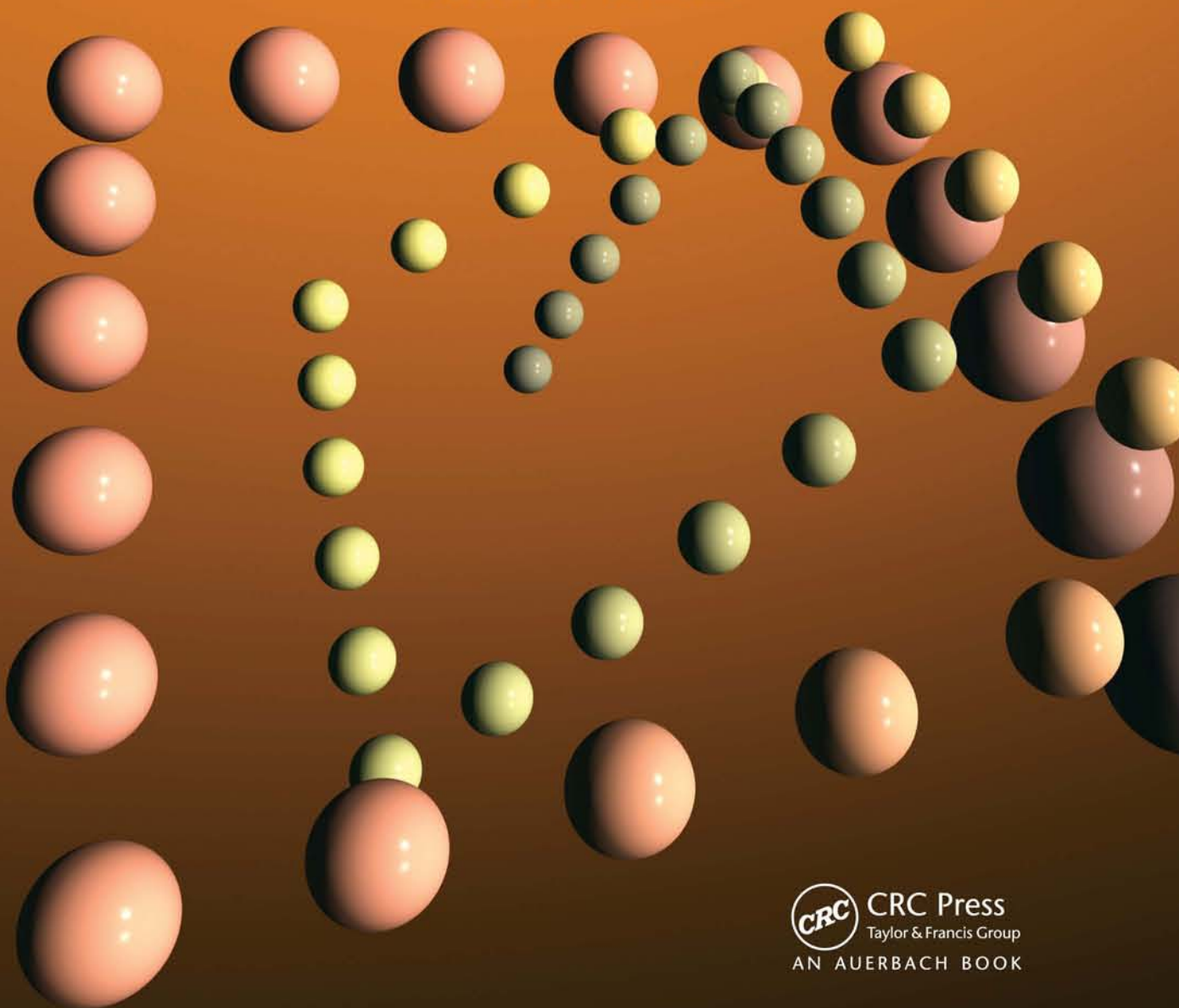
Real Life Applications of Soft Computing

Anupam Shukla
Ritu Tiwari
Rahul Kala



CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

Real Life Applications of Soft Computing

OTHER AUERBACH PUBLICATIONS

Cloud Computing:
Technologies and Strategies
of the Ubiquitous Data Center
Curtis Franklin Jr. and Brian J.S. Chee
ISBN 978-1-4398-0612-8

Cognitive Radio Networks:
Architectures, Protocols, and Standards
Edited by Yan Zhang, Jun Zheng,
and Hsiao-Hwa Chen
ISBN 978-1-4200-7775-9

Essential Project Management Skills
Kerry Wills
ISBN 978-1-4398-2716-1

Handbook of Public Information Systems,
Third Edition
Edited by Christopher M Shea
and G. David Garson
ISBN: 9781439807569

GIS in Hospital and Healthcare
Emergency Management
Ric Skinner
ISBN 978-1-4398-2129-9

Healthcare Informatics:
Improving Efficiency and Productivity
Edited by Stephan P. Kudyba
ISBN 978-1-4398-0978-5

HSDPA/HSUPA Handbook
Edited by Borko Furht and Syed A. Ahson
ISBN 978-1-4200-7863-3

Information Security Management:
Concepts and Practice
Bel G. Raggad
ISBN 978-1-4200-7854-1

Information Security Risk Analysis,
Third Edition
Thomas R. Peltier
ISBN 978-1-4398-3956-0

ITIL Release Management:
A Hands-on Guide
Dave Howard
ISBN 978-1-4398-1558-8

Mobile Device Security:
A Comprehensive Guide to Securing
Your Information in a Moving World
Stephen Fried
ISBN 978-1-4398-2016-2

Orthogonal Frequency Division Multiple
Access Fundamentals and Applications
Edited by Tao Jiang, Lingyang Song,
and Yan Zhang
ISBN 978-1-4200-8824-3

Overlay Networks:
Toward Information Networking
Sasu Tarkoma
ISBN 978-1-4398-1371-3

Process Improvement and CMMI®
for Systems and Software
Ron S. Kenett and Emanuel Baker
ISBN 978-1-4200-6050-8

Project Management Tools and
Techniques for Success
Christine B. Tayntor
ISBN 978-1-4398-1630-1

Real Life Applications of Soft Computing
Anupam Shukla, Ritu Tiwari, and Rahul Kala
ISBN 978-1-4398-2287-6

The Project Manager's
Communication Toolkit
Shankar Jha
ISBN 978-1-4398-0995-2

Transmission Techniques for Emergent
Multicast and Broadcast Systems
Mario Marques da Silva, Americo Correia,
Rui Dinis, Nuno Suoto, and Joao Carlos Silva
ISBN 978-1-4398-1593-9

Underwater Acoustic Sensor Networks
Edited by Yang Xiao
ISBN 978-1-4200-6711-8

Wireless Sensor Networks:
Principles and Practice
Fei Hu and Xiaojun Cao
ISBN 978-1-4200-9215-8

AUERBACH PUBLICATIONS

www.auerbach-publications.com
To Order Call: 1-800-272-7737 • Fax: 1-800-374-3401
E-mail: orders@crcpress.com

Real Life Applications of Soft Computing

Anupam Shukla
Ritu Tiwari
Rahul Kala


CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Version Date: 20140514

International Standard Book Number-13: 978-1-4398-2289-0 (eBook - PDF)

To my mother, Kanta Devi Shukla

Anupam Shukla

To my mother, Parwati Tiwari

Ritu Tiwari

To all young researchers round the world…

Rahul Kala

Contents

SECTION II Soft Computing in Biosystems

SECTION III Soft Computing in Other Application Areas

SECTION IV Soft Computing *Implementation Issues*

APPENDICES

REFERENCES

Foreword

Soft Computing today may be portrayed as a massively dynamic concept whose pace of development presents a promising picture of the future world. The mammoth of literature that has been compiled in various forms not only showcases the current development, but also speaks about the massive scope of work that is to come. The constant and exponential growth of the field has been a source of great encouragement and inspiration for all of us to contribute tirelessly to make an intelligent tomorrow where anything that can be perceived is available in reality. At one end, we find very positive steps being taken by numerous researchers worldwide who are making significant contributions towards the theoretical domains and addressing various problems that erstwhile restricted the performance of the systems in general. The increasing use of evolutionary concepts, modularity in design, hardware integration, massively hybrid systems, self-adaptive, and dynamic and online systems are a part of the complete landscape that is immense, but beautiful. Not only does it attract people to explore it in depth, but also presents its boundless and never-ending characteristics. At the other end we find a lot of people contributing towards the application domain where they use soft computing as an effective problem solving tool that ultimately results in effective systems of practical use. The bulk of research over the domains of robotics, wireless networks, optical character recognition, etc. is very encouraging. Each of these fields today stands as a well-established discipline with a long history, issues, concepts and future, mostly involving a multi-disciplinary treatment which makes the whole task an even bigger challenge.

We have made a significant movement toward making a computationally more intelligent world where intelligence lies in all machines that surround us. We have many more challenges to conquer and many more issues to be uncovered. The past decade especially has not only seen positive developments in research labs worldwide, but also witnessed a lot of practical use. Now intelligent systems using soft computing principles are being packaged as the hardware or software in various applications of commercial use. The R&D labs of all major corporates worldwide are using a variety of soft computing tools to provide effective solutions to the ever-increasing human demands. At this juncture, there is a need to educate people about the principles, concepts, applications, issues and solutions to the use of soft computing. I am sure that this book will be great contribution toward the same and would further carve a deep mark in the soft computing literature for reference of enthusiasts for a very long time.

Real Life Applications of Soft Computing is a one-stop book for all knowledge related to soft computing, computational intelligence, machine intelligence, and other related areas. The book is of a special interest to me due to the manner in which it has been conceptualized and ultimately penned down. The authors cover all the theoretical foundations with a practical approach and later present a variety of applications that further help in understanding soft computing systems. The scientific approach with discussion of commonly faced problems would certainly help readers build systems rather than just reading for pleasure. From our teaching experience, we also realize that it is always good to have a practical component attached to the courses whose performance ultimately reflects the overall understanding of the students. Such courses without practical components are like machines without electricity. Making students perform the practical is a much more difficult task due to the various issues that are not commonly found in what they read. Ample coverage of applications, issues and perspectives, makes the book rich and diverse. This book is a significant movement in this direction.

This book will deeply impact the intellect of those who own it, or can access it in their libraries. It is the kind of work that will be consulted often, providing enduring value to those who read it.

I hope that each section, chapter and unit of this book turns out to be thought provoking for the readers and helps them understand, appreciate, implement and contribute to the domain. I herby congratulate the authors for the book and wish the readers a happy reading and lifelong learning.

Prof. Amit Konar
Professor, Electronics and Telecommunications
Jadavpur University
Jadavpur, India

Preface

Soft computing has been a major field of development in the previous couple of decades. The field attracts the attention of a large number of people. The highly multidisciplinary nature of the field further results in a large number of students studying soft computing. The various developments in application areas of soft computing in the previous few years have shown a very large potential for the development of highly-scalable systems using the soft computing tools and techniques.

The present literature beautifully covers the theoretic concepts, but the practical applications might still be very difficult for students to comprehend and implement. Many students find it difficult to visualize the application of soft computing from a theoretical text alone. This book not only aids them in understanding these systems, but also exposes them to the current developments over various fields. This empowers them to pursue research in these areas and contribute towards the research and development at large. The basic purpose of this book is to expose the real life applications and systems of soft computing to the readers. This will enable them to get an in-depth, practical exposure to the methodology of the application of the soft computing approaches in various situations. This book also discusses the various issues and difficulties involved in the process.

The book can also be used as a standard text or reference book for courses related to soft computing, pattern matching, artificial neural networks, etc. The book may be rated as an "elementary" to "advanced" book that introduces the concepts of soft computing to a good extent and then advances to their real life practical examples. Every care has been given to nicely explain all the theoretical concepts, algorithms, applications, tools, and techniques. We have tried to ensure that readers using this as the first text of soft computing are also able to understand each and every line of book.

The book incorporates some of the recent developments over various areas. Many other models have been proposed in the recent years that still exist in theoretical studies and elementary experimentations. These models have not been discussed in the book. Also, due to limited space we have not been able to cover many new exciting areas where soft computing finds applications, such as weather monitoring and music compositions. We would be looking forward to include these in the future editions.

SALIENT FEATURES

- In-depth practical exposure to real life systems
- Discussion on various problems and issues in application of soft computing tools and techniques
- Updated information on developments in various real life application fields
- Multidimensional coverage of the concepts
- Detailed description of the problem and solution for easy reproduction
- Scope for future work and active areas of research presented at various places

WHO SHOULD READ THIS BOOK

Students of Universities as a Text or Reference for Soft-Computing Courses:

This is meant to be a good text or reference book for all the courses related to soft computing, pattern matching, machine learning, etc.

Young Researchers for BSEE/MSEE/PhD Thesis

The book will give them a good insight into the theoretical and application areas and would be of a great help to them to themselves develop the systems.

Researchers

A good coverage of latest technologies would give all researchers a good insight into the developments of the discussed domains. Further, a practical approach would help them easily experiment and reproduce the results.

ORIGIN OF THE BOOK

The book is a result of the research done over the time by the authors. Much of the work presented is done by the authors themselves. Databases to the various problems have either been self-generated by the authors or used from the public database repositories. At some places the authors have incorporated a comprehensive discussion of the problems and the solutions present in the literature. This work was largely possible as a result of the efforts put in by various students of Indian Institute of Information Technology and Management Gwalior in forms of projects: BTech dissertations, MTech dissertations, and PhD thesis.

Many topics are the result of the current research being pursued by the authors. This makes the book state of the art with respect to coverage. Newer advances into the application areas of soft computing have been incorporated. Many of the systems presented are still in the experimental phase and have not yet been implemented by the industry. Many others are being deployed at the industry.

The book has been made possible as a result of the course of artificial intelligence and soft computing being taken by the authors at Indian Institute of Information Technology and Management Gwalior besides other universities. The feedback provided by the students towards the course and their motivation to learn beyond the curriculum have helped a lot in improving the quality of the book.

