tuned using the principle of soft computing. In a dynamic environment, appropriate gain values of the PID controller can be determined on-line, using a previously tuned FL- or NN-based expert system [8].

It is important to mention that the selection of a technique depends on the problem to be solved. After gathering information of the problem, we generally first try to solve it using hard computing. However, if it fails for some reasons, we may switch over to soft computing. Sometimes, we should take the help of hybrid computing, if the problem so demands.

This book deals with the fundamentals and applications of soft computing and a detailed discussion on hard computing is beyond its scope.

1.4 Summary

This chapter has been summarized as follows:

1. Hard computing works based on the principle of mathematics and it generally yields precise solutions. However, we can go for hard computing, if the problem is well-defined and easy to model mathematically.
2. As most of the real-world problems are complex in nature and difficult to model mathematically, hard computing may not be suitable to solve such problems. One

may switch over to soft computing to solve the above problems involving inherent imprecisions and uncertainties.

3. Soft computing is a family composed of different members, namely FL, NN, GA and their different combinations, in which precision is traded for tractability, robustness and a low cost solution. It can provide some feasible solutions to the complex real-world problems.
4. The concept of hybrid computing that combines the hard computing with soft computing, has also been emerged to get advantages from both of them and eliminate their individual limitations.
5. Selection of a suitable computing scheme for solving a problem depends on its nature.

1.5 Exercise

1. Define briefly the terms: soft computing, hard computing and hybrid computing with some suitable examples.
2. Justify the use of the term: *soft* in soft computing.
3. How do you select a suitable scheme of computing (either hard computing or soft computing or hybrid computing) to solve a particular problem ?

Chapter 2

Optimization and Some Traditional Methods

This chapter introduces the concept of optimization and discusses some of its traditional methods (also known as conventional or classical methods). Traditional methods of optimization include both gradient-based and direct search techniques. A detailed discussion on all these traditional methods of optimization is beyond the scope of this book. An attempt has been made in this chapter to explain the working principles of a few methods only, namely **Exhaustive Search Method**, **Random Walk Method** and **Steepest Descent Method**.

2.1 Introduction to Optimization

Optimization is the process of finding the best one, out of all feasible solutions. To realize the need for an optimization, let us suppose that we are going to design a suitable gear-box to be used between a source of power and propeller shaft of a ship. We generally use the traditional principle of machine design (which includes determination of stress-strain relationship, checking of dynamic and wear loads, and others) for the above purpose. If we use this traditional principle only, there is a possibility that the designed gear-box will have no practical importance, at all. To overcome this difficulty, a designer should use an optimization tool along with the traditional principle of design. An optimization tool can make a design more efficient and cost effective.

The concept of optimization is defined mathematically as follows: Let us consider y to be the function of a single variable x , that is, $y = f(x)$. If the first derivative of this function, that is, $f'(x)$ becomes equal to zero, that is, $f'(x) = 0$, at a point $x = x^*$, we say that either the **optimum** (that is, **minimum** or **maximum**) or **inflection point** exists at that point. An inflection point (also known as a **saddle point**) is a point, that is neither a maximum nor a minimum one. To further investigate the nature of the point, we determine the first non-zero higher order derivative denoted by n . Two different cases may arise as stated below.

1. If n is found to be an odd number, x^* is an inflection point.
2. If n is seen to be an even number, x^* is a local optimum point. To investigate further for declaring it either a local minimum or a local maximum point, the following two conditions are checked:
 - If the value of the derivative is found to be positive, x^* is a local minimum point.
 - If the value of the derivative is seen to be negative, x^* is a local maximum point.

A Numerical Example:

Determine the minimum/maximum/inflection point of the function $f(x) = \frac{x^2}{2} + \frac{125}{x}$ for the positive values of x .

Solution:

The function is:

$$f(x) = \frac{x^2}{2} + \frac{125}{x}$$

Fig. 2.1 shows the plot of this function. Its first order derivative is given by $f'(x) = x - \frac{125}{x^2}$.

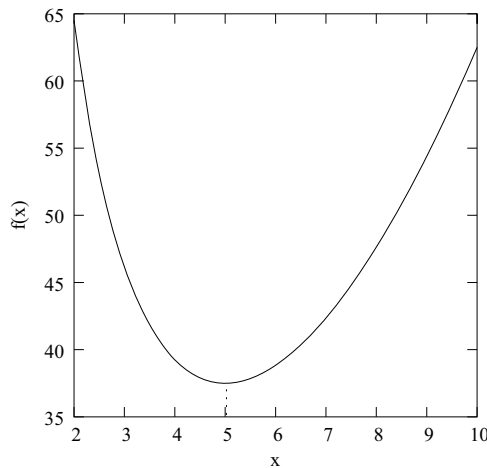


Figure 2.1: Plot of the function $f(x) = \frac{x^2}{2} + \frac{125}{x}$.

By considering $f'(x) = 0$ and solving it, we get $x = 5.0$.

Therefore, the minimum/maximum/inflection point exists at $x = 5.0$.

The second order derivative of this function is given by $f''(x) = 1 + \frac{250}{x^3}$.

At $x = 5.0$, $f''(x) = 3.0 \neq 0.0$.

Therefore, n is found to be equal to 2 and it indicates that the local optimum exists at $x = 5.0$.

As $f''(x)$ is seen to be equal to 3.0 (a positive number), it can be declared that the local minimum exists at $x = 5.0$.

The minimum value of $f(x)$ turns out to be equal to 37.5.

2.1.1 A Practical Example

Let us suppose that we will have to design an optimal wooden pointer generally used by a speaker while delivering a lecture. The pointer should be as light in weight as possible, after ensuring the conditions that there is no mechanical breakage and deflection of pointing end is negligible. This task can simply be thought of selecting the best one, out of all pointers contained in a bin.

This problem can be posed as an optimization problem as explained below. Fig. 2.2 shows the pointer in 2-D. The geometry is specified by its diameter at the gripping end d (the diameter of the pointing end has been assumed to be equal to zero) and length L' . Its density ρ has been assumed to be a constant. The geometrical parameters, namely d and L' are

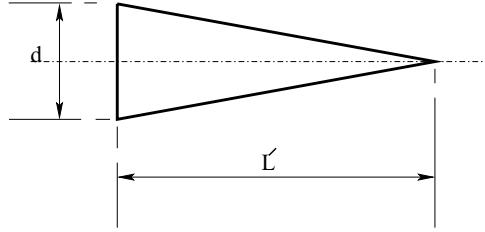


Figure 2.2: A wooden pointer.

known as **design** or **decision** variables and the fixed parameter ρ is called **pre-assigned** parameter. The design variables have some feasible bounds, which are known as **geometric** or **side** constraints. The aim of this optimization is to search a lighter design, for which its weight has been expressed with the help of an **objective function**. Moreover, the pointer should be able to satisfy a few constraints (in terms of its strength and deflection) during its usage, which are known as **functional** or **behavior** constraints.

The above optimization problem can be expressed mathematically as follows:

$$\text{Minimize Mass of the pointer } M = \frac{\pi d^2 L' \rho}{12} \quad (2.1)$$

subject to

$$\begin{aligned} \text{deflection } \delta &\leq \delta_{allowable}, \\ \text{strength of the stick } s &\geq s_{required} \end{aligned}$$

and

$$\begin{aligned} d^{min} &\leq d \leq d^{max}, \\ L'^{min} &\leq L' \leq L'^{max}. \end{aligned}$$

Here, equation (2.1) is called the objective function. The constraints related to deflection and strength of the stick are known as the functional constraints. The lower and upper limits of d and L' are expressed as the geometric or side constraints.

2.1.2 Classification of Optimization Problems

Optimization problems have been classified in a number of ways as discussed below.

1. Depending on the nature of equations involved in the objective function and constraints, they are divided into two groups, namely **linear** and **non-linear** optimization problems, which are defined below.

- **Linear optimization problem** – An optimization problem is called linear, if both the objective function as well as all the constraints are found to be linear functions of design variables.

Example:

$$\text{Maximize } y = f(x_1, x_2) = 2x_1 + x_2 \quad (2.2)$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 3, \\ 5x_1 + 2x_2 &\leq 10 \end{aligned}$$

and

$$x_1, x_2 \geq 0.$$

- **Non-linear optimization problem** – An optimization problem is known as non-linear, if either the objective function or any one of the functional constraints is non-linear function of design variables. Moreover, both the objective function as well as all functional constraints may be non-linear functions of design variables in a non-linear optimization problem.

Example:

$$\text{Maximize } y = f(x_1, x_2) = x_1^2 + x_2^3 \quad (2.3)$$

subject to

$$\begin{aligned} x_1^4 + x_2^2 &\leq 629, \\ x_1^3 + x_2^3 &\leq 133 \end{aligned}$$

and

$$x_1, x_2 \geq 0.$$

2. Based on the existence of any functional constraint, optimization problems are classified into two groups, such as **un-constrained** (without any functional constraint) and **constrained** (when at least one functional constraint is present) optimization problems, the examples of which are given below.

- **Un-constrained optimization problem**

$$\text{Minimize } y = f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2 \quad (2.4)$$

where

$$x_1, x_2 \geq 0.$$

- **Constrained optimization problem**

$$\text{Minimize } y = f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2 \quad (2.5)$$

subject to

$$g_i(x_1, x_2) \leq c_i, i = 1, 2, \dots, n$$

and

$$x_1, x_2 \geq 0.$$

3. Depending on the type of design variables, optimization problems are clustered into three groups, namely **integer programming**, **real-valued programming** and **mixed-integer programming** problems, which are briefly discussed below.

- **Integer programming problem** – An optimization problem is said to be an integer programming problem, if all the design variables take only integer values.

Example:

$$\text{Maximize } y = f(x_1, x_2) = 2x_1 + x_2 \quad (2.6)$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 3, \\ 5x_1 + 2x_2 &\leq 9 \end{aligned}$$

and

$$\begin{aligned} x_1, x_2 &\geq 0, \\ x_1, x_2 &\text{ are the integers.} \end{aligned}$$

- **Real-valued programming problem** – An optimization problem is known as a real-valued programming problem, if all the design variables are bound to take only real values.

Example:

$$\text{Maximize } y = f(x_1, x_2) = 2x_1 + x_2 \quad (2.7)$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 3.2, \\ 5x_1 + 2x_2 &\leq 10.0 \end{aligned}$$

and

$$\begin{aligned} x_1, x_2 &\geq 0.0, \\ x_1, x_2 &\text{ are the real variables.} \end{aligned}$$

- **Mixed-integer programming problem** – It is an optimization problem, in which some of the variables are integers and the remaining variables take real values.

Example:

$$\text{Maximize } y = f(x_1, x_2, x_3) = x_1 + 3x_2 + 4x_3 \quad (2.8)$$

subject to

$$\begin{aligned} 2x_1 + 5x_2 + x_3 &\leq 20.5, \\ x_1 + x_2 + x_3 &\leq 7.5, \\ 3x_1 + x_2 + 2x_3 &\leq 16.4 \end{aligned}$$

and

$$\begin{aligned} x_1, x_2, x_3 &\geq 0, \\ x_1, x_2 &\text{ are the integers and } x_3 \text{ is a real variable.} \end{aligned}$$

4. Depending on the nature of design variables, optimization problems are classified into two groups, such as **static** and **dynamic**, as discussed below.

- **Static optimization problem** – Let us consider a beam (either cantilever or simply-supported) of constant cross-section throughout its length, which is subjected to some forms of loading. Our aim is to determine its optimal cross-section to satisfy a few conditions. It is an example of static optimization problem, as its cross-section does not vary along its length.
- **Dynamic optimization problem** – In the above beam, if the cross-section varies along its length, it becomes a dynamic optimization problem.

2.1.3 Principle of Optimization

To explain the principle of optimization, let us consider a constrained optimization problem as given below.

$$\text{Minimize } y = f(x_1, x_2) = (x_1 - a)^2 + (x_2 - b)^2 \quad (2.9)$$

subject to

$$g_i(x_1, x_2) \leq c_i, i = 1, 2, \dots, n$$

and

$$\begin{aligned} x_1 &\geq x_1^{min}, \\ x_2 &\geq x_2^{min}, \end{aligned}$$

where a and b are the constants and n indicates the number of functional constraints. Fig. 2.3 shows the plots of the constraints. Depending on the nature of these constraints, a feasible zone has been identified. Any point lying in the feasible zone is a probable candidate

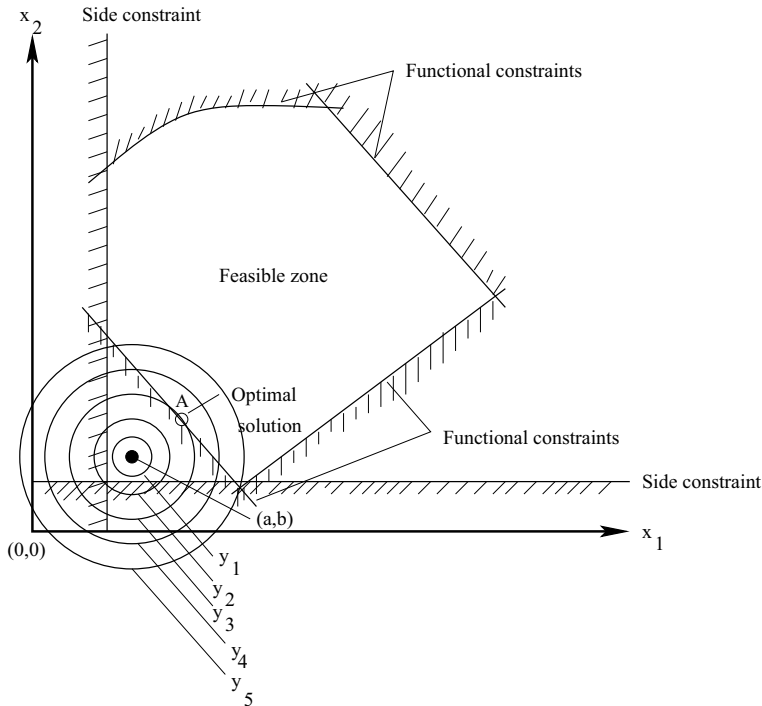


Figure 2.3: A schematic view explaining the principle of optimization.

for the optimal solution. The points residing inside the feasible zone are called **free points**, whereas the points lying on boundary of the feasible zone are known as **bound points**. Thus, an optimal solution can be either a free point or a bound point contained in the feasible zone. To determine the optimal solution, the following procedure is adopted. For different fixed values of y (say y_1, y_2, \dots), we draw contour plots of the objective function. The point (say A), at which one of the contour plots just touches the feasible zone, is declared as the optimal point and the corresponding optimized function value is determined. Thus, the optimal solution of the constrained optimization problem is denoted by A , whereas that of the corresponding un-constrained optimization problem is represented by the point (a, b) , the center of the contour plots.

2.1.4 Duality Principle

Let us consider a unimodal function of a single variable denoted by $y = f(x)$, which is to be minimized. The problem may be stated as follows:

$$\text{Minimize } y = f(x) \quad (2.10)$$

subject to

$$x \geq 0.0.$$

Fig. 2.4 shows the plot of the function. Let us also assume that the function reaches its minimum value, corresponding to a value of $x = x^*$. The above minimization problem can

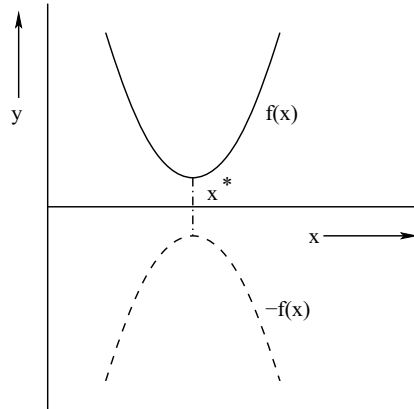


Figure 2.4: A function plot used to explain the duality principle of optimization.

be posed as a maximum problem as given below.

$$\textbf{Maximize} \quad -f(x) \quad (2.11)$$

subject to

$$x \geq 0.0.$$

It is important to mention that through this conversion of minimization problem into a maximization problem, the value of x at which the optimum occurs will remain the same as x^* . It is known as the duality principle of optimization.

2.2 Traditional Methods of Optimization

A huge literature is available on traditional methods of optimization. Various methods are in use (refer to Fig. 2.5) and the selection of an appropriate one is problem-dependent. For example, we use a group of methods for solving the linear programming problems, whereas for non-linear problems, we take the help of some other methods. In a non-linear optimization problem, the objective function may be composed of either a single variable or a number of variables. A single variable non-linear optimization problem can be solved using analytical, numerical methods. The analytical method works based on the principle of differential calculus. Numerical methods include both the elimination and interpolation methods. Multi-variable optimization problems may be either constrained or unconstrained in nature. Unconstrained optimization problems can be tackled utilizing either direct search or gradient-based methods. On the other hand, there are some direct and indirect methods to solve the constrained optimization problems.

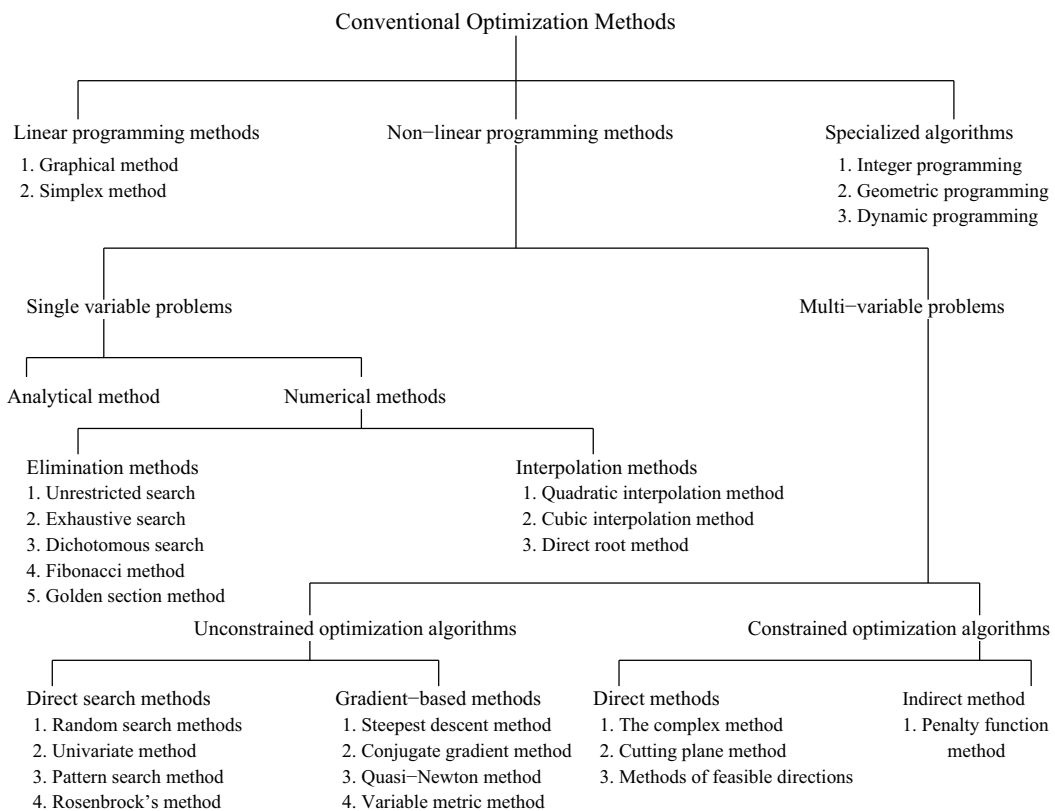


Figure 2.5: Classification of traditional methods of optimization.

A detailed discussion on all these traditional optimization methods is beyond the scope of this book. Interested readers may refer to the books [9, 10] for the above purpose. Although the scope is limited, a few traditional methods are explained below, in detail.

2.2.1 Exhaustive Search Method

Let us consider a unimodal function y (that is, it has only one peak) of a single variable x defined in the range of (x^{min}, x^{max}) , where x^{min} and x^{max} are the lower and upper limits of the variable x , respectively. Our aim is to maximize the function $y = f(x)$ in the above range. The problem may be mathematically stated as follows:

$$\text{Maximize } y = f(x) \quad (2.12)$$

subject to

$$x^{min} \leq x \leq x^{max}.$$

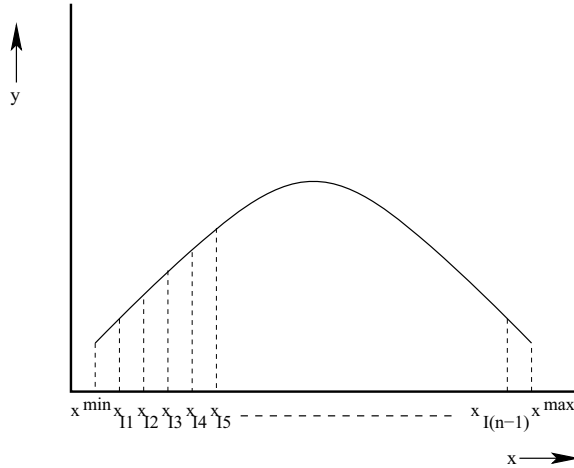


Figure 2.6: A schematic view used to explain the principle of an exhaustive search method.

Fig. 2.6 shows the plot of the function to be optimized. Depending on the desired accuracy in the solution, let us suppose that the range of the variable, that is, $(x^{max} - x^{min})$ is divided into n equal parts. Thus, the number of equi-spaced intermediate points lying in this range becomes equal to $(n - 1)$. The small change in x , that is, Δx is given by the expression $\frac{(x^{max} - x^{min})}{n}$. The following steps are used to locate the maximum value of the function:

- **Step 1:** We set

$$\begin{aligned} x_1 &= x^{min} \\ x_2 &= x_1 + \Delta x = x_{I1} \\ x_3 &= x_2 + \Delta x = x_{I2} \end{aligned}$$

Therefore, the values of x_1 , x_2 and x_3 are in ascending order.

- **Step 2:** We calculate the function values, that is, $f(x_1), f(x_2), f(x_3)$ and check for the maximum point.

If $f(x_1) \leq f(x_2) \geq f(x_3)$, the maximum point lies in the range of (x_1, x_3) . We terminate the program,

Else

$$\begin{aligned}x_1 &= x_2(\text{previous}) \\x_2 &= x_3(\text{previous}) \\x_3 &= x_2(\text{present}) + \Delta x\end{aligned}$$

- **Step 3:** We check whether x_3 exceeds x^{max} .

If x_3 does not exceed x^{max} , we go to Step 2,

Else we say that the maximum does not lie in the range of (x^{min}, x^{max}) .

A Numerical Example:

$$\text{Maximize } y = f(x) = 2x^2 - x^3 \quad (2.13)$$

subject to

$$0.0 \leq x \leq 2.0.$$

Solution:

Fig. 2.7 shows the plot of this function.

Given: $x^{min} = 0.0$ and $x^{max} = 2.0$.

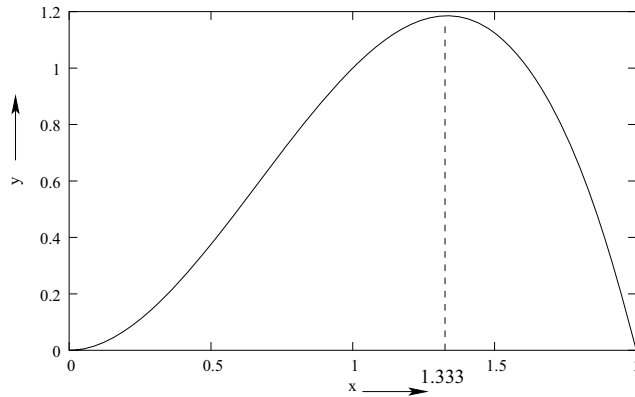


Figure 2.7: Plot of the function $y = 2x^2 - x^3$.

- **Step 1:** Let us suppose that the range $(x^{max} - x^{min})$ is divided into $n = 10$ equal parts. We calculate the incremental change in x value, that is, Δx as follows:

$$\Delta x = \frac{x^{max} - x^{min}}{n} = \frac{2.0 - 0.0}{10} = 0.2$$

We set

$$\begin{aligned} x_1 &= x^{min} = 0.0 \\ x_2 &= x_1 + \Delta x = 0.0 + 0.2 = 0.2 \\ x_3 &= x_2 + \Delta x = 0.2 + 0.2 = 0.4 \end{aligned}$$

- **Step 2:** We determine the function values at $x = x_1, x_2$ and x_3 like the following.

$$\begin{aligned} f(x_1) &= f(0.0) = 0.0 \\ f(x_2) &= f(0.2) = 0.072 \\ f(x_3) &= f(0.4) = 0.256 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.0, 0.4)$.

- **Step 3:** We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 0.2 \\ x_2 &= x_3(\text{previous}) = 0.4 \\ x_3 &= x_2(\text{present}) + \Delta x = 0.4 + 0.2 = 0.6 \end{aligned}$$

- **Step 4:** We calculate the function values as follows:

$$\begin{aligned} f(x_1) &= f(0.2) = 0.072 \\ f(x_2) &= f(0.4) = 0.256 \\ f(x_3) &= f(0.6) = 0.504 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.2, 0.6)$.

- **Step 5:** We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 0.4 \\ x_2 &= x_3(\text{previous}) = 0.6 \\ x_3 &= x_2(\text{present}) + \Delta x = 0.6 + 0.2 = 0.8 \end{aligned}$$

- **Step 6:** We calculate the function values as follows:

$$\begin{aligned}f(x_1) &= f(0.4) = 0.256 \\f(x_2) &= f(0.6) = 0.504 \\f(x_3) &= f(0.8) = 0.768\end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.4, 0.8)$.

- **Step 7:** We set

$$\begin{aligned}x_1 &= x_2(\text{previous}) = 0.6 \\x_2 &= x_3(\text{previous}) = 0.8 \\x_3 &= x_2(\text{present}) + \Delta x = 0.8 + 0.2 = 1.0\end{aligned}$$

- **Step 8:** We calculate the function values as follows:

$$\begin{aligned}f(x_1) &= f(0.6) = 0.504 \\f(x_2) &= f(0.8) = 0.768 \\f(x_3) &= f(1.0) = 1.0\end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.6, 1.0)$.

- **Step 9:** We set

$$\begin{aligned}x_1 &= x_2(\text{previous}) = 0.8 \\x_2 &= x_3(\text{previous}) = 1.0 \\x_3 &= x_2(\text{present}) + \Delta x = 1.0 + 0.2 = 1.2\end{aligned}$$

- **Step 10:** We calculate the function values as follows:

$$\begin{aligned}f(x_1) &= f(0.8) = 0.768 \\f(x_2) &= f(1.0) = 1.0 \\f(x_3) &= f(1.2) = 1.152\end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(0.8, 1.2)$.

- **Step 11:** We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 1.0 \\ x_2 &= x_3(\text{previous}) = 1.2 \\ x_3 &= x_2(\text{present}) + \Delta x = 1.2 + 0.2 = 1.4 \end{aligned}$$

- **Step 12:** We calculate the function values as follows:

$$\begin{aligned} f(x_1) &= f(1.0) = 1.0 \\ f(x_2) &= f(1.2) = 1.152 \\ f(x_3) &= f(1.4) = 1.176 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) < f(x_3).$$

Thus, the maximum does not lie in the interval $(1.0, 1.4)$.

- **Step 13:** We set

$$\begin{aligned} x_1 &= x_2(\text{previous}) = 1.2 \\ x_2 &= x_3(\text{previous}) = 1.4 \\ x_3 &= x_2(\text{present}) + \Delta x = 1.4 + 0.2 = 1.6 \end{aligned}$$

- **Step 14:** We calculate the function values as follows:

$$\begin{aligned} f(x_1) &= f(1.2) = 1.152 \\ f(x_2) &= f(1.4) = 1.176 \\ f(x_3) &= f(1.6) = 1.024 \end{aligned}$$

Therefore,

$$f(x_1) < f(x_2) > f(x_3).$$

Thus, the maximum lies in the interval $(1.2, 1.6)$.

A coarse interval of x , that is, $(1.2, 1.6)$ is obtained using $n = 10$. However, a finer interval can be obtained by setting n to a higher value.

Note: The maximum value of this function is found to be equal to 1.185 analytically and it occurs at $x = 1.333$.

2.2.2 Random Walk Method

Random walk method is one of the **direct search** methods, in which the search is carried out using the objective function value but its derivative information is not utilized [9]. Here, the present solution (that is, $(i + 1)$ -th) is determined using the previous solution (that is, i -th), according to the rule given below.

$$X_{i+1} = X_i + \lambda u_i, \quad (2.14)$$

where $X = (x_1, x_2, \dots, x_m)^T$, m is the number of variables, λ indicates the step length, u_i is a unit random vector determined as $u_i = \frac{1}{(r_1^2 + r_2^2 + \dots + r_n^2)^{\frac{1}{2}}} (r_1, r_2, \dots, r_n)^T$, where (r_1, r_2, \dots, r_n) are the random numbers lying between -1.0 and 1.0 , such that $n = m$ and $(r_1^2 + r_2^2 + \dots + r_n^2)^{\frac{1}{2}} = R' \leq 1.0$. It is to be noted that the number of random numbers, that is, n is kept equal to the number of variables, that is, m .

The following steps are considered to determine the optimum (say minimum) solution:

- **Step 1:** We start with an initial solution X_1 , created at random. We set initial values to λ , ϵ (permissible minimum value of λ) and maximum number of iterations to be tried for improvement of the solution (that is, N). We determine the function value $f_1 = f(X_1)$.
- **Step 2:** A set of n random numbers (lying between -1.0 and 1.0) are generated and we calculate u_1 .
- **Step 3:** We determine the function value using the expression given below.

$$f_2 = f(X_2) = f(X_1 + \lambda u_1)$$

- **Step 4:** If $f_2 < f_1$, then we set $X_1 = X_1 + \lambda u_1$, $f_1 = f_2$, and repeat the Steps 2 through 4.
Else repeat Steps 2 through 4, up to the maximum number of iterations N .
- **Step 5:** If a better point X_{i+1} is not obtained after running the program for N iterations, we reduce λ to 0.5λ .
- **Step 6:** Is the modified (new) $\lambda < \epsilon$?
If no, go to Step 2,
Else we declare $X^* = X_1$, $f^* = f_1$, and terminate the program.

A Numerical Example:

$$\text{Minimize } f(x_1, x_2) = 4x_1^2 + 3x_2^2 - 6x_1x_2 - 4x_1 \quad (2.15)$$

subject to

$$-10.0 \leq x_1, x_2 \leq 10.0.$$

using random walk method.

Assume: $\lambda = 1.2$, $\epsilon = 0.075$, $N = 100$.

Solution:

Fig. 2.8 shows the plot of the function.

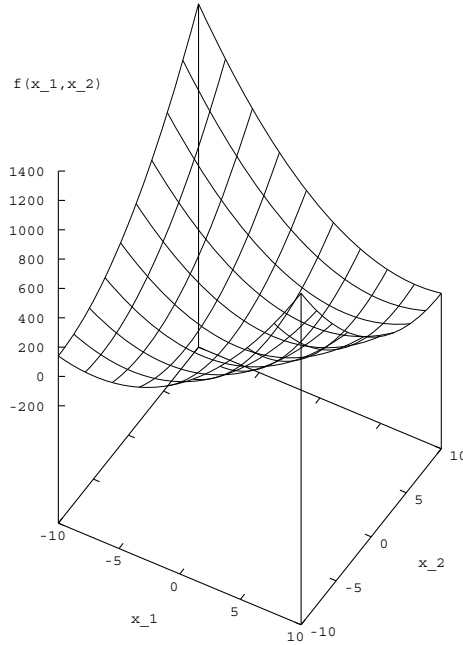


Figure 2.8: Plot of the function $f(x_1, x_2) = 4x_1^2 + 3x_2^2 - 6x_1x_2 - 4x_1$.

Some of the iterations carried out to solve the above problem using the random walk method are:

• **Iteration 1** It consists of the following steps:

– **Step 1:** Let us consider an initial solution generated at random as given below.

$$X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$$

We determine the function value f_1 corresponding to X_1 like the following.

$$f_1 = f(X_1) = 0.0.$$

– **Step 2:** Let us also assume that the following two random numbers lying between -1.0 and 1.0 have been generated:

$$\begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} -0.5 \\ 0.2 \end{Bmatrix}$$

We calculate $R' = (r_1^2 + r_2^2)^{\frac{1}{2}} = 0.5385$, which is less than 1.0. So, we consider the above r values for determination of the unit random vector as given below.

$$u = \frac{1}{(r_1^2 + r_2^2)^{\frac{1}{2}}} \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \frac{1}{0.5385} \begin{Bmatrix} -0.5 \\ 0.2 \end{Bmatrix} = \begin{Bmatrix} -0.9285 \\ 0.3714 \end{Bmatrix}$$

- **Step 3:** We determine the function value f_2 as follows:

$$f_2 = f(X_2) = f(X_1 + \lambda u_1) = f(-1.1142, 0.4457) = 12.9981.$$

As $f_2 > f_1$, $\begin{Bmatrix} -1.1142 \\ 0.4457 \end{Bmatrix}$ cannot be considered as X_2 and we go for the next iteration.

• Iteration 2

- **Step 1:** We have $X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$ and $f_1 = f(X_1) = 0.0$.

- **Step 2:** Let us consider another set of random numbers lying between -1.0 and 1.0 as follows:

$$\begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \begin{Bmatrix} 0.001 \\ 0.1 \end{Bmatrix}$$

We calculate $R' = (r_1^2 + r_2^2)^{\frac{1}{2}} = 0.1000$, which is less than 1.0. So, the above set of r values can be used to determine the unit random vector like the following.

$$u = \frac{1}{(r_1^2 + r_2^2)^{\frac{1}{2}}} \begin{Bmatrix} r_1 \\ r_2 \end{Bmatrix} = \frac{1}{0.1} \begin{Bmatrix} 0.001 \\ 0.1 \end{Bmatrix} = \begin{Bmatrix} 0.01 \\ 1.0 \end{Bmatrix}$$

- **Step 3:** We determine the function value f_2 as follows:

$$f_2 = f(X_2) = f(X_1 + \lambda u_1) = f(0.012, 1.2) = 4.1862.$$

As $f_2 > f_1$, $\begin{Bmatrix} 0.012 \\ 1.2 \end{Bmatrix}$ will not be considered as X_2 and we go for the next iteration.

If X_2 cannot be obtained after running the program for $N = 100$ iterations, we set $\lambda_{new} = 0.5 \times \lambda$. If $\lambda_{new} > \epsilon$, we repeat the iterations as explained above, else we declare optimal $X^* = X_1$, $f^* = f_1$.

2.2.3 Steepest Descent Method

Steepest descent method is one of the **gradient-based** methods, in which the search direction is dependent on the gradient of the objective function to be optimized [9]. Thus, this method is not applicable to a discontinuous function. Let us define the term: *gradient* of a function, prior to explaining the principle of this method.

Gradient of a function:

Let us consider a continuous function of m variables as given below.

$$y = f(X) = f(x_1, x_2, \dots, x_m) \quad (2.16)$$

Gradient of this function is defined as the collection of its partial derivatives with respect to each of the m variables and it is denoted as follows:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_m} \right)^T \quad (2.17)$$

It is important to mention that the rate of change of a function is found to be the maximum along its gradient direction. Thus, the direction of its gradient is nothing but the direction of steepest ascent. In a steepest descent method, the search is carried out along a direction opposite to its gradient. Fig. 2.9 shows the search directions of this algorithm at different iterations.

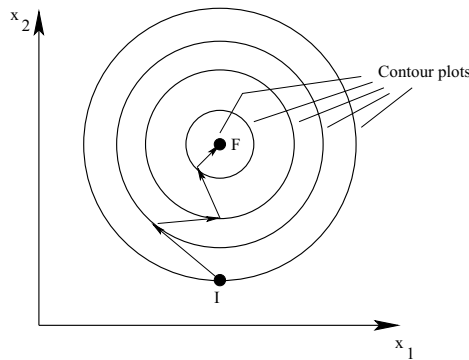


Figure 2.9: A schematic view showing search directions of a steepest descent method.

Principle of the method:

We start with an initial solution X_1 , created at random and move along the search direction, according to the rule given below until it reaches the optimum point.

$$X_{i+1} = X_i + \lambda_i^* S_i, \quad (2.18)$$

where X_i and X_{i+1} are the solution vectors at i -th and $(i+1)$ -th iterations, respectively, λ_i^* represents the optimal step length along the search direction $S_i = -\nabla f_i$.

Termination Criteria:

Any one of the following two criteria can be considered as the termination criterion of the algorithm.

1. If the rate of change of function value, that is, $\left| \frac{f(X_{i+1}) - f(X_i)}{f(X_i)} \right|$ becomes less than or equal to a pre-defined small quantity ϵ_1 , the algorithm is terminated.
2. If the derivatives $\left| \frac{\partial f}{\partial x_j} \right|$, $j = 1, 2, \dots, m$, come out to be less than or equal to a pre-defined small quantity ϵ_2 , the program is terminated.

Advantages:

It is one of the most popular traditional methods of optimization, due to the following reasons:

1. It has a faster convergence rate.
2. The algorithm is simple and easy to understand and implement.

Limitation of the algorithm:

The search direction of this algorithm is nothing but the steepest descent direction, which is opposite to the gradient direction. As gradient is a local property of the function (that means, it may vary from point to point lying on the function), there is a chance of the solutions of this algorithm for being trapped into the local minima.

A Numerical Example:

$$\text{Minimize } f(x_1, x_2) = 4x_1^2 + 3x_2^2 - 6x_1x_2 - 4x_1 \quad (2.19)$$

subject to

$$-10.0 \leq x_1, x_2 \leq 10.0.$$

Fig. 2.8 shows the plot of the function. A few iterations of the steepest descent method used to solve this problem are:

• Iteration 1:

Let us assume the initial solution created at random as $X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$.

The function value at X_1 is determined as $f_1 = 0.0$.

Gradient of the function is obtained as follows:

$$\nabla f = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{Bmatrix} = \begin{Bmatrix} 8x_1 - 6x_2 - 4 \\ -6x_1 + 6x_2 \end{Bmatrix}$$

Now, the gradient of the objective function is found to be like the following at X_1 .

$$\nabla f_1 = \nabla f(X_1) = \begin{Bmatrix} -4 \\ 0 \end{Bmatrix}$$

Therefore, the search direction at X_1 is determined as follows:

$$S_1 = -\nabla f_1 = \begin{Bmatrix} 4 \\ 0 \end{Bmatrix}$$

To determine X_2 , we need to know the value of optimal step length, that is, λ_1^* , for the estimation of which, we minimize

$$f(X_1 + \lambda_1 \times S_1) = f(4\lambda_1, 0) = 64\lambda_1^2 - 16\lambda_1$$

We determine $\frac{df}{d\lambda_1}$ and put it equal to 0.0. The optimal step length is found to be equal to $\frac{1}{8}$, that is, $\lambda_1^* = \frac{1}{8}$.

We calculate $X_2 = X_1 + \lambda_1^* S_1 = \left\{ \begin{array}{c} \frac{1}{2} \\ 0 \end{array} \right\}$

The function value at X_2 is calculated as $f_2 = -1.0$.

We determine the gradient of the function at X_2 , that is,

$$\nabla f_2 = \nabla f(X_2) = \left\{ \begin{array}{c} 0 \\ -3 \end{array} \right\} \neq \left\{ \begin{array}{c} 0 \\ 0 \end{array} \right\}$$

Therefore, X_2 is not the desired optimum point and we should go for the next iteration.

• **Iteration 2:**

The search direction at X_2 is determined as follows:

$$S_2 = -\nabla f_2 = \left\{ \begin{array}{c} 0 \\ 3 \end{array} \right\}$$

To determine X_3 , we need to know the value of optimal step length, that is, λ_2^* , for the estimation of which, we minimize

$$f(X_2 + \lambda_2 \times S_2) = f\left(\frac{1}{2}, 3\lambda_2\right) = 27\lambda_2^2 - 9\lambda_2 - 1$$

We put $\frac{df}{d\lambda_2} = 0.0$ and get the optimal step length, that is, $\lambda_2^* = \frac{1}{6}$.

We calculate $X_3 = X_2 + \lambda_2^* S_2 = \left\{ \begin{array}{c} \frac{1}{2} \\ \frac{1}{2} \end{array} \right\}$

The function value at X_3 is determined as $f_3 = -\frac{7}{4}$.

The gradient of the function at X_3 is determined as follows:

$$\nabla f_3 = \nabla f(X_3) = \left\{ \begin{array}{c} -3 \\ 0 \end{array} \right\} \neq \left\{ \begin{array}{c} 0 \\ 0 \end{array} \right\}$$

Therefore, X_3 is not the desired optimum point. We should go for the next iteration.

In the similar way, some more iterations are to be carried out, until it reaches the optimum (minimum) point. Finally, the optimal solution of this function is found to be as follows:

$$X_{opt} = \left\{ \begin{array}{c} 2.0 \\ 2.0 \end{array} \right\}, f_{opt} = -4.0.$$

2.2.4 Drawbacks of Traditional Optimization Methods

Traditional methods of optimization have the following drawbacks:

1. The final solution of an optimization problem solved using a traditional method depends on the randomly chosen initial solution. If the initial solution lies in the local basin, the final solution will get stuck at local optimum. Thus, if the user is lucky enough to choose the initial solution lying in the global basin, the obtained optimal solution may be global in nature (refer to Fig. 2.10).

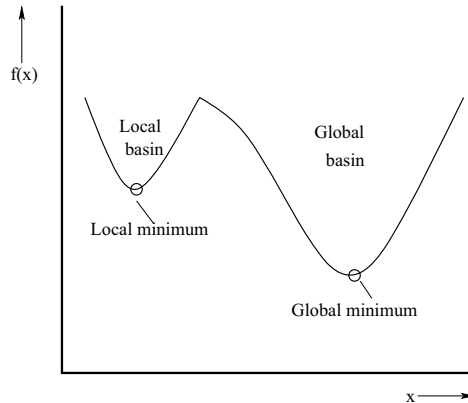


Figure 2.10: A schematic view showing the local and global basins of an objective function.

2. For a discontinuous objective function, the gradient cannot be determined at the point of discontinuity. Thus, the gradient-based methods cannot be used for such functions.
3. There is a chance of the solutions of a gradient-based optimization method for being trapped into the local minima.
4. Discrete (integer) variables are difficult to handle using the traditional methods of optimization, although there exists a separate integer programming method.
5. These methods may not be suitable for parallel computing, in which the total computation is distributed into a number of computers and they communicate one another using an MPI (Message Passing Interface) algorithm.
6. A particular traditional method of optimization may not be suitable to solve a variety of problems.

Thus, it is necessary to develop an optimization method, which is not only efficient but also robust in nature.

2.3 Summary

The content of this chapter may be summarized as follows:

1. A brief introduction is given to the concept of optimization. Different terms related to optimization, namely decision variable, objective function, functional constraint, geometric constraint, and others, have been defined with the help of a practical example.
2. Optimization problems have been classified in a number of ways, into different groups, namely linear optimization problem, non-linear optimization problem, constrained optimization problem, un-constrained optimization problem, integer variables problem, real variables problem, mixed-integer variables problem, static and dynamic optimization problems, and others.
3. The principle of optimization has been explained with the help of an example.
4. There exists a number of traditional methods of optimization. A few of them, namely Exhaustive Search Method, Random Walk Method and Steepest Descent Method, have been explained in detail with some suitable examples.
5. The drawbacks of the traditional optimization methods are highlighted at the end of this chapter.

2.4 Exercise

1. Define the following terms related to optimization by taking a suitable example: (i) decision variables, (ii) objective function, (iii) functional constraints, (iv) geometric constraints.
2. The presence of geometric constraint(s) is a must for an optimization problem but that of functional constraint(s) is optional – justify the statement.
3. Explain the principle of optimization with a suitable example.
4. In a constrained optimization, an optimal point is either a free point or a bound point lying in the feasible zone – justify the statement.
5. Why do the solutions of a steepest descent method get stuck at the local minima ?
6. Discuss briefly the drawbacks of traditional methods of optimization.
7. Determine the minimum/maximum/inflection point of the following functions: (i) $f(x) = x^3$; (ii) $f(x) = x^4$.

8. In case of turning operation carried out on a Lathe, cutting parameters (such as cutting speed v in m/min , feed t in mm/rev and depth of cut d in mm) are to be selected in such a way that it can produce the smoothest surface after ensuring a minimum life of the cutting tool TL_{min} . Let us assume that surface roughness in turning S ($micro-m$) and life of the turning tool TL (min) are given by the following expressions:

$$S = 15077v^{-1.52}t^{1.004}d^{0.25},$$

$$TL = 1.475 \times 10^9 v^{-4.0} t^{-4.29} d^{-4.35}$$

Formulate it as a constrained optimization problem. The cutting parameters are allowed to vary in the ranges given below.

$$30.0 \leq v \leq 190.0,$$

$$0.01 \leq t \leq 2.5,$$

$$0.5 \leq d \leq 4.0.$$

9. **Minimize** $y = f(x) = \frac{32}{x^2} + x$ in the range of $0.0 < x \leq 10.0$. Use (i) analytical approach based on differential calculus and (ii) exhaustive search method.
(Hints: Let the three values of x , say x_1 , x_2 and x_3 are in ascending order. For a minimization problem, if $f(x_1) \geq f(x_2) \leq f(x_3)$, then the minimum value lies in the range of (x_1, x_3) .)
10. **Minimize** $f(x_1, x_2) = 4x_1^2 + x_2^2 - 3x_1x_2 + 6x_1 + 12x_2$ in the range of $-100.0 \leq x_1, x_2 \leq 100.0$. Take the initial solution $X_1 = \begin{Bmatrix} 0.0 \\ 0.0 \end{Bmatrix}$.
 (i) Use Random Walk Method. Assume step length $\lambda = 1.0$, permissible minimum value of λ , that is, $\epsilon = 0.25$ and maximum number of iterations $N = 50$.
 (ii) Use Steepest Descent Method.

Chapter 3

Introduction to Genetic Algorithms

Genetic and evolutionary algorithms, which work based on Darwin's principle of natural selection (that is, survival of the fittest), have been used as optimization tools. These algorithms include Genetic Algorithm (GA) [11], Genetic Programming (GP) [12], Evolution Strategies (ES) [13], Evolutionary Programming (EP) [14, 15]. A detailed discussion on each of these algorithms is beyond the scope of this book and it concentrates only on the GA. An introduction is given to a Genetic Algorithm (GA), one of the most popular non-traditional methods of optimization, in this chapter. Several versions of GA are available in the literature, such as binary-coded GA [16], real-coded GA [17, 18], micro-GA [19], messy-GA [20], and others. An attempt will be made to explain the working principle of a binary-coded GA, in this chapter.

3.1 Working Cycle of a Genetic Algorithm

Genetic Algorithm (GA) is a population-based probabilistic search and optimization technique, which works based on the mechanism of natural genetics and Darwin's principle of natural selection. The GA was introduced by Prof. John Holland of the University of Michigan, Ann Arbor, USA, in 1965, although his seminal book was published in 1975 [11]. This book could lay the foundation of the GAs. It is basically an iterative search technique working based on the concept of probability. The working principle of a GA has been explained briefly as follows (refer to Fig. 3.1):

- A GA starts with a population of initial solutions generated at random.
- The fitness/goodness value (that is, objective function value in case of a maximization problem) of each solution in the population is calculated. It is important to mention that a GA is generally designed to solve a maximization problem. Thus, a minimization problem has to be converted into a corresponding maximization problem as discussed below.

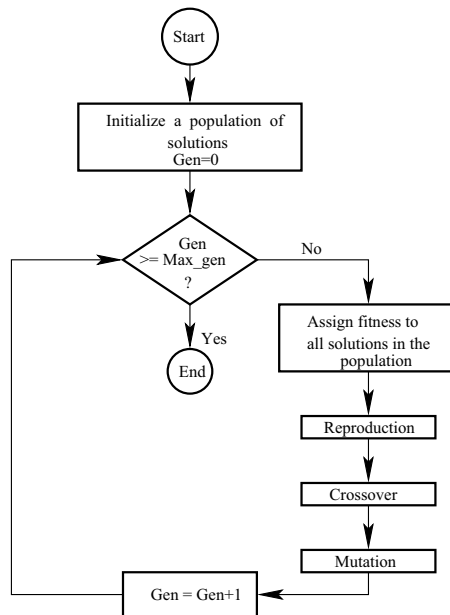


Figure 3.1: A schematic view showing the working cycle of a GA.

Let us suppose that a function $f(x)$ is to be minimized. It may be converted into a maximization problem as follows:

Either Maximize $-f(x)$, using duality principle,
 or Maximize $\frac{1}{f(x)}$, for $f(x) \neq 0$,
 or Maximize $\frac{1}{1+f(x)}$, for $f(x) \geq 0$,
 or Maximize $\frac{1}{1+\{f(x)\}^2}$, and so on.

- The population of solutions is then modified using different operators, namely reproduction, crossover, mutation, and others. The role of each of these operators is discussed below.
- All the solutions in a population may not be equally good in terms of their fitness values. An operator named **reproduction** (also known as selection scheme) is utilized to select the good solutions using their fitness values. Thus, it forms a mating pool consisting of good solutions probabilistically. It is important to note that the mating pool may contain multiple copies of a particular good solution. The size of the mating pool is kept equal to that of the population of solutions considered before reproduction. Thus, the average fitness of the mating pool is expected to be higher than that of the pre-reproduction population of solutions. There exists a number of reproduction schemes in the GA-literature, namely proportionate selection (such as Roulette-Wheel selection), tournament selection, ranking selection, and others [21].

- The mating pairs (also known as the parents) are selected at random from the above pool, which may participate in **crossover** depending on the value of crossover probability. In crossover, there is an exchange of properties between the parents and as a result of which, new children solutions are created. It is important to mention that if the parents are good, the children are expected to be good. Various types of crossover operators are available in the GA-literature, such as single-point crossover, two-point crossover, multi-point crossover, uniform crossover, and others [22].
- In biology, the word: **mutation** means a sudden change of parameter. For example, the crows are generally black in colour. However, if we could see a white crow by chance, it might have happened due to a sudden change in parameter on the gene level, which is known as biological mutation. In a GA-search, it is used for achieving a local change around the current solution. Thus, if a solution gets stuck at the local minimum, this operator may help it to come out of this situation and consequently, it may also jump into the global basin.
- After the reproduction, crossover and mutation are applied to the whole population of solutions, one generation of a GA is completed. Different criteria may be used to terminate the program, such as the maximum number of generations, a desired accuracy in the solution, and others.

3.2 Binary-Coded GA

Let us consider an optimization problem to explain the working principle of a binary-coded GA, as given below.

$$\text{Maximize } y = f(x_1, x_2) \quad (3.1)$$

subject to

$$\begin{aligned} x_1^{\min} &\leq x_1 \leq x_1^{\max}, \\ x_2^{\min} &\leq x_2 \leq x_2^{\max}, \end{aligned}$$

where x_1 and x_2 are the real variables. The following steps are utilized to solve the above problem using a binary-coded GA:

- **Step 1 - Generation of a population of solutions:** An initial population of solutions of size N (say $N = 100, 200, \dots$, depending on the complexity of the problem) is selected at random. In this type of GA, the solutions are represented in the form of binary strings composed of 1s and 0s. A binary string can be compared to a biological chromosome and each bit of the string is nothing but a gene value. The length of a binary string is decided based on a desired precision in the values of the variables. For example, if we need a precision level of ϵ in the values of a variable x_1 , we will

have to assign l bits to represent it, where l can be determined using the expression given below.

$$l = \log_2 \left(\frac{x_1^{max} - x_1^{min}}{\epsilon} \right). \quad (3.2)$$

It is obvious that computational complexity of a binary-coded GA is dependent on its string length L , which is found to be around $L \log L$.

Let us assume that 10 bits are assigned to represent each variable. Thus, in this problem, the GA-string is 20-bits long. Let us also suppose that the initial population of GA-strings created at random is:

$$\begin{array}{cccccc} 1 & 0 & 0 & \dots & 1 \\ 0 & 1 & 1 & \dots & 0 \\ 1 & 1 & 1 & \dots & 0 \\ 0 & 0 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & 0 & 1 & \dots & 1 \end{array}$$

- **Step 2 - Fitness evaluation:** To determine the fitness value of each solution (that is, string), the real values of the variables: x_1 and x_2 are to be calculated first. If the minimum and maximum limits of a variable (say x_1) and decoded value of the binary sub-string assigned to represent x_1 are known, its real value can be determined using the linear mapping rule given below.

$$x_1 = x_1^{min} + \frac{x_1^{max} - x_1^{min}}{2^l - 1} \times D, \quad (3.3)$$

where l is the length of the sub-string used to represent the variable x_1 and D represents the decoded value of the binary sub-string. The decoded value of a binary number is calculated as follows: Let us consider a binary number 1 0 1 1 0. Its decoded value is determined as $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22$. The real value of second variable x_2 is determined following the same procedure. Once the real values of the variables: x_1 and x_2 are known, the function value $f(x_1, x_2)$ can be calculated, which is considered as the fitness value of the binary-string. This procedure is repeated to determine fitness of each string of the GA-population.

- **Step 3 - Reproduction:** All the GA-strings contained in the population may not be equally good in terms of their fitness values calculated in Step 2. In this step, an operator named reproduction is used to select the good ones from the population of strings based on their fitness information. Several reproduction schemes (also called selection schemes) have been developed by various investigators. Some of these are explained below.

1. **Proportionate Selection/Roulette-Wheel Selection** - In this scheme, the probability of a string for being selected in the mating pool is considered to

be proportional to its fitness. It is implemented with the help of a Roulette-Wheel shown in Fig. 3.2. The top surface area of the wheel is divided into N

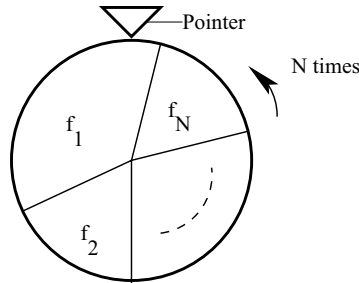


Figure 3.2: A Roulette-Wheel used to implement proportionate selection scheme.

parts (where N is the population size), in proportion to the functional values: f_1, f_2, \dots, f_N . The wheel is rotated in a particular direction (either clockwise or anti-clockwise) and a fixed pointer is used to indicate the winning area, after it stops. A particular sub-area representing a GA-solution is selected to be winner probabilistically and the probability that i - *th* area will be declared so, is given by the following expression:

$$p = \frac{f_i}{\sum_{i=1}^N f_i}. \quad (3.4)$$

The wheel is rotated/spun for N times and each time, only one area is identified by the pointer to be the winner. The procedure is shown below in detail.

<i>GA – strings</i>					<i>Fitness</i>	<i>Probability of being selected</i>
1	0	0	...	1	f_1	$\frac{f_1}{\sum_{i=1}^N f_i}$
0	1	1	...	0	f_2	$\frac{f_2}{\sum_{i=1}^N f_i}$
1	1	1	...	0	f_3	$\frac{f_3}{\sum_{i=1}^N f_i}$
0	0	1	...	1	f_4	$\frac{f_4}{\sum_{i=1}^N f_i}$
\vdots	\vdots	\vdots	...	\vdots	\vdots	\vdots
1	0	1	...	1	f_N	$\frac{f_N}{\sum_{i=1}^N f_i}$

In this scheme, a good string may be selected for a number of times. Thus, the resulting mating pool (of size N) may look as follows:

1	0	1	...	1
1	1	1	...	0
1	1	1	...	0
\vdots	\vdots	\vdots	...	\vdots
1	0	1	...	1

A GA-search depends on the factors like **population diversity** and **selection pressure**, which are similar to the concepts of exploration and exploitation, respectively, as used in some of the studies on GA. These two factors are inversely related to each other in the sense that if the selection pressure increases, the population diversity will decrease, and vice-versa. It is important to mention that if the selection pressure increases, the search will be focused only on the good individuals (in terms of fitness) and as a result of which, the diversity will be lost. It may also lead to a premature convergence of the solution to a sub-optimal value. On the other hand, a lower value of selection pressure may not be able to drive the search properly and consequently, a stagnation may occur. Thus, selection pressure plays an important role in the GA-search and it has to be put appropriately to ensure a good search. It is also to be noted that the fitness-based above reproduction scheme may lead to a premature convergence of the search [23]. To overcome this difficulty by providing a better control over the selection pressure, a rank-based reproduction scheme was proposed by Baker [24], which is explained next.

The following procedure may be adopted to implement Roulette-Wheel selection scheme in the computer program: For simplicity, let us assume that there are only four binary strings in the population of solutions having the fitness values: f_1, f_2, f_3 and f_4 . We arrange the strings in ascending order of their fitness values as f_3, f_1, f_4, f_2 . Let us also assume that their probability values for being selected in the mating pool are as follows: $p_1 = 0.15, p_2 = 0.6, p_3 = 0.05$ and $p_4 = 0.2$, according to this selection scheme. Let us consider a random number generator, which generates a number (say r) in the range of $(0.0, 1.0)$. If r is found to lie in the ranges: $0.0 \leq r < 0.05, 0.05 \leq r < 0.2, 0.2 \leq r < 0.4$ and $0.4 \leq r \leq 1.0$, then the $3 - rd, 1 - st, 4 - th$ and $2 - nd$ string will be selected for the mating pool, respectively.

2. **Ranking Selection** - Let us assume that there are only four binary strings in a population of GA-solution, whose fitness values are indicated by f_1, f_2, f_3 and f_4 . Let us also suppose that f_1, f_2, f_3 and f_4 take the values of 80%, 10%, 7% and 3% of the total fitness value (that is, $\sum_{i=1}^4 f_i$), respectively (refer to Fig. 3.3). If a fitness-based proportionate selection is carried out, the probabilities of being selected for f_1, f_2, f_3 and f_4 are turning out to be equal to 0.8, 0.1, 0.07 and 0.03, respectively. Thus, there is a chance of occurrence of the premature convergence. To overcome this problem, a rank-based reproduction scheme may be adopted, whose principle is discussed below.

The process of ranking selection consists of two steps. In the first step, the strings are arranged in an ascending order of their fitness values (that is, f_4, f_3, f_2, f_1). The string having the lowest fitness is assigned rank 1 and other strings are ranked accordingly. Thus, in this example, the strings f_4, f_3, f_2 and f_1 will be assigned the ranks 1, 2, 3 and 4, respectively. In the second step, a proportionate selection scheme based on the assigned rank is adopted, as discussed below.

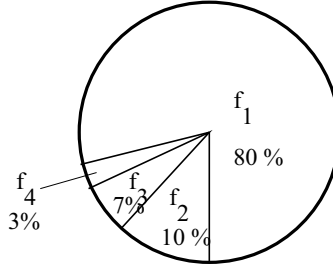


Figure 3.3: A schematic view showing the fitness-based proportionate selection.

The percent (%) area to be occupied by a particular string ($i - th$) is given by the expression $\frac{R_i}{\sum R} \times 100$, where R_i indicates the rank of $i - th$ string. Thus, the strings f_1 , f_2 , f_3 and f_4 will occupy 40%, 30%, 20% and 10% of the total area, respectively, as shown in Fig. 3.4, and consequently, their probabilities for being selected in the mating pool become equal to 0.4, 0.3, 0.2 and 0.1. It is important to mention that a rank-based proportionate selection scheme is

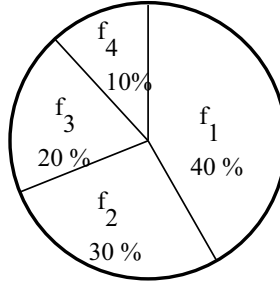


Figure 3.4: A schematic view showing the rank-based proportionate selection.

expected to perform better than the fitness-based proportionate selection.

3. **Tournament Selection** - Tournament selection was studied first in Brindle's dissertation [25]. In this scheme, we select the tournament size n (say 2 or 3) at random, which is a smaller number compared to the population size N . We pick n strings from the population, at random and determine the best one in terms of fitness value. The best string is copied into the mating pool and then all n strings are returned to the population. Thus, in this scheme, only one string is selected per tournament and N tournaments are to be played to make the size of mating pool equal to N . Here, there is a chance for a good string to be copied in the mating pool more than once. It is found to be computationally faster than both the fitness-based and rank-based proportionate selection schemes.

Interested readers may refer to the work of Goldberg and Deb [21] for a comparative analysis of different selection schemes.

4. **Elitism** - This scheme was proposed by Kenneth De Jong [26], in which an elite

string (in terms of fitness) is identified first in a population of strings. It is then directly copied into the next generation to ensure its presence. This is done because the already found best string may be lost, if it is not selected during reproduction using any one of the above schemes.

- **Step 4 - Crossover:** In crossover, there is an exchange of properties between two parents and as a result of which, two children solutions are produced. To carry out this operation, the parents or mating pairs (each pair consists of two strings) are selected at random from the mating pool. Thus, $\frac{N}{2}$ mating pairs are formed from a population of strings of size N . The parents are checked, whether they will participate in crossover by tossing a coin, whose probability of appearing head is p_c . If the head appears, the parent will participate in crossover to produce two children. Otherwise, they will remain intact in the population. It is important to mention that p_c is generally kept near to 1.0, so that almost all the parents can participate in the crossover.

The following procedure may be adopted to implement coin-flipping artificially: A number lying between 0.0 and 1.0 is generated using a random number generator. If the random number is found to be smaller than or equal to p_c , the outcome of coin-flipping is considered as true (that is, head), otherwise it is false. If the head appears, the parents will participate in crossover.

Once a particular mating pair has been selected for crossover, the crossing site is decided using a random number generator generating an integer lying between 1 and $L - 1$, where L is the length of the string. The number of individuals participating in the crossover can be probabilistically determined and it is found to be equal to Np_c . There exist a large number of crossover schemes in the GA-literature. Some of these schemes are explained below.

1. **Single-point Crossover** - We select a crossover site lying between 1 and $L - 1$, at random, where L indicates the string length. The left side of the crossover site is generally kept unaltered and swapping is done between the two sub-strings lying on right side of the crossover site. It is to be noted that children solutions will remain the same, whether the swapping is done on either left side or right side of the crossover site. However, we generally keep the left side unaltered and the bits lying on the right side of crossover site are interchanged. The parents participating in a single-point crossover are shown below.

0	1	0	1	1	0	1	0	1	1		1	0	0	1	1	0	1	0	0	1
0	0	1	1	0	1	0	0	1	0		1	1	1	0	1	0	0	1	0	1

Two children produced due to the single-point crossover are given below.

0	1	0	1	1	0	1	0	1	1		1	1	1	0	1	0	0	1	0	1
0	0	1	1	0	1	0	0	1	0		1	0	0	1	1	0	1	0	0	1

2. **Two-point Crossover** - We select two different crossover sites lying between 1 and $L - 1$, at random. The parent strings participating in this crossover are:

```

1 0 1 0 1 1 | 1 0 0 1 1 | 0 1 0 0 1 0 1 0 0
0 1 0 0 1 0 | 1 1 1 0 1 | 0 0 0 1 1 0 0 1 1

```

Two children strings produced due to the two-point crossover are:

```

1 0 1 0 1 1 | 1 1 1 0 1 | 0 1 0 0 1 0 1 0 0
0 1 0 0 1 0 | 1 0 0 1 1 | 0 0 0 1 1 0 0 1 1

```

3. **Multi-point Crossover** - In case of multi-point crossover [27], a number of crossover points (either an odd number or an even number greater than two) are selected along the length of the strings, at random. The bits lying between alternate pairs of sites are then interchanged. The parent strings are shown below, in which five crossover sites are chosen at random.

```

1 0 1 | 1 0 0 0 | 0 1 1 1 | 0 0 1 1 | 0 1 1 | 1 0
0 1 1 | 1 0 1 1 | 0 0 0 1 | 1 0 0 0 | 1 0 1 | 0 0

```

The bits of the two strings lying between sites 1 and 2; 3 and 4; and on the right side of site 5 are interchanged and the remaining bits are kept unaltered. The generated children strings are given below.

```

1 0 1 | 1 0 1 1 | 0 1 1 1 | 1 0 0 0 | 0 1 1 | 0 0
0 1 1 | 1 0 0 0 | 0 0 0 1 | 0 0 1 1 | 1 0 1 | 1 0

```

If four crossover sites are selected at random on the same parent strings, then it will look as follows:

```

1 0 1 | 1 0 0 0 | 0 1 1 1 | 0 0 1 1 | 0 1 1 1 0
0 1 1 | 1 0 1 1 | 0 0 0 1 | 1 0 0 0 | 1 0 1 0 0

```

In this case, the bits lying between the sites 1 and 2; 3 and 4 are interchanged keeping other bits intact. Thus, the children strings will look like the following:

```

1 0 1 | 1 0 1 1 | 0 1 1 1 | 1 0 0 0 | 0 1 1 1 0
0 1 1 | 1 0 0 0 | 0 0 0 1 | 0 0 1 1 | 1 0 1 0 0

```

4. **Uniform Crossover** - Uniform crossover [28] is a more general version of the multi-point crossover. Let us take an example of the parent strings shown below to explain the principle of uniform crossover.

```

1 0 1 1 0 0 0 0 1 1 1 0 0 1 1 0 1 1 1 0
0 1 1 1 0 1 1 0 0 0 1 1 0 0 0 1 0 1 0 0

```

In this scheme, at each bit position of the parent strings, we toss a coin (with a probability of 0.5 for appearing head) to determine whether there will be interchanging of the bits. If the head appears, there will be a swapping of bits among the parent strings, otherwise they will remain unaltered. Let us assume that the 2-nd, 4-th, 5-th, 8-th, 9-th, 12-th, 15-th, 18-th and 20-th bit positions are selected for swapping. Thus, the obtained children solutions will look as follows:

```

1 1 1 1 0 0 0 0 0 1 1 1 0 1 0 0 1 1 1 0
0 0 1 1 0 1 1 0 1 0 1 0 0 0 1 1 0 1 0 0

```

In another implementation of uniform crossover, a random mask (also known as template) consisting of 1s and 0s of the same string-length as that of the parents is created. The mask is then searched bit-wise starting from one side. At a particular bit-position, if the mask contains 1, then the corresponding bits of parents 1 and 2 will be copied in the said position of children 1 and 2, respectively. On the other hand, if the mask contains 0 at a specific bit-position, then the corresponding bits of parents 1 and 2 will be selected for the said position of children 2 and 1, respectively. Let us consider a mask of 20 bits as given below.

```

1 0 0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0

```

The children solutions will be obtained from the above parents using the said mask as follows:

```

1 1 1 1 0 0 1 0 0 1 1 0 0 0 1 1 1 1 1 0
0 0 1 1 0 1 0 0 1 0 1 1 0 1 0 0 0 1 0 0

```

Comparison of crossover operators [29, 30]:

The contribution of a crossover operator depends on its recombination potential and exploratory power. Due to this recombination potential, crossover can build the higher order hyperplanes by combining the lower order hyperplanes (also known as building blocks). Exploratory power of a crossover operator helps to provide a powerful search in a complex search space. It is to be noted that disruption rate of a crossover operator depends on both its recombination potential as well as exploratory power. Moreover, it is important to mention that disruption rate increases with the number of crossover points for both the multi-point and uniform crossovers. For a large search space, uniform crossover is found to perform better than both the single-point and two-point crossovers.

- **Step 5 - Mutation:** In a GA, the concept of biological mutation is modeled artificially to bring a local change over the current solution. In mutation, 1 is converted into 0 and vice-versa. The role of mutation operator is explained with the help of Fig. 3.5, which shows the plot of an objective function having one local basin and another global basin. Let us suppose that all the randomly generated initial solutions

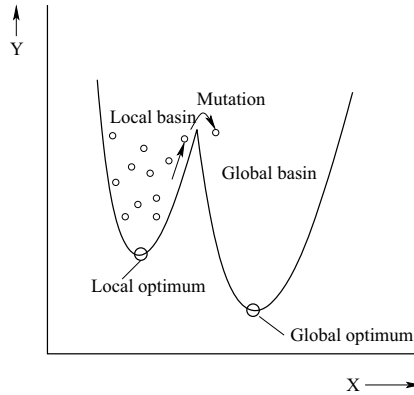


Figure 3.5: A schematic view showing the role of mutation.

of a GA fall on the local basin by chance. Let us also assume that the population of initial solutions looks as follows:

0	0	1	...	1
0	1	1	...	0
0	1	1	...	0
⋮	⋮	⋮	...	⋮
0	0	1	...	1

It is to be noted that the left-most position of each string contains a zero (0). If the situation is such that the globally optimal solution will be indicated only by a string having 1 at the left-most bit-position, the GA cannot find it using the crossover operators like single-point, two-point or multi-point. Under these circumstances, mutation (which changes 0 to 1 and vice-versa) can push a string from the local basin into the global basin (refer to Fig. 3.5) by changing the left-most bit from 0 into 1. Thus, mutation helps the GA to search the globally optimal solution.

To avoid a random search, mutation probability (p_m) is generally kept to a low value. It is varied generally in the range of $\frac{0.1}{L}$ to $\frac{1}{L}$, where L is the string length. To implement a bit-wise mutation scheme, a number lying between 0.0 and 1.0 is created using a random number generator at each bit-position. If this number comes out to be less than or equal to p_m at a particular bit-position, then that bit will be mutated (that is, 1 will be converted into 0 and vice-versa). It has been observed that the performance of a GA can be improved using an adaptive mutation rate [31].

In short, reproduction selects good strings from the population to form a mating pool and in crossover, there is an exchange of properties between two parent strings and consequently, two children strings are created. Mutation brings a local change at a particular bit position of the string. Once the population of strings is modified using the operators, namely reproduction, crossover and mutation, one generation of a GA is completed. The GA runs until

the termination criterion (that is, maximum number of generations or a desired precision in the solution) is reached.

3.2.1 Crossover or Mutation ?

The performance of a genetic/evolutionary algorithm depends significantly on its operators, namely crossover, mutation. Now, a question may arise – which one out of these two operators is more powerful ? A considerable amount of study has been made to search the relative importance of these two operators. However, till now, it has not been proved theoretically that crossover is more powerful than mutation and vice-versa. Spears [32] opined that a genetic operator has to perform two potential roles, such as **disruption** and **construction**. It has been pointed out that mutation is more powerful than crossover in terms of its disruption capability, whereas crossover outperforms mutation in terms of the construction capability. Due to this reason, Genetic Algorithm (GA) and Genetic Programming (GP) rely more on the crossover, whereas in Evolution Strategies (ES) and Evolutionary Programming (EP), a more weightage is given on the mutation.

