M.Sc. (IT) Final Year

MIT-13

ARTIFICIAL INTELLIGENCE

मध्यप्रदेश भोज (मुक्त) विश्वविद्यालय – भोपाल
MADHYA PRADESH BHOJ (OPEN) UNIVERSITY - BHOPAL

COURSE WRITERS

Dr. Angajala Srinivasa Rao, Professor and Principal, Computer Science and Engineering , Nova College of Engineering and Technology, Ibrahimpatnam, Andhra Pradesh
Units (1.0-1.3, 2.4-2.4.1, 2.9-2.14)

Dr. Preety Khatri, Assistant Professor, Computer Science, S.O.I.T., I.M.S., Noida
Units (1.3.1-1.9, 2.4.2-2.8, 3, 4, 5)

Information contained in this book has been published by VIKAS® Publishing House Pvt. Ltd. and has been obtained by its Authors from sources believed to be reliable and are correct to the best of their knowledge. However, the Madhya Pradesh Bhoj (Open) University, Bhopal, Publisher and its Authors shall in no event be liable for any errors, omissions or damages arising out of use of this information and specifically disclaim any implied warranties or merchantability or fitness for any particular use.

Published by Registrar, MP Bhoj (Open) University, Bhopal in 2020

VIKAS®

Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

SYLLABI-BOOK MAPPING TABLE

Artificial Intelligence

Syllabi	Mapping in Book
Unit – I **What is Artificial Intelligence**, Artificial Intelligence: An Introduction, AI Problems, The Underlying Assumption, AI Techniques, Games, Theorem Proving, Natural Language Processing, Vision Processing, Speech Processing, Robotics, Expert System, Search Knowledge, Abstraction.	**Unit-1:** Basics of Artificial Intelligence **(Pages 3-47)**
Unit – II **Problem, Problem Space and Search**, Defining the Problem as a State Space, Production Systems, Heuristic Search, Heuristic Search Techniques, Best-First Search, Branch-and-Bound, Problem Reduction, Constraint Satisfaction, Means-End Analysis. **Knowledge Representation**, Representation and Mapping, Approaches to Knowledge Representation, Issues in Knowledge Representation, The Frame Problem.	**Unit-2:** Problem Space, Search and Knowledge Representation **(Pages 49-130**
Unit – III **Predicate Logic**, Representing Simple Facts in Logic, Representing Instance and is a Relationships, Modus Ponens, Resolution, Natural Deduction, Dependency-Directed Backtracking, **Rule Based Systems**, Procedural versus Declarative Knowledge, Forward versus Backward Reasoning, Matching, Conflict Resolution, Use of Non Back Track,	**Unit-3:** Predicate Logic and Rule Based System **(Pages 131-163)**
Unit – IV **Structured Knowledge Representation Semantic Net**, Semantic Nets, Frames, Slots Exceptions, Slot-Values as Objects, Handling Uncertainties, Probabilistic Reasoning, Use of Certainty Factor, Fuzzy Logic	**Unit-4:** Structured Knowledge Representation and Semantic Net **(Pages 165-186)**
Unit – V **Learning,** Concept of Learning, Rote Learning, Learning by Taking Advice, Learning in Problem Solving, Learning by Induction, Explanation-Based Learning, Learning Automation, Learning in Neural Networks, **Expert Systems**, Need and Justification of Expert Systems, MYCIN, Representing and Using Domain Knowledge, RI.	**Unit-5:** Learning and Expert Systems **(Pages 187-219)**

CONTENTS

INTRODUCTION

NOTES

Artificial Intelligence (AI) is the realm of computer science that emphasizes on the creation of machines that can engage on behaviour that is considered to be intelligent according to humans. Researchers can now create systems that can mimic human thought, understand speech and perform various feats that were considered impossible earlier. The term 'Artificial Intelligence' was coined by John McCarthy in 1956 and he defined it as 'the science and engineering of making intelligent machines'. AI has become an essential part of the technology industry now, and scholars are trying to delve deeper into the field in order to provide solutions for various problems that require the intervention of machines. The central problems of AI include reasoning, knowledge, planning, learning, communication, perception and the ability to move and manipulate objects.

Technically, Artificial Intelligence (AI) is a technique that helps to create software programs to make computers perform operations that require human intelligence. Currently, AI is being used in various application areas, such as intelligent game playing, natural language processing, vision processing and speech processing. In AI, a large amount of knowledge is required to solve problems, such as natural language understanding and generation. To represent knowledge in AI, predicate logic is used, which helps represent simple facts. Artificial intelligence is also referred to as computational intelligence. Although the progress made in the field of AI is just a fraction of the computer revolution, AI has certainly helped to enhance the quality of life. The best way to explain AI to a layman is to tell him that it helps to make computers 'behave' intelligently like human beings. As a result, we now have systems that can monitor work in various production plants; or machines that can understand instructions and can be easily controlled by human beings. AI has also helped create computer programs that cannot only play chess but also defeat world champions at the game. However, it remains to be seen whether AI systems can become fast, efficient and intelligent enough to completely take the place of the human mind in any situation.

This book, *Artificial Intelligence*, follows the SIM format wherein each Unit begins with an Introduction to the topic followed by an outline of the 'Objectives'. The detailed content is then presented in a simple and an organized manner, interspersed with Answers to 'Check Your Progress' questions to test the understanding of the students. A 'Summary' along with a list of 'Key Terms' and a set of 'Self-Assessment Questions and Exercises' is also provided at the end of each unit for effective recapitulation.

UNIT 1 BASICS OF ARTIFICIAL INTELLIGENCE

NOTES

Structure

1.0 INTRODUCTION

The term 'Artificial Intelligence' was coined by John McCarthy in 1956. This term is used to describe the 'Intelligence' demonstrated by a system. It plays a key role in problem solving. Production systems help in searching for a solution to a problem. Certain heuristic algorithms have been developed to solve a problem within the scheduled time and space. Artificial Intelligence (AI) involves the task of creating intelligent computers that can perform activities similar to those performed by a human being, but more efficiently. The main objective of AI is to create an information processing theory, which can help develop intelligent computers. Currently, AI is used in various areas, such as games, natural language processing, vision processing, speech processing, robotics and expert system. Banks use software systems that are created using AI to organize operations, invest in stocks and manage property.

A natural language is a language which is written and spoken by human beings for communication. Natural languages are different from computer programming languages because they have evolved naturally while computer programming languages have been developed by human beings. There are basically two systems natural language generation system and natural language understanding system. The natural language generation involves conversion of information from computer databases into normal human language. Rule based system is the most used form of artificial intelligence in the industry, which is also known as the expert system. Rule-based system, also known as expert system or production system has immense importance in the building of knowledge system. In these systems, the domain expertise is encoded in the form of 'if–then' rules. This enables a modular portrayal of the knowledge, which facilitates its updating and maintenance.