Acknowledgments

This book is our first step into the world of authored books. It all started as a dream to use our past research as a base to create a big landmark into the research horizon of the world. It is natural that the dream required a lot of encouragement, guidance, help, and support that was provided by numerous people in various phases of the book. Before the book unveils its multidisciplinary contents, it would be wise to acknowledge the people who made this possible.

The authors wish to express their sincere gratitude to Dr. M.N. Buch, chairman, Board of Governors, ABV-IIITM Gwalior whose visionary ideas made it possible for us to think big and formulate the book. The authors further wish to thank Prof. S.G. Deshmukh, director, ABV-IIITM Gwalior for extending all sorts of help and support during the course of writing of this book. The book would not have been initiated without his encouragement and support.

The authors would like to thank Prof. Amit Konar, Jadavpur University, for providing the inspiration for this book, as well as the foreword. His precious words have indeed added a greater charm to the book.

The authors also express their sincere regards to the anonymous reviewers of the various journals and conferences for their valuable suggestions over numerous pieces of work that oriented the authors and guided them in various ways.

The authors further express their thanks to Dr. A.S. Zadgaonkar, vice chancellor–Dr. C.V. Raman Technical University Bilaspur, for the guidance and motivation that he has bestowed during the entire course of work with the authors. The spirit to learn, comprehend, develop, and dream is an inspiration from him that made this book possible. The strong foundation to research inculcated by him largely led to the immense amount of work that has been presented in the book.

The authors express their thanks to Sourabh Runta, director technical, Rungta College of Enineering and Technology, Bhilai for his assistance in developing various codes for speech synthesis and speaker recognition, which forms an integral part of the book.

The book would have never been possible without the pioneering work carried out by the co-researchers associated with the authors. The authors also wish to thank the student fraternity of the Institute for taking deep interest into the courses titled "Artificial Intelligence" and "Soft Computing." Their interest to learn beyond the classroom program has been the key motivation for the book. The authors are highly indebted to all the students who undertook challenging problems as a part of their PhD, MTech or BTech thesis and came up with excellent solutions and experimental results. The authors wish to thank Dhirender Sharma, Chandra Prakash, Rishi Kumar Anand, Sourabh Sharma, Piyush Sharma, Hemant Kumar Meena, Prateek Agrawal, Mamta Agrawal, Rahul Dubey, Akhil Kumar Meshram, Chandra Prakash Meena, Prabhdeep Kaur, A. K. Mitra, R. R. Janghel, Anuj Kumar, Gaurav Agarwal, Anand Ranjan, Shivanshu Mittal, Harsh Vazirani, Anuj Kumar Goyal, Siddharth Arjaria, and Ram Nivas Giri for the quality results they produced during the course of work with the authors. The authors further wish to thank Kshitij Verma, Anuj Nandanwar, Anurag Agrawal, Sanjay Kumar, and Pritesh Tiwari for ensuring each and every stage of author submission completed on time. The help provided by them in editing of the manuscript was of a large help at a time when it was needed the most.

The first author would also like to thank his son Apurv Shukla for his support during the course of writing the book. The second author thanks her mother Parvati Tiwari for motivation and blessings in writing the book.

This book required an immense amount of work by numerous people around the globe. The draft was heavily worked over to ensure high quality. The authors wish to thank the entire staff of Taylor

& Francis, Auerbach Press, Glyph International, and all the associated units who worked over different phases of the book. The authors thank all the people who have been associated with and have contributed toward the book.

Above all, we all thank the Almighty for showering us with His constant blessings and love. Without Him, this project would not have been possible.

Anupam Shukla

Ritu Tiwari

Rahul Kala

Authors

Dr. Anupam Shukla is an associate professor in the IT Department of the Indian Institute of Information Technology and Management Gwalior. He has 22 years of teaching experience. His research interest includes speech processing, artificial intelligence, soft computing and bioinformatics. He has published over 120 papers in various national and international journals/conferences. He is referee for 10 international journals including Elsevier, IEEE, and ACM computing and in the editorial board of International Journal of AI and Soft Computing. He also received Gold Medal from Jadavpur University during his postgraduate studies.

Dr. Ritu Tiwari is an assistant professor in the IT Department of Indian Institute of Information Technology and Management Gwalior. Her field of research includes biometrics, artificial neural networks, signal processing, robotics and soft computing. She has published around 50 papers in various national and international journals/conferences. She has received Young Scientist Award from Chhattisgarh Council of Science & Technology in the year 2006. She also received Gold Medal in her post graduation from NIT, Raipur.

Rahul Kala is a student at the Indian Institute of Information Technology and Management Gwalior. His areas of research are hybrid soft computing, robotic planning, biometrics, artificial intelligence, and soft computing. He has published over 25 papers in various international and national journals/conferences. He also takes a keen interest toward free/open source software. He secured All India 8th position in Graduates Aptitude Test in Engineeging-2008 with a percentile of 99.84. Rahul is the winner of Lord of the Code Scholarship Contest organized by KReSIT, IIT Bombay and Red Hat. He also secured 7th position in ACM-International Collegiate Programming Contest Kanpur Regional 2007.

Section I

Soft-Computing Concepts

1 Introduction

1.1 SOFT COMPUTING

From childhood, we have been taught strict rules that we at first believed always hold true. We may have assumed that the world moves by these strict rules. If you ask a child to solve an arithmetic problem, that child can follow the arithmetic rules to find a 100 percent correct solution. But there is another paradigm of thought. In the real world, it is not always possible to be precise in finding answers or solutions to problems. In many situations, natural systems tend to do things that they believe are true rather than things that are compulsorily true. The foundations of defining strict procedures, rules, and methods hence fail if the underlying belief fails. But this gives us an exciting view of a world that is driven by belief, impreciseness, and approximations and that we all have observed numerable times in our daily decision making.

To illustrate this fact, let us take a very simple example. Suppose a bag contains 1,000 pieces of fruit, and each piece can be either an apple or an orange. You are asked to instruct a child to separate the fruits. You might tell the child that the orange ones are oranges and the red ones are apples. But you might then realize that every apple is not perfectly red. Some apples may have lost some of their color over time. So you then might try to explain to the child all the variants of red and orange. You might even try explaining the difference between the odors of the two fruits. You then realize that some apples are highly decolorized or have lost their characteristic odor, so you have already compromised the precision of the child's sorting. Now imagine that there are 100 types of fruits, rather than just two. You would have a great deal of trouble explaining the differences between all the fruits in such a way that the child could precisely separate them. If some fruits do not match the criterion of any fruit, the child might get stuck. This is where imprecision comes into play. In reality, however, any child can easily sort all fruits based on his or her past experiences, beliefs, and confidence.

The example reveals how real life is filled with uncertainty. We usually proceed through this world by accepting the most certain computed fact and regarding it as true. But we can never deny the fact that uncertainty always exists. For example, the next time you get ill and consult a doctor, ask your doctor whether he is fully sure of his diagnosis. After he looks at the methodology of treatment, he might reveal that he took many preventive measures to stop diseases that could occur. Regarding the cure, the inferences were the result of perception rather than a full mathematical proof.

A small look into all the systems might give you a fair idea of the notion being presented here. Uncertainty exists not only in the natural systems with which we are all familiar but also in almost all human-made systems that we encounter. In fact, many of the systems that you appreciate as a marvelous creation of humankind actually use principles of soft computing. This includes speaker recognition, handwriting recognition, face recognition, disease identification, and many more. So the next time you swipe your finger at your fingerprint-secure laptop, be aware that the system is using soft computing. Or the next time you select your hello tune using interactive voice response, know that it also uses soft-computing techniques. And in any system that uses soft-computing techniques, the presence of uncertainties and impreciseness is a given.

But let us see what and how much uncertainty there is. The uncertainty exists in the preciseness of the results. This means that if you ask a soft-computing expert system to get sugar from the market, it might end up getting salt. Or if you asked your system to find the value of 5 divided by 2,

your system might give a long number whose value is almost that of the result. But our society loves perfection, so we definitely do not want our systems to be imperfect in any sense. So let us look at how much uncertainty we are talking about. When you go out of your home to a nearby shop, do you consider that you might actually be stabbed on the way or that there might be a bomb blast in which you might lose your life? Most of us are mature enough to believe that such events are so rare as to not occur, and thus we go out of our homes with full confidence. Another way to state this is we are sure that there is nothing unusual in the day, so such events would not take place. The same is the case with soft-computing systems, which have applications in almost every domain.

Another interesting fact that we observe in nature is its complexity. Even the simplest systems that you could imagine are actually very complex. This complexity sometimes reaches the extent that it might not be possible for even the fastest computers on Earth to simulate the system. If you tried to simulate such a system, you would realize the heavy computational load that is involved.

Even machine-controlled systems are not that far from such high computational loads. The well-known example of the traveling salesman problem (TSP) is one case in point. In this problem, n stations are given. A salesman has to start from one station, travel to all of them in some order, and then return to the starting station. The objective is to minimize the total traveling distance.

Many of such problems are very practical in day-to-day life. For example, imagine that you have designed a robot that is supposed to clean n number of places in your home, a problem that is similar to the TSP. This robot senses the surroundings and maps the n places. Then it computes the shortest cycle for implementation. If the number of places is large, the robot might actually end up calculating the shortest distance on one day and cleaning the house on another day. But suppose that the next day, the objects have been moved. This means that the robot must recalculate the shortest path. It can hence be observed that this is not the solution. Rather, the solution lies in finding the best possible path in a given span of time. This would again lead to approximations and uncertainties.

From this example, it is clear that all types of systems cannot run on precise decisions that are the result of hardbound rules. Instead, we need to enter the domain of soft computing. Here we find solutions to all the problems that the systems in our examples faced, with little loss of precision.

Before we continue our journey toward the versatile and unending field of soft computing and its applications, let us first formally define what soft computing is.

1.1.1 What Is Soft Computing?

According to Professor Lofti Zadeh, soft computing is "an emerging approach to computing, which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision" (Zadeh, 1992b) This definition clearly states the role of imprecision in effective decision making in a finite time.

Definition 1
Soft computing is an emerging approach to computing that parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision.

Natural processes have always been the inspiration behind the design of soft-computing techniques. Soft-computing systems try to imitate, to some extent, the functionality given by natural processes—especially the human brain. It is clear that even if we succeed in automating a small fraction of the natural processes, the results would be enormous. Definition 1 clearly explains the use of uncertainty and imprecision in making systems more robust and scalable. The motivating factor of natural processes and learning from cognition plays a key role in making complex systems possible. This is the beauty of soft computing.

Another commonly used definition of soft computing given by Professor Zadeh states, "The guiding principle of soft computing is to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness, and low solution cost" (Zadeh, 1996).

Definition 2
The guiding principle of soft computing is to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness, and low solution cost. (Zadeh, 1994)

This definition clearly states that the processes can be very computationally expensive. In reality, it might not be possible for any system to solve the problem in a finite amount of time. Hence we require soft-computing techniques to give the best possible results in the smallest amount of time. Soft-computing techniques are well suited to handling data for these types of problems. The techniques give the best solutions from the entire pool of solutions possible per the time constraints.

These classes of problems are optimization problems, in which we generate solutions according to the problem and then try to optimize them. Optimization problems may be defined as "finding optimal values for parameters of an object or a system which minimizes an objective (cost) function" (Kasabov, 1998). In optimization problems, we try to generate solutions by varying the solution parameters. The aim is to maximize or minimize the value of the objective function that judges the requirement of the problem or the quality of the solution.

Definition 3
Optimization is about finding optimal values for parameters of an object or a system which minimizes an objective (cost) function (Kasabov, 1998)

Soft-computing techniques have a great deal of relevance in optimization problems, giving very good, time-efficient results. These techniques generate a diverse set of solutions that they can then optimize to return the most optimal solution.

Soft computing, hence, means computation using soft conditions—that is, computing in which errors are undesirable but acceptable to a certain degree. Soft computing sacrifices the preciseness of the solution, but in return it is able to solve problems that either were impossible to solve using traditional methods or were computationally too expensive to solve. Soft computing allows us to model problems that would have been impossible to model due to the complex nature of those problems. This complexity exists not only in modeling; rather, it extends to the domains of space and time. But soft computing optimizes both space and time to solve problems that were unsolvable using traditional means. Soft-computing techniques have thus become a valuable tool for people in a variety of domains and disciplines and are becoming extremely popular due to their high performance, ease of modeling, and ease of use. In the subsequent sections of this chapter, we will discuss these techniques and how they are used.

1.1.2 Soft Computing versus Hard Computing

Now let us divert our attention to hard computing. Unlike soft computing, hard computing deals with precise computation. The rules of hard computing are strict and binding. You are given a procedure to solve a problem. The inputs, outputs, and procedure are all clearly defined. Hence you apply the procedure to get the desired output. Because the rules are so well defined and the inputs so precise, it is possible to get precise answers without any degree of uncertainty. The same results will be returned, no matter how many times you perform the experiment, where you perform the experiment, how you perform the experiment, and who performs the experiment. The results only change if the procedure, underlying assumptions, or rules change. The beauty of such a system is its preciseness.

An example of hard computing is the arithmetic sum. Whenever the inputs are 1 and 2, the output is 3. We can be assured that the answer is correct without worrying about anything else. Most of the laws of physics, mathematics, and science have been modeled using the principles of hard computing. Whenever there is a modeling problem, we solve by taking approximations and assumptions that are believed to hold true. Hence, hard computing models everything as a set of predefined rules that always hold. If these laws were ever to fail, the result would be unimaginable.

Hard computing is thus better than soft computing in that it gives precise answers. But when using hard computing, we need to be aware of the following points, which introduce limitations in the working of hard computing, especially when real life data of enormous size is used.