3.2.2 A Hand Calculation

Table 3.1 shows a hand calculation to solve an optimization problem given below and explain the working principle of a binary-coded GA. Let us try to examine whether the GA can improve the solution from one generation to the next one.

$$\text{Maximize } y = \sqrt{x} \quad (3.5)$$

subject to

$$1.0 \leq x \leq 16.0.$$

The initial population of binary-strings (of size 6) are created at random, which are modified using the proportionate selection (Roulette-Wheel selection), a single-point crossover ($p_c = 1.0$) and a bit-wise mutation ($p_m = 0.03$). The decoded values of the binary strings are determined and depending on the range of the variable, its real values are calculated corresponding to different binary-strings. Once the values of the variables are known, the function values can be determined. As it is a maximization problem, the fitness values of the GA-strings are nothing but the function values. Using the information of fitness values of different strings, their probabilities of being selected in the mating pool are calculated. The mating pool has been generated then using the principle of proportionate selection scheme. The mating pairs are identified and all the pairs participate in the single-point crossover, as the probability of crossover p_c has been assumed to be equal to 1.0. The children solutions created due to this crossover are shown in Table 3.1. As there are $6 \times 6 = 36$ bits in the population and $p_m = 0.03$, there is a chance that only one bit will be mutated. Let us suppose that the second bit (from left) of fourth string of the population created due to crossover, has been mutated (that is, 0 is changed into 1). Table 3.1 shows that

Table 3.1: A hand calculation to explain the working principle of a binary-coded GA

String No.	Initial population	Decoded value	x value	$f(x) = \sqrt{x}$	$p_{selection} = \frac{f_i}{\sum f}$	Expected count $\frac{f_i}{\bar{f}}$
1	100101	37	9.81	3.13	0.18	1.07
2	011010	26	7.19	2.68	0.15	0.91
3	010110	22	6.24	2.50	0.14	0.85
4	111010	58	14.81	3.85	0.22	1.31
5	101100	44	11.48	3.39	0.19	1.16
6	001101	13	4.09	2.02	0.12	0.69
				sum $\sum f = 17.57$ average $\bar{f} = 2.93$ maximum $f = 3.85$		

Actual count Roulette wheel	Mating pool	Mating pair	Parents	Crossover site	Children strings	Mutation
1	100101	3	100101	10 0101	101010	101010
1	011010	6	111010	11 1010	110101	110101
0	111010	1	011010	011 010	011100	011100
2	111010	5	101100	101 100	101010	111010
2	101100	4	111010	11 1010	111100	111100
0	101100	2	101100	10 1100	101010	101010

Decoded value	x value	$f(x) = \sqrt{x}$
42	11.00	3.32
53	13.62	3.69
28	7.67	2.77
58	14.81	3.85
60	15.28	3.91
42	11.00	3.32
		sum $\sum f = 20.86$ average $\bar{f} = 3.48$ maximum $f = 3.91$

the population average has increased from 2.93 to 3.48 and the maximum fitness value has improved from 3.85 to 3.91, in one generation of the GA-run. Thus, it indicates that the GA can improve the solutions.

3.2.3 Fundamental Theorem of GA/Schema Theorem

An attempt is made through this theorem to investigate the mathematical foundation of a GA. A schema (plural form is schemata) is a template, which may be present in the population of binary strings created at random [11]. We try to predict the growth and decay of a schema or the schemata through the generations.

Let us suppose that the population of binary-strings created at random, is represented as follows:

$$\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

If we look into this population of binary-strings carefully, we can find some similarities (in terms of presence of 1 and 0 at different bit-positions) among a number of strings. Let us consider that the following two schemata (templates) are present in the above population of strings.

$$\begin{array}{lcl} H_1 & : & \star \ 1 \ 0 \ \star \ \star \ \star \\ H_2 & : & \star \ 1 \ 0 \ 0 \ 0 \ 0 \end{array}$$

where \star could be either 1 or 0.

To proceed with the analysis of schema processing, we take the help of two terms, namely **order** $O(H)$ and **defining length** $\delta(H)$ of a schema, which are defined below.

- **Order of a schema H , that is, $O(H)$:** It is defined as the number of fixed positions (bits) present in a schema. For example, $O(H_1) = 2$ and $O(H_2) = 5$.
- **Defining length of a schema H , that is, $\delta(H)$:** It is defined as the distance between the first and last fixed positions in a string. For example, $\delta(H_1) = 3 - 2 = 1$ and $\delta(H_2) = 6 - 2 = 4$.

Let us assume that this population of strings is modified using the operators, namely reproduction, crossover and mutation. The effect of these operators on the growth and decay of a schema is explained below.

1. **Reproduction:** Let us concentrate on a proportionate selection (Roulette-Wheel selection) scheme, in which the probability of a string for being copied into the mating pool is proportional to its fitness value. Thus, the number of strings belonging to a particular schema H at $(t + 1) - th$ generation can be determined using the equation given below.

$$m(H, t + 1) = m(H, t) N \frac{f(H)}{\sum f}, \quad (3.6)$$

where $m(H, t + 1)$ and $m(H, t)$ represent the number of strings belong to the schema H at $(t + 1) - th$ and $t - th$ generations, respectively; N indicates the population size; $f(H)$ represents the average fitness of the strings represented by the schema H at $t - th$ generation, which may also be called as average schema fitness; $\sum f$ indicates the total fitness of the strings present in the population.

The above equation can also be written in the following form:

$$m(H, t + 1) = m(H, t) \frac{f(H)}{\bar{f}}, \quad (3.7)$$

where $\bar{f} = \frac{\sum f}{N}$ is the average fitness of the population.

2. **Crossover:** Let us consider a single-point crossover of the binary-strings. A schema survives, when the randomly selected crossover site falls outside the defining length $\delta(H)$ of the schema. For this crossover, the probability of destruction turns out to be equal to $p_c \frac{\delta(H)}{L-1}$, where p_c and L are the crossover probability and string length, respectively. Thus, the lower bound of crossover survival probability p_s can be calculated like the following:

$$p_s \geq 1 - p_c \frac{\delta(H)}{L-1}. \quad (3.8)$$

Now, combining the effect of reproduction and crossover, we get the schema processing equation as given below.

$$m(H, t + 1) \geq m(H, t) \frac{f(H)}{\bar{f}} [1 - p_c \frac{\delta(H)}{L-1}]. \quad (3.9)$$

3. **Mutation:** Let us consider a bit-wise mutation of the binary strings. To ensure the survival of a schema H , all the fixed bits must survive. In other words, mutation should not occur at the fixed bits to protect the schema.

Let us suppose that the probability of mutation/destruction for each bit is denoted by p_m . Thus, $(1 - p_m)$ represents the probability of survival for each bit. Similarly, the probability of survival considering all the fixed positions/bits in the schema will be given by the following expression:

$$\begin{aligned} p_s &= (1 - p_m)(1 - p_m) \dots O(H) \\ &= (1 - p_m)^{O(H)} \end{aligned}$$

As $p_m \ll 1$, p_s can approximately be set equal to $(1 - O(H)p_m)$.

Considering the contributions of all three operators, namely reproduction, crossover and mutation, the schema processing formula can be written as follows:

$$m(H, t + 1) \geq m(H, t) \frac{f(H)}{\bar{f}} [1 - p_c \frac{\delta(H)}{L-1} - O(H)p_m]. \quad (3.10)$$

From the above equation, it is interesting to observe that the schemata having low order, short defining length and whose fitness values are more than the average fitness of the population, will receive more and more copies in future generations. They are called the building blocks of GA-search. This theorem is also known as the **Building-Block Hypothesis** of a GA.

3.2.4 Limitations of a Binary-Coded GA

Binary-coded GA is the oldest of all versions of the GA. However, it has the following limitations:

1. If we need more precision in the values of the variables, we will have to assign more number of bits to represent them. To ensure a proper search in a such situation, population size is to be kept to a high value and as a result of which, computational complexity of the GA increases. Thus, a binary-coded GA may not be able to yield any arbitrary precision in the solution. This problem can be overcome using a real-coded GA, the principle of which has been explained in the next chapter.
2. In binary representation, if we want to move from one number to the next or previous number, the number of changes to be made in the bit-position may not be the same. For example, the numbers 14, 15 and 16 are represented by 01110, 01111 and 10000, respectively and thus, five bits are to be changed to shift from 15 to 16, whereas it requires a change of only one-bit to move from 15 to 14. This problem in binary-coding is known as **hamming cliff** problem, which may create an artificial hindrance to the gradual search of a GA. This problem can be eliminated using a gray-coding scheme, in place of a binary-coding scheme.

In a gray-coding system, the above shifting of numbers is possible by making only one change in the bit-position. Table 3.2 shows both the binary and gray-codings of the numbers starting from 0 to 15.

3.3 GA-parameters Setting

The performance of a genetic-search depends on the amount of **exploration** (**population diversity**) and **exploitation** (**selection pressure**). To have an effective search, there must be a proper balance between them and to ensure this, the GA-parameters, such as population size, crossover probability p_c and mutation probability p_m are to be selected in the optimal sense. The interactions among themselves are to be studied to select their optimal values. It is obvious that the optimal GA-parameters are problem-dependent. Several trials have been made to understand the mechanics of interactions of GA-parameters using empirical studies, Markov chain analysis, statistical analysis based on Design of Experiments (DOE), and others. In this section, a method of determining the near-optimal GA-parameters has been discussed.

Table 3.2: Binary and gray-codings of the numbers starting from 0 to 15

Number	Binary-code	Gray-code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Let us suppose that we have decided to vary the GA-parameters, such as crossover probability p_c , mutation probability p_m and population size N in the ranges of (0.6 to 1.0), (0.001 to 0.011) and (50 to 150), respectively, while solving a particular minimization problem. Moreover, the maximum number of generations G_{max} will be varied in the range of (50 to 150), say. This experiment is conducted in four stages as discussed below (refer to Fig. 3.6).

- **Stage 1:** We generally vary p_c at first, within its range, after keeping other parameters fixed at their respective mid-values. Thus, p_c is varied from 0.6 to 1.0 in steps of 0.1 say, and p_m , N and G_{max} are kept fixed to 0.006, 100 and 100, respectively. Let us suppose that the fitness value is found to be the minimum for a particular value of p_c , say 0.9.
- **Stage 2:** We fix the values of p_c , N and G to 0.9, 100 and 100, respectively and vary p_m starting from 0.001 to 0.011 in steps of 0.001. Let us assume that the fitness value is seen to be the minimum for $p_m = 0.002$.
- **Stage 3:** The values of p_c , p_m and G_{max} are kept fixed to 0.9, 0.002 and 100, respectively and the population size is varied from 50 to 150 in steps of 10. Let us suppose that the fitness is found to be the minimum for a population size of 110.
- **Stage 4:** In this stage, p_c , p_m and N are set equal to 0.9, 0.002 and 110, respectively and experiments are carried out after setting the number of maximum generations at

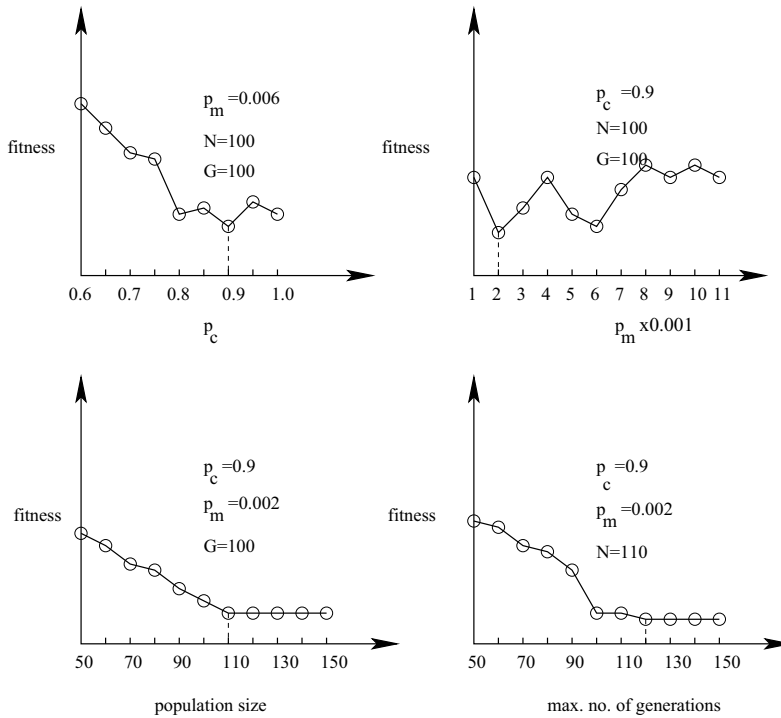


Figure 3.6: Results of the parametric study conducted for determining the optimal GA-parameters.

different values (starting from 50 up to 150 in steps of 10). Let us assume that the minimum fitness is obtained, when the GA is run for a maximum of 120 generations.

Thus, the optimal GA-parameters are found to be as follows: $p_c = 0.9$, $p_m = 0.002$, $N = 110$ and $G_{max} = 120$.

It is important to mention that the above method may not be able to determine the set of true optimal GA-parameters. It is an easier way of finding the suitable GA-parameters approximately. Let us assume that the range of each GA-parameter is divided into 9 equal divisions. Thus, only 10 different values are considered for each GA-parameter, while carrying out the parametric study. It is important to note that in the above method, experiments are carried out only with 40 (that is, $10 + 10 + 10 + 10 = 40$) sets of GA-parameters, out of a maximum of 10^4 possible sets. Thus, there is no guarantee that the above method will be able to yield the globally-optimal GA-parameters.

3.4 Constraints Handling in GA

Several attempts have been made by various investigators to solve the constrained optimization problems using a GA, in which the fitness function of GA-solution has been expressed in a number of ways to tackle the constraints after ensuring its proper search. It is to be

noted that the constraints may be of different natures, such as linear, non-linear, equality, inequality, and others, which basically divide the search space into feasible and infeasible regions.

A constrained optimization (either maximization or minimization) problem may be expressed as follows:

$$\text{Optimize } f(\mathbf{X}) \quad (3.11)$$

subject to

$$g_i(\mathbf{X}) \leq 0, i = 1, 2, \dots, n, \quad (3.12)$$

$$h_j(\mathbf{X}) = 0, j = 1, 2, \dots, p, \quad (3.13)$$

where n and p represent the number of inequality and equality constraints, respectively, $\mathbf{X} = (x_1, x_2, \dots, x_m)^T$ are the design variables having the bounds given below.

$$\begin{aligned} x_1^{\min} &\leq x_1 \leq x_1^{\max}, \\ x_2^{\min} &\leq x_2 \leq x_2^{\max}, \\ &\vdots \\ x_m^{\min} &\leq x_m \leq x_m^{\max}. \end{aligned}$$

It is important to mention that the above constraints could be either linear or non-linear in nature. Let us use the notation $\phi_k(\mathbf{X})$, $k = 1, 2, \dots, q$, to denote both the linear and non-linear constraints together. Thus, q becomes equal to $(n + p)$.

A number of constraint handling techniques have been proposed, such as **penalty function approach**, **method of maintaining a feasible population over infeasible solutions**, **approach aiming at preserving feasibility of solutions**, **approach separating the objectives and constraints** [33, 34, 35], and others. Out of all these techniques, the penalty function approach is the most popular one, which has been discussed below. A detailed discussion on all the above approaches is beyond the scope of this book. Interested readers may refer to [33, 34, 35] for the above purpose.

3.4.1 Penalty Function Approach

In this approach, the fitness function of a solution (say i -th) is expressed by modifying its objective function as follows:

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) \pm P_i, \quad (3.14)$$

where P_i indicates the penalty used to penalize an infeasible solution. For a feasible solution, P_i is set equal to 0.0, whereas for an infeasible solution P_i is expressed like the following:

$$P_i = C \sum_{k=1}^q \{\phi_{ik}(\mathbf{X})\}^2, \quad (3.15)$$

where C indicates the user-defined penalty coefficient and $\{\phi_{ik}(\mathbf{X})\}^2$ represents the penalty term for k -th constraint, corresponding to i -th objective function. It is important to mention

that the above penalty could be either static or dynamic or adaptive in nature, which are explained below.

Static Penalty [36]:

In this approach, amount of constraint violation is divided into several levels and penalty coefficient $C_{k,r}$ (for r -th level of violation of k -th constraint, where $r = 1, 2, \dots, v$; v indicates the user-defined level of violation) is pre-defined for each level. Thus, the fitness function for i -th solution can be represented like the following:

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) + \sum_{k=1}^q C_{k,r} \{\phi_{ik}(\mathbf{X})\}^2 \quad (3.16)$$

The main drawback of this approach lies in the fact that its performance is dependent on a number of user-defined parameters, which may be difficult to determine beforehand.

Dynamic Penalty [37]:

In this method, amount of penalty varies with the number of generations. The fitness of a GA-solution is calculated as given below.

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) + (C \times t)^\alpha \sum_{k=1}^q |\phi_{ik}(\mathbf{X})|^\beta, \quad (3.17)$$

where C , α , β are the user-defined constants, t represents the number of generations. The penalty term is dynamic in nature, as it increases with the number of generations.

The algorithm with dynamic penalties is found to perform better than that using static penalties. However, its performance depends on the pre-defined values of C , α and β , the selection of which may be difficult to do beforehand.

Adaptive Penalty [38, 39]:

In this approach, the penalty terms are updated by taking feed-back during the search. The concept of adaptive penalty has been developed in a number of ways by various researchers [35]. Bean and Hadj-Alouane's approach [38, 39] is one of such trials, in which the fitness of a solution is expressed as follows:

$$F_i(\mathbf{X}) = f_i(\mathbf{X}) + \lambda(t) \sum_{k=1}^q \{\phi_{ik}(\mathbf{X})\}^2, \quad (3.18)$$

where t indicates the number of generation. The parameter $\lambda(t)$ is updated like the following:

$$\lambda(t+1) = \begin{cases} (\frac{1}{\beta_1}) \times \lambda(t), & \text{for Case 1} \\ \beta_2 \times \lambda(t), & \text{for Case 2} \\ \lambda(t), & \text{Otherwise,} \end{cases} \quad (3.19)$$

where $\beta_1 \neq \beta_2$ and $\beta_1, \beta_2 > 1$, Case 1 represents the situations, where the previously found best individuals always remain feasible and Case 2 indicates those situations, in which the obtained best individuals are always seen to be infeasible. It is important to mention that its performance depends on the selection of β_1 and β_2 values, which may be difficult to pre-determine.

3.5 Advantages and Disadvantages of Genetic Algorithms

Genetic Algorithm has a number of advantages over the traditional optimization tools but it has some disadvantages too, which are discussed below.

Advantages of GA:

It is important to mention that a GA can overcome almost all the limitations of traditional optimization tools. It has the following advantages:

1. A GA starts with a population of initial solutions created at random. Out of all these initial solutions, if at least one solution falls in the global basin, there is a good chance for the GA to reach the globally optimal solution. It is important to mention that initial population may not contain any solution lying in the global basin. In that case, if the GA-operators (namely crossover, mutation) can push at least one solution into the global basin, there is a chance that the GA will find the globally optimal solution. Thus, the chance of its solutions for being trapped into the local minima is less.
2. It can handle the integer programming and mixed-integer programming problems effectively.
3. As gradient information of the objective function is not required by a GA, it can optimize discontinuous objective functions also.
4. It is suitable for parallel implementations.
5. The same GA with a little bit of modification in the string can solve a variety of problems. Thus, it is a versatile optimization tool.

Disadvantages of GA:

Although the GA has a number of advantages, it has the following disadvantages:

1. It is computationally expensive and consequently, has a slow convergence rate.
2. It works like a black-box optimization tool.
3. There is no mathematical convergence proof of the GA, till today.
4. A user must have a proper knowledge of how to select an appropriate set of GA-parameters.

3.6 Combination of Local and Global Optimum Search Algorithms

The aim is to develop an efficient optimization algorithm, which can determine the global optimum solution as quickly as possible even for a complex problem. A GA is a powerful tool for global optimization but it may not be equally efficient in carrying out the local search. Moreover, it may take a considerable amount of time to reach the optimal solution, as it is computationally expensive. On the other hand, the search speed of a gradient-based optimization algorithm is high and it is a potential tool for carrying out the local search. Therefore, the global search power of the GA can be combined with the local search capability of the gradient-based tool in order to develop an efficient optimization algorithm. In order to determine the optimal solution for a complex optimization problem, a GA can be used initially to conduct a global search, and then a gradient-based optimization tool may be employed to further carry out the search in a limited space specified by the GA.

3.7 Summary

This chapter has been summarized as follows:

1. The working cycle of a GA has been explained. In a GA, the solutions are modified using different operators, such as reproduction, crossover, mutation and others. Reproduction selects good strings from a population and copies them in the mating pool. In crossover, there is an exchange of properties between the parents and as a result of which, the children solutions are created. Mutation brings a local change in the current solution and thus, helps the solution to come out of the local minimum, if any. Thus, the chance of its solutions for getting stuck at the local minima is less.
2. To ensure an effective search of the GA, there must be a proper balance between its population diversity (exploration) and selection pressure (exploitation). An appropriate set of the parameters, such as crossover probability, mutation probability, population size, maximum number of generations, is to be determined through a careful study for the above purpose.
3. Schema theorem of the binary-coded GA is able to explain the reason behind the fact that it can improve the solutions from one generation to the next.
4. A binary-coded GA cannot obtain any arbitrary precision required. It is so, because a large number of bits are to be assigned to represent a variable for obtaining the better precision. As the number of bits in the GA-string increases, its computational complexity will be more and as a result of which, the GA will become slower.
5. A binary-coded GA suffers from the Hamming Cliff problem, which can be eliminated using a gray-coded GA.

6. A GA can overcome almost all drawbacks of the traditional methods of optimization but it is found to be computationally expensive.
7. An introduction is given to penalty function approach of constraint handling used along with the GA. The penalty term may be either static or dynamic or adaptive in nature.

3.8 Exercise

1. Explain briefly the role of different operators, namely reproduction, crossover and mutation used in a GA.
2. The performance of a GA depends on the balance between selection pressure and population diversity – justify the statement.
3. Is GA a random search technique? Explain it.
4. Are uniform crossover with probability of 0.5 and bit-wise mutation with probability of 0.5 the same ? Explain it.
5. Explain elitist strategy used in GA.
6. Why do we prefer ranking selection to a Roulette-Wheel selection in GA ?
7. Why do we use a high value of crossover probability and a low value of mutation probability in GA-applications ?
8. Do you prefer a Gray-coded GA to a Binary-coded GA ? Explain it.
9. Explain the Building-Block Hypothesis of a binary-coded GA.
10. Can you declare GA a global optimizer ? Explain it.
11. State the advantages and disadvantages of a GA.
12. A binary-coded GA is to be used to solve an optimization problem involving one real and another integer variables. The real and integer variables are allowed to vary in the ranges of (0.2, 10.44) and (0, 63), respectively. Design a suitable GA-string to ensure a precision level of 0.01 for the real variable.
13. Use a binary-coded GA to minimize the function $f(x_1, x_2) = x_1 + x_2 - 2x_1^2 - x_2^2 + x_1x_2$, in the range of $0.0 \leq x_1, x_2 \leq 5.0$. Use a random population of size $N = 6$, tournament selection, a single-point crossover with probability $p_c = 1.0$ and neglect mutation. Assume 3 bits for each variable and thus, the GA-string will be 6-bits long. Show only one iteration by hand calculations.

14. An initial population of size $N = 10$ of a binary-coded GA is created at random as shown below, while maximizing a function.

1	0	0	1	0	0
0	1	0	1	0	0
0	1	0	1	1	1
1	1	1	0	0	0
0	1	0	1	1	0
1	0	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1
0	1	1	0	0	1

The fitness of a GA-string is assumed to be equal to its decoded value. Calculate the expected number of strings to be represented by the schema $H : * 1 * * 1 *$, at the end of first generation, considering a single-point crossover of probability $p_c = 0.9$ and a bit-wise mutation of probability $p_m = 0.01$. Make comments on the result.

15. A closed-coil helical spring (refer to Fig. 3.7) made up of a wire of circular cross-section is to be designed, so that it weighs as minimum as possible. Although the load is applied along the axis or parallel to the axis of the spring, shear stress is produced in it due to twisting and its value has to be less than the specified value S . The volume of the spring can be determined using the following expression:

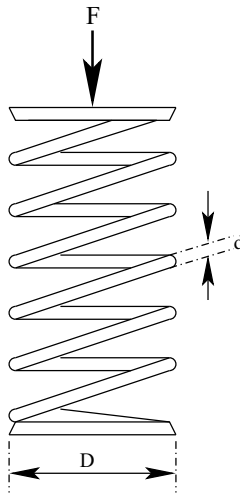


Figure 3.7: A closed-coil helical spring.

$$V = \frac{\pi^2}{4}(N_c + 2)Dd^2,$$

where d indicates the diameter of the wire, D represents the coil diameter, N_c denotes the number of active turns. The developed shear stress in the spring can be calculated using the expression given below.

$$\tau_{developed} = \frac{8C_f F D}{\pi d^3},$$

where $C_f = \frac{4C-1}{4C-4} + \frac{0.615}{C}$ and $C = \frac{D}{d}$; F represents the maximum working load. The ranges of different variables are kept as follows:

$$\begin{aligned} 0.5 &\leq d \leq 0.8, \\ 1.5 &\leq D \leq 8.0, \\ 16 &\leq N_c \leq 31. \end{aligned}$$

Assume that the density of spring material is represented by ρ .

- Formulate it as a constrained optimization problem.
- Use a binary-coded GA to solve the constrained (after assuming a suitable value of S) and corresponding un-constrained optimization problems. The real variables are assumed to have the accuracy level of 0.001 (approx).
Develop a computer program of the binary-coded GA and solve the above problem using different forms of reproduction scheme (such as proportionate selection, ranking selection, tournament selection), crossover operator (namely single-point crossover, two-point crossover, uniform crossover) and a bit-wise mutation.
(Hints: To determine the length of GA-substring (l) required for representing a variable x after ensuring a precision level of ϵ , follow the expression given below.

$$\frac{x^{max}-x^{min}}{\epsilon} = 2^l .)$$

16. Let us consider a constrained optimization problem of two variables: x_1 and x_2 as given below.

$$\text{Minimize } f(x_1, x_2) = x_1 + x_2 - 2x_1^2 - x_2^2 + x_1x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &< 9.5, \\ x_1^2 + x_2^2 - x_1x_2 &> 15.0 \end{aligned}$$

and

$$0.0 \leq x_1, x_2 \leq 5.0.$$

The above constrained optimization problem is to be solved by a GA using the concepts of static, dynamic and adaptive penalties. Assume suitable values for the constant terms. Show one set of calculations for the values of $x_1 = 2.0$ and $x_2 = 3.5$.

(**Hints:** To implement static and dynamic penalties, put the penalty value equal to zero, if a functional constraint is satisfied).

17. A shaft of circular cross-section of diameter d m and length L' m supported at two ends, is subjected to both a concentrated load of P N acting at its mid-span and torque of T Nm, as shown in Fig. 3.8. The allowable shear stress of shaft mate-

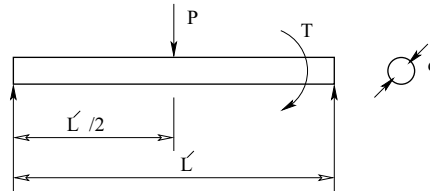


Figure 3.8: A schematic view showing combined loading of a shaft supported at its two ends.

rial is denoted by $\tau_{allowable}$. The shaft should be as light as possible. However, it should be able to withstand the combined loading. Assume density of shaft material $\rho = 7.8 \times 10^3 \text{ kg/m}^3$; $P = 200 \text{ N}$; $T = 400 \text{ Nm}$; $\tau_{allowable} = 11 \text{ MPa}$. Formulate it as a constrained optimization problem, which is to be solved using a penalty function approach and a binary-coded GA. Show a suitable representation scheme of the variables. How do you calculate the fitness of a GA-string? Assume suitable values for the constant terms of penalty function approach. Take the ranges of the variables as given below.

$$0.05 \leq d \leq 0.15$$

$$0.5 \leq L' \leq 2.5.$$

(**Hints:** For this combined loading, equivalent torque $T_e = \sqrt{M^2 + T^2}$, where the maximum bending moment $M = \frac{PL'}{4}$ and shear stress $\tau = \frac{16T_e}{\pi d^3}$.)

18. In straight turning carried out on a Lathe by using a single-point tool, it is subjected to a cutting force of magnitude P . Fig. 3.9 shows the schematic view of a single-point cutting tool. Let us assume an orthogonal turning, in which the main component of the cutting force is denoted by P_z (neglect the other two components of the cutting force: P_x and P_y). Design an optimal cross-section of the cutting tool, so that it becomes as light in weight as possible after ensuring the condition of no mechanical breakage. Assume allowable stress of the tool material $\sigma_{allowable} = 170 \text{ MPa}$, density of the tool material $\rho = 7860 \text{ kg/m}^3$. Carry out optimization for a known value of $P_z = 250 \text{ N}$. Take a fixed length of the cutting tool $L' = 0.20 \text{ m}$. The design variables are allowed to vary in the ranges given below.

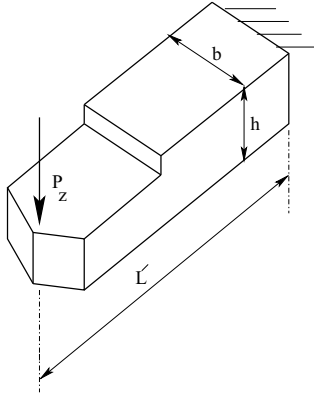


Figure 3.9: A single-point cutting tool.

$$\begin{aligned} 0.015 &\leq b \leq 0.07 \text{ m}, \\ 0.01 &\leq h \leq 0.06 \text{ m} \end{aligned}$$

- Formulate it as a constrained optimization problem.
- Solve the constrained optimization problem using a binary-coded GA. Use the concept of static penalty. Show hand calculations for one generation of the GA run after assuming population size $N = 8$, $p_c = 1.0$, $p_m = 0.01$. Use 10 bits to represent each variable.

(**Hints:** Maximum bending moment $M = P_z L'$, as it can be assumed a cantilever beam. The developed stress due to this maximum bending moment is give by the expression: $\sigma = \frac{M}{I} y$, where I is the moment of inertia, and for a rectangular cross-section (of width b and hieght h), it is calculated as $I = \frac{1}{12} b h^3$; $y = \frac{h}{2}$.)

19. A beam of rectangular cross-section of height h m and width b m, and length $L' = 1.5$ m supported at two ends, is subjected to two equal concentrated loads of magnitude $P = 500$ N each, as shown in Fig. 3.10. The beam should be as light as possible.

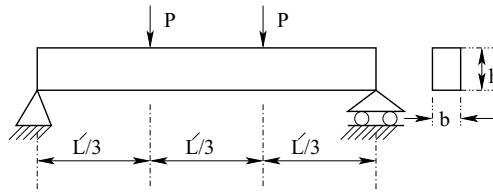


Figure 3.10: A rectangular crossection beam supported at two ends.

However, it should be able to withstand the above loading. Assume, the density of beam material $\rho = 7.6 \times 10^3 \text{ kg/m}^3$ and allowable bending stress $\sigma = 32000 \text{ N/m}^2$. Formulate it as a constrained optimization problem, which is to be solved using the

concept of dynamic penalty (for which assume $C = 10$, $\alpha = 1.5$, $\beta = 2$, the symbols carry usual meaning of dynamic penalty approach) in a binary-coded GA. The variables are allowed to vary in the ranges given below.

$$0.2 \leq b \leq 0.5 \text{ } m$$

$$0.4 \leq h \leq 0.7 \text{ } m$$

Show suitable representation of the variables to ensure the precision level of 0.001 m (approximately) each. Use a random population of the size $N = 4$, tournament selection, a single-point crossover with probability $p_c = 1.0$ and neglect mutation. Show only one iteration of the GA through hand calculations.

(Hints: Maximum bending moment $M = PL'/3$).

Chapter 4

Some Specialized Genetic Algorithms

To ensure good precision in the values of real variables and overcome Hamming Cliff problem, a binary-coded GA may be replaced by a **real-coded GA**, the principle of which has been explained in this chapter. A GA is found to be computationally expensive and it may not be suitable for on-line implementations. Several attempts have been made to develop some faster GAs, namely **micro-GA**, **Visualized Interactive GA (VIGA)**, which are discussed in the present chapter. Scheduling is a special type of optimization problem, in which not only the positions of the elements but also their order and adjacency are important. Thus, the conventional crossover operators discussed in the previous chapter may not be able to tackle this problem efficiently. A **Scheduling GA** requires a special type of crossover operator. The principles of some of these specialized crossover operators are explained in the present chapter.

4.1 Real-Coded GA

A real-coded GA can overcome the difficulties faced by a binary-coded GA (refer to Section 3.2.4) to carry out optimization in a continuous search space. For the real-coded GA, several versions of crossover and mutation operators have been proposed by various researchers. The principles of some of these operators are explained below.

4.1.1 Crossover Operators

A large number of crossover operators, namely **Linear Crossover** [40], **Blend Crossover** [18], **Simulated Binary Crossover** [41], and others, have been developed for the real-coded GA.

Linear Crossover:

It was proposed by Wright in 1991 [40]. To explain its principle, let us consider that two parents: Pr_1 and Pr_2 are participating in crossover. They produce three solutions as follows: $0.5(Pr_1 + Pr_2)$, $(1.5Pr_1 - 0.5Pr_2)$, $(-0.5Pr_1 + 1.5Pr_2)$. Out of these three solutions, the best two are selected as the children solutions.

Example:

Let us assume that the parents are: $Pr_1 = 15.65$, $Pr_2 = 18.83$.

Using the linear crossover operator, three solutions are found to be like the following:

$$\begin{aligned} 0.5(15.65 + 18.83) &= 17.24, \\ 1.5 \times 15.65 - 0.5 \times 18.83 &= 14.06, \\ -0.5 \times 15.65 + 1.5 \times 18.83 &= 20.42. \end{aligned}$$

The parents and obtained solutions are shown in Fig. 4.1.

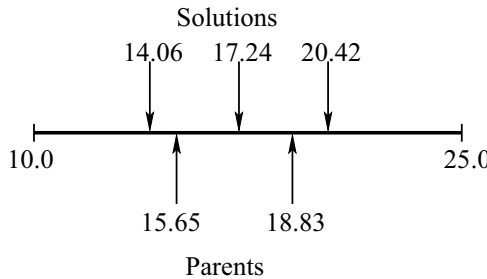


Figure 4.1: The parents and their solutions obtained using linear crossover.

Blend Crossover (BLX - α):

This operator was developed by Eshelman and Schaffer in 1993 [18]. Let us consider two parents: Pr_1 and Pr_2 , such that $Pr_1 < Pr_2$. It creates the children solutions lying in the range of $[\{Pr_1 - \alpha(Pr_2 - Pr_1)\}, \{Pr_2 + \alpha(Pr_2 - Pr_1)\}]$, where the constant α is to be selected, so that the children solutions do not come out of the range. Another parameter γ has been defined by utilizing the said α and a random number r varying in the range of (0.0, 1.0) like the following:

$$\gamma = (1 + 2\alpha)r - \alpha.$$

The children solutions (Ch_1 , Ch_2) are determined from the parents as follows:

$$\begin{aligned} Ch_1 &= (1 - \gamma)Pr_1 + \gamma Pr_2, \\ Ch_2 &= (1 - \gamma)Pr_2 + \gamma Pr_1. \end{aligned}$$

Example:

Let us assume that the parents are: $Pr_1 = 15.65$, $Pr_2 = 18.83$.

Assume: $\alpha = 0.5$, $r = 0.6$.

The parameter γ is calculated like the following:

$$\gamma = (1 + 2\alpha)r - \alpha = (1 + 2 \times 0.5)0.6 - 0.5 = 0.7$$

The children solutions are then determined as follows:

$$\begin{aligned} Ch_1 &= (1 - \gamma)Pr_1 + \gamma Pr_2 = 17.876 \\ Ch_2 &= (1 - \gamma)Pr_2 + \gamma Pr_1 = 16.604 \end{aligned}$$

The parents and obtained children are shown in Fig. 4.2.

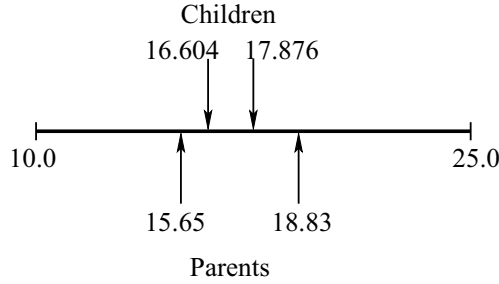


Figure 4.2: The parents and their children obtained using blend crossover.

Simulated Binary Crossover (SBX) [41, 34]:

It was proposed by Deb and Agrawal in 1995 [41]. Its search power is represented with the help of a probability distribution of generated children solutions from the given parents. A **spread factor** α has been introduced to represent the spread of the children solutions with respect to that of the parents, as given below.

$$\alpha = \left| \frac{Ch_1 - Ch_2}{Pr_1 - Pr_2} \right|, \quad (4.1)$$

where Pr_1, Pr_2 represent the parent points and Ch_1, Ch_2 are the children solutions. Three different cases may occur:

- **Case 1:** Contracting crossover ($\alpha < 1$)—the spread of the children solutions is less than that of the parents.
- **Case 2:** Expanding crossover ($\alpha > 1$)—the spread of the children solutions is more than that of the parents.
- **Case 3:** Stationary crossover ($\alpha = 1$)—the spread of the children solutions is exactly the same with that of the parents.

In Deb and Agrawal [41], the probability distributions for creating children solutions from the parents have been assumed to be polynomial in nature (refer to Fig. 4.3): It is important to mention that probability distributions depend on the exponent q , which is a non-negative

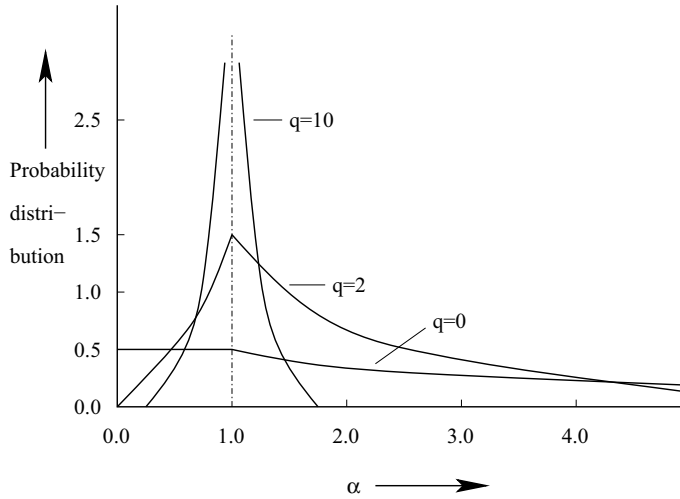


Figure 4.3: Probability distributions for creating the children solutions from the parents vs. spread factor α [41].

real number. For the contracting crossover, the probability distribution is given by the following expression:

$$C(\alpha) = 0.5(q+1)\alpha^q. \quad (4.2)$$

On the other hand, for the expanding crossover, it is expressed as follows:

$$Ex(\alpha) = 0.5(q+1)\frac{1}{\alpha^{(q+2)}}. \quad (4.3)$$

It is also interesting to note that for small values of q , the children are generally far away from the parents, whereas for high values of q , they are created in the close neighborhood of the parents. Fig. 4.3 shows the variations of probability distributions for different values of q (say 0, 2 and 10). It is also important to state that the area under the probability distribution curve in the contracting crossover zone (that is, $\int_{\alpha=0}^1 C(\alpha)d\alpha$) and that in the expanding crossover zone (that is, $\int_{\alpha=1}^{\infty} Ex(\alpha)d\alpha$) are found to be equal to 0.5 each.

The following steps are used to create two children solutions: Ch_1 and Ch_2 from the parents: Pr_1 and Pr_2 :

- **Step 1:** Create a random number r lying between 0.0 and 1.0.
- **Step 2:** Determine α' for which the cumulative probability

$$\int_0^{\alpha'} C(\alpha)d\alpha = r, \text{ if } r < 0.5.$$

and

$$\int_1^{\alpha'} Ex(\alpha)d\alpha = r - 0.5, \text{ if } r > 0.5.$$

- **Step 3:** Knowing the value of α' , the children solutions are determined like the following:

$$\begin{aligned} Ch_1 &= 0.5[(Pr_1 + Pr_2) - \alpha' |Pr_2 - Pr_1|], \\ Ch_2 &= 0.5[(Pr_1 + Pr_2) + \alpha' |Pr_2 - Pr_1|]. \end{aligned}$$

Example:

Let us assume that the parents are: $Pr_1 = 15.65$, $Pr_2 = 18.83$. Determine children solutions using the SBX. Assume exponent $q = 2$.

Let us consider that the generated random number r is equal to 0.6. As $r > 0.5$, we calculate α' , such that

$$\begin{aligned} \int_1^{\alpha'} Ex(\alpha) d\alpha &= r - 0.5, \\ \text{i.e., } \int_1^{\alpha'} 0.5(q+1) \frac{1}{\alpha^{q+2}} d\alpha &= 0.1. \end{aligned}$$

After solving the above equation, we get $\alpha' = 1.0772$. Thus, the children solutions are found to be as follows:

$$\begin{aligned} Ch_1 &= 0.5[(Pr_1 + Pr_2) - \alpha' |Pr_2 - Pr_1|] = 15.5273 \\ Ch_2 &= 0.5[(Pr_1 + Pr_2) + \alpha' |Pr_2 - Pr_1|] = 18.9527 \end{aligned}$$

The parents and their children obtained using the simulated binary crossover are shown in Fig. 4.4.

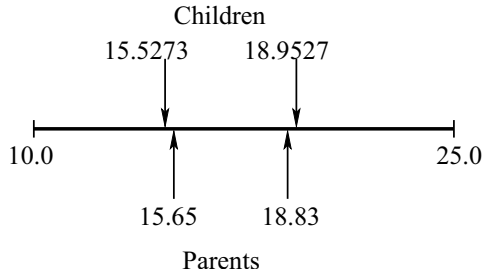


Figure 4.4: The parents and their children obtained using simulated binary crossover.

4.1.2 Mutation Operators

Several versions of mutation operator have been proposed for the real-coded GA by various researchers. Some of these are explained below.

Random Mutation [42]:

Here, mutated solution is obtained from the original solution using the rule given below.

$$Pr_{mutated} = Pr_{original} + (r - 0.5)\Delta, \quad (4.4)$$

where r is a random number varying in the range of $(0.0, 1.0)$, Δ is the maximum value of perturbation defined by the user.

Example:

Let us assume the original parent solution $Pr_{original} = 15.6$. Determine the mutated solution $Pr_{mutated}$, corresponding to $r = 0.7$ and $\Delta = 2.5$.

The mutated solution is calculated like the following:

$$Pr_{mutated} = 15.6 + (0.7 - 0.5) \times 2.5 = 16.1.$$

Polynomial Mutation:

Deb and Goyal [43] proposed a mutation operator based on polynomial distribution. The following steps are considered to obtain the mutated solution from an original solution:

- **Step 1:** Generate a random number r lying between 0.0 and 1.0.
- **Step 2:** Calculate the perturbation factor $\bar{\delta}$ corresponding to r using the following equation:

$$\bar{\delta} = \begin{cases} (2r)^{\frac{1}{q+1}} - 1 & \text{if } r < 0.5, \\ 1 - [2(1 - r)]^{\frac{1}{q+1}} & \text{if } r \geq 0.5, \end{cases} \quad (4.5)$$

where q is an exponent (positive real number).

- **Step 3:** The mutated solution is then determined from the original solution as follows:

$$Pr_{mutated} = Pr_{original} + \bar{\delta} \times \delta_{max},$$

where δ_{max} is the user-defined maximum perturbation allowed between the original and mutated solutions.

Example:

Let us assume the original parent solution $Pr_{original} = 15.6$. Determine the mutated solution $Pr_{mutated}$, considering $r = 0.7$, $q = 2$ and $\delta_{max} = 1.2$.

Corresponding to $r = 0.7$ and $q = 2$, the perturbation factor $\bar{\delta}$ is found to be as follows:

$$\bar{\delta} = 1 - [2(1 - r)]^{\frac{1}{q+1}} = 0.1565$$

The mutated solution is then determined from the original solution like the following.

$$Pr_{mutated} = Pr_{original} + \bar{\delta} \times \delta_{max} = 15.7878.$$

4.2 Micro-GA

As the string length of a binary-coded Simple GA (SGA) increases to ensure better precision in the values of the variables, it is advised to run the GA with a large population of solutions. The larger population ensures a better schema processing and consequently, there is a less chance of occurrence of premature convergence. Thus, the global optimal solutions may be obtained but at the cost of a slow convergence rate. The SGA cannot be used for on-line optimization of most of the real-world problems (such as on-line control of a mobile robot), in which the objective function may change at a rate faster than the SGA can reach the optimal solution. Realizing the need of a faster GA for on-line implementations, a small population-GA (known as **micro-GA**) was introduced by Krishnakumar [19], which is basically a binary-coded GA. The working principle of the micro-GA is explained below in steps.

- **Step 1:** Select an initial population of binary strings of size 5, at random.
- **Step 2:** Evaluate the fitness of the strings and identify the best one. Mark the best string as string 5 and copy it directly into the mating pool. It is known as **elitist strategy**. Thus, there is a guarantee that the already found good schema will not be lost.
- **Step 3:** Select the remaining four strings (the best/elite string also participates in reproduction) based on a deterministic tournament selection strategy.
- **Step 4:** Carry out crossover with a probability $p_c = 1.0$ to ensure the better schema processing. The mutation probability is kept equal to zero.
- **Step 5:** Check for convergence. If the convergence criterion is reached, terminate the program. Otherwise, go to the next step.
- **Step 6:** Create a new population of strings of size 5 by copying the best (elite) string of the semi-converged population and then generating the remaining four strings at random. Go to Step 2.

Micro-GA is found to be faster than the conventional SGA. Initially, all five strings are generated at random and new four strings are added to the population in all subsequent generations. Therefore, the necessary diversity has been maintained in the population during its search. However, the chance of pre-mature convergence of the algorithm cannot be totally ignored.

4.3 Visualized Interactive GA

Visualized Interactive GA (VIGA) [44, 45] has been developed to serve the following two purposes:

- To investigate topological information of the surface of objective function to be optimized,
- To accelerate the GA-search.

To develop the VIGA, some mapping methods are used to map the higher dimensional data to a lower dimensional space/plane for visualization. Some of these mapping tools are discussed below.

4.3.1 Mapping Methods

Our visualization capability is restricted to three dimensions ($3 - D$) only. We are unable to visualize the higher dimensional (more than $3 - D$) data. Thus, these higher dimensional (say L -dimensional) data are to be mapped to 2- or $3 - D$ for visualization. A number of methods had been tried in the past to map the data from a higher dimensional space to a lower dimension. These mapping methods can be classified into two groups, namely **linear** and **non-linear methods**. The linear methods include **principal component analysis**, **least square mapping**, **projection pursuit mapping** and others [46, 47]. On the other hand, there are some non-linear mapping methods, namely **Sammon's Non-Linear Mapping (NLM)** [48], **VISOR algorithm** [49], **Self-Organizing Maps (SOM)** [50] (which is a non-linear generalization of Principal Component Analysis [51]), and others. Out of all these mapping tools, only two are discussed below, in detail. Moreover, the principle of another mapping tool, namely SOM will be discussed in Chapter 10.

A. Sammon's Nonlinear Mapping:

Sammon's Non-Linear Mapping (NLM) is a distance preserving technique, in which the error in mapping is minimized through a number of iterations using a gradient descent method [48].

Let us consider N vectors in a multivariate data space of L -dimension, which are denoted by X_i , where $i = 1, 2, \dots, N$. Our objective is to map these N -vectors from L -dimensional space to 2 or 3-dimensional space. Let us also suppose that the mapped data in 2 or 3-dimension are represented by Y_i , where $i = 1, 2, \dots, N$. The scheme is shown in Fig. 4.5.

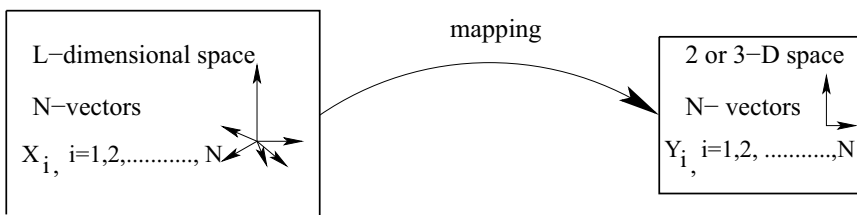


Figure 4.5: Mapping from L -dimensional space to 2 or 3 - D space using NLM.

The N vectors in L -space are expressed as follows:

$$X_1 = \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1L} \end{bmatrix}; \quad X_2 = \begin{bmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2L} \end{bmatrix}; \quad \dots; \quad X_N = \begin{bmatrix} x_{N1} \\ x_{N2} \\ \vdots \\ x_{NL} \end{bmatrix}.$$

Similarly, the N -vectors can be represented in $2 - D$ plane or $3 - D$ space as given below.

$$Y_1 = \begin{bmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1D} \end{bmatrix}; \quad Y_2 = \begin{bmatrix} y_{21} \\ y_{22} \\ \vdots \\ y_{2D} \end{bmatrix}; \quad \dots; \quad Y_N = \begin{bmatrix} y_{N1} \\ y_{N2} \\ \vdots \\ y_{ND} \end{bmatrix}.$$

The mapping technique can be described as follows:

1. Create N vectors in 2-dimensional plane at random.
2. Let d_{ij}^* be the Euclidean distance between two vectors X_i and X_j in a higher dimensional space and d_{ij} is the distance between the corresponding two mapped points Y_i and Y_j , in a lower (say 2) dimension. For an exact mapping, the following condition is to be satisfied:

$$d_{ij}^* = d_{ij}. \quad (4.6)$$

3. Let us assume that $E(m)$ represents the mapping error in m -th iteration, which can be expressed mathematically as follows:

$$E(m) = \frac{1}{C} \sum_{i=1}^N \sum_{j=1(i < j)}^N \frac{[d_{ij}^* - d_{ij}(m)]^2}{d_{ij}^*}, \quad (4.7)$$

where $C = \sum_{i=1}^N \sum_{j=1(i < j)}^N d_{ij}^*$ and $d_{ij}(m) = \sqrt{\sum_{k=1}^D [y_{ik}(m) - y_{jk}(m)]^2}$.

4. The mapping error is minimized using a steepest descent method, for which the relationship between the m -th and $(m+1)$ -th iterations can be expressed like the following:

$$y_{pq}(m+1) = y_{pq}(m) - (MF)\Delta_{pq}(m), \quad (4.8)$$

where MF is a magic factor, which varies in the range of 0.0 to 1.0, and

$$\Delta_{pq}(m) = \frac{\frac{\partial E(m)}{\partial y_{pq}(m)}}{|\frac{\partial^2 E(m)}{\partial y_{pq}^2(m)}|}.$$

Now, $\frac{\partial E}{\partial y_{pq}}$ and $\frac{\partial^2 E}{\partial y_{pq}^2}$ are expressed as follows:

$$\frac{\partial E}{\partial y_{pq}} = \frac{-2}{C} \sum_{j=1, j \neq p}^N \left[\frac{d_{pj}^* - d_{pj}}{d_{pj} d_{pj}^*} \right] (y_{pq} - y_{jq}),$$

$$\frac{\partial^2 E}{\partial y_{pq}^2} = \frac{-2}{C} \sum_{j=1, j \neq p}^N \frac{1}{d_{pj}^* d_{pj}} \left[(d_{pj}^* - d_{pj}) - \frac{(y_{pq} - y_{jq})^2}{d_{pj}} \left(1 + \frac{d_{pj}^* - d_{pj}}{d_{pj}} \right) \right].$$

It is important to mention that this algorithm is iterative in nature and could be computationally expensive.

B. VISOR Algorithm:

VISOR Algorithm [49] is a mapping technique, that uses a geometrical method for data projection by preserving distance information. As the algorithm uses some simple geometrical rules, it is simple to understand and easy to implement.

Let us suppose that a number of points (say N) of L -dimensional space are to be mapped to a 2-dimensional plane with a minimum error. Figure 4.6 shows the scheme of VISOR algorithm.

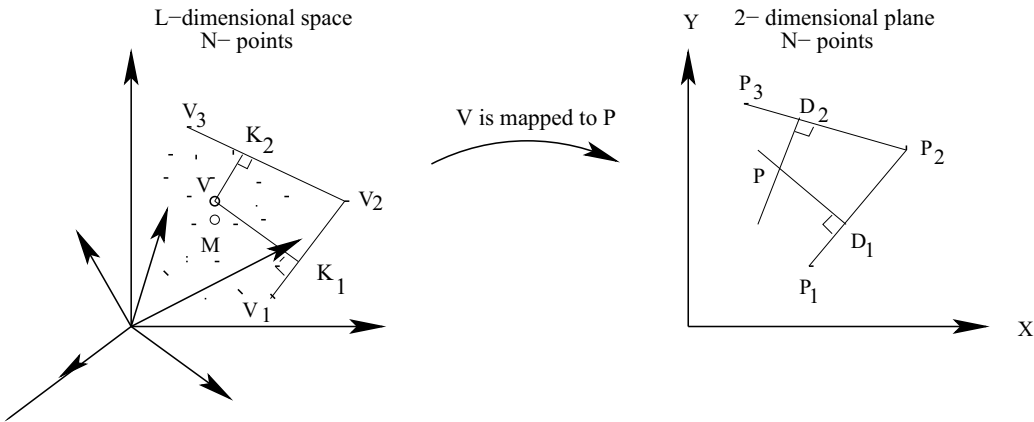


Figure 4.6: VISOR algorithm.

The method can be explained as follows:

1. The pivot-vectors: V_1, V_2, V_3 are determined in the L -dimensional space, which provide a convex enclosure to the remaining data points. The following approach is adopted for the said purpose:
 - Compute centroid M of all data points N in the original L -dimensional space,
 - Determine the pivot vector V_1 such that distance $d(V_1, M) = \max_{i=1}^N (d(v_i, M))$,
 - Determine the pivot vector V_2 such that distance $d(V_2, V_1) = \max_{i=1}^{N-1} (d(v_i, V_1))$,
 - Determine the pivot vector V_3 such that distance $d(V_3, V_2) = \max_{i=1}^{N-2} (d(v_i, V_2))$ and $d(V_3, V_1) = \max_{i=1}^{N-2} (d(v_i, V_1))$.

2. The pivot-vectors: P_1, P_2, P_3 are located in 2-dimensional plane corresponding to the vectors V_1, V_2, V_3 , respectively, using the information of Euclidean distance.
3. Mapping of the remaining $(N - 3)$ points will follow this approach:
 - Supposing that a particular point V of L -dimensional space is to be mapped to a corresponding point in 2-dimensional plane. The lines V_1V_2 and V_2V_3 are considered. Two perpendicular lines are drawn from the point V to the straight lines V_1V_2 and V_2V_3 . Thus, the points K_1 and K_2 are obtained on the lines V_1V_2 and V_2V_3 , respectively.
 - In $2 - D$, the point D_1 is located on the line P_1P_2 , such that D_1 divides the line P_1P_2 in the same proportion as K_1 has divided the line V_1V_2 (in L -dimensional space). Similarly, the point D_2 is determined on the line P_2P_3 .
 - The perpendiculars are drawn at the points D_1 and D_2 to the lines P_1P_2 and P_2P_3 , respectively. They intersect at the point P . Thus, the point V of L -dimension is mapped to a point P into $2 - D$.

It is interesting to note that in this algorithm, the higher dimensional data are mapped into a lower dimension in one iteration only. It is expected to be a faster algorithm, as it is a non-iterative one.

4.3.2 Simulation Results

Figure 4.7 shows the surface plot of Schaffer's F1.

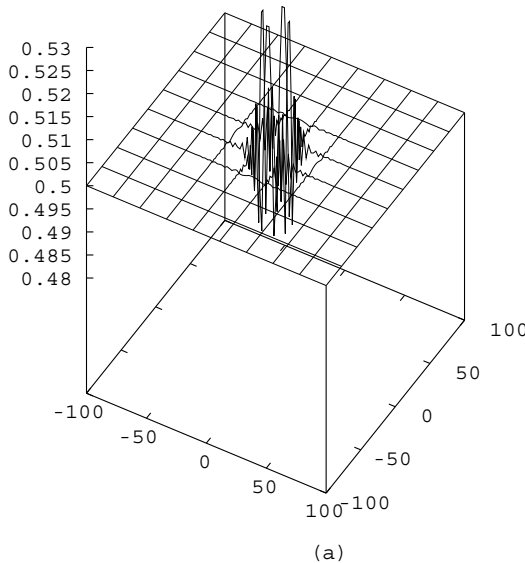


Figure 4.7: Surface plot of Schaffer's F1 function.

It is mathematically expressed as follows:

$$y = 0.5 + \frac{\sin^2 \sqrt{\sum_{i=1}^4 x_i^2} - 0.5}{1.0 + 0.001(\sum_{i=1}^4 x_i^2)^2}. \quad (4.9)$$

Let us suppose that 200 random points lying on the surface of this function are to be mapped to $2-D$ for visualization. Figure 4.8 shows the mapped data in $2-D$ using Sammon's NLM

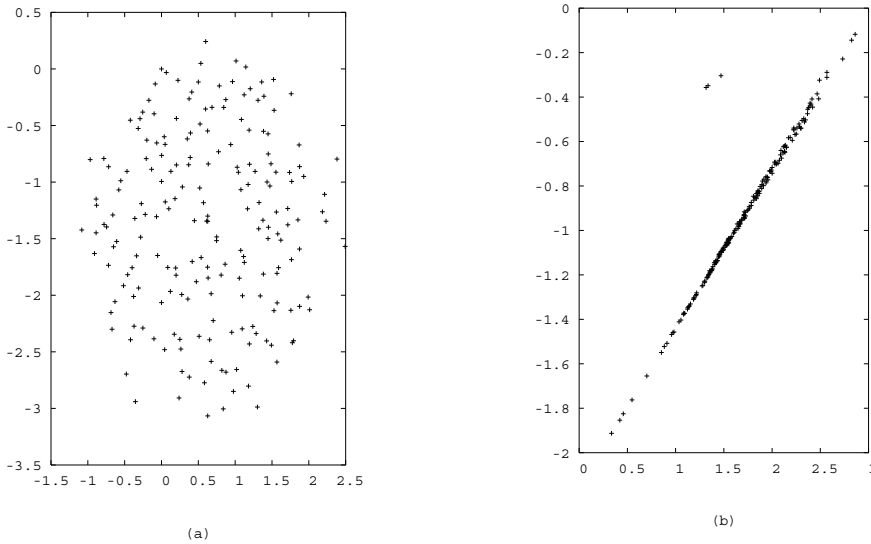


Figure 4.8: Comparison of the mapped data obtained using (a) Sammon's NLM and (b) Visor algorithm .

and Visor algorithm. It is important to note that Sammon's NLM is able to outperform VISOR algorithm in terms of accuracy in mapping. The mapped data obtained using the Sammon's NLM are seen to be widely distributed, whereas those determined by the VISOR algorithm are found to assemble together. Thus, a particular point and its neighbors can easily be identified from the mapped data yielded by Sammon's NLM, whereas VISOR algorithm cannot offer a good visibility of the mapped data. Moreover, VISOR algorithm is found to be computationally faster than the Sammon's NLM. Interested readers may refer to Dutta and Pratihari [52] for a detailed comparison of different mapping methods.

Note: Sometimes a data set consisting of a very large number of features (that is, a very high dimensional data set) may contain some redundant and irrelevant information. For such a high dimensional data, feature extraction may help to select a subset of original features based on some optimization criteria and thus, it helps the machine learning algorithm to improve its accuracy in predictions and enhance comprehensibility of the obtained results. For feature extraction of high dimensional data set, two models, namely **filter model** [53] and **wrapper model** [54] are generally used. The filter model tries to select some important features of the training data without using any learning algorithm. On the other hand, the wrapper model uses a learning algorithm to select the important features

of the data set. The wrapper model is expected to perform better than the filter model but at the cost of more computation.

4.3.3 Working Principle of the VIGA

Let us suppose that an objective function involving L dimensions (where $L > 3$) is to be optimized using a GA. In such a case, we cannot visualize the search direction of the GA on the surface of objective function but it finds the optimal solution after conducting its search through a few iterations. We may not even know the type of processing actually takes place inside the GA, although we get the optimal solution. Thus, a GA works almost like a black box. In a VIGA, an attempt will be made to collect topological information of the higher dimensional space by mapping a set of data into a lower dimensional space (2 or 3-D) using some methods, namely Sammon's NLM, VISOR, SOM, and others. As the neighboring points in a higher dimensional space try to keep their positional information and topological relation intact in a lower dimensional space during mapping, it is possible to locate the global basin in a higher dimensional space by examining the data points in $2 - D$ or $3 - D$. A human being has a capability to study the data points in $2 - D$ or $3 - D$ and thus, to identify the probable location of the global optimum. This information regarding a possible region of global optimum in a lower dimensional space is transformed into a higher dimensional space using a reverse mapping, that is generally implemented with the help of a look-up table. Figure 4.9 shows a schematic view explaining the working principle of the VIGA. The GA is being informed by the user of the possible location of good solutions at

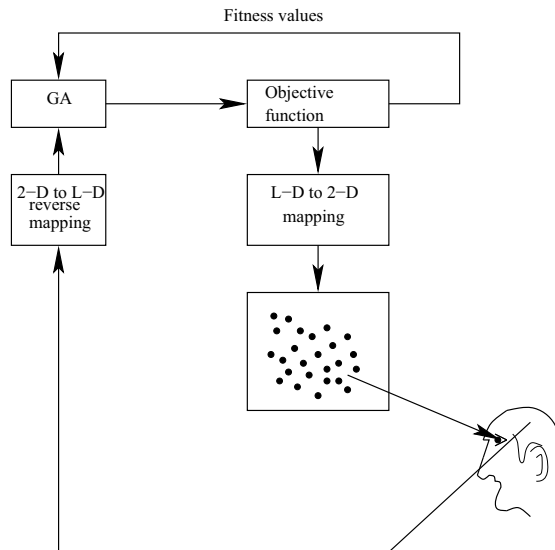


Figure 4.9: A schematic view to show the working principle of a VIGA.

each iteration. As good solutions are added at each iteration, the convergence rate of the VIGA is expected to be higher than that of the conventional SGA. The VIGA is seen to

be almost five times faster than the normal SGA [44, 45]. However, the performance of the VIGA is found to be function-dependent.

4.4 Scheduling GA

Scheduling problems belong to a special class of optimization problems, in which not only the position of different elements but also their adjacency and order are important. Let us take an example of a **Travelling Salesman Problem (TSP)** (that is, a special type of scheduling problem), where a salesman has to visit all n cities (each city is visited only once) by travelling either through a minimum distance path or in minimum time or at a minimum cost. Let us assume that all n cities are connected to each other by some suitable paths. Let us also consider this problem to be a symmetrical one, in which the travelling distance/time/cost between any two cities is independent of the direction of travel. Fig. 4.10 shows a symmetrical TSP.

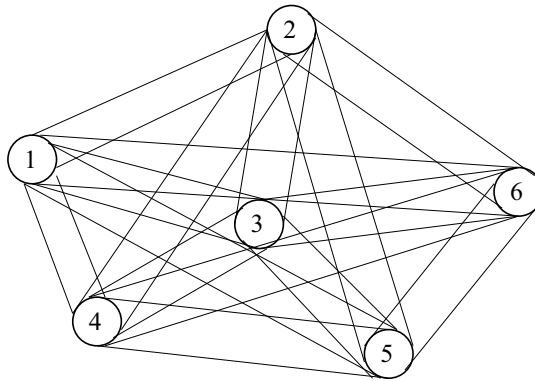


Figure 4.10: A symmetrical TSP, in which all the cities are connected to each other in both ways.

The symbol d_{ij} represents the Euclidean distance between the cities i and j . The distance matrix is shown below.

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} & d_{26} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} & d_{36} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} & d_{46} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} & d_{56} \\ d_{61} & d_{62} & d_{63} & d_{64} & d_{65} & d_{66} \end{bmatrix} = \begin{bmatrix} 0 & 50 & 45 & 40 & 70 & 75 \\ 50 & 0 & 40 & 60 & 60 & 50 \\ 45 & 40 & 0 & 40 & 35 & 40 \\ 40 & 60 & 40 & 0 & 45 & 70 \\ 70 & 60 & 35 & 45 & 0 & 40 \\ 75 & 50 & 40 & 70 & 40 & 0 \end{bmatrix}$$

With these assumptions, the above scheduling problem can be posed as an optimization problem. If a salesman starts from a particular city, he will have $(n - 1)!$ possible routes/sequences through which he can touch all the cities once. He will have to find the optimal path out of all $(n - 1)!$ possibilities. As the salesman has to visit all n cities, the optimal path will be independent of selection of the starting city. Thus, he has a maximum of $n!$ possible sequences and the optimal one is to be determined.

Let us consider another situation, in which all n cities are not connected to each other (in both ways) and some cities are connected to some other cities only in one direction. Thus, it is a partially connected asymmetrical TSP, as shown in Figure 4.11.

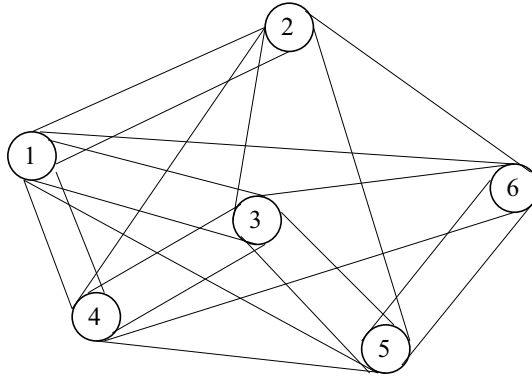


Figure 4.11: An asymmetrical TSP.

The distance matrix is given below.

$$\begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & d_{16} \\ d_{21} & d_{22} & d_{23} & d_{24} & d_{25} & d_{26} \\ d_{31} & d_{32} & d_{33} & d_{34} & d_{35} & d_{36} \\ d_{41} & d_{42} & d_{43} & d_{44} & d_{45} & d_{46} \\ d_{51} & d_{52} & d_{53} & d_{54} & d_{55} & d_{56} \\ d_{61} & d_{62} & d_{63} & d_{64} & d_{65} & d_{66} \end{bmatrix} = \begin{bmatrix} 0 & 50 & 45 & 40 & 70 & - \\ 55 & 0 & - & 60 & 60 & 50 \\ 50 & 35 & 0 & 40 & 35 & 40 \\ 35 & - & 35 & 0 & 45 & 70 \\ - & - & 30 & - & 0 & 40 \\ 70 & - & - & - & 45 & 0 \end{bmatrix}$$

It is clear from the above discussion that a Simple GA (SGA) may not be suitable to solve the scheduling problems, as the conventional crossover operators (such as single-point, two-point, multi-point, uniform crossovers) may yield some infeasible children solutions. It is important to note that mutation is generally not used to solve the above problem, unless it is required in a special situation. Thus, a special type of crossover operator is to be used to determine the optimal schedule. Some of these specialized crossover operators are discussed below in detail [55].

4.4.1 Edge Recombination

It was developed by Whitley et al. [56], in which an importance is given on adjacency information but not on the order and position of the elements. We maintain an edge table, which carries information of each element and its possible links. Thus, it gives the connectivity information of each element.

Let us consider a travelling salesman problem involving nine cities (refer to Fig. 4.12). Let us also assume that children solutions are to be created using edge recombination from

the two parents given below.

Pr1 : 1 2 3 4 5 6 7 8 9
Pr2 : 9 3 1 4 5 8 2 6 7

Fig. 4.12 shows the edge table also constructed for the said purpose.

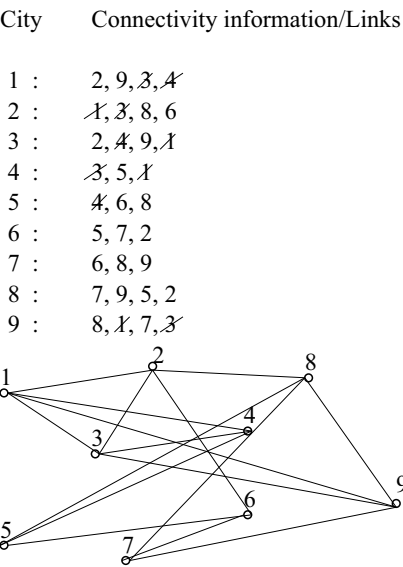


Figure 4.12: Edge recombination operator.

The following procedure is adopted to determine a child solution, say child 1, that is, Ch1:

- Let us start the child tour with the starting city of Pr1, that is, 1.
- As city 1 has been selected, all occurrences of 1 are to be deleted from the right-hand side of edge table.
- City 1 is connected to the cities 2, 3, 4 and 9. Now, the cities 2, 3 and 9 have three remaining connectivities each in the edge table, whereas city 4 has only two remaining links. Thus, 4 is selected as the next city to city 1.
- We eliminate all entries of city 4 from the right-hand side of edge table.
- City 4 has the links to the cities 3 and 5 (as 1 has already been selected). Now, both the cities 3 and 5 have two remaining links in the edge table. Thus, any one out of the cities 3 and 5 may be selected at random. Let us select city 3 as the next city to city 4.
- We remove all entries of city 3 from the right-hand side of edge table.

We continue this process, until the tour is completed after touching all the cities once. The resulting child 1 will be as follows:

$$Ch1: 1 \ 4 \ 3 \ 2 \ 6 \ 5 \ 8 \ 7 \ 9$$

Similarly, child 2, that is, Ch2 may be determined considering city 9 (that is, starting city of Pr2) as the starting city of Ch2. Using the above steps, Ch2 is obtained like the following:

$$Ch2: 9 \ 7 \ 6 \ 5 \ 8 \ 2 \ 1 \ 3 \ 4$$

4.4.2 Order Crossover #1

Order crossover #1 was proposed by Davis [57], in which a part of one child solution is obtained by directly copying from a part of one parent and the other part of that child inherits the order of remaining elements of other parent. A care has to be taken, such that an element already selected in the child solution should not appear again.

Let us suppose that order crossover #1 is to be carried out on the following two parents to create two children:

$$\begin{array}{cccc|cccc|cc} Pr1: & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ Pr2: & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \\ & & & & * & & & * & & \end{array}$$

The steps to be followed to get two children solutions from the above two parents are:

- Select two crossover sites at random, as shown above.
- To determine child 1, that is, Ch1, the elements: 1, 2, 9, 8 (lying between the two crossover sites) are directly copied from Pr2, keeping their locations and order intact.
- Child 1 tour then begins from Pr1, starting from the first position after the second crossover site and searching towards the initial elements of Pr1.
- Cities (elements) 8, 9, 1, 2 are already present in child 1 solution, which should not be considered again.
- The 8 – *th*, 9 – *th*, 1 – *st*, 2 – *nd* and 3 – *rd* positions of Ch1 are selected as follows:

$$\begin{aligned} Ch1[8] &= Pr1[3] = 3, \\ Ch1[9] &= Pr1[4] = 4, \\ Ch1[1] &= Pr1[5] = 5, \\ Ch1[2] &= Pr1[6] = 6, \\ Ch1[3] &= Pr1[7] = 7. \end{aligned}$$

Thus, the obtained Ch1 will look as follows:

$$Ch1: 5 \ 6 \ 7 \ (\ 1 \ 2 \ 9 \ 8 \) \ 3 \ 4$$

Similarly, child 2, that is, Ch2 can be determined and it will look like the following:

$$Ch2: 2 \ 9 \ 8 \ (\ 4 \ 5 \ 6 \ 7 \) \ 3 \ 1$$

4.4.3 Order Crossover #2

Order crossover #2 was developed by Syswerda [58], in which some key positions are selected at random and the order in which the elements of these positions appear in one parent is imposed on other parent to create the children solutions.

Let us suppose that the following two parents participate in order crossover #2:

$$\begin{array}{rcccccccc} Pr1 : & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & & * & & * & & & * & * & \\ Pr2 : & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \end{array}$$

The principle of this operator is explained below in steps.

- Select the positions: 2, 4, 7 and 8 as the key positions, at random.
- The elements of Pr2 at the above mentioned key positions are 4, 1, 8, 7. To determine Ch1, the ordering of these elements: 4, 1, 8, 7 in Pr2 is imposed on Pr1.
- In Pr1, the elements: 4, 1, 8 and 7 are found in positions: 4 - *th*, 1 - *st*, 8 - *th* and 7 - *th*, respectively.
- In Ch1, the elements in these positions (that is, 1 - *st*, 4 - *th*, 7 - *th* and 8 - *th*) are selected by matching the order of the elements 4, 1, 8, 7 as found in Pr2, that is, $Ch1[1] = 4$; $Ch1[4] = 1$; $Ch1[7] = 8$; $Ch1[8] = 7$.
- The remaining elements of Ch1 are directly copied from Pr1.