In this unit, you will learn about the basic concept of artificial intelligence, underlying assumptions of AI, AI techniques, AI problems, games, natural language processing, vision processing, speech processing, robotics, expert systems, search knowledge and abstraction.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic concept of Artificial Intelligence (AI)
- Learn about underlying assumption of AI
- Analyse AI techniques
- Discuss about the AI problems
- Explain the application areas of AI

1.2 BASIC CONCEPTS OF ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) is the branch of computer science that deals with the creation of computers with human skills. It recognizes its surrounding and initiates actions that maximize its change of success. The term AI is used to describe the 'intelligence' that the system demonstrates. Tools and insights from fields, including linguistics, psychology, computer science, cognitive science, neuroscience, probability, optimization and logic are used. Since the human brain uses many techniques to plan and cross-check facts system integrations is also essential for AI.

Influence of AI on Communication Systems

The telephone is one of the most marvellous inventions of the communications' era. It helps in conquering the physical distance instantly.

Any telephone in the world can be accessed through a vast communication network that spans oceans and continents and the form of communication is natural, namely, human speech.

Humans communicate with a knowledge source to gather facts and is much more than two people merely talking. They communicate with intelligent systems and experts for solving problems requiring higher mental process seeking specialized opinion. They communicate with logic machines to seek guidance and to get new knowledge. Advances in communication technologies have led to increased worldwide connectivity and mobility.

Timeline of Telecommunication-Related Technology

- The development of communication systems began two centuries ago with wire-based electrical systems called telegraph and telephone. Before that human messengers on foot or horseback were used. Egypt and China built messenger relay stations.
- The electric telegraph was invented by Samuel Morse in 1831.
- Morse code was (invented by Samuel Morse in 1835) a method for transmitting telegraphic information.
- The typewriter was invented by Christopher Latham Sholes in 1867, as the first practical typewriting business office machine.
- The telephone was invented by Alexander Graham Bell in 1875. It was an instrument through which speech sounds, not voice, were first transmitted electrically.
- The telephone exchange, a rotary dialling system, became operational in New Haven, Connecticut in 1878.
- Wireless telegraphy was invented by Guglielmo Marconi in 1902; it transmitted MF band radio signals across the Atlantic Ocean, from Cornwall to Newfoundland.
- Audio vacuum tube was invented by Lee Deforest in 1906, a two-electrode detector device and later in 1908, a three electrode amplifier device too was invented.
- Cross-continental telephone call was invented by Graham Bell in 1914.
- Radios with tuners came in 1916, a technological revolution of its time.
- The iconoscope was invented by Vladimir Zworykin in 1923. It was a tube for television camera needed for TV transmission.
- The television system was invented by John Logie Baird in 1925. It's TV signals were transmitted in 1927 between London and Glasgow over the telephone line.
- Radio networks came in 1927. They distributed programms (content) to multiple stations simultaneously, in order to extend total coverage beyond the limits of a single broadcast station.

Timeline of AI-Related Technology

The timeline of AI-related technology is as follows:

- The development of artificial intelligence actually began centuries ago, long before the computer.
- Roman abacus (5000 years ago): Machine with memory.
- Pascaline (1652): Calculating machines that mechanized arithmetic.
- Difference engine (1849): Mechanical calculating machine programmed to tabulate polynomial functions.
- Boolean algebra (1854): 'Investigation of Laws of Thought' which was the symbolic language of calculus.

- Turing machine (1936): An abstract symbol-manipulating device, adapted to simulate the logic.
- Von Neumann architecture (1945): It was a computer design model with a processing unit and a shared memory structure to hold both instructions and data.
- ENIAC (1946): Electronic Numerical Integrator and Calculator, the first electronic 'general-purpose' digital computer by Eckert and Mauchly.

Timeline of AI Events

The concept of AI as a true scientific pursuit is very new. It remained a plot for popular science fiction stories over centuries. Most researchers associate the beginning of AI with Alan Turing.

- Turing test, by Alan Turing in 1950, in the paper 'Computing Machinery and Intelligence' was used to measure machine intelligence.
- Intelligent behaviour by Norbert Wiener in 1950 observed a link between human intelligence and machines and theorized intelligent behaviour.
- Logic Theorist, a program by Allen Newell and Herbert Simon in 1955, claimed that machines can contain minds just as human bodies do. It proved 38 out of the first 52 theorems in Principia Mathematica.
- AI was bornat Dartmouth Summer Research Conference on Artificial Intelligence in 1956, which was organized by John McCarthy, who is regarded as the father of AI.
- Seven years later, in 1963, AI began to pick up momentum. The field was still undefined and the ideas formed at the conference were re-examined.
- In 1957, General Problem Solver (GPS) was tested. GPS was an extension of Wiener's feedback principle and capable of solving to a great extent common sense problems.
- In 1958, LISP was invented by McCarthy and was soon adopted as the language of choice among most AI developers.
- In 1963, DoD's Advanced Research projects started at MIT. They researched on machine-aided cognition (artificial intelligence) by drawing computer scientists from around the world.
- In 1968, at MIT Micro-world program, SHRDLU controlled a robot arm.
- In the mid-1970s, expert systems for medical diagnosis (MYCIN), chemical data analysis (Dendral) and mineral exploration (Prospector) were developed.
- During the 1970s, Computer Vision (CV) technology for machines that can 'see' emerged. David Marr was the first to model the functions of the visual system.
- In 1972, Prolog a logic programming language, was invented by Alain Colmerauer. Logic programming is the use of logic in both declarative and procedural representation language.

Modern Digital Communications

In 1947, Shannon created a mathematical theory, which formed the basis for modern digital communications. Since then the developments have been as follows:

- 1960s: Three geosynchronous communication satellites were launched by NASA.
- 1961: Packet switching theory was published by Leonard Kleinrock at MIT.
- 1965: Wide-area computer network, a low speed dial-up telephone line was created by Lawrence Roberts and Thomas Merrill.
- 1966: Optical fibre was used for transmission of telephone signals.
- Late 1966: Roberts went to DARPA to develop the computer network concept and put his plan for the 'Advanced Research Projects Agency Network – ARPANET', which he presented in a conference in 1967. There Paul Baran and others at RAND presented a paper on packet switching networks for secure voice communication in military use.
- 1968: Roberts and DARPA revised the overall structure and specifications for the ARPANET, released an RFQ for development of key components; the packet switches called Interface Message Processors (IMP's).
- Architectural design by Bob Kahn.
- Network topology and economics design and optimization by Roberts with Howard Frank and his team at Network Analysis Corporation.
- Data networking technology, network measurement system preparation by Kleinrock's team at University of California, Los Angeles (UCLA).
- The year 1969 saw the beginning of the Internet era, the development of ARPANET, an unprecedented integration of capabilities of telegraph, telephone, radio, television, satellite, optical fibre and computer.