- **Model complexity and assumptions:** Some problems are very difficult to model. Normally the approach for modeling a problem is to use a set of mathematical equations and variables that mathematically interpret the problem. These equations are then solved simultaneously, keeping the constraints constant to get the final solution. But it is not always possible to model a problem in this way due to its complexity. In such instances, we may opt to use approximations that do not affect the answer much and that are able to give the results. Although we would not make assumptions that would affect the solution much, many problems can still not be modeled due to their complex nature.

  As an example, imagine that you are supposed to solve the problem of character recognition. Here, the input is in the form of a grid on M rows and N columns. Your problem is to identify the character based on this input. Suppose you start making rules to identify each and every character. The number of rules needed to identify only 26 English characters is very large. If you had to create each rule, you would probably become highly confused and leave the work. This means the problem is too difficult for hard computing. Likewise, if we move into the domain of face recognition, the problem increases even more. Thus hard computing is not the method of choice for such complex systems.
- **Computationally expensive problems:** Suppose that you have been able to model the given problem. But then you try giving real life data that are too large in size or dimensionality. "Large size of data" refers to the data required by the system for its initialization or setup, while "dimensionality" refers to the number of parameters on which the output depends. With such large data, it will take the system a long time to compute the final result. But it is natural that the system you are trying to make has to respond in a finite amount of time. Hence such systems would not serve your purpose.

  Imagine that you are asked to optimize a given function when given a set of constraints. You use all sorts of mathematics to try to get the answer, but still the system takes a long time. Even if you get a solution, you may find that other better solutions also exist. As a result, the system fails to perform up to your expectations.
- **Space complexity:** Many systems require a huge amount of memory to perform. It may not always be possible to have such a large amount of memory available. Thus, memory constraints may cause a problem, as they may not allow the system to perform, even if all other factors are favorable.

  Imagine that you are trying to recognize a speaker using point-to-point matching over a set database. You need to make this system on one chip. Your chip may not have enough memory to store all the data sets in any form, especially as the chip memory might be expensive. Hence, the memory constraints would cause another problem.

It should be clear that hard computing, though preferable, is not always able to solve problems. It has a limited scope and cannot be used in all problems. The problems with hard-computing techniques increase exponentially whenever we increase the size of the data or the data's dimensionality.

Soft computing, on the other hand, offers a very simple solution to all the problems. With soft computing, we can solve the problems and obtain good results. Another good feature of soft computing is that we only need to make a model. Tools exist that automatically convert this model into a working code using a graphical user interface (GUI) or simple commands. Soft-computing techniques are now available as in-built libraries in MATLAB®, Java, C, and C++, among others. This makes working with soft computing fun and easy.

1.1.3 Soft-Computing Systems

Soft computing started in the 1990s. Over the past two decades, the field has rapidly matured and found applications in various domains. The field of soft computing is highly multidisciplinary. People specializing in soft computing are studying newer and newer systems and trying to find the solutions to various problems.

The major areas of application of soft computing are given below. It should be noted, however, that the list is much longer and more versatile than this.

- **Biometrics:** These systems deal with the identification and verification of people based on their biometric properties, including speech, face, iris, fingerprint, and so forth. These systems are used for security and automation purposes and are being actively deployed in industry (see Chapters 7, 8, and 9).
- **Bioinformatics:** These systems study and analyze molecular biology using soft-computing tools. Bioinformatics systems have been used to study the complex structures of proteins, which was not possible using any other technique due to their very large size (see Chapter 10).
- **Biomedical systems:** These systems are used to identify the presence of diseases. The diagnosis has varied applications to assist doctors and help them make decisions regarding the presence or absence of diseases. These systems are being actively used for various kinds of analysis, diagnosis, and monitoring (see Chapters 11 and 12).
- **Robotics:** Soft-computing techniques are extensively used for robotic knowledge representation, learning, path planning, control, coordination, and decision making, to name a few. Robotics tends to be a vast area for various artificial intelligence and soft-computing techniques at multiple levels (see Chapter 14).
- **Vulnerability analysis:** Soft-computing techniques are extensively used to model systems in order to find vulnerability. This type of analysis has applications in various domains. For example, in banks, these models are used for credit risk analysis; in information security, for risk analysis; and in social living for social risk analysis (see Chapter 13).
- **Character recognition:** Another exciting field of soft computing is character recognition. Soft-computing techniques identify characters in various scripts that may be typed (or optical) or handwrittens in nature. These systems are extensively used in scanner software, text extractors, and so forth (see Chapter 15).
- **Data mining:** In data mining, we try to extract rules, features, and trends from a large database. Soft-computing techniques are extremely time effective for such purposes. This task is also called knowledge discovery in database.
- **Music:** Soft-computing techniques can also be used for the automatic generation of music, genre recognition, artist recognition, and so forth. These systems can be used to assist composers in generating music or in generating the sounds of musical instruments (see Chapter 17).
- **Natural language processing:** Natural language processing (NLP) refers to the understanding, representation, and conversion of the natural languages used by humans. Soft-computing techniques are being used for all aspects of NLP. The success of these techniques has made communication possible between people and machines, regardless of their original languages.
- **Multiobjective optimizations:** Various natural problems have been modeled as multiobjective optimizations, in which some constraints are given. The problem is to optimize a set of objective functions keeping these constraints valid. Soft-computing techniques offer the best solution to these problems.
- **Wireless networks:** Soft computing is used in wireless networks because it offers the most optimal placement of wireless sensors, intelligent channel allocation, and so on.

Soft-computing techniques have thus made it possible to optimize the performance of wireless networks.

- **Financial and time series prediction:** This is one of the most exciting and heavily studied fields, in which people try to predict the financial timeline and other series w.r.t. time based on a set of parameters. Soft-computing techniques are used to predict the future on the basis of past trends.
- **Image processing:** Soft computing has applications in the field of image processing, including in noise removal, segmentation, and extraction. Image processing is the basic step for any system working with images or videos.
- **Toxicology:** In toxicology, soft computing is being used to predict the affect of drugs on animals.
- **Machine control:** Soft-computing techniques can be used to control the various machines used in factories, including controllers for temperature, speed, trajectory, and so on. This eliminates the need of humans and gives better performances.
- **Software engineering:** Soft-computing techniques may be used for the planning and scheduling of various parts of software engineering projects. They are also used for automatic code generation and testing purposes.
- **Information management:** Soft-computing techniques are used to represent information, to learn from information, and so on.
- **Picture compression:** Another exciting field of soft computing is in picture compression. The compression makes the pictures more compact for storage, handling, and communication purposes (see Chapter 16).
- **Noise removal:** The signals used for various techniques need to first be processed to remove noise. This noise removal is provided by soft computing, which first tries to find the possible amount of noise based on past noises and then generates an antinoise to cancel its effect.
- **Social network analysis:** In social network analysis, the social behaviors of people are analyzed on the basis of social networking sites, forums, and so forth. This field deals with the study of patters and behaviors in these networks.

As is obvious, soft computing is used in a long list of domains. All of these are active fields of research in the modern scientific society. In this book, we study some aspects of some of these fields. Keep in mind that these methods may be generalized for other systems as well. In this book, we attempt to cover as many soft-computing techniques as possible in a diverse set of applications to allow readers to try their hand at any of these fields using any of the methods so they can see the results for themselves.

1.2 ARTIFICIAL INTELLIGENCE

We all know that human beings are intelligent animals. This means that we can perform many tasks that require understanding, perception, and decision making. Through our understanding, we are able to complete tasks in the best possible ways. This intelligence has separated us from machines, which use a definite input, output, and procedure. Suppose you ask your child to get a glass of water; it is likely that your child will do this naturally, without much thought. But asking a machine to do the same work may not be possible. We talk more about humans in comparison to other animals here as they are believed to be species with extraordinary intelligence level. Artificial intelligence (AI) deals with making machines intelligent. In researching and applying AI, we try to induce intelligence in machines by artificial means.

Over the past few decades, researchers have been trying to learn from the way humans work. The quest to create machines that imitate human beings has resulted in many models for solving various problems. Even after so many decades, however, we have been able to imitate humans to only a very brief extent. For the most part, how we learn and perform still remains a mystery. It is fantastic to see how even small children can do such complex tasks as recognizing parents, words, and places

so easily. If you tried to get machines to learn the same tasks, you would realize the complexity of the problem. Suppose a child is walking down a road. If some obstacle comes up, that child would move to the side to avoid the obstacle. If you wanted a machine to do the same thing, it would be extremely difficult. The human brain is filled with mysteries that need to be unfolded if we want to learn from them and implement them.

Before we move further in our discussion, let us first formally define *intelligence*. The *Oxford English Dictionary* defines *intelligence* as "intellect, understanding. Intellect is a faculty or reasoning, knowledge, and thinking." Another definition of *intelligence* is "the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines" (McCarthy, 2007). This clearly states our notion of intelligence. Humans are called intelligent because they are able to perceive a situation, comprehend it, look for solutions, and make suitable decisions. This decision-making ability separates us from the "nonintelligent" machines. Intelligence in humans is a natural phenomenon that takes place as a result of lifelong experimentation, learning, and practice.

Definition 4
Intelligence means intellect, understanding. Intellect is a faculty or reasoning, knowledge, and thinking. (Oxford English Dictionary)

Definition 5
Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines. (McCarthy, 2007)

Let us now try to understand artificial intelligence.

1.2.1 What Is Artificial Intelligence?

Artificial intelligence (AI) means adding intelligence to machines artificially so that they become intelligent and behave in ways similar to humans. AI can be formally defined as "the simulation of human intelligence on a machine, so as to make the machine efficient to identify and use the right piece of 'knowledge' at a given step of solving a problem" (Konar, 2000). *Knowledge* is defined as awareness or familiarity. Another definition of artificial intelligence describes it as "the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable" (McCarthy, 2007). This definition clearly states that AI techniques are attempting to induce in machines the capability of acquiring knowledge, storing and retrieving knowledge, learning from the available knowledge, comprehending knowledge, and, above all, making decisions.

Definition 6
Artificial intelligence is the simulation of human intelligence on a machine, so as to make the machine efficient to identify and use the right piece of "knowledge" at a given step of solving a problem. (Konar, 2000)

Definition 7
[Artificial intelligence] is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable. (McCarthy, 2007)

AI systems have a great deal scope in making effective expert systems and decision support systems (DSS). Because of the added intelligence, these systems can operate autonomously, which

enables them to take end-to-end responsibility of a task. This further alleviates the need of humans in such problems. The need for and use of autonomous machines can easily be seen in robotic systems. For example, robots are being built for rescue operations that are dangerous for humans to perform, as well as for survey operations in inaccessible areas, on other planets, and elsewhere. This saves expense and protects humans from the dangers of such situations. These autonomous robots are a big boon in getting the needed information easily. In addition, the robots can perform in all kinds of conditions and even in unlikely environments.

1.2.2 Problem Solving in AI

The concepts of AI are not restricted to a single set of algorithms. Rather, they extend to the intelligent use of a large number of tools and techniques and to many more algorithms that are designed per the problem requirements. Advances in algorithm design have given us many algorithms that are now being used for problem solving. These advances allow AI techniques to incorporate many diverse fields of algorithms that can be used in multiple ways and on multiple levels.

The most commonly used and most basic set of algorithms are the graph search algorithms. A *graph* is defined as a collection of vertices and edges and may be denoted by $G(V, E)$. Here we are given a set of vertices and a set of edges that connect vertices. We are given a vertex from which we start our search. The problem is to reach the goal vertex by traveling along the edges from vertex to vertex. In the example in Figure 1.1, the edges are $E = \{1, 2, 3, 4, 5, 6\}$, and the vertices are $V = \{(1, 2), (1, 3), (1, 4), (2, 7), (4, 5), (4, 6), (3, 9), (6, 9), (5, 6), (7, 5), (5, 8)\}$. In this example, we are supposed to reach the goal vertex starting from the source vertex. One such solution has been highlighted. In notation, we would present this solution as $1 \rightarrow 4 \rightarrow 5 \rightarrow 8$. The algorithm returns true if the goal vertex is found; otherwise, it returns false. In the same example, the algorithm would return true in the given configuration with 1 as the source and 8 as the destination. However, if we change the source to vertex 2 and the goal to vertex 6, the algorithm would return false, as there is no way to reach vertex 6 by starting from vertex 2.

The number of vertices in a real AI problem may be very large. Thus it may not be possible to generate the entire graph as we did for this example. Instead, the graph is generated as it is explored, and the vertices and edges are added dynamically as the algorithm runs. Using this technique, we are able to save both space and time.

These algorithms have many applications in scheduling, planning, game solving, and other such problems. In each, we model the whole problem as a graph, where every vertex corresponds to a state of the game. To understand this better, let us first explain what is meant by a *state*. A state may be defined as the status of the present situation out of all the possible statuses that the problem can

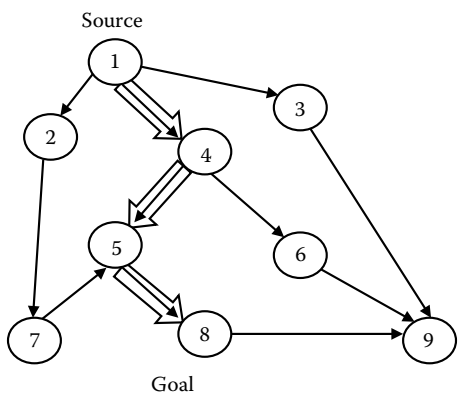


FIGURE 1.1 A general graph.

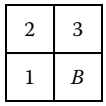


FIGURE 1.2a The general game board in a four-puzzle problem.

take. Suppose the problem we are working on can take any of the n possible states. Here n may be very large. Each of these n states is represented by a vertex of the graph. Note that because n is very large, the vertices might not be initialized or generated while coding, but they do exist conceptually. In the algorithm, they may be generated whenever needed or whenever explored.

Definition 8
A state may be defined as the status of the present situation out of all the possible statuses that the problem can take.