Thus, Ch1 will look like the following:

$$Ch1 : 4 \quad 2 \quad 3 \quad 1 \quad 5 \quad 6 \quad 8 \quad 7 \quad 9$$

Similarly, Ch2 can be determined, which will look as follows:

$$Ch2 : 3 \quad 2 \quad 5 \quad 1 \quad 4 \quad 9 \quad 7 \quad 8 \quad 6$$

4.4.4 Cycle Crossover

It was proposed by Oliver et al. [59], in which the absolute positions of elements of a parent are preserved while determining a child solution.

Let us consider two parents, as shown below, which will participate in a cycle crossover to create two children solutions.

$$\begin{array}{rcccccccc} Pr1 : & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & & & * & & & & & & \\ Pr2 : & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \end{array}$$

The mechanism of cycle crossover is explained with the help of the following steps:

- To determine $Ch1$, select a parent (say $Pr1$) and the starting position of the cycle (say $Pr1[3] = 3$), at random. Thus, the 3 – rd element of $Ch1$ is nothing but element 3, that is, $Ch1[3] = 3$.
- $Pr2$ is then searched to check the presence of element 3 and it has been found in the 1 – st position. The first element of $Ch1$ is selected from the first element of $Pr1$, that is, $Ch1[1] = Pr1[1] = 1$.
- $Pr2$ is again searched for a presence of element 1 and it has occurred at the 4 – th position. Thus, the 4 – th element of $Pr1$ has been copied as the 4 – th element of $Ch1$, that is, $Ch1[4] = Pr1[4] = 4$. Similarly, we determine

$$\begin{aligned} Ch1[2] &= Pr1[2] = 2, \\ Ch1[5] &= Pr1[5] = 5. \end{aligned}$$

This completes one cycle because element 5 is seen to be present at the 3 – rd position of $Pr2$ and the corresponding 3 – rd position element of $Pr1$ is element 3, which has already been selected as the starting element of the cycle.

- The remaining elements of $Ch1$ are selected directly from $Pr2$, as follows:

$$\begin{aligned} Ch1[6] &= Pr2[6] = 9, \\ Ch1[7] &= Pr2[7] = 8, \\ Ch1[8] &= Pr2[8] = 7, \\ Ch1[9] &= Pr2[9] = 6. \end{aligned}$$

Thus, $Ch1$ is found to be like the following:

$$Ch1 : 1 \ 2 \ 3 \ 4 \ 5 \ 9 \ 8 \ 7 \ 6$$

Using the same procedure, $Ch2$ can be determined as follows:

$$Ch2 : 3 \ 4 \ 5 \ 1 \ 2 \ 6 \ 7 \ 8 \ 9$$

4.4.5 Position-Based Crossover

Position-based crossover was introduced by Syswerda [58], in which an attempt is made to preserve position information of different parent elements in the child solution.

Let us consider the following two parents, which will participate in the position-based crossover:

$$\begin{array}{cccccccccc} Pr1 : & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ & & * & & * & & * & & * & \\ Pr2 : & 3 & 4 & 5 & 1 & 2 & 9 & 8 & 7 & 6 \end{array}$$

The principle of position-based crossover is discussed with the help of following steps:

- To determine $Ch1$, choose a number of crossover points (say 1-st, 4-th, 7-th, 9-th) on a parent, say $Pr1$. The elements: 1, 4, 7, 9 are directly copied from $Pr1$ (by keeping their position information intact) to $Ch1$. Thus, we get $Ch1[1] = 1$, $Ch1[4] = 4$, $Ch1[7] = 7$, $Ch1[9] = 9$.
- The remaining elements of $Ch1$ are selected from $Pr2$ as follows:
 $Ch1[2] = Pr2[1] = 3$;
 $Ch1[3] = Pr2[3] = 5$, as $Pr2[2] = 4$ has already been included in $Ch1$;
 $Ch1[5] = Pr2[5] = 2$, as $Pr2[4] = 1$ has already been selected as $Ch1[1]$ element;
 $Ch1[6] = Pr2[7] = 8$, as $Pr2[6] = 9$ has already been considered as $Ch1[9]$;
 $Ch1[8] = Pr2[9] = 6$, as $Pr2[8] = 7$ has already occurred as $Ch1[7]$.

Thus, the resulting $Ch1$ solution will look as follows:

$$Ch1: 1 \ 3 \ 5 \ 4 \ 2 \ 8 \ 7 \ 6 \ 9$$

Similarly, $Ch2$ can be determined, which will look like the following:

$$Ch2: 3 \ 2 \ 4 \ 1 \ 5 \ 7 \ 8 \ 9 \ 6$$

4.4.6 Partially Mapped Crossover (PMX)

It was proposed by Goldberg and Lingle [60], in which a part of a parent solution lying between two crossover sites is directly copied into a child solution. Thus, the position, adjacency and order of that part of the parent will remain intact in the child solution.

Let us take an example of the following two parents, which will participate in a Partially Mapped Crossover (PMX):

$$\begin{array}{cccc|cccc|cc}
 Pr1: & 1 & 2 & 3 & & 4 & 5 & 6 & 7 & & 8 & 9 \\
 Pr2: & 3 & 4 & 5 & & 1 & 2 & 9 & 8 & & 7 & 6 \\
 & & & & & * & & & & & * &
 \end{array}$$

The steps involved in the PMX are:

- Select two crossover sites at random, as shown above.
- The elements: 4, 5, 6, 7 of $Pr1$ (lying within the two crossover sites) are directly copied into $Ch1$. Thus, we get $Ch1[4] = 4$, $Ch1[5] = 5$, $Ch1[6] = 6$, and $Ch1[7] = 7$.
- The remaining elements of $Ch1$ are determined as follows: The search starts with the elements of $Pr1$ residing in between the two crossover sites. The element $Pr1[4] = 4$ is located at the 2 – nd position of $Pr2$, that is, $Pr2[2]$. Thus, the 2 – nd position of $Ch1$ will be filled up by an element of $Pr2$ located at 4 – th position, that is, city 1.

$$Ch1[2] = Pr2[4] = 1.$$

Similarly, the element $Pr1[5] = 5$ is seen to be present at the $3 - rd$ position of $Pr2$, that is, $Pr2[3]$. Thus, the $3 - rd$ position of $Ch1$ will be occupied by an element of $Pr2$ located at the $5 - th$ position, that is, city 2.

$$Ch1[3] = Pr2[5] = 2.$$

Similarly, we can determine the following elements of $Ch1$:

$$\begin{aligned} Ch1[9] &= Pr2[6] = 9, \\ Ch1[8] &= Pr2[7] = 8. \end{aligned}$$

- The remaining element of $Ch1$ is directly copied from $Pr2$, that is, $Ch1[1] = Pr2[1] = 3$.

Thus, $Ch1$ is obtained as follows:

$$Ch1: 3 \ 1 \ 2 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9$$

$Ch2$ can be determined following the similar procedure and it will look like the following:

$$Ch2: 4 \ 5 \ 3 \ 1 \ 2 \ 9 \ 8 \ 7 \ 6$$

Note: Partially mapped crossover may sometimes give rise to a child solution, in which a particular city may occur more than once and some other cities might also be missing from it.

Let us take an example, in which two parents are participating in the PMX, as shown below.

$$\begin{array}{cccc|cccc|cccc} Pr1: & 1 & 2 & & 3 & 4 & 5 & 6 & & 7 & 8 & 9 \\ Pr2: & 3 & 4 & & 5 & 1 & 2 & 9 & & 8 & 7 & 6 \\ & & & & * & & & * & & & & \end{array}$$

The following children solutions are obtained using the PMX:

$$\begin{aligned} Ch1: & 5 \ 1 \ (\ 3 \ 4 \ 5 \ 6 \) \ 8 \ 7 \ 9 \\ Ch2: & 4 \ 5 \ (\ 5 \ 1 \ 2 \ 9 \) \ 7 \ 8 \ 6 \end{aligned}$$

In $Ch1$, city 5 has occurred twice and city 2 is missing from it. The cities lying inside the first brackets are generally not disturbed. Thus, the first element of $Ch1$, that is, city 5 is to be replaced by the missing city, that is, 2. The modified $Ch1$ may be written as follows:

$$Ch1: 2 \ 1 \ 3 \ 4 \ 5 \ 6 \ 8 \ 7 \ 9$$

Similarly, the modified $Ch2$ is found to be like the following:

$$Ch2: 4 \ 3 \ 5 \ 1 \ 2 \ 9 \ 7 \ 8 \ 6$$

It is important to mention that the performances of different crossover operators are problem-dependent [55].

4.5 Summary

The content of this chapter has been summarized as follows:

The mechanisms of different crossover and mutation operators used in the real-coded GAs by various investigators, have been explained with the help of suitable numerical examples. The working principle of a micro-GA has been discussed, in which elitism has been used. An introduction is given to another faster GA named Visualized Interactive GA, where some mapping methods like Sammon's NLM, VISOR algorithm, and others, are used to map the data from a higher dimensional space to a lower dimensional space for visualization. In a scheduling problem, not only the position of different elements but also their adjacency and order are to be considered. A number of crossover operators have been proposed by various researchers to solve the said problem. The principles of some of these operators have been explained with the help of some appropriate examples.

4.6 Exercise

1. When and why do we prefer a real-coded GA to a binary-coded GA ?
2. Define elitism and briefly explain the principle of micro-GA.
3. Can the VIGA be faster than the SGA ? Explain it.
4. Why do we need a special type of crossover operator for solving a scheduling problem ? Explain briefly different crossover operators used in the scheduling GA.
5. To solve an optimization problem using a real-coded GA, let us assume that a mating pair (consisting of two solutions) is found to be as follows:

$$Pr1 = 16.85, Pr2 = 19.50$$

Determine the children solutions using different crossover operators given below.

- (a) Linear crossover,
- (b) Blend crossover with $\alpha = 0.6$, that is, $BLX - 0.6$, assume the random number $r = 0.7$.
- (c) Simulated Binary Crossover (SBX) assuming the probability distributions for the contracting and expanding zones as follows:

$$C(\alpha) = 0.5(q+1)\alpha^q,$$

$$Ex(\alpha) = 0.5(q+1)\frac{1}{\alpha^{(q+2)}},$$

where α is the spread factor and $q = 4$. Assume the random number $r = 0.7$.

6. To solve an optimization problem utilizing a real-coded GA, determine the mutated value corresponding to an original solution $Pr_{original} = 19.68$ under the following conditions:

- (a) Use random mutation, assuming random number $r = 0.6$ and the maximum perturbation $\Delta = 2.0$.
- (b) Use polynomial mutation, assuming the random number $r = 0.6$; $q = 2$; $\delta_{max} = 1.0$.
7. Let us assume that power generation of a plant is dependent on three input real variables: x_1 , x_2 and x_3 varying in the ranges of (5.0, 10.0), (1.5, 5.5) and (10.0, 20.0), respectively. Let us also consider that the generated power P_g depends on the input variables as given below.

$$P_g = f(x_1, x_2, x_3) = x_1 - x_2 + x_3 - x_1^2 + x_2^2 + x_3^2 + x_1x_2 + x_2x_3 + x_3x_1 + x_1x_2x_3$$

The power demands for two consecutive hours are found to be 450 and 750 MW, respectively. Formulate it as an optimization problem, so that the demand can be reached as closely as possible by properly setting the input variables. Use a real-coded GA with simulated binary crossover (SBX) and polynomial mutation to solve the said problem. Show only one iteration through hand calculations. Take population size $N = 4$, $p_c = 1.0$, $p_m = 0.05$ and use tournament selection scheme.

8. Develop the computer programs for Sammon's NLM and VISOR algorithm. Map the 4-D data shown in Table 4.1 into 2-D using the above methods.

Table 4.1: A set of 20×4 data

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.4	1.4	0.3
5.4	3.7	1.5	0.2
5.2	3.5	1.5	0.2
5.2	3.4	1.4	0.2
4.7	3.2	1.6	0.2
5.5	4.2	1.4	0.2
4.5	2.3	1.3	0.3
5.1	3.8	1.9	0.4
4.8	3.0	1.4	0.3
5.7	2.5	5.0	2.0
5.8	2.8	5.1	2.4
6.1	3.0	4.9	1.8
7.9	3.8	6.4	2.0
6.3	3.4	5.6	2.4
6.4	3.1	5.5	1.8
6.2	3.4	5.4	2.3
5.9	3.0	5.1	1.8

9. Let us consider a TSP problem involving eight cities A, B, C, D, E, F, G, H. A scheduling GA with various types of crossover operator is to be used to solve the said optimization problem. Determine children solutions of the following two parents:

$$\begin{array}{l} Pr1 : A \ B \ C \ D \ E \ F \ G \ H \\ Pr2 : C \ A \ D \ B \ F \ H \ E \ G \end{array}$$

- (a) Use edge recombination. The edge table is shown in Fig. 4.13.

City Connectivity information/Links

A: D, F, H, G, B

B: A, G, E, C, H

C: B, D, F

D: A, C, G, E

E: B, D, F, H

F: A, C, E

G: A, B, D, H

H: A, B, E, G

Figure 4.13: Edge table.

- (b) Order crossover #1, assuming 2-nd and 5-th sites as the crossover sites.
 (c) Order crossover #2, considering 3-rd, 4-th and 7-th as the key positions.
 (d) Cycle crossover assuming 4-th as the starting position.
 (e) Position-Based Crossover considering 2-nd, 4-th and 6-th as the crossover points.
 (f) PMX assuming 2-nd and 5-th sites as the crossover sites.

Note: The positions are counted from the left side.

Chapter 5

Overview of Other Non-Traditional Optimization Methods

Non-traditional optimization methods take the help of some natural phenomena in order to solve some problems. A huge literature is available on non-traditional optimization methods. A number of optimization algorithms had been proposed based on the principle of biological adaptation and evolution, namely Genetic Algorithms (GAs) [11] (as discussed in chapters 3 and 4), Genetic Programming (GP) [12], Evolution Strategies (ES) [13], Evolutionary Programming (EP) [14, 15]. Later on, a few more optimization algorithms were also developed by extending the working principle of the GA, such as Differential Evolution (DE) [61], Cultural Algorithm (CA) [62], and others. Simulated Annealing (SA) [63], another popular non-traditional optimization algorithm, was proposed by modeling the solidification process of molten metal artificially. Inspired by the principles of bird flocking, fish schooling and social behavior of people, Particle Swarm Optimization (PSO) [64] was proposed in 1995, as an optimization algorithm. Based on the principle of vertebrate immune system, Artificial Immune System (AIS) [65] was developed as a potential optimization algorithm. Dorigo [66] proposed a powerful optimization algorithm named Ant Colony Optimization (ACO) by modeling the behavior of an ant seeking an optimal path to search for food. Similarly, Karaboga [67] introduced another optimization algorithm called Artificial Bee Colony (ABC) based on foraging behavior of honey bee swarm. A social counterpart of the GA named Imperialist Competitive Algorithm (ICA) [68] was proposed in 2007 as an optimization algorithm, which works based on the socio-politically motivated strategy. In 2010, another non-traditional optimization algorithm called Charged System Search (CSS) [69] was developed by using the Coulomb and Gauss's laws of electro-statics. In this chapter, the working principles of two of the above non-traditional optimization algorithms, such as SA and PSO have been explained in detail.

5.1 Simulated Annealing

Annealing is a process of slow cooling. In Simulated Annealing (SA), cooling process of molten metal is modeled artificially to develop an optimization algorithm. It was proposed by Metropolis et al. [63]. A molten metal of high energy state is converted into its minimum energy state, that is, solid state, as shown in Fig. 5.1. It is important to mention that the

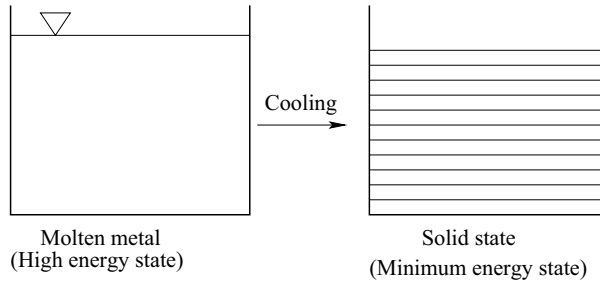


Figure 5.1: A schematic view showing cooling of molten metal into its solid state.

nature of crystal depends on the cooling rate of molten metal. At a high cooling rate, molten metal is converted into its poly-crystalline state. However, a crystalline state is formed, if a slow cooling rate of molten metal is maintained. It is also to be noted that a crystalline state is at a lower energy level compared to the poly-crystalline state. Thus, a method of slow cooling (that is, annealing) of molten metal is followed to reach the lowest level of its energy state, that is, to minimize an objective function during optimization.

5.1.1 Working Principle

Let us assume that a minimization problem involving m design variables: x_1, x_2, \dots, x_m (denoted by X) is to be solved using the SA. The problem may be stated as follows:

$$\text{Minimize } y = E(X)$$

subject to

$$X^{min} \leq X \leq X^{max}$$

The following steps are used to solve this minimization problem [10, 63]:

- **Step 1:** We assign a high initial temperature to molten metal, say T_0 ; select an initial solution X_0 at random and set a termination criterion ϵ to a small value and iteration number $t = 0$.
- **Step 2:** We calculate the temperature of $(t + 1)$ -th iteration, that is, T_{t+1} as 50% of that of t -th iteration, that is, T_t . Therefore, $T_{t+1} = 0.5 \times T_t$. We generate a candidate

solution for $(t + 1) - th$ iteration, that is, X_{t+1} at random in the neighbourhood of X_t .

- **Step 3:** If the change in energy $\delta E = E(X_{t+1}) - E(X_t) < 0$, then we accept X_{t+1} as the next solution and set $t = t + 1$,
Else we generate a random number r lying in the range of $(0.0, 1.0)$ and if $r \leq \exp(-\delta E/T_{t+1})$, we accept X_{t+1} as the next solution and set $t = t + 1$ (refer to the note given below);
 else we reject X_{t+1} and set $t = t$ and go to Step 2.
- **Step 4:** If $|E(X_{t+1}) - E(X_t)| < \epsilon$ and T reaches a small value, we terminate the program.

Note: It is important to mention that the concept of Boltzmann probability distributions has been used in this algorithm in order to implement that the probability of selecting bad solutions may be more at high temperature but it has to be less at low temperature (as the iteration proceeds). The above probability distributions state that a system kept in thermal equilibrium at temperature T has its energy distributed probabilistically as $p(E) = \exp(-E/kT)$, where k denotes Boltzmann constant. This concept of probability has been artificially implemented in the SA. Here, k has been set equal to 1.0. The probability has been expressed as follows: $p = \exp(-\delta E/T)$. Fig. 5.2 shows both the temperature and probability of energy distributions with the number of iterations. Therefore, the probability

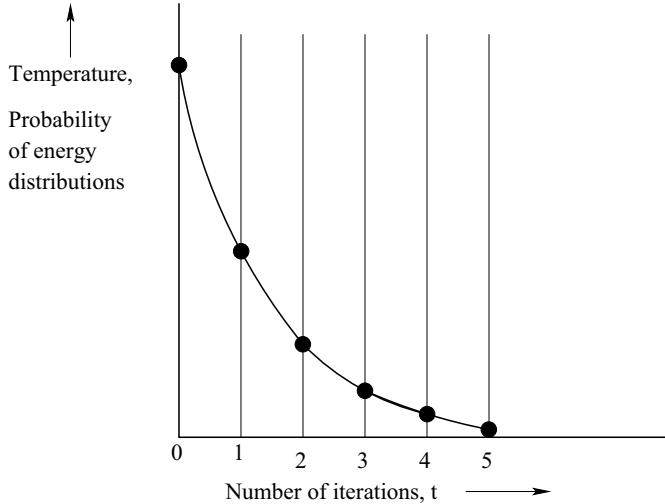


Figure 5.2: A schematic view showing the temperature and probability of energy distributions with the number of iterations.

of choosing bad solutions has been kept equal to a high value at the higher temperature and that is maintained at a low value at the lower temperature.

A Numerical Example:

$$\text{Minimize } y = E(X) = f(x_1, x_2) = x_1 + x_2 - x_1^2 - x_2^2$$

subject to

$$0.0 \leq x_1, x_2 \leq 5.0$$

Use Simulated Annealing to solve this minimization problem. Assume, the initial temperature of molten metal $T_0 = 3000^\circ K$; initial solution selected at random $X_0 = \begin{Bmatrix} 2.5 \\ 2.5 \end{Bmatrix}$ and termination criterion $\epsilon = 0.001$. Let us assume the random numbers varying in the range of $(0.0, 1.0)$ are as follows: 0.3, 0.8, 0.7, 0.6, 0.2, 0.5, 0.6, 0.8, 0.3, 0.5, and so on.

Solution:

Given $T_0 = 3000^\circ K$, $X_0 = \begin{Bmatrix} 2.5 \\ 2.5 \end{Bmatrix}$.

The value of objective function corresponding to X_0 , $f(X_0) = -7.5$.

- **Iteration 1**

Temperature of the molten metal considered in this iteration $T_1 = 50\%T_0 = 1500^\circ K$. Let us generate a candidate solution (lying in the ranges of the variables) for this iteration using the random numbers 0.3 and 0.8 (as given above) as $X_1 = \begin{Bmatrix} 1.5 \\ 4.0 \end{Bmatrix}$.

The value of objective function corresponding to X_1 , $E(X_1) = -12.75$.

As $E(X_1) < E(X_0)$, we select X_1 as the next point.

We calculate the change in energy

$$\delta E = |E(X_1) - E(X_0)| = 5.25$$

As $\delta E > \epsilon$, we go the next iteration.

- **Iteration 2**

Temperature of the molten metal considered in this iteration $T_2 = 50\%T_1 = 750^\circ K$. The next candidate solution is generated using the random numbers 0.7 and 0.6 (as given above) as $X_2 = \begin{Bmatrix} 3.5 \\ 3.0 \end{Bmatrix}$.

The value of objective function corresponding to X_2 , $E(X_2) = -14.75$.

As $E(X_2) < E(X_1)$, we select X_2 as the next point.

We calculate the change in energy

$$\delta E = |E(X_2) - E(X_1)| = 2.0$$

As $\delta E > \epsilon$, we go the next iteration.

• Iteration 3

Temperature of the molten metal considered in this iteration $T_3 = 50\%T_2 = 375^\circ K$.

The next candidate solution is generated using the random numbers 0.2 and 0.5 (as

given above) as $X_3 = \begin{Bmatrix} 1.0 \\ 2.5 \end{Bmatrix}$.

The value of objective function corresponding to X_3 , $E(X_3) = -3.75$.

As $E(X_3) > E(X_2)$, we cannot select X_3 as the next point now and go for the next checking.

The next random number taken from the given set is $r = 0.6$. We calculate the change in energy

$$\delta E = |E(X_3) - E(X_2)| = 11.0$$

We calculate $\exp(\frac{-\delta E}{T_3}) = \exp(\frac{-11.0}{375}) = 0.97$

As $r < 0.97$, we accept X_3 as the next point.

The above procedure continues, till it reaches the termination criterion, and the optimum solution will be obtained.

5.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO), introduced by Kennedy and Eberhart [64] in 1995, is a population-based evolutionary computation technique. It has been developed by simulating bird flocking, fish schooling or sociological behavior of a group of people artificially. Here, the population of solutions is called **swarm**, which is composed of a number of agents known as **particles**. Each particle is treated as a point in d -dimensional search space, which modifies its position according to its own flying experience and that of other particles present in the swarm. The algorithm starts with a population (swarm) of random solutions (particles). Each particle is assigned a random velocity and allowed to move in the problem space. The particles have memory and each of them keeps track of its previous (local) best position (that is, P_{best}). There exist a number of P_{best} for the respective particles in the swarm and the particle with the greatest fitness (goodness) is known as the global best P_g of the swarm. Here, i -th particle in the problem space is represented as $X_i = (x_{i1}, x_{i2}, \dots, x_{id})$, where d indicates the dimensions of the particle. Let the best previous position of i -th particle ($P_{best,i}$) is denoted by $P_i = (p_{i1}, p_{i2}, \dots, p_{id})$ and the global best P_g is represented as $P_g = (p_{g1}, p_{g2}, \dots, p_{gd})$. Let the velocity of i -th particle is denoted by $V_i = (v_{i1}, v_{i2}, \dots, v_{id})$. The updated velocities and positions of i -th particle for its d -th dimension are determined as follows:

$$v_{id}(t+1) = wv_{id}(t) + c_1r_1(p_{id} - x_{id}(t)) + c_2r_2(p_{gd} - x_{id}(t)), \quad (5.1)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1), \quad (5.2)$$

where w is a constant known as inertia weight; c_1 and c_2 represent two positive constants called the cognitive and social parameters, respectively; r_1 and r_2 are random numbers lying the range of $(0.0, 1.0)$; t denotes iteration number; $i = 1, 2, \dots, N$, where N is the population size. The first part of equation (5.1) denotes the previous velocity that provides the necessary momentum to the particles to move across the search space. The second part of this equation is the cognitive component representing personal thinking of i -th particle, which encourages i -th particle to move toward its best position found so far. The third part of this equation is called the social component that indicates the collaborative effect of the particles in finding the globally optimal solution. It pulls the particle toward the globally best particle found so far. Fig. 5.3 shows the flowchart of PSO algorithm.

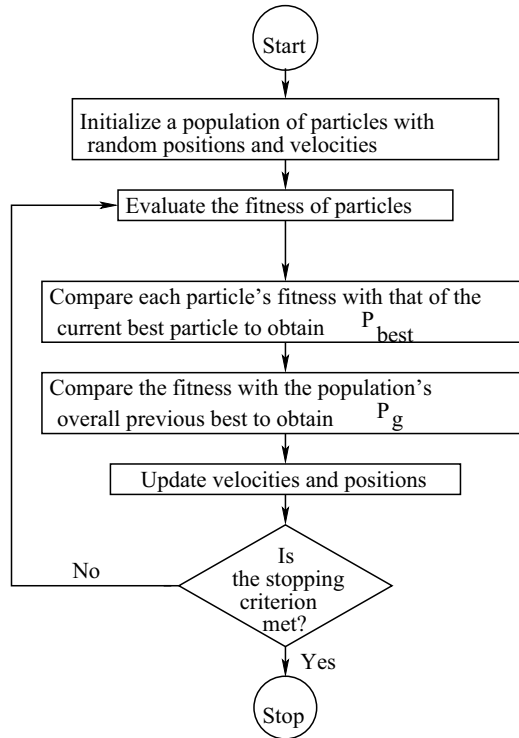


Figure 5.3: The flowchart of PSO algorithm.

It is important to mention that the performance of this algorithm is dependent on the selection of w , c_1 and c_2 . In order to improve its performance, suitable values of these parameters are to be selected iteration-wise in an adaptive way [70].

5.2.1 Comparisons Between PSO and GA

There are some similarities between PSO and GA. Both these algorithms start with a population of solutions generated at random, and the quality of these solutions is expressed in terms of their fitness values. However, there are some dissimilarities also between them. For example, in PSO, there is no operator like crossover or mutation, whereas these are considered as important operators of the GA. In PSO, the particles have memory, and consequently, already found good information of the particles are carried forward iteratively. On the other hand, the previous knowledge of the problem is lost in the GA, once the population changes. A GA is a powerful tool for global optimization. On the other hand, PSO carries out both the global and local searches simultaneously. The PSO algorithm is simpler in construction and faster than the GA. It is expected to provide the better results compared to the GA for most of the optimization problems.

5.3 Summary

The content of this chapter has been summarized as follows:

Here, the working principles of two non-traditional optimization tools have been explained in detail, after providing a brief introduction to other methods. The similarities and dissimilarities between the PSO and GA have been discussed at the end of this chapter.

5.4 Exercise

1. Explain the working principle of simulated annealing as an optimization algorithm.
2. Discuss the principle of PSO algorithm as an optimizer. How does it differ from the GA?
3. Use simulated annealing to solve the optimization problem given below.

$$\text{Minimize } y = E(X) = f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 5)^2$$

subject to

$$0.0 \leq x_1, x_2 \leq 8.0$$

Assume, the initial temperature of molten metal $T_0 = 3600^\circ K$; initial solution selected at random $X_0 = \begin{Bmatrix} 1.0 \\ 1.5 \end{Bmatrix}$ and termination criterion $\epsilon = 0.002$. Let us assume the random numbers varying in the range of $(0.0, 1.0)$ are as follows: 0.2, 0.8, 0.5, 0.4, 0.7, 0.9, 0.3, 0.1, 0.6, 0.3, 0.5, 0.7, 0.9, 0.3, 0.4, 0.7, 0.2, 0.6, and so on. Show three iterations only.

Chapter 6

Multi-Objective Optimization

A number of optimization problems involving a single objective function have been dealt in Chapters 2 through 5 of this book. However, there exist a large number of other optimization problems, in which either two or even more than two objectives are to be considered. These are popularly known as **multi-objective optimization** problems. It is interesting to observe that for some of these problems, the objectives conflict each other. The present chapter deals with some of the approaches used to tackle multi-objective optimization problems.

6.1 Introduction

Let us suppose that a person is planning to purchase a car. Before he takes the decision of which model to be purchased, most probably he considers the factors like cost of a car and its accident rate. If he purchases a cheaper car, its control and braking systems may not be so much good and reliable to ensure a low accident rate. On the other hand, the chance of accident may be less for a costlier car. However, a person wants to purchase a car, which is cheap but at the same time, its chance of accident is less. Thus, these two factors (each of which depends on a number of common decision variables) oppose each other. Fig. 6.1 displays the above fact that if the cost is more, the accident rate becomes less and vice-versa. Therefore, a number of feasible solutions may exist on the optimal front. It is known as **Pareto-optimal front** of solutions, according to the name of Vilfredo Pareto. A person has the option now to choose a particular solution out of a number of feasible ones.

To further investigate the nature of Pareto-optimal front of solutions, let us consider two objective functions: f_1 and f_2 , both of which are to be minimized, but their relationships are such that if one increases, the other one is bound to decrease and vice-versa. Thus, Figs. 6.2(a), (b) and (c) obey the above conditions and therefore, display the valid Pareto-optimal fronts of solutions. However, if both f_1 and f_2 either increase (refer to Fig. 6.2(d)) or decrease (as shown in Fig. 6.2(e)), valid Pareto-optimal fronts of solutions will not be obtained.

Note: Any optimization problem having two or more than two objectives may not give

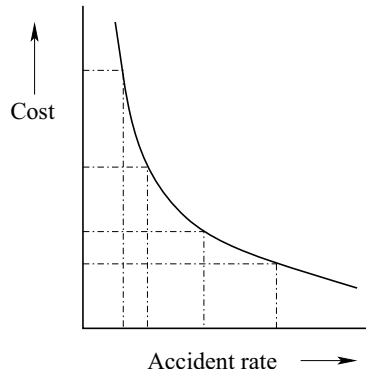


Figure 6.1: A schematic view showing cost vs. accident rate of a car.

rise to valid Pareto-optimal front(s) of solutions. In order to ensure a valid Pareto-optimal front of solutions, two objectives should oppose each other.

6.2 Some Approaches to Solve Multi-Objective Optimization Problems

A number of approaches had been developed by various investigators to solve multi-objective optimization problems. Some of these approaches are discussed below.

6.2.1 Weighted Sum Approach

Let us consider a minimization problem involving N objectives, whose function values are denoted by $f_i(X)$, where $i = 1, 2, \dots, N$. Let us also assume that each objective is a function of m independent decision variables, that is, $X = (x_1, x_2, \dots, x_m)^T$. This multi-objective optimization problem can be converted into a single-objective one using some weighting factors as given below [71].

$$\text{Minimize } \sum_{i=1}^N w_i f_i(X),$$

subject to

$$X^{min} \leq X \leq X^{max},$$

where w_i are the weighting factors, which vary in the range of (0.0, 1.0), and $\sum_{i=1}^N w_i = 1.0$. Thus, an optimal (that is, minimum here) value of the above weighted objective function can be obtained for a set of w_i values by using an optimizer. It is to be noted that the obtained optimal solution is dependent on the selection of weighting factors [72].

It is easy to implement and found to be efficient. However, it does not provide the complete Pareto-optimal front of solutions directly, which can be indirectly determined by considering different combinations of weighting factors one after another.

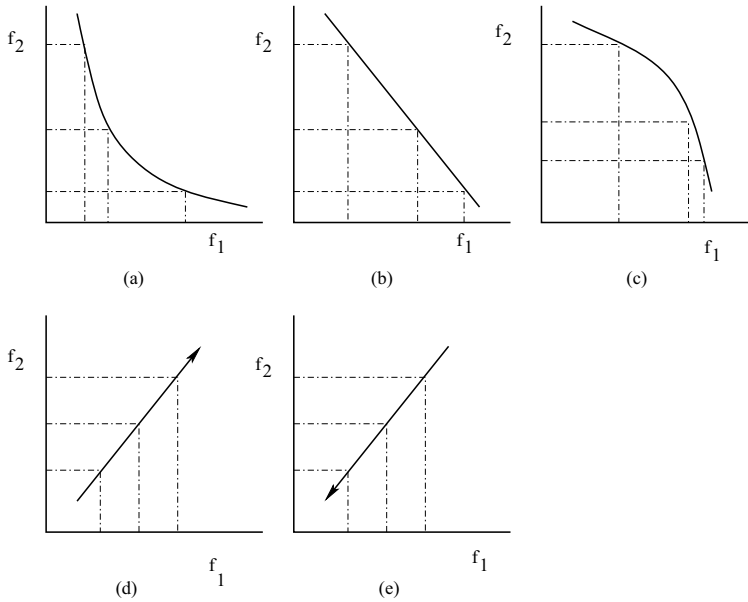


Figure 6.2: Various possibilities of the relationship between two objectives: f_1 and f_2 .

6.2.2 Goal/Target Programming

Let us assume an optimization problem of N objective functions represented by $f_i(X)$, where $i = 1, 2, \dots, N$, and each of them is a function of m independent variables, that is, $X = (x_1, x_2, \dots, x_m)^T$.

A decision maker assigns the targets or goals (T_i) that he wants to achieve for each of N objectives first, and then the sum of the absolute deviations of the objectives from their respective targets is calculated as $\sum_{i=1}^N |f_i(X) - T_i|$. A minimization problem is then formulated as follows:

$$\text{Minimize } \sum_{i=1}^N |f_i(X) - T_i|$$

subject to

$$X^{min} \leq X \leq X^{max}.$$

The main difficulty of this approach lies with the inability of the decision maker to set the target values T_i , beforehand.

6.2.3 Vector Evaluated Genetic Algorithm (VEGA)

To solve multi-objective optimization problem, Vector Evaluated Genetic Algorithm (VEGA) was proposed by Schaffer in 1985 [73]. It is nothing but the weighted sum approach implemented through a GA, in which the weighting factors are selected artificially. It does not provide the Pareto-optimal front of solutions directly for a multi-objective optimization

problem. Let us consider an optimization problem with q objectives. This approach consists of the following steps:

- **Step 1:** An initial population of solutions (of size N) of the GA is created at random.
- **Step 2:** The whole population of size N is divided into q (that is, number of objectives) number of sub-populations using a proportionate selection scheme. Each sub-population of size equal to $\frac{N}{q}$ is formed corresponding to a particular objective.
- **Step 3:** Shuffling is done to obtain a new population of size N by putting some weights on different objectives, artificially. It is to be noted that in shuffling, only the positions of the strings are changed. It is also important to mention that a multi-objective optimization problem is thus converted into a single-objective one by using some weighting factors.
- **Step 4:** Crossover operator is used to generate the children solutions.
- **Step 5:** Mutation operator is utilized to further modify the population of solutions (strings).

Thus, one generation of the GA is completed. The population of GA-strings will be modified and split into q sub-populations, each of them becomes strong particularly in one of the objectives through a number of generations.

6.2.4 Distance-based Pareto-GA (DPGA)

It was proposed by Osyczka and Kundu in 1995 [74], in which the Pareto-optimal solutions (that is, non-dominated solutions) are updated gradually using the fitness information with respect to the objective functions. To explain its principle, let us consider multi-objective optimization problems involving q objectives and each objective is a function of m variables. The problem may be stated as follows:

$$\text{Minimize } f_i(X), i = 1, 2, \dots, q, \quad (6.1)$$

subject to

$$X^{min} \leq X \leq X^{max},$$

where $X = (x_1, x_2, \dots, x_m)^T$.

This approach consists of the following steps:

- **Step 1:** Generate an initial population of solutions of size N , at random. The first solution is assigned a fitness F (also called starting distance) arbitrarily and copied into the Pareto-optimal/non-dominated set. Thus, the population of solutions of size N is divided into two sub-populations: non-dominated set N_1 and dominated set N_2 . Let us denote j -th solution of the non-dominated set N_1 by the symbol: P_j (where $j = 1, 2, \dots, J$), which has q objective function values represented by $P_{j1}, P_{j2}, \dots, P_{jq}$.

- **Step 2:** To decide whether a particular solution X will be put in the non-dominated set N_1 , its distance is calculated from the existing non-dominated solution(s) (say, j -th) as follows:

$$d_{j,X} = \sqrt{\sum_{k=1}^q \left(\frac{P_{jk} - f_k(X)}{P_{jk}} \right)^2} \quad (6.2)$$

The values of $d_{j,X}$ are calculated and then compared. The minimum of $d_{j,X}$ values is represented by d_{min} and the corresponding value of j is denoted by j^* .

If the solution X is found to be non-dominated (in terms of at least one of the objectives) with respect to the present non-dominated set of solutions, it is included in the Pareto-optimal set of solutions, and its fitness (F) is calculated as follows:

$$F(X) = F(P_{j^*}) + d_{min} \quad (6.3)$$

The non-dominated set of solutions is then updated by deleting the solutions dominated by X .

However, if the solution X is seen to be dominated by the present non-dominated set of solutions, it is not selected for the Pareto-optimal set and it is assigned a fitness of $F(P_{j^*}) - d_{min}$. As this fitness cannot be negative, it is set equal to 0.0, if $(F(P_{j^*}) - d_{min})$ is found to be negative.

- **Step 3:** The fitness values are thus calculated for the whole population of solutions. The maximum value of fitness (F_{max}) is then determined for all the existing solutions of non-dominated set. All the non-dominated solutions are then assigned the fitness equal to F_{max} .
- **Step 4:** The population of solutions is modified using the operators, namely reproduction, crossover and mutation.

One generation of the GA is thus completed, and it continues till it reaches the termination criterion.

6.2.5 Non-dominated Sorting Genetic Algorithms (NSGA)

Let us consider a two-objective optimization problem, where both the objectives: f_1 and f_2 are to be minimized. A Pareto-optimal front of solutions can be obtained through a number of generations for this problem using Non-dominated Sorting Genetic Algorithm (NSGA), as proposed by Srinivas and Deb [75]. Its working principle is described below, in brief.

- **Step 1:** Generate an initial population of solutions (N), at random. If a particular solution is found to be better than another in terms of objective(s), the former is called a non-dominated one. Each solution of the population is thus declared as either non-dominated or dominated one. The solutions lying in the non-dominated group are assigned rank 1. The dominated solutions are once again sorted into two

groups, namely non-dominated and dominated ones. These non-dominated solutions are assigned rank 2. The whole population of solutions is thus sorted into different ranks like 1, 2, 3, and so on. It is obvious that the average fitness of rank 1 solutions will be better than that of rank 2 solutions. Similarly, rank 2 solutions are better than the rank 3 solutions, in terms of average fitness value. It is to be noted that all rank 1 solutions are assigned fitness numerically equal to the population size N .

- **Step 2:** Use reproduction operator to form the mating pool, where the number of rank 1 solutions is expected to be more. Thus, the selection pressure is maintained.

In order to maintain diversity in the population, the concept of sharing is used. Let us assume that the whole population of solutions has been sorted into the ranks: 1, 2 and 3, as shown in Fig. 6.3. Let us also consider that the rank 1 front consists of 13

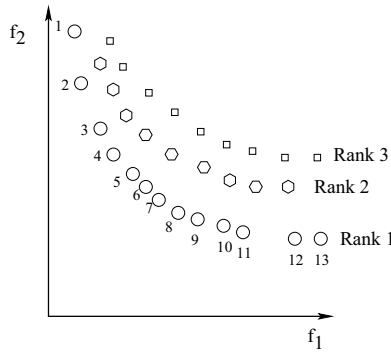


Figure 6.3: A schematic view showing the sorting of solutions into different ranks.

solutions, out of which the solutions: 1, 2, 12 and 13 are far away from the others. If by chance, the solutions: 1, 2, 12 and 13 are lost from the population, a significant part of the front will be lost. To overcome this problem, the concept of sharing is used. A sharing function is defined as follows:

$$Sh(d_{ij}) = \begin{cases} 1.0 - (\frac{d_{ij}}{\sigma_{share}})^2, & \text{if } d_{ij} < \sigma_{share}, \\ 0.0, & \text{otherwise,} \end{cases} \quad (6.4)$$

where d_{ij} represents the euclidean distance between two points: i and j , and a fixed value is assigned to σ_{share} . If d_{ij} is found to be less than σ_{share} , the sharing function $Sh(d_{ij})$ will have a non-zero value. Otherwise, $Sh(d_{ij})$ is assigned a value equal to 0.0. Moreover, if two points coincide, the value of d_{ij} becomes equal to 0.0, and consequently, the sharing function will take a value equal to 1.0. A parameter called niche count is then calculated, which is nothing but the summation of sharing function values. Therefore, the value of niche count will be less for the points: 1, 2, 12 and 13, and those for the other points like 3, 4, ..., 10, 11 will turn out to be more. A shared fitness is then calculated by dividing the individual fitness by niche count. Thus, the shared fitness of the points: 1, 2, 12 and 13 will be more compared to the other points.

Now, if the selection is carried out based on these values of shared fitness, there is a chance of having more copies of the solution points: 1, 2, 12 and 13 in the mating pool. Therefore, these points will not be lost from the population and the desired diversity will be maintained.

- **Step 3:** The population of solutions is modified using the operators like crossover and mutation.

One generation of the GA-run is thus completed and it continues, till the termination criterion is reached. The Pareto-optimal front of solutions will be updated iteratively through a number of generations, and ultimately, the desired Pareto-optimal front of solutions will be obtained.

This algorithm received the following criticisms:

- High computational complexity of non-dominated sorting,
- Need for the selection of a fixed value of σ_{share} ,
- Lack of elitism.

The above weak points of this algorithm were removed and a modified version of it was proposed later on by Deb et al. [76], which is popularly known as NSGA-II. In this algorithm, the concept of sharing function has also been replaced by that of crowding-distance, which is explained below. Fig. 6.4 displays a schematic view related to crowding-distance calculation. In order to calculate the value of crowding-distance of i -th point lying on the

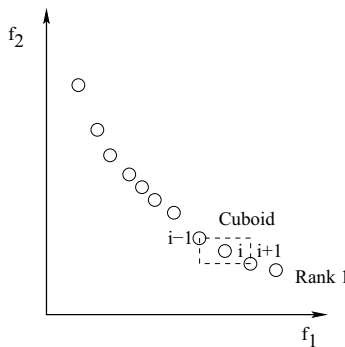


Figure 6.4: A schematic view related to crowding-distance calculation.

rank 1 front, its two nearest neighbors, such as $(i - 1)$ -th and $(i + 1)$ -th are considered. A cuboid is then formed as shown in Fig. 6.4, and its perimeter is considered as its crowding distance value. The solutions lying on the extreme positions (that is, at the beginning and end positions) of the front are assumed to have very high crowding distance values. On the other hand, the points of more crowded regions will have low crowding distance values. During the selection, if the two solutions belong to two different ranks, then one with the lower rank will be selected. However, if they belong to the same rank, then the solution

lying in the less crowded region (which has a high crowding distance value) is preferred during the selection.

Note: A huge literature is available on multi-objective optimization. Interested readers may refer to some other algorithms like Fonseca and Fleming's MOGA [77], Niche Pareto Genetic Algorithm (NPGA) [78], Pareto-Archived Evolution Strategy (PAES) [79], Strength-Pareto Evolutionary Algorithm (SPEA) [80], and others. A detailed discussion on these algorithms is beyond the scope of this book. Interested readers may refer to Deb's book [34] for the detailed descriptions of various algorithms of multi-objective optimization.

6.3 Summary

This chapter has been summarized as follows:

It deals with multi-objective optimization problems. The concept of Pareto-optimal front of solutions in connection with these problems has been explained in detail. The working principles of some algorithms, such as Weighted Sum Approach, Goal/Target programming, VEGA, DPGA and NSGA have been discussed in this chapter.

6.4 Exercise

1. Define Pareto-optimal front of solutions in connection with multi-objective optimization.
2. Explain the working principles of the following algorithms used in multi-objective optimization:
 - (a) Weighted Sum Approach
 - (b) Goal/Target programming
 - (c) Vector Evaluated GA (VEGA)
 - (d) Distance-based Pareto-GA (DPGA)
 - (e) Non-dominated Sorting GA (NSGA)
3. Any two-objective optimization problem may not be a candidate problem for determining the Pareto-optimal front of solutions – justify the statement.
4. Can NSGA-II provide better solutions compared to NSGA for solving the multi-objective optimization problems? Explain.

5. Use Distance-based Pareto-GA (DPGA) to update non-dominated front of solutions for a few solution points: (2.0, 5.0), (6.5, 2.5), (8.0, 1.5), (3.2, 7.8) through hand-calculations in order to solve the two-objective optimization problem as given below.

$$\begin{aligned} \text{Minimize } f_1(x_1, x_2) &= x_1 + x_2, \\ \text{Minimize } f_2(x_1, x_2) &= \frac{1}{x_1} + \frac{1}{x_2} \end{aligned}$$

subject to

$$1.0 \leq x_1, x_2 \leq 10.0$$

Chapter 7

Introduction to Fuzzy Sets

The aim of this chapter is to introduce the concept of fuzzy set and find its relationship with crisp set (also known as classical set).

7.1 Crisp Sets

To define the term: **crisp set**, let us first concentrate on the concept of **Universal set** (also called the **universe of discourse**). A **Universal set** (represented by X) is a set consisting of all possible elements (also known as members) related to a particular context. For example, if our aim is to investigate on the technical universities, then all the technical universities in the world may be assumed to form a Universal set. It is important to mention that a number of sets can be derived from the Universal set. Let us try to find the set of technical universities with only three departments each. To obtain the above set, we check with each university whether it belongs to that set. Thus, there is a fixed and well-defined boundary between the elements of that set and those lying outside it. The above set with a fixed boundary (represented by solid line) is known as a crisp set.

A set is generally represented by its elements: Let us assume that there are n technical universities, each of which has three departments. The set of technical universities with three departments each is denoted by

$$A = \{a_1, a_2, \dots, a_n\} \quad (7.1)$$

A set can also be represented by the property of its element like the following:

$$A = \{x | P(x)\}, \quad (7.2)$$

that is, set A contains all x belonging to X that have the property P .

For example: $A = \{x, \text{ such that they have three technical departments each}\}$.

A set A can also be indicated by its characteristic function as follows:

$$\mu_A(x) = \begin{cases} 1, & \text{if } x \text{ belongs to } A, \\ 0, & \text{if } x \text{ does not belong to } A. \end{cases} \quad (7.3)$$

Thus, $\mu_A(x)$ has any one of two values: 1 (true) or 0 (false).

7.1.1 Notations Used in Set Theory

The following notations are used in the crisp sets:

1. \emptyset : It indicates an **empty set** (also known as **null set**) that does not contain any element.
2. $x \in A$: It states that an element x of the Universal set X belongs to set A .
3. $x \notin A$: It represents a condition that an element x of the Universal set X does not belong to set A .
4. $A \subseteq B$: It is used to indicate that set A is a **subset** of set B , that is, each element of A is also an element of B but the reverse may not be true.
5. $A \supseteq B$: It indicates that set A is a **superset** of set B , that is, each element of B is also an element of A but the reverse may not be true.
6. $A = B$: It states that the sets: A and B are **equal**, that is, both of them contain the same elements. If $A \subseteq B$ and $B \subseteq A$, then $A = B$.
7. $A \neq B$: It says that the sets: A and B are **not equal**.
8. $A \subset B$: It is used to represent that set A is a **proper subset** of set B , for which both the conditions: $A \subseteq B$ and $A \neq B$ are to be true and consequently, B contains at least one element that is missing in A .
9. $A \supset B$: It states that the set A is a **proper superset** of set B , for which both the conditions: $A \supseteq B$ and $A \neq B$ are to be satisfied. It means that set A contains at least one element that is missing in set B .
10. $|A|$: It indicates the **cardinality** of set A , which is defined as the total number of elements present in that set. For example, in case of a singleton set $A = \{a_1\}$, $|A| = 1$.
11. $p(A)$: It represents the **power set** of A , which is nothing but the maximum number of subsets including the null that can be constructed from the set A . Let us consider that set A contains three elements, that is, $A = \{a_1, a_2, a_3\}$. The subsets (8 in number), those can be constructed from the set A are: $\{a_1\}$, $\{a_2\}$, $\{a_3\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, $\{a_2, a_3\}$, $\{a_1, a_2, a_3\}$, $\{\emptyset\}$. Now, the cardinality of set A , that is, $|A| = 3$ and that of $p(A)$ is given by $|p(A)| = 2^{|A|} = 2^3 = 8$.

7.1.2 Crisp Set Operations

A few set operations are explained below.

- **Difference:** The difference between two sets: A and B is represented as follows:

$$A - B = \{x | x \in A \text{ and } x \notin B\} \quad (7.4)$$

It is also known as **relative complement** of set B with respect to set A , which contains all the elements of A , those are not included in B (refer to Fig. 7.1(a)).

Absolute complement of a set A is indicated by \bar{A} or A^c and it is defined as the

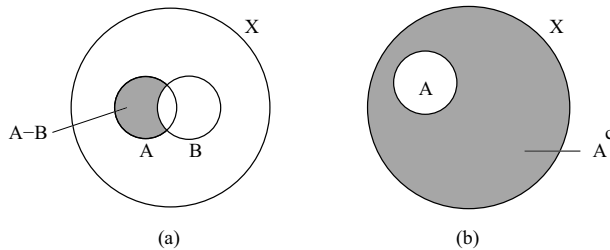


Figure 7.1: The diagrams showing the complements of a set: (a) relative complement, (b) absolute complement.

difference between the Universal set X and set A as given below (refer to Fig. 7.1(b)).

$$\bar{A} = A^c = X - A = \{x | x \in X \text{ and } x \notin A\}. \quad (7.5)$$

Notes: It is important to note the following points:

1. Complement of a complement yields the original set.
2. Absolute complement of an empty set is nothing but the Universal set.
3. Absolute complement of the Universal set yields an empty set.

Example: Let $A = \{a, c, d, f, r, g, h\}$ and $B = \{c, d, r, m, n\}$.

Therefore, $A - B = \{a, f, g, h\}$.

- **Intersection:** Intersection of two sets, namely A and B is denoted by $A \cap B$ and it is defined as follows:

$$A \cap B = \{x | x \in A \text{ and } x \in B\} \quad (7.6)$$

Fig. 7.2 shows the intersection of two sets: A and B , as the shaded area.

Two sets: A and B are called disjoint, if there is no common element between them and it is indicated by $A \cap B = \emptyset$.

Example:

Let $A = \{a, c, d, f, r, g, h\}$ and $B = \{c, d, r, m, n\}$.

Therefore, $A \cap B = \{c, d, r\}$.

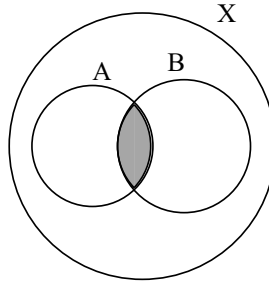


Figure 7.2: A diagram showing the intersection of two sets.

- **Union:** Union of two sets, such as A and B is represented by $A \cup B$ and it is defined like the following:

$$A \cup B = \{x | x \in A \text{ or } x \in B\} \quad (7.7)$$

The union of two sets: A and B is shown in Fig. 7.3, as the shaded area.

Example:

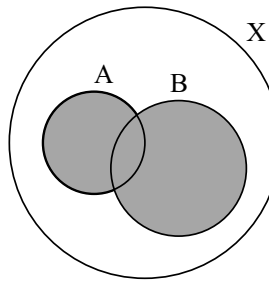


Figure 7.3: A diagram showing the union of two sets.

Let $A = \{a, c, d, f, r, g, h\}$ and $B = \{c, d, r, m, n\}$.

Therefore, $A \cup B = \{a, c, d, f, r, g, h, m, n\}$.

7.1.3 Properties of Crisp Sets

Let us consider some crisp sets: A , B and C defined in the Universal set X . The following properties of the crisp sets are important to mention:

1. **Law of involution:**

$$\overline{\overline{A}} = A$$

2. **Laws of commutativity:**

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

3. Laws of associativity:

$$(A \cup B) \cup C = A \cup (B \cup C)$$
$$(A \cap B) \cap C = A \cap (B \cap C)$$

4. Laws of distributivity:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

5. Laws of idempotence/tautology:

$$A \cup A = A$$
$$A \cap A = A$$

6. Laws of absorption:

$$A \cup (A \cap B) = A$$
$$A \cap (A \cup B) = A$$

7. Laws of identity:

$$A \cup X = X$$
$$A \cap X = A$$
$$A \cup \emptyset = A$$
$$A \cap \emptyset = \emptyset$$

8. De Morgan's Laws:

$$\overline{(A \cup B)} = \overline{A} \cap \overline{B}$$
$$\overline{(A \cap B)} = \overline{A} \cup \overline{B}$$

9. Law of contradiction:

$$A \cap \overline{A} = \emptyset$$

10. Law of excluded middle:

$$A \cup \overline{A} = X$$

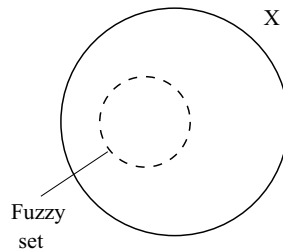


Figure 7.4: A schematic representation of fuzzy set.

7.2 Fuzzy Sets

Fuzzy sets are the sets with imprecise (vague) boundaries. Let us assume that we will have to find the set of technical universities with three **good** departments each. It becomes a set with imprecise boundary, which is popularly known as fuzzy set. Fig. 7.4 displays a fuzzy set with its boundary denoted by dotted line. The concept of fuzzy set was introduced by Prof. L.A. Zadeh of the University of California, USA, in 1965 [81], although an idea related to it was visualized by Max Black, an American Philosopher, in 1937 [82]. Prof. Zadeh's paper was a challenge over the crisp set theory, the reasons for which are discussed below.

Real-world problems are generally associated with different types of uncertainties (such as vagueness, imprecision). In the past, a considerable amount of effort was made to model those uncertainties. Prior to 1965, people used to consider **probability theory** (which works based on Aristotelian two-valued logic) as the prime agent for dealing with uncertainties. It is to be noted that the above logic uses the concept of classical or crisp set theory. Prof. Zadeh argued that probability theory can handle only one out of several different types of possible uncertainties. Thus, there are some uncertainties, which cannot be tackled using the probability theory. Let us take one example, in which Mr. A requests Mr. B, one of his friends, to bring some **red** apples for him from the market. There are two uncertainties at least, which are related to the following: (i) availability of the apples and (ii) a guarantee that the apple is red. Depending on the season, there is a probability (that is, the frequency of likelihood that an element is in a class) of obtaining the apples, which varies between 0.0 and 1.0. According to the crisp set, the apples will be either red (1) or non-red (0). On the other hand, in fuzzy set, the colour: **red** can be defined as follows (refer to Fig. 7.5) using the concept of membership (that is, similarity of an element to a class) function value (μ): If the colour is Perfectly Red (PR), then it may be said red with a membership value of 1.0, if it is Almost Red (AR), then it is considered as red with a membership value of 0.8, if it is Slightly Red (SR), then it is assumed to be red with a membership value of 0.4, even if it is Not Red (NRD), then also it is called red with a membership value of 0.0, and so on. Thus, in fuzzy set, an element can be a member of the set with some membership value (that is, degree of belongingness). In this way, the uncertainty related to colour of the apples can be handled. Thus, a fuzzy set is considered to be a more general concept of the classical set. It is to be noted that the characteristic function used in crisp set has been renamed as the membership function in fuzzy set.

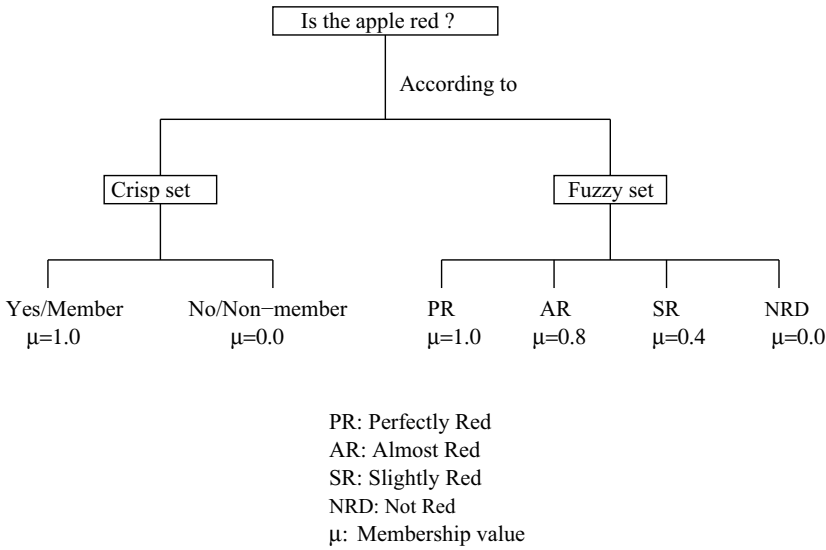


Figure 7.5: A diagram used for explaining the concept of crisp set and fuzzy set.

Note: Belief measure of a set A (denoted by $bel(A)$) is nothing but its degree of support or certainty or evidence [83]. On the other hand, **plausibility measure** of the set A (represented by $pl(A)$) is defined as the complement of the belief of the complement of A , that is, $pl(A) = 1 - bel(\bar{A})$. Now, **ignorance** is determined as $1 - \{bel(A) + bel(\bar{A})\}$. If $bel(A) + bel(\bar{A}) = 1$, then **ignorance** turns out to be equal to 0 and **belief** becomes equal to **plausibility** (that is, $bel(A) = pl(A)$). Here, we can use **probability** measure. On the other hand, if **ignorance** becomes non-zero, we will have to use **plausibility** (also known as **possibility**) measure, which is equivalent to fuzzy set.

7.2.1 Representation of a Fuzzy Set

A fuzzy set $A(x)$ is represented by a pair of two things: the first one is the element x and the second one is its membership value $\mu_A(x)$, as given below.

$$A(x) = \{(x, \mu_A(x)), x \in X\} \quad (7.8)$$

It may be either discrete or continuous in nature, as discussed below.

Discrete Fuzzy Set:

If a fuzzy set $A(x)$ is discrete in nature, it can be expressed using the membership values of its elements. Thus, the set $A(x)$ is denoted like the following:

$$A(x) = \sum_{i=1}^n \mu_A(x_i)/x_i, \quad (7.9)$$

where n is the number of elements present in the set.

Let us take an example, in which atmospheric temperature during the first fifteen days of a month has been found to be **medium**. This **medium** temperature (M) has been

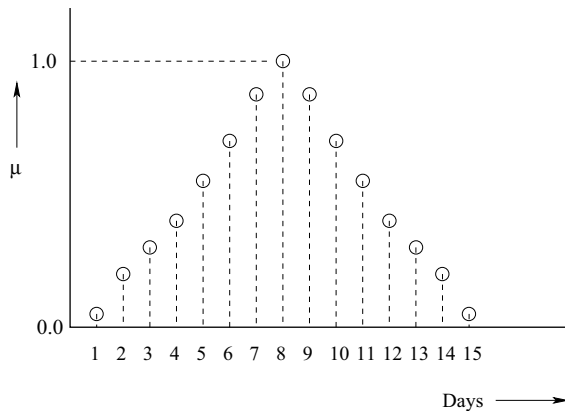


Figure 7.6: Representation of a discrete fuzzy set (that is, medium atmospheric temperature).

represented as follows:

$$M = 0.05/1 + 0.2/2 + 0.3/3 + 0.4/4 + 0.55/5 + 0.7/6 + 0.875/7 + 1.0/8 + 0.875/9 + 0.7/10 + 0.55/11 + 0.4/12 + 0.3/13 + 0.2/14 + 0.05/15.$$

The + sign does not indicate any algebraic sum but collection of the pairs of elements and their membership values used to represent a fuzzy set. Fig. 7.6 shows the above representation in a plot. It is important to note that it can also be expressed as follows:

$$M = \{(1, 0.05), (2, 0.2), (3, 0.3), (4, 0.4), (5, 0.55), (6, 0.7), (7, 0.875), (8, 1.0), (9, 0.875), (10, 0.7), (11, 0.55), (12, 0.4), (13, 0.3), (14, 0.2), (15, 0.05)\}.$$

Continuous Fuzzy Set:

A continuous fuzzy set $A(x)$ is expressed mathematically like the following:

$$A(x) = \int_X \mu_A(x)/x. \quad (7.10)$$

It is important to mention that an integral sign is utilized here to indicate the collection of the pairs of elements and their membership values used to represent a continuous fuzzy set. It is represented with the help of a membership function distribution. Before we discuss the different types of membership function distributions used generally, let us define the concept of a **convex** fuzzy set.

A fuzzy set $A(x)$, such that $x \in X$ (where X is the Universal set) will be convex in nature, if the following condition is satisfied:

$$\mu_A\{\lambda x_1 + (1 - \lambda)x_2\} \geq \min\{\mu_A(x_1), \mu_A(x_2)\}, \quad (7.11)$$

where x_1 and x_2 are two values of x lying within X , λ is a constant lying in the range of $(0.0, 1.0)$, that is, $0.0 \leq \lambda \leq 1.0$. Fig. 7.7 shows the convex and non-convex membership

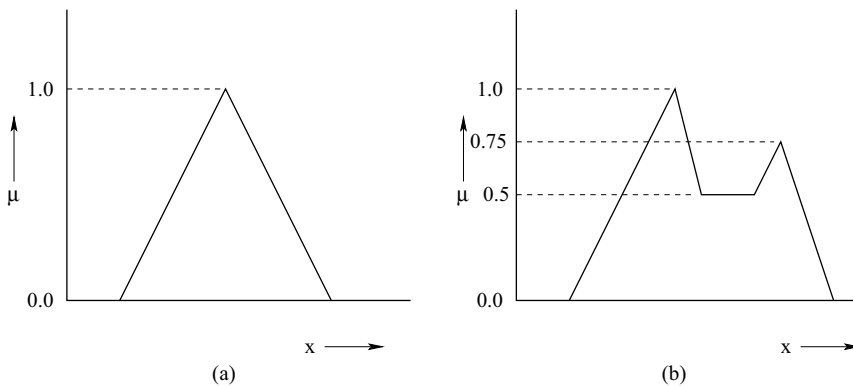


Figure 7.7: Schematic view showing (a) convex and (b) non-convex membership function distributions.

function distributions. Various types of membership function distributions are in use, some of them are discussed below.

- **Triangular Membership:** It is denoted by **Triangle (x ; a , b , c)** (refer to Fig. 7.8). The membership function value ($\mu_{triangle}$) for this distribution is determined as fol-

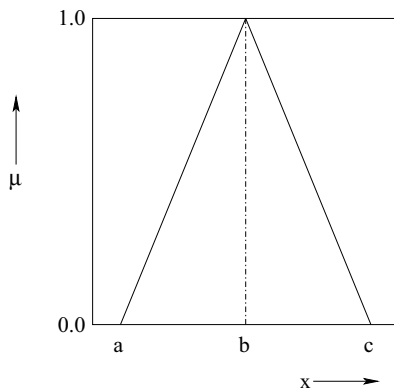


Figure 7.8: Triangular membership function distribution.

lows:

$$\mu_{triangle} = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right). \quad (7.12)$$

Thus, $\mu_{triangle}$ values are set equal to 0.0, 1.0 and 0.0 at $x = a$, $x = b$ and $x = c$, respectively.

- **Trapezoidal Membership:** Fig. 7.9 shows a trapezoidal membership function distribution, that is represented as **Trapezoidal (x ; a , b , c , d)**. The membership value

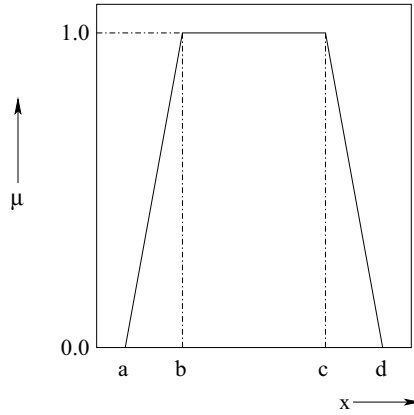


Figure 7.9: Trapezoidal membership function distribution.

$\mu_{trapezoidal}$ can be expressed like the following:

$$\mu_{trapezoidal} = \max(\min(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}), 0). \quad (7.13)$$

The membership function values at $x = a$, $x = b$, $x = c$ and $x = d$ are set equal to 0.0, 1.0, 1.0 and 0.0, respectively.

- **Gaussian Membership:** The Gaussian membership function distribution shown in Fig. 7.10, is indicated by **Gaussian** ($\mathbf{x}; \mathbf{m}, \sigma$), where m and σ represent the mean and standard deviation of the distribution, respectively. The membership function

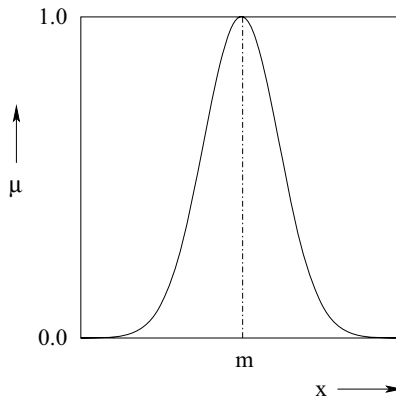


Figure 7.10: Gaussian membership function distribution.

$\mu_{Gaussian}$ is represented as follows:

$$\mu_{Gaussian} = \frac{1}{e^{\frac{1}{2}(\frac{x-m}{\sigma})^2}}. \quad (7.14)$$

At the point $x = m$, $\mu_{Gaussian}$ comes out to be equal to 1.0 and as the value of x deviates more and more from m , the value of $\mu_{Gaussian}$ tends towards 0.0.

- **Bell-shaped Membership Function:** A bell-shaped membership function distribution (refer to Fig. 7.11) is represented as **Bell-shaped (x ; a , b , c)** and it is expressed as follows:

$$\mu_{Bell-shaped} = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}, \quad (7.15)$$

where a controls the width of the function, b (a positive number) indicates the slope of the distribution and c is the center of the function.

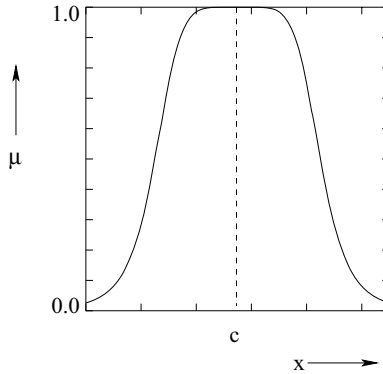


Figure 7.11: Bell-shaped membership function distribution.

- **Sigmoid Membership:** A sigmoid function (shown in Fig. 7.12) can also be used as membership function of a fuzzy set. It is denoted by **Sigmoid (x ; a , b)** and its membership function $\mu_{sigmoid}$ is expressed as follows:

$$\mu_{sigmoid} = \frac{1}{1 + e^{-a(x-b)}}, \quad (7.16)$$

where a indicates the slope of the distribution. Thus, $\mu_{sigmoid}$ becomes equal to 0.5, at $x = b$. Moreover, it tends towards 0.0 and 1.0, as x approaches to 0.0 and a value higher than b , respectively.

7.2.2 Difference Between Crisp Set and Fuzzy Set

Let us take another example to distinguish the concept of crisp set from that of fuzzy set, in which Mr. Y requests Mr. Z to meet him at 7.00 pm. Fig. 7.13 shows the difference between the crisp pm and fuzzy sets. According to the crisp set theory, if Mr. Z can meet Mr. Y exactly at 7.00 pm, then it is **true** (generating full membership, that is, 1.0); otherwise it is **false** (yielding non-membership, that is, 0.0). On the other hand, the time 7.00 pm can be expressed within a range according to its representation in the fuzzy set (refer to Fig. 7.13), in which different times will have different membership function values lying in

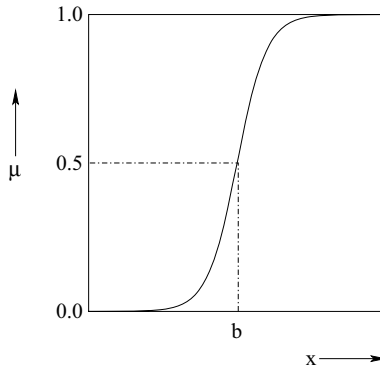


Figure 7.12: Sigmoid membership function distribution.

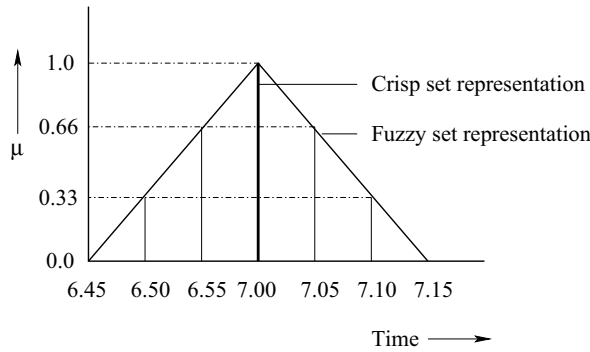


Figure 7.13: A schematic view showing the difference between crisp and fuzzy sets.

the range of 0.0 to 1.0. Thus, if Mr. Z meets Mr. Y at 7.05 pm, then also it will be said that he has met his friend at 7.00 pm with the membership value $\mu = 0.666$, and so on. A crisp set has a fixed boundary, whereas the boundary of a fuzzy set is vague in nature.

7.2.3 A Few Definitions in Fuzzy Sets

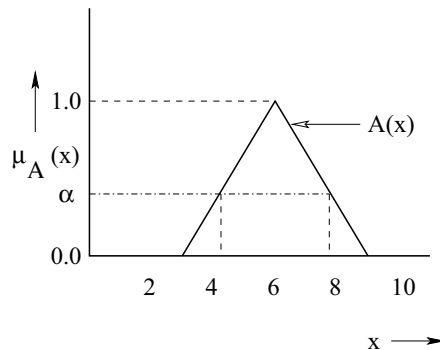
Some important terms related to fuzzy sets are defined below.

1. **α -cut of a fuzzy set $A(x)$:** It is a set consisting of elements x of the Universal set X , whose membership values are either greater than or equal to the value of α (refer to Fig. 7.14). It is denoted by the symbol ${}^{\alpha}\mu_A(x)$. Thus, α -cut of a fuzzy set $A(x)$ is expressed as follows:

$${}^{\alpha}\mu_A(x) = \{x | \mu_A(x) \geq \alpha\}. \quad (7.17)$$

Similarly, strong α -cut of a fuzzy set $A(x)$ is represented like the following:

$${}^{\alpha+}\mu_A(x) = \{x | \mu_A(x) > \alpha\}. \quad (7.18)$$

Figure 7.14: α -cut of a fuzzy set.**Example:**

The membership function distribution of a fuzzy set is assumed to follow a Gaussian distribution with mean $m = 100$ and standard deviation $\sigma = 20$. Determine 0.6-cut of this distribution.

The Gaussian membership function distribution can be expressed as follows:

$$\mu = \frac{1}{e^{\frac{1}{2}\left(\frac{x-m}{\sigma}\right)^2}}$$

After substituting the value of $\mu = 0.6$ in the above equation and taking $\log(\ln)$ on both the sides, we get

$$\ln e^{\frac{1}{2}\left(\frac{x-100}{20}\right)^2} = \ln 1.6667$$

We simplify the above relationship and solve to get the range of x , which is found to be as follows: (79.7846, 120.2153). It is called the 0.6-cut (refer to Fig. 7.15).