  In 1969, UCLA became the first node to join the ARPANET. That meant, the UCLA team connected the first switch (IMP) to the first host computer (a minicomputer from Honeywell).

  A month later the second node was added at SRI (Stanford Research Institute) and the first host-to-host message on the Internet was launched from UCLA. It worked in the following way:

  - Programmers for 'logon' to the SRI Host from the UCLA Host typed in 'log' and the system at SRI added 'in', thus creating the word 'login'.
  - Programmers could communicate by voice as the message was transmitted using telephone headsets at both ends.
  - Programmers at the UCLA end typed in the 'l' and asked SRI if they received it; came the voice reply 'got the l'.
- By 1969, they connected four nodes (UCLA, SRI, UC SANTA BARBARA and University of Utah). UCLA served for many years as the ARPANET Measurement Centre.
- In the mid-1970s, UCLA controlled a geosynchronous satellite by sending messages through ARPANET from California to East Coast satellite dish.

By 1970, they connected ten nodes. In 1972, the International Network Working Group, INWG was formed to further explore packet switching concepts and internetworking, as there would be multiple independent networks of arbitrary design.

- In 1973, Kahn developed a protocol that could meet the needs of an open architecture network environment. This protocol is popularly known as TCP/IP (Transmission Control Protocol/Internet Protocol).

Note: TCP/IP is named after two of the most important protocols in it.

IP is responsible for moving packet of data from node to node. TCP is responsible for verifying delivery of data from client to server. Sockets are subroutines that provide access to TCP/IP on most systems.

- In 1976, X.25 protocol was developed for public packet networking.
- In 1977, the first Internet work was demonstrated by Cerf and Kahn. They connected three networks with TCP: the Radio network, the Satellite Network (SATNET), and the Advanced Research Projects Agency Network (ARPANET).
- In the 1980s, ARPANET was evolved into INTERNET. Internet is defined officially as networks using TCP/IP.
- On 1 January 1983, the ARPANET and every other network attached to the ARPANET officially adopted the TCP/IP networking protocol. From then on, all networks that use TCP/IP are collectively known as the Internet. The standardization of TCP/IP allows the number of Internet sites and users to grow exponentially.
- Today, Internet has millions of computers and hundreds of thousands of networks. The network traffic is dominated by its ability to promote 'people-to-people' interaction.

Building Intelligent Communication Systems

Modern telecommunications are based on coordination and utilization of individual services, such as telephones, cables, microwave terrestrial and satellite and their integration into a seamless global network. Successful design, planning, coordination, management and financing of global communications network require a broad understanding of these services, their costs and their advantages and limitations. The next generation of wireless and wired communication systems and networks has requirements for many AI and AI-related concepts, algorithms, techniques and technologies. Research groups are currently exploring and using Semantic Web languages in monitoring, learning, adaptation and reasoning.

To present an idea of Intelligent Communication Systems, some examples the same are briefly illustrated herewith.

Intelligent Mobile Platform: Magitti

The mobile phone is no more a simple two-way communication device. Intelligent mobiles infer our behaviour and suggest appropriate lists of restaurants, stores and events.

The difference between Magitti and GPS-enabled mobile applications is artificial intelligence. Magitti is an intelligent personal assistant. Magitti's specification has not been released but mobile phones are becoming increasingly powerful with sensors, entertainment tools, accelerometer, GPS, etc. AI would perhaps make more sense in near future.

Voice Recognition: Lingo

Mobile phones can do a lots of things but a majority of people never use them for more than calls and short text messages. A voice recognition-correction interface across mobile phone applications is coming to market to provide speech recognition.

Project Knowledge-Based Networking: DARPA

Military research aims to develop self-configuring, secure wireless nets. Academic concepts of artificial intelligence and semantic web, combined with technologies, such as the Mobile Ad-Hoc Network (MANET), cognitive radio and peer-to-peer networking provide the elements of such a network. This project by Defence Advanced Research Products Agency (DARPA) is intended for soldiers in the field.

Semantic Web

Web is the first generation of the World Wide Web and contains static HTML pages. Web 2.0 is the second generation of the World Wide Web, focused on the ability of people to collaborate and share information online. It is dynamic, serves applications to users and offers open communications. Web requires a human operator, using computer systems to perform the tasks required to find, search and aggregate its information. It is impossible for a computer to do these tasks without human guidance because Web pages are specifically designed for human readers.

Mobile Ad-Hoc Network (MANET)

The traditional wireless mobile networks have been 'infrastructure-based' in which mobile devices communicate with access points, like base stations connected to the fixed network. Typical examples of this kind of wireless networks are GSM, WLL, WLAN; etc. Approaches to the next generation of wireless mobile networks are 'Infrastructure less'. MANET is one such network. A MANET is a collection of wireless nodes that can dynamically form a network to exchange information without using any existing fixed network infrastructure. This is very important because in many contexts information exchange between mobile units cannot rely on any fixed network infrastructure, but on rapid configuration of a wireless connections on-the-fly. The MANET is a self-configuring network of mobile routers and associated hosts connected by wireless links, the union of which forms an arbitrary topology that may change rapidly and unpredictably.

The MANET traffic includes the following:

- Peer-to-peer: Between two nodes.
- Remote-to-remote: Two nodes beyond a single hop.
- Dynamic traffic: Nodes are moving around.

Cognitive Radio

The concept of 'cognitive radio' was originally developed by DARPA scientist, Joseph Mitola in 1999 and is the 'next step up' for Software Defined Radios (SDR) that are emerging today, primarily in military applications. Most commercial radios and many two-way communication devices are hardware-based with predetermined, analog operating parameters.

A Software Defined Radio (SDR) is a radio communication system where components that have typically been in hardware (e.g., mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are implemented using software.

Beginning of AI

Although the computer provided the technology necessary for AI, it was not until the early 1950s that the link between human intelligence and machines was really observed. Norbert Wiener was one of the first Americans to make observations on the principle of feedback theory. The most familiar example of feedback theory is the thermostat. It controls the temperature of an environment by gathering the actual temperature of the house, comparing it to the desired temperature, and responding by turning the heat up or down. What was so important about Wiener's research into feedback loops was that he theorized that all intelligent behaviour was the result of feedback mechanisms—mechanisms that could possibly be simulated by machines. This discovery influenced much of the early development of AI.

In late 1955, Newell and Simon developed *The Logic Theorist*, considered by many to be the first AI program. The program, representing each problem as a tree model, would attempt to solve it by selecting the branch that would most likely result in the correct conclusion. The impact that the logic theorist made on both the public and the fields of AI has made it a crucial stepping stone in developing the AI field.

In 1956, John McCarthy, the father of AI, organized a conference to draw the talent and expertise of others interested in machine intelligence for a month of brainstorming. He invited them to Vermont for the conference of 'The Dartmouth summer research project on artificial intelligence.' From that point, because of McCarthy, the field was known as Artificial intelligence. Although not a huge success, the Dartmouth conference did bring together the founders of AI and served to lay the basis for the future of AI research.