Now let us come to the edges. An edge exists between two states if it is possible to reach the latter state from the former. Hence if we can make a move of 1 unit and reach a state j from a particular state i, we would proceed with the making of the edge (i, j). The edges may be traversed to explore the possible transitions between the states as the moves are made.

To better understand the concepts of graph and search presented above, let us take an example of the famous four-puzzle problem. In this problem, there are four cells. Three of the cells have a number, while the fourth cell is blank (see Figure 1.2a). We can slide any adjacent cell in the place of the blank cell vertically or horizontally. The board is shuffled, and our task is to arrange the numbers in a particular order. In this problem, a state corresponds to a particular configuration of the board. Let us say that we denote a board by <*P*, *Q*, *R*, *S*>, where *P* is the top left cell, *Q* is the top right cell, *R* is the bottom left cell, and *S* is the bottom right cell. Let *B* denote the blank. Possible states are <1, 2, 3, *B*>, <1, 2, *B*, 3>, <1, 3, *B*, 2>, and so on. An edge exists between two states if we are able to reach one state from the other. We know that if there is a *B* is at the bottom right, we can either slide *Q* down or *R* to the right. So if we consider the state <1, 2, 3, *B*>, we can change the state in one move to <1, *B*, 3, 2> by sliding 2 down or to <1, 2, *B*, 3> by sliding 3 to the right (see Figure 1.2b). Note the use of bidirectional arrows in the example. This indicates that the second state can be reached from the first as well as the first from the second. Hence an edge exists from vertex <1, 2, 3, *B*> to <1, *B*, 3, 2> and to <1, 2, *B*, 3> as shown.

The problem is to reach a goal state, such as <3, 2, 1, *B*>, starting from an initial state, such as <1, 2, 3, *B*>. The graph for this is given in Figure 1.2c.

Once the initial source and the final goal are known, the search for the solution can be done using any standard graph search algorithm. We will now discuss the general fundamentals of graph searches. Then we will briefly discuss the various graph searching algorithms.

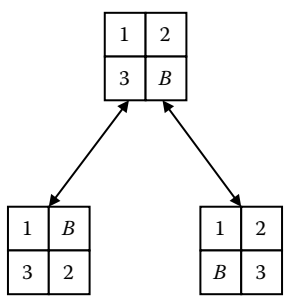


FIGURE 1.2b The game graph for the four-puzzle problem.

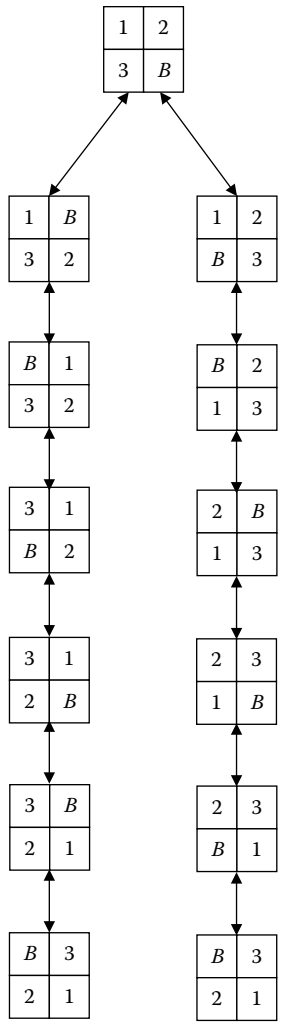


FIGURE 1.2c The complete solution graph of the four-puzzle problem.

A graph search always starts from the source. We maintain two separate lists. One is called the open list, and the second is the closed list. The closed list contains all the vertices (also called *nodes*) that have been visited in the past. The open list stores the vertices that have not yet been visited but that have been discovered. Initially the closed list is empty, and the open list contains only the source node, because the source has been discovered but not yet visited. At all iterations, we select a vertex (say x) from the open list. We remove this vertex x from the open list and expand this vertex. Expansion of a node means to find the vertices next to the vertex. This gives all vertices y such that the (x, y) edge exists. In the Figure 1.2 example, expansion of vertex <1, 2, 3, *B*> would give <1, *B*, 3, 2> and <1, 2, *B*, 3>. If the newly generated vertices are not in an open or closed list, then they are added to the open list. This is because the vertex being added has been discovered but not yet visited. The vertex x is marked as visited and is added to the closed list. In this manner, we proceed until we extract the goal from the open list. If at any point the open list becomes empty, we conclude that it is not possible to reach the goal from the source.

The various graph searching algorithms mainly differ in the way they extract a vertex from the open list. The difference in the strategies used decides the algorithm. Different algorithms work well in different conditions. Choosing an algorithm is much more a question of trying to predict the

input. In general, the performance of the different algorithms cannot be compared if the inputs are completely unknown.

The algorithms may be classified into two categories. The first category forms the class of algorithms that do not have any means to evaluate the closeness of a vertex to the goal. The second category can evaluate the possible closeness of a vertex to the goal based on a function known as the *heuristic function*. The word *heuristic may* be defined as "rule of thumb. People develop their own rule of thumb based on their experience acquired over the years." Heuristic functions may be used to "estimate, calculate, uncover, discover, purify, and simplify solutions to problems" (Kliem and Ludin, 1997). Thus, heuristic functions may be defined as functions that "are used in some approaches to search to measure how far a node in a search tree seems to be from a goal" (McCarthy, 2007). The heuristic function is used to measure the fitness or effectiveness of a vertex; the closer a vertex is to the goal, the better would be its heuristic function.

Definition 9
Heuristic means rule of thumb. People develop their own rule of thumb based on their experience acquired over the years. Heuristic may be used to estimate, calculate, uncover, discover, purify, and simplify solutions to problems. (Kliem and Ludin, 1997)

Definition 10
Heuristic functions are used in some approaches to search to measure how far a node in a search tree seems to be from a goal. (McCarthy, 2007)

The following are the most commonly used graph algorithms.

- **Breadth-first search (BFS):** This algorithm first visits the various nodes that are closer to the source and then proceeds to the farther ones. The BFS algorithm works very well if the goal is believed to be very near the source. The BFS gives good results when applied to social network graphs, as it is normally observed that in these graphs, the various nodes lie quite close to one another. In BFS, a node closer to the source is visited first. The open list is implemented by a queue that follows a first-in, first-out (FIFO) structure.
- **Depth-first search (DFS):** Unlike the BFS, the DFS starts by going deep into the graph. It tries to expand nodes and penetrate deeper into the graph. When it is not possible to further expand the nodes at any level, it then moves to expand the nodes of the previous level. The open list is maintained by a stack, which follows a last-in, first-out (LIFO) implementation.
- **Iterative deepening depth-first search (IDDFS):** This type of search follows the principles of the depth-first search, but we limit the maximum depth to a certain limit. This limit keeps increasing as we proceed with the algorithm.
- **Best-first search:** This search selects the best possible nodes from the open list and expands them further. The motivation for this search is that the best nodes would reach the end nodes in the shortest possible time. It continues to select and expand the best nodes until the goal is reached. The nodes are evaluated based on the heuristic function.
- **A* search:** This algorithm is similar to the best-first search. However, instead of optimizing only the probable distance from the goal (heuristic), this search tries to optimize the total path, including the distance traveled to reach the vertex so far from source and the heuristic value of this vertex. The total cost is calculated; this total cost consists of the historic cost and the heuristic cost. The least total cost vertex is selected from the open list.
- **Multi Neuron heuristic search:** This is similar to the A* algorithm. However, instead of selecting the node with the best cost, it selects α number of vertices from the open list. The costs of these vertices vary from best to worst. These vertices are then processed sequentially. This algorithm is believed to work better when the heuristic function may change its

value very sharply between vertices. α is a parameter whose value may be varied to control the algorithm. A value of 1 converts this algorithm to an A* algorithm, while a value of infinity converts this into breadth-first search. The details of this algorithm can be found in Shukla and Kala, 2008a.

One of the most commonly used algorithms for real life problems is the A* algorithm. Other algorithms may be made by making small changes to this algorithm. The A* algorithm is presented in Algorithm 1.1. An example of using this algorithm can be found in Chapter 14.

Algorithm 1.1

A*(graph, source, goal)
Step 1: closed, open ← empty list
Step 2: Calculate cost of open
Step 3: Add source to open
Step 4: While open is not empty
Do
Step 5: Extract the node *n* from open with the least total cost
Step 6: If *n* = goal position, then break
Step 7: For each vertex *v* in expansion of *n*
Do
Step 8: if *v* is in neither open nor closed list, then compute costs of *v* and add to open list with parent *n*
Step 9: if *v* is already in open list or closed list, then select the better of the current solution or the solution present already and update the cost and parent of the vertex
Step 10: if *v* is already in closed list, then select the better of the current solution or the solution present already and update the cost and parent of the vertex and all children from the vertex
Step 11: Add *n* to closed

1.2.3 Logic in Artificial Intelligence

Just as logic drives many systems in real life, it also has a prominent role in AI. Logic may primarily be seen as a method of knowledge representation. Here we try to represent a system by a set of rules or in a way that can be easily understood and implemented in the machine. Knowledge is of deep interest to system developers because it provides a means for the machine to understand, act, and make decisions and inferences based on a common understanding of the knowledge. It helps erase the gap between human and machine understanding.

Before we go further, let us first give a formal definition of logic: "What a program knows about the world in general, the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals" (McCarthy, 2007).

Definition 11
Logic is what a program knows about the world in general, the facts of the specific situation in which it must act, and its goals are all represented by sentences of some mathematical logical language. The program decides what to do by inferring that certain actions are appropriate for achieving its goals. (McCarthy, 2007)

Another term associated with logic is knowledge. *Knowledge* is defined as "a function that maps a domain of clauses onto a range of clauses. The function may take algebraic or relational form depending on the type of applications" (Konar, 2000).

Definition 12
Knowledge is a function that maps a domain of clauses onto a range of clauses. The function may take algebraic or relational form depending on the type of applications. (Konar, 2000)

We now have a fair idea that the motivation is to induce intelligence in machines by introducing some means of knowledge representation. In essence, we are trying to explain logic to a machine that until recently had been doing only trivial tasks. The simplest implementation is called as the production system.

The production system consists of simple if-then clauses. The *if* part, or the antecedent, denotes the condition that must be true for the particular rule to fire. The condition expresses the particular case in which the action would hold true. The *then* part consists of the action or conclusion, also known as the consequent, that results from the rule being fired. The entire knowledge is present in the system in the form of a rule set. These systems use these rules for all operations. Once these rules are ready, we know that all available information has been incorporated into the system in the rules. These systems also have a memory, known as the working memory. This memory stores all the information regarding the system's state. Based on this state, the rules are fired by a rule implementer.

Consider the production rule:

If (*X* marks are more than 80) & (*X* attendance is more than 75%), then (*X* grade is A).

Here the *if* part states all the conditions that if true lead to the action.

1.3 SOFT-COMPUTING TECHNIQUES

In the next few subsections, we state the various concepts that collectively make up the field of soft computing. Soft computing mainly incorporates artificial neural networks, fuzzy logic, and evolutionary computing. New models, called hybrid systems, are currently being built from a combination of these techniques.

1.3.1 Artificial Neural Networks

The artificial neural network (ANN) is an attempt to imitate the functionality of the human brain. The human brain is believed to be composed of millions of small processing units, called neurons, that work in parallel. Neurons are connected to each other via neuron connections. Each individual neuron takes its inputs from a set of neurons. It then processes those inputs and passes the outputs to another set of neurons. The outputs are collected by other neurons for further processing. The human brain is a complex network of neurons in which connections keep breaking and forming. Many models similar to the human brain have been proposed.

ANNs are able to learn past data, just like the human brain. The network learns the past data by repeating the learning process for a number of iterations. ANNs learn the data and then reproduce the correct output whenever the same input is applied. The precision and correctness of the answer depends on the learning and the nature of the data given. Sometimes an ANN may refuse to learn certain data items, while other times it may be able to learn complex data very easily. The precision depends on how well the neural network was able to learn the presented data.

ANNs have another extraordinary capability that allows them to be used for all sorts of applications: Once they have been well taught the historical data, they are able to predict the outputs of unknown inputs with quite high precision. This capability, known as generalization, results from the fact that ANNs can imitate any sort of function, be it simple or complex. This gives ANNs the power to model almost any problem that we see in real life applications.

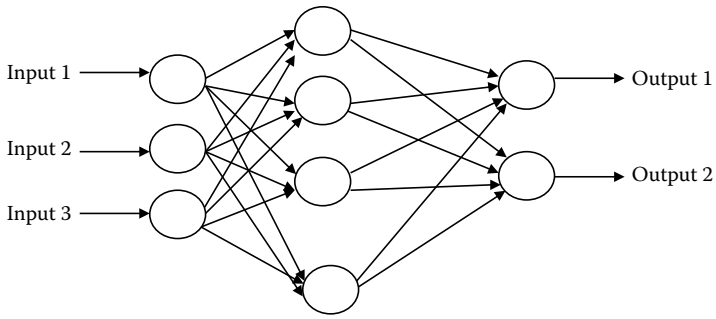


FIGURE 1.3 The architecture of the artificial neural network.

Furthermore ANNs are highly resistant to noise. Thus, if there is any noise in the input data, ANNs are still able to perform appreciably well. This ability makes it possible to make robust applications for ANNs.

Let us now formally define ANNs based on our understanding so far: "The neural network, by its simulating a biological neural network, is a novel computer architecture and a novel algorithmization architecture relative to conventional computers. It allows using very simple computational operations (additions, multiplication and fundamental logic elements) to solve complex, mathematically ill-defined problems, nonlinear problems or stochastic problems" (Graupe, 2007).

Definition 13
The neural network, by its simulating a biological neural network, is a novel computer architecture and a novel algorithmization architecture relative to conventional computers. It allows using very simple computational operations (additions, multiplication and fundamental logic elements) to solve complex, mathematically ill-defined problems, nonlinear problems or stochastic problems. (Graupe, 2007)

The general structure of the ANN is given in Figure 1.3, which shows how the information is processed. Every node takes inputs from the previous nodes and gives its output to subsequent nodes. In this way, there is constant parallel processing by the various nodes. The last node processes the information to generate the final output. From this simple model, many variations can be created (see Chapter 2).