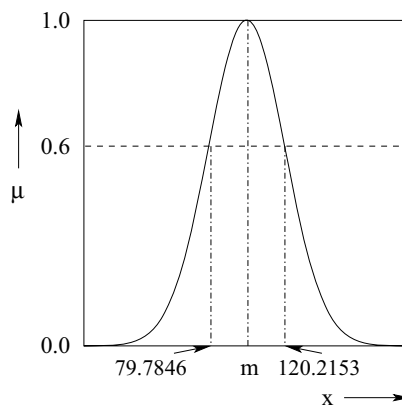


Figure 7.15: 0.6-cut of a fuzzy set.

2. **Support of a fuzzy set $A(x)$:** It is defined as the set of all $x \in X$, such that $\mu_A(x) > 0$. It is indicated by **supp(A)** and can be represented as follows:

$$\text{supp}(A) = \{x \in X | \mu_A(x) > 0\}. \quad (7.19)$$

Thus, the support of a fuzzy set is nothing but its **strong 0-cut**.

3. **Scalar cardinality of a fuzzy set $A(x)$:** It is denoted by $|A(x)|$ and can be defined like the following:

$$|A(x)| = \sum_{x \in X} \mu_A(x) \quad (7.20)$$

Example:

Let us consider a fuzzy set $A(x)$ as follows:

$$A(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\}$$

Its cardinality $|A(x)| = 0.1 + 0.2 + 0.3 + 0.4 = 1.0$.

4. **Core of a fuzzy set $A(x)$:** It is nothing but its 1-cut.
5. **Height of a fuzzy set $A(x)$:** It is denoted by $h(A)$ and defined as the largest of membership values of the elements contained in that set.
6. **Normal fuzzy set:** A fuzzy set $A(x)$ is called **normal**, if its height $h(A) = 1.0$.
7. **Sub-normal fuzzy set:** A fuzzy set $A(x)$ is known as a **sub-normal** fuzzy set, if

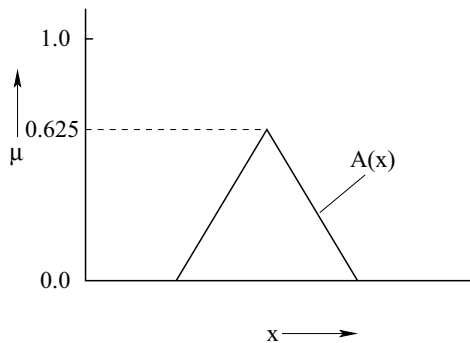


Figure 7.16: Sub-normal fuzzy set.

its height $h(A) < 1.0$ (refer to Fig. 7.16).

7.2.4 Some Standard Operations in Fuzzy Sets and Relations

As a fuzzy set is expressed in terms of membership values of its elements, operations of the fuzzy sets are represented by their membership values. Some standard operations used in the fuzzy sets are explained below.

1. **Proper subset of a fuzzy set:** Let us consider two fuzzy sets: $A(x)$ and $B(x)$, such that all $x \in X$. The fuzzy set $A(x)$ is called the proper subset of $B(x)$, if $\mu_A(x) < \mu_B(x)$. It can be represented as follows:

$$A(x) \subset B(x), \text{ if } \mu_A(x) < \mu_B(x). \quad (7.21)$$

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

As $\mu_A(x) < \mu_B(x)$ for all $x \in X$, $A(x) \subset B(x)$.

2. **Equal fuzzy sets:** Two fuzzy sets: $A(x)$ and $B(x)$ are said to be equal, if $\mu_A(x) = \mu_B(x)$ for all $x \in X$. It is expressed as follows:

$$A(x) = B(x), \text{ if } \mu_A(x) = \mu_B(x). \quad (7.22)$$

Note: Two fuzzy sets: $A(x)$ and $B(x)$ are said to be un-equal, if $\mu_A(x) \neq \mu_B(x)$ for at least one $x \in X$.

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

As $\mu_A(x) \neq \mu_B(x)$ for different $x \in X$, $A(x) \neq B(x)$.

3. **Complement of a fuzzy set $A(x)$:** It is denoted by $\bar{A}(x)$ and defined with respect to the Universal set X as follows:

$$\bar{A}(x) = 1 - A(x), \text{ for all } x \in X. \quad (7.23)$$

Fig. 7.17 shows the complement of a fuzzy set.

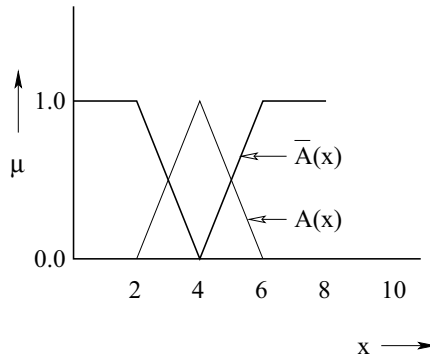


Figure 7.17: Complement of a fuzzy set.

Example:

Let us consider a fuzzy set $A(x)$:

$$A(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\}$$

Its complement $\bar{A}(x) = \{(x_1, 0.9), (x_2, 0.8), (x_3, 0.7), (x_4, 0.6)\}$.

4. **Intersection of fuzzy sets:** Intersection of two fuzzy sets: $A(x)$ and $B(x)$ for all $x \in X$, is denoted by $(A \cap B)(x)$ and its membership function value is determined as follows (refer to the shaded area of Fig. 7.18):

$$\mu_{(A \cap B)}(x) = \min\{\mu_A(x), \mu_B(x)\}. \quad (7.24)$$

It is to be noted that intersection is analogous to logical **AND** operation.

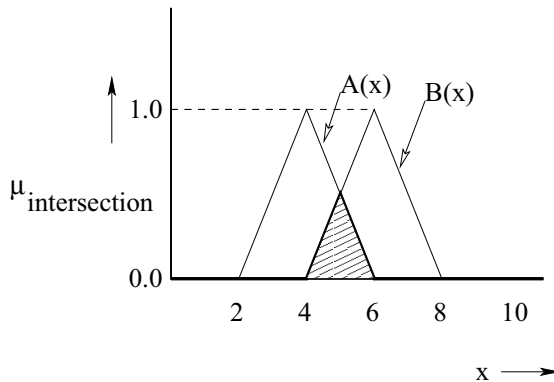


Figure 7.18: Intersection of two fuzzy sets.

Example:

Let us consider the following two fuzzy sets:

$$A(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\}$$

$$B(x) = \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\}$$

Now, $\mu_{(A \cap B)}(x_1) = \min\{\mu_A(x_1), \mu_B(x_1)\} = \min\{0.1, 0.5\} = 0.1$.

Similarly, $\mu_{(A \cap B)}(x_2)$, $\mu_{(A \cap B)}(x_3)$, $\mu_{(A \cap B)}(x_4)$ are found to be equal to 0.2, 0.3 and 0.4, respectively.

5. **Union of fuzzy sets:** Union of two fuzzy sets: $A(x)$ and $B(x)$ for all $x \in X$, is represented by $(A \cup B)(x)$ and its membership function value is determined as follows (refer to the shaded area of Fig. 7.19):

$$\mu_{(A \cup B)}(x) = \max\{\mu_A(x), \mu_B(x)\}. \quad (7.25)$$

It is important to note that union is analogous to logical **OR** operation.

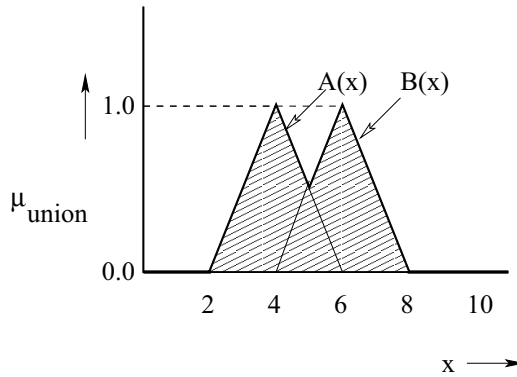


Figure 7.19: Union of two fuzzy sets.

Example:

Let us consider the following two fuzzy sets:

$$A(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\}$$

$$B(x) = \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\}$$

Now, $\mu_{(A \cup B)}(x_1) = \max\{\mu_A(x_1), \mu_B(x_1)\} = \max\{0.1, 0.5\} = 0.5$.

Similarly, $\mu_{(A \cup B)}(x_2)$, $\mu_{(A \cup B)}(x_3)$, $\mu_{(A \cup B)}(x_4)$ are found to be equal to 0.7, 0.8 and 0.9, respectively.

6. **Algebraic product of fuzzy sets:** An algebraic product of two fuzzy sets: $A(x)$ and $B(x)$ is denoted by $A(x).B(x)$ and defined like the following:

$$A(x).B(x) = \{(x, \mu_A(x).\mu_B(x)), x \in X\}. \quad (7.26)$$

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

Now, $A(x).B(x) = \{(x_1, 0.05), (x_2, 0.14), (x_3, 0.24), (x_4, 0.36)\}$.

7. **Multiplication of a fuzzy set by a crisp number:** The product of a fuzzy set $A(x)$ and a crisp number d is expressed as follows:

$$d.A(x) = \{(x, d \times \mu_A(x)), x \in X\}. \quad (7.27)$$

Example:

Let us consider a fuzzy set $A(x)$:

$$A(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\}$$

If $d = 0.2$, then $d.A(x) = \{(x_1, 0.02), (x_2, 0.04), (x_3, 0.06), (x_4, 0.08)\}$.

8. **Power of a fuzzy set:** The p -th power of a fuzzy set $A(x)$ yields another fuzzy set $A^p(x)$, whose membership value can be determined as follows:

$$\mu_{A^p}(x) = \{\mu_A(x)\}^p, x \in X. \quad (7.28)$$

It is important to mention that if p is set equal to 2 and $\frac{1}{2}$, then $A^p(x)$ is called **concentration** and **dilation**, respectively.

Example:

Let us consider a fuzzy set $A(x)$:

$$A(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\}$$

Now, for $p = 2$,

$$A^2(x) = \{(x_1, 0.01), (x_2, 0.04), (x_3, 0.09), (x_4, 0.16)\}.$$

9. **Algebraic sum of two fuzzy sets $A(x)$ and $B(x)$:** It is determined as follows:

$$A(x) + B(x) = \{(x, \mu_{A+B}(x)), x \in X\}, \quad (7.29)$$

where $\mu_{A+B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$.

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

Now, $A(x) + B(x) = \{(x_1, 0.55), (x_2, 0.76), (x_3, 0.86), (x_4, 0.94)\}$.

10. **Bounded sum of two fuzzy sets $A(x)$ and $B(x)$:** It is defined like the following:

$$A(x) \oplus B(x) = \{(x, \mu_{A \oplus B}(x)), x \in X\}, \quad (7.30)$$

where $\mu_{A \oplus B}(x) = \min\{1, \mu_A(x) + \mu_B(x)\}$.

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

Now, $A(x) \oplus B(x) = \{(x_1, 0.6), (x_2, 0.9), (x_3, 1.0), (x_4, 1.0)\}$.

11. **Algebraic difference of two fuzzy sets $A(x)$ and $B(x)$:** It will give rise to a new fuzzy set as given below.

$$A(x) - B(x) = \{(x, \mu_{A-B}(x)), x \in X\}, \quad (7.31)$$

where $\mu_{A-B}(x) = \mu_{A \cap \bar{B}}(x)$.

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

Now, $\bar{B} = \{(x_1, 0.5), (x_2, 0.3), (x_3, 0.2), (x_4, 0.1)\}$.

Therefore, $A(x) - B(x) = \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.2), (x_4, 0.1)\}$.

12. **Bounded difference of two fuzzy sets $A(x)$ and $B(x)$:** It is calculated as follows:

$$A(x) \ominus B(x) = \{(x, \mu_{A \ominus B}(x)), x \in X\}, \quad (7.32)$$

where $\mu_{A \ominus B}(x) = \max\{0, \mu_A(x) + \mu_B(x) - 1\}$.

Example:

Let us consider the following two fuzzy sets:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

Now, $A(x) \ominus B(x) = \{(x_1, 0), (x_2, 0), (x_3, 0.1), (x_4, 0.3)\}$.

13. **Cartesian product of two fuzzy sets:** Let us consider two fuzzy sets: $A(x)$ and $B(y)$ defined on the Universal sets X and Y , respectively. The Cartesian product of fuzzy sets: $A(x)$ and $B(y)$ is denoted by $A(x) \times B(y)$, such that $x \in X$ and $y \in Y$. It is determined, so that the following condition is satisfied.

$$\mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y)). \quad (7.33)$$

Example:

Let us consider the following two fuzzy sets:

$$A(x) = \{(x_1, 0.2), (x_2, 0.3), (x_3, 0.5), (x_4, 0.6)\}$$

$$B(y) = \{(y_1, 0.8), (y_2, 0.6), (y_3, 0.3)\}$$

Now, $\min(\mu_A(x_1), \mu_B(y_1)) = \min(0.2, 0.8) = 0.2$.

Similarly, the values of $\min(\mu_A(x_1), \mu_B(y_2))$, $\min(\mu_A(x_1), \mu_B(y_3))$, $\min(\mu_A(x_2), \mu_B(y_1))$, $\min(\mu_A(x_2), \mu_B(y_2))$, $\min(\mu_A(x_2), \mu_B(y_3))$, $\min(\mu_A(x_3), \mu_B(y_1))$, $\min(\mu_A(x_3), \mu_B(y_2))$, $\min(\mu_A(x_3), \mu_B(y_3))$, $\min(\mu_A(x_4), \mu_B(y_1))$, $\min(\mu_A(x_4), \mu_B(y_2))$ and $\min(\mu_A(x_4), \mu_B(y_3))$ are found to be equal to 0.2, 0.2, 0.3, 0.3, 0.3, 0.5, 0.5, 0.3, 0.6, 0.6 and 0.3, respectively. Thus, $A \times B$ will come out to be as follows:

$$A \times B = \begin{bmatrix} 0.2 & 0.2 & 0.2 \\ 0.3 & 0.3 & 0.3 \\ 0.5 & 0.5 & 0.3 \\ 0.6 & 0.6 & 0.3 \end{bmatrix}$$

14. **Composition of fuzzy relations:** Let $A = [a_{ij}]$ and $B = [b_{jk}]$ be two fuzzy relations expressed in the matrix form. Composition of these two fuzzy relations, that is, C is represented by

$$C = A \circ B, \quad (7.34)$$

such that it can be expressed in the matrix form:

$$[c_{ik}] = [a_{ij}] \circ [b_{jk}],$$

where

$$c_{ik} = \max[\min(a_{ij}, b_{jk})].$$

Example:

Let us consider the following two fuzzy relations:

$$A = [a_{ij}] = \begin{bmatrix} 0.2 & 0.3 \\ 0.5 & 0.7 \end{bmatrix}$$

$$B = [b_{jk}] = \begin{bmatrix} 0.3 & 0.6 & 0.7 \\ 0.1 & 0.8 & 0.6 \end{bmatrix}$$

Now, the elements of $[c_{ik}]$ matrix can be determined as follows:

$$\begin{aligned} c_{11} &= \max[\min(a_{11}, b_{11}), \min(a_{12}, b_{21})] \\ &= \max[\min(0.2, 0.3), \min(0.3, 0.1)] \\ &= 0.2 \end{aligned}$$

$$\begin{aligned}
c_{12} &= \max[\min(a_{11}, b_{12}), \min(a_{12}, b_{22})] \\
&= \max[\min(0.2, 0.6), \min(0.3, 0.8)] \\
&= 0.3
\end{aligned}$$

$$\begin{aligned}
c_{13} &= \max[\min(a_{11}, b_{13}), \min(a_{12}, b_{23})] \\
&= \max[\min(0.2, 0.7), \min(0.3, 0.6)] \\
&= 0.3
\end{aligned}$$

$$\begin{aligned}
c_{21} &= \max[\min(a_{21}, b_{11}), \min(a_{22}, b_{21})] \\
&= \max[\min(0.5, 0.3), \min(0.7, 0.1)] \\
&= 0.3
\end{aligned}$$

$$\begin{aligned}
c_{22} &= \max[\min(a_{21}, b_{12}), \min(a_{22}, b_{22})] \\
&= \max[\min(0.5, 0.6), \min(0.7, 0.8)] \\
&= 0.7
\end{aligned}$$

$$\begin{aligned}
c_{23} &= \max[\min(a_{21}, b_{13}), \min(a_{22}, b_{23})] \\
&= \max[\min(0.5, 0.7), \min(0.7, 0.6)] \\
&= 0.6
\end{aligned}$$

Therefore, C is found to be like the following:

$$C = \begin{bmatrix} 0.2 & 0.3 & 0.3 \\ 0.3 & 0.7 & 0.6 \end{bmatrix}$$

7.2.5 Properties of Fuzzy Sets

Fuzzy sets follow the properties of crisp sets stated in Section 7.1.3, except the following two, as explained below.

1. Law of excluded middle:

In crisp set,

$$A \cup \overline{A} = X.$$

However, according to the fuzzy set,

$$A \cup \overline{A} \neq X.$$

2. Law of contradiction:

As per crisp set,

$$A \cap \bar{A} = \emptyset.$$

However, in the fuzzy set,

$$A \cap \bar{A} \neq \emptyset.$$

It happens due to the fact that an element belongs to both a fuzzy set as well as its complement. For example, if the apple is **red** with membership value 0.8 (that is, $\mu_{red}(apple) = 0.8$), it is also **not red** (\bar{red}) with membership value of 0.2 (that is, $\mu_{\bar{red}}(apple) = 0.2$) (refer to Fig. 7.20). Fig. 7.20(a) shows the violation of the law of excluded middle, whereas the

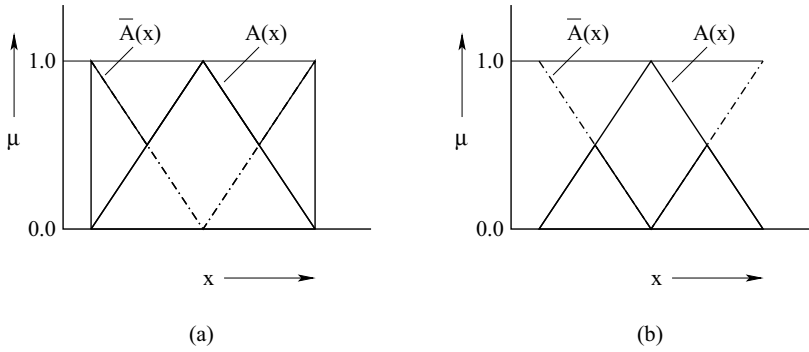


Figure 7.20: Two properties of crisp sets violated by the fuzzy sets, such as (a) Law of excluded middle, (b) Law of contradiction.

reason behind violation of the law of contradiction can be understood from Fig. 7.20(b).

7.3 Measures of Fuzziness and Inaccuracy of Fuzzy Sets

The fuzziness of a fuzzy set was measured using the concept of entropy by DeLuca and Termini [84] as given below.

Let us consider $X = \{x_1, x_2, \dots, x_n\}$ be a discrete universe of discourse. The entropy of a fuzzy set $A(x)$ could be determined as follows:

$$H(A) = -\frac{1}{n} \sum_{i=1}^n [\mu_A(x_i) \log\{\mu_A(x_i)\} + \{1 - \mu_A(x_i)\} \log\{1 - \mu_A(x_i)\}] \quad (7.35)$$

Example:

Let $A(x)$ be a fuzzy set in a discrete universe of discourse as given below.

$A(x) = \{(x_1, 0.1), (x_2, 0.3), (x_3, 0.4), (x_4, 0.5)\}$. Calculate its entropy value.

The value of entropy, $H(A) = -\frac{1}{4}[\{0.1 \times \log(0.1) + 0.9 \times \log(0.9)\} + \{0.3 \times \log(0.3) + 0.7 \times \log(0.7)\} + \{0.4 \times \log(0.4) + 0.6 \times \log(0.6)\} + \{0.5 \times \log(0.5) + 0.5 \times \log(0.5)\}] = 0.2499$

Now, let us consider two fuzzy sets: $A(x)$ and $B(x)$ defined in the same discrete universe of discourse $X = \{x_1, x_2, \dots, x_n\}$, whose membership values are denoted by $\mu_A(x_i)$ and $\mu_B(x_i)$, respectively, where $i = 1, 2, \dots, n$.

The inaccuracy of the fuzzy set $B(x)$ can be measured with respect to the fuzzy set $A(x)$ as follows [85]:

$$I(A; B) = -\frac{1}{n} \sum_{i=1}^n [\mu_A(x_i) \log(\mu_B(x_i)) + (1 - \mu_A(x_i)) \log(1 - \mu_B(x_i))] \quad (7.36)$$

Example:

Let us consider the following two fuzzy sets in the same discrete universe of discourse:

$$\begin{aligned} A(x) &= \{(x_1, 0.1), (x_2, 0.2), (x_3, 0.3), (x_4, 0.4)\} \\ B(x) &= \{(x_1, 0.5), (x_2, 0.7), (x_3, 0.8), (x_4, 0.9)\} \end{aligned}$$

Determine inaccuracy of the fuzzy set $B(x)$ with respect to the fuzzy set $A(x)$.

The inaccuracy of $B(x)$ with respect to $A(x)$, $I(A; B) = -\frac{1}{4}[\{0.1 \times \log(0.5) + 0.9 \times \log(0.5)\} + \{0.2 \times \log(0.7) + 0.8 \times \log(0.3)\} + \{0.3 \times \log(0.8) + 0.7 \times \log(0.2)\} + \{0.4 \times \log(0.9) + 0.6 \times \log(0.1)\}] = 0.4717$

7.4 Summary

This chapter has been summarized as follows:

1. An introduction is given to crisp sets, which are the sets having fixed boundaries. The notations used in set theory have been listed. After defining a few crisp set operations (such as difference, intersection, union), different properties of crisp sets have been stated.
2. The concept of fuzzy sets has been introduced. Fuzzy sets are the sets with vague boundaries. It is important to mention that a crisp set is a special case of the more general concept of a fuzzy set. A fuzzy set may be either discrete or continuous in nature. Some of the frequently-used membership function distributions for representing the fuzzy sets are discussed, in detail. After defining a few terms related to fuzzy sets (namely α -cut, support, scalar cardinality, core, height, and others), some of their standard operations are explained in detail.
3. It deals with the measures of fuzziness and inaccuracy of fuzzy sets.

7.5 Exercise

1. Justify the following statements:
 - (i) Every set is the subset of both itself as well as the Universal set.

- (ii) An empty set is a subset of any other set.
- (iii) A fuzzy set is said to be convex, if all α -cut sets are convex.
2. Let us consider the following two fuzzy sets:
 $A(x) = \{(x_1, 0.15), (x_2, 0.27), (x_3, 0.47), (x_4, 0.57)\}$
 $B(x) = \{(x_1, 0.35), (x_2, 0.48), (x_3, 0.58), (x_4, 0.69)\}$
 Determine
- Scalar cardinality of above two fuzzy sets,
 - $\overline{A}(x)$ and $\overline{B}(x)$,
 - Intersection of $A(x)$ and $B(x)$,
 - Union of $A(x)$ and $B(x)$,
 - Algebraic product of $A(x)$ and $B(x)$,
 - $d.A(x)$ and $d.B(x)$, where $d = 0.3$ is a crisp number,
 - Concentration and dilation of $A(x)$ and $B(x)$,
 - Algebraic sum of $A(x)$ and $B(x)$,
 - Bounded sum of $A(x)$ and $B(x)$,
 - Algebraic difference of $A(x)$ and $B(x)$,
 - Bounded difference of $A(x)$ and $B(x)$.
3. Determine Cartesian product of the following two fuzzy sets:
 $A(x) = \{(x_1, 0.1), (x_2, 0.25), (x_3, 0.34)\}$
 $B(y) = \{(y_1, 0.8), (y_2, 0.7)\}$.
4. Two fuzzy relations A and B are given below (expressed in the matrix form).

$$A = [a_{ij}] = \begin{bmatrix} 0.3 & 0.4 \\ 0.1 & 0.5 \end{bmatrix}$$

$$B = [b_{jk}] = \begin{bmatrix} 0.2 & 0.6 & 0.7 \\ 0.9 & 0.8 & 0.1 \end{bmatrix}$$

Determine the composition of these two fuzzy relations.

5. Let us consider a fuzzy set $A(x)$ in a discrete universe of discourse as follows:
 $A(x) = \{(x_1, 0.2), (x_2, 0.25), (x_3, 0.3), (x_4, 0.4), (x_5, 0.6)\}$. Calculate its entropy value.
6. Let us consider the following two fuzzy sets in the same discrete universe of discourse:

$$A(x) = \{(x_1, 0.2), (x_2, 0.25), (x_3, 0.3), (x_4, 0.35), (x_5, 0.4)\}$$

$$B(x) = \{(x_1, 0.3), (x_2, 0.32), (x_3, 0.4), (x_4, 0.45), (x_5, 0.5)\}$$

Determine inaccuracy of the fuzzy set $B(x)$ with respect to the fuzzy set $A(x)$.

Chapter 8

Fuzzy Reasoning and Clustering

This chapter deals with two of the most important applications of fuzzy set theory, which are developed in the forms of reasoning tool and clustering technique.

8.1 Introduction

The concept of fuzzy set theory has been used in a number of ways, such as Fuzzy Logic Controller (FLC), fuzzy clustering algorithms, fuzzy mathematical programming, fuzzy graph theory, and others. Out of all such applications, the FLC is the most popular one, due to the following reasons: (i) ease of understanding and implementations, (ii) ability to handle uncertainty and imprecision. Moreover, an exact mathematical formulation of the problem is not essential for development of an FLC. This feature makes it a natural choice for solving complex real-world problems, which are either difficult to model mathematically or the mathematical model becomes highly non-linear. Two important tools developed based on the fuzzy set theory, namely fuzzy logic controller and fuzzy clustering algorithm are explained below in detail.

8.2 Fuzzy Logic Controller

Fuzzy Logic Controller (FLC) was first developed by Mamdani and Assilian around 1975 [86], although the concept of fuzzy set was published in 1965. Human beings have the natural quest to know input-output relationships of a process. The behavior of a human being is modeled artificially for designing a suitable FLC. The performance of an FLC depends on its Knowledge Base (KB), which consists of both Data Base (DB) (that is, data related to membership function distributions of the variables of the process to be controlled) as well as Rule Base (RB). However, designing a proper KB of an FLC is a difficult task, which can be implemented in one of the following ways:

- Optimization of the DB only,
- Optimization of the RB only,

- Optimization of the DB and RB in stages,
- Optimization of the DB and RB simultaneously.

The membership function distributions are assumed to be either linear (such as triangular, trapezoidal) or non-linear (namely Gaussian, bell-shaped, sigmoid). To design and develop a suitable FLC for controlling a process, its variables need to be expressed in the form of some linguistic terms (such as VN: Very Near, VFR: Very Far, A: Ahead etc.) and the relationships between input (antecedent) and output (consequent) variables are expressed in the form of some rules. For example, a rule can be expressed as follows (refer to Fig. 8.1):

IF I_1 is NR AND I_2 is A THEN O is ART .

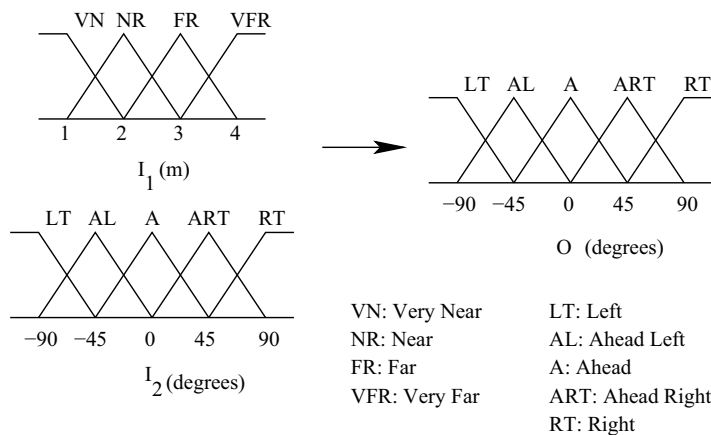


Figure 8.1: A diagram showing some membership function distributions of input and output variables of an FLC.

It is obvious that the number of rules to be present in the rule base will increase, as the number of linguistic terms used to represent the variables increases (to ensure a better accuracy in prediction). Moreover, computational complexity of the controller will increase with the number of rules. For an easy implementation in either the software or hardware, the number of rules present in the RB should be as small as possible. Realizing this fact, some investigators have also tried to design and develop a hierarchical FLC, in which the number of rules will be kept to the minimum [87, 88]. It has been observed that the performance of an FLC largely depends on the RB and optimizing the DB is a fine tuning process [89]. An FLC does not have an in-built optimization module. An external optimizer is to be used to develop its optimal KB through a proper tuning and this helps to improve the performance.

8.2.1 Two Major Forms of Fuzzy Logic Controller

System modeling done using the concept of fuzzy sets can be classified into two groups, namely **linguistic fuzzy modeling** and **precise fuzzy modeling**. Linguistic fuzzy mod-

eling, such as **Mamdani Approach** [86] is characterized by its high interpretability and low accuracy, whereas the aim of precise fuzzy modeling like **Takagi and Sugeno's Approach** [90], is to obtain high accuracy but at the cost of interpretability. Interpretability of a fuzzy modeling is defined as its capability to express the behavior of a system in an understandable form. It is generally expressed in terms of compactness, completeness, consistency and transparency. The accuracy of a fuzzy model indicates how closely it can represent the modeled system. The working principles of both these approaches are briefly explained below.

Mamdani Approach [86]:

An FLC consists of four modules, namely a rule base, an inference engine, fuzzification and de-fuzzification. Fig. 8.2 shows a schematic view explaining the working cycle of an FLC.

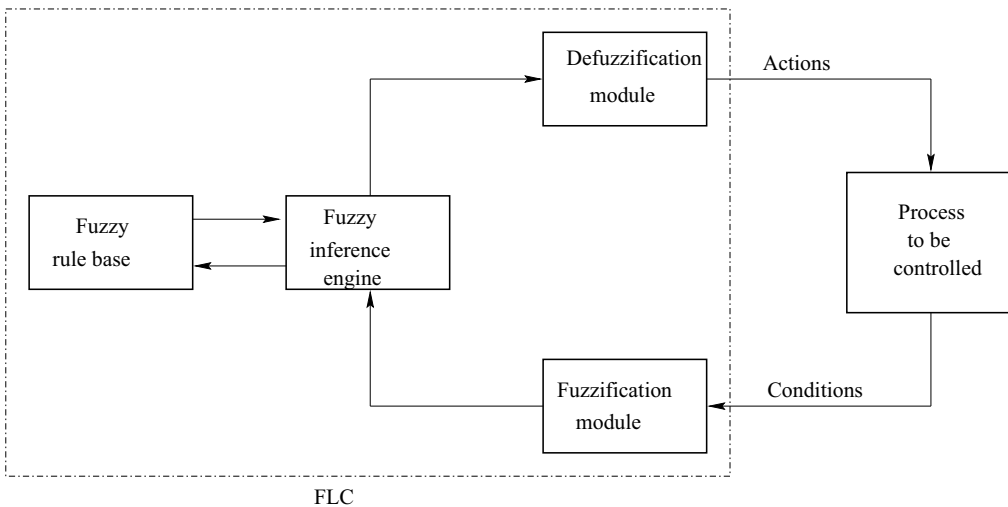


Figure 8.2: A schematic view showing the working cycle of an FLC.

The following steps are involved in the working cycle of an FLC:

- The **condition** (also known as **antecedent**) and **action** (also called **consequent**) variables needed to control a process are identified and measurements are taken of all the condition variables.
- The measurements taken in the previous step are converted into appropriate fuzzy sets to express measurement uncertainties. This process is known as **fuzzification**.
- The fuzzified measurements are then used by inference engine to evaluate the control rules stored in the fuzzy rule base and a fuzzified output is determined.

- The fuzzified output is then converted into a single crisp value. This conversion is called **de-fuzzification**. The de-fuzzified values represent the actions to be taken by the FLC in controlling the process.

The fuzzy reasoning process is illustrated in Figure 8.3. Let us assume for simplicity that

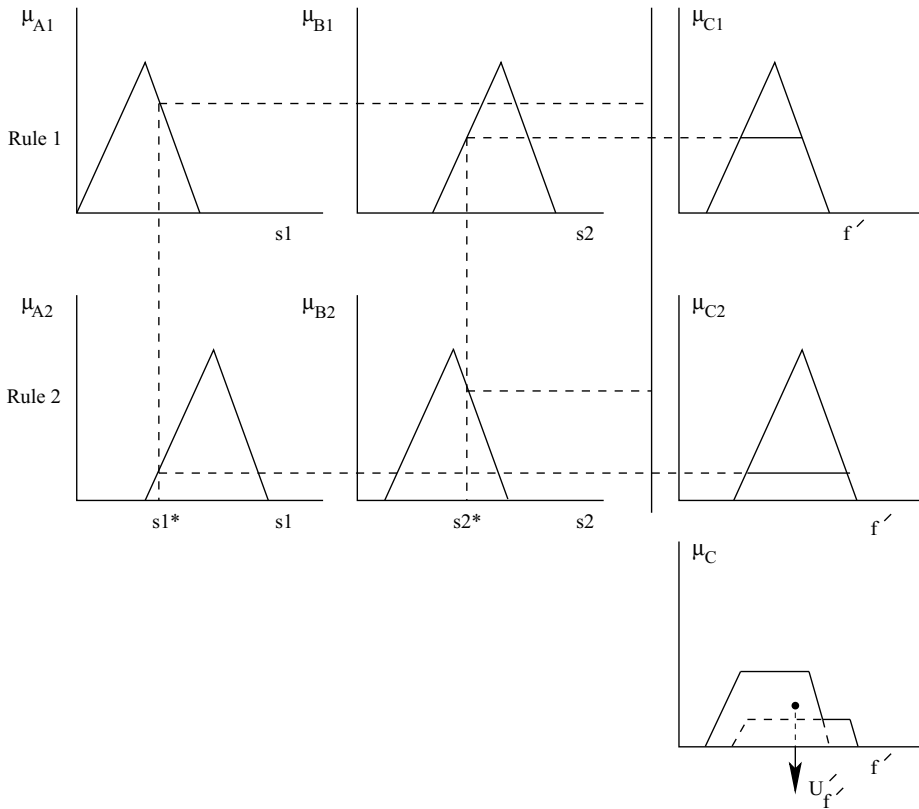


Figure 8.3: A schematic view showing the working principle of an FLC.

only two fuzzy control rules (out of many rules present in the rule base) are being fired as given below, for a set of inputs: $(s1^*, s2^*)$.

Rule 1: IF $s1$ is $A1$ AND $s2$ is $B1$ THEN f' is $C1$

Rule 2: IF $s1$ is $A2$ AND $s2$ is $B2$ THEN f' is $C2$

If $s1^*$ and $s2^*$ are the inputs for fuzzy variables: $s1$ and $s2$ and if μ_{A1} and μ_{B1} are the membership function values for A and B , respectively, then the grade of membership of $s1^*$ in $A1$ and that of $s2^*$ in $B1$ are represented by $\mu_{A1}(s1^*)$ and $\mu_{B1}(s2^*)$, respectively, for rule 1. Similarly, for rule 2, $\mu_{A2}(s1^*)$ and $\mu_{B2}(s2^*)$ are used to represent the membership

function values. The firing strengths of the first and second rules are calculated as follows:

$$\alpha_1 = \min(\mu_{A1}(s1^*), \mu_{B1}(s2^*)), \quad (8.1)$$

$$\alpha_2 = \min(\mu_{A2}(s1^*), \mu_{B2}(s2^*)). \quad (8.2)$$

The membership value of the combined control action C is given by

$$\mu_C(f') = \max(\mu_{C1}^*(f'), \mu_{C2}^*(f')). \quad (8.3)$$

There are several methods of defuzzification (refer to Fig. 8.4), which are explained below.

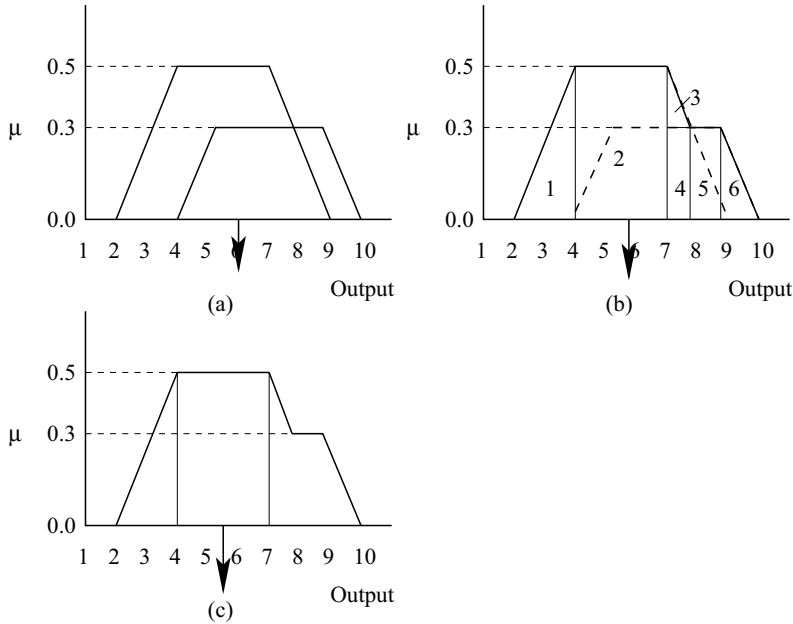


Figure 8.4: Different methods of defuzzification: (a) center of sums method, (b) centroid method, (c) mean of maxima method.

1. **Center of Sums Method:** In this method of defuzzification (refer to Fig. 8.4(a)), the crisp output can be determined as follows:

$$U'_{f'} = \frac{\sum_{j=1}^p A(\alpha_j) \times f_j}{\sum_{j=1}^p A(\alpha_j)}, \quad (8.4)$$

where $U'_{f'}$ is the output of the controller, $A(\alpha_j)$ represents the firing area of j -th rule, p is the total number of the fired rules and f_j represents the center of the area.

2. **Centroid Method:** The total area of the membership function distribution used to represent the combined control action is divided into a number of standard sub-areas,

whose area and the center of area can be determined easily (refer to Fig. 8.4(b)). The crisp output of the controller can be calculated using the expression given below.

$$U'_{f'} = \frac{\sum_{i=1}^N A_i f_i}{\sum_{i=1}^N A_i}, \quad (8.5)$$

where N indicates the number of small areas or regions, A_i and f_i represent the area and center of area, respectively, of i -th small region.

3. **Mean of Maxima Method:** From the membership function distribution of combined control action, the range of the output variable corresponding to the maximum value of membership is identified. The mid-value of this range is considered to be the crisp output of the controller (refer to Fig. 8.4(c)).

A Numerical Example:

Fig. 8.5 shows a typical problem scenario related to navigation of a mobile robot in the presence of four moving obstacles. The directions of movement of the obstacles are shown

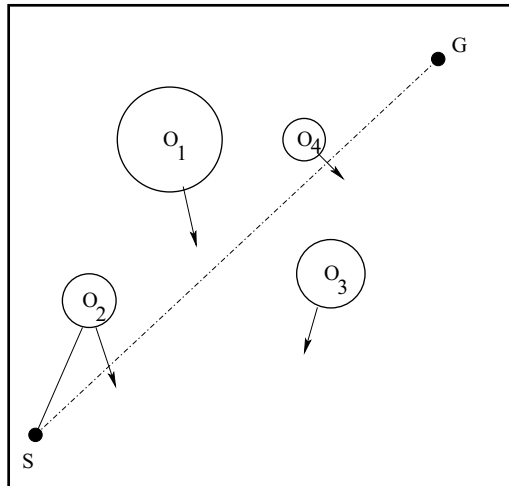


Figure 8.5: A typical robot motion planning problem.

in the figure. The obstacle O_2 is found to be the most critical one. Our aim is to develop a fuzzy logic-based motion planner that will be able to generate the collision-free path for the robot. There are two inputs, namely the **distance** between the robot and the obstacle (D) and **angle** ($\angle GSO_2$) for the motion planner and it will generate one output, that is, **deviation**. Distance is represented using four linguistic terms, namely Very Near (VN), Near (NR), Far (FR) and Very Far (VFR), whereas the input: angle and output: deviation are expressed with the help of five linguistic terms, such as Ahead (A), Ahead Left (AL),

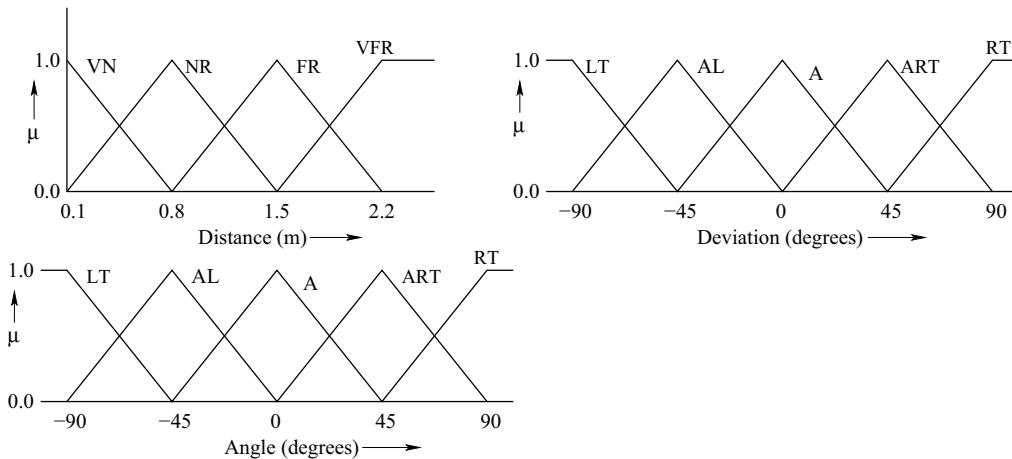


Figure 8.6: Data Base of the fuzzy logic controller.

Left (LT), Ahead Right (ART) and Right (RT). Fig. 8.6 shows the DB of the FLC. The rule base of the fuzzy logic-based motion planner is given in Table 8.1. Determine the output -

Table 8.1: Rule Base of the fuzzy logic controller

		Angle				
		LT	AL	A	ART	RT
Distance	VN	A	ART	AL	AL	A
	NR	A	A	RT	A	A
	FR	A	A	ART	A	A
	VFR	A	A	A	A	A

deviation for the set of inputs: distance $D = 1.04$ m and angle $\angle GSO_2 = 30$ degrees, using Mamdani Approach. Use different methods of defuzzification.

Solution:

The inputs are:

$$\text{Distance} = 1.04 \text{ m, Angle} = 30 \text{ degrees}$$

The distance of 1.04 m may be called either *NR* (Near) or *FR* (Far). Similarly, the input angle of 30 degrees can be declared either *A* (Ahead) or *ART* (Ahead Right). Fig. 8.7 shows a schematic view used to determine the membership value, corresponding to the distance of 1.04 m. Using the principle of similar triangle, we can write the following relationship:

$$\frac{x}{1.0} = \frac{1.5-1.04}{1.5-0.8}$$

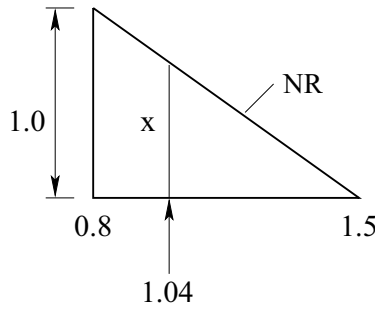


Figure 8.7: A schematic view used to determine the membership value corresponding to the distance of 1.04 m.

From the above expression, x is found to be equal to 0.6571. Thus, the distance of 1.04 m may be declared *NR* with a membership value of 0.6571, that is, $\mu_{NR} = 0.6571$. Similarly, the distance of 1.04 m can also be called *FR* with a membership value of 0.3429, that is, $\mu_{FR} = 0.3429$.

In the same way, an input angle of 30 degrees may be declared either *A* with a membership value of 0.3333 (that is, $\mu_A = 0.3333$) or *ART* with a membership value of 0.6667 (that is, $\mu_{ART} = 0.6667$).

For the above set of inputs, the following four rules are being fired from a total of 20:

If *Distance* is **NR** AND *Angle* is **A** Then *Deviation* is **RT**
 If *Distance* is **NR** AND *Angle* is **ART** Then *Deviation* is **A**
 If *Distance* is **FR** AND *Angle* is **A** Then *Deviation* is **ART**
 If *Distance* is **FR** AND *Angle* is **ART** Then *Deviation* is **A**

The strengths (α values) of the fired rules are calculated as follows:

$$\begin{aligned}\alpha_1 &= \min(\mu_{NR}, \mu_A) = \min(0.6571, 0.3333) = 0.3333 \\ \alpha_2 &= \min(\mu_{NR}, \mu_{ART}) = \min(0.6571, 0.6667) = 0.6571 \\ \alpha_3 &= \min(\mu_{FR}, \mu_A) = \min(0.3429, 0.3333) = 0.3333 \\ \alpha_4 &= \min(\mu_{FR}, \mu_{ART}) = \min(0.3429, 0.6667) = 0.3429\end{aligned}$$

The fuzzified outputs corresponding to above four fired rules are shown in Fig. 8.8. Fig. 8.9 shows the union of the fuzzified outputs, corresponding to the above four fired rules.

The above fuzzified output cannot be used as a control action and its crisp value has been determined using the following methods of defuzzification:

1. Center of Sums Method

Fig. 8.10 shows the Center of Sums Method of defuzzification. The shaded region corresponding to each fired rule is shown in this figure. The values of area and center of area of the above shaded regions are also given in this figure, in a tabular form. The crisp output U of above four fired rules can be calculated as follows (refer to equation (8.4)):

$$U = \frac{12.5 \times 71 + 39.7089 \times 0.0 + 25.0 \times 45.0 + 25.5699 \times 0.0}{12.5 + 39.7089 + 25.0 + 25.5699}$$

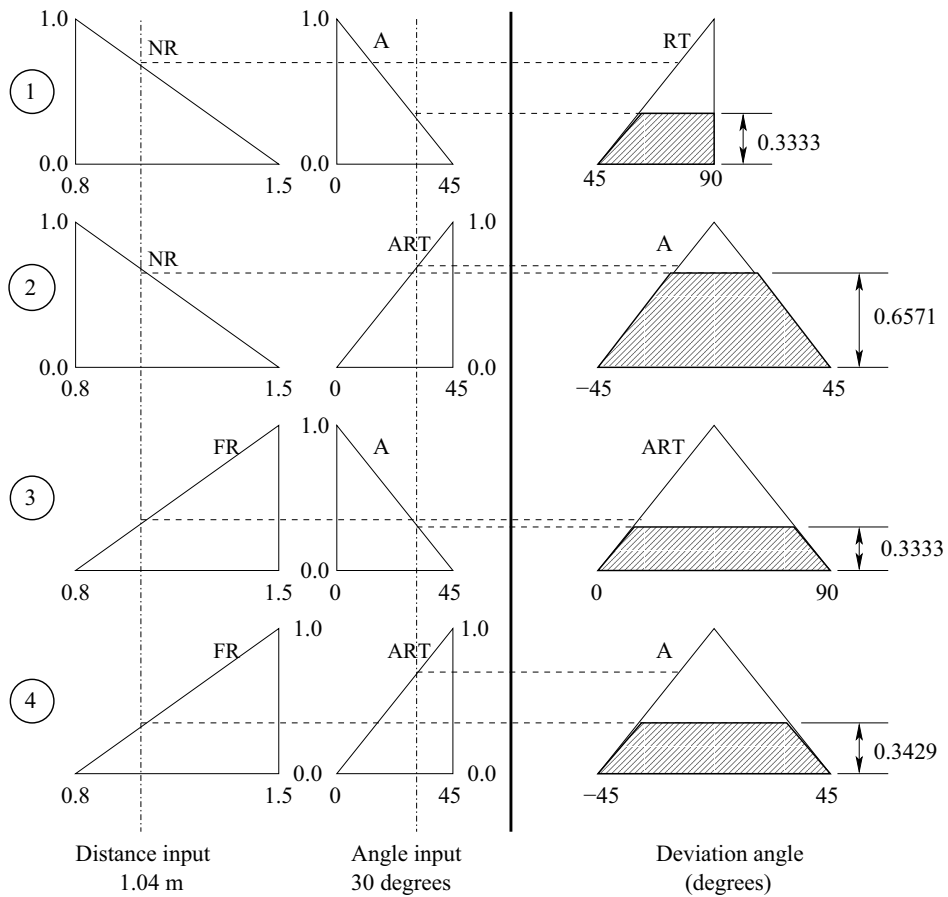


Figure 8.8: The fuzzified outputs corresponding to four different fired rules.

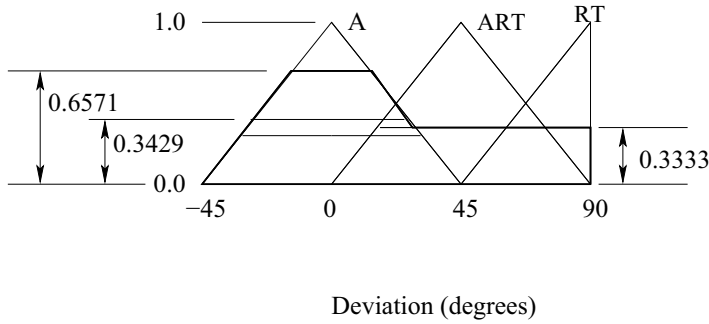


Figure 8.9: Combined fuzzified output considering all four fired rules.

$$= 19.5809$$

Therefore, the robot should deviate by 19.5809 degrees towards right with respect to the line joining the present position of the robot and the goal to avoid collision with the obstacle.

2. Centroid Method

The shaded region representing the combined output corresponding to above four fired rules (refer to Fig. 8.9) has been divided into a number of regular sub-regions, whose area and center of area can easily be determined. For example, the shaded region of Fig. 8.9 has been divided into four sub-regions (two triangles and two rectangles) as shown in Fig. 8.11. The values related to area and center of area of the above four sub-regions are shown in this figure, in a tabular form. In this method, the crisp output U can be obtained like the following (refer to equation (8.5)):

$$U = \frac{A}{B},$$

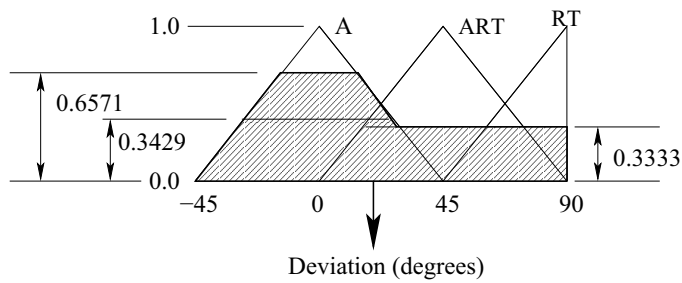
where $A = 9.7151 \times (-25.2860) + 20.2788 \times 0.0 + 2.3588 \times 20.2870 + 24.8540 \times 52.7153$,
 $B = 9.7151 + 20.2788 + 2.3588 + 24.8540$.

Therefore, U turns out to be equal to 19.4450.

Thus, the robot should deviate by 19.4450 degrees towards right with respect to the line joining the present position of the robot and the goal to avoid collision with the obstacle.

3. Mean of Maxima

Fig. 8.12 shows the fuzzified output of the controller for the above four fired rules. From this figure, it is observed that the maximum value of membership (that is, 0.6571) has occurred in a range of deviation angle starting from -15.4305 to 15.4305 degrees. Thus, its mean is coming out to be equal to 0.0. Therefore, the crisp output U of the controller becomes equal to 0.0, that is, $U = 0.0$. The robot will move along the line joining its present position and the goal.



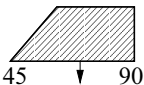
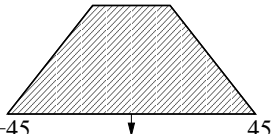
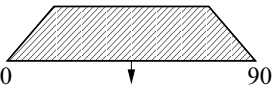
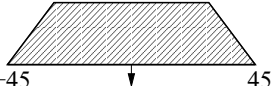
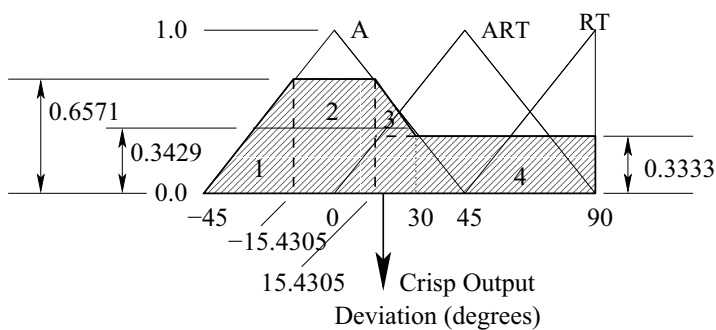
Fired rule	Shaded region	Area	Center of area
1		12.5	71
2		39.7089	0
3		25	45
4		25.5699	0

Figure 8.10: Defuzzification using the Center of Sums Method.



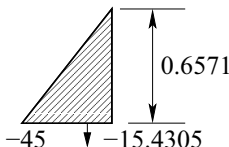
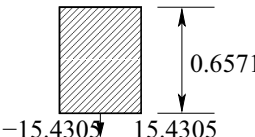
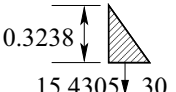
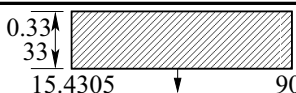
Marked regions	Shaded regions	Area	Center of area
1		9.7151	-25.2860
2		20.2788	0
3		2.3588	20.2870
4		24.8540	52.7153

Figure 8.11: Defuzzification using the Centroid Method.

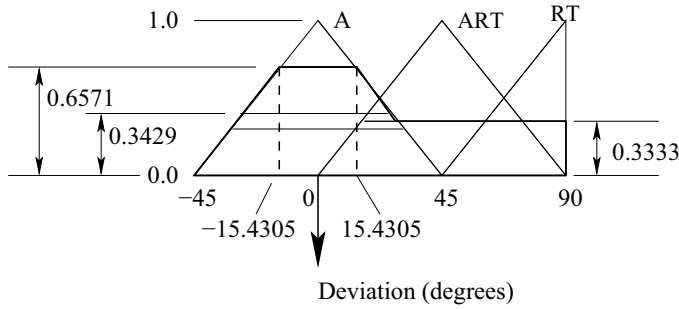


Figure 8.12: Defuzzification using the Mean of Maxima Method.

Takagi and Sugeno's Approach [90]:

Here, a rule is composed of fuzzy antecedent and functional consequent parts. Thus, a rule (say i -th) can be represented as follows:

$$\begin{aligned} &\text{If } x_1 \text{ is } A_1^i \text{ and } x_2 \text{ is } A_2^i \dots \text{ and } x_n \text{ is } A_n^i \\ &\text{then } y^i = a_0^i + a_1^i x_1 + \dots + a_n^i x_n, \end{aligned}$$

where a_0, a_1, \dots, a_n are the coefficients. It is to be noted that i does not represent the power but it is a superscript only. In this way, a nonlinear system is considered to be a combination of several linear systems. The weight of i -th rule can be determined for a set of inputs (x_1, x_2, \dots, x_n) like the following:

$$w^i = \mu_{A_1}^i(x_1) \mu_{A_2}^i(x_2) \dots \mu_{A_n}^i(x_n), \quad (8.6)$$

where A_1, A_2, \dots, A_n indicate membership function distributions of the linguistic terms used to represent the input variables and μ denotes the membership function value. Thus, the combined control action can be determined as follows:

$$y = \frac{\sum_{i=1}^k w^i y^i}{\sum_{i=1}^k w^i}, \quad (8.7)$$

where k indicates the total number of rules.

A Numerical Example:

A fuzzy logic-based expert system is to be developed that will work based on Takagi and Sugeno's approach to predict the output of a process. The DB of the FLC is shown in Fig. 8.13. As there are two inputs: I_1 and I_2 , and each input is represented using three linguistic terms (for example, LW , M , H for I_1 and NR , FR , VFR for I_2), there is a maximum of $3 \times 3 = 9$ feasible rules. The output of i -th rule, that is, y^i ($i = 1, 2, \dots, 9$) is expressed as follows:

$$y^i = f(I_1, I_2) = a_j^i I_1 + b_k^i I_2,$$

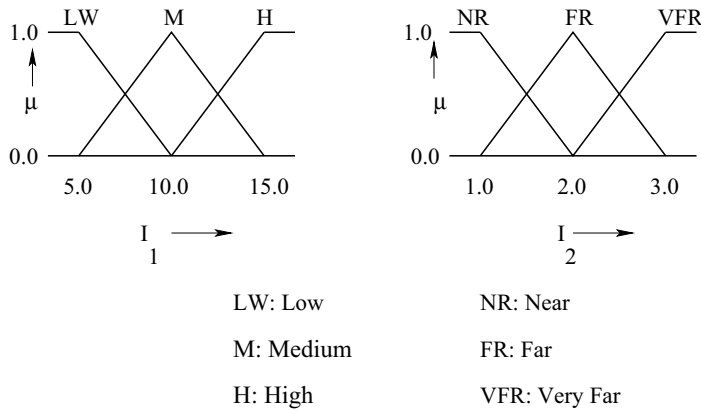


Figure 8.13: Data Base of the FLC.

where $j, k = 1, 2, 3$; $a_1^i = 1$, $a_2^i = 2$ and $a_3^i = 3$, if I_1 is found to be LW , M and H , respectively; $b_1^i = 1$, $b_2^i = 2$ and $b_3^i = 3$, if I_2 is seen to be NR , FR and VFR , respectively. Calculate the output of the FLC for the inputs: $I_1 = 6.0$, $I_2 = 2.2$.

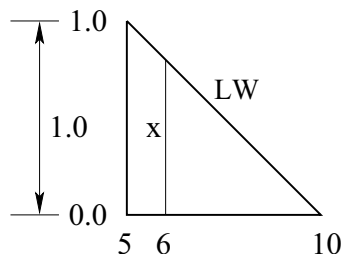
Solution:

The inputs are:

$$I_1 = 6.0$$

$$I_2 = 2.2.$$

The input I_1 of 6.0 unit can be called either LW (Low) or M (Medium). Similarly, the second input I_2 of 2.2 unit may be declared either FR (Far) or VFR (Very Far). Fig. 8.14 shows a schematic view used to determine the membership value corresponding to the first input $I_1 = 6.0$.

Figure 8.14: A schematic view used to determine the membership value corresponding to $I_1 = 6.0$.

Using the principle of similar triangle, we can write the following relationship:

$$\frac{x}{1.0} = \frac{10-6}{10-5}$$

From the above expression, x is coming out to be equal to 0.8. Thus, the input $I_1 = 6.0$ may be called LW with a membership value of 0.8, that is, $\mu_{LW} = 0.8$.

Similarly, the same input $I_1 = 6.0$ may also be called M with a membership value of 0.2, that is, $\mu_M = 0.2$.

In the same way, the input $I_2 = 2.2$ may be declared either FR with a membership value of 0.8 (that is, $\mu_{FR} = 0.8$) or VFR with a membership value of 0.2 (that is, $\mu_{VFR} = 0.2$).

For the above set of inputs, the following four combinations of input variables are being fired from a total of nine.

$$\begin{aligned} I_1 \text{ is } \mathbf{LW} \text{ and } I_2 \text{ is } \mathbf{FR}, \\ I_1 \text{ is } \mathbf{LW} \text{ and } I_2 \text{ is } \mathbf{VFR}, \\ I_1 \text{ is } \mathbf{M} \text{ and } I_2 \text{ is } \mathbf{FR}, \\ I_1 \text{ is } \mathbf{M} \text{ and } I_2 \text{ is } \mathbf{VFR}. \end{aligned}$$

Now, the weights: w^1 , w^2 , w^3 and w^4 of the first, second, third and fourth combination of fired input variables, respectively, have been calculated as follows:

$$\begin{aligned} w^1 &= \mu_{LW} \times \mu_{FR} = 0.8 \times 0.8 = 0.64 \\ w^2 &= \mu_{LW} \times \mu_{VFR} = 0.8 \times 0.2 = 0.16 \\ w^3 &= \mu_M \times \mu_{FR} = 0.2 \times 0.8 = 0.16 \\ w^4 &= \mu_M \times \mu_{VFR} = 0.2 \times 0.2 = 0.04 \end{aligned}$$

The functional consequent values: y^1 , y^2 , y^3 and y^4 of the first, second, third and fourth combination of fired input variables can be determined like the following:

$$\begin{aligned} y^1 &= I_1 + 2I_2 = 6.0 + 2 \times 2.2 = 10.4 \\ y^2 &= I_1 + 3I_2 = 6.0 + 3 \times 2.2 = 12.6 \\ y^3 &= 2I_1 + 2I_2 = 2 \times 6.0 + 2 \times 2.2 = 16.4 \\ y^4 &= 2I_1 + 3I_2 = 2 \times 6.0 + 3 \times 2.2 = 18.6 \end{aligned}$$

Therefore, the output y of the controller can be determined as follows (refer to equation (8.7)):

$$\begin{aligned} y &= \frac{w^1 y^1 + w^2 y^2 + w^3 y^3 + w^4 y^4}{w^1 + w^2 + w^3 + w^4} \\ &= \frac{0.64 \times 10.4 + 0.16 \times 12.6 + 0.16 \times 16.4 + 0.04 \times 18.6}{0.64 + 0.16 + 0.16 + 0.04} \\ &= 12.04 \end{aligned}$$

8.2.2 Hierarchical Fuzzy Logic Controller

Let us suppose that an FLC is to be designed and developed for controlling a process having n input variables, say I_1, I_2, \dots, I_n , and its input-output relationship can be expressed as $O = f(I_1, I_2, \dots, I_n)$. Fig. 8.15 shows a schematic view of the above controller.

Let us also assume that m fuzzy sets are used to represent each variable. Thus, a set of m^n rules is to be designed to develop the FLC completely. It is important to mention that the number of rules increases exponentially with the number of variables, which increases computational complexity of the algorithm and difficulties associated with controlling the

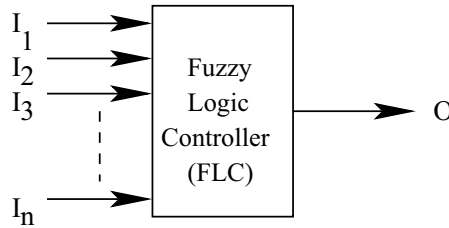


Figure 8.15: A schematic view showing the inputs and output of an FLC.

process. The above problem of rule explosion is known as the **curse of dimensionality**. Due to this reason, it becomes difficult to develop a suitable FLC for controlling a complex real-world problem involving many variables, such as weather forecasting.

To overcome this difficulty, hierarchical FLC was developed by Raju et al. [91], in which a higher dimensional FLC is considered to be a combination of some lower dimensional fuzzy logic systems connected in a hierarchical fashion. Fig. 8.16 shows the schematic view of a hierarchical FLC [87].

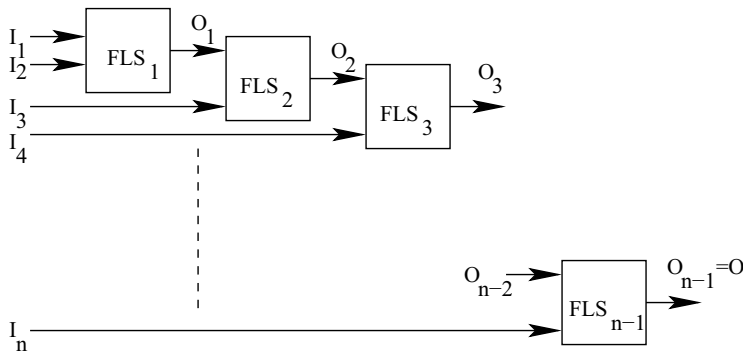


Figure 8.16: A schematic view of a hierarchical fuzzy logic controller.

As there are n input variables, let us consider $(n - 1)$ fuzzy logic systems as shown in Fig. 8.16. Two input variables (say I_1 and I_2) are fed to first Fuzzy Logic System (FLS) and it produces an output O_1 . This output O_1 along with another input variable (say I_3) are then fed to the second FLS. This process will continue until all the input variables are utilized.

As we consider only two inputs for each of the FLSs and each variable is expressed using m fuzzy sets, m^2 rules are to be designed for each FLS. Moreover, there are $(n - 1)$ such FLSs. Thus, a total of $(n - 1)m^2$ (in place of m^n) rules are necessary to develop the FLC completely, which clearly indicates that the number of rules of the hierarchical FLC increases linearly with the number of input variables n .

It is important to mention that the input variables pass through different number of

FLSs. For example, I_1 and I_2 pass through all $(n - 1)$ FLSs, whereas I_n passes through only $(n - 1)$ -th FLS. It clearly indicates that different weights are put on different input variables depending on their contributions towards the output of the controller. The most important input is passed through all $(n - 1)$ FLSs and the least important input may be passed through $(n - 1)$ -th FLS only. It is to be mentioned that the importance of an input is generally determined through a sensitivity analysis, the principle of which is discussed below.

8.2.3 Sensitivity Analysis

Sensitivity (s) of a controller is defined as the ratio of change in output to that in input, that is,

$$s = \frac{\delta O}{\delta I}. \quad (8.8)$$

A controller is said to be linear, if it has a constant sensitivity over the entire range of the variables.

The sensitivity of an FLC can be checked in the following ways:

1. Once the controller has been developed, we carry out an experiment, in which we vary the input variables by different amounts, say 0.1%, 1.0%, 10.0% etc. and note down the outputs of the controller. Using the above information, we can determine the sensitivity of the controller using equation (8.8) [92].
2. Let us consider a simple FLC having two inputs: I_1 and I_2 and one output, say O . The input-output relationship can be expressed like the following.

$$O = f(I_1, I_2). \quad (8.9)$$

We change the input I_1 by a small amount δI_1 and determine the change in output using the Mean-Value Theorem, as follows [87]:

$$\delta O = f(I_1 + \delta I_1, I_2) - f(I_1, I_2) \quad (8.10)$$

$$= \left. \frac{\partial f}{\partial I_1} \right|_{\bar{I}_1, I_2} \delta I_1, \quad (8.11)$$

where \bar{I}_1 is a point lying in the range of $(I_1, I_1 + \delta I_1)$. Now, it is clear from the above equation that if $|\frac{\partial f}{\partial I_1}| > 1$, then $\delta O > \delta I_1$; otherwise, $\delta O < \delta I_1$. Thus, the contribution of I_1 on the output O can be determined. In the similar way, sensitivity analysis of the second variable I_2 can also be conducted.

8.2.4 Advantages and Disadvantages of Fuzzy Logic Controller

An FLC has the following advantages:

- It is a potential tool for dealing with imprecision and uncertainty.

- It does not require an extensive mathematical formulation of the problem.
- The rules are expressed in terms of *IF ... THEN* form. Thus, it becomes easier for the user to understand the control action.

However, it suffers from the following drawbacks:

- The performance of an FLC depends mainly on its Knowledge Base (KB) and designing a proper KB of an FLC is not an easy task. The designer should have a thorough knowledge of the process to be controlled.
- The number of rules of an FLC depends on number of variables and that of the linguistic terms used to represent each variable. Thus, computational complexity of an FLC increases, when it is developed for controlling a process involving many variables.

8.3 Fuzzy Clustering

Clustering is a powerful method of data mining, whose aim is to extract useful information from a set of data. A clustering technique analyzes the pattern of the data set and groups the data into several clusters based on similarity of the pattern. Clusters may be either crisp or fuzzy in nature. The crisp clusters have well-defined and fixed boundaries among themselves, whereas the fuzzy clusters are characterized by their vague boundaries. The present chapter deals with fuzzy clustering only.

Several methods of fuzzy clustering, such as **fuzzy ISODATA** [93], **Fuzzy C-Means (FCM)** [94], **fuzzy k-nearest neighborhood algorithm** [95], **potential-based clustering** [96], **entropy-based clustering** [97], and others, have been proposed by various researchers. The principles of fuzzy C-means clustering and entropy-based fuzzy clustering are explained below, in detail.

8.3.1 Fuzzy C-Means Clustering

Fuzzy C-Means (FCM) is one of the most popular fuzzy clustering techniques, in which a particular data of the set may be the member of several clusters with different values of membership [94]. In this algorithm, an attempt is made to minimize dissimilarity measure (expressed in terms of Euclidean distance) of the data points with the pre-defined clusters. It is obvious that similarity of two data points residing in the same cluster will have a high value and two points belonging to two different clusters will be dissimilar in nature. It is an iterative algorithm, in which the cluster centers and membership values of the data points with the clusters are going to be updated to minimize the dissimilarity measure.

Let us consider L -dimensional N data points represented by x_i ($i = 1, 2, \dots, N$) (refer to Fig. 8.17), which are to be clustered. Each data point x_i ($i = 1, 2, \dots, N$) is expressed by a vector of L values (that is, $x_{i1}, x_{i2}, \dots, x_{iL}$). Let us also assume that the above data are to be clustered into C groups, where $2 \leq C \leq N$ and g ($g > 1$) is a weighting factor indicating

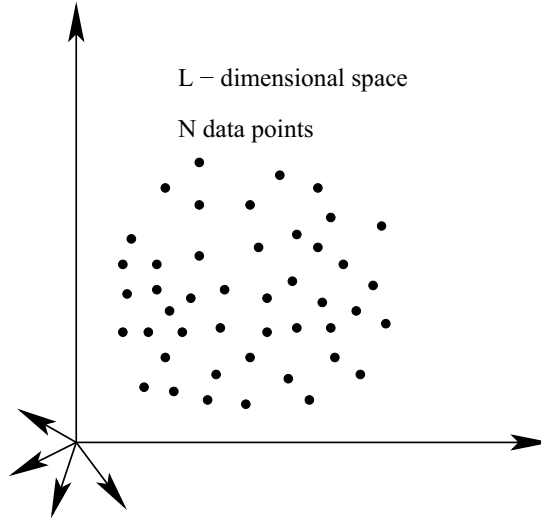


Figure 8.17: L -dimensional hyperspace containing N data points.

the level of cluster fuzziness. Let us represent the membership matrix by $[\mu]$ and it has the dimension of $N \times C$. Thus, the membership value of i -th data point with j -th cluster is represented by μ_{ij} . It is to be noted that μ_{ij} lies in the range of $(0.0, 1.0)$ and it follows the condition given below.

$$\sum_{j=1}^C \mu_{ij} = 1.0 \quad (8.12)$$

As the Euclidean distance is considered for dissimilarity measure, the objective function of the optimization problem can be written as follows:

$$F(\mu, C) = \sum_{j=1}^C \sum_{i=1}^N \mu_{ij}^g d_{ij}^2, \quad (8.13)$$

where d_{ij} is the Euclidean distance between i -th point and j -th cluster, that is calculated like the following: $d_{ij} = \|c_j - x_i\|$. Now, to accommodate the constraints given in equation (8.12), the above objective function can be re-written as follows:

$$\overline{F}(\mu, C, \lambda_1, \dots, \lambda_N) = \sum_{j=1}^C \sum_{i=1}^N \mu_{ij}^g d_{ij}^2 + \sum_{i=1}^N \lambda_i \left(\sum_{j=1}^C \mu_{ij} - 1.0 \right), \quad (8.14)$$

where λ_i , $i = 1, 2, \dots, N$, represent the Lagrange multipliers. After differentiating \overline{F} with respect to its arguments, putting them equal to zero and solving, we get the following relationships for the center of cluster j (CC_j) and membership values (μ_{ij}):

$$CC_j = \frac{\sum_{i=1}^N \mu_{ij}^g x_i}{\sum_{i=1}^N \mu_{ij}^g}, \quad (8.15)$$

$$\mu_{ij} = \frac{1}{\sum_{m=1}^C \left(\frac{d_{ij}}{d_{im}}\right)^{\frac{2}{g-1}}}. \quad (8.16)$$

It is to be noted that the cluster centers and membership values can be updated using the above equations (8.15) and (8.16).