In 1957 at Dartmouth, the New Hampshire conference discussed about the possibilities of simulating human intelligence and thinking in computers. Today, Artificial Intelligence is a well-established, natural and scaling branch of computer science. It 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, understanding language, learning, reasoning and solving problems. But AI does not have to confine itself to the methods that are biologically observable.

Some alternative definitions of artificial intelligence have been discussed ahead.

Definitions

Artificial intelligence is the study of how to make computers do things, which at the moment people do better. This is ephemeral as it refers to the current state of computer science and excludes major problem areas that cannot be solved well, either by computers or by people at the moment.

Artificial intelligence is also known as the branch of computer science that is concerned with the automation of intellectual performance. AI is based upon the principles of computer science, namely data structures used in knowledge representation, the algorithms needed to apply that knowledge and the languages and programming techniques used in their implementation.

It is a field of study that encompasses computational techniques for performing tasks that seem to require intelligence when performed by humans. It seeks to explain and follow intellectual performance in terms of computational processes.

Artificial intelligence is about generating representations and procedures that automatically or autonomously solve problems heretofore solved by humans.

There are various definitions of AI according to different authors. Most of these definitions take a very technical direction and avoid philosophical problems connected with the idea that AI's purpose is to construct an artificial human. These definitions are categorized into the following four categories:

1. Systems that think like humans (focus on reasoning and human framework).
2. Systems that think rationally (focus on reasoning and a general concept of intelligence).
3. Systems that act like humans (focus on behaviour and human framework).
4. Systems that act rationally (focus on behaviour and a general concept of intelligence).

AI Vocabulary

The following terms are most commonly used in AI vocabulary:

- **Intelligence** relates to tasks involving higher mental processes, e.g., creativity, solving problems, pattern recognition, classification, learning, induction, deduction, building analogies, optimization, language processing, knowledge and many more. Intelligence is the computational part of the ability to achieve goals.
- **Intelligent Behaviour** is depicted by perceiving one's environment, acting in complex environments, learning and understanding from experience, reasoning to solve problems and discover hidden knowledge, applying knowledge successfully in new situations, thinking abstractly, using analogies, communicating with others and more.
- **Science-Based Goals of AI** pertain to developing concepts, mechanisms and understanding biological intelligent behaviour. The emphasis is on understanding intelligent behaviour.

- **Engineering-Based Goals of AI** relate to developing concepts, theory and practice of building intelligent machines. The emphasis is on system building.
- **AI Techniques** depict how we represent, manipulate and reason with knowledge in order to solve problems. Knowledge is a collection of 'facts'. To manipulate these facts by a program, a suitable representation is required. A good representation facilitates problem solving.
- **Learning** means that programs learn from what facts or behaviour can represent. Learning denotes changes in the systems that are adaptive in other words, it enables the system to do the same task(s) more efficiently next time.
- **Applications of AI** refers to problem solving, search and control strategies, speech recognition, natural language understanding, computer vision, expert systems, etc.

Branches of AI

A list of branches of AI is given ahead. However, some branches are surely missing, because no one has identified them yet. Some of these may be regarded as concepts or topics rather than full branches.

Logical AI

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.

Search

AI programs often examine large numbers of possibilities. For example, moves in a chess game and inferences by a theorem proving program. Discoveries are frequently made about how to do this more efficiently in various domains.

Pattern Recognition

When a program makes observations of some kind, it is often planned to compare what it sees with a pattern. For example, a vision program may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns are like a natural language text, a chess position or in the history of some event. These more complex patterns require quite different methods than do the simple patterns that have been studied the most.

Representation

Usually, languages of mathematical logic are used to represent the facts about the world.

Inference

Other facts can be inferred from some facts. Mathematical logical deduction is sufficient for some purposes but new methods of non-monotonic inference have been added to the logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default. However, the conclusion can be withdrawn if there is evidence to the divergent. For example, when we hear a bird, we infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Normal logical reasoning is monotonic, in that the set of conclusions can be drawn from a set of premises, i.e., monotonic increasing function of the premises. Circumscription is another form of non-monotonic reasoning.

Common Sense Knowledge and Reasoning

This is the area in which AI is farthest from the human level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress in developing systems of non-monotonic reasoning and theories of action, yet more new ideas are needed.

Learning from Experience

There are some rules expressed in logic for learning. Programs can only learn what facts or behaviour their formalisms can represent and unfortunately almost all learning systems are based on very limited abilities to represent information.

Planning

Planning starts with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, planning programs generate a strategy for achieving the goal. In most common cases, the strategy is just a sequence of actions.

Epistemology

It is the branch of philosophy concerned with the nature and scope of knowledge. Epistemology, along with athor main subfields such as this, logic, and metaphysics, is considered a key subfield of philosophy.

Ontology

Ontology is the philosophical study of the nature of being existence or realing in general. In AI the programs and sentences deal with various kinds of objects and we study what these kinds are and what their basic properties are. Ontology assumed importance from the 1990s.

Heuristics

A heuristic is a way of trying to discover something or an idea embedded in a program. The term is used variously in AI. Heuristic functions are used in some approaches to search or to measure how far a node in a search tree seems to be from a goal. Heuristic predicates that compare two nodes in a search tree to see if one is better than the other, constitute an advance toward the goal.

Genetic Programming

Genetic programming is an automated method for creating a working computer program from a high-level problem statement. Genetic programming starts from a high-level statement of 'what needs to be done' and automatically creates a computer program to solve the problem. It is being developed by John Koza's group.

Perception

Perception is defined as 'the formation, from a sensory signal, of an internal representation suitable for intelligent processing'. Computer perception is an example of artificial intelligence. It focuses on the following:

- **Machine Vision:** It is easy to interface a TV camera to a computer and get an image into memory; the problem understands what the image represents. Vision takes *lot* of computation; in humans, roughly 10 per cent of all calories consumed are burned in vision computation.
- **Speech Understanding:** Speech understanding is available now. Some systems must be trained for the individual user and they require pauses between words. Understanding continuous speech with a larger vocabulary is harder.
- **Touch (Tactile or Haptic) sensation:** Important for robot assembly tasks.

1.2.1 Underlying Assumption of AI

Newell and Simon presented the Physical Symbol System Hypothesis, which lies in the heart of the research in artificial intelligence.

A physical symbol system consists of a set of entities called symbols, which are physical patterns that can occur as components of another entity called an expression or symbol structure. A symbol structure is a collection of instances or tokens of symbols related in some physical way. At any instant, the system will contain a collection of these symbol structures. In addition, the system also contains a collection of processes that operate on expressions to produce other expressions; processes of creation, modification, reproduction and destruction.

A physical symbol system is a machine that produces an evolving collection of symbol structures. A physical symbol system has the necessary and sufficient attributes for general intelligent action. It is only a hypothesis and there is no way to prove it other than on empirical validation.