In the next couple of chapters, we will see the underlying reasons why ANNs are able to solve complex problems so easily. We present the complete working of ANNs, as well as the various models that exist. We will also study the application of ANNs to various problems. Finally, we will see how we can use ANNs to solve real life problems.

1.3.2 Fuzzy Systems

Impreciseness is always prevalent in systems. Observations might reveal that you are never able to make discrete rules over a set that always holds precisely true. Rather the rules are always imprecise in nature. We can never work over definite systems that have definite rules. It is from the same observations that fuzzy systems evolved. The impreciseness that is introduced in the system makes them work in a better way. Fuzzy systems are hence being used to model many real life applications.

Before we continue our discussion of fuzzy systems, however, let us first introduce fuzzy sets. In fuzzy sets, every member of the set has a certain degree of belongingness to the set. This degree is called the *membership degree*; the function that decides the membership degree, or belongingness, is known as the *membership function*. This function associates each member of a set with a degree or probability of belongingness. The higher this degree, the more the member is a part of the set.

To illustrate, consider the set good students. A nonfuzzy system would divide the whole class into students who are good students and students who are not good students. If a good student falls below some threshold, that student suddenly becomes a bad student. The fuzzy system, however, would associate some degree by which every student is a good student. Suppose a very bright student has a degree of goodness of 0.9, and a very dull student has a goodness degree of 0.1. If the performance of the good student were to reduce by some amount, there would only be a change in membership degree depending on the extent to which the performance has reduced. The perks and benefits that good students get in the case of fuzzy sets would not suddenly vanish after a small drop in performance, as would be the case with nonfuzzy systems.

Fuzzy systems are implemented by a set of rules called *fuzzy rules*, which may be defined on sets. A nonfuzzy system has very discrete ways of dealing with rules—either a rule fires fully or it does not fire at all, depending on the truth of the expression in the condition specified. On the other hand, in fuzzy systems, the conditions are true to a certain degree and false to a certain degree. The degree of truthfulness and falseness decides the degree to which the rule fires. Hence, every rule is fired to some degree. The outputs of all the rules are aggregated to get the system's final output.

To illustrate, consider a rule that says, "Good students with good attendance get good grades." The grades are a result of the degree of the performance and attendance. These two inputs together decide the grades. Likewise, if there are five such rules, they are aggregated, and the final outcome is computed.

The general procedure of working with fuzzy systems consists of input modeling. Then the membership functions are applied over the crisp inputs to get the fuzzified inputs. Then the rules are applied over these inputs to generate the fuzzy outputs. The various outputs are then aggregated and defuzzified to get the final crisp output. This procedure is given in Figure 1.4.

Fuzzy systems are also known as fuzzy inference systems (FIS). These intelligent systems generate outputs from given inputs using fuzzy logic. These systems are defined as "systems formulating the mapping from a given input to an output using fuzzy logic" (MATLAB Guide).

Definition 14
Fuzzy inference systems are systems formulating the mapping from a given input to an output using fuzzy logic. (MATLAB Guide)

Fuzzy systems continue to be a fascinating playground for experimenting with, working on, and modeling new problems (see Chapter 4).

1.3.3 Evolutionary Algorithm

If you look around the physical world, you might be astonished at how well the various species have adapted over time. Natural evolution has been a key point of motivation for people working in the

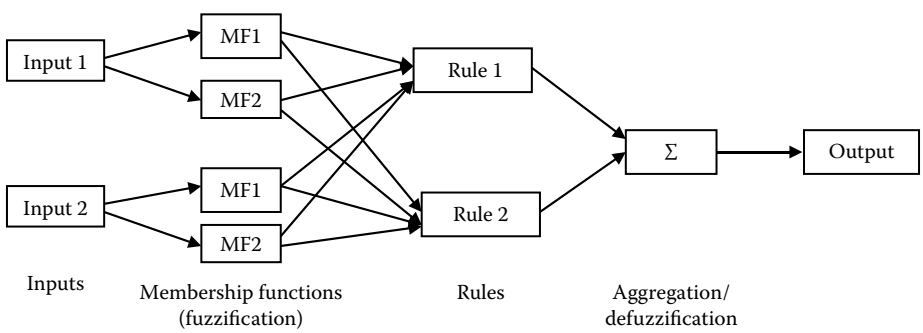


FIGURE 1.4 The architecture of the fuzzy inference system.

field of evolutionary algorithms (EAs), who are trying to imitate the same phenomenon of natural evolution to generate good or optimized solutions. These people are constantly trying to learn from natural evolution and the means and methods by which the best individuals evolve from one generation to the next over time.

Natural evolution results from the fusion of male and female chromosomes to generate one or more offspring. The offspring contain mixed characters of both the male and female counterparts. The offspring may be fitter or weaker than the participating parents. This process of evolution leads to the development of one generation from the previous generation. Over time, the newer generations evolve, and the older ones die out. This survival process goes on and on, following Charles Darwin's theory of survival of the fittest. According to this theory, only the characteristics of the fittest individuals in a population go to the next generation. Others die out. It can easily be seen that both the average quality of fitness and the fittest individual in the population are found as the generations go on evolving.

Evolutionary algorithms work in a similar way. The first step is to generate a random set of solutions to the given problem comprising a population. These random individuals are then made to participate in the evolution process. From these solutions, a few individuals with high fitness are chosen through a method called selection. These individuals are then made to generate offspring. This process of generating offspring by two parents is known as *crossover.* The system then randomly selects some of the newly generated solutions and tries to add new characteristics to them through a process known as *mutation*. Various other operations may also be done by specialized operators. Once the new generation is ready, the same process is applied to that new generation. The fitness of any one solution may be measured by a function known as the *fitness function*.

As we move through newer and newer generations, we find that the quality of solution continues to improve. This improvement is very rapid over the first few generations, but with later generations, it slows. The general structure of simple genetic algorithm, a type of evolutionary algorithm is given in Figure 1.5, and an in-depth analysis and study of this algorithm is in chapter 5.

A formal definition of evolutionary algorithms is "algorithms that maintain a population of structures that evolve according to the rules of selection and other operators, such as recombination and mutation. Each individual in the population is evaluated, receiving a measure of its fitness in the environment" (Spears, 2000).

Definition 15
Evolutionary algorithms are algorithms that maintain a population of structures that evolve according to the rules of selection and other operators, such as recombination and mutation. Each individual in the population is evaluated, receiving a measure of its fitness in the environment. (Spears, 2000)

1.3.4 Hybrid Systems

Each system discussed above has some advantages and some disadvantages. In hybrid systems, we try to mix the existing systems, with the aim of combining the advantages of various systems into a single, more complex system that gives better performances.

Many hybrid systems have been proposed, and many models have been made to better cater to the data. One of the best examples is the fusion of ANNs and FIS to make fuzzy-neuro systems. In this hybrid model, a fuzzy system is built over ANN architecture and is then trained using the back propagation algorithm (BPA). These hybrid systems are highly beneficial in real life applications, such as in controllers. These controllers primarily use the fuzzy logic approach, but optimization of the various parameters is done using ANN training. This combination makes the systems perform with a very high degree of precision. An evolution from this hybrid model is the adaptive neuro-fuzzy inference system (ANFIS), which is similar to the neuro-fuzzy system but is adaptive in nature.

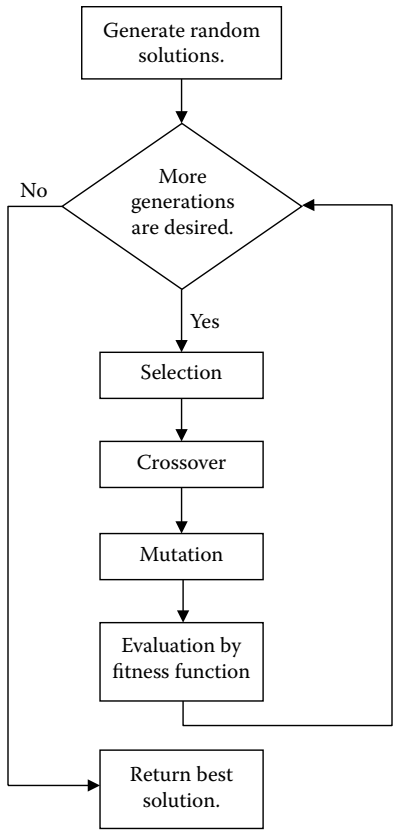


FIGURE 1.5 Simple genetic algoirthm.

Another system that is being used in hybrid models are the evolutionary algorithms. These systems are being added to ANNs to give better performance and obtain the global minima. EAs give the hybrid systems a good performance optimization. They are also being used to make an evolutionary model of ANNs. This system starts from a simple ANN; as time continues, the system evolves the ANN, based on performance, by constantly adding neurons. These systems are also found to give very good performance. In a similar system, EAs evolve the fuzzy system to ensure that the generated system gives a high performance.

The fusion of AI techniques with other algorithms is another common technique. Various models exist, with one system fused to another according to the problem requirements. These systems often cater to the needs of the problem in a better way. They ensure that the performance is good, even as traditional methods fail to deliver good results.

For more information on hybrid models, see Chapter 6.

1.4 EXPERT SYSTEMS

Whenever we need help in everyday life, we consult an expert. An expert is a person who has profound knowledge of the subject; this knowledge is a result of the expert's capabilities and experience. The expert looks at the problems, makes some inferences, and presents the results. As an end customer, we follow the results and assume that the results are true. The expert also continues to update knowledge and to learn from past cases. Experts provide end-to-end solutions to meet our needs. They can make correct decisions, even in the absence or impreciseness of some information.

Expert system (ES) design is motivated by this concept of experts. In this type of design, we try to make systems that can act autonomously. Once these systems are presented with the conditions as inputs, they work over the solutions. The solution given by these systems is then implemented. These systems are a complete design of whatever it might take to draw conclusion once the problem is given. To reach the conclusions may require preprocessing, processing algorithms, or any other procedures. These systems may require a historical database or a knowledge base to learn from.

Expert systems are being built as an end tool to meet the customer's various needs. These tools behave just like an expert who takes care of all the subject's needs. These tools have made it possible for people to enjoy the fruits of soft computing without knowing anything much about the underlying system.

Before we proceed with our discussion of ES, let us formally explain what an expert system is.

1.4.1 What Are Expert Systems?

An expert system may be defined as "knowledge-based systems that provide expertise, similar to that of experts in a restricted application area" (Kasabov, 1998). This definition clearly depicts the quest to make systems that can cater to the user's complete needs, even in the absence of data or in the impreciseness of the data. Another important dimension of an expert system is that of knowledge. Just as an expert works only on the basis of knowledge, the expert systems also require knowledge that needs to constantly update itself. All ES decisions are based on this knowledge. Learning and proper representation of knowledge is an important aspect of these expert systems.

Definition 16
Expert systems are knowledge-based systems that provide expertise, similar to that of experts in a restricted application area. (Kasabov, 1998)

The general architecture of an expert system is given in Figure 1.6, which highlights all the points discussed above. The entire system can be divided into two parts: The first consists of the interface with the user where the required inputs are taken and the required outputs given. The other part is the expert, which processes the inputs to generate outputs and tries to learn new things.

- **User interface:** This is where the user interacts with the system. The user can select options, give inputs, and receive outputs. The user interface is designed to keep all the functionality available in the system in some or the other way.
- **Knowledge base:** This is the pool of knowledge available in the system. Knowledge may be represented in many ways. Standard AI and soft-computing techniques are used for this purpose. This pool of knowledge can be easily queried to solve common problems per the requirements of the user.

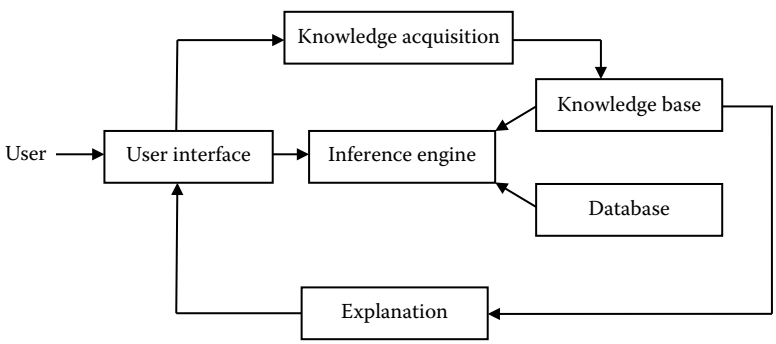


FIGURE 1.6 The architecture of an expert system.

- **Database:** This module is a collection of data. All past or historical data are stored here. The database itself has the potential to generate a huge amount of knowledge once any learning technique, or any other means, is applied over it. Analysis or learning of these data generates the knowledge used by the knowledge base.
- **Inference engine:** This module is responsible for interpreting the knowledge or its inference. It assembles and presents the output to the user in the form desired. This engine controls the system's overall working.
- **Knowledge acquisition:** This module traps data or inputs and adds them to the pool so the system can learn new things from the data that have been presented. Knowledge acquisition is responsible for making the system robust against the newer inputs. This increases the system's performance.
- **Explanation:** This module is responsible for tracing the execution of the entire process. It records how the various inferences were made and why the specific rules were fired. In expert systems, it is responsible for the reasoning behind the generation of the output from the inputs.

Once an ES is designed, we can obtain a robust system that can perform and learn autonomously. The knowledge base and database provide excellent means for drawing relevant conclusions and even for learning. Consider the example of character recognition. The expert system may provide an interface on which the user can draw some character; in return, it identifies that character. The recognized character may be output from the system, or it may be used for any specialized work the software desires.