The FCM algorithm consists of the following steps:

- Step 1: Assume the number of clusters to be made, that is, C , where $2 \leq C \leq N$.
- Step 2: Choose an appropriate level of cluster fuzziness $g > 1$.
- Step 3: Initialize the $N \times C$ sized membership matrix $[\mu]$ at random, such that $\mu_{ij} \in [0.0, 1.0]$ and $\sum_{j=1}^C \mu_{ij} = 1.0$, for each i .
- Step 4: Calculate k -th dimension of j -th cluster center CC_{jk} using the expression given below.

$$CC_{jk} = \frac{\sum_{i=1}^N \mu_{ij}^g x_{ik}}{\sum_{i=1}^N \mu_{ij}^g}. \quad (8.17)$$

- Step 5: Calculate the Euclidean distance between i -th data point and j -th cluster center like the following:

$$d_{ij} = \|(CC_j - x_i)\|. \quad (8.18)$$

- Step 6: Update fuzzy membership matrix $[\mu]$ according to d_{ij} . If $d_{ij} > 0$, then

$$\mu_{ij} = \frac{1}{\sum_{m=1}^C \left(\frac{d_{ij}}{d_{im}}\right)^{\frac{2}{g-1}}}. \quad (8.19)$$

If $d_{ij} = 0$, then the data point coincides with j -th cluster center CC_j and it will have the full membership value, that is, $\mu_{ij} = 1.0$.

- Step 7: Repeat from Step 4 to Step 6 until the changes in $[\mu]$ turn out to be less than some pre-specified values.

Using this algorithm, fuzzy clusters of the data set will be generated. The boundaries of the clusters may be such, that there will be some overlapping of two or more clusters. It is important to mention that the performance of the algorithm depends on initial membership values selected at random.

A Numerical Example:

Table 8.2 contains the coordinates of ten points lying on a free-form surface shown in Fig. 8.18. Carry out clustering using the fuzzy C-means algorithm. Assume $g = 1.25$ and termination criterion $\epsilon = 0.01$.

Table 8.2: Ten points lying on a free-form surface

Points	X-coordinate	Y-coordinate	Z-coordinate
1	0.2	0.4	0.6
2	0.4	0.3	0.8
3	0.8	0.2	0.5
4	0.9	0.5	0.4
5	0.6	0.6	0.6
6	0.3	0.4	0.5
7	0.7	0.6	0.5
8	0.2	0.5	0.3
9	0.3	0.6	0.8
10	0.8	0.3	0.1

Solution:

Number of data points to be clustered $N = 10$

Dimensions of the data $L = 3$

Level of cluster fuzziness $g = 1.25$

Let us assume that the number of clusters to be made $C = 2$. Let us also suppose that a membership matrix $[\mu]$ of size 10×2 is generated at random, as given below.

$$[\mu] = \begin{bmatrix} 0.680551 & 0.319449 \\ 0.495150 & 0.504850 \\ 0.821897 & 0.178103 \\ 0.303795 & 0.696205 \\ 0.333966 & 0.666034 \\ 0.431538 & 0.568462 \\ 0.415384 & 0.584616 \\ 0.509643 & 0.490357 \\ 0.469850 & 0.530150 \\ 0.189164 & 0.810836 \end{bmatrix}$$

Now, the first dimension of first cluster center CC_{11} is determined like the following:

$$\begin{aligned} CC_{11} &= \frac{\sum_{i=1}^N \mu_{i1}^g x_{i1}}{\sum_{i=1}^N \mu_{i1}^g} \\ &= \frac{A}{B}, \end{aligned}$$

where $A = 0.680551^{1.25} \times 0.2 + 0.495150^{1.25} \times 0.4 + 0.821897^{1.25} \times 0.8 + 0.303795^{1.25} \times 0.9 + 0.333966^{1.25} \times 0.6 + 0.431538^{1.25} \times 0.3 + 0.415384^{1.25} \times 0.7 + 0.509643^{1.25} \times 0.2 + 0.469850^{1.25} \times 0.3 + 0.189164^{1.25} \times 0.8 = 1.912120$,

$B = 0.680551^{1.25} + 0.495150^{1.25} + 0.821897^{1.25} + 0.303795^{1.25} + 0.333966^{1.25} + 0.431538^{1.25} +$

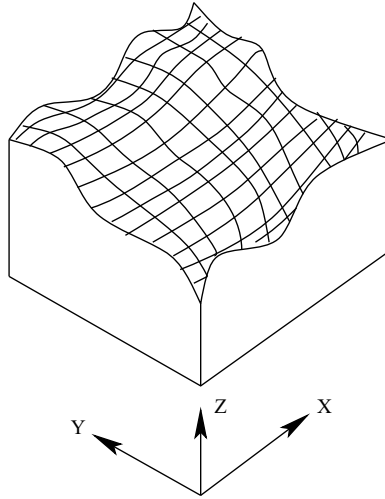


Figure 8.18: A free-form surface.

$$0.415384^{1.25} + 0.509643^{1.25} + 0.469850^{1.25} + 0.189164^{1.25} = 3.923066.$$

Therefore, $CC_{11} = 0.487404$.

Similarly, we determine $CC_{12} = 0.412837$, $CC_{13} = 0.543316$, $CC_{21} = 0.554860$, $CC_{22} = 0.459073$ and $CC_{23} = 0.477262$.

Thus, the coordinates of two cluster centers are found to be as follows:

$$\begin{aligned} c_1 &: (0.487404, 0.412837, 0.543316) \\ c_2 &: (0.554860, 0.459073, 0.477262) \end{aligned}$$

We determine the modified values of membership grades like the following:

$$\mu_{11} = \frac{1}{1 + \left(\frac{d_{11}}{d_{12}}\right)^{\frac{2}{g-1}}},$$

where $d_{11} = \sqrt{(0.487404 - 0.2)^2 + (0.412837 - 0.4)^2 + (0.543316 - 0.6)^2} = 0.293221526$,
 $d_{12} = \sqrt{(0.554860 - 0.2)^2 + (0.459073 - 0.4)^2 + (0.477262 - 0.6)^2} = 0.380105056$.

Therefore, $\mu_{11} = \frac{1}{1 + \left(\frac{d_{11}}{d_{12}}\right)^8} = 0.888564$, corresponding to $g = 1.25$. The values of other elements of $[\mu]$ can be calculated in the same way and these are found to be as follows:

$$[\mu] = \begin{bmatrix} 0.888564 & 0.111436 \\ 0.909207 & 0.090793 \\ 0.376533 & 0.623467 \\ 0.142926 & 0.857074 \\ 0.217307 & 0.782693 \\ 0.922229 & 0.077771 \\ 0.060962 & 0.939038 \\ 0.562085 & 0.437915 \\ 0.788303 & 0.211697 \\ 0.232446 & 0.767554 \end{bmatrix}$$

It completes one iteration of the algorithm and this will continue until the termination criterion: change in μ , that is, $\Delta\mu \leq \epsilon$ is reached.

After a few iterations, the cluster-centers will be obtained as follows:

$$\begin{aligned} c_1 &: (0.296189, 0.447977, 0.600323) \\ c_2 &: (0.773939, 0.425782, 0.404034) \end{aligned}$$

and the values of membership grades are determined like the following:

$$[\mu] = \begin{bmatrix} 0.999999 & 0.000001 \\ 0.997102 & 0.002898 \\ 0.001223 & 0.998777 \\ 0.000008 & 0.999992 \\ 0.351332 & 0.648668 \\ 0.999992 & 0.000008 \\ 0.002764 & 0.997236 \\ 0.992437 & 0.007563 \\ 0.999452 & 0.000548 \\ 0.001836 & 0.998164 \end{bmatrix}$$

Thus, the first cluster will be composed of 1-st, 2-nd, 6-th, 8-th and 9-th data points and the second cluster will contain 3-rd, 4-th, 5-th, 7-th and 10-th data points.

8.3.2 Entropy-based Fuzzy Clustering

Here, entropy (probability) values of the data points are determined based on a similarity measure [97]. The similarity of two data points depends on the distance (say Euclidean distance) between them. The data point with the minimum entropy value is selected as the cluster center. The data points having similarity with this cluster center more than a pre-specified value will form a cluster. It is based on the philosophy that the data points having high similarity with the cluster center should belong to the same cluster with a high probability. This algorithm has been explained below, in detail.

Let us consider a set of N data points in an $L - D$ hyperspace (refer to Figure 8.17). The following steps are used to determine entropy of each point:

- Arrange the data set in N rows and L columns.
- Calculate the Euclidean distances between the points i and j as follows:

$$d_{ij} = \sqrt{\sum_{k=1}^L (x_{ik} - x_{jk})^2}. \quad (8.20)$$

It is to be noted that N^2 distances are possible among N data points. It is also interesting to note that out of N^2 distances, ${}^N C_2$ distances belong to d_{ij} and d_{ji} each and there are N diagonal elements of the distance matrix. As d_{ij} is equal to d_{ji} and the diagonal elements of distance matrix are turning out to be equal to zero, it is required to calculate ${}^N C_2$ distances (d_{ij}) only.

- Determine the similarity S_{ij} between two data points (i and j) using the expression given below.

$$S_{ij} = e^{-\alpha d_{ij}}, \quad (8.21)$$

where α is a constant to be determined as follows: We assume a similarity S_{ij} of 0.5, when the distance between two data points d_{ij} becomes equal to mean distance of all pairs of data points, that is, $d_{ij} = \bar{d}$, where $\bar{d} = \frac{1}{{}^N C_2} \sum_{i=1}^N \sum_{j=1}^{j < i, j \neq i} d_{ij} = \frac{1}{{}^N C_2} \sum_{i=1}^N \sum_{j > i} d_{ij}$.

From equation (8.21), we can write

$$0.5 = e^{-\alpha \bar{d}}$$

Taking log (ln) on both sides, we get

$$\alpha = \frac{\ln 2}{\bar{d}}. \quad (8.22)$$

It is important to note that similarity S_{ij} varies in the range of 0.0 to 1.0.

- Calculate the entropy of all N data points separately as follows: Entropy of a data point with respect to another data point is determined using the expression given below.

$$E = -S \log_2 S - (1 - S) \log_2 (1 - S). \quad (8.23)$$

From equation (8.23), it can be observed that entropy E becomes equal to 0.0, for a value of $S = 0$ and $S = 1.0$. Moreover, entropy E takes the maximum value of 1.0 and it occurs, when S becomes equal to 0.5. Thus, the entropy of a point with respect to another point varies between 0.0 and 1.0.

- Determine total entropy value at a data point x_i with respect to all other data points using the expression given below.

$$E_i = - \sum_{\substack{j \neq i \\ j \in x}} (S_{ij} \log_2 S_{ij} + (1 - S_{ij}) \log_2 (1 - S_{ij})) \quad (8.24)$$

During clustering, the point having the minimum total entropy may be selected as the cluster center.

Clustering Algorithm:

Let us suppose that $[T]$ is the input data set containing N points and each data point has L dimensions. Thus, $[T]$ is an $N \times L$ matrix. The clustering algorithm consists of the following steps:

- **Step 1:** Calculate entropy E_i for each data point x_i lying in $[T]$.
- **Step 2:** Identify x_i that has the minimum E_i value and select it ($x_{i,min}$) as the cluster center.
- **Step 3:** Put $x_{i,min}$ and the data points having similarity with $x_{i,min}$ greater than β (which is a threshold value of similarity) in a cluster and remove them from $[T]$.
- **Step 4:** Check if $[T]$ is empty. If yes, terminate the program, else go to Step 2.

In the above algorithm, entropy has been defined in such a way that a data point, which is far away from the rest of the data, may also be selected as a cluster center. It may happen so, because a very distant point (from the rest of the data points) may have a low value of entropy. To overcome this, another parameter γ (in %) is introduced, which is nothing but a threshold used to declare a cluster to be a valid one. After the clustering is over, we count the number of data points lying in each cluster and if this number becomes greater than or equal to $\gamma\%$ of total number of data points, we declare this cluster to be a valid one. Otherwise, these data points, which are unable to form a valid cluster, will be declared as the outliers.

A Numerical Example:

The coordinates of ten points lying on a free-form surface are found to be as shown in Table 8.2. Carry out fuzzy clustering based on their similarity and entropy values. Assume $\beta = 0.5$ and $\gamma = 10\%$.

Solution:

We first calculate Euclidean distances between the points using equation (8.20). As there are 10 points, a total of $10 \times 10 = 100$ distances are possible and they can be expressed as a 10×10 matrix. It is to be noted that out of these 100 distances, there are 10 diagonal elements (such as $d_{00}, d_{11}, d_{22}, d_{33}, \dots, d_{99}$), whose values are equal to 0.0. Thus, there are effectively $100 - 10 = 90$ elements, whose numerical values are non-zero. As the distance between the points i and j , that is, d_{ij} is the same with that between the points j and i , that is, d_{ji} , a total of ${}^{10}C_2 = 45$ distance values are to be really calculated. Table 8.3 shows those Euclidean distance values. The mean of the distance values, that is, \bar{d} is found to be equal to 0.518373. The value of α is determined using equation (8.22) and it is seen to be

Table 8.3: Euclidean distance and similarity values

Combination of data points	Euclidean distance	Similarity
0,1	0.3000000	0.669551
0,2	0.640312	0.424773
0,3	0.734847	0.374334
0,4	0.447214	0.549912
0,5	0.141421	0.827701
0,6	0.547723	0.480757
0,7	0.316228	0.655179
0,8	0.300000	0.669551
0,9	0.787401	0.348931
1,2	0.509902	0.505695
1,3	0.670820	0.407793
1,4	0.412311	0.576186
1,5	0.331662	0.641795
1,6	0.519615	0.499170
1,7	0.574456	0.463875
1,8	0.316228	0.655179
1,9	0.806226	0.340257
2,3	0.331662	0.641795
2,4	0.458258	0.541851
2,5	0.538516	0.486712
2,6	0.412311	0.576186
2,7	0.700000	0.392189
2,8	0.707107	0.388479
2,9	0.412311	0.576186
3,4	0.374166	0.606337
3,5	0.616441	0.438550
3,6	0.244949	0.720697
3,7	0.707107	0.388479
3,8	0.728011	0.377771
3,9	0.374166	0.606337
4,5	0.374166	0.606337
4,6	0.141421	0.827701
4,7	0.509902	0.505695
4,8	0.360555	0.617473
4,9	0.616441	0.438550
5,6	0.447214	0.549912
5,7	0.244949	0.720697
5,8	0.360555	0.617473
5,9	0.648074	0.420387
6,7	0.547723	0.480757
6,8	0.500000	0.512436
6,9	0.509902	0.505695
7,8	0.519615	0.499170
7,9	0.663325	0.411901
8,9	0.911043	0.295759

equal to 1.337160. The similarity values are also calculated using equation (8.21) and those are also shown in Table 8.3. The entropy values of the data points are then calculated using equation (8.24) and those are found to be like the following:

$$\begin{aligned} E_0 &= 8.285456, E_1 = 8.665640, \\ E_2 &= 8.815512, E_3 = 8.568537, \\ E_4 &= 8.516900, E_5 = 8.348550, \\ E_6 &= 8.490743, E_7 = 8.686420, \\ E_8 &= 8.560217, E_9 = 8.632634. \end{aligned}$$

It is observed from the above values of entropy that 0-th point has the least entropy and thus, it is declared to be the first cluster center. Now, corresponding to a threshold value of similarity $\beta = 0.5$, 1-st, 4-th, 5-th, 7-th and 8-th points are found to make cluster with the 0-th point. The remaining points are as follows: 2-nd, 3-rd, 6-th, 9-th. Out of these four points, the 6-th point is seen to have the least value of entropy. Thus, it is declared as the center of second cluster. For $\beta = 0.5$, 2-nd, 3-rd and 9-th points are found to make cluster with the 6-th point. As all ten points have been clustered (the first cluster contains six point and the second cluster consists of four points), the number of outliers is found to be equal to zero.

Comparison of Fuzzy C-Means Algorithm and Entropy-Based Fuzzy Clustering Algorithm [98]:

In entropy-based fuzzy clustering algorithm, the number of clusters, number of outliers, quality of the clusters (determined in terms of compactness and distinctness) are dependent on the threshold value of similarity. On the other hand, in fuzzy C-means algorithm, the quality of clusters may depend on the initial matrix of membership values selected at random. It is interesting to note that entropy-based fuzzy clustering algorithm is very sensitive to the threshold value of similarity and thus, it is more flexible compared to the fuzzy C-means algorithm. It is also to be noted that compactness and distinctness of the clusters are decided based on the intra-cluster and inter-cluster distances of the elements, respectively. In general, entropy-based fuzzy clustering algorithm is able to yield less compact but more distinct clusters (if the parameters are set properly) compared to those obtained by the fuzzy C-means algorithm. However, the performances of clustering algorithms are found to be data-dependent.

Note: To perform clustering of the large spatial data set, some special types of clustering algorithms were proposed. Ng and Han [99] suggested a crisp clustering algorithm for the large data sets utilizing the concept of k representative objects (known as **medoids**). The medoids were determined from the data set, so that the sum of dissimilarities within a cluster became the minimum. The algorithm tried to find a new set of k medoids lying in the neighborhood of the current set of the same. This algorithm is popularly known as CLARANS (Clustering Large Applications based on RANdomized Search). Ester et al. [100] developed a density-based clustering algorithm called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) for the large data sets. A cluster can be

easily identified, as its density of points is higher than that of the outside. The density of noise-region is generally less than that of a cluster. A cluster is declared to be a valid one, if each point of it has a minimum number of neighborhood points. It is to be noted that DBSCAN could outperform CLARANS algorithm in terms of its ability to form clusters of arbitrary shapes.

8.4 Summary

This chapter has been summarized as follows:

1. Fuzzy set theory has been used in a number of ways. Out of all such applications, the present chapter concentrates on two of them, namely fuzzy reasoning and clustering.
2. Fuzzy logic controller is a potential tool for dealing with imprecision and uncertainty. Two of the most popular forms of fuzzy logic controller, such as Mamdani Approach and Takagi & Sugeno's Approach, have been explained in this chapter. Mamdani Approach is characterized by its high interpretability and low accuracy, whereas the aim of Takagi & Sugeno's Approach is to obtain a high accuracy, which may be possible to achieve at the cost of interpretability.
3. The number of rules of an FLC increases exponentially with the number of variables, which increases computational complexity of the algorithm. Due to this problem of rule explosion, a conventional FLC is unable to solve complex real-world problems involving a large number of variables. To overcome this difficulty, the concept of hierarchical FLC has been introduced, in which the number of rules is reduced significantly.
4. Clustering is a powerful tool for data mining. The clusters may be either crisp (having well-defined and fixed boundaries) or fuzzy (having vague boundaries) in nature. This chapter deals with the fuzzy clusters only.
5. The principles of fuzzy C-means algorithm and entropy-based fuzzy clustering algorithms have been explained, in detail. Entropy-based fuzzy clustering algorithm is more flexible compared to the fuzzy C-means algorithm. The performances of clustering algorithms are seen to be data-dependent.

8.5 Exercise

1. Explain briefly the principle of Mamdani Approach of fuzzy logic controller. State its advantages and disadvantages.
2. Discuss briefly various methods of defuzzification.
3. Explain the principle of Takagi and Sugeno's Approach of FLC.

4. When and why do we go for hierarchical FLC ?
5. What do you mean by fuzzy clustering ?
6. Explain the principles of fuzzy C-means algorithm and entropy-based fuzzy clustering algorithm.
7. There are two inputs (I_1 and I_2) and one output (O) of a process. It is required to develop a fuzzy logic-based expert system based on Mamdani Approach. Let us assume that the inputs and output are expressed using three linguistic terms, namely Low (LW), Medium (M) and High (H). The membership function distributions of the above inputs and output are shown in Fig. 8.19. The rule base of the fuzzy logic

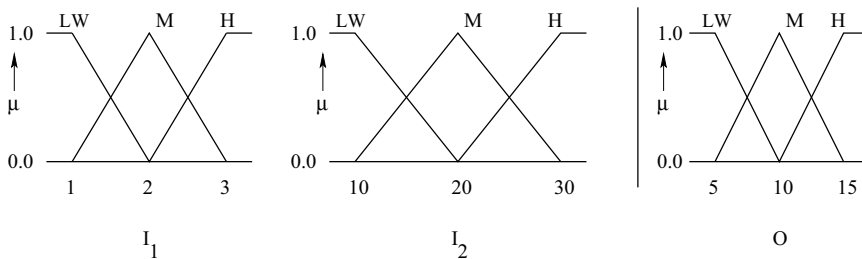


Figure 8.19: Data Base of the fuzzy logic controller.

Table 8.4: Rule Base of the fuzzy logic controller

		I_2		
		LW	M	H
I_1	LW	LW	LW	M
	M	LW	M	H
	H	M	H	H

controller is shown in Table 8.4. Determine the output of the controller for a set of inputs: $I_1 = 1.6$, $I_2 = 22.0$. Use the following methods of defuzzification: (i) Center of sums method, (ii) Centroid method, (iii) Mean of maxima method.

8. An expert system based on Takagi and Sugeno's approach of FLC is to be developed to predict the output of a process. The membership function distributions of the input and output variables are shown in Fig. 8.20. As there are two inputs: I_1 and I_2 , and each input is represented using three linguistic terms (such as, LW , M , H), there is a maximum of $3 \times 3 = 9$ feasible rules. The output of i -th rule, that is, y^i ($i = 1, 2, \dots, 9$) is expressed as follows:

$$y^i = f(I_1, I_2) = a_j^i I_1 + b_k^i I_2,$$

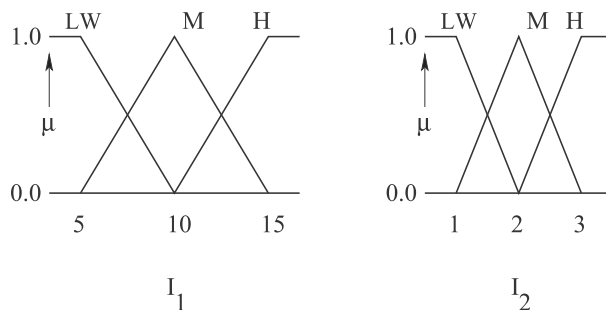


Figure 8.20: Data base of the FLC.

where $j, k = 1, 2, 3$; $a_1^i = 1.1$, $a_2^i = 1.2$ and $a_3^i = 1.3$, if I_1 is found to be LW , M and H , respectively; $b_1^i = 1.5$, $b_2^i = 1.7$ and $b_3^i = 1.9$, if I_2 is considered to be LW , M and H , respectively. Calculate the output of the FLC for the inputs: $I_1 = 8.0$, $I_2 = 2.5$.

9. Table 8.5 shows a set of 20×4 data.

Table 8.5: A set of 20×4 data

5.2	3.6	1.5	0.2
4.9	3.1	1.5	0.2
4.7	3.2	1.3	0.2
4.6	3.4	1.4	0.3
5.4	3.7	1.5	0.2
5.2	3.5	1.5	0.2
5.2	3.4	1.4	0.2
4.7	3.2	1.6	0.2
5.5	4.2	1.4	0.2
4.5	2.3	1.3	0.3
5.2	3.9	1.9	0.5
4.8	3.0	1.4	0.3
5.7	2.5	5.0	2.0
5.8	2.8	5.1	2.4
6.1	3.0	4.9	1.8
7.9	3.8	6.4	2.0
6.3	3.4	5.6	2.4
6.4	3.1	5.5	1.8
6.3	3.5	5.5	2.4
5.9	3.0	5.1	1.8

- Carry out clustering of this data set using fuzzy C-means algorithm. Assume the level of fuzziness $g = 1.5$ and termination criterion $\epsilon = 0.001$.
- Determine the clusters and outliers (if any) using entropy-based fuzzy clustering algorithm. Assume the threshold value of similarity $\beta = 0.52$ and $\gamma = 10\%$.

Chapter 9

Fundamentals of Neural Networks

This chapter explains the working principle of a biological neuron. An artificial neuron has been modeled by copying the principle of a biological neuron. After introducing a few transfer functions generally used in neural networks, a layer of neurons has been designed. A network consisting of multiple layers has then been introduced. The difference between static and dynamic neural networks has been pointed out. After discussing the principle of supervised and un-supervised learning schemes, the mechanisms of incremental and batch modes of training have been explained.

9.1 Introduction

Biological nervous system consists of a large number of interconnected processing units called **neurons** (each has a size of around $100\ \mu m$) operating in parallel. For example, human brain contains approximately 10^{11} neurons communicating with each other with the help of electrical impulses. Thus, the brain may be considered as a highly complex parallel computer. Artificial Neural Network (ANN) has been designed by copying the human brain artificially. The concept of Neural Network (NN) was introduced through the work of McCulloch and Pitts [101] in 1943. An ANN is an architecture consisting of a large number of neurons organized in different layers and the neurons of one layer are connected to those of another layer by means of weights. An NN can be trained to perform a particular task by making proper adjustment of its architecture, connecting weights and other parameters. Before we proceed further, let us try to understand the working principle of a biological neuron.

9.1.1 Biological Neuron

Fig. 9.1 shows the schematic view of a biological neuron. It consists of a bush of thin fibers called **dendrites**, **cell body** (also known as **soma**), a long cylindrical fiber known as **axon**, **synapse**, and others. A synapse is that part of a neuron, where its axon makes contact with the dendrites of its neighboring neuron. A neuron collects information from

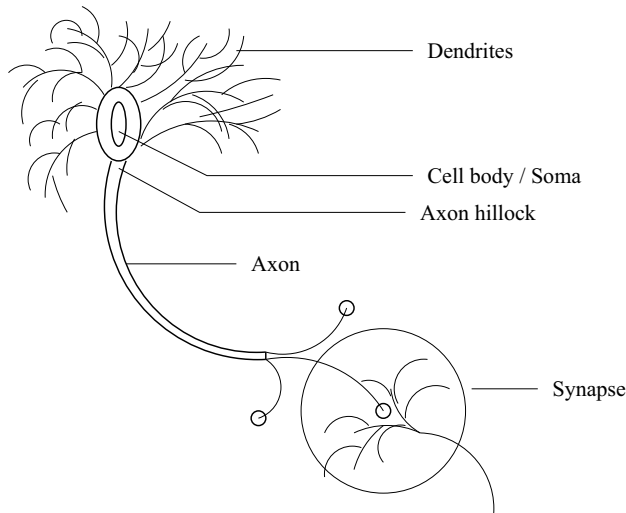


Figure 9.1: A schematic view showing a biological neuron.

its neighbors with the help of its dendrites. The collected information is then summed up in the cell body, before it passes through the axon. This information is then transferred to the next neuron through the synapse using the difference in concentration of Na^+ , K^+ ions between them.

9.1.2 Artificial Neuron

Fig. 9.2 displays the schematic view of an artificial neuron, in which a biological neuron has been modeled artificially. Let us suppose that there are n inputs (such as I_1, I_2, \dots, I_n) to a neuron j . The weights connecting n number of inputs to j -th neuron are represented by $[W] = [W_{1j}, W_{2j}, \dots, W_{nj}]$. The function of summing junction of an artificial neuron is to collect the weighted inputs and sum them up. Thus, it is similar to the function of combined dendrites and soma. The activation function (also known as the transfer function) performs the task of axon and synapse. The output of summing junction may sometimes become equal to zero and to prevent such a situation, a bias of fixed value b_j is added to it. Thus, the input to transfer function f is determined as $u_j = \sum_{k=1}^n I_k W_{kj} + b_j$. The output of j -th neuron, that is, O_j can be obtained as follows:

$$O_j = f(u_j) = f\left(\sum_{k=1}^n I_k W_{kj} + b_j\right). \quad (9.1)$$

In an ANN, the output of a neuron largely depends on its transfer function. Different types of transfer function are in use, such as hard-limit, linear, log-sigmoid, tan-sigmoid, and others (refer to Fig. 9.3), which are discussed below.

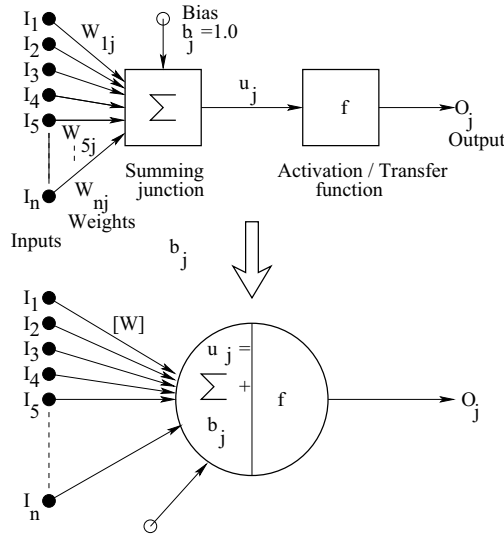


Figure 9.2: A schematic view showing an artificial neuron.

Hard-limit Transfer Function:

It generates either 1.0 or 0.0 depending on its input u (refer to Fig. 9.3 (a)). The output O is calculated from input u like the following:

$$O = \begin{cases} 0.0 & \text{if } u < 0.0, \\ 1.0 & \text{otherwise.} \end{cases} \quad (9.2)$$

If u is found to be less than 0.0, the output of this transfer function becomes equal to 0.0; otherwise it yields 1.0. It is generally used in a **perceptron** neuron.

Linear Transfer Function:

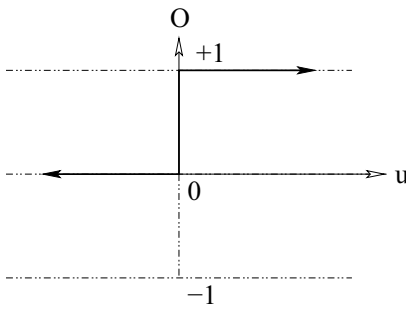
The output of this transfer function is made equal to its input and it lies in the range of (-1.0 to 1.0). The input-output relationship of this transfer function may be expressed like the following (refer to Fig. 9.3 (b)):

$$O = u. \quad (9.3)$$

It is generally utilized in a **linear filter**.

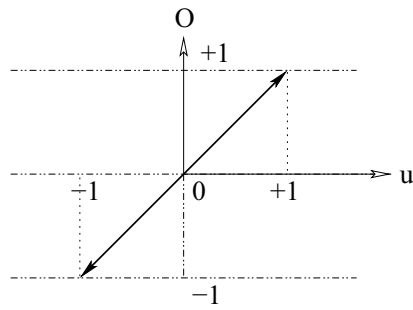
Log-Sigmoid Transfer Function:

It generates the output lying in a range of (0.0, 1.0) and becomes equal to 0.5, corresponding to an input $u = 0.0$. The output O of the transfer function can be expressed as the function



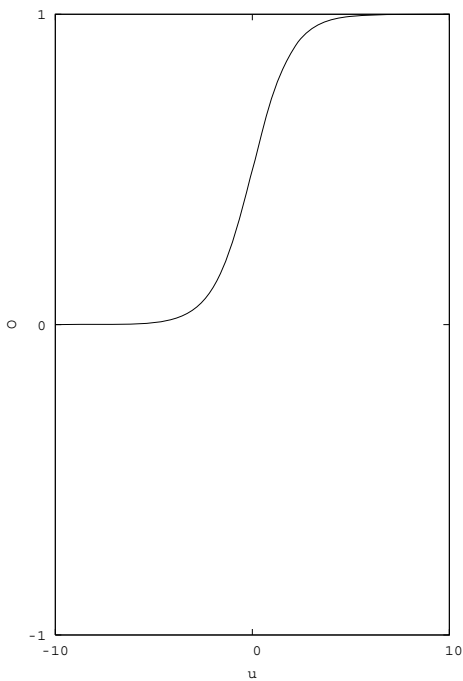
(a)

(a) Hard-limit transfer function



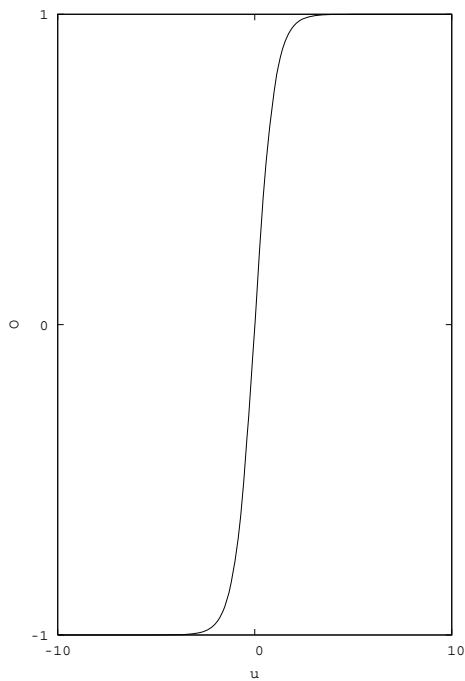
(b)

(b) Linear transfer function



(c)

(c) Log-sigmoid transfer function



(d)

(d) Tan-sigmoid transfer function

Figure 9.3: Some of the transfer functions generally used in NN.

of input u as follows (refer to Fig. 9.3 (c)):

$$O = \frac{1}{1 + e^{-au}}, \quad (9.4)$$

where a represents the coefficient of transfer function. It is obvious that the nature of distribution of this transfer function depends on the value of a . It is generally used in a Back-Propagation Neural Network (BPNN).

Tan-Sigmoid Transfer Function:

This transfer function yields an output lying in the range of $(-1.0, 1.0)$. The input-output relationship can be expressed as follows (refer to Fig. 9.3 (d)):

$$O = \frac{e^{au} - e^{-au}}{e^{au} + e^{-au}}, \quad (9.5)$$

where a is the coefficient of transfer function. It generates zero output, corresponding to its input $u = 0.0$. It is frequently used in BPNN.

The neurons are connected by the weights and depending on the structure/topology of the networks, they are generally classified into several types, namely feed-forward NN, recurrent NN, self-organizing map, and others, which have been explained in the next chapter.

9.1.3 A Layer of Neurons

Fig. 9.4 shows a layer of p neurons. Let us consider that there are n inputs, such as

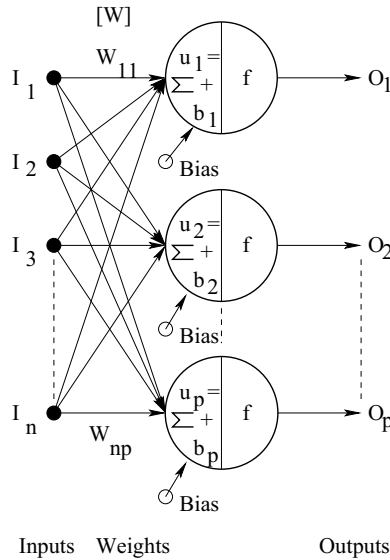


Figure 9.4: A schematic view showing a layer of neurons.

I_1, I_2, \dots, I_n , connected to this layer of neurons through the weights $[W]$. Thus, the weights

$[W]$ can be represented as follows:

$$[W] = \begin{bmatrix} W_{11} & W_{12} & \dots & W_{1p} \\ W_{21} & W_{22} & \dots & W_{2p} \\ \vdots & \vdots & & \vdots \\ W_{n1} & W_{n2} & \dots & W_{np} \end{bmatrix}$$

The output of k -th neuron can be determined like the following:

$$O_k = f\left(\sum_{j=1}^n I_j W_{jk} + b_k\right), \quad (9.6)$$

where $k = 1, 2, \dots, p$ and b_k represents the bias value of k -th neuron.

9.1.4 Multiple Layers of Neurons

Fig. 9.5 shows the schematic view of a network consisting of three layers, where each layer contains p neurons. The inputs: I_1, I_2, \dots, I_n are fed to the network and the weights between the inputs and first layer of neurons are represented by $[W]^1$. Similarly, the weights between the first and second layers and those between the second and third layers are denoted by $[W]^2$ and $[W]^3$, respectively. Let us assume that f^1 , f^2 and f^3 represent the transfer

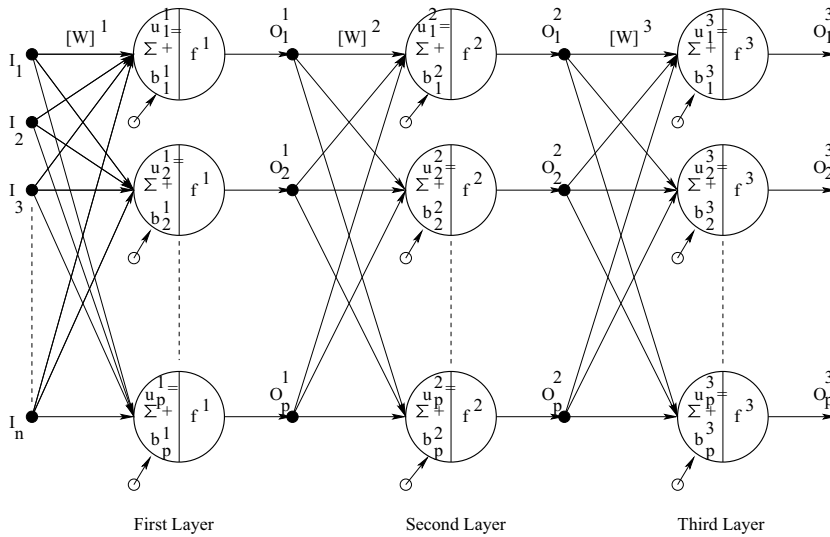


Figure 9.5: A schematic view showing multiple layers of neurons.

functions of neurons lying on the first, second and third layers, respectively. Similarly, the bias values of p -th neuron lying on the first, second and third layers are indicated by b_p^1 , b_p^2

and b_p^3 , respectively. Moreover, the outputs of k -th, l -th and m -th neurons of first, second and third layers are denoted by O_k^1 , O_l^2 and O_m^3 , respectively, and these are found to be as follows:

$$O_k^1 = f^1\left(\sum_{j=1}^n I_j W_{jk}^1 + b_k^1\right) \quad (9.7)$$

$$O_l^2 = f^2\left(\sum_{j=1}^p O_j^1 W_{jl}^2 + b_l^2\right) \quad (9.8)$$

$$O_m^3 = f^3\left(\sum_{j=1}^p O_j^2 W_{jm}^3 + b_m^3\right) \quad (9.9)$$

9.2 Static vs. Dynamic Neural Networks

A Neural Network (NN) can be either static or dynamic in nature. In a static NN, the output is not compared with its target value to determine an error, if any. Thus, the error in prediction is neither calculated nor fed back for updating the neural network.

In a dynamic NN, the error (that is, the difference between the target and calculated outputs) is determined and then fed back to the network to modify its architecture and update the weights. Fig. 9.6 shows the schematic view of a dynamic NN. A dynamic NN

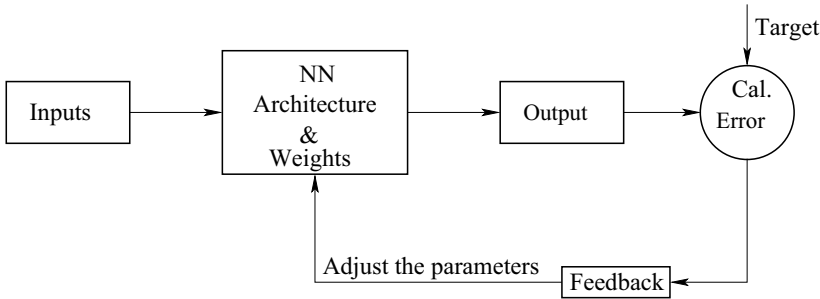


Figure 9.6: A schematic view showing a dynamic NN.

has a chance of error compensation and thus, it may yield better predictions compared to the static NN can do.

9.3 Training of Neural Networks

The process of modifying a neural network by updating its weights, biases and other parameter, if any, is known as training. During the training, the parameters of the network are optimized and as a result of which, it undergoes through an internal process of curve fitting. It is then said that the network has passed through a learning phase. Learning may be either **supervised** or **un-supervised** in nature, which are explained below.

9.3.1 Supervised Learning

This form of learning is possible to implement, if the input-output relationships of the training scenarios are available. Here, the output of a network is compared with the corresponding target value and the error is determined. It is then fed back to the network for updating of the same through its error minimization. It may also be called a learning with the help of the teachers.

9.3.2 Un-supervised Learning

If the target output is not available, the error in prediction cannot be determined and in such a situation, the network passes through a self-organizing process. It is known as un-supervised learning. It may also be declared as a learning without a teacher.

A network is generally trained using either an incremental (also known as a sequential) or a batch mode, the principles of which are discussed below.

9.3.3 Incremental Training

Here, a particular training scenario is passed through the network and depending on its output, the error is calculated using the corresponding target value. The said error is then propagated in the backward direction to update the connecting weights of the neurons and biases. In this approach, the network is modified utilizing the training scenarios sent one after another in a sequence. It resembles to the situation, in which the training scenarios are collected on-line and sent to the network to be trained one after another. This mode of training is also known as **on-line** training.

Let us consider the incremental training of an NN using a number of scenarios (say 20), sent one after another. There is a chance that the optimal network obtained after passing 20-th training scenario will be too much different from that got after using 1-st training scenario.

9.3.4 Batch Mode of Training

In this approach, the whole training set consisting of a large number of scenarios, is passed through the network and an average error in predictions is determined. It is important to mention that the whole training set mentioned above may also be called an **epoch**. The average error in prediction is then propagated back to update the weights and bias values of the network, so that it can yield a more accurate prediction. In this mode of training, the necessary data set is to be collected before the actual commencement of training. It is also known as an **off-line** training.

As the network is optimized using an average error in predictions, there is a chance of the network of being adaptive in nature. The adaptability generally grows due to interpolation capability of the trained network.

It is important to mention that incremental training is easier to implement and computationally faster than the batch mode of training.

9.4 Summary

This chapter starts with an explanation of the working principle of a biological neuron. An artificial neuron has been designed following the working principle of a biological neuron. After discussing a few transfer functions generally used in the neural network, a layer consisting of a few neurons has been modeled. A network has been constructed then using three layers of neurons. A static NN does not have any feedback, whereas a dynamic NN updates itself based on the principle of error minimization (feedback). The principles of supervised and un-supervised schemes of learning have been explained and both incremental as well as batch modes of training have been discussed.

9.5 Exercise

1. How do we model an artificial neuron from a biological neuron ?
2. What is the role of an activation function in neural networks ?
3. Why do we use a bias value in neural networks ?
4. Draw the architecture of a perceptron neuron.
5. Draw the architecture of a linear filter.
6. State the differences between a static neural network and a dynamic neural network.
7. Explain the principles of supervised and un-supervised learning schemes of a neural network.
8. State the differences between incremental and batch modes of training of a neural network.

Chapter 10

Some Examples of Neural Networks

In this chapter, the principles of both feed-forward as well as recurrent neural networks have been explained. Moreover, the mechanism of back-propagation learning algorithm has been discussed in detail. The principle of counter-propagation neural network has also been explained in this chapter.

10.1 Introduction

A Neural Network (NN) is a network of some processing elements called neurons that can be used to determine input-output relationships of a complex process. It may be considered as a non-linear statistical data modeling tool. Various types of neural networks are in use, such as **Feed-Forward Neural Networks (FFNNs)**, **Recurrent Neural Networks (RNNs)**, **Stochastic Neural Networks (SNNs)**, **Modular Neural Networks (MNNs)**, and others. The present chapter deals with the feed-forward and recurrent NNs only. In FFNNs, information is passed into one direction, that is, starting from the input layer towards the output layer through the hidden layer(s). So, it does not form any cycle or loop. On the other hand, in RNNs, information is processed in both the directions, namely feed-forward and feedback. Thus, it forms a loop.

A few FFNNs, such as Multi-Layer Feed-Forward Neural Networks (MLFFNNs), Radial Basis Function Networks (RBFNs), Self-Organizing Map (SOM), Counter-Propagation Neural Network (CPNN) have been discussed in this chapter. Moreover, the principles of some RNNs like Elman network and Jordan network will be explained.

10.2 Multi-Layer Feed-Forward Neural Network (MLFFNN)

The universal approximation theorem of NNs states that a standard multi-layer feed-forward network with a single hidden layer containing a finite number of neurons with arbitrary

activation function is a universal approximator [102]. Fig. 10.1 shows an NN consisting of three layers, namely input, hidden and output layers, that contain M , N and P number of neurons, respectively. Let us assume that the neurons of input, hidden and output layers

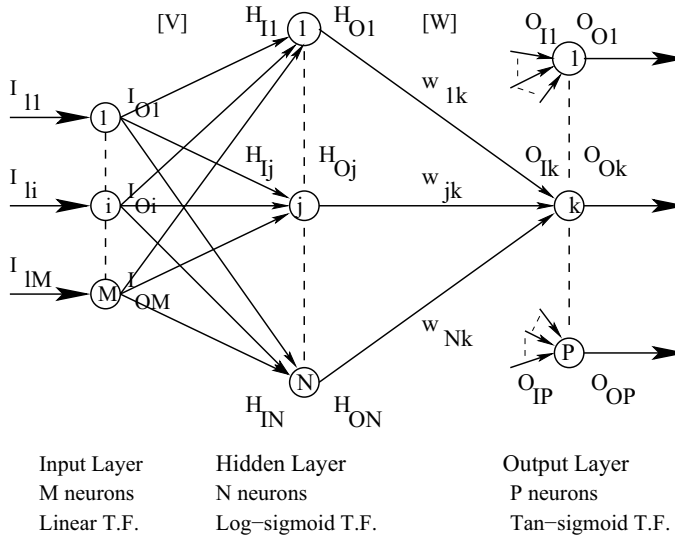


Figure 10.1: Multi-layer feed-forward neural network.

have linear, log-sigmoid and tan-sigmoid transfer functions, respectively. The symbols: I_{I1} , H_{I1} and O_{I1} indicate the inputs of 1-st neuron lying on the input, hidden and output layers, respectively. Similarly, the output of 1-st neuron belonging to the input, hidden and output layers are represented by I_{O1} , H_{O1} and O_{O1} , respectively. Let us consider a specific neuron in each layer, say i -th, j -th and k -th neurons of the input, hidden and output layers, respectively. The connecting weight between i -th neuron of input layer and j -th neuron of hidden layer is denoted by v_{ij} . Similarly, w_{jk} represents the connecting weight between j -th neuron of hidden layer and k -th neuron of output layer. The weight matrices: $[V]$ and $[W]$ can be written as follows:

$$[V] = \begin{bmatrix} v_{11} & \dots & v_{1j} & \dots & v_{1N} \\ \vdots & & \vdots & & \vdots \\ v_{i1} & \dots & v_{ij} & \dots & v_{iN} \\ \vdots & & \vdots & & \vdots \\ v_{M1} & \dots & v_{Mj} & \dots & v_{MN} \end{bmatrix}$$

$$[W] = \begin{bmatrix} w_{11} & \dots & w_{1k} & \dots & w_{1P} \\ \vdots & & \vdots & & \vdots \\ w_{j1} & \dots & w_{jk} & \dots & w_{jP} \\ \vdots & & \vdots & & \vdots \\ w_{N1} & \dots & w_{Nk} & \dots & w_{NP} \end{bmatrix}$$

It is to be noted that $[V]$ is an $M \times N$ matrix, whereas $[W]$ is an $N \times P$ matrix.

10.2.1 Forward Calculation

It consists of the following steps:

- **Step 1: Determination of the outputs of input layer**

The outputs of the neurons lying on input layer are the same with the corresponding inputs as given below.

$$I_{Oi} = I_{Ii}, \quad (10.1)$$

where $i = 1, 2, \dots, M$.

- **Step 2: Calculation of the inputs of hidden layer**

The input of j -th hidden neuron can be calculated like the following:

$$H_{Ij} = v_{1j}I_{O1} + \dots + v_{ij}I_{Oi} + \dots + v_{Mj}I_{OM}, \quad (10.2)$$

where $j = 1, 2, \dots, N$.

- **Step 3: Determination of the outputs of hidden neurons**

The hidden neurons are assumed to have log-sigmoid transfer function. The output of j -th hidden neuron can be obtained as follows:

$$H_{Oj} = \frac{1}{1 + e^{-a_1 H_{Ij}}}, \quad (10.3)$$

where a_1 is the coefficient of the transfer function.

- **Step 4: Determination of the inputs of output layer**

The input of k -th output neuron can be determined like the following:

$$O_{Ik} = w_{1k}H_{O1} + \dots + w_{jk}H_{Oj} + \dots + w_{Nk}H_{ON}, \quad (10.4)$$

where $k = 1, 2, \dots, P$.

- **Step 5: Estimation of the outputs of output layer**

The neurons lying on output layer are assumed to have tan-sigmoid transfer function. The output of k -th output neuron can be estimated as follows:

$$O_{Ok} = \frac{e^{a_2 O_{Ik}} - e^{-a_2 O_{Ik}}}{e^{a_2 O_{Ik}} + e^{-a_2 O_{Ik}}}, \quad (10.5)$$

where a_2 is the coefficient of the transfer function.

• **Step 6: Determination of error in prediction**

Let T_{Ok} represents the target output of k -th neuron lying on the output layer. The error in prediction at k -th output neuron corresponding to a particular training scenario can be calculated as follows:

$$E_k = \frac{1}{2}(T_{Ok} - O_{Ok})^2. \quad (10.6)$$

For a training scenario, the error in prediction considering all output neurons can be determined like the following:

$$E = \sum_{k=1}^P \frac{1}{2}(T_{Ok} - O_{Ok})^2, \quad (10.7)$$

where P represents the number of output neurons. The total error in prediction for all output neurons can be determined considering all training scenarios as follows:

$$E_{tot} = \sum_{l=1}^L \sum_{k=1}^P \frac{1}{2}(T_{Ok_l} - O_{Ok_l})^2, \quad (10.8)$$

where L indicates the number of training scenarios.

10.2.2 Training of Network Using Back-Propagation Algorithm

A neural network is to be trained to obtain an optimal network by updating its weights, bias values, architecture etc. Back-Propagation (BP) algorithm [103] has been implemented below for the said purpose using both incremental as well as batch modes of training.

Incremental Mode of Training:

Although the performance of an NN is dependent on its topology, connecting weights, bias values and others, the problem of weight updating only has been considered here, for simplicity. Thus, we can write the error E corresponding to a particular training scenario, as a function of the variables V and W like the following:

$$E = f(V, W). \quad (10.9)$$

Fig. 10.2 shows the plot of a hypothetical error function E . Let us assume that the point X denotes an error in prediction corresponding to a set of initial weights. The aim of training is to reach point Y , at which the error in prediction E reaches its minimum value. In a BP algorithm, the error E is minimized using a steepest descent method, where the changes in weight values can be obtained as follows:

$$\Delta V = -\eta \frac{\partial E}{\partial V}, \quad (10.10)$$

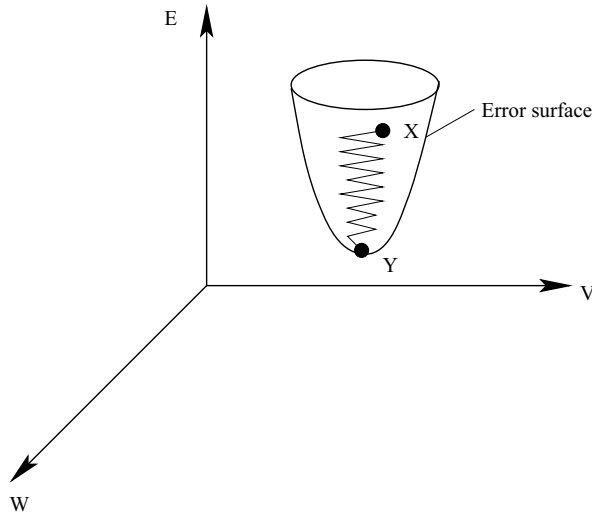


Figure 10.2: A hypothetical error function.

$$\Delta W = -\eta \frac{\partial E}{\partial W}, \quad (10.11)$$

where η represents the learning rate lying between 0.0 and 1.0. It is known as the **delta rule**. It is important to mention that the smaller the value of η , the slower will be the rate of convergence resulting into a smoother network. On the other hand, a higher value of η will make the convergence faster but the resulting network may become unstable.

To update the connecting weight w_{jk} (refer to Fig. 10.1), the following rule is used:

$$w_{jk,updated} = w_{jk,previous} + \Delta w_{jk}, \quad (10.12)$$

where the change in w_{jk} , that is, Δw_{jk} can be determined as follows:

$$\Delta w_{jk} = -\eta \frac{\partial E_k}{\partial w_{jk}}. \quad (10.13)$$

Now, $\frac{\partial E_k}{\partial w_{jk}}$ can be computed using the chain rule of differentiation as given below.

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial w_{jk}}. \quad (10.14)$$

Now, $\frac{\partial E_k}{\partial O_{Ok}} = -(T_{Ok} - O_{Ok})$

$\frac{\partial O_{Ok}}{\partial O_{Ik}} = a_2(1 + O_{Ok})(1 - O_{Ok})$

$\frac{\partial O_{Ik}}{\partial w_{jk}} = H_{Oj}$.

Substituting the values of $\frac{\partial E_k}{\partial O_{Ok}}$, $\frac{\partial O_{Ok}}{\partial O_{Ik}}$ and $\frac{\partial O_{Ik}}{\partial w_{jk}}$ in equation (10.14), we get

$$\frac{\partial E_k}{\partial w_{jk}} = -(T_{Ok} - O_{Ok})a_2(1 + O_{Ok})(1 - O_{Ok})H_{Oj}.$$

Again, substituting the value of $\frac{\partial E_k}{\partial w_{jk}}$ in equation (10.13), Δw_{jk} can be determined as follows:

$$\Delta w_{jk} = \eta a_2 (T_{Ok} - O_{Ok}) (1 + O_{Ok}) (1 - O_{Ok}) H_{Oj}.$$

Therefore, the updated value of w_{jk} can be obtained using equation (10.12).

To update the connecting weight v_{ij} between i -th neuron of input layer and j -th neuron of hidden layer, the following rule is adopted:

$$v_{ij,updated} = v_{ij,previous} + \Delta v_{ij}, \quad (10.15)$$

where the change in v_{ij} , that is, Δv_{ij} can be determined as follows:

$$\Delta v_{ij} = -\eta \left\{ \frac{\partial E}{\partial v_{ij}} \right\}_{av}, \quad (10.16)$$

where $\left\{ \frac{\partial E}{\partial v_{ij}} \right\}_{av} = \frac{1}{P} \sum_{k=1}^P \frac{\partial E_k}{\partial v_{ij}}$.

Now, $\frac{\partial E_k}{\partial v_{ij}}$ can be calculated utilizing the chain rule of differentiation as shown below.

$$\frac{\partial E_k}{\partial v_{ij}} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial H_{Oj}} \frac{\partial H_{Oj}}{\partial H_{Ij}} \frac{\partial H_{Ij}}{\partial v_{ij}}. \quad (10.17)$$

Now, $\frac{\partial E_k}{\partial O_{Ok}} = -(T_{Ok} - O_{Ok})$

$$\frac{\partial O_{Ok}}{\partial O_{Ik}} = a_2 (1 + O_{Ok}) (1 - O_{Ok})$$

$$\frac{\partial O_{Ik}}{\partial H_{Oj}} = w_{jk}$$

$$\frac{\partial H_{Oj}}{\partial H_{Ij}} = a_1 H_{Oj} (1 - H_{Oj})$$

$$\frac{\partial H_{Ij}}{\partial v_{ij}} = I_{Oj} = I_{Ii}.$$

Substituting the values of $\frac{\partial E_k}{\partial O_{Ok}}$, $\frac{\partial O_{Ok}}{\partial O_{Ik}}$, $\frac{\partial O_{Ik}}{\partial H_{Oj}}$, $\frac{\partial H_{Oj}}{\partial H_{Ij}}$ and $\frac{\partial H_{Ij}}{\partial v_{ij}}$ in equation (10.17), we get

$$\frac{\partial E_k}{\partial v_{ij}} = -a_1 a_2 (T_{Ok} - O_{Ok}) (1 + O_{Ok}) (1 - O_{Ok}) (1 - H_{Oj}) w_{jk} H_{Oj} I_{Ii}.$$

Thus, $\left\{ \frac{\partial E}{\partial v_{ij}} \right\}_{av}$ can be determined and substituting its value in equation (10.16), the change in v_{ij} , that is, Δv_{ij} can be calculated. Therefore, the updated value of v_{ij} can be obtained using equation (10.15).

Batch Mode of Training:

A batch mode of training is generally implemented through the minimization of Mean Squared Deviation (*MSD*) in prediction, as given below (corresponding to k -th output neuron).

$$E' = \frac{1}{2} \frac{1}{L} \sum_{l=1}^L (T_{Ok l} - O_{Ok l})^2, \quad (10.18)$$

where $T_{Ok l}$ indicates the target output of k -th output neuron corresponding to l -th training scenario, $O_{Ok l}$ represents the model predicted value of k -th output neuron corresponding to l -th training scenario.

The change in w_{jk} , that is, Δw_{jk} is determined as follows:

$$\Delta w_{jk} = -\eta \frac{\partial E'}{\partial w_{jk}}, \quad (10.19)$$

where $\frac{\partial E'}{\partial w_{jk}} = \frac{\partial E'}{\partial E_l} \frac{\partial E_l}{\partial E_k} \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial w_{jk}}$ and it can be determined following the above procedure.

Similarly, Δv_{ij} can be calculated like the following:

$$\Delta v_{ij} = -\eta \left\{ \frac{\partial E'}{\partial v_{ij}} \right\}_{av}, \quad (10.20)$$

where $\left\{ \frac{\partial E'}{\partial v_{ij}} \right\}_{av} = \frac{1}{P} \sum_{k=1}^P \frac{\partial E'_k}{\partial v_{ij}}$.

Now, $\frac{\partial E'_k}{\partial v_{ij}}$ can be calculated as follows:

$$\frac{\partial E'_k}{\partial v_{ij}} = \frac{\partial E'_k}{\partial E_{kl}} \frac{\partial E_{kl}}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial H_{Oj}} \frac{\partial H_{Oj}}{\partial H_{Ij}} \frac{\partial H_{Ij}}{\partial v_{ij}}.$$

The value of $\frac{\partial E'_k}{\partial v_{ij}}$ can be obtained as discussed above. Therefore, the updated values of w_{jk} and v_{ij} can be obtained.

In this mode of training, if the number of training scenarios becomes less than the number of network parameters $P_{network}$ (that is, synaptic weights, biases etc.), the trained network will be mathematically undetermined. To ensure a good generalization of the network, the number of training scenarios is selected in the order of $\frac{P_{network}}{\epsilon}$, where ϵ indicates the percent of error permitted. Therefore, the number of training scenarios is to be either equal to or more than the number of network parameters to ensure its proper functioning. However, if a very large set of training data is used, the network may memorize those training data during its testing and as a result of which, it may lose its generalization capability. It is known as **over-training** of the network.

Momentum Constant (α')

To determine the change of a parameter of the NN (say connecting weight) at a particular iteration (say t -th) of the algorithm, sometimes the change of that parameter in the previous ($t - 1$)-th iteration is also considered with the help of a momentum constant α' as given below [103].

$$\Delta w(t) = -\eta \frac{\partial E}{\partial w}(t) + \alpha' \Delta w(t - 1). \quad (10.21)$$

It is known as the **generalized delta rule**. It is to be noted that α' varies in the range of 0.0 to 1.0. It is also important to mention that a higher value of η may result into an unstable/oscillatory network. The momentum constant α' is used to ensure a stable network even at a higher value of learning rate η .

Notes:

1. A BP algorithm works based on the principle of a steepest descent algorithm. Being a gradient-based method, the chance of its solutions for being trapped into the local minima is more.
2. For a successful implementation of the BP algorithm, the selected transfer functions are to be differentiable in nature.
3. To model a process, the inputs are to be selected in such a way, that they are independent to each other. However, inter-dependency among some of the outputs cannot be neglected for some of the processes. A Back-Propagation Neural Network (BPNN) can model the non-linearity of a process. However, it may not be able to capture the dynamics of a highly dynamic process fully.
4. The inputs of a process are fed to the network for modeling, in terms of their normalized values. The connecting weights are initially generated at random, in the range of either (0.0, 1.0) or (-1.0, 1.0).
5. During the training of an NN, if the updated connecting weight becomes more than 1.0, the network will become unstable.
6. It is said that the network has converged during its training, if the absolute value of rate of change of error in prediction becomes less than or equal to a pre-specified value.
7. A neural network can have either a fully-connected or a partially-connected structure as shown in Fig. 10.3.

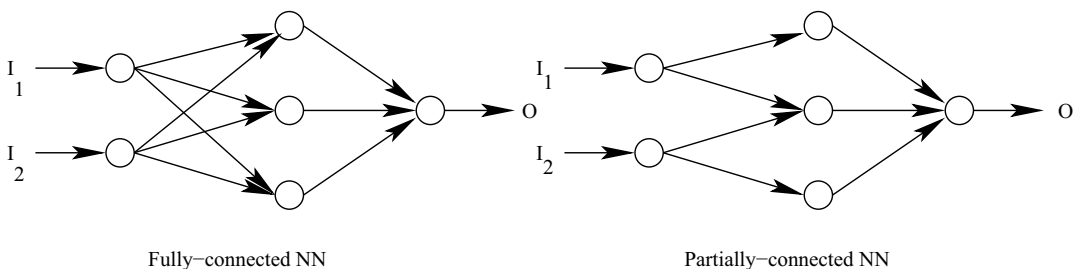


Figure 10.3: A fully-connected and a partially-connected NN.

8. Levenberg-Marquardt algorithm is also used sometimes in place of the BP algorithm. It is a hybrid algorithm, which combined Gauss-Newton algorithm with the steepest descent algorithm. Interested readers may refer to [104] for a detailed description of this algorithm.

10.2.3 Steps to be Followed to Design a Suitable NN

The following steps are to be followed to design a suitable neural network:

- **Step 1:** Identify input and output variables of the process to be modeled. Input variables are to be independent in nature.
- **Step 2:** Normalize the variables in the range of either 0.0 to 1.0 or -1.0 to 1.0 .
- **Step 3:** Initialize the number of hidden layers and that of neurons in each layer. Select appropriate transfer functions for the neurons of each layer.
- **Step 4:** Generate the connecting weights in normalized scale, bias values and coefficients of the transfer functions at random.
- **Step 5:** Update the above parameters iteratively using a BP algorithm implemented through either an incremental or a batch mode of training.
- **Step 6:** Continue the iterations until the termination criterion is reached.
- **Step 7:** Testing of the optimized neural network.

10.2.4 Advantages and Disadvantages

An NN has the following advantages:

- It can handle a control problem involving many variables.
- It may not require so much problem information, as that is needed for developing an FLC.

However, it has the following disadvantages:

- The solutions of a BPNN may get stuck at the local minima.
- Training of an NN is more involved computationally compared to tuning of an FLC.
- It works like a black box.

10.2.5 A Numerical Example

Fig. 10.4 shows the schematic view of an NN consisting of three layers, such as input, hidden and output layers. The neurons lying on the input, hidden and output layers have the transfer functions represented by $y = x$, $y = \frac{1}{1+e^{-x}}$, $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, respectively. There are two inputs, namely I_1 and I_2 and one output, that is, O . The connecting weights between the input and hidden layers are represented by $[V]$ and those between the hidden and output layers are denoted by $[W]$. The initial values of the weights are assumed to be as follows:

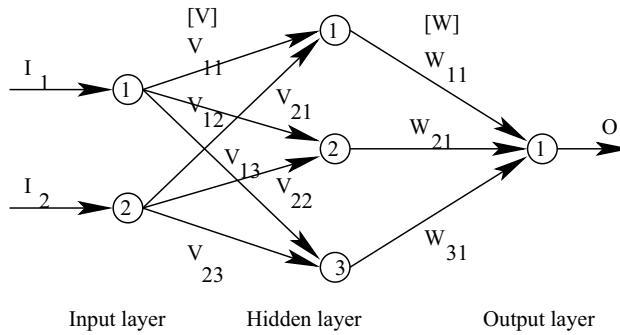


Figure 10.4: Schematic view of a neural network.

$$\begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.4 & 0.3 \\ 0.1 & 0.6 & 0.5 \end{bmatrix};$$

$$\begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.1 \end{bmatrix}.$$

Table 10.1: Training cases.

Sl. No.	I_1	I_2	T_O
1	0.5	-0.4	0.15
2	-	-	-
\vdots	\vdots	\vdots	\vdots

Using an incremental mode of training for the case shown in Table 10.1, calculate the changes in V (that is, ΔV) and W (that is, ΔW) values during back-propagation of error. The learning rate η is assumed to be equal to 0.2. Show only one iteration.

Solution:

Forward Calculations:

The neurons of input layer have linear transfer function ($y = x$). Therefore, the output of each input layer neuron is made equal to its corresponding input value.

$$I_{O1} = I_{I1} = 0.5$$

$$I_{O2} = I_{I2} = -0.4$$

The inputs of different neurons of the hidden layer are calculated as follows:

$$\begin{aligned}
H_{I1} &= I_{O1}v_{11} + I_{O2}v_{21} = 0.06 \\
H_{I2} &= I_{O1}v_{12} + I_{O2}v_{22} = -0.04 \\
H_{I3} &= I_{O1}v_{13} + I_{O2}v_{23} = -0.05
\end{aligned}$$

The neurons of the hidden layer are assumed to have log-sigmoid transfer function ($y = \frac{1}{1+e^{-x}}$). The outputs of different hidden neurons are determined like the following:

$$\begin{aligned}
H_{O1} &= \frac{1}{1+e^{-H_{I1}}} = 0.514995 \\
H_{O2} &= \frac{1}{1+e^{-H_{I2}}} = 0.490001 \\
H_{O3} &= \frac{1}{1+e^{-H_{I3}}} = 0.487503
\end{aligned}$$

Now, the input of the output neuron can be calculated as follows:

$$O_{I1} = H_{O1}w_{11} + H_{O2}w_{21} + H_{O3}w_{31} = 0.198249$$

As the output neuron has a tan-sigmoid transfer function, its output can be determined like the following:

$$O_{O1} = \frac{e^{O_{I1}} - e^{-O_{I1}}}{e^{O_{I1}} + e^{-O_{I1}}} = 0.195692$$

The squared error in prediction is found to be as follows:

$$E = \frac{1}{2}(T_O - O_{O1})^2 = 0.001044$$

Back-propagation Algorithm:

The change in w_{11} can be determined using the procedure given below.

$$\Delta w_{11} = -\eta \frac{\partial E}{\partial w_{11}},$$

where $\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial O_{O1}} \frac{\partial O_{O1}}{\partial O_{I1}} \frac{\partial O_{I1}}{\partial w_{11}}$.

Now, $\frac{\partial E}{\partial O_{O1}} = -(T_O - O_{O1})$

$$\frac{\partial O_{O1}}{\partial O_{I1}} = \frac{4}{(e^{O_{I1}} + e^{-O_{I1}})^2}$$

$$\frac{\partial O_{I1}}{\partial w_{11}} = H_{O1}.$$

Substituting the values of T_O , O_{O1} , O_{I1} and H_{O1} in the above expression of $\frac{\partial E}{\partial w_{11}}$, we get

$$\frac{\partial E}{\partial w_{11}} = 0.022630$$

Now, substituting the values of $\frac{\partial E}{\partial w_{11}}$ and η in the expression of Δw_{11} , we get

$$\Delta w_{11} = -0.004526$$

Similarly, we can determine Δw_{21} and Δw_{31} and these are found to be as follows:

$$\Delta w_{21} = -0.004306$$

$$\Delta w_{31} = -0.004284$$

The necessary change in v_{11} can be obtained as follows:

$$\Delta v_{11} = -\eta \frac{\partial E}{\partial v_{11}}$$

where $\frac{\partial E}{\partial v_{11}} = \frac{\partial E}{\partial O_{O1}} \frac{\partial O_{O1}}{\partial O_{I1}} \frac{\partial O_{I1}}{\partial H_{O1}} \frac{\partial H_{O1}}{\partial H_{I1}} \frac{\partial H_{I1}}{\partial v_{11}}$.

Now, $\frac{\partial E}{\partial O_{O1}} = -(T_O - O_{O1})$

$$\frac{\partial O_{O1}}{\partial O_{I1}} = \frac{4}{(e^{O_{I1}} + e^{-O_{I1}})^2}$$

$$\frac{\partial O_{I1}}{\partial H_{O1}} = w_{11}$$

$$\frac{\partial H_{O1}}{\partial H_{I1}} = \frac{e^{-H_{I1}}}{(1 + e^{-H_{I1}})^2}$$

$$\frac{\partial H_{I1}}{\partial v_{11}} = I_{O1}.$$

Substituting the values of T_O , O_{O1} , O_{I1} , w_{11} , H_{I1} and I_{O1} in the above expression of $\frac{\partial E}{\partial v_{11}}$, we obtain

$$\frac{\partial E}{\partial v_{11}} = 0.000549.$$

Now, substituting the values of η and $\frac{\partial E}{\partial v_{11}}$, we get $\Delta v_{11} = -0.000110$.

Similarly, the values of Δv_{21} , Δv_{12} , Δv_{22} , Δv_{13} and Δv_{23} are determined and found to be as follows:

$$\begin{aligned}\Delta v_{21} &= 0.000088 \\ \Delta v_{12} &= -0.000220 \\ \Delta v_{22} &= 0.000176 \\ \Delta v_{13} &= -0.000110 \\ \Delta v_{23} &= 0.000088\end{aligned}$$

Therefore, the updated values of the weights are coming out to be as follows:

$$\begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix} = \begin{bmatrix} 0.199890 & 0.399780 & 0.299890 \\ 0.100088 & 0.600176 & 0.500088 \end{bmatrix};$$

$$\begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix} = \begin{bmatrix} 0.095474 \\ 0.195694 \\ 0.095716 \end{bmatrix}.$$

10.3 Radial Basis Function Network (RBFN)

A Radial Basis Function (RBF) is a special type of function, whose response decreases or increases monotonically with its distance from a central point. Various types of RBFs are in use, such as

- Thin plate spline function: $f(x) = x^2 \log(x)$,
- Gaussian function: $f(x) = \exp(-\frac{\|x-\mu\|^2}{2\sigma^2})$, where μ and σ indicate the mean and standard deviation of the distribution, respectively,
- Multi-quadratic function: $f(x) = \sqrt{x^2 + \sigma^2}$,

- Inverse multi-quadratic function: $f(x) = \frac{1}{\sqrt{x^2 + \sigma^2}}$.

Fig. 10.5 shows the plots of the above RBFs.

Radial Basis Function Network (RBFN) was proposed by Broomhead and Lowe [105] in 1988. It is a special type of Artificial Neural Network (ANN), that can fit a surface in multi-dimensional space after ensuring the best match to the training data. Besides function approximation, it can tackle the problems related to time-series forecasting, classification, clustering, recognition of handwritten characters, face recognition, voice identification and others. This network consists of a layer of some input data nodes (whose number is kept equal to the number of independent input variables of the system to be modeled), generally one hidden layer of a few neurons with radial basis transfer functions and an output layer composed of some neurons (whose number is made equal to the number of outputs) with linear transfer function usually. As the layer of input nodes used here is different from the conventional input layer of an ANN, the RBFN is also known as a two-layer network (that is, it consists of hidden and output layers). The performance of the RBFN is dependent on the number of hidden neurons, their centers and radii. For example, the performance of a network with Gaussian transfer functions depends on their centers and radii, which are nothing but their means and standard deviations, respectively. The network with an insufficient number of hidden neurons may not be able to offer good approximation. On the other hand, a large number of hidden neurons may yield good approximation but at the cost of predictive power. It is known as an **over-fitting problem** of the network. The RBFN with either narrow or wide radii may produce some bad results. Thus, a proper tuning of the network is necessary to carry out.