Evidence in support of the physical symbol system hypothesis has come not only from the areas such as game playing but also from the areas such as visual perception.

The importance of the physical symbol system hypothesis is twofold. It is the major theory of human intelligence and also forms the basic belief that, it is possible to build programs that can execute intelligent tasks now performed by people.

1.2.2 AI Techniques

Artificial intelligence research during the last three decades has concluded that Intelligence requires knowledge. To compensate for the overwhelming quality, knowledge possesses the following less desirable properties:

- It is huge.
- It is difficult to characterize correctly.

- It is constantly varying.
- It differs from data by being organized in a way that corresponds to its application.
- It is complicated.

An AI technique is a method that exploits represented knowledge such that:

- The knowledge captures generalizations that share properties and these are grouped together rather than being allowed separate representation.
- The knowledge is understood by people who have provided it—even though for programs where bulk of the data comes automatically from readings.
- The knowledge can be easily modified to correct errors and reflect changes in real conditions.
- The knowledge can be widely used even if it is incomplete or inaccurate.
- The knowledge can be used to overcome its own sheer bulk by narrowing the range of possibilities that must be usually considered.

In order to characterize an AI technique let us consider initially OXO or tic-tac-toe and use a series of different approaches to play the game.

The programs increase in complexity, their use of generalizations, the clarity of their knowledge and the extensibility of their approach. In this way, they move towards being representations of AI techniques.

Tic-Tac-Toe

The Tic-Tac-Toe game consists of a nine element vector called BOARD; it represents the numbers 1 to 9 in three rows.

1	2	3
4	5	6
7	8	9

The First Approach

An element contains the value 0 for blank, 1 for X and 2 for O. A MOVETABLE vector consists of 19,683 elements (3^9) and is needed where each element is a nine element vector. The contents of the vector are especially chosen to help the algorithm. The algorithm makes moves by pursuing the following steps:

1. View the vector as a ternary number. Convert it to a decimal number.
2. Use the decimal number as an index in MOVETABLE and access the vector.
3. Set BOARD to this vector indicating how the board looks after the move. This approach is capable in time but it has several disadvantages. It takes more space and requires stunning effort to calculate the decimal numbers. This method is specific to this game and cannot be completed.

The Second Approach

The structure of the data is as before but we use 2 for a blank, 3 for an X and 5 for an O. A variable called TURN indicates 1 for the first move and 9 for the last. The algorithm consists of three actions:

(i) MAKE2 which returns 5 if the centre square is blank; otherwise it returns any blank non-corner square, i.e., 2, 4, 6 or 8.

(ii) POSSWIN (p) returns 0 if player p cannot win on the next move and otherwise returns the number of the square that gives a winning move. It checks each line using products $3 \times 3 \times 2 = 18$ gives a win for X, $5 \times 5 \times 2 = 50$ gives a win for O, and the winning move is the holder of the blank.

(iii) GO (n) makes a move to square n setting BOARD[n] to 3 or 5.

This algorithm is more involved and takes longer but it is more efficient in storage which compensates for its longer time. It depends on the programmer's skill.

The Final Approach

The structure of the data consists of BOARD which contains a nine element vector, a list of board positions that could result from the next move and a number representing an estimation of how the board position leads to an ultimate win for the player to move.

This algorithm looks ahead to make a decision on the next move by deciding which the most promising move or the most suitable move at any stage would be and selects the same.

Consider all possible moves and replies that the program can make. Continue this process for as long as time permits until a winner emerges, and then choose the move that leads to the computer program winning, if possible in the shortest time.

Actually this is most difficult to program by a good limit but it is as far that the technique can be extended to in any game. This method makes relatively fewer loads on the programmer in terms of the game technique but the overall game strategy must be known to the adviser.

Question Answering

Let us consider question answering systems that accept input in English and provide answers also in English. This problem is harder than the previous one as it is more difficult to specify the problem properly. Another area of difficulty concerns deciding whether the answer obtained is correct, or not, and further what is meant by 'correct'. For example, consider the following situation:

Rani went shopping for a new coat. She found a red one she really liked. When she got home, she found that it went perfectly with her favourite dress.

Question

1. What did Rani go shopping for?
2. What did Rani find that she liked?
3. Did Rani buy anything?

Method 1

This method can be analysed as follows.

Data Structures

A set of templates that match common questions and produce patterns used to match against inputs. Templates and patterns are used so that a template that matches a given question is associated with the corresponding pattern to find the answer in the input text. For example, the template who did **x y** generates **x y z** if a match occurs and **z** is the answer to the question. The given text and the question are both stored as strings.

Algorithm

Answering a question requires the following four steps to be followed:

- Compare the templates against the questions and store all successful matches to produce a set of text patterns.
- Pass these text patterns through a substitution process to change the person or voice and produce an expanded set of text patterns.
- Apply each of these patterns to the text; collect all the answers and then print the answers.

Example

In **question 1** we use the template WHAT DID X Y which generates

Rani go shopping for **z** and after substitution we get

Rani goes shopping for **z** and Rani went shopping for **z** giving **z** [equivalence] a new coat

In **question 2** we need a very large number of templates and also a scheme to allow the insertion of 'find' before 'that she liked'; the insertion of 'really' in the text; and the substitution of 'she' for 'Rani' gives the answer 'a red one'.

Question 3 cannot be answered.

Comments

This is a very primitive approach basically not matching the criteria we set for intelligence and worse than that, used in the game.

Method 2

This method can be analysed as follows.

Data Structures

A structure called English consists of a dictionary, grammar and some semantics about the vocabulary we are likely to come across. This data structure provides the knowledge to convert English text into a storable internal form and also to convert the response back into English. The structured representation of the text is a processed form and defines the context of the input text by making explicit all references such as pronouns. There are three types of such *knowledge representation* systems: production rules of the form 'if x then y', slot and filler systems and statements in mathematical logic. The system used here will be the slot and filler system. Take, for example sentence:

'She found a red one she really liked'.

Event 2		**Event 2**	
instance:	finding	instance:	liking
tense: past	tense:	past	
agent: Rani	modifier:	much	
object:	Thing1	object:	Thing1

Thing 1

instance:	coat
colour:	red

The question is stored in two forms: as input and in the above form.

Algorithm

- Convert the question to a structured form using English know how, then use a marker to indicate the substring (like 'who' or 'what') of the structure, that should be returned as an answer. If a slot and filler system is used a special marker can be placed in more than one slot.
- The answer appears by matching this structured form against the structured text.
- The structured form is matched against the text and the requested segments of the question are returned.

Examples

Both questions 1 and 2 generate answers via a new coat and a red coat respectively. Question 3 cannot be answered, because there is no direct response.

Comments

This approach is more meaningful than the previous one and so is more effective. The extra power given must be paid for by additional search time in the knowledge bases. A warning must be given here: that is – to generate an unambiguous English knowledge base is a complex task and must be left until later in the course. The problems of handling pronouns are difficult. For example:

Rani walked up to the salesperson: she asked where the toy department was.