1.4.2 Expert System Design

In this section, we briefly discuss the design issues of the expert systems. Because these systems must solve the user's needs, the foremost thing to understand is these needs. The basic objective and motive of the ES must be very clear. We further need to think about all the kinds of work that the ES is supposed to perform. The ES must be able to replace the existing system. We need to be clear about the kind of processing that the system needs and about the outputs the system must return for all types of identified user requirements.

The first major task is identification of inputs. All the inputs that can be used should be well known. We need to ensure that all inputs are relevant to the system. We also need to be clear about the way in which these inputs are to be used by the system. There must be a suitable interface for the inputs to be taken from the user. Similarly the outputs must be clearly known. We must work out the number and kind of outputs that the system is expected to produce. The outputs generated must be of some interest to the user or of relevance to the system. There also must be a good interface to present the output to the user.

We need to again look forward for the flow of data. We need to think about which system gets what data, what work the system will perform, what the outputs are, and whether the outputs would be used or processed. Doing all of this will ensure that we are able to meet all the functionalities of the system. All the data needed for a process must be made available so that the system can perform and the functionality can be met.

We further need to work out the correct means for achieving each constituent task. This includes the correct method for storing and processing knowledge, the types of AI and soft-computing tools that need to be used, and so forth. In totality, we must be able to justify the output as the best possible solution considering the problem constraints and the advantages and disadvantages of the system. We present the various techniques and the conditions for them to be used in subsequent chapters.

Once the method has been chosen, we again need to ensure that the ES design is correct. The ES must be able to perform in the given constraints and in the best possible way. It also must be cross verified. Once the implementation is done, it may be checked for performance and errors.

1.4.3 High-End Planning and Autonomous Systems

Now that we know about the building and working of expert systems, let us briefly discuss the highly intelligent autonomous systems, which require a great deal of planning and effective decision making in an autonomous mode. These autonomous robots are designed to do such tasks as chasing thieves, performing rescue operations, surveying unknown lands, and so forth. These robots require AI and soft-computing tools used at various levels to perform the given task without any help from humans.

Robots behave just like humans. Humans have the capability to do multiple things at once. For example, a person might be walking to his room and listening to music, while also answering a phone call to discuss a meeting. The ability of the human brain to do all of this at the same time without problems is a beautiful phenomenon. Unfortunately, most of the systems we develop do not actually incorporate such activities.

When it comes to the field of robotics, various activities need to be performed. These activities vary from higher-end activities, in which strategies must be made, to lower-end activities that deal with simply taking data from sensors. The higher-level activities cater to the needs of the lower-end activities. The following are a few major activities:

- **Data acquisition:** This step deals with the acquisition of data from various sensors, including infrared sensors, global positioning sensors, wireless sensors, or video and audio signals. Each sensor is used for a different purpose, but all of them give valuable information about the surroundings.
- **Processing:** Data, in the form of audio and/or video, must be processed and recognized. Recognition systems are used to extract useful information from the data. The common steps involved are preprocessing or noise removal, segmentation, feature extraction, and recognition. Even raw data from other sensors must be processed to extract useful data and to represent those data in a more useful form.
- **Map building:** This section deals with building maps from the data that have been gathered and processed. The map represents all the surroundings of the robotic world. It tells about the locations of obstacles and traversable paths. It may be represented in various forms, depending on the problem and the constraints. The most commonly used maps are geometric maps, Voronoi maps, topological maps, and hybrid maps.
- **Control:** The ways and means for controlling and moving robots are provided by the robotic controllers, which deal with the task of making the robot move along a desired path. The control uses various techniques to guide the robot and ensure that the robot's physical movement is always according to the desired plan. The controls ultimately drive the motors of the robot wheels to make them move.
- **Path planning:** For the robot to perform a task, we require it to move from some point to another. Path planning deals with the task of finding a path from a given source to a

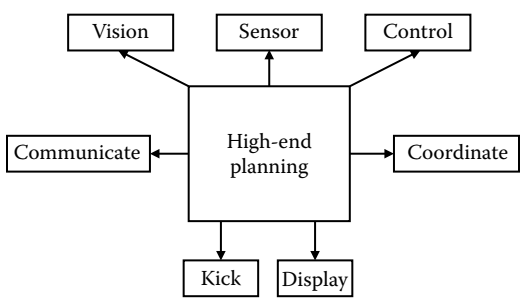


FIGURE 1.7 Software architecture for a robot.

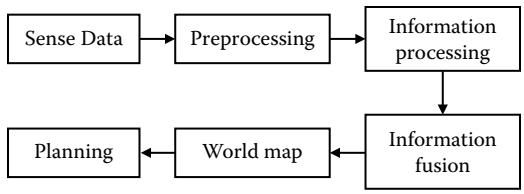


FIGURE 1.8 Architecture of the information processing and planning system in robots.

destination. Various AI and soft-computing techniques are used to find the robot's path. Some of these techniques facilitate guiding the robot in a real-time dynamic environment. If a path is possible, it is found out, and the robot is made to move over the same path.

- **Intelligent planning:** Intelligent planning deals with carrying out the job for which the robot was made. The planning consists of making real-time strategies for the robot. The robot is then guided to act on these strategies. The planning involves various AI and soft-computing techniques to make proper plans and execute them. The feedback mechanism makes the necessary adjustments or even changes the strategy. An example is the Robo Cup, in which robots are made to play a football match autonomously.
- **Multirobot coordination:** Some tasks may require more than one robot operating simultaneously. In such cases, there must be proper planning and coordination among the robots. All robots in a system need to plan and act upon a common strategy. Teamwork and cooperation are the keys to success.

  A picture of a system that requires multirobot coordination is given in Figure 1.7, which shows robots competing autonomously in a football match. The system has been made as a black box without exposing its internal functionality and will not be discussed here. Figure 1.8 shows various tasks performed by the robot for planning.

  The specific problems of path planning and navigation control are discussed in chapter 14.

1.5 TYPES OF PROBLEMS

So far we have been using the word *problems* in a very general context. Looking at various types of problems that exist in the world, we can easily say that there are three main categories of problems for which soft computing is used—classification, functional approximation, and optimization. Every real life application involves one or more of these three problems. Throughout this book, we make frequent references to these problems. Various specialized techniques and soft-computing models exist for each type of problem. First let us explain each category.

1.5.1 Classification

In classification problems, we are supposed to map the inputs to a class of outputs. The output set consists of discrete values or classes. The system is supposed to see the inputs and select one of the classes to which the input belongs. Every input can map only to a single class. Hence, once an input is presented, if the system maps it to the correct class, the answer is regarded as correct. If the system maps it to the incorrect class, the answer is incorrect.

In the case of classification problems, the system tries to learn the differences between the various classes based on the inputs presented. The differentiation among the classes is very simple if there are many differences between the two classes. However, if the two classes are very similar, it becomes very difficult to differentiate.

Before we proceed, let us give a formal definition of *classification*. According to the *Oxford English Dictionary,* classification is "the process of arranging in classes or categories or assigning to a class or category."

Definition 17
Classification is the process of arranging in classes or categories or assigning to a class or category. (Oxford English Dictionary)

The system's performance may be measured as the percentage of inputs correctly classified. It can be observed that the system's performance depends on the number of classes. Classifying an input into two classes is probably much easier than classifying the inputs into 2,000 classes. The output of the system in every case needs to be the specific class to which the system believes that input belongs.

To better explain the concept of classes and classification, let us take the example of speech recognition. Suppose we are making an automatic phone dialer that dials the number we speak. We know that the person speaking will say any one of the ten digits from 0 to 9. Hence, the output for every input can be any one of the ten digits. The system is supposed to identify the digit. Here, we can say that there are ten classes, and the classification problem is to map the input to any of the ten classes.

1.5.2 Functional Approximation

In functional approximation problems, the output is continuous and is an unknown function of the inputs. For every input, there is a specific output in the entire output space. The system is supposed to predict the unknown function that maps the inputs to the outputs. The system tries to predict the same unknown function in the learning phase. Whenever the input is provided, the system calculates the outputs. In these systems, it is very difficult for any input to get the precisely correct output. But the output given by the system is usually very close to the actual output. The closeness depends on the input given and the kind of learning the system was able to perform. The system's performance may be measured by the deviation of the actual output to the desired output. The purpose of such a system's design and learning is to make it possible to imitate the function that maps the inputs to the outputs. Here performance may be measured by the mean percent deviation or root mean square error.

Consider this example: We are trying to make a system that controls a boiler's temperature. The system takes as its input the present temperature and the quantity of fluid in the boiler. The system then outputs the fuel to be added per second to make the boiler work. The more fuel is added, the higher the heat is applied and the higher the temperature. This system is a functional approximation system that tries to map the inputs and the outputs by an otherwise unknown function.

1.5.3 Optimizations

We have already discussed optimization and optimization problems in the introduction. We only elaborate a few points here.

In optimization, we are given a function, known as the objective function. The aim is to minimize or maximize the value of the objective function by adjusting its various parameters. Each combination of parameters marks a solution that may be good or bad, depending on the value of the objective function. Soft-computing techniques generate the best possible parameter set that gives the best values of the objective function considering the time constraints.

1.6 MODELING THE PROBLEM

We talked about design issues in the discussion of expert system designs, where we introduced the design of an effective system. Here we concentrate our attention on the general modeling of the problem and the application of soft-computing techniques to that problem. We also discuss how to

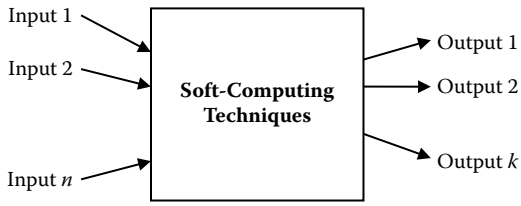


FIGURE 1.9 The black box treatment of soft-computing techniques.

go about problem solving using the soft-computing techniques described here. To begin with, let us first describe the problem.

Soft-computing problems usually involve some situation, and we need to understand that situation to accordingly generate an output. The foremost job is to think about how to represent the entire situation. We try to model the problem in a way that can be given as input to the machine in order to generate output. It is important for the machine to be presented with knowledge, data, and inputs or outputs in a form with which it can work.

Let us look at the system described in Figure 1.9. The inputs need to be preprocessed per the requirements of the problem before being used by the system. Preprocessing makes the inputs more relevant to the system. The soft-computing techniques obtain the outputs from the given input.

To understand better, let us take the example of speaker recognition. We know that the input will be in the form of voice. The output is the identity of the user. But directly taking voice as an input has numerous problems. The first problem the system faces is representation. The voice or speech must be converted to a digital format so the system can understand it; computers do not work with analog inputs. In addition, the range of the inputs must be such that the system can work with it. The second problem involves the amount of data being entered. The voice may be sampled at a high rate, and the amount of data associated with a single word may be too high. The system might not be able to work with such a high amount of data.

One problem with outputs is that it may not be easy for the system to say the speaker's name. There must be a representation such that the system can communicate the identified speaker. Hence the system's output needs to be understandable to the user. In this example, the output may be the speaker number, rather than a name.

We also need to be sure of the soft-computing method being used. We need to ensure that the method works for the data being considered given the problem constraints. The pros and cons of the various soft-computing methods should be analyzed before deciding which model to use.

1.6.1 Input

For any soft-computing technique, it is important to identify the inputs well. Too many inputs may make the system very slow. Hence, we only select those inputs that can appreciably affect the output. If our system has some feature that is not making a fine contribution to the output, that contribution may be neglected from the list of inputs. The input set contains the candidates that affect the output by a reasonable amount. Often while making the system, we may be required to guess the system inputs, because the manner in which different aspects of the problem or different inputs affect the output may not be known at all. The decision of selecting inputs is made using common sense. Let us say, for example, that we are making a system that recognizes some circular object in an image. The color may not be an important input to us, because all objects would be circular whatever color they may be.

As we explained earlier, the system may become very slow and difficult to train if the number of inputs becomes too large. Many times we may be required to adopt feature extraction techniques that extract relevant features from the situation that forms the system's input. In the speaker recognition

system, feature extraction consists of power spectral density (PSD), number of zero crossings, resonant frequencies, and so forth. It is impossible for the entire data of continuous speech to be given to the system. On doing so, the system would have faced two problems. First, the training time would have been very large. Second, there would be a problem with performance. We would need too much training data for the system to perform well. So instead we consider the entire system. Calculate PSD, number of zero crossings, resonant frequencies, and so on. Each of these has a limited number of inputs. In addition, because the collection of these features is unique for every speaker of a word, these features make an effective input for the speaker recognition system. Hence it is better to extract these features rather than give the system the complete speech. Note that we did not drop or delete any input or feature here; rather we summarized the entire data onto a few values.

Not that we have given a fair idea of feature extraction, let us provide a formal definition: In feature extraction, "features are essential attributes of an object that distinguishes it from the rest of the objects. Extracting features of an object being a determining factor in object recognition is of prime consideration" (Konar, 2000). The uses of this technique can easily be imagined in the context of practical applications of soft-computing techniques.

Definition 18
Features are essential attributes of an object that distinguishes it from the rest of the objects. Extracting features of an object being a determining factor in object recognition is of prime consideration. (Konar, 2000)

Another term of importance is *dimensionality reduction*. To understand this concept, let us first talk about the terms *dimensionality* and *input space*.

Dimensionality refers to the number of inputs or attributes used by a system in solving a problem. Suppose we have plotted the outputs against the inputs in a graph that has as many axes as the number of inputs and outputs combined. We would end up with a graph that shows the inputs and outputs in a graphical manner, allowing us to study the trends or manners in which the various outputs change with changes in inputs. Using graphs to study these changes is common—corporations make extensive use of graphs to show their profit trends over time. The problem here is that the graph has a very high number of axes (also called dimensionality), and we can only see and understand graphs up to a maximum of three dimensions. This means it is impossible for us to draw these graphs. However, it is still possible to visualize the same information using the best of our visualizing capabilities. To do this, we use a graph known as input space, which represents a high-dimensional space to show the inputs and outputs. From this concept of input space we get the word *dimensionality,* which points toward the number of inputs. We will be referring to this input space throughout the rest of this book.