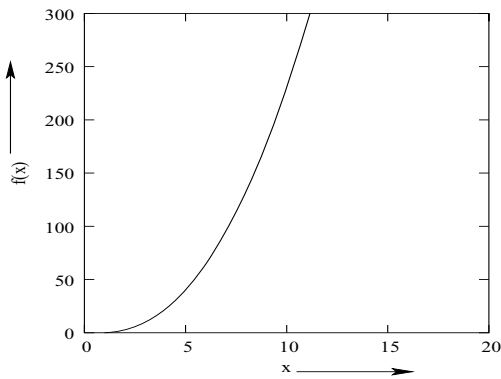
Training has been provided to the RBFN following various methods as discussed below.

- **Method 1:** The number of hidden neurons is pre-defined and suitable values of their centers and radii are determined through a proper training [106].
- **Method 2:** The algorithm starts with an empty set of hidden neurons, which grows in number until a given condition is reached [107, 108]. Thus, the number of hidden neurons and their centers and radii are searched iteratively.
- **Method 3:** Here, the algorithm begins with a large number of hidden neurons. Some of the hidden neurons and their connecting weights are removed iteratively [109] and ultimately, the optimal network is obtained.

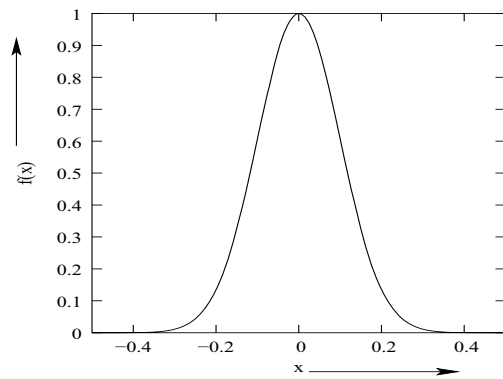
Fig. 10.6 shows an RBFN, whose aim is to determine input-output relationships of a process having M independent input variables and P outputs. Let us suppose that the network consists of a layer having M input nodes, one hidden layer containing N neurons and an output layer of P neurons. Let us also assume that a set of L training scenarios will be used to train the network. For a particular training scenario (say l -th), calculations related to the forward and backward propagations are given below.

10.3.1 Forward Calculations

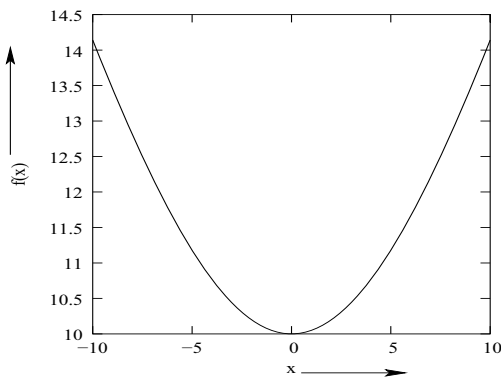
The following steps are considered in the forward calculations:



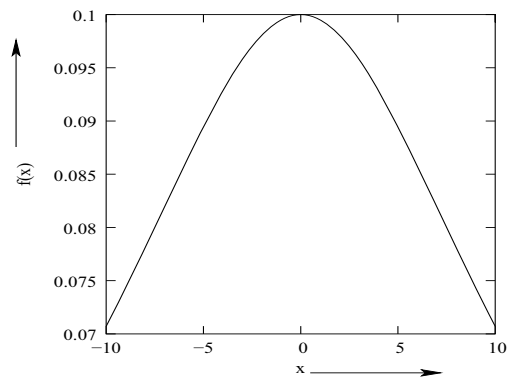
(a) Plate spline function



(b) Gaussian function



(c) Multi-quadratic function



(d) Inverse multi-quadratic function

Figure 10.5: Plots of various radial basis functions.

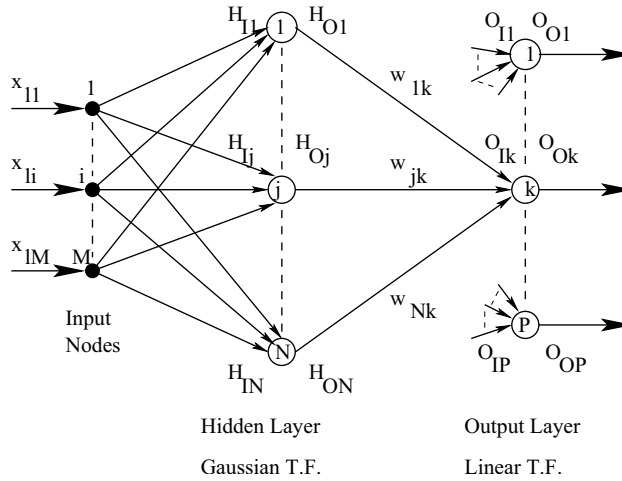


Figure 10.6: Radial Basis Function Network.

- **Step 1: Determination of the outputs of input nodes**

Let us represent L training scenarios, which constitute the input vector, as given below.

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \\ \vdots \\ x_L \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1i} & \dots & x_{1M} \\ x_{21} & x_{22} & \dots & x_{2i} & \dots & x_{2M} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{l1} & x_{l2} & \dots & x_{li} & \dots & x_{lM} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{L1} & x_{L2} & \dots & x_{Li} & \dots & x_{LM} \end{bmatrix}$$

Let us assume that l -th training scenario (that is, $x_{l1}, x_{l2}, \dots, x_{li}, \dots, x_{lM}$ expressed in the normalized form) is passed through the network. The outputs of different nodes are nothing but the inputs of corresponding nodes.

- **Step 2: Determination of the outputs of hidden layer**

The hidden layer consists of N neurons. Each of them has a Gaussian transfer function represented by a separate set of mean μ and standard deviation σ values. Thus, the values of μ and σ associated with the transfer functions of 1-st, 2-nd, \dots , N -th neurons are represented by $(\mu_1, \sigma_1), (\mu_2, \sigma_2), \dots, (\mu_N, \sigma_N)$, respectively. The output of j -th hidden neuron can be expressed like the following:

$$H_{Oj} = \exp\left[-\frac{\|x_l - \mu_j\|^2}{2\sigma_j^2}\right] \quad (10.22)$$

- **Step 3: Determination of the inputs of output layer**

The input of k -th neuron lying on output layer can be represented as follows:

$$O_{Ik} = \sum_{j=1}^N w_{jk} H_{Oj}, \quad (10.23)$$

where w_{jk} indicates the connecting weight between j -th hidden neuron and k -th output neuron.

- **Step 4: Determination of the outputs of output layer**

The output of k -th neuron lying on output layer is determined as follows:

$$O_{Ok} = O_{Ik}, \quad (10.24)$$

as the output neurons are assumed to have linear transfer function.

Let us suppose that the target output of k -th output neuron is denoted by T_{Ok} . Therefore, the error in prediction can be calculated like the following:

$$E_k = \frac{1}{2} (T_{Ok} - O_{Ok})^2. \quad (10.25)$$

10.3.2 Tuning of RBFN Using Back-Propagation Algorithm

Back-propagation algorithm can be implemented through either an incremental or a batch mode of training as discussed below.

Incremental Mode of Training:

Let us consider an incremental mode of training, that is, the connecting weights, mean and standard deviation values of the Gaussian distributions will be updated to minimize the error in prediction, corresponding to l -th training scenario.

- **Step 1: Weight updating**

The weights are updated following the rule given below.

$$w_{updated} = w_{previous} + \Delta w, \quad (10.26)$$

where Δw indicates the change in w value. The change in w value at t -th iteration is given by the following expression:

$$\Delta w_{jk}(t) = -\eta \frac{\partial E_k}{\partial w_{jk}}(t) + \alpha' \Delta w_{jk}(t-1), \quad (10.27)$$

where η and α' represent the learning rate and momentum constant, respectively.

Now, $\frac{\partial E_k}{\partial w_{jk}}$ can be determined corresponding to k -th output neuron, using the chain rule of differentiation as given below.

$$\frac{\partial E_k}{\partial w_{jk}} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial w_{jk}}, \quad (10.28)$$

where $\frac{\partial E_k}{\partial O_{Ok}} = -(T_{Ok} - O_{Ok})$,

$$\frac{\partial O_{Ok}}{\partial O_{Ik}} = 1,$$

$$\frac{\partial O_{Ik}}{\partial w_{jk}} = H_{Oj}.$$

Therefore, equation (10.28) can be re-written as

$$\frac{\partial E_k}{\partial w_{jk}} = -(T_{Ok} - O_{Ok})H_{Oj}. \quad (10.29)$$

Now, equation (10.26) can be expressed like the following:

$$w_{jk,updated} = w_{jk,previous} + \eta(T_{Ok} - O_{Ok})H_{Oj} + \alpha' \Delta w_{jk,previous}. \quad (10.30)$$

The similar procedure can be followed to determine updated values of other weights.

- **Step 2: Mean updating**

The mean of Gaussian distribution corresponding to j -th hidden neuron can be updated as follows:

$$\mu_{j,updated} = \mu_{j,previous} + \Delta\mu_j, \quad (10.31)$$

where the change in μ at t -th iteration can be determined like the following:

$$\Delta\mu_j(t) = -\eta\left\{\frac{\partial E}{\partial \mu_j}(t)\right\}_{av} + \alpha' \Delta\mu_j(t-1), \quad (10.32)$$

where $\left\{\frac{\partial E}{\partial \mu_j}\right\}_{av} = \frac{1}{P} \sum_{k=1}^P \frac{\partial E_k}{\partial \mu_j}$,

$$\frac{\partial E_k}{\partial \mu_j} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial H_{Oj}} \frac{\partial H_{Oj}}{\partial \mu_j}. \quad (10.33)$$

We have already seen that $\frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} = -(T_{Ok} - O_{Ok})$.

Now, $\frac{\partial O_{Ik}}{\partial H_{Oj}} = w_{jk}$,

$$\frac{\partial H_{Oj}}{\partial \mu_j} = H_{Oj} \left\{ \frac{(x_{l1} + x_{l2} + \dots + x_{li} + \dots + x_{lM}) - M\mu_j}{\sigma_j^2} \right\}.$$

Therefore, we get

$$\frac{\partial E_k}{\partial \mu_j} = -(T_{Ok} - O_{Ok})w_{jk}H_{Oj} \left\{ \frac{(x_{l1} + x_{l2} + \dots + x_{li} + \dots + x_{lM}) - M\mu_j}{\sigma_j^2} \right\}. \quad (10.34)$$

- **Step 3: Standard deviation updating**

The standard deviation corresponding to j -th hidden neuron will be updated as follows:

$$\sigma_{j,updated} = \sigma_{j,previous} + \Delta\sigma_j, \quad (10.35)$$

where the change in σ at t -th iteration is written like the following:

$$\Delta\sigma_j(t) = -\eta\left\{\frac{\partial E}{\partial \sigma_j}(t)\right\}_{av} + \alpha' \Delta\sigma_j(t-1), \quad (10.36)$$

where $\{\frac{\partial E}{\partial \sigma_j}\}_{av} = \frac{1}{P} \sum_{k=1}^P \frac{\partial E_k}{\partial \sigma_j}$,

$$\frac{\partial E_k}{\partial \sigma_j} = \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial H_{Oj}} \frac{\partial H_{Oj}}{\partial \sigma_j}. \quad (10.37)$$

We have already seen that $\frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} = -(T_{Ok} - O_{Ok})$.

Now, $\frac{\partial O_{Ik}}{\partial H_{Oj}} = w_{jk}$,

$$\frac{\partial H_{Oj}}{\partial \sigma_j} = H_{Oj} \left\{ \frac{(x_{l1} - \mu_j)^2 + (x_{l2} - \mu_j)^2 + \dots + (x_{li} - \mu_j)^2 + \dots + (x_{lM} - \mu_j)^2}{\sigma_j^3} \right\}.$$

Therefore, we get

$$\frac{\partial E_k}{\partial \sigma_j} = -(T_{Ok} - O_{Ok}) w_{jk} H_{Oj} \left\{ \frac{\sum_{i=1}^M (x_{li} - \mu_j)^2}{\sigma_j^3} \right\}. \quad (10.38)$$

Batch Mode of Training:

In this scheme of training, the whole training set is passed through the network and then the connecting weights, mean and standard deviation values of the Gaussian distributions are updated based on the value of Mean Squared Deviation (MSD) in prediction. It can be determined corresponding to k -th output neuron as follows:

$$E = \frac{1}{2} \frac{1}{L} \sum_{l=1}^L (T_{Ok} - O_{Ok})^2, \quad (10.39)$$

where L represents the number of training scenarios considered. The change in w_{jk} value at t -th iteration is determined like the following:

$$\Delta w_{jk}(t) = -\eta \frac{\partial E}{\partial w_{jk}}(t) + \alpha' \Delta w_{jk}(t-1), \quad (10.40)$$

where $\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial E_l} \frac{\partial E_l}{\partial E_k} \frac{\partial E_k}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial w_{jk}}$.

The change in mean value at t -th iteration can be determined as follows:

$$\Delta \mu_j(t) = -\eta \left\{ \frac{\partial E}{\partial \mu_j}(t) \right\}_{av} + \alpha' \Delta \mu_j(t-1), \quad (10.41)$$

where $\{\frac{\partial E}{\partial \mu_j}\}_{av} = \frac{1}{P} \sum_{k=1}^P \frac{\partial E_k}{\partial \mu_j}$,

$$\frac{\partial E_k}{\partial \mu_j} = \frac{\partial E_k}{\partial E_{kl}} \frac{\partial E_{kl}}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial H_{Oj}} \frac{\partial H_{Oj}}{\partial \mu_j}.$$

Similarly, the change in standard deviation value can also be obtained as follows:

$$\Delta \sigma_j(t) = -\eta \left\{ \frac{\partial E}{\partial \sigma_j}(t) \right\}_{av} + \alpha' \Delta \sigma_j(t-1), \quad (10.42)$$

where $\{\frac{\partial E}{\partial \sigma_j}\}_{av} = \frac{1}{P} \sum_{k=1}^P \frac{\partial E_k}{\partial \sigma_j}$,

$$\frac{\partial E_k}{\partial \sigma_j} = \frac{\partial E_k}{\partial E_{kl}} \frac{\partial E_{kl}}{\partial O_{Ok}} \frac{\partial O_{Ok}}{\partial O_{Ik}} \frac{\partial O_{Ik}}{\partial H_{Oj}} \frac{\partial H_{Oj}}{\partial \sigma_j}.$$

The principle of back-propagation has been widely used to tune the RBFN. However, the chance of its solutions for being trapped into the local minima is more, as it works based on the principle of a steepest descent method. To overcome this difficulty, a Genetic Algorithm (GA) [11] may be utilized (in place of the steepest descent method), along with forward propagation of the network for the said purpose. A GA was utilized by Billings and Zheng [110] to optimize the structure of RBFN and configure the network in terms of the number of hidden neurons and center locations. The concept of a variable string-length GA was implemented in their work. Sheta and de Jong [111] used a GA-tuned RBFN to solve time-series forecasting problem. The GA-string was designed to carry information of the centers, radii of transfer functions and connecting weights of the RBFN.

A radial basis function network has almost the same learning capability as that of the conventional ANN. However, it has a simpler structure and is computationally less expensive compared to the conventional ANN.

10.4 Self-Organizing Map (SOM)

Self-Organizing Map (SOM) (also known as Kohonen Network) is a data visualization technique proposed by Kohonen, which reduces the dimension of data using a self-organizing neural network [50]. It is a sequential clustering algorithm, which produces the self-organizing feature maps similar to those present in our brain. It is rather a modified version of neural network working based on unsupervised and competitive learning. Figure 10.7 shows the schematic view of a self-organizing map. The most promising feature

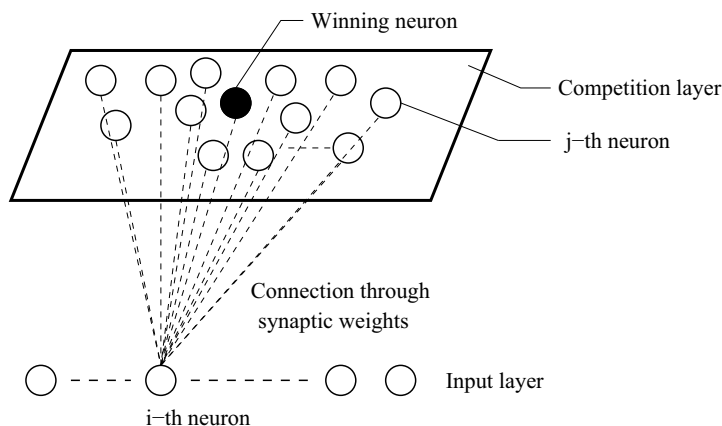


Figure 10.7: A schematic view representing the self-organizing map.

of SOM lies in the fact that it can preserve topology in a non-linear mapping process. The SOM can be viewed as a non-linear generalization of Principal Component Analysis (PCA) [51]. There are two layers in the SOM, namely input and competition layers. The input layer contains multivariate data, which are to be mapped into a lower dimensional space. The number of neurons of the competition layer is generally kept equal to that of

the data points present in the input layer. Each unit of multivariate data present in the input layer is connected to all neurons lying in the competition layer through some synaptic weights. The neurons of the competition layer undergo three basic operations, such as **competition**, **cooperation** and **updating**, in stages. In competition stage, the neurons of the competition layer compete among themselves to be identified as a winner, which has the best match with the input unit. In cooperation stage, a neighborhood surrounding the winning neuron is identified, which contains some neurons similar to the winner. The winning neuron along with its neighborhood is updated in third stage. All these stages are explained below in detail.

10.4.1 Competition

Let us suppose that there are N points (neurons) in the input layer and each point has m dimension. Thus, a particular input neuron X_i can be expressed as follows:

$$X_i = [x_{i1}, x_{i2}, \dots, x_{im}]^T \quad \text{where } i = 1, 2, \dots, N.$$

Let us also assume that the synaptic weight vector between input neuron i and neuron j lying in the competition layer is denoted by

$$W_j^i = [w_{j1}^i, w_{j2}^i, \dots, w_{jm}^i]^T \quad \text{where } j = 1, 2, \dots, N.$$

It is based on the consideration that the number of neurons lying in the competition layer is made equal to that of the input layer. To find the best match for the input data vector X_i , we need to identify the neuron lying in the competition layer, which has the largest inner product (that is, $[W_j^i]^T X_i$) with it. Now, it is to be noted that maximizing the inner product is mathematically equivalent to minimizing the Euclidean distance between the vectors, X_i and W_j^i . Let us denote the neuron lying in the competition layer that has the best match with the input vector X_i by n . Thus, the Euclidean distance between n and X_i can be expressed as follows:

$$n(X_i) = \text{Minimum of } \sqrt{(X_i - W_j^i)^2} \quad \text{where } j = 1, 2, \dots, N.$$

10.4.2 Cooperation

The winning neuron determines its topological neighborhood (which is expressed with the help of a neighborhood function) of excited neurons and they will cooperate with each other. They will try to update their synaptic weights through this cooperation. The neighborhood function is assumed to follow a Gaussian distribution for the said purpose, as given below.

$$h_{j,n(X_i)}(t) = \exp\left(-\frac{d_{j,n(X_i)}^2}{2\sigma_t^2}\right) \quad \text{where } t = 0, 1, 2, \dots,$$

where $d_{j,n(X_i)}$ is the lateral distance between the winning neuron n and excited neuron j ; σ_t is the value of standard deviation at t -th iteration, which can be determined as $\sigma_t =$

$\sigma_0 \exp(-\frac{t}{\tau})$, where σ_0 is the initial value of standard deviation and τ indicates the predefined number of maximum iterations. Thus, it is assumed that topological neighborhood shrinks with the number of iterations. It is important to note that while updating the weight of winning neuron itself, the neighborhood function $h_{j,n(X_i)}(t)$ is calculated considering the average of its distances from all excited neurons. It is done just to have a varying $h_{j,n(X_i)}(t)$ at different iterations for updating the winning neuron. It ensures a contribution of the excited neurons towards updating of the weights (iteration-wise) of winning neuron.

10.4.3 Updating

The synaptic weights of the winning neuron and excited neurons lying in its neighborhood are updated using the rule given below.

$$W_j^i(t+1) = W_j^i(t) + \eta(t)h_{j,n(X_i)}(t)[X_i - W_j^i(t)],$$

where $\eta(t)$ is the learning rate lying between 0.0 and 1.0.

10.4.4 Final Mapping

The above method is followed to determine a winning neuron corresponding to each input data. Our aim is to map a set of higher dimensional (input) data into a lower dimension (output), after keeping topological information of the data intact. To ensure this, a scheme may be adopted as shown in Fig. 10.8. The Euclidean distances of the winning neurons from

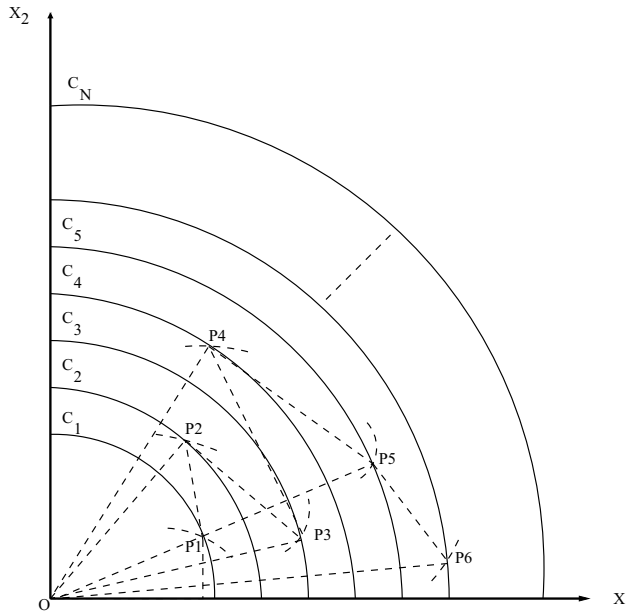


Figure 10.8: Final mapping of the SOM.

the origin of higher dimensional space are calculated and this information is used to draw

a number of circular arcs (equal to the number of winning neurons) in a lower dimensional space (say $2 - D$), after keeping their centers at the origin $(0.0, 0.0)$. The winning neurons are then located in $2 - D$, such that the distance between any two neurons is kept the same as that in the higher dimensional space. Thus, topological information of the multivariate data can be kept unaltered in the lower dimensional space.

10.4.5 Simulation Results

Let us take an example, in which 200 points (selected at random) lying on the surface of Schaffer's first test function are to be mapped to $2 - D$ for visualization. The said function

$$\text{can be mathematically expressed as } y = 0.5 + \frac{\sin^2 \sqrt{\sum_{i=1}^4 x_i^2} - 0.5}{1.0 + 0.001(\sum_{i=1}^4 x_i^2)^2}.$$

The mapped points obtained using the SOM are shown in Figure 10.9. The SOM is able to provide the well-distributed points, which are easy to visualize.

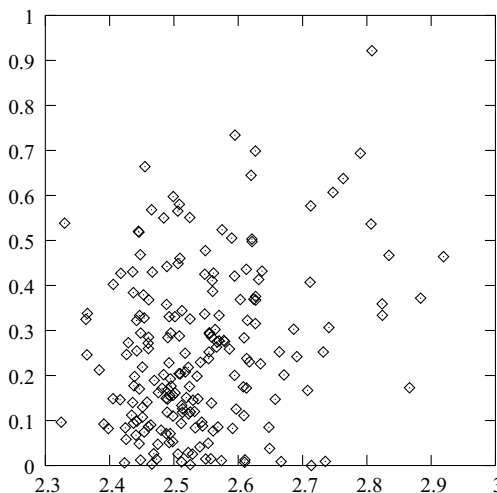


Figure 10.9: Mapping obtained by the SOM.

10.5 Counter-Propagation Neural Network (CPNN)

Counter-Propagation Neural Network (CPNN) was introduced by Robert Hecht-Nielsen [112]. It consists of three layers, namely input layer, unsupervised Kohonen layer and teachable output layer. The first and second layers, that is, input layer and Kohonen layer form an in-star model, and the second and third layers, that is, Kohonen layer and output layer constitute an out-star model. In this network, the second and third layers perform Kohonen and Grossberg learnings, respectively. Thus, it combines unsupervised Kohonen learning and supervised Grossberg learning. The CPNN is much faster than the BPNN, and there is no chance of its weights to get trapped into the local minima. However, it may be inferior

to the BPNN in mapping applications. A CPNN may be either a full CPNN or a forward only CPNN, whose working principles are discussed below.

10.5.1 Full CPNN

It consists of two input layers and two output layers with a common hidden layer to them, as shown in Fig. 10.10. Let us assume that there are m number of independent variables x (that

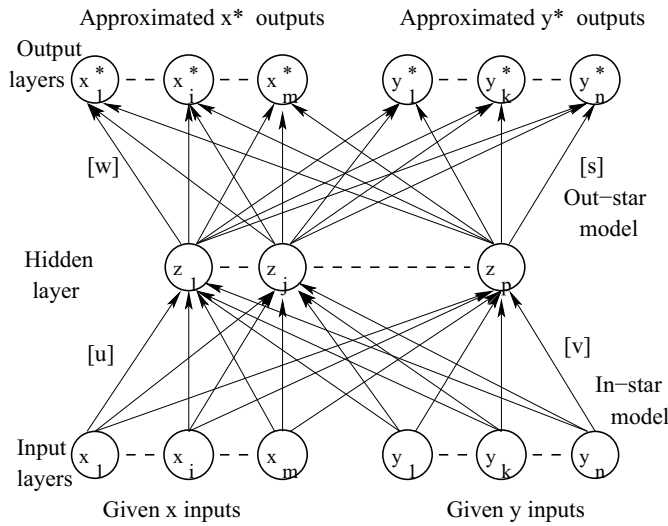


Figure 10.10: A full CPNN.

is, $x_1, \dots, x_i, \dots, x_m$) and n number of dependent variables y (that is, $y_1, \dots, y_k, \dots, y_n$) in a data set to be modeled, which are shown in two input layers. It is to be noted that these values are expressed in a normalized scale of $(0, 1)$. Let us consider p neurons in the hidden layer denoted by $z_1, \dots, z_j, \dots, z_p$. Two output layers of the network contain information of the approximated x^* (that is, $x_1^*, \dots, x_i^*, \dots, x_m^*$) and y^* (that is, $y_1^*, \dots, y_k^*, \dots, y_n^*$) outputs, as shown in the above figure. In the in-star model, the connecting weight between x_i and z_j is denoted by u_{ij} and that between y_k and z_j is represented by v_{kj} . Similarly, in the out-star model, the connecting weight between z_j and x_i^* , and that between z_j and y_k^* are indicated by w_{ji} and s_{jk} , respectively.

Training of In-Star Model

Fig. 10.11 shows the in-star model only of the complete network displayed in Fig. 10.10. Initially, the connecting weights $[u]$ and $[v]$ are generated at random in the range of $(0, 1)$. Let the learning rate between x inputs layer and hidden layer is represented by α and that between the y inputs layer and hidden layer is denoted by β . The principle of Kohonen's self-organizing map (that is, unsupervised learning) is used during the training of in-star model. To decide the winner neuron, the Euclidean distance values are calculated as follows:

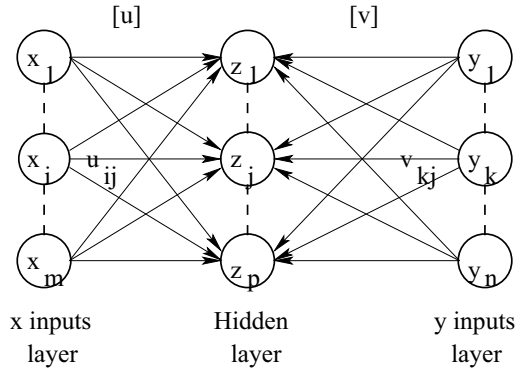


Figure 10.11: In-star model of CPNN.

$$d_j = \sqrt{\sum_{i=1}^m (x_i - u_{ij})^2 + \sum_{k=1}^n (y_k - v_{kj})^2}$$

Here, $j = 1, \dots, j, \dots, p$, and the neuron with the minimum value of d_j is declared as the winner. The connecting weights of the winner are then updated as follows:

$$\begin{aligned} u_{ij}(\text{updated}) &= u_{ij}(\text{previous}) + \alpha(x_i - u_{ij}(\text{previous})), \\ i &= 1, 2, \dots, m; \\ v_{kj}(\text{updated}) &= v_{kj}(\text{previous}) + \beta(y_k - v_{kj}(\text{previous})), \\ k &= 1, 2, \dots, n. \end{aligned}$$

It completes one iteration of in-star training. It is to be noted that the values of learning rate, that is, α and β are then reduced for the next iteration.

Training of Out-Star Model

Fig. 10.12 displays the out-star model only. Let z_j be the winner. The connecting weights:

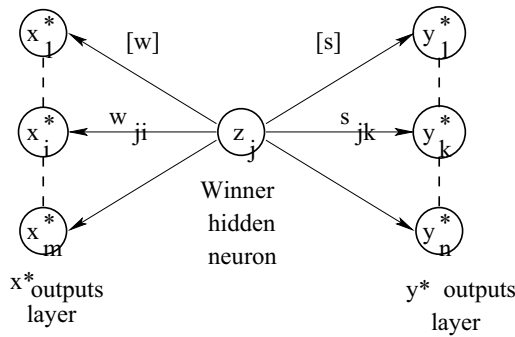


Figure 10.12: Out-star model of CPNN.

$[w]$ and $[s]$ are generated at random, initially, in the range of $(0, 1)$. Let γ be the learning rate between z_j and x^* outputs layer, and that between z_j and y^* outputs layer is denoted

by δ . The principle of Grossberg's learning is utilized during the training of out-star model. The connecting weights: $[w]$ and $[s]$ are updated as follows:

$$\begin{aligned} w_{ji}(\text{updated}) &= w_{ji}(\text{previous}) + \gamma(x_i - w_{ji}(\text{previous})), \\ &\quad i = 1, 2, \dots, m; \\ s_{jk}(\text{updated}) &= s_{jk}(\text{previous}) + \delta(y_k - s_{jk}(\text{previous})), \\ &\quad k = 1, 2, \dots, n. \end{aligned}$$

The values of x_i^* and y_k^* are then calculated as follows:

$$\begin{aligned} x_i^* &= w_{ji}(\text{updated}), \\ y_k^* &= s_{jk}(\text{updated}). \end{aligned}$$

It completes one iteration of out-star training. The values of learning rate: γ and δ are then reduced, which are used in the next iteration.

10.5.2 A Numerical Example

Let us assume that a full CPNN is to be used to model a data set having three inputs and two outputs. One such data is represented as $(x_1, x_2, x_3) = (0.3, 0.5, 0.6)$ and $(y_1, y_2) = (0.3, 0.4)$. Let us consider two hidden neurons only. Fig. 10.13 shows the CPNN architecture. The

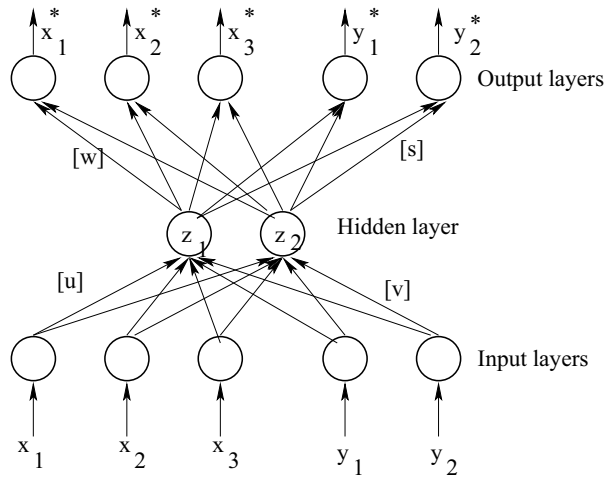


Figure 10.13: Schematic view of CPNN.

connecting weights are initially assumed to be as follows:

$$\begin{aligned} [u] &= \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.3 \\ 0.1 & 0.6 \\ 0.8 & 0.5 \end{bmatrix}; \\ [v] &= \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \end{bmatrix} = \begin{bmatrix} 0.4 & 0.7 \\ 0.2 & 0.3 \end{bmatrix}. \end{aligned}$$

$$[w] = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0.4 & 0.5 & 0.6 \\ 0.2 & 0.3 & 0.4 \end{bmatrix}.$$

$$[s] = \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} = \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.7 \end{bmatrix}.$$

The initial values of learning rates are assumed to be as follows: $\alpha = 0.2, \beta = 0.3, \gamma = 0.1, \delta = 0.4$ (the symbols carry the meaning as discussed in Section 10.5). Calculate the values of $x_1^*, x_2^*, x_3^*, y_1^*$ and y_2^* at the end of first iteration.

Solution:

Given $x_1 = 0.3, x_2 = 0.5, x_3 = 0.6, y_1 = 0.3, y_2 = 0.4$

Let us consider the in-star model first, as displayed in Fig. 10.14. In order to decide the

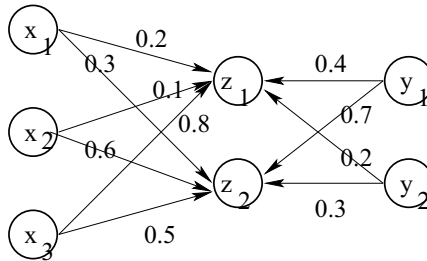


Figure 10.14: In-star model of the CPNN.

winner out of two hidden neurons, we calculate Euclidean distances, as follows:

$$\begin{aligned} d_1 &= \sqrt{\sum_{i=1}^3 (x_i - u_{i1})^2 + \sum_{k=1}^2 (y_k - v_{k1})^2} \\ &= \sqrt{(0.3 - 0.2)^2 + (0.5 - 0.1)^2 + (0.6 - 0.8)^2 + (0.3 - 0.4)^2 + (0.4 - 0.2)^2} \\ &= 0.51 \end{aligned}$$

$$\begin{aligned} d_2 &= \sqrt{\sum_{i=1}^3 (x_i - u_{i2})^2 + \sum_{k=1}^2 (y_k - v_{k2})^2} \\ &= \sqrt{(0.3 - 0.3)^2 + (0.5 - 0.6)^2 + (0.6 - 0.5)^2 + (0.3 - 0.7)^2 + (0.4 - 0.3)^2} \\ &= 0.44 \end{aligned}$$

As $d_2 < d_1$, z_2 is the winner. Its connecting weights are updated as given below.

$$u_{12}(\text{updated}) = u_{12}(\text{previous}) + \alpha(x_1 - u_{12}(\text{previous})) = 0.3 + 0.2(0.3 - 0.3) = 0.3$$

$$u_{22}(\text{updated}) = u_{22}(\text{previous}) + \alpha(x_2 - u_{22}(\text{previous})) = 0.6 + 0.2(0.5 - 0.6) = 0.58$$

$$u_{32}(\text{updated}) = u_{32}(\text{previous}) + \alpha(x_3 - u_{32}(\text{previous})) = 0.5 + 0.2(0.6 - 0.5) = 0.52$$

$$v_{12}(\text{updated}) = v_{12}(\text{previous}) + \beta(y_1 - v_{12}(\text{previous})) = 0.7 + 0.3(0.3 - 0.7) = 0.58$$

$$v_{22}(\text{updated}) = v_{22}(\text{previous}) + \beta(y_2 - v_{22}(\text{previous})) = 0.3 + 0.3(0.4 - 0.3) = 0.33$$

Now, let us consider the out-star model (refer to Fig. 10.15) involving the winner neuron only.

The updated values of the connecting weights are determined as follows:

$$w_{21}(\text{updated}) = w_{21}(\text{previous}) + \gamma(x_1 - w_{21}(\text{previous})) = 0.2 + 0.1(0.3 - 0.2) = 0.21$$

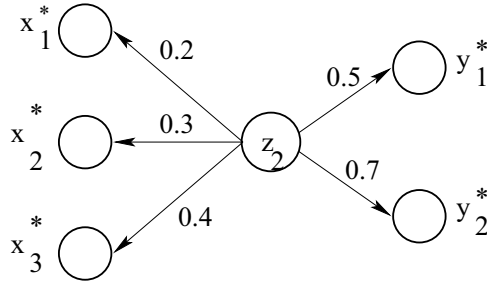


Figure 10.15: Out-star model of the CPNN.

$$w_{22}(\text{updated}) = w_{22}(\text{previous}) + \gamma(x_2 - w_{22}(\text{previous})) = 0.3 + 0.1(0.5 - 0.3) = 0.32$$

$$w_{23}(\text{updated}) = w_{23}(\text{previous}) + \gamma(x_3 - w_{23}(\text{previous})) = 0.4 + 0.1(0.6 - 0.4) = 0.42$$

$$s_{21}(\text{updated}) = s_{21}(\text{previous}) + \delta(y_1 - s_{21}(\text{previous})) = 0.5 + 0.4(0.3 - 0.5) = 0.42$$

$$s_{22}(\text{updated}) = s_{22}(\text{previous}) + \delta(y_2 - s_{22}(\text{previous})) = 0.7 + 0.4(0.4 - 0.7) = 0.58$$

Therefore, the values of x^* and y^* are obtained as follows:

$$x_1^* = 0.21, x_2^* = 0.32, x_3^* = 0.42, y_1^* = 0.42, y_2^* = 0.58$$

10.5.3 Forward-Only CPNN

Forward-only CPNN consists of one input layer and one output layer with a common hidden layer, which performs the clustering. The independent variables (inputs) (that is, $x_1, \dots, x_i, \dots, x_m$) are passed through the input layer and the dependent variables (outputs) (that is, $y_1, \dots, y_k, \dots, y_n$) are represented by the neurons of output layer. Therefore, clustering is done in the hidden layer using the input data only. Fig. 10.16 shows the schematic view of a forward-only CPNN. Similar to the full CPNN, the input and hidden

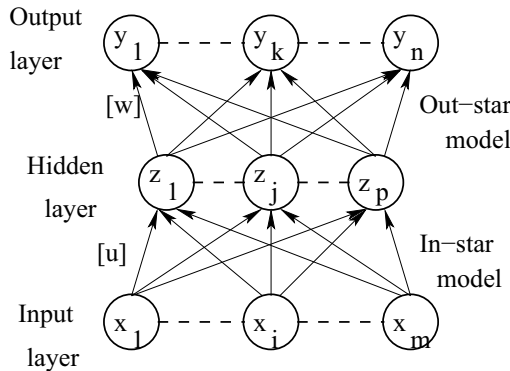


Figure 10.16: Forward-only CPNN.

layers form in-star model, and the hidden and output layers constitute the out-star model. The matrices: $[u]$ and $[w]$ represent the connecting weights of the in-star and out-star models, respectively. Kohonen's self-organizing map is utilized for the training of in-star model,

and the principle of Grossberg's learning is used in the training of out-star model. It is important to mention that the training is provided to this network by following the similar procedure adopted in the full CPNN. However, this network is much simpler compared to the full CPNN.

10.6 Recurrent Neural Networks (RNNs)

A Recurrent Neural Network (RNN) has both feed-forward as well as feed-back connections and as a result of which, information can be processed from the input layer to output layer and vice-versa. Thus, forms a cycle or loop. It requires a less number of neurons and consequently, becomes less expensive computationally compared to a feed-forward network. Two popular forms of RNN, namely Elman network, Jordan network and their combined form have been discussed below in detail.

10.6.1 Elman Network

It consists of three layers, such as input, hidden and output (refer to Fig. 10.17) [113, 115]. The inputs $[I_{ex}]$ (also known as external inputs) are fed to the network and feed-backs are

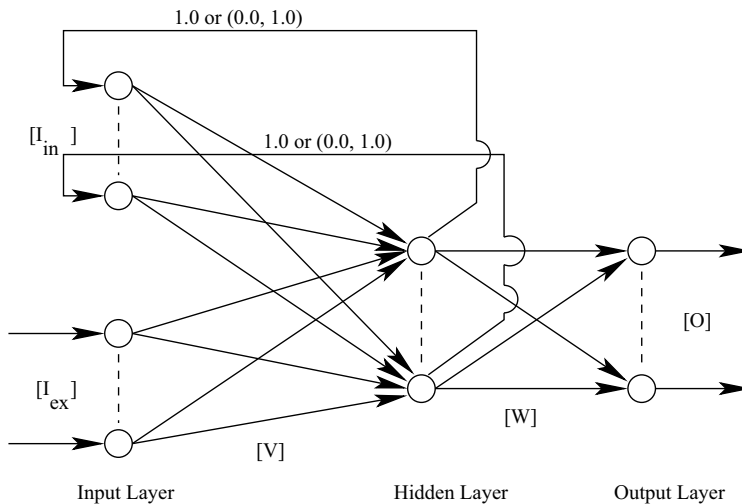


Figure 10.17: Elman network.

taken from the hidden neurons. The outputs of the hidden neurons are fed back as internal inputs $[I_{in}]$ to the network. Thus, the input layer contains both external as well as internal neurons (also called context neurons). The connecting weights between the hidden neurons and context neurons (also known as feed-back connection weights) can be either kept fixed to 1.0 or varied in the range of (0.0, 1.0). The final outputs of the network are represented by $[O]$. The $[V]$ and $[W]$ matrices represent the connecting weights between the input and hidden layers, and hidden and output layers, respectively. To train the network using a BP

algorithm, the $[V]$ and $[W]$ matrices are updated but the feed-back connection weights are generally kept fixed to 1.0. However, to update all the said three sets of weights, the BP algorithm may be replaced by a GA.

10.6.2 Jordan Network

It is composed of three layers, such as input, hidden and output layers [114, 115]. Here, external inputs $[I_{ex}]$ are fed to the network. The feed-backs are taken from the neurons of output layer and fed as internal inputs $[I_{in}]$ to the network. Fig. 10.18 shows the Jordan network. The feed-back weights can be either kept equal to 1.0 or updated in the range

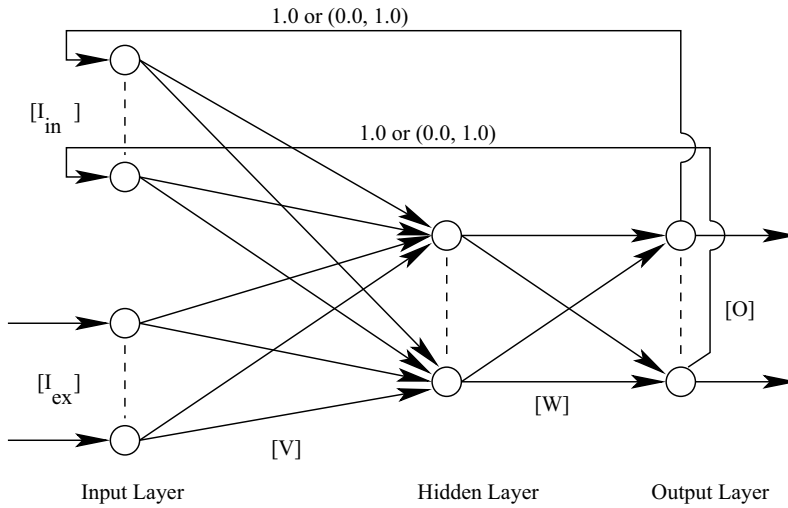


Figure 10.18: Jordan network.

of 0.0 to 1.0. However, the feed-forward connection weights, such as $[V]$ and $[W]$ will be modified during its training.

10.6.3 Combined Elman and Jordan Network

Fig. 10.19 shows the scheme of combined Elman and Jordan network. It consists of three layers, namely input, hidden and output layers. External inputs $[I_{ex}]$ are fed to the neurons of input layer. The neurons lying on the input, hidden and output layers will produce their outputs. The outputs of both hidden as well as output layers will be used as feed-backs to the network. Thus, internal inputs $[I_{in}]$ will be generated and then passed into the network through context neurons. The feed-back connection weights will be either kept equal to 1.0 or modified in the range of (0.0, 1.0). However, the feed-forward connecting weights: $[V]$ and $[W]$ will be updated during its training using an optimizer.

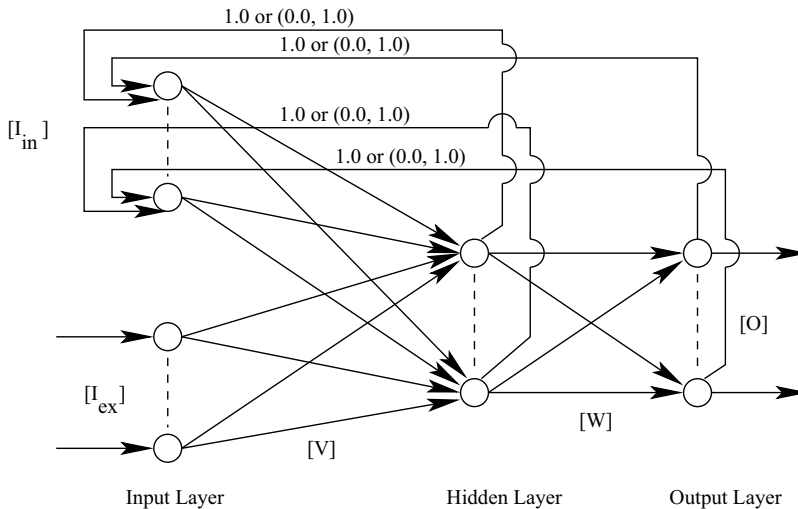


Figure 10.19: Combined Elman and Jordan network.

10.7 Summary

The principles of both feed-forward as well as recurrent neural networks have been explained in this chapter. In a feed-forward network, information is processed in one direction: starting from the input layer towards the output layer. The networks like multi-layer feed-forward neural network, radial basis function network, self-organizing map etc. come under the umbrella of feed-forward networks. In a recurrent neural network, information is processed in both forward as well as backward directions and thus, it forms a cycle or loop.

This chapter starts with the discussion of a multi-layer feed forward network. Training is provided to the said network using a back-propagation algorithm implemented through both incremental as well as batch modes. The learning ability of a radial basis function network is almost the same with that of a conventional multi-layer feed-forward network. However, the former may be computationally faster than the latter. Both the above feed-forward networks are trained using the principle of a supervised learning. A self-organizing map works based on the mechanism of a competitive learning, which is un-supervised in nature. It is an efficient topology preserving tool of mapping. The principle of counter-propagation neural network has also been explained. The chapter ends with a discussion on some forms of recurrent neural networks, such as Elman and Jordan networks.

10.8 Exercise

1. What do you mean by a feed-forward neural network ? How does it differ from a recurrent neural network ?
2. Explain the principle of a back-propagation algorithm. State its advantages and disadvantages.

3. Do you prefer a radial basis function network to a conventional multi-layer feed-forward network ? Explain it.
4. Make comments on the role of a self-organizing map as a dimensionality reduction technique.
5. Explain the principle of counter-propagation neural network. How does it differ from a back-propagation neural network?
6. Explain briefly the principles of Elman and Jordan networks.
7. Fig. 10.20 shows the schematic view of an NN consisting of three layers, such as input, hidden and output layers. The neurons lying on the input, hidden and output layers

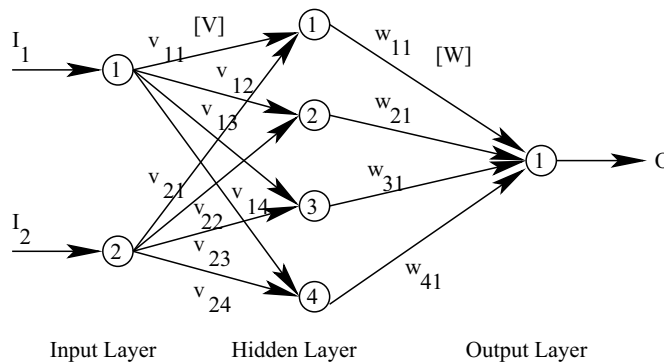


Figure 10.20: Schematic view of a neural network.

have the transfer functions represented by $y = x$, $y = \frac{1}{1+e^{-x}}$, $y = \frac{1}{1+e^{-x}}$, respectively. There are two inputs, namely I_1 and I_2 and one output, that is, O . The connecting weights between input and hidden layers are represented by $[V]$ and those between hidden and output layers are denoted by $[W]$. The initial values of the weights are assumed to be as follows:

$$\begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} \\ v_{21} & v_{22} & v_{23} & v_{24} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.3 & 0.2 & 0.4 \\ 0.6 & 0.3 & 0.5 & 0.2 \end{bmatrix};$$

$$\begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.7 \\ 0.3 \\ 0.2 \end{bmatrix}.$$

Using an incremental mode of training for the case shown in Table 10.2, calculate the changes in V (that is, ΔV) and W (that is, ΔW) values during back-propagation of error, assuming a learning rate $\eta = 0.4$. Show only one iteration.

Table 10.2: Training cases.

Sl. No.	I_1	I_2	O
1	0.6	0.4	0.9
2	-	-	-
\vdots	\vdots	\vdots	\vdots

8. A radial basis function network (RBFN) is to be used to model input-output relationships of a manufacturing process having three inputs and one output. Fig. 10.21 shows the RBFN with one hidden layer containing three neurons. The hidden neurons are assumed to have Gaussian transfer functions of the form: $y = f(x) = \exp[-\frac{(x-\mu)^2}{2\sigma^2}]$ with the values of mean μ and standard deviation σ as follows: $(\mu_1 = 4.0, \sigma_1 = 0.4)$; $(\mu_2 = 4.5, \sigma_2 = 0.6)$; $(\mu_3 = 5.0, \sigma_3 = 0.8)$. Assume initial weights as $w_{11} = 0.2$,

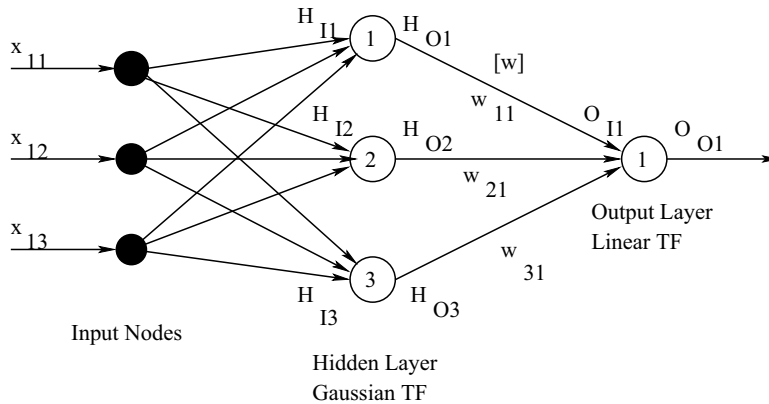


Figure 10.21: A radial basis function neural network.

$w_{21} = 0.1$, $w_{31} = 0.3$. Use incremental training scheme with the help of a scenario: $x_{11} = 0.8$, $x_{12} = 1.5$, $x_{13} = 2.5$ and output $O = 0.5$. Use back-propagation algorithm with a learning rate $\eta = 0.2$. Calculate the updated values of w_{11} , w_{21} , w_{31} , μ s, σ s. Show only one iteration.

Chapter 11

Combined Genetic Algorithms: Fuzzy Logic

Genetic Algorithm (GA) and Fuzzy Logic Controller (FLC) have been combined in two different ways. In the first approach, an FLC has been used to improve the performance of a GA (known as **fuzzy-genetic algorithm**), whereas in the second approach, a GA is utilized to improve the performance of an FLC (called **genetic-fuzzy system**). This chapter introduces the above two combinations of GA and FLC.

11.1 Introduction

Genetic Algorithm (GA) is a potential tool for global optimization; Fuzzy Logic (FL) is a powerful tool for dealing with imprecision and uncertainty and Neural Network (NN) is an important tool for learning and adaptation. However, each of these tools has its inherent limitations. Combined techniques are developed to remove the limitations of constituent tools and at the same time, to utilize their strengths. A large number of combined techniques (also known as soft computing techniques) have been developed to solve a variety of problems. These techniques include **fuzzy-genetic algorithm**, **genetic-fuzzy systems**, **genetic-neural systems**, **neuro-fuzzy systems**, **fuzzy-neural networks**, **genetic-neuro-fuzzy systems**, and others. The present chapter deals with two combinations of the GA and FL technique, namely fuzzy-genetic algorithm and genetic-fuzzy system.

11.2 Fuzzy-Genetic Algorithm

In a Fuzzy-Genetic Algorithm (FGA), a Fuzzy Logic (FL) technique is used to improve the performance of a Genetic Algorithm (GA). A GA is found to be an efficient tool for global optimization but its local search capability is seen to be poor. On the other hand, an FLC is a powerful tool for local search. Therefore, the global search power of a GA may

be combined with the local search capability of an FLC to develop an FGA. Moreover, the performance of a GA depends on its parameters, such as probability of crossover, probability of mutation, population size etc. and an FLC may be used to control these parameters.

A considerable amount of work had been carried out in the past and some of those are discussed below.

Lee and Takagi [116] proposed a dynamic parametric GA, in which the FLCs were used for controlling the GA-parameters. They used three FLCs for this purpose. The performance was measured in terms of Phenotypic Diversity Measures (PDM), as defined below:

$$PDM_1 = \frac{f_{best}}{\bar{f}}, \quad (11.1)$$

$$PDM_2 = \frac{\bar{f}}{f_{worst}}. \quad (11.2)$$

where f_{best} , \bar{f} , f_{worst} represent the best, average and worst fitness values, respectively. It is important to note that both PDM_1 and PDM_2 lie in the range of (0.0,1.0). If they are found to be near to 1.0, the convergence is reached, whereas if they are seen to be near to 0.0, the population shows a high level of diversity. The changes in the best fitness values since the last control action were taken as inputs of the FLCs and the outputs of three FLCs were the variables that control variations in the current mutation probability, crossover probability, and population size, respectively. Fig. 11.1 shows a schematic view of the dynamic parametric GA. They used it for controlling an inverted pendulum and showed

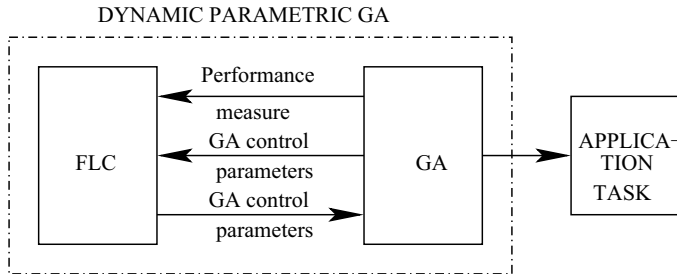


Figure 11.1: A schematic view showing the principle of a dynamic parametric GA [116].

that the performance of a dynamic parametric GA is better than that of a static GA.

Xu and Vukovich [117], Xu et al. [118] developed the FGA, whose working principle consists of the following steps:

- Choose control parameters before the GA is run,
- Adjust the control parameters on-line to dynamically adapt to new situations,
- Assist the user in accessing, designing, implementing and validating the GA for a given task.

Two FLCs were used in their work. The number of maximum generations and population size were considered as inputs of the FLCs, which produced two outputs, namely probability of crossover p_c and probability of mutation p_m . The FGAs were found to be more efficient than the conventional GA in solving traveling salesman and other optimization problems.

Herrera et al. [119] used Genotypic Diversity Measures (GDMs) for describing the state of population in a real-coded GA. They introduced Variance Average of the Chromosomes (VAC) and Average Variance of the Alleles (AVA) for the purpose of diversity measures. The VAC and AVA are defined as follows:

$$VAC = \frac{\sum_{i=1}^N (\bar{S}_i - \bar{S})^2}{N}, \quad (11.3)$$

$$AVA = \frac{\sum_{j=1}^N \sum_{i=1}^L (S_{ij} - \bar{S}_j)^2}{LN}. \quad (11.4)$$

where S_i ($i = 1, \dots, N$) denotes the chromosome i ; S_j ($j = 1, \dots, L$) indicates the vector of genes with position j in the population; S_{ij} represents the gene with position j in the chromosome i , and $\bar{S}_i = \frac{\sum_{j=1}^L S_{ij}}{L}$, $\bar{S} = \frac{\sum_{i=1}^N \sum_{j=1}^L S_{ij}}{LN}$, and $\bar{S}_j = \frac{\sum_{i=1}^N S_{ij}}{N}$. It is to be noted that the VAC and AVA will have low values, when all chromosomes in a population are almost identical. They used two FLCs for controlling p_c and p_m , depending on the VAC and AVA, respectively. The fuzzy rules developed were the following:

If VAC is low **then** p_c should be adjusted up-wards slightly,
If VAC is high **then** p_c should be forced down-wards slightly,
If AVA is low **then** p_m should be adjusted up-wards slightly,
If AVA is high **then** p_m should be forced down-wards slightly.

A fuzzy-genetic algorithm (FGA) had also been developed by Pratihari [120] to solve **find-path problems** of a mobile robot, which is described below.

A point mobile robot has to move from its initial position to a fixed final position in minimum traveling time by avoiding collision with some fixed obstacles. The said problem has been formulated as a constrained optimization problem and solved using a real-coded GA. The initial population of the GA created at random, may contain some infeasible solutions. Here, a fuzzy logic technique has been used to generate some feasible initial solutions for the GA and develop some efficient GA-operators like crossover and mutation. As the initial population of the GA contains some feasible solutions, the FGA is expected to be faster than the normal GA.

Fig. 11.2 shows a typical problem scenario, in which the robot will have to find its time-optimal, collision-free path while moving in the presence of eight static obstacles placed in a field size of $14 \times 14 \text{ m}^2$. Using the FGA, the robot is able to reach its destination G in the optimal sense, after starting from three different positions (S_1 , S_2 and S_3). The FGA is found to be computationally faster than the conventional technique, that is, tangent graph technique [121, 122, 123] along with A^* algorithm to solve the above problem and their performances are found to be comparable.

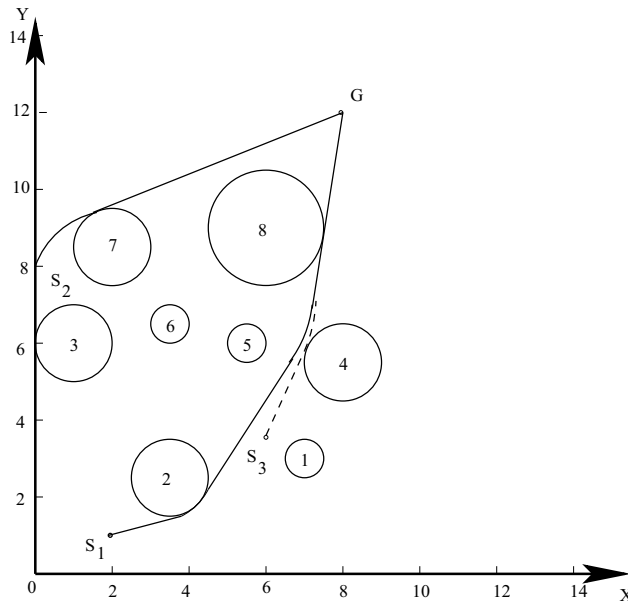


Figure 11.2: A typical find-path problem scenario.

11.3 Genetic-Fuzzy System

In this system, a GA is used to improve the performance of an FLC. It is to be noted that the performance of an FLC depends on its Knowledge Base (KB), which consists of Data Base (DB) and Rule Base (RB), and a GA-based tuning is done to improve it. As a GA is computationally expensive, the GA-based tuning is generally carried out off-line. Fig. 11.3 shows the schematic view of a Genetic-Fuzzy System (GFS). Once the optimized FLC is obtained, it can be used for on-line control of a process.

A GFS can be developed through the following ways:

- Optimizing/tuning the DB only,
- Optimizing/tuning the RB only,
- Optimizing/tuning both the DB as well as RB.

11.3.1 A Brief Literature Review

Several attempts had been made by various investigators to develop the GFSs and consequently, there exists a huge amount of literature [124]. Only a few of those attempts are mentioned below.

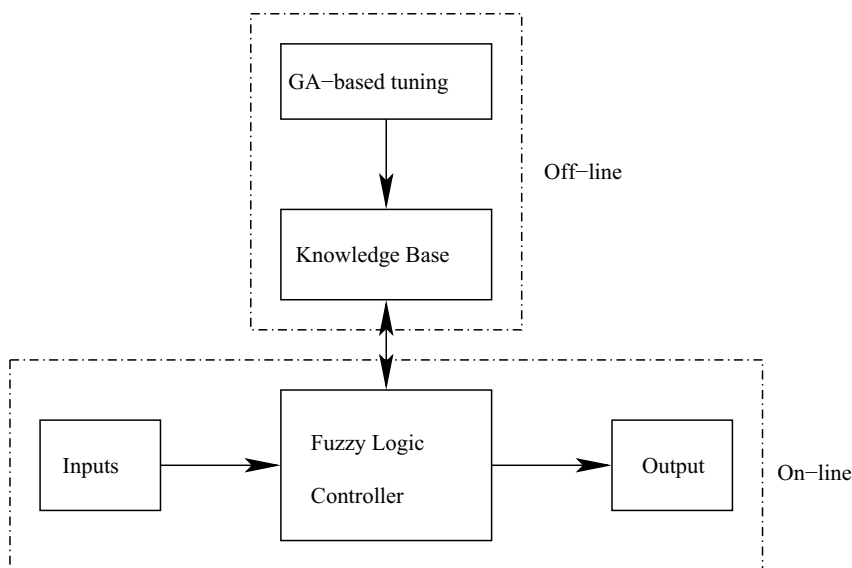


Figure 11.3: Schematic view of a Genetic-Fuzzy System (GFS).

Optimization of the DB of FLC:

Karr [125] proposed a GA-based learning process for determining the optimal DB of an FLC keeping its RB (previously defined) fixed. He used a simple GA with binary coding, proportionate selection scheme, simple crossover and random mutation. He considered

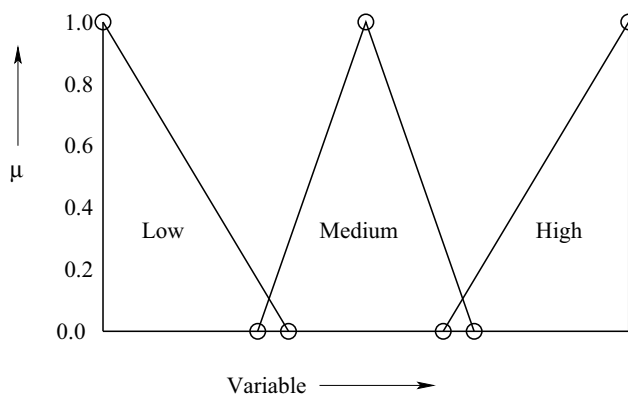


Figure 11.4: Tuning of membership function distributions [125].

triangular-shaped membership function distributions in his study, as shown in Fig. 11.4. The GA was used to expand or shrink the base of each interior isosceles triangle. The extreme triangles will be right-angled ones and the GA will make its base either the bigger

or smaller. In his study, the fitness function was expressed in terms of error.

Herrera et al. [126] implemented a GA-based tuning of the previously defined DB for an FLC. The process will start with a complete KB defined a priori. They used a real-coded GA with a stochastic universal sampling selection scheme, Michalewicz's non-uniform mutation operator and a Max-Min-Arithmetical crossover. They assumed the membership function distributions to be trapezoid in nature. In their coding, each individual of the population represents a complete KB. Two individuals in the population are different only in their DB definition but their RB definition will be the same. Thus, the GA will find a good DB for the FLC through evolution.

Optimization of the RB of FLC:

The DB of an FLC is pre-defined and kept unaltered, and the RB is optimized using a GA to improve its performance. There exist a number of approaches for RB optimization of an FLC, namely **Pittsburgh approach** [127, 128, 129, 130], **Michigan approach** [131, 132], **iterative rule learning approach** [133, 134, 135], and others.

In Pittsburgh approach, a string or chromosome of the GA represents the entire RB. Thus, a population of GA-strings represents a population of candidate RBs for the FLC. The GA tries to find the optimal RB with the help of its operators.

In Michigan approach, one rule of the FLC is represented by a GA-string and thus, the entire RB is indicated by the whole population of GA-strings.

In an iterative rule learning approach, the rules are represented by the GA-strings, the bad rules are removed from the RB and good ones are added to it iteratively. It consists of two stages [133]. In the first stage, a preliminary RB of the FLC is generated based on collected information of the process to be controlled. The generated RB is then modified in the second stage, after removing the redundant rule, if any, using a GA. In this approach, the rules are represented following the Michigan Approach (in which, one rule is coded using a chromosome/GA-string and the RB is indicated by the entire population of the GA) and the evaluation method follows the Pittsburgh Approach (where the cooperation of the rules has been attempted). There are two popular modules of the iterative rule learning approach, namely MOGUL (Methodology to Obtain GFRBSs Under the IRL approach), SLAVE (Structural Learning Algorithm in Vague Environment) and the interested readers may refer to [134], [135], respectively, to collect further information.

Optimization of both the DB and RB of FLC:

Lee and Takagi [136] suggested one method for derivation of complete KB of an FLC using a GA. They used triangular-shaped membership functions, in their study, although their method can work with all kinds of parameterized membership functions, namely Gaussian, bell, trapezoidal or sigmoid. They used a binary-coded GA. Information regarding the shape of membership function distributions is coded in the chromosome using the first 24-bits. The last part of the chromosome is built by coding the parameters w_i associated with each combination of the input values. In this way, each chromosome represents a complete KB.

The fitness function is defined based on optimizing two different criteria, namely a measure of convergence and number of rules present in the obtained RB.

Cooper and Vidal [137] developed a novel approach for genetic learning of FLC KB, in which they used a GA (with integer coding) with variable chromosomal length. A GA with an excessive length of the chromosome encoding the KB may not be able to find accurate solutions due to high complexity of the search space. They proposed an encoding scheme, which maintains only those rules necessary to control the systems. Thus, a GA will find the RB with the optimal number of rules. They considered triangular membership functions, each of them is specified by the location of its center and half base-width. The fitness function is designed using a measure of convergence.

Ng and Lee [138] proposed a method for GA-based learning of the complete KB of an FLC suitable for modelling of two-inputs-one-output systems, whose input and output spaces were partitioned into exactly seven primary fuzzy sets. It is important to note that their approach was suitable for a particular class of problems. The whole decision table (representing the RB) was encoded as the part of a chromosome. They used exponential membership function distributions, which were encoded in the second part of the chromosome. Thus, each chromosome represented a complete KB. In their approach, the fitness value was expressed as a measure of convergence. It is important to mention that their approach was not able to learn the number of rules that will constitute the optimal RB.

Liska and Melsheimer [139] used a GA for simultaneously discovering the fuzzy rules and membership functions, with a final stage of fine-tuning the membership functions using conjugate gradient descent method. They applied the system for learning a dynamic model of plant using the known input-output data.

Eiji Nawa et al. [140] proposed a Pseudo-Bacterial Genetic Algorithm (PBGA) with an adaptive operator to design an FLC for a semi-active suspension system. They introduced an adaptive operator in the PBGA to determine the points for bacterial mutation and also the cutting points for crossover operator. In their approach, each chromosome in the population encodes both the rules as well as membership functions of the variables. Their approach was able to design an FLC with the optimal rules.

11.3.2 Working Principle of Genetic-Fuzzy Systems

Genetic-Fuzzy Systems (GFSs) can be developed using the approaches explained below [120].

Approach 1: GA-based tuning of manually-constructed FLC

To control a process using an FLC, the designer first tries to identify its input (antecedent) and output (consequent) variables. Based on the designer's knowledge of the process to be controlled, the ranges of the variables are decided and different linguistic terms are utilized to express their qualities. The number of input combinations that decides the number of rules to be considered in the FLC depends on the number of input variables and their linguistic terms. The designer decides the nature of membership function distributions of the variables and thus, the DB of the FLC is designed manually. He/She then attempts to

design a suitable RB of the FLC. It is to be mentioned that the above manually-designed KB of the FLC may not be optimal in any sense. In this approach, a GA is used to tune the DB and/or RB of the FLC with the help of a set of training scenarios. After the GA-based tuning is over, the FLC will be able to determine the output for a set of inputs within a reasonable accuracy limit.

The principle of Approach 1 has been explained below with the help of one numerical example.

A Numerical Example:

A binary-coded GA is used to obtain optimal DB and RB of an FLC (Mamdani Approach) developed to model input-output relationships of a process. There are two inputs (I_1 and I_2) and one output (O) of the process. The membership function distributions of the inputs and output are shown in Fig. 11.5 and the manually-constructed rule base is shown in Table 11.1.

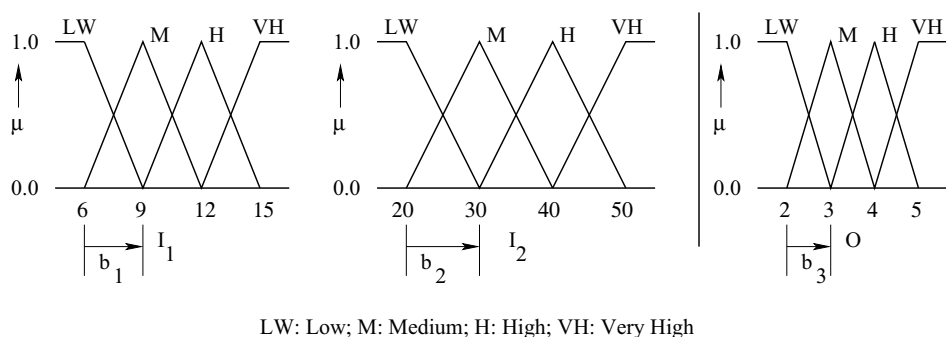


Figure 11.5: Membership function distributions of the inputs and output.

Table 11.1: Manually-constructed rule base of the FLC

		I_2			
		LW	M	H	VH
I_1	LW	LW	LW	M	H
	M	LW	M	H	H
	H	M	M	H	VH
	VH	M	H	VH	VH

The first rule of the above table can be read as follows:

IF I_1 is *LW* AND I_2 is *LW* THEN O is *LW*.

A binary-coded GA is used to optimize both DB as well as RB of the FLC with the help of a set of training cases. Some of the training cases are shown in Table 11.2.

Table 11.2: Training cases.

Sl. No.	I_1	I_2	O
1	10.0	28.0	3.5
2	14.0	15.0	4.0
\vdots	\vdots	\vdots	\vdots
T	17.0	31.0	4.6

An initial population of the binary-coded GA (size $N = 100$) is created at random (refer to Table 11.3). Starting from the left-most bit-position, five bits are assigned to indicate

Table 11.3: A population of GA-strings.

Sl. No.	GA-string
1	1011001101110111000101010111001
2	01100101110110100010101110110010
\vdots	\vdots
N	101000111101011110100100110111011

each of the b values (that is, b_1 , b_2 and b_3) and the next 16 bits represent the RB of the FLC (1 and 0 indicate the presence and absence of a rule, respectively). Thus, the GA-string is 31-bits long. Determine the deviation in prediction for the first training case shown in Table 11.2, using the first GA-string of Table 11.3. Explain the principle of GA-based tuning of manually-constructed KB of the FLC. The b values are assumed to vary in the ranges given below.

$$\begin{aligned} 2.0 &\leq b_1 \leq 4.0, \\ 5.0 &\leq b_2 \leq 15.0, \\ 0.5 &\leq b_3 \leq 1.5. \end{aligned}$$

Solution:

Let us consider the first string of initial population of GA-solutions, as given below.

$$\underbrace{10110}_{b_1} \underbrace{01101}_{b_2} \underbrace{11011}_{b_3} \underbrace{1000101010111001}_{rule\ base}$$

Linear mapping rule is used to determine real values of the variables. The Decoded Value (DV) of the binary sub-string 10110 is found to be equal to $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22$. Therefore, the real value of b_1 is determined as follows:

$$b_1 = b_1^{min} + \frac{b_1^{max} - b_1^{min}}{2^l - 1} \times DV$$

Here, $b_1^{min} = 2.0$, $b_1^{max} = 4.0$, $l = 5$, $DV = 22$. Substituting the values of b_1^{min} , b_1^{max} , l and DV in the above expression, we get $b_1 = 3.419355$.