Rani walked up to the salesperson: she asked her if she needed any help.

Whereas in the original text the linkage of 'she' to 'Rani' is easy, linkage of 'she' in each of the above sentences to Rani and to the salesperson requires additional knowledge about the context via the people in a shop.

Method 3

This method can be analysed as follows.

Data Structures

NOTES

World model contains knowledge about objects, actions and situations that are described in the input text. This structure is used to create integrated text from input text. Figure 1.1 shows how the system's knowledge of shopping might be represented and stored. This information is known as a script and in this case is a shopping script.

Shopping Script: C - Customer, S - Salesperson

Props: M - Merchandize, D - Money-dollars, Location: L - a Store.

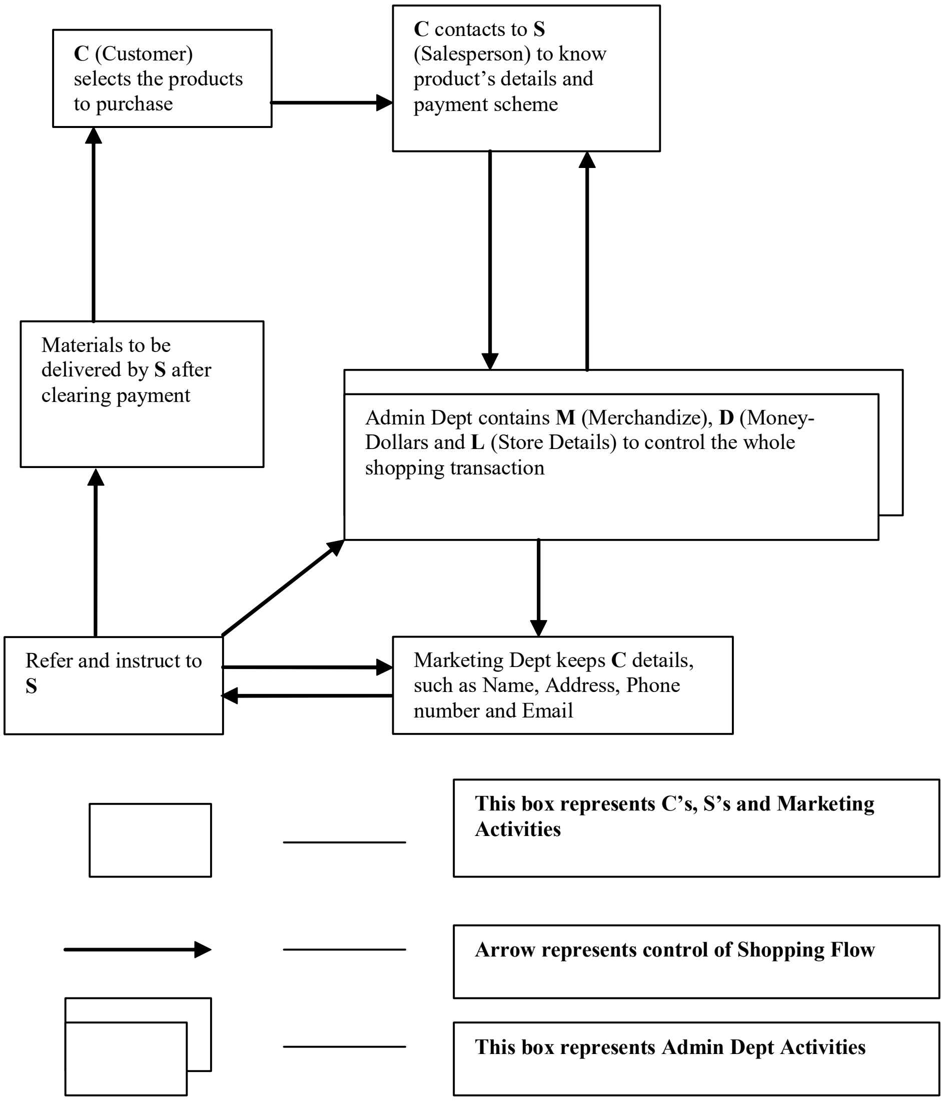

C (Customer) selects the products to purchase
C contacts to S (Salesperson) to know product's details and payment scheme
Materials to be delivered by S after clearing payment
Admin Dept contains M (Merchandize), D (Money-Dollars and L (Store Details) to control the whole shopping transaction
Refer and instruct to S
Marketing Dept keeps C details, such as Name, Address, Phone number and Email
This box represents C's, S's and Marketing Activities
Arrow represents control of Shopping Flow
This box represents Admin Dept Activities

Fig. 1.1 Diagrammatic Representation of Shopping Script

NOTES

Algorithm

Convert the question to a structured form using both the knowledge contained in Method 2 and the World model, generating even more possible structures, since even more knowledge is being used. Sometimes filters are introduced to prune the possible answers. To answer a question, the scheme followed is:

Convert the question to a structured form as before but use the world model to resolve any ambiguities that may occur. The structured form is matched against the text and the requested segments of the question are returned.

Example

Both questions 1 and 2 generate answers, as in the previous program.

Question 3 can now be answered. The shopping script is instantiated and from the last sentence the path through step 14 is the one used to form the representation.

'M' is bound to the red coat-got home. 'Rani buys a red coat' comes from step 10 and the integrated text generates that she bought a red coat.

Comments

This program is more powerful than both the previous programs because it has more knowledge. Thus, like the last game program it is exploiting AI techniques. However, we are not yet in a position to handle any English question. The major omission is that of a general reasoning mechanism known as inference to be used when the required answer is not explicitly given in the input text. But this approach can handle, with some modifications, questions of the following form with the answer—Saturday morning Rani went shopping.

Her brother tried to call her but she did not answer.

Question

Why could not Rani's brother reach her?

Answer

Because she was not in.

This answer is derived because we have supplied an additional fact that a person cannot be in two places at once.

This patch is not sufficently general so as to work in all cases and does not provide the type of solution we are really looking for.

1.3 AI PROBLEMS

Intelligence does not imply perfect understanding; every intelligent being has limited perception, memory and computation. Many points on the spectrum of intelligence versus cost are viable, from insects to humans. AI seeks to understand the computations required from intelligent behaviour and to produce computer systems that exhibit intelligence. Aspects of intelligence studied by AI include perception, communicational using human languages, reasoning, planning, learning and memory.

Let us consider some of the problems that artificial intelligence is used to solve. Early examples are game playing and theorem proving, which involves resolution. Common sense reasoning formed the basis of GPS a general problem solver. Natural language processing met with early success; then the power of the computers hindered progress, but currently this topic is experiencing a flow. The question of expert systems is interesting but it represents one of the best examples of an application of AI, which appears useful to non-AI people. Actually the expert system solves particular subsets of problems using knowledge and rules about a particular topic.