For the system to work properly, many inputs need to be preprocessed to reduce the final number of inputs. This preprocessing is done by standard dimensionality reduction techniques. Dimensionality reduction maps the large number of inputs to a smaller set of inputs that can be worked with easily. Dimensionality reduction is formally defined as "the process of mapping high dimensional patterns to a lower dimensional subspace and is typically used for visualization or as a preprocessing step in classification applications" (Lotlikar and Kothari, 2000).

Definition 19
Dimensionality reduction is the process of mapping high dimensional patterns to a lower dimensional subspace and is typically used for visualization or as a preprocessing step in classification applications. (Lotlikar and Kothari, 2000)

Note that the specific technique of feature extraction or dimensionality reduction depends on the problem of study. We present some of these techniques in the subsequent units.

Now let us concentrate our attention on the quality of inputs. We know that in almost any soft-computing technique, changing the inputs by a very small amount does not change the outputs by an exceptional amount. In most cases, the change is small, which means the inputs that are very close

to each other in value have almost the same output. It may be noted, however, that whenever we use the word *small*, we actually mean a negligible difference; this difference is relative and depends on the problem. In other words, *negligible* may have different magnitudes for different problems. This is a very practical point that can be easily verified. Suppose that in the problem of speech recognition, we experiment n number of times. For each time, the readings may be different. The difference from reading to reading may even exist in consecutive experiments because of the changes in voice, even if the speaker does not change. Hence, in this case, small changes usually will not change the output appreciably.

Another important aspect is that if the same input exists any number of times, the output should always be the same. For the system inputs that we choose, these properties need to hold true in order for the system to perform well. In the example of speaker recognition, if for the set of inputs we have considered two people happen to have all features the same, then our system would fail.

It may also be observed that all soft-computing techniques require that all inputs lie within a finite range of values. It is also better if the inputs given for training are evenly distributed throughout the entire range. This means that all input values must be equally likely to occur in the input. This enables the soft-computing techniques to perform better.

1.6.2 Outputs

Output modeling is as important as input modeling. Outputs must be well known in advance. Because the system can only give valid outputs, not abstract ones, we need to decide the number of outputs and the type of every output according to the problem.

As was the case with inputs, the number of outputs must be limited to keep the complexity of the system low. A very complex system requires a huge amount of time for processing. The amount of data required for training might also be very high. Thus, it is better to limit the number of outputs.

Consider the example of a speaker recognition system. Here the output is the name of the speaker that the system believes is the speaker. The easiest way to deal with this problem is to assign a number to every speaker out of the possible set of speakers. Thus the first speaker would be 1, the second would be 2, and so on. The system then is supposed to output the speaker number (1, 2, etc.). Other methods for modeling the output of such types of classification problems are discussed later.

As was the case for inputs, the outputs must lie between finite ranges of values. It is also desirable for the outputs in the training data to cover the entire output space with equal distribution. This may help the system to perform better.

1.6.3 System

We now know the inputs and the outputs. We next need to build a system that maps the inputs to the outputs. The system may be built using any of the AI or soft-computing approaches that are covered in this text. A combination of two or more of these techniques may also be used to better cater to the needs of the problem. Once the inputs and outputs are known, designing the system using the tools and techniques presented in this text is not a very difficult task.

To get the best efficiency, however, we need to thoroughly understand the technique we are using. We need to know how it works, as well as its merits and demerits. We must use the technique that we are convinced is the best means for solving the problem. In addition, if the existing methods do not perform well, we may need to use a combination of them, along with some intelligent techniques. All of this requires a good understanding of the problem and the systems.

This book describes each method and how it works. Chapter 19 discusses the various techniques that help make an effective system design. The perfect system design may require extensive time, trying newer and newer architectures, experimenting with different scenarios, and working over different parameter values. Only after extensive experimentation may we be able to come up with a good design and validate it against standard solutions.

1.7 MACHINE LEARNING

So far we have talked a great deal about the term *learning* in relation to how a machine tries to learn. Soft computing helps the machine as it tries to memorize the things with which it is presented. This section defines the *machine learning* and all the related terminologies.

It is said that history repeats itself. Hence, it is natural requirement for any intelligent system to learn from the past. If we take a glimpse into the past, we may see many things from which we can learn. The same is true for machines. Machines learn from the past in two ways. Either they are exposed to past happenings to adaptively learn from whatever they experience. In this scenario, there is continuous learning, similar to what happens in human beings. The second form is when a great deal of data is collected and presented collectively to the machine. The machine then takes this pile of data and tries to learn the same.

Whichever method is followed, the end result is a big collection of data. The larger the data, the better the scope of learning by the machine. The data carry hidden facts, rules, and inferences that even the most intelligent humans would not be able to find out easily. The machine, when presented with this pool of data, is supposed to uncover the facts, rules, and inferences so that next time, if the same data occur, it may identify and calculate the correct answer. This capability of the machine is called *machine learning.*

Before we continue our discussion of machine learning, let us first give it a formal definition.

1.7.1 What Is Machine Learning?

Machine learning is the ability of a machine to learn from the given data and to improve its performance by learning. The motivation is for the machine to find the hidden facts in the data. By doing so, the machine will better be able to get the correct results if the same data are again given as input. The machine summarizes the entire input data into a smaller set that can be consulted to find outputs to inputs in a manageable amount of time.

Thus, machine learning may be formally defined as "computer methods for accumulating, changing, and updating knowledge in an AI computer system" (Kasabov, 1998).

Definition 20
Machine learning refers to computer methods for accumulating, changing, and updating knowledge in an AI computer system. (Kasabov, 1998)

1.7.2 Historical Database

The main focus in machine learning is to provide a good database from which the machine can learn. The underlying assumption is that the outputs can always be reproduced by some function of the inputs. Hence it is important for the system to be provided with enough data so the machine can try to predict the correct function. To a large extent, the system's performance depends on the volume of data. The data need to be sufficiently large; otherwise the machine will make false predictions that might look correct to the user. In addition, the system must be well validated against all sorts of inputs to ensure that it can cater to the needs of any future input. The diversity of the inputs is also important, and data from all sections of the input space need to be presented.

Let us take the example of a risk assessment system. Suppose the risk of fire at a place depends on the quantity of inflammables at the location and the fire extinguishers available. First we would need a great deal of historical data regarding the standard threat value for different combinations of the two factors. Suppose we have abundant data about cases in which fire extinguishers were few in number and very few data about cases in which fire extinguishers were large in number. Our system might not give a correct prediction for the cases with many fire extinguishers.

The data requirement is met by making a good database. This database is then given to the machine for learning purposes.

1.7.3 Data Acquisition

Now that we know the importance of data—and specifically good data—for the system, let us look at how these databases are built. The standard practice for good machine learning is to make a database for all types of inputs. We use means specific to the problem to collect data for the purpose of machine learning. For example, a speech recognition system requires numerous people to speak the words to be identified multiple times. All the audio clips may then be recorded and stored in the system, and the inputs and outputs may be calculated and stored in the database. In this case, we would need to conduct a survey to collect live data from people.

Many standard problems have standard databases that are available on the Internet. A few examples of these database repositories are the University of California—Irvine (UCI) Machine Learning Repository, the Time Series Data Library (TSDL), Delve data sets, and the UCI Knowledge Discovery in Databases (KDD) Archive. These databases have been built by researchers around the world and donated to be shared under public domain. These databases act as benchmark databases to analyze the performance of any soft-computing technique.

1.7.4 Pattern Matching

We have already discussed the presence of historical data that can be consulted whenever any unknown input is given. We also discussed that the system needs to give the correct answer to the inputs based on the database and the rules it generates. In this section, we briefly discuss another related term—*pattern matching.*

Pattern matching is a collection of methods that can tell us whether and to what degree two given patterns or inputs match. Pattern-matching techniques can hence be used for any recognition system in which we are trying to match any given input to the known patterns available in a database. The pattern that corresponds to the highest match is regarded as the final pattern. These techniques may also be used for verification systems in which the degree of matching decides the authenticity of the person.

For example, in a character recognition system, we have a number of ways in which all the letters can be written. Whenever any input is given, we need to match the input with the available database. This would give us the correct letter that the input represents.

Pattern recognition is a concept similar to pattern matching, which is mostly used for recognition systems. *Pattern recognition* may be defined as "the scientific discipline whose goal is the classification of objects (or patterns) into a number of categories or classes. Depending upon the problem, these objects may be images or signal waveforms or any type of measurements that need to be classified" (Theodoridis and Koutroumbas, 2006).

Definition 21
Pattern recognition is the scientific discipline whose goal is the classification of objects (or patterns) into a number of categories or classes. Depending upon the problem these objects may be images or signal waveforms or any type of measurements that need to be classified. (Theodoridis and Koutroumbas, 2006)

Pattern matching commonly uses AI and soft-computing techniques for the purpose of matching the two sets of patterns. These techniques ensure a timely response by the system, unlike other techniques that are often computationally expensive and hence slower. In most of the real life systems we discuss, soft-computing techniques are applied to extract the rules from the available data; then those rules are used for recognition purposes. This is usually better than matching the input with each and every piece of data available in the database.

Traditional pattern-matching techniques involve statistical methods in which statistical tools are used to determine the degree to which two patterns match. These statistical approaches tend to be very computationally expensive and hence are not commonly used if there are numerous

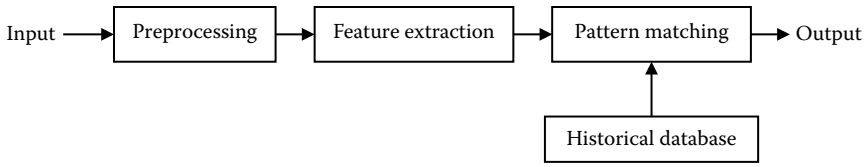


FIGURE 1.10 The various steps involved in a recognition system.

templates available for every kind of pattern in the database. This book does not cover the statistical approaches.

The general structure of the pattern-matching system is given in Figure 1.10, which shows all the steps usually followed by a recognition system. First, the system preprocesses the inputs to remove any noise. The features are then extracted. These extracted features are then matched against a historical database by pattern-matching techniques. The matching results denote the output of the system.

1.8 HANDLING IMPRECISENESS

So far we have talked about the amount of data and about the ways and means of collecting this data. Any sort of experimentation has limitations, and no matter how hard you try, data will always have some amount of impreciseness. This impreciseness can pose a serious limitation to the system. Suppose that your system was supposed to learn data. If there is too much noise, the system may learn the wrong rule, which in turn can make the system behave abnormally.

Any system will fail to perform unless the input is perfect. However, the effect of impreciseness in soft-computing techniques is actually rather low. Soft-computing techniques are appreciably resistant to noise and uncertainties prevalent in the data. They are able to perform well even when placed in highly noisy conditions. The level to which the system can handle noise and uncertainty decides the robustness of the system.

1.8.1 Uncertainties in Data

Uncertainties occur in data when we are not sure that the data presented are correct. It is possible that the data might be wrong. Because of this uncertainty, we cannot decide whether the data should be used for machine learning. If the data are wrong, the training and learning will suffer.

One example of uncertainty is when two sensors get the same data but the data do not match each other. Ideally when two sensors are used for the same reading, the data should match. If the data do not match, then there is uncertainty as to whether we should accept the data. If the data do need to be accepted, then we must determine which is correct.

A related problem is the absence of data. In this case, the data for some of the inputs are completely absent and are not available in the system. Here again, we need to consider how to use this data. A related example is when a sensor is employed to capture some data and the sensor fails to make a reading.

1.8.2 Noise

Noise in the data means that the data present in the database are not the same data present in the system. Rather the data in the database have been deformed or changed to a different value. The amount of deformation depends on the amount of noise. Noise may be present in data for various

reasons. It may be due to poor instruments used to measure data or due to mixing of unwanted signals.

Consider our example of speech recognition. If the person speaking is standing where a train can be heard in the background, the voice our system receives would be that of the speaker as well as of the train. Hence the system will not get the correct data and may behave abnormally.

Two very common forms of noise are prevalent in soft-computing systems. The first is impulse noise, in which an attribute, or a set of attributes, is highly noisy for some very small amount of data. The rest of the data has permissible noise. In our speech recognition example, the data being recorded to make the database is from people speaking. If the spoken words of one speaker got mixed with a sudden noise of dropping glass, then this recording would reveal an impulse noise. All other recordings would not have much noise.

The other form of noise occurs when some general noise is randomly distributed through the entire data set. In this case, each and every data set is noisy by some small amount, and the noise is random in nature. Consider the same example of speech recognition. If a fan in the recording room were making noise, this would induce a random noise in all the recorded data sets.

1.9 CLUSTERING

Many times data become abundant in size. Any data-analysis technique has limitations in the amount of data it can handle. Adding more data would result in the system becoming computationally slow. Hence we are often required to limit data within computational limits. Many specialized models of soft computing also face these problems of large data processing. We hence need a technique for restricting the amount of data that the system can serve. This is achieved by clustering the data.

Clustering is an effective way to limit these data. Clustering groups data into various pools. Each pool can then be represented by a representative element of that group; this element has average values of the various parameters. The entire pool may be replaced by this representative, which conveys the same information as the entire pool. In this way, we are able to limit the data without much loss of performance.

Clustering may be formally defined as "the process to partition a data set into clusters or classes, where similar data are assigned to the same cluster whereas dissimilar data should belong to different clusters" (Melin and Castillo, 2005). The distance measure, or the difference in parameters between the two clusters, measures the closeness of the clusters to each other.

Definition 22
Clustering is the process to partition a data set into clusters or classes, where similar data are assigned to the same cluster whereas dissimilar data should belong to different clusters. (Melin and Castillo, 2005)

Clustering is used very commonly in soft computing for the purpose of analysis. Clustering reduces the amount of data so that we can easily visualize, plan, and model the system. In addition, clustering reduces the system's execution time.