Similarly, the real values of b_2 and b_3 are calculated, corresponding to their respective sub-strings and those are found to be equal to 9.193548 and 1.370968, respectively.

Fig. 11.6 shows the modified membership function distributions of the input and output variables.

The sub-string: 1000101010111001 indicates that the rules shown in Table 11.4, are present in the RB. Now, corresponding to the first training scenario given in Table 11.2,

Table 11.4: Rules represented by the GA-substring

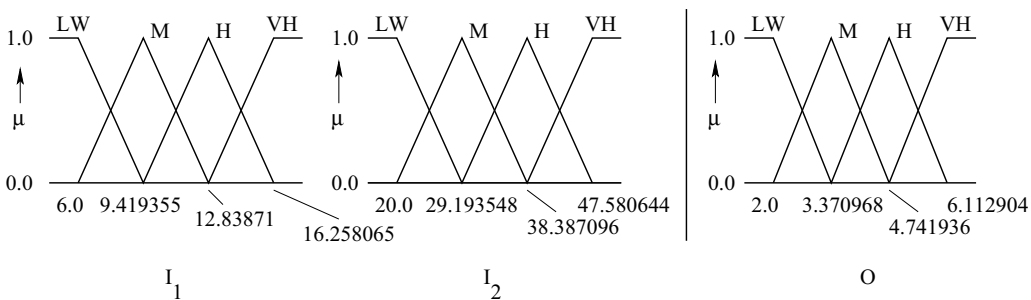
		I_2			
		LW	M	H	VH
I_1	LW	LW	-	-	-
	M	LW	-	H	-
	H	M	-	H	VH
	VH	M	-	-	VH

the output is calculated for the inputs: $I_1 = 10.0$, $I_2 = 28.0$, as follows:

The input $I_1 = 10.0$ may be declared either M or H and the other input $I_2 = 28.0$ may be called either LW or M .

Therefore, the following two rules are being fired from a total of eight rules shown in Table 11.4:

If I_1 is M AND I_2 is LW Then O is LW ,
If I_1 is H AND I_2 is LW Then O is M .



LW: Low; M: Medium; H: High; VH: Very High

Figure 11.6: Modified membership function distributions of the inputs and output.

Using the above two fired rules of the FLC (Mamdani Approach), the fuzzified output corresponding to the said input variables has been obtained and the **Center of Sums**

method of de-fuzzification (refer to Section 8.2.1) is used to determine its crisp value. The output of the controller is found to be equal to 3.127984.

Therefore, the absolute value of deviation in prediction, that is, d_1 can be calculated as follows:

$$d_1 = |3.5 - 3.127984| = 0.372016$$

The values of absolute deviation in predictions (d_2, d_3, \dots, d_T) can be determined for 2-nd, 3-rd, \dots , T -th training scenarios, respectively, following the similar procedure (refer to Table 11.5). The average of absolute values of deviation in predictions (\bar{d}) can be calculated

Table 11.5: Absolute values of deviation in prediction for the training cases.

Sl. No.	I_1	I_2	O	Deviation in prediction
1	10.0	28.0	3.5	d_1
2	14.0	15.0	4.0	d_2
\vdots	\vdots	\vdots	\vdots	\vdots
T	17.0	31.0	4.6	d_T

like the following:

$$\bar{d} = \frac{\sum_{i=1}^T d_i}{T}.$$

The fitness of first GA-string, that is, f_1 is made equal to \bar{d} . Similarly, the fitness values of other strings of the GA-population can be determined (refer to Table 11.6).

Table 11.6: A population of GA-strings.

Sl. No.	GA-string	Fitness
1	1011001101110111000101010111001	f_1
2	0110010110110100010101110110010	f_2
\vdots	\vdots	\vdots
N	1010001110101110100100110111011	f_N

The population of GA-strings is then modified using different operators, such as reproduction, crossover and mutation and after a few generations, the GA will be able to evolve an optimal FLC.

Approach 2: Automatic design of an FLC using a GA

For a complicated process, it might be difficult for the designer to design a KB manually, beforehand and in that case, the performances of the GFS may not be up to the mark.

To overcome this difficulty, a method for automatic design of the FLC has been proposed. Here, a GA-string carries information of the DB, RB and consequent part of each rule of the FLC. The GA-based tuning of the FLC is done utilizing a pre-defined set of training scenarios. The GA will be able to evolve an optimal FLC through iterations.

In this approach, no time is spent on manual design of the FLC and the GA takes the whole responsibility of designing an optimal FLC. Thus, in this approach, the GA faces a more difficult task compared to that it does in Approach 1.

A Numerical Example:

Let us consider the same numerical example given above for Approach 1. However, it is assumed that the RB shown in Table 11.1 is missing. As there are four linguistic terms for each of the two variables, there are $4 \times 4 = 16$ possible combinations of them. The output of each of these 16 rules is not pre-defined and this task of determining an appropriate RB is given to the GA. As four linguistic terms are used to represent the output, only two bits may be used to represent each of them. For example, the linguistic terms: LW , M , H and VH are indicated by 00, 01, 10 and 11, respectively. Thus, the GA-string will be $5 + 5 + 5 + 16 + 2 \times 16 = 63$ -bits long. Table 11.7 shows the population of GA-strings. Determine the deviation in prediction for the first training scenario shown in Table 11.2.

Table 11.7: A population of GA-strings.

Sl. No.	GA-string
1	101100110111011100010101011100100011010010101001001011011101011
2	011001011011010001010111011001010000110101011100010101000010101
\vdots	\vdots
N	101000111010111010010011011101100111010100011100101010001100101

Solution:

Let us consider the first GA-string shown in Table 11.7.

$$\underbrace{10110}_{b_1} \underbrace{01101}_{b_2} \underbrace{11011}_{b_3} \underbrace{1000101010111001}_{\text{rule base}} \underbrace{00011010010101001001011011101011}_{\text{consequent part of the rules}}$$

Corresponding to the sub-string: 101100110111011, the real values of b_1 , b_2 and b_3 are found to be equal to 3.419355, 9.193548 and 1.370968, respectively (refer to the previous example). The modified membership function distributions of the input and output variables will be the same with those shown in Fig. 11.6. The sub-strings: 1000101010111001 and 00011010010101001001011011101011 indicate that the following rules are present in the RB of the FLC (refer to Table 11.8).

Now, corresponding to the first training scenario shown in Table 11.2, the output is calculated for the inputs: $I_1 = 10.0$ and $I_2 = 28.0$, in the following way.

The input $I_1 = 10.0$ may be declared either M or H and other input $I_2 = 28.0$ may be

Table 11.8: Rules represented by the GA-substring

		I_2			
		LW	M	H	VH
I_1	LW	LW	-	-	-
	M	M	-	M	-
	H	H	-	M	H
	VH	VH	-	-	VH

called either LW or M . Therefore, only the following two rules are being fired from a total of eight shown in Table 11.8:

If I_1 is M AND I_2 is LW Then O is M ,
If I_1 is H AND I_2 is LW Then O is H .

Using the above two fired rules of the FLC (Mamdani Approach), fuzzified output corresponding to the said input variables has been obtained and the **Center of Sums** method of de-fuzzification (refer to Section 8.2.1) is used to determine its crisp value. The output of the controller is found to be equal to 4.056452.

Therefore, the absolute value of deviation in prediction, that is, d_1 can be determined as follows:

$$d_1 = |3.5 - 4.056452| = 0.556452$$

The values of absolute deviation in predictions (d_2, d_3, \dots, d_T) can be determined for 2-nd, 3-rd, ..., T -th training scenarios, respectively, using the similar procedure (refer to Table 11.9). The average of absolute values of deviation in predictions (\bar{d}) can be calculated

Table 11.9: Absolute values of deviation in prediction for the training cases.

Sl. No.	I_1	I_2	O	Deviation in prediction
1	10.0	28.0	3.5	d_1
2	14.0	15.0	4.0	d_2
\vdots	\vdots	\vdots	\vdots	\vdots
T	17.0	31.0	4.6	d_T

like the following:

$$\bar{d} = \frac{\sum_{i=1}^T d_i}{T}.$$

The fitness of first GA-string, that is, f_1 is made equal to \bar{d} . Similarly, the fitness values of other strings of the GA-population can be obtained (refer to Table 11.10).

Table 11.10: A population of GA-strings.

Sl. No.	GA-string	
1	101100110111011100010101011100100011010010101001001011011101011	f_1
2	011001011011010001010111011001010000110101011100010101000010101	f_2
\vdots	\vdots	\vdots
N	101000111010111010010011011101100111010100011100101010001100101	f_N

The population of GA-strings is then modified utilizing different operators, such as reproduction, crossover and mutation and after a few generations, the GA will be able to find the optimal FLC.

Note: The GA-optimized RB of the FLC obtained above may contain some redundant rules. It happens due to iterative nature of the GA. The said RB may be further reduced by removing the redundant rule(s) (if any). To identify the redundant rules, the concept of **importance factor** has been used by Hui and Pratihari [141]. To determine the importance of a rule, both its probability of occurrence as well as worth have been considered. A rule is declared to be redundant and thus, may be eliminated from the RB, if its importance factor turns out to be smaller than a pre-specified value. The removal of a redundant rule will not create any non-firing situation.

11.4 Summary

In this chapter, the GA and FLC have been combined in two different ways, as discussed below.

1. In a fuzzy-genetic algorithm, the performance of a GA has been improved using an FLC. Here, an initial population of feasible solutions is created for the GA utilizing an FLC. The solutions are further improved using different GA-operators, namely reproduction, crossover and mutation.
2. In a genetic-fuzzy system, a GA is used to improve the performance of an FLC by optimizing its DB and/or RB. As a GA is found to be computationally expensive, the GA-based optimization of the FLC is carried out off-line. Once the optimized KB of the FLC is obtained, it can be utilized for making on-line prediction of the output.

The principles of both the above approaches have been explained with suitable examples.

11.5 Exercise

1. Explain the principle of a fuzzy-genetic algorithm.
2. Discuss briefly the principle of a genetic-fuzzy system with a suitable example.

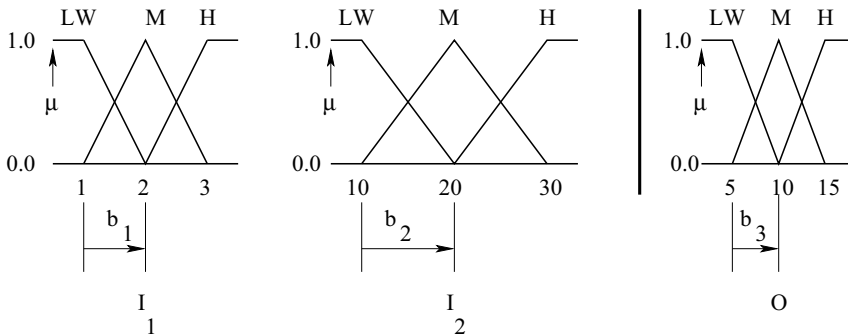
3. How can you identify redundant rule (if any) of a fuzzy logic controller during its training ? Explain with a suitable example.
4. An optimal FLC (Mamdani Approach) is to be developed using a binary-coded GA to predict input-output relationships of a manufacturing process. There are two inputs (I_1 and I_2) and one output (O) of the process. The membership function distributions of the inputs and output are shown in Fig. 11.7 and the manually-constructed RB is shown in Table 11.11.

Table 11.11: Manually-constructed RB of the FLC

		I_2		
		LW	M	H
I_1	LW	LW	LW	M
	M	LW	M	H
	H	M	H	H

The first rule of the above table can be read as follows:

IF I_1 is *LW* AND I_2 is *LW* THEN O is *LW*.



LW: Low; M: Medium; H: High

Figure 11.7: Membership function distributions of the inputs and output.

A binary-coded GA is used to optimize both DB as well as RB of the FLC with the help of a set of training cases. One of the training cases is shown in Table 11.12.

An initial population of solutions of a binary-coded GA (size $N = 100$) is created at random. One of the strings of initial population is shown below.

$\underbrace{100111010}_{rule\ base} \underbrace{10110}_{b_1} \underbrace{01110}_{b_2} \underbrace{11010}_{b_3}$

Table 11.12: Training cases

Sl. No.	I_1	I_2	O
1	1.5	22.0	14.0
2	-	-	-
\vdots	\vdots	\vdots	\vdots

The first 9 bits of the GA-string represent RB of the FLC (1 and 0 indicate the presence and absence of a rule, respectively) and 5 bits are assigned to denote each of the b values (that is, b_1 , b_2 and b_3). Thus, the GA-string is 24-bits long. Determine the deviation in prediction for the training case shown in Table 11.12, using the GA-string shown above. The b values are assumed to vary in the ranges given below.

$$0.5 \leq b_1 \leq 1.5,$$

$$5.0 \leq b_2 \leq 15.0,$$

$$2.0 \leq b_3 \leq 8.0.$$

Use Center of Sums method of defuzzification.

Chapter 12

Combined Genetic Algorithms: Neural Networks

This chapter introduces three approaches of combined Genetic Algorithms (GAs) - Neural Networks (NNs), popularly known as **Genetic-Neural Systems (GNSs)**. The working principle of a GNS has been explained in detail, with an example.

12.1 Introduction

Genetic Algorithms (GAs) have been combined with Neural Networks (NNs) to develop the combined GA-NN approaches (also known as Genetic-Neural Systems) by various researchers. In the combined GA-NN approaches, attempts are made to improve the performance of an NN using a GA. Whitley [142] provides an extensive survey on the combined GA-NN approaches. The GA has been combined with the NN in three different ways as discussed below.

- **Approach 1: GA-based tuning of connecting weights, bias values and other parameters**

The performance of a fixed architecture NN is dependent on the weights connecting the neurons of input and hidden layers and those of the hidden and output layers, bias values, learning rate, momentum term, co-efficients of the transfer functions used. A Back-Propagation (BP) algorithm may be utilized to optimize the above parameters in a supervised learning scheme. Being a gradient-based method, the chance of the BP algorithm for being trapped into the local minima is more. To overcome this difficulty, the BP algorithm has been replaced by a GA. Some of these attempts are mentioned below.

Harp et al. [143], Schaffer et al. [144] used a GA to set learning rate and momentum term of the feed-forward neural networks. Smith [145] utilized a GA to determine optimal weights in a fixed architecture NN. Sixteen inputs (sensory information), sixteen hidden neurons and two outputs (motor control information) were considered in

the NN. The optimal network was first developed through computer simulations and then downloaded to a real robot: **Khepera**, a miniature test mobile robot [146]. The robot was successful in locating the ball and scoring the goals on both computer simulations as well as real experiments. A large number of NN-based motion planners had been developed for the mobile robots following this approach by various investigators. Thus, a new field of robotic research called **evolutionary robotics** has emerged. It aims to evolve a suitable motion planner of the robot through artificial adaptation. Evolution and learning are two forms of biological adaptation, that operate on different time scales. Evolution captures slow environmental changes that might occur through several generations, whereas learning may produce adaptive changes in one's lifetime. In evolutionary robotics, artificial evolution technique, such as a GA and learning technique, namely an NN are merged to study interactions between evolution and learning, which accelerate the process of adaptation. Pratihari [147] has carried out a thorough review on evolutionary robotics. More recently, Hui and Pratihari [148] has developed an NN-based optimal motion planner using a GA for a two-wheeled differential drive robot.

It is important to mention that gradient-based methods could be more effective than the GA for weight optimization of the NN during its supervised learning, as the latter may suffer from the permutation problem. The GA may find it difficult to search and identify the set of optimal weights, out of a large number of possible sets. On the other hand, the chance of the solutions of a gradient-based method for being trapped into the local minima is more, whereas the probability of GA-solutions for getting stuck at the local minima is less. However, the performances of BPNN and GA-NN may depend on the nature of error surface. If the error surface is found to be uni-modal, the BPNN may find optimal solution faster than the GA-NN does. On the other hand, GA-NN may perform better than the BPNN, if the error surface is found to be multi-modal in nature [149].

- **Approach 2: GA-based tuning of neural network topologies**

The number of neurons lying on input and output layers is decided by the number of input and output variables, respectively, of the process to be controlled. The topology of the network can be varied by changing the number of hidden layer and putting different number of neurons in the said layer(s). Moreover, the network could be either a fully or partially connected one.

In some of the early efforts, the number of hidden layers and that of maximum hidden neurons were pre-defined. The networks were coded in the GA-strings of fixed length and the GA through its exhaustive search could determine optimal topology of the network. Besides connecting weights and other parameters, the GA was able to determine the optimized topology of the network. It is to be noted that the resulting network could most probably be a partially-connected one. In this connection, the work of Miller et al. [150], Whitley et al. [151], Hui and Pratihari [148] are worth-mentioning. If the number of hidden layer(s) and maximum number of neurons to be

present in the layer(s) are not pre-defined, it becomes a more difficult problem for the GA to find optimal topology of the network. It is so, because the length of GA-string will vary and it might be difficult to perform crossover operation of the GA.

- **Approach 3: GA-based preprocessing of data and interpretation of the outputs of an NN**

A large data set may sometimes contain some redundant data, for the removal of which, a GA may be used. It can remove an input or some of the inputs after ensuring that there is no significant change in behavior by doing this. Chang and Lippmann [152], Brill et al. [153] made some significant contributions in this field of research. Sometimes, it might also be required to determine a set of inputs, corresponding to a set of desired outputs. It may be called an inversion of a network, that is, running the network in backward direction. A GA might be helpful to carry out the inversion of an NN. Eberhart and Dobbins [154] used a GA to search the input space that produced the desired output. More recently, Mahesh et al. [4] has successfully implemented reverse mapping to control of a manufacturing process using the combined GA-NN approach. It is interesting to note that if the GA is run for a number of times, multiple sets of input parameters may be obtained that can yield a particular set of outputs.

12.2 Working Principle of a Genetic-Neural System

Fig. 12.1 shows a Neural Network (NN) consisting of three layers, such as input, hidden and output layers containing M , N and P number of neurons, respectively. The neurons of input, hidden and output layers are assumed to have linear, log-sigmoid and tan-sigmoid transfer functions, respectively. The symbols: I_{I1} , H_{I1} and O_{I1} represent the inputs of 1-st neuron lying on the input, hidden and output layers, respectively. Similarly, the outputs of 1-st neuron of the input, hidden and output layers are indicated by I_{O1} , H_{O1} and O_{O1} , respectively. The connecting weight between i -th neuron of input layer and j -th neuron of hidden layer is denoted by v_{ij} . Similarly, w_{jk} represents the connecting weight between j -th neuron of hidden layer and k -th neuron of output layer. The weight matrices $[V]$ and $[W]$ can be represented like the following:

$$[V] = \begin{bmatrix} v_{11} & \dots & v_{1j} & \dots & v_{1N} \\ \vdots & & \vdots & & \vdots \\ v_{i1} & \dots & v_{ij} & \dots & v_{iN} \\ \vdots & & \vdots & & \vdots \\ v_{M1} & \dots & v_{Mj} & \dots & v_{MN} \end{bmatrix}$$

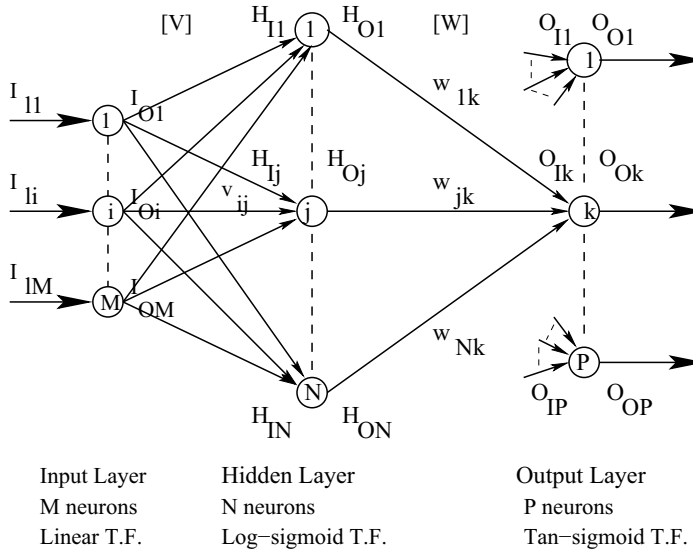


Figure 12.1: A feed-forward neural network.

$$[W] = \begin{bmatrix} w_{11} & \dots & w_{1k} & \dots & w_{1P} \\ \vdots & & \vdots & & \vdots \\ w_{j1} & \dots & w_{jk} & \dots & w_{jP} \\ \vdots & & \vdots & & \vdots \\ w_{N1} & \dots & w_{Nk} & \dots & w_{NP} \end{bmatrix}$$

It is to be noted that $[V]$ is an $M \times N$ matrix, whereas $[W]$ is an $N \times P$ matrix.

12.2.1 Forward Calculation

It consists of the following steps:

- **Step 1: Calculation of the outputs of input layer.** The outputs of the neurons of input layer are kept equal to the corresponding inputs, as they have linear transfer function of the form $y = x$. Therefore, we get

$$I_{Oi} = I_{Ii}, \quad (12.1)$$

where $i = 1, 2, \dots, M$.

- **Step 2: Calculation of the inputs of hidden layer.** The input of j -th hidden neuron can be determined as follows:

$$H_{Ij} = v_{1j}I_{O1} + \dots + v_{ij}I_{Oi} + \dots + v_{Mj}I_{OM}, \quad (12.2)$$

where $j = 1, 2, \dots, N$.

- **Step 3: Calculation of the outputs of hidden neurons.** As the hidden neurons are assumed to have log-sigmoid transfer function, the output of j -th hidden neuron can be determined like the following:

$$H_{Oj} = \frac{1}{1 + e^{-a_1 H_{Ij}}}, \quad (12.3)$$

where a_1 represents the co-efficient of transfer function.

- **Step 4: Calculation of the inputs of output layer.** The input of k -th output neuron can be determined as follows:

$$O_{Ik} = w_{1k}H_{O1} + \dots + w_{jk}H_{Oj} + \dots + w_{Nk}H_{ON}, \quad (12.4)$$

where $k = 1, 2, \dots, P$.

- **Step 5: Calculation of the outputs of output layer.** The neurons of output layer are assumed to have tan-sigmoid transfer function. The output of k -th neuron lying on the output layer can be determined like the following:

$$O_{Ok} = \frac{e^{a_2 O_{Ik}} - e^{-a_2 O_{Ik}}}{e^{a_2 O_{Ik}} + e^{-a_2 O_{Ik}}}, \quad (12.5)$$

where a_2 indicates the coefficient of transfer function.

- **Step 6: Calculation of error in prediction.** Let T_{Ok} represents the target output of k -th output neuron. The error in prediction at k -th output neuron corresponding to a particular training scenario can be determined as follows:

$$E_k = \frac{1}{2}(T_{Ok} - O_{Ok})^2. \quad (12.6)$$

The error in prediction considering all the output neurons can be calculated like the following:

$$E = \sum_{k=1}^P \frac{1}{2}(T_{Ok} - O_{Ok})^2, \quad (12.7)$$

where P denotes the number of outputs of the network.

In a GNS [147], an optimal network is evolved by minimizing the error in prediction of the outputs using a GA (in place of a steepest descent method utilized in back-propagation algorithm). A GA-string carries information of the NN, such as weight values: $[V]$, $[W]$ and coefficients of the transfer functions used, which will look as follows:

$$\underbrace{1001101011}_{v_{11}} \dots \underbrace{0100110101}_{v_{MN}} \underbrace{1100101001}_{w_{11}} \dots \underbrace{0110110100}_{w_{NP}} \underbrace{110 \dots 1}_{a_1} \underbrace{001 \dots 0}_{a_2}$$

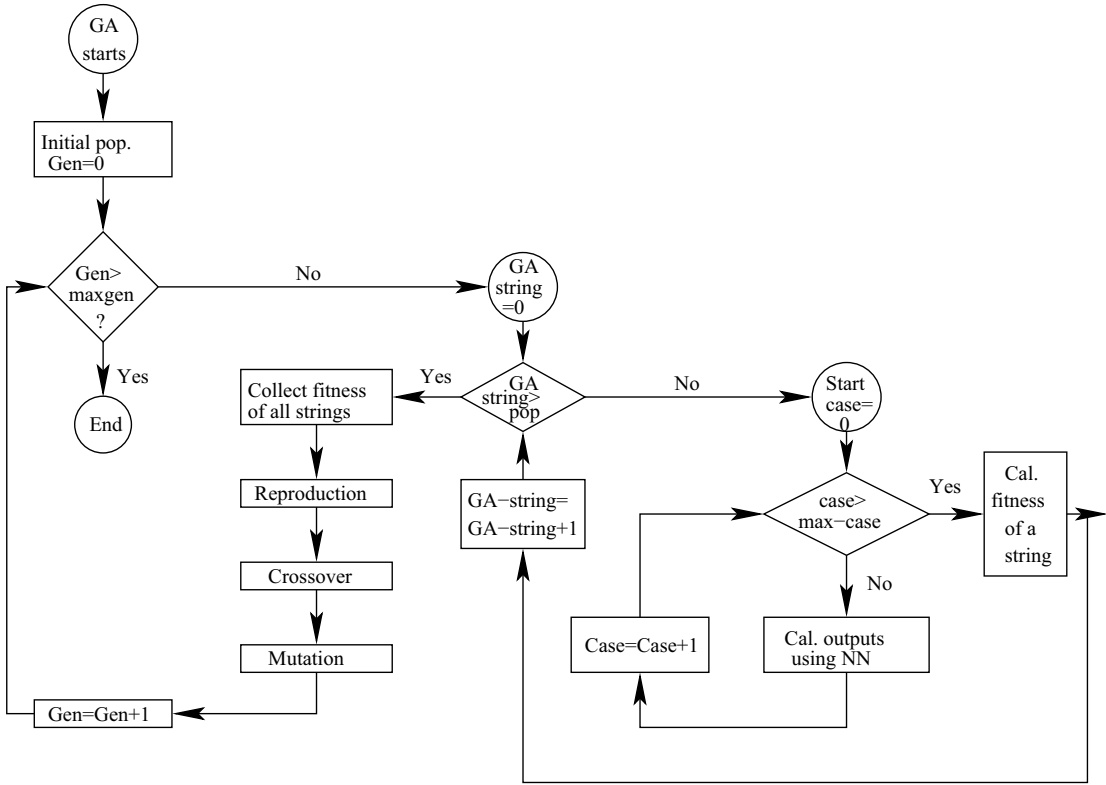


Figure 12.2: Flowchart of the GA-NN system.

Thus, a population of GA-strings represents a number of candidate NNs (whose number is kept equal to the population size).

Fig. 12.2 shows the flowchart of a GA-NN system. As a batch mode of training has been adopted, the whole training set is passed through the NN represented by a GA-string. The fitness f of the GA-string is made equal to Mean Squared Deviation (MSD) in prediction of the responses, which can be represented using the following expression:

$$f = \frac{1}{L} \frac{1}{P} \sum_{l=1}^L \sum_{k=1}^P \frac{1}{2} (T_{Ok l} - O_{Ok l})^2, \quad (12.8)$$

where L indicates the number of training scenarios. The fitness values of all strings lying in the GA-population are determined. The population of GA-strings (that is, NNs) is then modified using the operators, namely reproduction, crossover and mutation. The GA, through its search, will try to evolve an optimal NN.

12.2.2 A Hand Calculation

A binary-coded GA is to be used to update the connecting weights, bias value and coefficients of transfer functions of an NN shown in Fig. 12.3. The neurons of input, hidden

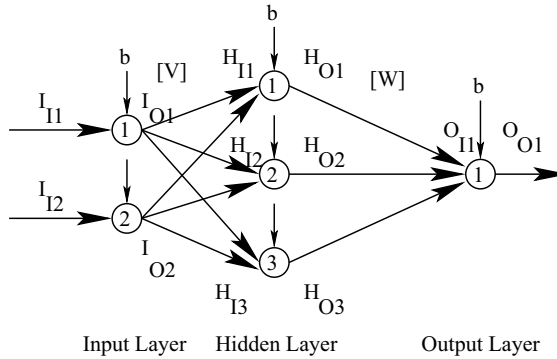


Figure 12.3: A 2 – 3 – 1 fully-connected feed-forward NN.

and output layers are assumed to have transfer functions of the forms: $y = x$, $y = \frac{1}{1+e^{-a_1x}}$, $y = \frac{e^{a_2x} - e^{-a_2x}}{e^{a_2x} + e^{-a_2x}}$, respectively. The weights: $[V]$, $[W]$ vary in the range of 0.0 to 1.0. The bias value b varies in the range of 0.001 to 0.01. Both the co-efficients of transfer functions: a_1 , a_2 are assumed to vary in the range of (0.5, 2.0). Table 12.1 shows one of the training cases.

One of the GA-strings is given below, in which five bits are used to represent each real

Table 12.1: Training cases

Sl. No.	I_1	I_2	O
1	0.6	0.7	0.9
\vdots	\vdots	\vdots	\vdots
L	-	-	-

variable.

$\underbrace{10110}_{v_{11}} \underbrace{01101}_{v_{12}} \underbrace{10001}_{v_{13}} \underbrace{01011}_{v_{21}} \underbrace{11011}_{v_{22}} \underbrace{00011}_{v_{23}} \underbrace{11001}_{w_{11}} \underbrace{11110}_{w_{21}} \underbrace{11101}_{w_{31}} \underbrace{10101}_{a_1} \underbrace{01110}_{a_2} \underbrace{11000}_b$

Determine the deviation in prediction for the training case shown in Table 12.1, corresponding to the above GA-string.

Solution:

The variable v_{11} is represented by the binary sub-string 10110. Its decoded value is found to be equal to 22. It varies in the range of (0.0, 1.0). Using the linear mapping rule, its real value can be determined as follows:

$$v_{11} = 0.0 + \frac{1.0-0.0}{2^5-1} \times 22 = 0.709677$$

Similarly, the real values of all variables represented by the above GA-string can be calculated and those are shown in Table 12.2.

Table 12.2: Real values of the variables represented by a GA-string

Sl. No.	Variable	Binary String	Decoded value	Range	Real value
1	v_{11}	10110	22	0.0, 1.0	0.709677
2	v_{12}	01101	13	0.0, 1.0	0.419355
3	v_{13}	10001	17	0.0, 1.0	0.548387
4	v_{21}	01011	11	0.0, 1.0	0.354839
1	v_{22}	11011	27	0.0, 1.0	0.870968
1	v_{23}	00011	03	0.0, 1.0	0.096774
1	w_{11}	11001	25	0.0, 1.0	0.806452
1	w_{21}	11110	30	0.0, 1.0	0.967742
1	w_{31}	11101	29	0.0, 1.0	0.935484
1	a_1	10101	21	0.5, 2.0	1.516129
1	a_2	01110	14	0.5, 2.0	1.177419
1	b	11000	24	0.001, 0.01	0.007968

The outputs of the neurons of input layer are calculated as follows:

$$I_{O1} = I_{I1} + b = 0.6 + 0.007968 = 0.607968$$

$$I_{O2} = I_{I2} + b = 0.7 + 0.007968 = 0.707968$$

The inputs of different neurons of hidden layer after adding the bias value are turning out to be like the following:

$$H_{I1} + b = I_{O1} \times v_{11} + I_{O2} \times v_{21} + 0.007968 = 0.690644$$

$$H_{I2} + b = I_{O1} \times v_{12} + I_{O2} \times v_{22} + 0.007968 = 0.879539$$

$$H_{I3} + b = I_{O1} \times v_{13} + I_{O2} \times v_{23} + 0.007968 = 0.409883$$

The outputs of hidden neurons are found to be as follows:

$$H_{O1} = 0.740219$$

$$H_{O2} = 0.791418$$

$$H_{O3} = 0.650545$$

The input of the neuron lying on output layer after adding the bias value is turning out to be like the following:

$$O_{I1} + b = H_{O1} \times w_{11} + H_{O2} \times w_{21} + H_{O3} \times w_{31} + b = 1.971413$$

The output of the network is found to be as follows:

$$O_{O1} = \frac{e^{a_2(O_{I1}+b)} - e^{-a_2(O_{I1}+b)}}{e^{a_2(O_{I1}+b)} + e^{-a_2(O_{I1}+b)}} = 0.981265$$

Now, the target output for this training case, is equal to 0.9 (refer to Table 12.1). Thus, the deviation in prediction is found to be equal to $0.9 - 0.981265 = -0.081265$.

12.3 Summary

Three different approaches of developing the GNS have been introduced. The mechanism of a GNS has been explained with the help of one numerical example.

12.4 Exercise

1. How do you design a genetic-neural system ? Explain with an example.
2. Is GA-NN preferred to the BPNN always ? Explain.
3. Suggest a suitable scheme of designing a genetic-neural system to optimize a neural network, after considering the varying topologies.
4. Make comments on the role of combined GA-NN approach in robot motion planning.
5. A binary-coded GA is used to update connecting weights of a partially-connected NN shown in Fig. 12.4. The neurons of input, hidden and output layers are assumed

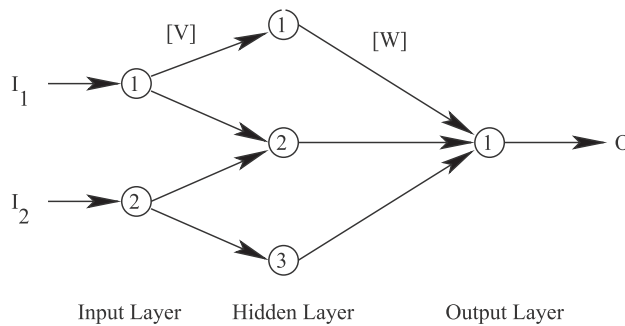


Figure 12.4: A 2 – 3 – 1 partially-connected feed-forward NN.

Table 12.3: One of the training cases

Sl. No.	I_1	I_2	O
1	0.6	0.7	0.8
\vdots	\vdots	\vdots	\vdots
-	-	-	-

to have transfer functions of the forms: $y = x^2$, $y = \frac{1}{1+e^{-x}}$, $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, respectively. The weights: $[V]$, $[W]$ vary in the range of 0.0 to 1.0. One of the training cases is shown in Table 12.3. Determine the deviation in prediction for the said training case, corresponding to a GA-string shown below, in which four bits have been assigned to represent each real variable.

$$\underbrace{1010}_{v_{11}} \underbrace{1100}_{v_{12}} \underbrace{0101}_{v_{22}} \underbrace{1111}_{v_{23}} \underbrace{0111}_{w_{11}} \underbrace{1001}_{w_{21}} \underbrace{0111}_{w_{31}}$$

Chapter 13

Combined Neural Networks: Fuzzy Logic

In this chapter, the principles of neuro-fuzzy systems designed and developed to obtain optimized FLCs based on Mamdani Approach and Takagi and Sugeno's Approach, separately, have been explained with appropriate examples.

13.1 Introduction

Neural Networks (NNs) have been combined with Fuzzy Logic (FL) techniques in two different ways. In one approach, a Fuzzy Logic Controller (FLC) is represented using the structure of an NN and trained utilizing either a Back-Propagation (BP) algorithm or another optimizer, say a Genetic Algorithm (GA). Thus, the performance of the FLC can be tuned using an NN. It is popularly known as **Neuro-Fuzzy System (NFS)**. The NFSs have been developed by various investigators to solve a variety of problems. In this connection, attempts of Keller et al. [155], Takagi et al. [156], Berenji and Khedkar [157], Takagi and Hayashi [158], Ishibuchi et al. [159], Jang [160] are worth mentioning.

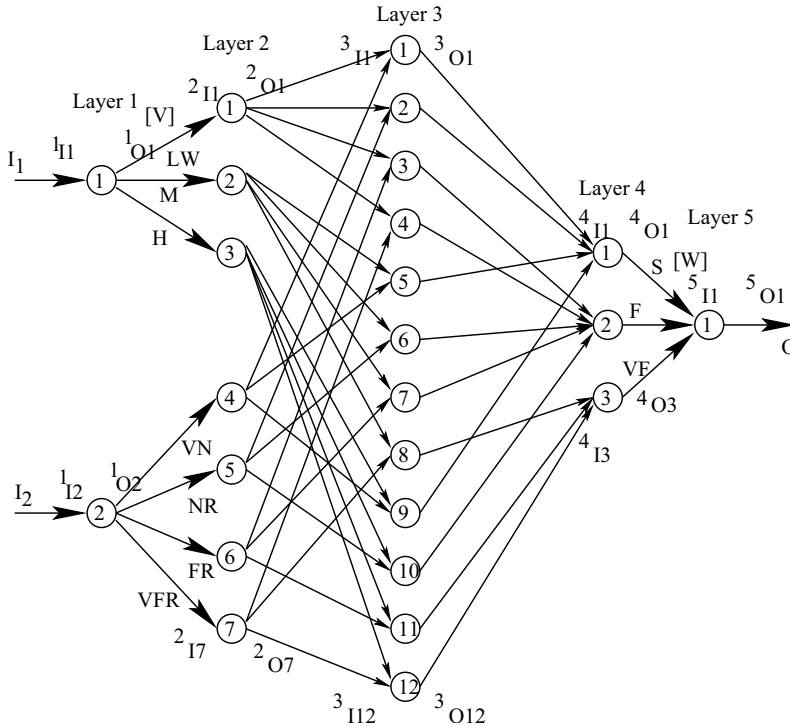
In another approach, the neurons of an NN have been designed using the concept of fuzzy set theory (refer to Section 7.2) and the developed network is generally called **Fuzzy Neural Network (FNN)** [161]. An FNN can be designed using the following combinations of input signals and connecting weights [162]:

- Real input signals but fuzzy weights,
- Fuzzy input signals but real weights,
- Fuzzy input signals and fuzzy weights.

In this chapter, the principles of two NFSs have been explained to develop the optimized FLCs working based on Mamdani Approach and Takagi and Sugeno's Approach, separately, as discussed below.

13.2 Neuro-Fuzzy System Working Based on Mamdani Approach

An attempt has been made by Hui et al. [163] to model Mamdani Approach of FLC using the structure of a feed-forward NN. Fig. 13.1 shows the schematic view of a Neuro-Fuzzy System (NFS). It consists of five layers, namely Layer 1 (input layer), Layer 2 (fuzzifica-



utilizing three (LW: Low, M: Medium, H: High), four (VN: Very Near, NR: Near, FR: Far, VFR: Very Far) and three (S: Slow, F: Fast, VF: Very Fast) linguistic terms, respectively. The membership function distributions of the inputs and output variables are assumed to be triangular in nature as shown in Fig. 13.2. Table 13.1 shows the manually designed Rule

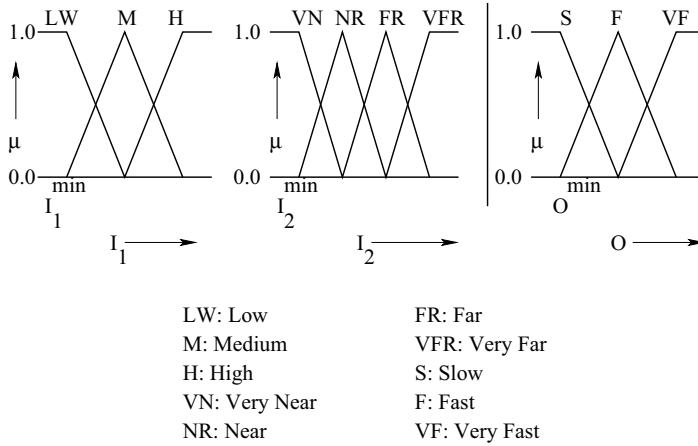


Figure 13.2: Membership function distributions of the input and output variables.

Base (RB) (consisting of $3 \times 4 = 12$ rules) of the FLC. A particular rule (say the first rule

Table 13.1: Rule base of the fuzzy logic controller

		I_2			
		VN	NR	FR	VFR
I_1	LW	S	S	F	F
	M	S	F	F	VF
	H	S	F	VF	VF

of the above table) will look as follows:

If I_1 is LW AND I_2 is VN THEN O is S.

To explain the principle of this NFS, let us consider j -th ($j = 1, 2$), k -th ($k = 1, 2, \dots, 7$), l -th ($l = 1, 2, \dots, 12$), m -th ($m = 1, 2, 3$) and n -th ($n = 1$) neurons of the first, second, third, fourth and fifth layers, respectively (refer to Fig. 13.3). The following notations are used: 1_{Ij} and 1_{Oj} represent the input and output, respectively, of j -th neuron lying on the first layer; v_{jk} indicates the connecting weight between j -th neuron of first layer and k -th neuron of second layer; $[W]$ denotes the connecting weight matrix between the neurons of fourth and fifth layers. The following assumptions are made to simplify the analysis (refer to Fig. 13.1):

$$v_{1, av} = \frac{v_{11} + v_{12} + v_{13}}{3},$$

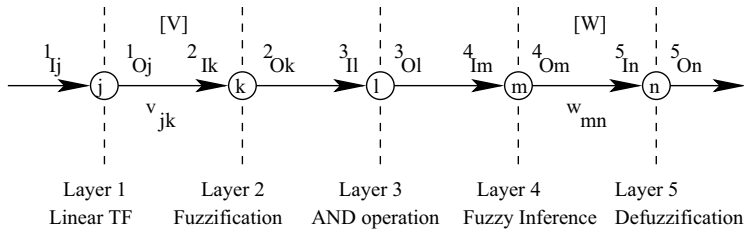


Figure 13.3: A specific neuron at each layer of the network.

$$\begin{aligned}
 v_{11} &= v_{12} = v_{13} = v_{1, \text{av}} (\text{say}), \\
 v_{2, \text{av}} &= \frac{v_{24} + v_{25} + v_{26} + v_{27}}{4}, \\
 v_{24} &= v_{25} = v_{26} = v_{27} = v_{2, \text{av}} (\text{say}), \\
 w_{av} &= \frac{w_{11} + w_{21} + w_{31}}{3}, \\
 w_{11} &= w_{21} = w_{31} = w_{av} (\text{say}).
 \end{aligned}$$

The functions of different layers of the NFS (refer to Fig. 13.1) have been explained below in detail.

- **Layer 1:** Two inputs I_1 and I_2 are fed to the network, that is, $1_{I1} = I_1$ and $1_{I2} = I_2$. As a linear transfer function has been considered in this layer, the outputs of the neurons lying on it will be equal to the corresponding inputs, that is, $1_{O1} = 1_{I1}$ and $1_{O2} = 1_{I2}$.
- **Layer 2:** It consists of seven neurons – the first three are connected to first input neuron and the remaining four are linked to second input neuron. The inputs of first three neurons of this layer are taken to be equal to the output of first input neuron, that is, $2_{I1} = 2_{I2} = 2_{I3} = 1_{O1}$. Similarly, the inputs of 4-th through 7-th neurons of this layer are kept the same with the output of second input neuron, that is, 1_{O2} . The aim of this layer is to convert the crisp values of the inputs into their corresponding membership values with the help of some distributions. For simplicity, the shape of above membership function distributions is assumed to be triangular in nature (refer to Fig. 13.2). The connecting weights between the first and second layers are represented by $[V]$ matrix. A particular element of this matrix is indicated by v_{jk} , that denotes the connecting weight between j -th neuron of input layer and k -th neuron of this layer. It is important to mention that the said connecting weights carry information of the sizes of triangular membership functions used to represent the corresponding linguistic terms of the variables. To determine the input-output relationships of this layer, three different possibilities are to be considered as discussed below.

1. **First Possibility:** The membership function distribution is a right-angled triangle representing the left-most triangle for each input variable of Fig. 13.2. Fig. 13.4 shows the above right-angled triangle used to indicate Low (LW) I_1

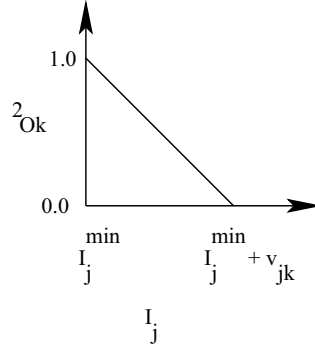


Figure 13.4: Left-most right-angled triangle as membership function distribution.

and Very Near (VN) I_2 . The output 2_{Ok} can be expressed in terms of I_j , I_j^{min} and v_{jk} as given below.

$$2_{Ok} = \begin{cases} 1.0 & \text{if } I_j \leq I_j^{min}, \\ \frac{I_j^{min} - I_j}{v_{jk}} + 1 & \text{if } I_j^{min} \leq I_j \leq (I_j^{min} + v_{jk}), \\ 0.0 & \text{if } I_j \geq (I_j^{min} + v_{jk}). \end{cases}$$

It is to be noted that for Low (LW) I_1 , j and k take the values of 1 and 1, respectively, whereas j and k are to be put equal to 2 and 4, respectively, for Very Near (VN) I_2 .

2. **Second Possibility:** The membership function distribution is a right-angled triangle representing the right-most triangle for each input variable of Fig. 13.2. The above right-angled triangle utilized to represent High (H) I_1 and Very Far (VFR) I_2 are shown in Fig. 13.5. The output 2_{Ok} can be expressed in terms of

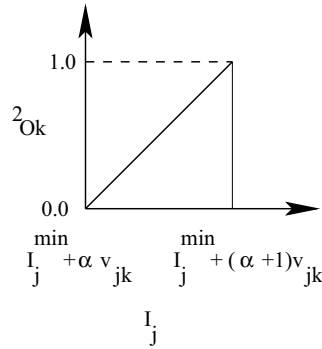


Figure 13.5: Right-most right-angled triangle as membership function distribution.

I_j , I_j^{min} and v_{jk} as given below.

$$2_{Ok} = \begin{cases} 0.0 & \text{if } I_j \leq (I_j^{min} + \alpha v_{jk}), \\ \frac{I_j - I_j^{min}}{v_{jk}} - \alpha & \text{if } I_j^{min} + \alpha v_{jk} \leq I_j \leq (I_j^{min} + (\alpha + 1)v_{jk}), \\ 1.0 & \text{if } I_j \geq (I_j^{min} + (\alpha + 1)v_{jk}). \end{cases}$$

It is to be noted that α , j and k are set equal to 1, 1 and 3, respectively, for the linguistic term: High (H) of I_1 . Similarly, corresponding to Very Far (VFR) I_2 , α , j and k take the values of 2, 2 and 7, respectively.

3. **Third Possibility:** The membership function distribution is a triangle shown in Fig. 13.6. The output 2_{Ok} can be expressed in terms of I_j , I_j^{min} and v_{jk} as

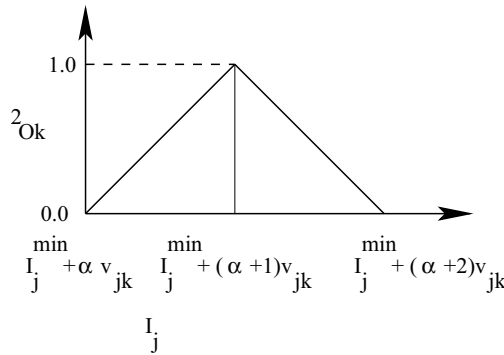


Figure 13.6: Triangular membership function distribution.

given below.

$$2_{Ok} = \begin{cases} 0.0 & \text{if } I_j \leq (I_j^{min} + \alpha v_{jk}) \text{ or } I_j \geq I_j^{min} + (\alpha + 2)v_{jk}, \\ \frac{I_j - I_j^{min}}{v_{jk}} - \alpha & \text{if } I_j^{min} + \alpha v_{jk} \leq I_j \leq (I_j^{min} + (\alpha + 1)v_{jk}), \\ \frac{I_j^{min} - I_j}{v_{jk}} + (\alpha + 2) & \text{if } I_j^{min} + (\alpha + 1)v_{jk} \leq I_j \leq (I_j^{min} + (\alpha + 2)v_{jk}). \end{cases}$$

It is important to note that α , j and k take the values of 0, 1 and 2, respectively, for the linguistic term: Medium (M) of input I_1 . Similarly, the values of (α, j, k) are kept equal to (0, 2, 5) and (1, 2, 6) for Near (NR) and Far (FR) linguistic terms, respectively, of input I_2 .

- **Layer 3:** It performs logical AND operation, which determines the minimum of a number of membership values. Each neuron of this layer is connected to two neurons (that is, one from each input) of the previous layer (refer to Fig. 13.1). Thus, a particular neuron of this layer indicates one combination of the input variables. For example, the first neuron of this layer represents the combination of two inputs as follows:

If I_1 is LW AND I_2 is VN .

The inputs of a particular neuron of this layer (say l -th) are nothing but the membership values of two inputs connected to it. These two membership values are compared and the lower one is selected as the output of this neuron, that is, 3_{Ol} .

- **Layer 4:** This layer carries out the task of fuzzy inference, as a result of which, the fired rules are identified along with their strengths for a set of inputs.
- **Layer 5:** The connecting weights between the Layers 4 and 5 decide the sizes of the triangles representing membership function distributions of the output variable. Once the membership function distributions of the linguistic terms are known, the outputs of the fired rules can be determined. It is to be noted that these outputs are expressed in terms of the areas of membership function distributions. To determine the combined fuzzified output of all fired rules, their outputs are superimposed. As the fuzzified output is nothing but an area, it cannot be utilized directly to control a process. Thus, a method of defuzzification is to be used to calculate the crisp value corresponding to this fuzzified output. Let us suppose that the Center of Sums Method (refer to Section 8.2.1) will be used for defuzzification. Thus, the final output (that is, a crisp value) can be obtained like the following:

$$5_{On} = 5_{O1} = \frac{\sum_{i=1}^r A_i f_i}{\sum_{i=1}^r A_i}, \quad (13.1)$$

where A_i and f_i are the area and center of area, respectively, for i -th fired rule; r represents the number of fired rules.

Let us suppose that a batch mode of training has been adopted to optimize the performance of the controller. It can be implemented using either a Back-Propagation (BP) algorithm or a Genetic Algorithm (GA), as explained below.

13.2.1 Tuning of the Neuro-Fuzzy System Using a Back-Propagation Algorithm

Let us assume that the Rule Base (RB) of the FLC shown in Table 13.1, has been kept unaltered during its training. Let us also suppose that the Data Base (DB) (that is, membership function distributions of the variables) of the FLC has been optimized during its training. It is to be noted that the DB of the FLC can be represented using the connecting weights of the neural network, that is, $[V]$ and $[W]$. A BP algorithm can be utilized to update these weight values.

Let us consider C training scenarios and Mean Squared Deviation (MSD) in prediction can be calculated as follows:

$$E = \frac{1}{2} \frac{1}{C} \frac{1}{N'} \sum_{c=1}^C \sum_{n=1}^{N'} (T_{Onc} - 5_{Onc})^2, \quad (13.2)$$

where N' indicates the number of outputs of the controller; T_{Onc} and 5_{Onc} represent the target and calculated outputs, respectively, of n -th output neuron of the network, corresponding to c -th training scenario.

In the network shown in Fig. 13.1, only one output has been considered, that is, $N' = 1$. Thus, the above expression of MSD can be re-written like the following:

$$E = \frac{1}{2} \frac{1}{C} \sum_{c=1}^C (T_{O1c} - 5_{O1c})^2. \quad (13.3)$$

The updated value of connecting weight w_{mn} (refer to Fig. 13.3) can be determined as

$$w_{mn,updated} = w_{mn,previous} + \Delta w_{mn},$$

where the change in w_{mn} can be calculated as follows:

$$\Delta w_{mn} = -\eta \frac{\partial E}{\partial w_{mn}}, \quad (13.4)$$

where η indicates the learning rate. For simplicity, the momentum term α' has been neglected in the above expression.

Now, $\frac{\partial E}{\partial w_{mn}}$ can be determined like the following:

$$\frac{\partial E}{\partial w_{mn}} = \frac{\partial E}{\partial E_c} \frac{\partial E_c}{\partial 5_{O1}} \frac{\partial 5_{O1}}{\partial 5_{I1}} \frac{\partial 5_{I1}}{\partial w_{mn}}.$$

Therefore, the updated value of w_{mn} can be determined.

Similarly, the connecting weight v_{jk} (refer to Fig. 13.3) can be updated according to the rule given below.

$$v_{jk,updated} = v_{jk,previous} + \Delta v_{jk},$$

where the change in v_{jk} can be obtained as follows:

$$\Delta v_{jk} = -\eta \frac{\partial E}{\partial v_{jk}}. \quad (13.5)$$

Now, $\frac{\partial E}{\partial v_{jk}}$ can be calculated like the following:

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial E_c} \frac{\partial E_c}{\partial 5_{O1}} \frac{\partial 5_{O1}}{\partial 5_{I1}} \frac{\partial 5_{I1}}{\partial 4_{Om}} \frac{\partial 4_{Om}}{\partial 4_{Im}} \frac{\partial 4_{Im}}{\partial 3_{Ol}} \frac{\partial 3_{Ol}}{\partial 3_{Il}} \frac{\partial 3_{Il}}{\partial 2_{Ok}} \frac{\partial 2_{Ok}}{\partial 2_{Ik}} \frac{\partial 2_{Ik}}{\partial v_{jk}}.$$

Therefore, the updated value of v_{jk} can be calculated.

13.2.2 Tuning of the Neuro-Fuzzy System Using a Genetic Algorithm

To improve the performance of FLC, the neuro-fuzzy system can also be trained using a GA, which is known as a **genetic neuro-fuzzy system**. A binary-coded GA may be utilized for the said purpose, whose strings will carry information of the RB and DB of the FLC. The neuro-fuzzy system shown in Fig. 13.1 involves 12 rules and there are three values of v

(such as v_{11}, v_{12}, v_{13}), four values of v (namely $v_{24}, v_{25}, v_{26}, v_{27}$) and three values of w (such as w_{11}, w_{21}, w_{31}) to represent the size of triangular membership function distributions of the first input I_1 , second input I_2 and output O , respectively. It is to be noted that the RB containing 12 rules will be denoted by 12 bits (1 for presence and 0 for absence) and to represent the real variables $[V]$ and $[W]$, some bits are to be assigned to ensure a desired accuracy. Thus, a particular GA-string will look as follows:

$$\underbrace{110\dots11}_{\text{Rule Base}} \underbrace{01\dots11}_{v_{11}} \dots \underbrace{10\dots10}_{v_{27}} \underbrace{01\dots11}_{w_{11}} \dots \underbrace{11\dots10}_{w_{31}}$$

The fitness of a GA-string may be determined like the following:

$$f = \frac{1}{2} \frac{1}{C} \sum_{c=1}^C (T_{O1c} - 5_{O1c})^2, \quad (13.6)$$

where C indicates the number of training scenarios; T_{O1c} and 5_{O1c} represent the target and calculated outputs, respectively, of the output neuron of the network, corresponding to c -th training scenario.

The GA creates its initial population of binary strings at random. The population of strings is then modified using the operators, namely reproduction, crossover and mutation. One iteration of these three operators followed by the evaluation is called a generation. The generations proceed until a termination criterion is reached. The GA, thus, optimizes both the DB as well as RB of the FLC simultaneously.

13.2.3 A Numerical Example

Fig. 13.7 shows the schematic view of a neuro-fuzzy system, in which an FLC (Mamdani Approach) has been represented using the structure of an NN. There are two inputs (such as I_1 and I_2) and one output (that is, O) of the controller. The NN consists of five layers and the function of each layer is indicated in the above figure. The input I_1 has been expressed using three linguistic terms: Near (NR), Far (FR), Very Far (VFR). Similarly, three other linguistic terms, namely Small (SM), Medium (M) and Large (LR) have been utilized to represent the second input I_2 . Moreover, the output O has been represented using three other linguistic terms, such as Low (LW), High (H) and Very High (VH). The membership function distributions of the inputs and output are assumed to be triangular in nature (refer to Fig. 13.8). For the first input I_1 , the base-widths of NR , VFR and half base-width of FR triangles are kept equal to v_{11} , v_{13} and v_{12} , respectively. Similarly, v_{24} , v_{26} and v_{25} represent the base-widths of SM , LR and half base-width of M triangles, respectively. Moreover, w_{11} , w_{31} and w_{21} indicate the base-widths of the triangles representing LW , VH outputs and half base-width of H triangle, respectively. The starting values of I_1 , I_2 and O are assumed to be equal to 1.0, 10.0 and 5.0, respectively. Determine the deviation in prediction for a case given below.

$$I_1 = 1.6, I_2 = 18.0, O = 9.0.$$

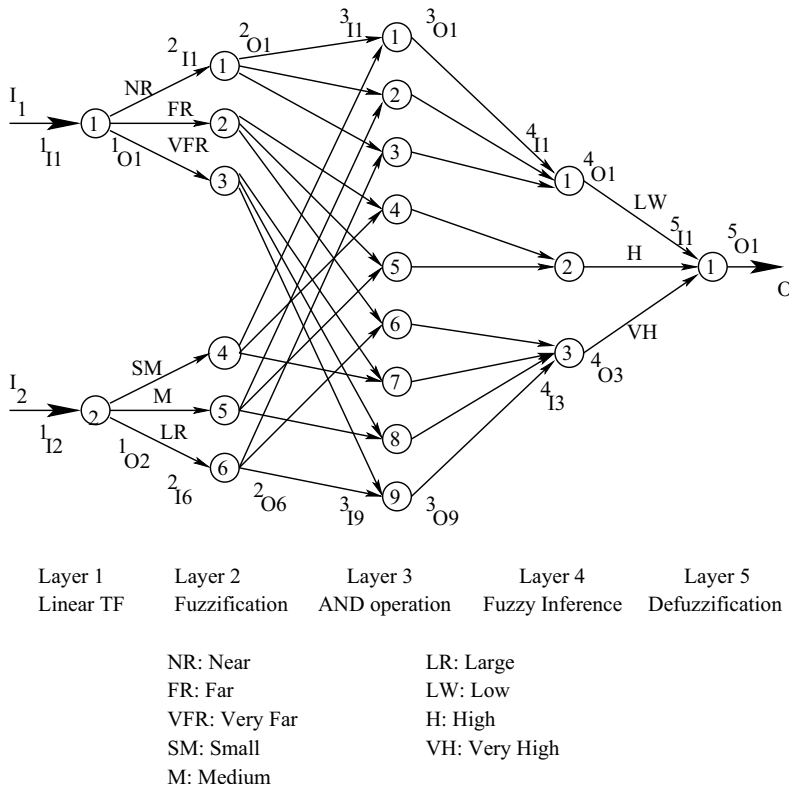


Figure 13.7: A neuro-fuzzy system.

Let us assume that $v_{11} = v_{12} = v_{13}$, $v_{24} = v_{25} = v_{26}$ and $w_{11} = w_{21} = w_{31}$ (refer to Fig. 13.7) for simplicity. The $[V]$ and $[W]$ matrices are considered as follows:

$$[v_{11} = v_{12} = v_{13} \ v_{24} = v_{25} = v_{26} \ w_{11} = w_{21} = w_{31}]^T = [0.3 \ 0.6 \ 0.4]^T.$$

Let us also suppose that b_1 , b_2 and b_3 of Fig. 13.8 represent the real values, corresponding to the normalized weights $v_{11} = v_{12} = v_{13}$, $v_{24} = v_{25} = v_{26}$ and $w_{11} = w_{21} = w_{31}$, respectively. The b values are assumed to lie within the following ranges:

$$\begin{aligned} 0.5 &\leq b_1 \leq 1.5, \\ 5.0 &\leq b_2 \leq 15.0, \\ 2.0 &\leq b_3 \leq 8.0. \end{aligned}$$

Use Center of Sums method for defuzzification.

Solution:

The normalized values of the variables are assumed to vary in the range of (0.0, 1.0). The relationship between the normalized value (n) and real value (x) of a variable can be expressed as follows:

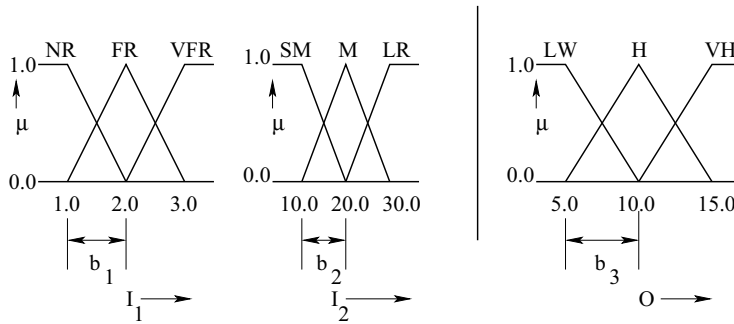


Figure 13.8: Manually constructed membership function distributions of the variables.

$$x = n \times (x^{max} - x^{min}) + x^{min}.$$

Using the above relationship, the b values are found to be like the following:

$$\begin{aligned} b_1 &= 0.3(1.5 - 0.5) + 0.5 = 0.8 \\ b_2 &= 0.6(15.0 - 5.0) + 5.0 = 11.0 \\ b_3 &= 0.4(8.0 - 2.0) + 2.0 = 4.4 \end{aligned}$$

Corresponding to these values of b_1 , b_2 and b_3 , the modified membership function distributions of the inputs and output variables are shown in Fig. 13.9.

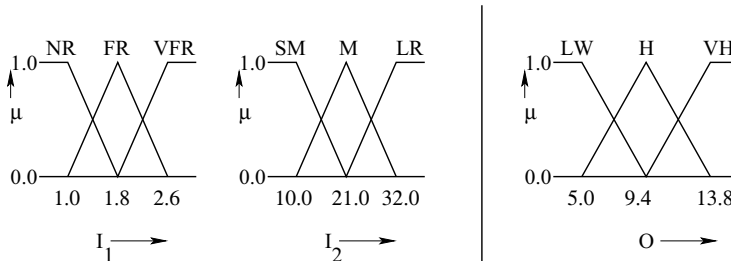


Figure 13.9: Modified membership function distributions of the variables.

Fig. 13.7 indicates that $3 \times 3 = 9$ rules are present in the RB of the FLC as shown in Table 13.2.

Table 13.2: Rule Base of the FLC

		I_2		
		SM	M	LR
I_1	NR	LW	LW	LW
	FR	H	H	VH
	VFR	VH	VH	VH

To determine output of the controller (refer to Fig. 13.10), input-output relationships of different layers are calculated as follows:

Layer 1:

$$\begin{aligned} 1_{I1} &= I_1 = 1.6 \\ 1_{I2} &= I_2 = 18.0 \end{aligned}$$

As the neurons of first layer are assumed to have linear transfer function, the outputs are calculated like the following:

$$\begin{aligned} 1_{O1} &= 1_{I1} = 1.6 \\ 1_{O2} &= 1_{I2} = 18.0 \end{aligned}$$

Layer 2:

The inputs of first three neurons of second layer are coming out to be the same and they are equal to 1_{O1} . Therefore,

$$2_{I1} = 2_{I2} = 2_{I3} = 1_{O1} = 1.6.$$

The input I_1 can be called either NR or FR . Therefore,

$$\begin{aligned} 2_{O1} &= \mu_{NR} = 0.25, \\ 2_{O2} &= \mu_{FR} = 0.75. \end{aligned}$$

Similarly, we get

$$2_{I4} = 2_{I5} = 2_{I6} = 1_{O2} = 18.0.$$

The input I_2 can be declared either SM or M . Therefore,

$$\begin{aligned} 2_{O4} &= \mu_{SM} = 0.272727, \\ 2_{O5} &= \mu_M = 0.727272. \end{aligned}$$

Layer 3:

It indicates that there are $3 \times 3 = 9$ possible combinations of the input variables. Only four input combinations will be activated corresponding to $I_1 = 1.6$ and $I_2 = 18.0$, as given below.

If I_1 is NR AND I_2 is SM Then –
If I_1 is NR AND I_2 is M Then –
If I_1 is FR AND I_2 is SM Then –
If I_1 is FR AND I_2 is M Then –

Thus, the first, second, fourth and fifth neurons of this layer will be activated.

$$\begin{aligned} 3_{I1} &= (0.25, 0.272727) \\ 3_{I2} &= (0.25, 0.727272) \\ 3_{I4} &= (0.75, 0.272727) \\ 3_{I5} &= (0.75, 0.727272) \end{aligned}$$

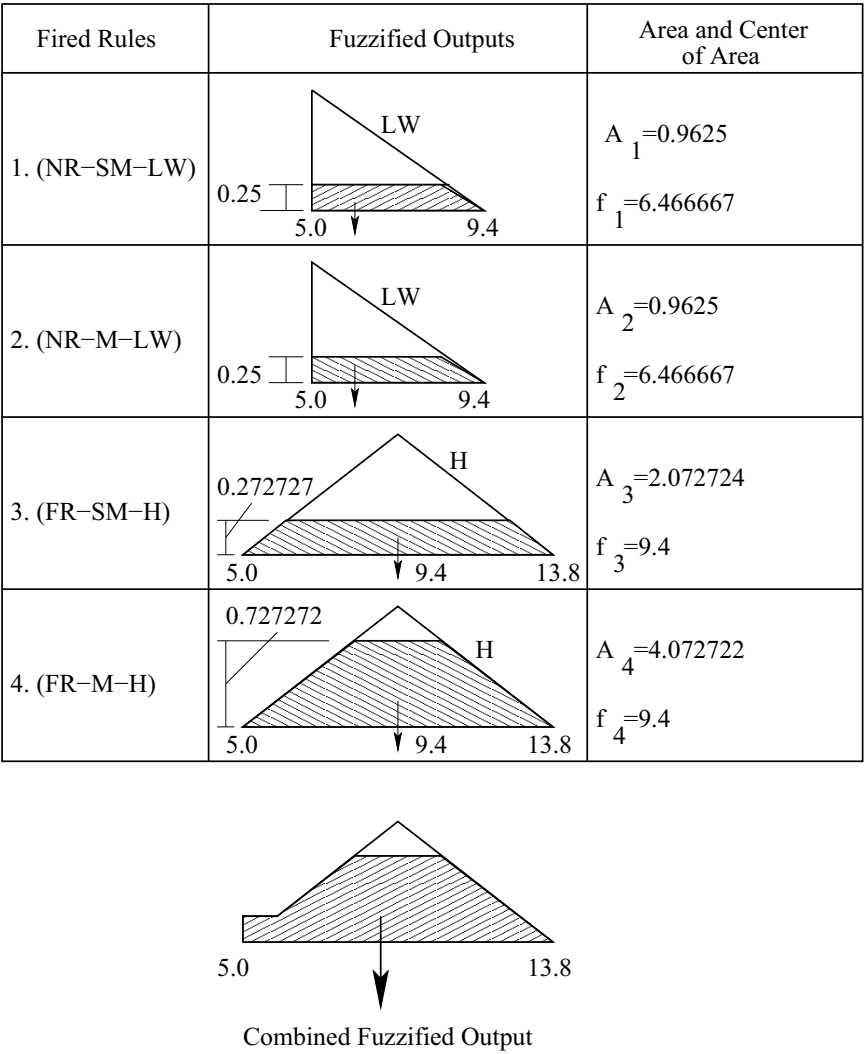


Figure 13.10: Fuzzified outputs of different fired rules.

As this layer performs **AND** operation, the outputs of above neurons can be determined like the following:

$$\begin{aligned} 3_{O1} &= \min(0.25, 0.272727) = 0.25 \\ 3_{O2} &= \min(0.25, 0.727272) = 0.25 \\ 3_{O4} &= \min(0.75, 0.272727) = 0.272727 \\ 3_{O5} &= \min(0.75, 0.727272) = 0.727272 \end{aligned}$$

Layer 4:

It indicates the outputs (consequent parts) of the activated input combinations. The output of this layer is nothing but the set of fired rules along with their strengths. The following four rules will be fired:

If I_1 is NR AND I_2 is SM Then O is LW
If I_1 is NR AND I_2 is M Then O is LW
If I_1 is FR AND I_2 is SM Then O is H
If I_1 is FR AND I_2 is M Then O is H

The strengths of above fired rules are found to be equal to 0.25, 0.25, 0.272727 and 0.727272, respectively.

Layer 5:

It determines the fuzzified output of different fired rules as shown in Fig. 13.10.

The output 5_{O1} is then calculated using the Center of Sums Method as follows:

$$5_{O1} = \frac{A_1 f_1 + A_2 f_2 + A_3 f_3 + A_4 f_4}{A_1 + A_2 + A_3 + A_4} = 8.700328$$

Now, Target output $T_{O1} = 9.0$.

Therefore, the deviation in prediction d is found to be equal to $9.0 - 8.700328 = 0.299672$.

13.3 Neuro-Fuzzy System Based on Takagi and Sugeno's Approach

The neuro-fuzzy system developed based on Takagi and Sugeno's Approach of FLC is also known as **Adaptive Neuro-Fuzzy Inference System (ANFIS)** [160]. Let us suppose that an ANFIS is to be developed to control a process involving two inputs (I_1 , I_2) and one output (O). The input I_1 is represented using three linguistic terms, namely LW : Low, M : Medium, H : High. Similarly, three other linguistic terms, such as SM : Small, LR : Large, VL : Very Large are utilized to indicate another input I_2 . For simplicity, the membership function distributions of the above linguistic terms are assumed to be triangular and symmetric in nature (refer to Fig. 13.11). The symbols: d_1 and d_2 indicate the base-width of right-angled triangles and half base-width of isosceles triangles representing membership function distributions of the first and second inputs, respectively. As there are two inputs and each input variable has been represented using three linguistic terms, there is a maximum of $3 \times 3 = 9$ possible combinations of them. According to first-order

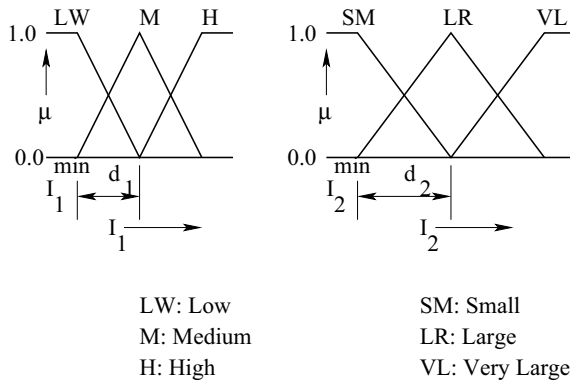


Figure 13.11: Membership function distributions of the input variables.

Takagi and Sugeno's model of FLC (refer to Section 8.2.1), the output of each rule can be expressed as follows:

$$y^i = a_i I_1 + b_i I_2 + c_i, \quad (13.7)$$

where $i = 1, 2, \dots, 9$; a_i, b_i, c_i are the coefficients.

To solve the said problem, an ANFIS architecture shown in Fig. 13.12 may be considered. It consists of six layers. The functions of different layers are explained below.

Let us assume that the following four rules are fired, corresponding to a set of inputs, say (I_1^*, I_2^*) :

- If** I_1 is *LW* and I_2 is *SM*, **then** $y^1 = a_1 I_1 + b_1 I_2 + c_1$,
If I_1 is *LW* and I_2 is *LR*, **then** $y^2 = a_2 I_1 + b_2 I_2 + c_2$,
If I_1 is *M* and I_2 is *SM*, **then** $y^4 = a_4 I_1 + b_4 I_2 + c_4$,
If I_1 is *M* and I_2 is *LR*, **then** $y^5 = a_5 I_1 + b_5 I_2 + c_5$.

To represent the inputs and outputs of a layer R , the following notations are used: R_{Ij} and R_{Oj} indicate the input and output, respectively, of j -th node lying on R -th layer ($R = 1, 2, 3, 4, 5, 6$).