The following questions are to be considered before we can step forward:

1. What are the underlying assumptions about intelligence?
2. What kinds of techniques will be useful for solving AI problems?
3. At what level human intelligence can be modelled?
4. When will it be realized when an intelligent program has been built?

To solve the problem of building a system you should take the following steps:

1. Define the problem accurately including detailed specifications and what constitutes a suitable solution.
2. Scrutinize the problem carefully, for some features may have a central affect on the chosen method of solution.
3. Segregate and represent the background knowledge needed in the solution of the problem.
4. Choose the best solving techniques for the problem to solve a solution.

***Problem solving* is a process** of generating solutions from observed data.

- A *'problem'* is characterized by a set of *goals*,
- A set of *objects*, and
- A set of *operations*.

These could be ill-defined and may evolve during problem solving.

- A **'*problem space*'** is an abstract space.
    - A problem space encompasses all *valid states* that can be generated by the application of any combination of *operators* on any combination of *objects*.
    - The problem space may contain one or more *solutions*. A solution is a combination of *operations* and *objects* that achieve the *goals*.
- A '***search***' refers to the search for a solution in a problem space.
    - Search proceeds with different types of '*search control strategies*'.
    - The *depth first search and breadth first search* are the two common *search strategies.*

AI - General Problem Solving

Problem solving has been the key area of concern for Artificial Intelligence.

Problem solving is a process of generating solutions from observed or given data. It is however not always possible to use direct methods (i.e., go directly from data to solution). Instead, problem solving often needs to use indirect or model-based methods.

General Problem Solver (GPS) was a computer program created in 1957 by Simon and Newell to build a universal problem solver machine. *GPS* was based on Simon and Newell's theoretical work on logic machines. *GPS* in principle can solve any formalized symbolic problem, such as theorems proof and geometric problems and chess playing.

GPS solved many simple problems, such as the Towers of Hanoi, that could be sufficiently formalized, but ***GPS could not solve any real-world problems***.

To build a system to solve a particular problem, you need to take the following steps:

1. Define the problem precisely – find input situations as well as final situations for an acceptable solution to the problem
2. Analyse the problem – find few important features that may have impact on the appropriateness of various possible techniques for solving the problem
3. Isolate and represent task knowledge necessary to solve the problem
4. Choose the best problem-solving technique(s) and apply to the particular problem

Problem Definitions

A problem is defined by its '*elements*' and their '*relations*'. To provide a formal description of a problem, you need to take the following steps:

1. Define a *state space* that contains all the possible configurations of the relevant objects, including some impossible ones.
2. Specify one or more states that describe possible situations, from which the problem-solving process may start. These states are called *initial states*.
3. Specify one or more states that would be acceptable solution to the problem. These states are called *goal states*.

   Specify a set of *rules* that describe the actions (*operators*) available.

The problem can then be solved by using the *rules*, in combination with an appropriate *control strategy*, to move through the *problem space* until a *path* from an *initial state* to a *goal state* is found. This process is known as ***'search'***. Thus:

- *Search* is fundamental to the problem-solving process.
- *Search* is a general mechanism that can be used when a more direct method is not known.

- *Search* provides the framework into which more direct methods for solving subparts of a problem can be embedded. A very large number of AI problems are formulated as search problems.
- Problem space

  A *problem space* is represented by a directed graph, where *nodes* represent search state and *paths* represent the operators applied to change the *state*. To simplify search algorithms, it is often convenient to logically and programmatically represent a problem space as a **tree**. A *tree* usually decreases the complexity of a search at a cost. Here, the cost is due to duplicating some nodes on the tree that were linked numerous times in the graph, e.g., node ***B*** and node ***D.***

  A tree is a graph in which any two vertices are connected by exactly one path. Alternatively, any connected *graph with no cycles is a tree*.

  Figure 1.2(a) shows a graph and Figure 1.2(b) shows a tree.

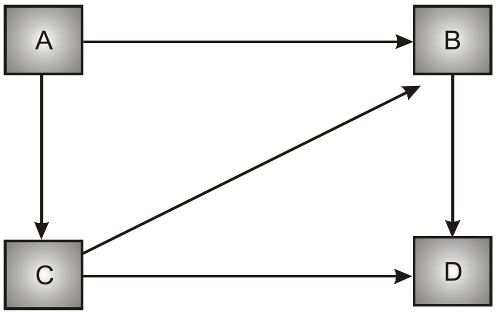


Fig. 1.2 (a) *Graph*

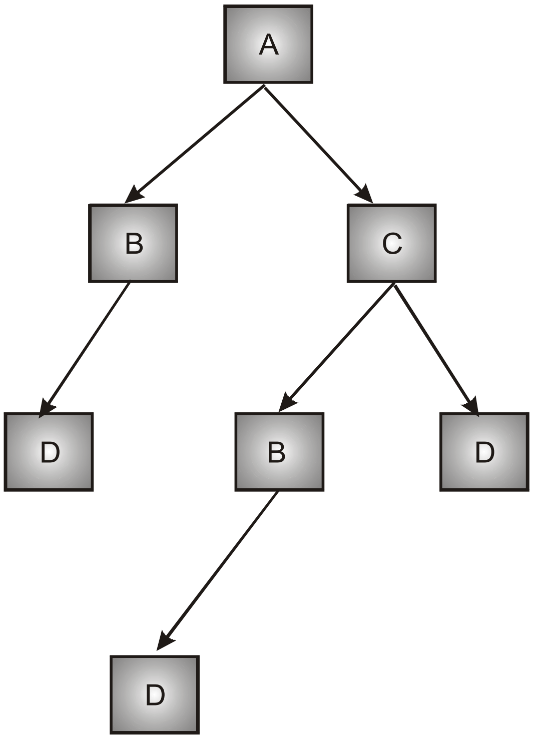


Fig. 1.2(b) *Tree*

The term, 'Problem Solving' relates to analysis in AI. Problem solving may be characterized as a systematic search through a range of possible actions to reach some predefined goal or solution. Problem-solving methods are categorized as *special purpose* and *general purpose*.

- A *special-purpose method* is tailor-made for a particular problem, often exploits very specific features of the situation in which the problem is embedded.
- A *general-purpose method* is applicable to a wide variety of problems. One General-purpose technique used in AI is *'means-end analysis'*, which is a step-by-step or incremental reduction of the difference between current state and final goal.

Problem Characteristics

Heuristics cannot be generalized, as they are domain specific. Production systems provide ideal techniques for representing such heuristics in the form of IF-THEN rules. Most problems requiring simulation of intelligence use heuristic search extensively. Some heuristics are used to define the control structure that guides the search process, as seen in the example described above. But heuristics can also be encoded in the rules to represent the domain knowledge. Since most AI problems make use of knowledge and guided search through the knowledge, AI can be described as *the study of techniques for solving exponentially hard problems in polynomial time by exploiting knowledge about problem domain*.