Now let us briefly discuss some of the commonly used clustering techniques.

1.9.1 K-Means Clustering

This algorithm takes as its input a number k, along with the data to be clustered, and in return generates k clusters, which is the number of clusters we are supposed to supply in advance. The algorithm divides the data into k clusters. The algorithm is presented in Algorithm 1.2.

Algorithm 1.2

k-Means Clustering (*k*)
C ← random *k* centers
While change in cluster centers is not negligible
Do
    For all elements *e* in data set
    Do
        Find the center c_i, which is closest to *e*
        Add *e* to cluster *i*
    For all centers c_i in *C*
    Do
        C_i ← arithmetic mean of all elements in cluster *i*

To begin, *k* random points are chosen. These points are regarded as the centers of the *k* clusters. The elements are then distributed to these clusters. Any element belongs to the cluster that is closest to it. After all the elements have been distributed in this manner, the cluster centers are modified. Each cluster has a set of elements given to it during distribution. The new cluster center lies at the mean of all the cluster members. All the *k* cluster centers are modified in this manner. We again start with the distribution of elements to the clusters. We repeatedly do this until the changes in the cluster centers become very small or the centers stop changing their positions.

1.9.2 Fuzzy C-Means Clustering

Another commonly used clustering algorithm is the fuzzy C-means clustering. This algorithm is also iterative, which means that the steps are repeated until the stopping criterion is met. The clustering improves with every step. In this algorithm, rather than associating every element with a cluster, we take a fuzzy approach in which every element is associated with every cluster by a certain degree of membership. This is why the algorithm is prefixed *fuzzy*. The degree of membership is large if the element is found near the cluster center. However, if the element is far from the cluster center, then the degree of membership is low. The algorithm is presented in Algorithm 1.3.

For any element, the sum of the degree of belongingness to all clusters is always 1, as is represented in Equation 1.1:

$$\sum_{i=1}^{k} u(e,i) = 1 \tag{1.1}$$

where $u(e, i)$ denotes the membership of the element *e* to the cluster *i*.

The center of any of the clusters is the mean of the points weighted by the degree of membership to this center. The center c_i for any cluster *i* can be given by Equation 1.2:

$$c_i = \frac{\sum_e e * u(e,i)^m}{\sum_e u(e,i)^m} \tag{1.2}$$

The degree of belongingness of any element *e* to the cluster *i* is the inverse of the distance between the element and the cluster center c_i. This is given by Equation 1.3:

$$u(e,i) = \frac{1}{\text{distance}(e,c_i)} \tag{1.3}$$

The coefficients are normalized and fuzzified with a real parameter $m > 1$ in order to make their sum 1. This degree of membership of any element *e* to cluster *i* can hence be given by Equation 1.4:

$$u(e,i) = \frac{1}{\sum_j \left(\frac{\text{distance}(c_k,e)}{\text{distance}(c_j,e)}\right) 2/(m-1)} \tag{1.4}$$

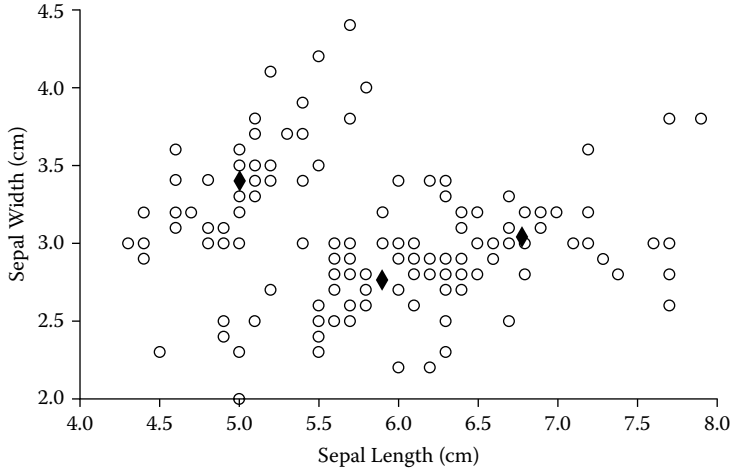


FIGURE 1.11a Fuzzy C-means clustering. *Note*: The black dots indicate centers.

1.9.3 Subtractive Clustering

In Algorithms 1.1 and 1.2, we had to give the number of clusters in advance. In subtractive clustering, however, the number of clusters does not need to be specified in advance. It thus may be used to find clusters when we do not wish to specify the number of clusters; instead, we expect the algorithm to inspect the data and decide the number of clusters on its own. This is a fast algorithm that is used to estimate the number of clusters and the cluster centers in a data set. Due to lack of space in this text and the complexity of the algorithm, we do not study this algorithm in detail.

As an example, we will use the IRIS classification data from the UCI Machine Learning Repository to carry out clustering by these three algorithms. The results are given in Figures 1.11a and 1.11b for fuzzy C-means (FCM) and subtractive clustering. The IRIS data have four features that were measured: sepal length in centimeters, sepal width in centimeters, petal length in centimeters, and petal width in centimeters. The graphs have been plotted between the first two parameters. The solid black circles denote the cluster centers. Note that here, the FCM clustering was intended to give three clusters, and hence it gave 3 clusters. The subtractive clustering gave four clusters on this database.

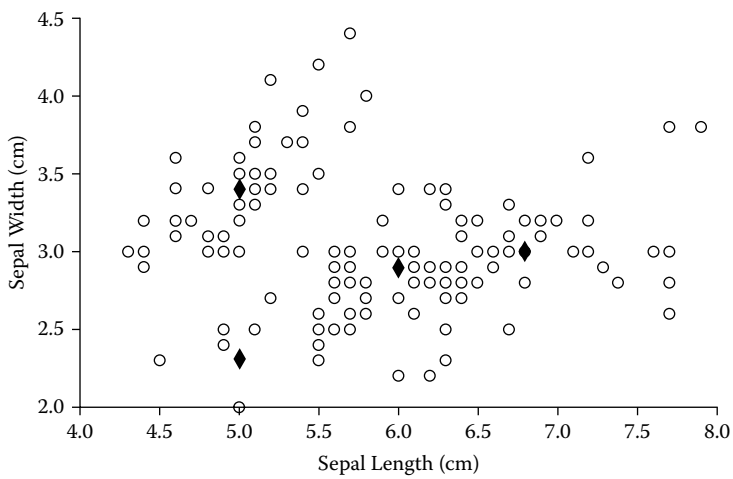


FIGURE 1.11b Subtractive clustering. *Note*: The black dots indicate centers.

1.10 HAZARDS OF SOFT COMPUTING

Every system has a liability associated with it. This is most obvious when the system fails. Every system has certain assumptions and constraints that change with time. It cannot be guaranteed that any system will continue performing without failures, as errors and bugs are prone to every system. Thus, it is possible for a system to generate wrong answers or even crash.

The case of soft computing is even more special. Even after long research, we have not been able to achieve 100 percent accuracy. This means that soft-computing systems are not to be relied upon blindly. Every soft-computing system has some small amount of error that must be considered. Normally such an error is acceptable to the user or application, and hence it does not make much of a difference. But in certain conditions, such an error can prove bad.

Say we make a diagnostic system that detects diseases in humans. Further suppose that these systems replace doctors entirely and take the job of diagnosis entirely. Now suppose a patient is wrongly diagnosed as a result of the system's limitations. The cure the patient takes might be completely irrelevant or it may prove to be disastrous for the patient. The condition is even more tragic if the presence of disease is not shown and the disease actually exists. In any case, the need for doctors to make the final conclusions would prevail.

1.11 ROAD MAP FOR THE FUTURE

Soft-computing techniques have seen research in every domain, ranging from practical applications to theoretical foundations. As a result, these techniques give much better performance and are able to solve problems very efficiently. There is, however, still a great deal of scope for things to be done in the future to enable people to better enjoy the fruits of soft computing and to tap the full potential of soft computing.

Soft computing is still far behind in providing the functionality of the human brain. The brain is still the biggest motivator and challenge for engineers working on soft computing. Both the computational power and performance of the human brain is beyond the reach of our present systems.

The existing systems also have far to go in terms of robustness. Performance is still a major issue for various systems. In addition, so many domains still remain untouched by soft computing, and soft-computing techniques are still in the growing stage in many other fields. Hence, we find a very bright future and much potential for further research into the fields of soft computing. We can surely expect many wonders in the near future as a result of this field.

1.11.1 Accuracy

The best accuracies of well-established systems may revolve around 99 percent to 99.9 percent. These accuracies are achieved when many assumptions have been imposed on a system. The accuracies of some of the more complex problems, however, are much lower. Attempts are being made to build better architectures and to extract better features in order to improve this accuracy. Simultaneous work in all the fields, be it biometrics, bioinformatics, or some other field, are being carried out in the hopes that these systems will one day be able to cross all barriers to achieve 100 percent accuracy.

Systems are also prone to noise. Many different preprocessing techniques have been devised for specific applications to compress a good amount of noise. Along with better preprocessing, better recognition systems might result in good performances in real life applications.

1.11.2 Input Limitations

It is well known that the performance of soft-computing techniques largely depends on the size of the input data set, with higher data sets being a big boon to the system. Presently all systems have a limited data set through which they are trained and through which performance is tested. The limited data-set size gives good results in experimentation but has huge limitations in real life problems. The types of data are also limited in experimentation. In real life problems, the data size needed for a system to perform might be too high.

A speaker recognition system might work well for ten speakers. But for the practical real life problem, if we increase the number of speakers to 10,000, the system might refuse to work at all. This imposes a serious issue on deployment of these systems into the real world.

1.11.3 Computational Constraints

Every system needs to perform under a given time. Even though computation power has increased rapidly over the past few decades, an increase in expectations and performance of soft-computing techniques means we need greater computation power. An exponential rise in the size of data sets and in the number of inputs has also had a deep effect on the computational requirements.

If computation power continues to grow, we may soon have systems that can perform much better in a shorter time. The increase in computation would mean a greater autonomy in the selection of features, data sets, or complexity.

1.11.4 Analogy with the Human Brain

The human brain is one of the most magnificent developments and is responsible for the intelligence of human beings. The history of humankind narrates the wonderful creations that the human brain is capable of producing. We now give a brief glimpse into the kind of system the brain is.

It is estimated that the human brain contains 50 to 100 billion neurons. Of these, about 10 billion are cortical pyramidal cells, which pass signals to each other via approximately 100 trillion synaptic connections. Even the most complex machines that we make, including our super computers, are too small in computation when compared with the brain's massive structure.

The large computation in parallel by the biological neurons is what makes it possible for the brain to do everything it does. It will take a very long time for engineers to build any machine that is comparable to the human brain. But who knows, one day artificial life and artificial humans might turn out to be a reality.

1.11.5 What to Expect in the Future

We have already discussed how performance, robustness, and computation play a major role in changing the face of soft computing in the future. We have even discussed that in the future, soft computing will move from the laboratories and into the practical areas of industry and consumer life. But there is much more that is possible in the very near future.

For years, one of the major problems with soft computing has been its multidisciplinary nature. To make effective systems, either the soft-computing experts will have to master other domains or people from other domains will have to master soft computing. This problem has prevented soft computing from entering many domains.

As a result, there is a considerable amount of work that is yet to be done by soft computing in many domains. Examples include the study of human behavior, the effects of chemicals, or the

study of nature at large. Some very positive developments may be seen in the near future when soft computing enters into these newer domains.

In the fields where soft computing has already established itself, the application of newer models may also give some great results. Many of the recently proposed models are yet to find a place in solving various problems. In many of the problems, these models may be adjusted for specific applications.

Making newer and newer models by mixing existing models is another very active field of research. We may soon find many innovative ways to handle the problems that soft-computing techniques are currently facing.

1.12 CONCLUSIONS

Soft-computing tools and techniques have already proved to be a big boon for the industry, where they have been able to solve problems that earlier had no solutions. Soft-computing techniques have found a place in almost every domain and have affected every sphere of human life. The next decade might see a complete transformation in the way society lives and functions, as soft computing moves out of the laboratories and into industry. This book is intended to be a step in the same direction.

CHAPTER SUMMARY

This chapter introduced the field of soft computing. In it, we presented various terms and concepts that are commonly used in the literature to discuss any soft-computing system. The chapter also covered the methodologies, technologies, and concepts of any real life system to fulfill the prerequisite of understanding and making real life systems that we discuss in Section II.

In this chapter, we discussed the importance of uncertainties in systems and proposed the application of uncertainties in making the most complex systems easier to engineer. We provided a formal definition for *soft computing* and described the differences between soft and hard computing. We explained why soft-computing techniques make it possible to develop systems that are impossible to develop with hard computing. We also introduced the multidisciplinary nature of soft computing and gave a brief outline of the various fields in which soft computing is being applied.

We then diverted our attention to the related field of artificial intelligence. We saw how we can model any problem in artificial intelligence and the basic solving techniques it uses. We learned about the graph search algorithm and how it is used to solve common problems. We further presented the concepts of logic and production rules and how they are used to solve real life problems.

We also introduced the various techniques that contribute to soft computing, such as artificial neural networks, evolutionary computing, fuzzy systems, and hybrid systems. We use these techniques for all real life problems. We even talked about expert systems, which are designed for end-to-end customer needs and which behave just like an expert who has complete knowledge of his or her field. We also saw how these systems are designed.

We discussed the various kinds of problems that are solved by soft-computing techniques, including functional approximation, classification, and optimization. We again saw the importance of optimal performance and how to achieve it using soft-computing techniques. We also introduced machine learning and pattern recognition.

We then highlighted the presence of impreciseness and noise and showed how these can reduce a system's performance. We highlighted the importance of systems that can handle impreciseness better, as they are more practical in real life problems.

We also saw the role of clustering in controlling the inputs and bringing them within the computation limits. We discussed the various clustering techniques and presented the hazards of soft computing. Finally, we talked about the future of soft computing in terms of better performance, better systems, and newer domains.