- **Layer 1:** It is the input layer of the network. As there are two inputs, this layer consists of two nodes only (that is, one for each input). The function of this layer is to pass the inputs I_1^* and I_2^* to the next layer. The outputs of the nodes have been kept the same with the corresponding inputs, that is, $1_{O1} = 1_{I1} = I_1^*$ and $1_{O2} = 1_{I2} = I_2^*$.
- **Layer 2:** It determines membership values (μ s) for a set of inputs: (I_1^*, I_2^*) , corresponding to their appropriate linguistic terms. For example,

$$\begin{aligned} 2_{I1} &= I_1^*, \\ 2_{I2} &= I_1^*, \\ 2_{I4} &= I_2^*, \end{aligned}$$

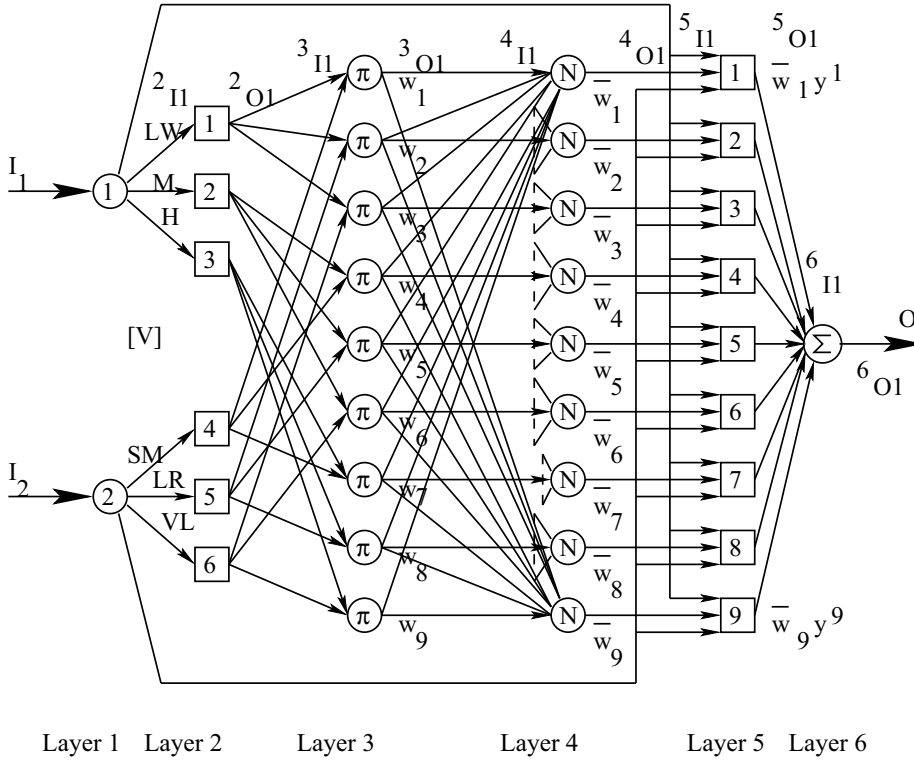


Figure 13.12: An ANFIS architecture.

$$\begin{aligned}
 2_{I5} &= I_2^*, \\
 2_{O1} &= \mu_{LW}(I_1^*), \\
 2_{O2} &= \mu_M(I_1^*), \\
 2_{O4} &= \mu_{SM}(I_2^*), \\
 2_{O5} &= \mu_{LR}(I_2^*).
 \end{aligned}$$

- Layer 3:** It contains $3 \times 3 = 9$ nodes, which are generally denoted by the symbol π . Each node indicates a possible combination of the input variables. Only four nodes (such as 1-st, 2-nd, 4-th and 5-th) will be activated, corresponding to above four fired rules. This layer takes the membership values (μ s) of different combinations of input variables as the inputs and determines their products as the outputs (also known as firing strengths (w s)) of corresponding nodes. For example,

$$\begin{aligned}
 3_{O1} &= \mu_{LW}(I_1^*) \times \mu_{SM}(I_2^*) = w_1 \text{ (say),} \\
 3_{O2} &= \mu_{LW}(I_1^*) \times \mu_{LR}(I_2^*) = w_2, \\
 3_{O4} &= \mu_M(I_1^*) \times \mu_{SM}(I_2^*) = w_4, \\
 3_{O5} &= \mu_M(I_1^*) \times \mu_{LR}(I_2^*) = w_5.
 \end{aligned}$$

- **Layer 4:** The number of nodes of this layer has been kept the same with that of the previous layer. These nodes are generally indicated by the symbol N (refer to Fig. 13.12). The normalized firing strength (\bar{w}) of each node of this layer is calculated as the ratio of firing strength of that node to the sum of strengths of all fired rules. For example,

$$\begin{aligned} 4_{O1} &= \frac{w_1}{w_1+w_2+w_4+w_5} = \bar{w}_1 \text{ (say),} \\ 4_{O2} &= \frac{w_2}{w_1+w_2+w_4+w_5} = \bar{w}_2, \\ 4_{O4} &= \frac{w_4}{w_1+w_2+w_4+w_5} = \bar{w}_4, \\ 4_{O5} &= \frac{w_5}{w_1+w_2+w_4+w_5} = \bar{w}_5. \end{aligned}$$

- **Layer 5:** It contains nine nodes and out of them, only a maximum of four will be activated (such as 1-st, 2-nd, 4-th and 5-th) for a set of inputs. The output of each fired node is calculated as the product of its normalized firing strength \bar{w} and output of the corresponding fired rule y . For example,

$$\begin{aligned} 5_{O1} &= \bar{w}_1 \times y^1, \\ 5_{O2} &= \bar{w}_2 \times y^2, \\ 5_{O4} &= \bar{w}_4 \times y^4, \\ 5_{O5} &= \bar{w}_5 \times y^5. \end{aligned}$$

- **Layer 6:** As there is only one output, this layer consists of one node. It performs the summation of $\bar{w}y$ values for the fired rules and consequently, is represented by the symbol \sum . The overall output can be determined as follows:

$$6_{O1} = \bar{w}_1 y^1 + \bar{w}_2 y^2 + \bar{w}_4 y^4 + \bar{w}_5 y^5.$$

The performance of an ANFIS depends on membership function distributions of the input variables decided by d values (that is, d_1 and d_2) of Fig. 13.11; coefficients: a_i , b_i , c_i of equation (13.7), and the optimal values of them can be determined using a batch mode of training carried out with the help of an optimizer, say a GA, as explained below.

Note: In layers 2 and 5, the nodes are represented by squares (but not circles). It is done so, just to indicate that input-output relationships of these nodes can be directly tuned during the training of the network. These nodes are called adaptive nodes, whereas those of layers 1, 3, 4 and 6 are fixed nodes.

13.3.1 Tuning of the ANFIS Using a Genetic Algorithm

A binary-coded GA can be used for tuning of the ANFIS, in which the string carries information of d_1 , d_2 , a_i , b_i and c_i ($i = 1, 2, \dots, 9$). For simplicity, the normalized values of the connecting weights between first node of Layer 1 and first three nodes of Layer 2 have been assumed to be equal to each other, that is, $v_{11} = v_{12} = v_{13}$ (refer to Fig. 13.12). Similarly, it has also been assumed that $v_{24} = v_{25} = v_{26}$ (in the normalized scale). It is important to mention that d_1 and d_2 indicate the real values corresponding to normalized values of v_{11} and v_{24} , respectively. Thus, a particular GA-string will look as follows:

$$\underbrace{110\dots11}_{d_1} \underbrace{01\dots11}_{d_2} \dots \underbrace{10\dots10}_{a_i} \underbrace{01\dots11}_{b_i} \underbrace{11\dots10}_{c_i} \dots$$

The fitness f of a GA-string may be calculated like the following:

$$f = \frac{1}{2} \frac{1}{Q} \sum_{q=1}^Q (T_{O1q} - 6_{O1q})^2, \quad (13.8)$$

where Q represents the number of training scenarios; T_{O1q} and 6_{O1q} are the target and calculated outputs, respectively, of the output layer of the network, corresponding to q -th training scenario. The GA through its search determines optimal weights of the network and coefficients of different rules. Thus, it forms a **genetic neuro-fuzzy system**.

13.3.2 A Numerical Example

Fig. 13.13 shows the schematic view of an ANFIS used to model a process having two inputs: I_1 , I_2 and one output O . The network consists of six layers. Two linguistic terms, such as LW (Low) and H (High) have been utilized to represent first input I_1 . Similarly, the second input I_2 has been expressed using two other linguistic terms, namely SM (Small) and LR (Large). The connecting weights (expressed in the normalized form from 0.0 to 1.0) between the nodes of first and second layers are denoted by $[V]$ matrix. The membership

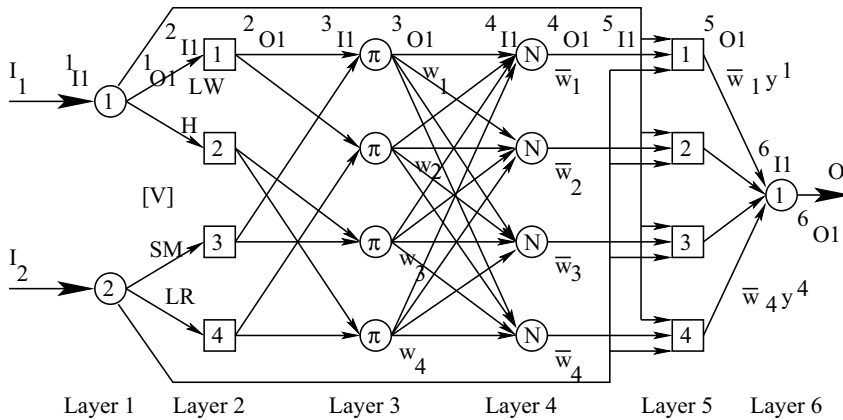


Figure 13.13: Schematic view of an ANFIS.

function distributions of the input variables are assumed to be triangular in nature, as shown in Fig. 13.14. The starting values of I_1 (that is, I_1^{min}) and I_2 (that is, I_2^{min}) are assumed to be equal to 1.0 unit and 5.0 units, respectively. For the first input I_1 , the base-width of LW and H right-angled triangles is kept equal to d_1 . Similarly, d_2 represents the base-width of SM and LR right-angled triangles. For simplicity, it has been assumed that $v_{11} = v_{12}$ and $v_{23} = v_{24}$. Moreover, d_1 and d_2 indicate the real values corresponding to

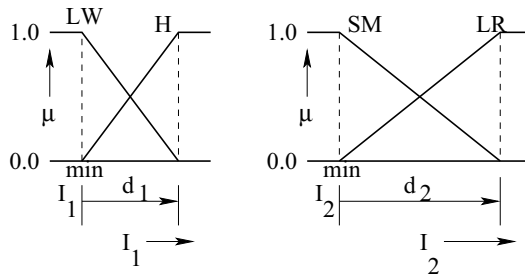


Figure 13.14: Membership function distributions of the input variables.

the normalized weights $v_{11} = v_{12}$ and $v_{23} = v_{24}$, respectively. As there are only two inputs and two linguistic terms have been utilized to represent each variable, there is a maximum of $2 \times 2 = 4$ possible rules. According to first-order Takagi and Sugeno's model of FLC, the rules can be expressed as follows:

$$y^i = a_i I_1 + b_i I_2 + c_i,$$

where $i = 1, 2, 3, 4$; a_i , b_i , c_i are the coefficients of the rules, whose values are given in Table 13.3.

The values of d_1 and d_2 vary in the ranges given below.

Table 13.3: The values of the coefficients for different rules.

Rule Number	a_i	b_i	c_i
1	0.2	0.3	0.10
2	0.2	0.4	0.11
3	0.3	0.3	0.13
4	0.3	0.4	0.14

$$0.8 \leq d_1 \leq 1.5,$$

$$4.0 \leq d_2 \leq 6.0.$$

Assume the normalized weight values as follows: $[v_{11} = v_{12} \ v_{23} = v_{24}]^T = [0.3 \ 0.5]^T$ and determine the deviation in prediction for a training scenario given below.

$$I_1 = 1.1, \ I_2 = 6.0, \ \text{Output } O = 2.3.$$

Solution:

The normalized values of the variables are assumed to vary in the range of (0.0, 1.0). The relationship between the normalized value (n) and real value (x) of a variable can be represented as follows:

$$x = n \times (x^{max} - x^{min}) + x^{min}$$

Using the above relationship, the d values are determined like the following:

$$d_1 = 0.3(1.5 - 0.8) + 0.8 = 1.01$$

$$d_2 = 0.5(6.0 - 4.0) + 4.0 = 5.0$$

Fig. 13.15 shows the modified membership function distributions of input variables, corresponding to the above values of d_1 and d_2 . The input-output relationships of different

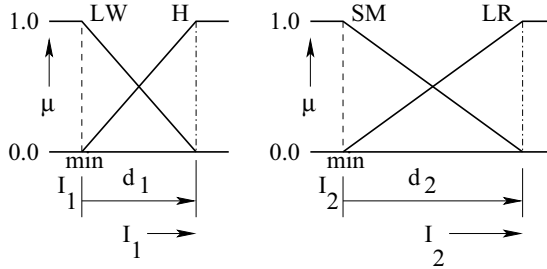


Figure 13.15: Modified membership function distributions of the variables.

layers are found to be as follows:

Layer 1:

$$1_{I1} = I_1^* = 1.1$$

$$1_{I2} = I_2^* = 6.0$$

As the neurons of first layer are assumed to have linear transfer function, the outputs are calculated like the following:

$$1_{O1} = 1_{I1} = 1.1$$

$$1_{O2} = 1_{I2} = 6.0$$

Layer 2:

The inputs of first two neurons of second layer are coming out to be equal to 1_{O1} . Therefore,

$$2_{I1} = 2_{I2} = 1_{O1} = 1.1.$$

The input I_1 can be called either LW or H . Therefore,

$$2_{O1} = \mu_{LW}(I_1^*) = 0.900990,$$

$$2_{O2} = \mu_H(I_1^*) = 0.099009.$$

Similarly, we get

$$2_{I3} = 2_{I4} = 1_{O2} = 6.0.$$

The input I_2 can be declared either SM or LR . Therefore,

$$2_{O3} = \mu_{SM}(I_2^*) = 0.8,$$

$$2_{O4} = \mu_{LR}(I_2^*) = 0.2.$$

Layer 3:

Corresponding to the inputs: $I_1^* = 1.1$, $I_2^* = 6.0$, all possible four rules will be fired and their inputs and outputs are given below.

$$\begin{aligned} 3_{I1} &= (0.900990, 0.8) \\ 3_{I2} &= (0.900990, 0.2) \\ 3_{I3} &= (0.099009, 0.8) \\ 3_{I4} &= (0.099009, 0.2) \end{aligned}$$

As this layer performs multiplication operation, the outputs of the above neurons can be determined like the following:

$$\begin{aligned} 3_{O1} &= \mu_{LW}(I_1^*) \times \mu_{SM}(I_2^*) = 0.720792 = w_1 \\ 3_{O2} &= \mu_{LW}(I_1^*) \times \mu_{LR}(I_2^*) = 0.180198 = w_2 \\ 3_{O3} &= \mu_H(I_1^*) \times \mu_{SM}(I_2^*) = 0.079207 = w_3 \\ 3_{O4} &= \mu_H(I_1^*) \times \mu_{LR}(I_2^*) = 0.019802 = w_4 \end{aligned}$$

Layer 4:

The normalized firing strengths (\overline{ws}) can be calculated as follows:

$$\begin{aligned} 4_{O1} &= \frac{w_1}{w_1+w_2+w_3+w_4} = 0.720792 = \overline{w_1} \\ 4_{O2} &= \frac{w_2}{w_1+w_2+w_3+w_4} = 0.180198 = \overline{w_2} \\ 4_{O3} &= \frac{w_3}{w_1+w_2+w_3+w_4} = 0.079207 = \overline{w_3} \\ 4_{O4} &= \frac{w_4}{w_1+w_2+w_3+w_4} = 0.019802 = \overline{w_4} \end{aligned}$$

Layer 5:

The outputs of the rules can be obtained like the following:

$$\begin{aligned} y^1 &= a_1 I_1^* + b_1 I_2^* + c_1 = 2.12 \\ y^2 &= a_2 I_1^* + b_2 I_2^* + c_2 = 2.73 \\ y^3 &= a_3 I_1^* + b_3 I_2^* + c_3 = 2.26 \\ y^4 &= a_4 I_1^* + b_4 I_2^* + c_4 = 2.87 \end{aligned}$$

The outputs of different neurons of fifth layer can be determined like the following:

$$\begin{aligned} 5_{O1} &= \overline{w_1} y^1 = 1.528079 \\ 5_{O2} &= \overline{w_2} y^2 = 0.491941 \\ 5_{O3} &= \overline{w_3} y^3 = 0.179008 \\ 5_{O4} &= \overline{w_4} y^4 = 0.056832 \end{aligned}$$

Layer 6:

The overall output 6_{O1} can be calculated as follows:

$$6_{O1} = \overline{w_1} y^1 + \overline{w_2} y^2 + \overline{w_3} y^3 + \overline{w_4} y^4 = 2.255860$$

Now, Target output $T_{O1} = 2.3$.

Therefore, the deviation in prediction is found to be equal to $2.3 - 2.255860 = 0.044140$.

13.4 Summary

The principles of two neuro-fuzzy systems have been explained with suitable examples. The optimal FLCs working based on Mamdani Approach and Takagi and Sugeno's Approach can be developed, separately, using the above principles.

13.5 Exercise

1. Explain the principle of a neuro-fuzzy system.
2. What do you mean by a fuzzy neural network ?
3. Refer to the numerical example 13.2.3 related to a neuro-fuzzy system. Let us assume that the matrices $[V]$ and $[W]$ given in the said numerical example are missing. Let us also suppose that a binary-coded GA is used to optimize both the DB as well as RB of the FLC simultaneously. To optimize the performance of the FLC using a GA, a set of training scenarios is utilized. Table 13.4 shows one of such training scenarios. One of the GA-strings contained in its initial population is shown below.

Table 13.4: One of the training cases

Sl. No.	I_1	I_2	O
1	1.6	18.0	9.0
\vdots	\vdots	\vdots	\vdots
-	-	-	-

$$\underbrace{101}_{b_1} \underbrace{111}_{b_2} \underbrace{011}_{b_3} \underbrace{100101110}_{RB}$$

Three bits are assigned to represent the base-width of right-angled triangle or half base-width of isosceles triangle and the remaining nine bits (1 for presence and 0 for absence of a rule) indicate the RB of the FLC. The b_1 , b_2 and b_3 values vary within their respective ranges given below.

$$\begin{aligned} 0.5 &\leq b_1 \leq 1.5, \\ 5.0 &\leq b_2 \leq 15.0, \\ 2.0 &\leq b_3 \leq 8.0. \end{aligned}$$

Determine the deviation in prediction for the training case given in Table 13.4, using the GA-string shown above.

4. Refer to the numerical example of Section 13.3.2 related to an ANFIS. Let us suppose that a binary-coded GA is utilized to optimize the base-widths (d_1, d_2) of membership function distributions of the input variables and co-efficients (a_i, b_i, c_i) of the rules. The said real variables are assumed to vary in the ranges given below.

$$\begin{aligned} 0.8 &\leq d_1 \leq 1.5, \\ 4.0 &\leq d_2 \leq 6.0, \\ 0.001 &\leq a_i, b_i, c_i \leq 1.0. \end{aligned}$$

To optimize the performance of the FLC using a GA, a set of training scenarios is used, one of which is shown in Table 13.5. Four bits are assigned to represent each

Table 13.5: One of the training cases

Sl. No.	I_1	I_2	O
1	1.1	6.0	5.0
\vdots	\vdots	\vdots	\vdots
-	-	-	-

of the real variables. One of the GA-strings contained in the initial population of solutions is shown below.

$$\underbrace{1011}_{d_1} \underbrace{1110}_{d_2} \underbrace{0111}_{a_1} \underbrace{1010}_{b_1} \underbrace{1010}_{c_1} \underbrace{0100}_{a_2} \underbrace{1001}_{b_2} \underbrace{0011}_{c_2} \underbrace{0010}_{a_3} \underbrace{1110}_{b_3} \underbrace{1010}_{c_3} \underbrace{0110}_{a_4} \underbrace{1100}_{b_4} \underbrace{1000}_{c_4}$$

Determine the deviation in prediction for the training case shown in Table 13.5, utilizing the GA-string given above.

Chapter 14

Applications of Soft Computing

14.1 Introduction

The principle of soft computing has been widely used to solve a variety of problems related to different fields, such as engineering science, general science, medical science, arts and commerce. It has been used as an optimizer, input-output modeler, controller, predictive tools, and so on. A huge literature is available on various applications of soft computing. A detailed discussion on all these applications is beyond the scope of this book. The present chapter deals with the applications of soft computing in two areas only, namely design and development of intelligent and autonomous robots, and data analysis.

14.2 Applications of Soft Computing in Design and Development of Intelligent Autonomous Robots

The term: **intelligence** has been defined in a number of ways. Out of these, the most popular definition is given by Albus [164], which is as follows: "..... intelligence is defined as an ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral sub-goals that supports the system's ultimate goal." An intelligent robot should ideally be able to learn, reason and perform like a human-being. A robotic system is called autonomous, if it can independently perform the assigned task without human guidance. It is to be noted that all intelligent systems may not be autonomous. An intelligent autonomous system will be capable of making decisions independently on-line, as the situation demands under varying conditions. The design and development of such a system involve the following sub-tasks:

- representation and incorporation of intelligence into the system,
- testing and validation of the system's performance.

Recent research in robotics aims to design and develop intelligent and autonomous robots, which include both fixed-based manipulators as well as mobile robots like wheeled robots, multi-legged robots (for example hexapod, quadrapod, biped), tracked vehicles, and others. These robots are now-a-days used in industries, sea-beds, space, medical science, military and security environments, and utilized as consumer products also. They should have the capability to plan their motion in a highly dynamic environment. They should be able to perform the following tasks in an unknown or a partially-known environment, as given below.

- **Step 1:** Collecting information of the environment using some sensors and/or cameras in order to determine the positions and orientations of both the robots as well as objects present in it.
- **Step 2:** Planning the decision of the robot as the situation demands.
- **Step 3:** Determining appropriate signals for the motors, which drive the mechanical structure of the robot to its desired locations.
- **Step 4:** Learning from the past experience to improve its performance.

It is important to mention that in all the above steps, the principle of soft computing has been used as discussed below.

14.2.1 Information Collection of the Environment

Either some sensors and/or cameras are generally used to gather information of the dynamic environment. Various types of sensors (such as position, velocity and acceleration sensors and others) are used in robotics [165]. The collected data may be imprecise in nature, as usual, which need further processing to retrieve useful information. Sometimes, these data are gathered utilizing multiple sensors, which are to be analyzed to extract useful information. Thus, multi-sensor data fusion is an important task and it could attract a number of researchers [166] working in soft computing.

On a number of applications, camera(s) has/have been used to collect information of the surroundings by the robots. Either over-head or on-board camera(s) is/are used for this purpose. The performance of a camera depends on a number of parameters, which are to be decided through calibration. The problem of camera calibration had been posed as an optimization problem and solved using some optimizers like GAs [167]. Once the images are collected using the camera(s), a suitable and fast image processing technique is to be used, so that the robot gets information of the environment as accurately as possible. The principle of soft computing had also been widely used to develop an efficient image processing technique [168, 169].

14.2.2 Motion Planning of Robots in Dynamic Environment

In a dynamic environment, the positions and orientations of other objects (obstacles) are to be known with respect to the robot on-line, so that it can decide its movement accordingly.

Thus, the robot will have to plan its path locally at different time steps to reach its goal. Both algorithmic and soft computing-based approaches had been tried to solve the motion planning problems of robots [170]. Latombe [171] provides an extensive survey on different algorithmic approaches of robot motion planning. These approaches include path velocity decomposition, incremental planning, relative velocity approach, potential field method, reactive control scheme, and others. Most of these approaches are computationally expensive, and thus, are not suitable for on-line implementations. Moreover, each of these approaches may be suitable for solving a particular type of problems. Thus, there is still a need for the development of an efficient, adaptive, versatile and computationally tractable algorithm for solving these problems of mobile robots.

Soft computing-based approaches had been utilized by a number of researchers to tackle the motion planning problems of mobile robots. Fuzzy logic-based motion planners had been developed by various investigators for the mobile robots [172, 173, 120]. Neural networks had also been utilized by some researchers to solve the said problem of mobile robots [174, 175, 176, 177]. After realizing the fact that it is difficult to explicitly (directly) design and develop a suitable motion planner for a robot, attempts were also made to evolve it and consequently, a new field of robotic research was started, which is popularly known as **evolutionary robotics** [147]. Here, an adaptive NN-based motion planner will be evolved using a GA through its interactions with the environment.

The problem of motion planning becomes more difficult and challenging, if the environment contains more than one mobile robots. Thus, planning is to be done for multiple robots working in the common space, which constitutes a multi-agent system (MAS) of robotics. For example, soccer playing robots form an MAS. In 1993, Alan Mackworth set the ultimate goal of RoboCup (that is, World Cup Robot Soccer) as follows [178]: "By the mid-21-st century, a team of autonomous humanoid robots shall beat the human world cup champion team under the official regulations of FIFA". A few attempts were also made to solve this problem using the principle of soft computing. In this connection, the work of Pratihari and Bibel [179], Taniguchi et al. [180] are worth to mention.

14.2.3 Determination of Appropriate Control Signals for the Motors

A robotic system consists of a number of mechanical members, and motors (actuators) are to be used for their movement. Each of the robotic joints is moved using a DC motor generally, which is controlled by a PID controller having some pre-defined gain values: K_P , K_I , K_D . However, it may be required to obtain adaptive values of K_P , K_I , K_D to efficiently control the motors. A few attempts were also made to determine optimal mechanical structure of the robot in terms of its power consumption and ease of control using some optimizers [181].

14.2.4 Learning from Previous Experience

Soft computing techniques had been used as learning/training tools for developing efficient motion planners of the robots. It is to be noted that this training can be provided either off-line or on-line. Either fuzzy logic- or neural networks-based expert system (also known

as knowledge-based system) may be developed through off-line training using some known scenarios (that is, training scenarios). Once the training is provided, this expert system can be used for on-line decision making. On the other hand, the said expert system may learn from its past experience on-line, while executing the task. Thus, its performance may not be good initially, but it improves with time. The GA had been widely used as a learning tool also along with fuzzy logic- or neural networks-based approaches [120, 182]. The principle of reinforcement learning could also receive much attention for this purpose, which works based on the history of success and failure while executing the task [183].

Therefore, soft computing plays an important role to design and develop intelligent and autonomous robots.

14.3 Applications of Soft Computing in Data Analysis

Classification and prediction are two forms of data analysis, whose aim is to extract useful information from a data set pertaining to a process/system. Thus, both the classification tools (also known as classifiers) and predictive models come under the umbrella of data mining, the principles of which are discussed below.

14.3.1 Classification Tools

Classification deals with discrete response and it works based on the principle of supervised learning implemented with the help of some pre-collected training data set.

A number of tools had been used by various researchers for classification, which include both conventional and soft computing-based ones. Conventional approaches like decision-tree classifier [184], Bayesian classifier [185] are some of the popularly used classification tools. A decision-tree, nothing but a flowchart-like structure, consists of a starting node, some internal and terminal nodes. The class labels are indicated by the terminal nodes. Bayes' theorem of probability has been utilized in Bayesian classifier, whose aim is to determine the probability that a given data set belongs to a class.

Several classifiers based on soft computing had been developed by various investigators. Multi-layer feed-forward neural network trained by back-propagation algorithm had been used as classifier [186] to classify inherent patterns in a data set. Rule-based system developed based on the fuzzy sets was also used as classifier [187], which was expressed in the form of IF-THEN rules.

Binary-coded GAs had been widely used as classifiers [188], in which each rule for classification was represented by a string of bits. The leftmost bits of the GA-string represent the attributes and the remaining bits indicate the class.

14.3.2 Predictive Models

We, human beings, have natural quest to gather input-output relationships of a process/system. To automate any process, its input-output relationships are to be known

in both forward and reverse directions as accurately as possible, on-line. In forward mapping, output(s) of a process (also known as response(s)) is/are expressed as the function(s) of input variables (also called the process parameters), whereas the process parameters are represented as the functions of the responses in reverse mapping.

Conventional regression analysis, which works based on the principle of least square error minimization, can be used to establish input-output relationships of a process involving multiple variables. It is conducted using the data collected through experiments carried out according to some designs of experiments, namely full-factorial design, fractional factorial design, central composite design, and others. Interested readers may refer to [189] for a detailed discussion on this analysis. The following steps are used to carry out the statistical regression analysis:

- Identification of input process parameters and output variables (that is, responses),
- Selection of an appropriate design of experiments,
- Data collection through experiments,
- Regression analysis to determine input-output relationships of the process,
- Adequacy checking of the model using analysis of variance,
- Studying the effects of inputs on the responses,
- Checking the performance of the developed model on some test cases.

It is important to mention that the problem of forward mapping can be solved efficiently using the statistical regression analysis. However, it may not be always possible to handle the problem of reverse mapping using the obtained regression equations, as the transformation matrix pertaining to input-output relationships may be singular. Moreover, this regression analysis is carried out response-wise. Therefore, it may not be able to capture the information of the process completely. It is also to be noted that regression analysis may be either linear or non-linear in nature. Let us assume that a non-linear response equation is derived for the entire range of the input process parameters. However, the degree of non-linearity may not be the same for the entire ranges of the inputs. To overcome this problem, input-output data may be clustered based on their similarity (refer to section 8.3), and once the clusters are obtained, regression analysis may be carried out cluster-wise to capture the input-output relationships of the process more accurately [190].

To overcome the above problem related to both the forward and reverse mappings of multi-inputs and multi-outputs process, either fuzzy logic [191, 5] or neural networks [4, 192]-based approaches had been developed. Fuzzy logic-based expert systems designed according to Mamdani approach (refer to section 8.2.1), Takagi and Sugeno's approach (refer to section 8.2.1) had been adopted for this purpose. On the other hand, multi-layer feed-forward neural networks (refer to section 10.2), radial basis function neural networks (refer to section 10.3) had been widely used to tackle the problems of forward and reverse mappings.

Visualization of input-output space of a process may help to extract useful information of a data set. As we cannot visualize more than $3 - D$ space, these higher dimensional data are mapped into either $2 - D$ or $3 - D$ for visualization using some techniques, namely Sammon's non-linear mapping, VISOR algorithm (refer to section 4.3.1), self-organizing map (SOM) (refer to section 10.4), and others.

The performances of these data miners are dependent on a number of parameters of the reasoning (that is, input-output modelling) and clustering tools. An intelligent data miner may be evolved, which will be able to select these parameters adaptively in order to establish the input-output relationships more accurately on-line in both the forward and reverse directions [193]. It may be treated as one of the key steps to be followed to automate a process. As soft computing can handle imprecision and uncertainty inherent to the data obtained through real experiments, it has the potential to tackle this problem of determining the input-output relationships of a process.

14.4 Summary

Out of various applications of soft computing, only two have been discussed in this chapter. The issues related to design and development of intelligent autonomous robots and intelligent data miners are explained in detail, and the roles of soft computing have been discussed in these fields of applications.

14.5 Exercise

1. Discuss the various steps to be followed to design and develop intelligent autonomous robots and make comments on the possibility of using the principle of soft computing in this development.
2. What do you mean by an intelligent data miner? How does the principle of soft computing help to develop it?

Bibliography

- [1] L.A. Zadeh, "The role of soft computing and fuzzy logic in conception, design and development of intelligent systems," *Proc. of International Workshop on Soft Computing in Industry*, Muroran, Japan, 27-28 April, 1996, pp. 136–137.
- [2] L.A. Zadeh, "Foreword," *Proc. of the Second International Conference on Fuzzy Logic and Neural Networks*, Iizuka, Japan, 1992, pp. XIII–XIV.
- [3] D.K. Pratihari, "Adaptive robot controller using soft computing," Edited book on *Focus on Robotics and Intelligent Systems Research*, NOVA Science Publishers, Inc. NY, USA, pp. 153–187, 2006.
- [4] M. Parappagoudar, D.K. Pratihari, G.L. Datta, "Forward and reverse mappings in green sand mould system using neural networks," *Applied Soft Computing*, vol. 8, no. 8, pp. 239–260, 2008.
- [5] K. Maji, D.K. Pratihari, "Forward and reverse mappings of electrical discharge machining process using adaptive network-based fuzzy inference system," *Expert Systems with Applications*, vol. 37, pp. 8566–8574, 2010.
- [6] S.J. Ovaska, Y. Dote, T. Furuhashi, A. Kamiya, H.F. VanLandingham, "Fusion of soft computing and hard computing techniques: a review of applications," *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, Tokyo, Japan, 12-15 October, 1999, pp. I-370–I-375.
- [7] A.V. SubbaRao, D.K. Pratihari, "Fuzzy logic-based expert system to predict the results of finite element analysis," *Knowledge-Based Systems*, vol. 20, pp. 37–50, 2007.
- [8] R. Rajendra, *Modeling and Simulations of Robotic Systems Using Soft Computing*, Ph.D. Thesis, IIT Kharagpur, India, 2012.
- [9] S.S. Rao, *Optimization: Theory and Applications*, Wiley Eastern Limited, 1978.
- [10] K. Deb, *Optimization for Engineering Design*, Prentice-Hall of India Private Limited, 1995.
- [11] J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, USA, 1975.

- [12] J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA:MIT Press, 1992.
- [13] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*, Stuttgart, Germany: Frommann-Holzboog, 1973.
- [14] L.J. Fogel, A.J. Owens, M.J. Walsh, *Artificial Intelligence Through Simulated Evolution*, John Wiley & Sons, New York, 1966.
- [15] D. Fogel, *Evolving Artificial Intelligence*, Ph.D. Thesis, University of California, San Diego, USA, 1992.
- [16] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
- [17] D.E. Goldberg, Real-coded genetic algorithms, virtual alphabets, and blocking," *Complex Systems*, vol. 5, no. 2, pp. 139–168, 1991.
- [18] L.J. Eshelman, J.D. Schaffer, "Real-coded genetic algorithms and interval schemata," In D. Whitley, (Ed.), *Foundation of Genetic Algorithms – II*, 1993, pp. 187–202.
- [19] K. Krishnakumar, "Micro-genetic algorithms for stationary and non-stationary function optimization," *Proc. of SPIE'89, Intelligent Control and Adaptive Systems*, Philadelphia, USA, vol. 1196, 1989, pp. 289–296.
- [20] D.E. Goldberg, B. Korb, K. Deb, "Messy genetic algorithms: Motivation, analysis and first results," *Complex Systems*, vol. 3, pp. 493–530, 1989.
- [21] D.E. Goldberg, K. Deb, "A comparison of selection schemes used in genetic algorithms," In G. Rawlins (Ed.), *Proc. of Foundation of Genetic Algorithms – I*, San Mateo: Morgan Kaufmann, 1991, pp. 69–93.
- [22] W.M. Spears, K.A. De Jong, "On the virtues of parameterized uniform crossover," In R.K. Belew, L.B. Booker (Eds.), *Proc. of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991, pp. 230–236.
- [23] D. Whitley, "The GENITOR algorithm and selective pressure - why rank-based allocation of reproduction trials is best," In D. Schaffer (Ed.), *Proc. of 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, 1989, pp. 116–121.
- [24] J. Baker, "Adaptive selection methods for genetic algorithms," In J. Grefenstette (Ed.), *Proc. of International Conference on Genetic Algorithms and Their Applications*, 1985, pp. 101–111.
- [25] A. Brindle, *Genetic Algorithms for Function Optimization*, Doctoral Dissertation and Technical Report TR 81-2, Edmonton: University of Alberta, Dept. of Computer Science, 1981.

- [26] K.A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, University of Michigan, Ann Arbor, USA.
- [27] W. Spears, K.A. De Jong, "An analysis of multi-point crossover," *Proc. of the Foundation of Genetic Algorithms Workshop*, Indiana, 1990, pp. 301–315.
- [28] G. Syswerda, "Uniform crossover in genetic algorithms," *Proc. of 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishing, 1989, pp. 2–9.
- [29] W.M. Spears, V. Anand, "A study of crossover operators in genetic programming," *Proc. of 6th International Symposium on Methodologies for Intelligent Systems*, 1991, pp. 409–418.
- [30] K.A. DeJong, W.M. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms," *Annals of Mathematics and Artificial Intelligence*, vol.5, pp. 1–26, 1992.
- [31] T.C. Fogarty, "Varying the probability of mutation in genetic algorithms," *Proc. of 3rd International Conference on Genetic Algorithms*, Morgan Kaufmann Publishing, 1989, pp. 104–109.
- [32] W.M. Spears, "Crossover or mutation ?," *Proc. of Foundations of Genetic Algorithms Workshop*, 1992, pp. 221–237.
- [33] D. Dasgupta, Z. Michalewicz (Eds.), *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, Berlin, 1997.
- [34] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons (Asia) Pte. Ltd., Singapore, 2001.
- [35] A.C.C. Carlos, "A survey of constraint handling techniques used with evolutionary algorithms," Technical Report Lania-RI-99-04, Laboratorio Nacional de Informatica Avanzada, Xalapa, Veracruz, Mexico, 1999.
- [36] A. Homaifar, S.H.Y. Lai, X. Qi, "Constrained optimization via genetic algorithms," *Simulation*, vol. 62, no. 4, pp. 242–254, 1994.
- [37] J. Joines, C. Houck, "On the use of non-stationary penalty functions to solve non-linear constrained optimization problems with GAs," In David Fogel (ed.), *Proc. of the IEEE Conference on Evolutionary Computation*, Orlando, Florida, USA, 1994, pp. 579–584.
- [38] J.C. Bean, A.B. Hadj-Alouane, "A dual genetic algorithm for bounded integer programs," *Technical Report TR92-53*, Department of Industrial and Operations Engineering, The University of Michigan, 1992.
- [39] A.B. Hadj-Alouane, J.C. Bean, "A genetic algorithm for the multiple-choice integer program," *Technical Report TR92-50*, Department of Industrial and Operations Engineering, The University of Michigan, 1992.

- [40] A. Wright, "Genetic algorithms for real parameter optimization," *Proc. of Foundations of Genetic Algorithms 1 (FOGA-1)*, 1991, pp. 205–218.
- [41] K. Deb, R.B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, no. 2, pp. 115–148, 1995.
- [42] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, Germany, 1992.
- [43] K. Deb, M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, vol. 26, no. 4, pp. 30–45, 1996.
- [44] N. Hayashida, H. Takagi, "Acceleration of EC convergence with landscape visualization and human intervention," *Applied Soft Computing*, vol. 1, no. 4, pp. 245–256, 2002.
- [45] D.K. Pratihari, N. Hayashida, H. Takagi, "Comparison of mapping methods to visualize the EC landscape," *Proc. of 5-th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Allied Technology – KES'2001*, Nara, Japan, 2001, pp. 223–227.
- [46] W. Siedlecki, K. Seidlecka, J. Sklansky, "An overview of mapping techniques for exploratory data analysis," *Pattern Recognition*, vol. 21, no. 5, pp. 411–429, 1988.
- [47] A. König, "Dimensionality reduction techniques for interactive visualization, exploratory data analysis and classification," In N.R. Pal (ed.), *Feature Analysis, Clustering and Classification by Soft Computing*, FLSI Soft Computing Series, vol. 3, pp. 1–37, 2000.
- [48] J.W. Sammon, "A nonlinear mapping for data structure analysis," *IEEE Trans. on Computers*, vol. C-18(5), pp. 401–409, 1969.
- [49] A. König, O. Bulmahn, M. Glessner, "Systematic methods for multivariate data visualization and numerical assessment of class separability and overlap in automated visual industrial quality control," *Proc. of 5th British Machine Vision Conference.*, vol. 1, Sept., 1994, pp. 195–204.
- [50] T. Kohonen, *Self-Organizing Maps*, Springer-Verlag, Heidelberg, Germany, 1995.
- [51] I.T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, Heidelberg, Germany, 1986.
- [52] P. Dutta, D.K. Pratihari, "Some studies on mapping method," *International Journal of Business Intelligence and Data Mining*, vol. 1, no. 3, pp. 347–370, 2006.
- [53] L. Yu, H. Liu, "Feature selection for high-dimensional data: a fast correlation-based filter solution," *Proc. of 20-th International Conference on Machine Learning*, pp. 856–863, 2003.

- [54] R. Ruiz, J.C. Riquelme, J.S. Aguilar-Ruiz, "Incremental wrapper-based gene selection from microarray data for cancer classification," *Pattern Recognition*, vol. 39, pp. 2383–2392, 2006.
- [55] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, C. Whitley, "A comparison of genetic scheduling operators," *Proc. of International Conference on Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, 1991, pp. 69–76.
- [56] D. Whitley, T. Starkweather, D. Shaner, "The travelling salesman and sequence scheduling: quality solutions using genetic edge recombination," In *Handbook of Genetic Algorithms*, L. Davis (ed.), Van Nostrand Reinhold, pp. 350–372, 1991.
- [57] L. Davis, "Applying adaptive algorithms to epistatic domains," *Proc. of International Joint Conference on Artificial Intelligence*, 1985, pp. 161–163.
- [58] G. Syswerda, "Schedule optimization using genetic algorithms," In *Handbook of Genetic Algorithms*, I. Davis (Ed.), Van Nostrand Reinhold, New York, 1991, pp. 332–349.
- [59] J.M. Oliver, D.J. Smith, J.R.C. Holland, "A study of permutation crossover operators on the travelling salesman problem," *Proc. of 2nd International Conference on Genetic Algorithms and their Applications*, 1987, pp. 224–230.
- [60] D. Goldberg, R. Lingle, "Alleles, loci and the travelling salesman problem," *Proc. of International Conference on Genetic Algorithms and their Applications*, 1985, pp. 154–159.
- [61] R. Storn, K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.
- [62] R.G. Reynolds, "An introduction to cultural algorithms," *Proc. of the 3-rd Annual Conference on Evolutionary Programming*, World Scientific Publishing, 1994, pp. 131–139.
- [63] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.
- [64] J. Kennedy, R. Eberhart, "Particle swarm optimization," *Proc. of IEEE International Conference on Neural Networks*, Perth, Australia, 1995, pp. 1942–1948.
- [65] J.O. Kephart, "A biologically inspired immune system for computers," *Proc. of Artificial Life IV: The Fourth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, 1994, pp. 130–139.
- [66] M. Dorigo, *Optimization, Learning and Natural Algorithms*, Ph.D. Thesis, Politecnico di Milano, Italie, 1992.

- [67] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," *Technical Report-TR06*, Computer Engineering Department, Erciyes University, 2005.
- [68] E. Atashpaz-Gargari, C. Lucas, "Imperialist Competitive Algorithm: An algorithm for optimization inspired by imperialistic competition," *Proc. of IEEE Congress on Evolutionary Computation*, 2007, pp. 4661–4666.
- [69] A. Kaveh, S. Talatahari, "A novel heuristic optimization method: charged system search," *Acta Mechanica*, vol. 213, nos. 3-4, pp. 267–289, 2010.
- [70] S. Kumar, D.K. Chaturvedi, "Tuning of particle swarm optimization parameter using fuzzy logic," *Proc. of International Conference on Communication Systems and Network Technologies (CSNT)*, 2011, pp. 174–179.
- [71] L. Zadeh, "Optimality and non-scalar-valued performance criteria," *IEEE Trans. on Automatic Control*, vol. 8, no. 1, pp. 59–60, 1963.
- [72] K. Choudhury, D.K. Pratihar, D.K. Pal, "Multi-objective optimization in turning using a genetic algorithm," *IE(I) Journal, Production Engineering division*, vol. 82, pp. 37–44, 2002.
- [73] J.D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," *Proc. of ICGA*, 1985, pp. 93–100.
- [74] A. Osyczka, S. Kundu, "A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm," *Structural Optimization*, vol. 10, no. 2, pp. 94–99, 1995.
- [75] N. Srinivas, K. Deb, "Multiobjective function optimization using nondominated sorting genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1995.
- [76] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [77] C.M. Fonseca, P.J. Fleming, "Genetic algorithms for multi-objective optimization: Formulation, discussion and generalization," *Proc. of the Fifth ICGA*, S. Forrest, Ed. San Mateo, CA: Morgan Kauffman, 1993, pp. 416–423.
- [78] J. Horn, N. Nafpliotis, D.E. Goldberg, "A niched Pareto genetic algorithm for multi-objective optimization," *Proc. of the First IEEE Conference on Evolutionary Computation*, Z. Michalewicz (Ed.) Piscataway, NJ: IEEE Press, 1994, pp. 82–87.
- [79] J. Knowles, D. Corne, "The Pareto-archived evolution strategy: A new baseline algorithm for multi-objective optimization," *Proc. of the 1999 Congress on Evolutionary Computation*, Piscataway, NJ: IEEE Press, 1999, pp. 98–105.

- [80] E. Zitzler, *Evolutionary Algorithms for Multi-objective Optimization: Methods and Applications*, Ph.D. Thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1999.
- [81] L.A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338–353, 1965.
- [82] M. Black, "Vagueness: an exercise in logical analysis," *Philosophy of Science*, vol. 4, no. 4, pp. 427–455, 1937.
- [83] T.J. Ross, *Fuzzy Logic with Engineering Applications*, McGraw-Hill Book Co., Singapore, 1997.
- [84] A. DeLuca, S. Termini, "A definition of non-probabilistic entropy in the setting of fuzzy set theory," *Information and Control*, vol. 20, pp. 301–312, 1971.
- [85] R. Verma, B.D. Sharma, "A measure of inaccuracy between two fuzzy sets," *Cybernetics and Information Technologies (Bulgarian Academy of Sciences)*, vol. 11, no. 2, pp. 13–23, 2011.
- [86] E.H. Mamdani, S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, pp. 1–13, 1975.
- [87] L.X. Wang, Analysis and design of hierarchical fuzzy systems, *IEEE Trans. on Fuzzy Systems*, vol. 7, no. 5, pp. 617–624, 1999.
- [88] M.L. Lee, H.Y. Chung, F.M. Yu, Modeling of hierarchical fuzzy systems, *Fuzzy Sets and Systems*, vol. 138, pp. 343–361, 2003.
- [89] D.K. Pratihari, K. Deb, A. Ghosh, "A genetic-fuzzy approach for mobile robot navigation among moving obstacles," *International Journal of Approximate Reasoning*, vol. 20, pp. 145–172, 1999.
- [90] T. Takagi, M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-15, pp. 116–132, 1985.
- [91] G.V.S. Raju, I. Zhou, R.A. Kisner, "Hierarchical fuzzy control," *International Journal of Control*, vol. 54, no. 5, pp. 1201–1216, 1991.
- [92] L.M. Liebrock, "Empirical sensitivity analysis for computational procedures," *Proc. of TAPIA-05*, Albuquerque, New Mexico, USA, October 19–22, 2005, pp. 32–35.
- [93] J.C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32–57, 1973.
- [94] J.C. Bezdek, "Fuzzy Mathematics in Pattern Classification," *Ph.D. Thesis*, Applied Mathematics Center, Cornell University, Ithaca, 1973.

- [95] J. Keller, M.R. Gary, J.A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-15, no. 4, pp. 580–585, 1985.
- [96] S.L. Chiu, "Fuzzy model identification based on cluster estimation," *Journal of Intelligent Fuzzy Systems*, vol. 2, pp. 267–278, 1994.
- [97] J. Yao, M. Dash, S.T. Tan, H. Liu, "Entropy-based fuzzy clustering and fuzzy modeling," *Fuzzy Sets and Systems*, vol. 113, pp. 381–388, 2000.
- [98] S. Chattopadhyay, D.K. Pratihari, S.C. De Sarkar, "Performance studies of some similarity-based fuzzy clustering algorithms," *International Journal of Performativity Engineering*, vol. 2, no. 2, pp. 191–200, 2006.
- [99] R. Ng, J. Han, "Very large data bases," *Proc. of 20-th International Conference on Very Large Data Bases*, Berkeley, CA, 1994, pp. 144–155.
- [100] M. Ester, H.P. Kriegel, J. Sander, X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Proc. of the 2-nd International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, 1996, pp. 226–231.
- [101] W.S. McCulloch, W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
- [102] K. Hornik, "Approximation capabilities of multilayer feed-forward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [103] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning representations of back-propagation errors," *Nature (London)*, vol. 323, pp. 533–536, 1986.
- [104] Website: http://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm
- [105] D.S. Broomhead, D. Lowe, "Multi-variable functional interpolation and adaptive networks," *Complex Systems*, vol. 11, pp. 321–355, 1988.
- [106] J.E. Moody, C. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computation*, vol. 2, no. 1, pp. 281–294, 1989.
- [107] S. Chen, C.F. Cowan, P.M. Grant, "Orthogonal least squares algorithm for learning radial basis function networks," *IEEE Trans. on Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.
- [108] S. Chen, E.S. Chng, K. Alkadhimi, "Regularized orthogonal least squares algorithm for constructing radial basis function networks," *International Journal of Control*, vol. 64, no. 5, pp. 829–837, 1996.
- [109] A. Leonardis, H. Bischof, "An efficient MDL-based construction of RBF networks," *Neural Networks*, vol. 11, pp. 963–973, 1998.

- [110] S.A. Billings, G.L. Zheng, "Radial basis function network configuration using genetic algorithms," *Neural Networks*, vol. 8, no. 6, pp. 877–890, 1995.
- [111] A.F. Sheta, K. de Jong, "Time-series forecasting using GA-tuned radial basis functions," *Information Sciences*, vol. 133, nos. 3-4, pp. 221–228, 2001.
- [112] R. Hecht-Nielsen, "Counterpropagation networks," *Applied Optics*, vol. 26, no. 23, pp. 4979–4984, 1987.
- [113] J.L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1999.
- [114] M.I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," *Proc. of the 8-th Annual Conference of the Cognitive Science Society*, Englewood Cliffs, NJ: Lawrence Erlbaum Associates, 1986, pp. 531–546.
- [115] D.T. Pham, D. Karaboga, "Training Elman and Jordan networks for system identification using genetic algorithms," *Artificial Intelligence in Engineering*, vol. 13, pp. 107–117, 1999.
- [116] M.A. Lee, H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," *Proc. of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, 1993, pp. 76–83.
- [117] H.Y. Xu, G. Vukovich, "A fuzzy genetic algorithm with effective search and optimization," *Proc. of International Joint Conference on Neural Networks*, 1993, pp. 2967–2970.
- [118] H.Y. Xu, G. Vukovich, Y. Ichikawa, Y. Ishii, "Fuzzy evolutionary algorithms and automatic robot trajectory generation," *Proc. of the First IEEE Conference on Evolutionary Computation*, 1994, pp. 595–600.
- [119] F. Herrera, E. Herrera-Viedma, M. Lozano, J.L. Verdegay, "Fuzzy tools to improve genetic algorithms," *Proc. of the Second European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1994, pp. 1532–1539.
- [120] D.K. Pratihar, *Path and Gait Generation of Legged Robots Using GA-Fuzzy Approach*, Ph.D. Thesis, IIT Kanpur, India, 2000.
- [121] Y.H. Liu, S. Arimoto, "Proposal of tangent graph and extended tangent graph for path planning of mobile robots," *Proc. of IEEE International Conference on Robotics and Automation*, 1991, pp. 312–317.
- [122] Y.H. Liu, S. Arimoto, "Path planning using a tangent graph for mobile robots among polynomial and curved obstacles," *International Journal of Robotics Research*, vol. 11, no. 4, pp. 376–382, 1992.

- [123] Y.H. Liu, S. Arimoto, "Finding the shortest path of a disc among polygonal obstacles using a radius-independent graph," *IEEE Trans. on Robotics and Automation*, vol. 11, no. 5, pp. 682–691, 1995.
- [124] O. Cordon, F. Gomide, F. Herrera, F. Hoffmann, L. Magdalena, "Ten years of genetic fuzzy systems: current framework and new trends," *Fuzzy Sets and Systems*, vol. 141, pp. 5–31, 2004.
- [125] C. Karr, "Genetic algorithms for fuzzy controllers," *AI Expert*, pp. 26–33, 1991.
- [126] F. Herrera, M. Lozano, J.L. Verdegay, "Tuning fuzzy logic controllers by genetic algorithms," *International Journal of Approximate Reasoning*, vol. 12, pp. 293–315, 1995.
- [127] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," *Proc. of Fourth International Conference on Genetic Algorithms (ICGA '91)*, San Diego, USA, Morgan Kaufmann, Los Altos, CA, 1991, pp. 509–513.
- [128] B. Carse, T.C. Fogarty, A. Munro, "Evolving fuzzy rule based controllers using genetic algorithms," *Fuzzy Sets and Systems*, vol. 80, pp. 273–294, 1996.
- [129] F. Hoffmann, G. Pfister, "Evolutionary design of a fuzzy knowledge base for a mobile robot," *International Journal of Approximate Reasoning*, vol. 17, no. 4, pp. 447–469, 1997.
- [130] L. Magdalena, F. Monasterio, "A fuzzy logic controller with learning through the evolution of its knowledge base," *International Journal of Approximate Reasoning*, vol. 16, nos. 3-4, pp. 335–358, 1997.
- [131] J.R. Velasco, "Genetic-based on-line learning for fuzzy process control," *International Journal of Intelligent Systems*, vol. 13, nos. 10-11, pp. 891–903, 1998.
- [132] H. Ishibuchi, T. Nakashima, T. Murata, "Performance evaluation of fuzzy classifier systems for multi-dimensional pattern classification problems," *IEEE Trans. on Systems, Man and Cybernetics, Part B*, vol. 29, pp. 601–618, 1999.
- [133] A. González, F. Herrera, "Multi-stage genetic fuzzy systems based on the iterative rule learning approach," *Mathware & Soft Computing*, vol. 4, pp. 233–249, 1997.
- [134] O. Cordón, M.J. del Jesus, F. Herrera, M. Lozano, "MOGUL: A methodology to obtain genetic fuzzy rule-based systems under the iterative rule learning approach," *International Journal of Intelligent Systems*, vol. 14, no. 11, pp. 1123–1153, 1999.
- [135] A. González, R. Pérez, "SLAVE: a genetic learning system based on an iterative approach," *IEEE Trans. on Fuzzy Systems*, vol. 7, no. 2, pp. 176–191, 1999.

- [136] M.A. Lee, H. Takagi, "Embedding a priori knowledge into an integrated fuzzy system design method based on genetic algorithms," *Proc. of Fifth International Fuzzy Systems Association World Congress (IFSA '93)*, Seoul, 1993, pp. 1293–1296.
- [137] M.G. Cooper, J.J. Vidal, "Genetic design of fuzzy logic controllers", *Proc. of Second International Conference on Fuzzy Theory and Technology (FTT'93)*, Durham, 1993.
- [138] K.C. Ng, Y. Lee, "Design of sophisticated fuzzy logic controllers using genetic algorithms," *Proc. of Third IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'94)*, Orlando, USA, 1994, pp. 1708–1712.
- [139] J. Liska, S.S. Melsheimer, "Complete design of fuzzy logic systems using genetic algorithms," *Proc. of the Third IEEE International Conference on Fuzzy Systems*, IEEE, Piscataway NJ, 1994, pp. 1377–1382.
- [140] N. Eiji Nawa, T. Hashiyama, T. Furuhashi, Y. Uchikawa, "Fuzzy Logic Controllers Generated by Pseudo-Bacterial Genetic Algorithm with Adaptive Operator," *Proc. of IEEE International Conference on Neural Networks - ICNN'97*, Houston, USA, 1997, pp. 2408–2413.
- [141] N.B. Hui, D.K. Pratihari, "Automatic design of fuzzy logic controller using a genetic algorithm for collision-free, time-optimal navigation of a car-like robot," *International Journal of Hybrid Intelligent Systems*, vol. 2, pp. 161–187, 2005.
- [142] D. Whitley, "Genetic algorithms and neural networks," Winter, Periaux, Galan and Cuesta (Eds.), in *Genetic Algorithms in Engineering and Computer Science*, John Wiley & Sons Ltd., pp. 203–216, 1995.
- [143] S.A. Harp, T. Samad, A. Guha, "Towards the genetic synthesis of neural networks," *Proc. of Third International Conference on Genetic Algorithms*, 1989, pp. 360–369.
- [144] J.D. Schaffer, R.A. Caruana, L.J. Eshelman, "Using genetic search to exploit the emergent behavior of neural networks," In S. Forest (Ed.), *Emergent Computation*, pp. 244–248, 1990.
- [145] T. Smith, *Adding Vision to Khepera: An Autonomous Robot Footballer*, Masters' thesis, School of Cognitive and Computing Sciences, University of Sussex, UK.
- [146] F. Mondada, E. Franzi, P. Ienne, "Mobile robot miniaturization: A tool for investigation in control algorithms," *Proc. of 3rd International Symposium on Experimental Robotics*, Kyoto, Japan, 1993, pp. 501–513.
- [147] D.K. Pratihari, "Evolutionary robotics – a review," *Sadhana*, vol. 28, no. 6, pp. 999–1009, 2003.
- [148] N.B. Hui, D.K. Pratihari, "Neural network-based approaches vs. potential field approach for solving navigation problems of a car-like robot," *Machine Intelligence and Robotic Control*, vol. 6, no. 2, pp. 39–59, 2004.

- [149] P. Dutta, D.K. Pratihari, "Modeling of TIG welding process using conventional regression analysis and neural network-based approaches," *Journal of Materials Processing Technology*, vol. 184, nos. 1-3, pp. 56-68, 2007.
- [150] G. Miller, P. Todd, S. Hedge, "Designing neural networks using genetic algorithms," *Proc. of 3rd International Conference on Genetic Algorithm*, Morgan Kaufmann, 1989, pp. 379-384.
- [151] D. Whitley, T. Starkweather, C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Computing*, vol. 14, pp. 347-361, 1990.
- [152] E.J. Chang, R.P. Lippmann, "Using genetic algorithms to improve pattern classification performance," In R.P. Lippmann, J.E. Moody, D.S. Touretsky (Eds.), *Advances in Neural Information Processing*, 3, San Mateo, CA: Morgan Kaufmann, pp. 797-803, 1991.
- [153] F.Z. Brill, D.E. Brown, W.N. Martin, "Fast genetic selection of features for neural network classifiers," *IEEE Trans. on Neural Networks*, vol. 3, no. 2, pp. 324-328, 1992.
- [154] R.C. Eberhart, R.W. Dobbins, "Designing neural network explanation facilities using genetic algorithms," *Proc. of IEEE International Joint Conference on Neural Networks*, Singapore, 1991, pp. 1758-1763.
- [155] J.M. Keller, R.R. Yager, H. Tahani, "Neural network implementation of fuzzy logic," *Fuzzy Sets and Systems*, vol. 45, no. 1, pp. 1-12, 1992.
- [156] H. Takagi, N. Suzuki, T. Koda, Y. Kojima, "Neural networks designed on approximate reasoning architecture and their applications," *IEEE Trans. Neural Networks*, vol. 3, pp. 752-760, 1992.
- [157] H.R. Berenji, P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724-740, 1992.
- [158] H. Takagi, I. Hayashi, "NN-driven fuzzy reasoning," *International Journal of Approximate Reasoning*, vol. 5, no. 3, pp. 191-212, 1991.
- [159] H. Ishibuchi, H. Tanaka, H. Okada, "Interpolation of fuzzy if-then rules by neural networks," *International Journal of Approximate Reasoning*, vol. 10, no. 1, pp. 3-27, 1994.
- [160] J.R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. on Systems, Man and Cybernetics, Part B*, vol. 23, no. 3, pp. 665-685, 1993.
- [161] S.C. Lee, E.T. Lee, "Fuzzy neural networks," *Mathematical Biosciences*, vol. 23, pp. 151-177, 1975.

- [162] J.J. Buckley, Y. Hayashi, "Fuzzy neural networks: A survey," *Fuzzy Sets and Systems*, vol. 66, pp. 1–13, 1994.
- [163] N.B. Hui, V. Mahendar, D.K. Pratihari, "Time-optimal, collision-free navigation of a car-like mobile robot using a neuro-fuzzy approach," *Fuzzy Sets and Systems*, vol. 157, no. 16, pp. 2171–2204, 2006.
- [164] J. Albus, "Outline for a theory of intelligence," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 21, no. 3, pp. 473–509, 1991.
- [165] K.S. Fu, R.C. Gonzalez, C.S.G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill Book Co., Singapore, 1987.
- [166] W.K. Choi, S.J. Kim, H.T. Jeon, "Multiple sensor fusion and motion control of snake robot based on soft computing," in *Bioinspiration and Robotics: walking and Climbing Robots* M.K. Habib (Ed.), I-Tech Education and Publishing, 2007.
- [167] N.B. Hui, D.K. Pratihari, "Camera calibration using a genetic algorithm," *Engineering Optimization*, vol. 40, no. 12, pp. 1151–1169, 2008.
- [168] C.S. Lee, Y.H. Kuo, "Adaptive fuzzy filter and its application to image enhancement," in *Fuzzy Techniques in Image Processing*, E.E. Kerre, M. Nachtegaal (Eds.), Physica-Verlag, Heidelberg, New York, pp. 172–193, 2000.
- [169] K. Arakawa, "Fuzzy rule-based image processing with optimization," in *Fuzzy Techniques in Image Processing* E.E. Kerre, M. Nachtegaal (Eds.), Physica-Verlag, Heidelberg, New York, pp. 222–247, 2000.
- [170] D.K. Pratihari, "Algorithmic and soft computing approaches to robot motion planning," *Machine Intelligence and Robotic Control*, vol. 5, no. 1, pp. 1–16, 2003.
- [171] J.C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Dordrecht, 1991.
- [172] T. Fraichard, P. Garnier, "Fuzzy control to drive car-like vehicles," *Robotics and Autonomous Systems*, vol. 34, pp. 1–22, 2001.
- [173] M. Akbarzadeh, K. Kumbla, E. Tunstel, M. Jamshidi, "Soft computing for autonomous robotic systems," *Computers and Electrical Engineering*, vol. 26, pp. 5–32, 2000.
- [174] S.X. Yang, M. Meng, "An efficient neural network approach to dynamic robot motion planning," *Neural Networks*, vol. 13, no. 2, pp. 143–148, 2000.
- [175] D. Gu, H. Hu, "Neural predictive control for a car-like mobile robot," *Robotics and Autonomous Systems*, vol. 39, pp. 73–86, 2002.
- [176] F. Mondada, D. Floreano, "Evolution of neural control structures: some experiments on mobile robots," *Robotics and Autonomous Systems*, vol. 16, pp. 183–195, 1995.

- [177] S. Nolfi, D. Parsi, "Learning to adapt to changing environments in evolving neural networks," *Adaptive Behavior*, vol. 5, no. 1, pp. 75–98, 1997.
- [178] A. Mackworth, "On seeing robots," in *Computer Vision: Systems, Theory and Applications*, World Scientific Press, Singapore, pp. 1-13, 1993.
- [179] D.K. Pratihari, W. Bibel, "Time-optimal, collision-free path planning for multiple cooperating robots using a genetic-fuzzy system," *Machine Intelligence and Robotic Control*, vol. 5, no. 2, pp. 45–58, 2003.
- [180] S. Taniguchi, Y. Dote, S.J. Ovaska, "Control of intelligent agent systems (robots) using extended soft computing," *Proc. of IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 2000, pp. 3568–3572.
- [181] R. Rajendra, D.K. Pratihari, "Particle swarm optimization vs. genetic algorithm to develop integrated scheme for obtaining optimal mechanical structure and adaptive controller of a robot," *Intelligent Control and Automation*, vol. 2, pp. 430–449, 2011.
- [182] N.B. Hui, *Design and Development of Adaptive Motion Planners for Wheeled Robots*, Ph.D. Thesis, IIT Kharagpur, India, 2007.
- [183] C.T. Lin, C.P. Jou, C.J. Lin, "GA-based reinforcement learning for neural networks," *International Journal of Systems Science*, vol. 29, no. 3, pp. 233–247, 1998.
- [184] J.R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.
- [185] P. Domingos, M. Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss," *Machine Learning*, vol. 29, pp. 103–130, 1997.
- [186] R.C. Lacher, P.K. Coats, S.C. Sharma, L.F. Fant, "A neural network tool for classifying the financial health of a firm," *European Journal of Operational Research*, vol. 85, no. 1, pp. 53–65, 1995.
- [187] R.L. Lawrence, A. Wright, "Rule-based classification systems using classification and regression tree (CART) analysis," *Photogrammetric Engineering and Remote Sensing*, vol. 67, pp. 1137–1142, 2001.
- [188] S.K. Pal, S. Bandyopadhyay, C.A. Murthy, "Genetic classifiers for remotely sensed images: comparison with standard methods," *International Journal of Remote Sensing*, vol. 22, no. 13, pp. 2545–2569, 2001.
- [189] D.C. Montgomery, *Design and Analysis of Experiments*, John Wiley, New York, 1997.
- [190] J.P. Ganjigatti, D.K. Pratihari, A. RoyChoudhury, "Global vs. cluster-wise regression analysis for prediction of bead geometry in MIG welding process," *Journal of Materials Processing Technology*, vol. 189, nos. 1-3, pp. 352–366, 2007.

- [191] J.P. Ganjigatti, D.K. Pratihar, "Forward and reverse modeling in MIG welding process using fuzzy logic-based approaches," *Journal of Intelligent and Fuzzy Systems*, vol. 19, no. 2, pp. 115–130, 2008.
- [192] M.V.V. Amarnath, D.K. Pratihar, "Forward and reverse mappings of TIG welding process using radial basis function neural networks (RBFNNs)," *IMech. E., Part B, Journal of Engineering Manufacture*, vol. 223, pp. 1575–1590, 2009.
- [193] D.K. Pratihar, "Some studies on data mining," in *Handbook on Reasoning-based Intelligent Systems*, K. Nakamatsu, L.C. Jain (Eds.), World Scientific Publishers, pp. 61–80, 2012.

Index

A

- Activation function, 158
- Adaptation, 216
- Adaptive penalty, 50,53
- ANFIS, 238,241
- Antecedents, 1,127
- Ant colony optimization, 83
- Artificial bee colony, 83
- Artificial neuron, 158
- Axon, 157

B

- Back-propagation, 170,174,231
- Batch mode of training, 164,172,184
- Behavior constraint, 9
- Binary code, 33
- Biological neuron, 157
- Bound points, 13
- Building-Block Hypothesis, 46

C

- Cardinality, 102
- Cell body, 157
- Charged system search, 83
- Classical set, 101
- Classification tools, 252
- Clustering, 142
- Competition, 186
- Complement
 - absolute, 103
 - relative, 103
- Concentration, 118
- Consequents, 1,127
- Constrained optimization, 10,11,15,28
- Continuous fuzzy set, 108
- Convex fuzzy set, 108
- Cooperation, 186
- Counter-propagation NN, 188
- Crisp clustering, 142
- Crisp set, 101,103,104

Crossover

- blend, 59,60
- contracting, 61
- cycle, 76
- expanding, 61
- linear, 59,60
- multi-point, 33,39
- order, 75,76
- partially mapped, 78
- position-based, 77
- simulated binary, 59,61
- single point, 33,38
- stationary, 61
- two-point, 33,39
- uniform, 33,39

- Cultural algorithm, 83

D

- Data analysis, 249,252
- Data base, 125
- Data mining, 142
- Decision variable, 9,28
- Defining length, 44,45
- Defuzzification, 128-130
- Delta rule, 171
- Dendrites, 157
- Design variable, 9
- Differential evolution, 83
- Dilation, 118
- Direct search, 21
- Distance-based Pareto-GA, 94
- Discrete fuzzy set, 107
- Distance preserving technique, 66
- Duality principle, 13,14
- Dynamic penalty, 50,53

E

- Edge recombination, 73
- Elitism, 37
- Elman network, 194

Empty set, 102
 Entropy, 147, 151
 Epoch, 164
 Equal set, 102
 Evolution, 216
 Evolutionary programming, 31,42,83
 Evolutionary robotics, 216,251
 Evolution strategies, 31,42,83
 Exhaustive search method, 7,16-17
 Exploitation, 36,46,52
 Exploration, 36,46,52
 Exploratory power, 40

F

Free points, 13
 Functional constraints, 9,28
 Fuzzy clustering
 entropy-based, 147,151
 FCM, 142,151
 Fuzzification, 127
 Fuzzy logic controller
 Hierarchical, 139
 Mamdani approach, 127,208,226
 Takagi and Sugeno's approach, 127,137,238
 Fuzzy-Genetic Algorithm, 199
 Fuzzy Neural Network, 199,225
 Fuzzy set, 101,106-108,112-122

G

Genetic algorithm, 31,42,83
 Genetic-Fuzzy System, 199,202,205
 Genetic-Neural System, 199,215,217
 Genetic-Neuro-Fuzzy System, 199,232,242
 Genetic Programming, 31,42,83
 Generalized Delta rule, 173
 Geometric constraint, 9,28
 Goal programming, 93
 Gradient-based method, 15,23,27
 Gray code, 47,52

H

Hamming cliff, 46,52,59
 Hard computing, 1,2,5
 Hierarchical FLC, 139
 Hybrid computing, 1,4,5,6

I

Imperialist competitive algorithm, 83
 Importance factor, 212
 Incremental mode of training, 164,170,182
 Inflection point, 7,8

Intelligent autonomous robots, 249
 Integer programming, 11,15,27,51
 Intersection of sets, 103
 Iterative rule learning, 204

J

Jordan network, 195

K

Khepera, 216
 Knowledge base, 125

L

Learning, 164
 Learning rate, 171,182
 Linear
 filter, 159
 mapping rule, 34
 optimization, 10,28
 Linguistic fuzzy modeling, 126

M

Mapping, 66
 Mean squared deviation, 172,231
 Medoids, 151
 Membership, 109-111
 Michigan approach, 204
 Micro-GA, 31,59,65
 Mixed-integer programming, 11,12,28,51
 Momentum constant, 173
 Multi-objective optimization, 91
 Mutation, 33,40,63

N

Neural Network
 dynamic, 163
 feed-forward, 167
 Kohonen network, 185
 modular, 167
 radial basis function, 167,178
 recurrent, 167,194
 self-organizing map, 66,167,185
 static, 163
 stochastic, 167
 Neurons, 157
 Neuro-Fuzzy System, 225,226,231,232,238
 Niched Pareto GA, 98
 Non-dominated sorting GA, 95
 Non-linear optimization, 10,15,28
 Null set, 102

O

- Objective function, 9,28
- Off-line training, 164
- On-line training, 164
- Optimization, 7,12,28
- Optimum, 7
- Order, 44
- Overtraining, 173

P

- Parallel computing, 27
- Pareto-Archived ES, 98
- Pareto-optimal front, 91
- Particle swarm optimization, 83,87
- Penalty function approach, 49
- Perceptron neuron, 159
- Pittsburgh approach, 204
- Polynomial mutation, 64
- Population diversity, 36,46,52
- Power set, 102
- Pre-assigned parameter, 9
- Precise fuzzy modeling, 126
- Predictive models, 252
- Premature convergence, 36
- Proper subset, 102
- Proper superset, 102
- Proportionate selection, 34

R

- Random mutation, 64
- Random walk method, 21
- Real-coded GA, 31,59
- Real-valued programming, 11
- Reproduction
 - ranking selection, 32,36
 - roulette-wheel selection, 32,34
 - tournament selection, 32,37
- Rule base, 125,131

S

- Saddle point, 7
- Sammon's nonlinear mapping, 66
- Scheduling GA, 72
- Schema, 44
- Selection pressure, 36,46,52
- Side constraint, 9
- Simulated annealing, 83,84

- Soft computing, 1-6,249-252,254
- Soma, 157
- Spread factor, 61
- Static penalty, 50,53
- Steepest descent method, 7,15,23
- Strength-Pareto evolutionary algorithm, 98
- Subset, 102
- Superset, 102
- Supervised learning, 164,165
- Synapse, 157

T

- Takagi and Sugeno's approach, 127,137,238
- Transfer function
 - hard-limit, 159
 - linear, 159
 - log-sigmoid, 159
 - tan-sigmoid, 161
- TSP problem
 - asymmetrical, 73
 - symmetrical, 72

U

- Unconstrained optimization, 10,15,28
- Union of sets, 104
- Universal approximation theorem, 167
- Universal set, 101
- Universe of discourse, 101
- Un-supervised learning, 163,185
- Updating, 187

V

- Vector evaluated GA, 93
- Visor algorithm, 66,68
- Visualization, 66
- Visualized Interactive GA, 59,65,71