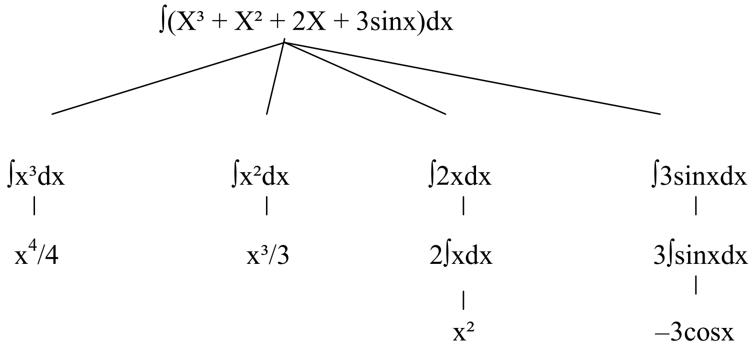
To use the heuristic search for problem solving, the problem shoul be analysed for the following considerations:

- Decomposability of the problem into a set of independent smaller subproblems.
- Possibility of undoing solution steps, if they are found to be unwise.
- Predictability of the problem universe.
- Possibility of obtaining an obvious solution to a problem without comparison of all other possible solutions.
- Type of the solution: Whether it is a state or a path to the goal state.
- Role of knowledge in problem solving.
- Nature of solution process: With or without interacting with the user.

The general classes of engineering problems such as planning, classification, diagnosis, monitoring and design are generally knowledge intensive and use a large amount of heuristics. Depending on the type of problem, the knowledge representation schemes and control strategies for search are to be adopted. Combining heuristics with the two basic search strategies have been discussed above. There are a number of other general-purpose search techniques which are essentially heuristics based. Their efficiency primarily depends on how they exploit the domain-specific knowledge to abolish undesirable paths. Such search methods are called 'weak methods', since the progress of the search depends heavily on the way the domain knowledge is exploited. A few of such search techniques which form the centre of many AI systems are briefly presented in the following sections.

Problem Decomposition

Suppose you have to solve the expression: $+\int(X^3 + X^2 + 2X + 3\sin x)dx$



This problem can be solved by breaking it into smaller problems, each of which we can solve by using a small collection of specific rules. Using this technique of problem decomposition, we can solve very large problems very easily. This can be considered as an intelligent behaviour.

Can Solution Steps Be Ignored?

Suppose we are trying to prove a mathematical theorem. First we proceed considering that proving a lemma will be useful. Later we realize that it is not at all useful. We start with another method and to prove the theorem simply ignore the first method.

Consider the 8-puzzle problem to solve. Here if we make a wrong move and realize the mistake we can go back as the control strategy keeps track of all the moves. So we can backtrack to the initial state and start with some new move.

Consider the problem of playing chess. Here, once we make a move we can never recover from that step. These problems are illustrated in the three important classes of problems mentioned below:

1. Ignorable, in which solution steps can be ignored.
   E.g.: Theorem Proving
2. Recoverable, in which solution steps can be undone.
   E.g.: 8-Puzzle
3. Irrecoverable, in which solution steps cannot be undone.
   E.g.: Chess

Is the Problem Universe Predictable?

Consider the 8-Puzzle problem. Every time we make a move, we know exactly what will happen. This means that it is possible to plan an entire sequence of moves and be confident what the resulting state will be. We can backtrack to earlier moves if they prove unwise.

Suppose we want to play Bridge. We need to plan before the first play, but we cannot play with certainty. So, the outcome of this game is very uncertain. In case of 8-Puzzle, the outcome is very certain. To solve uncertain outcome problems, we follow the process of plan revision as the plan is carried out and the necessary feedback is provided. The disadvantage is that the planning in this case is often very expensive.

Is Good Solution Absolute or Relative?

Consider the problem of answering questions based on a database of simple facts such as the following:

- Siva was a man.
- Siva was a worker in a company.
- Siva was born in 1905.
- All men are mortal.
- All workers in a factory died when there was an accident in 1952.
- No mortal lives longer than 100 years.

Suppose we ask a question; 'Is Siva alive?'

By representing these facts in a formal language, such as predicate logic, and then using formal inference methods we can derive an answer to this question easily. There are two ways to answer the question shown below:

Method I

- Siva was a man.
- Siva was born in 1905.
- All men are mortal.
- Now it is 2008, so Siva's age is 103 years.
- No mortal lives longer than 100 years.

Method II

- Siva is a worker in the company.
- All workers in the company died in 1952.

Answer: So Siva is not alive. It is the answer from the above methods.

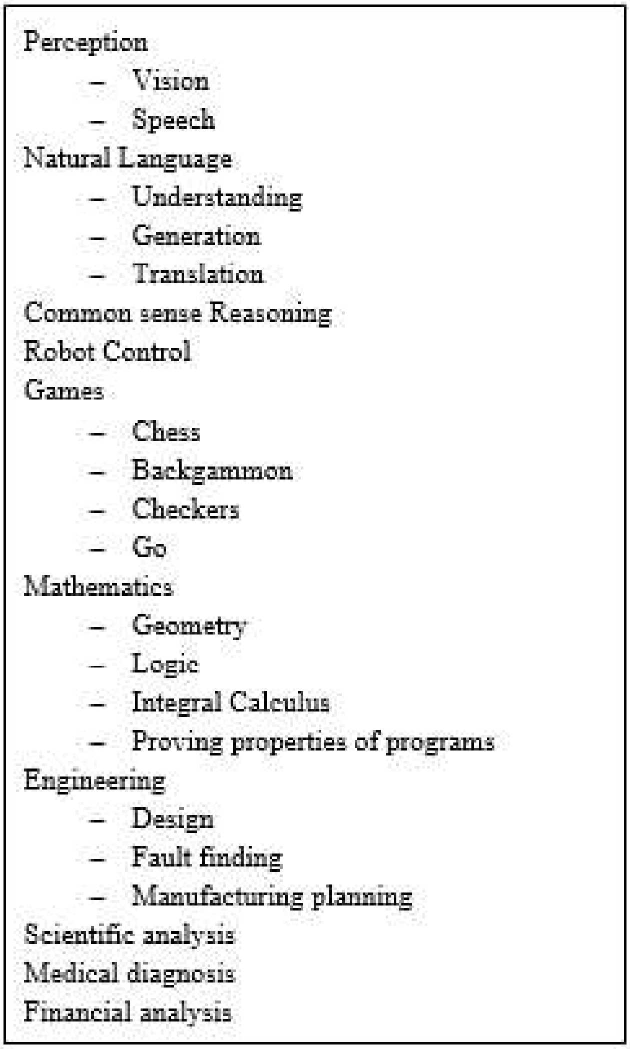
We are interested to answer the question; it does not matter which path we follow. If we follow one path successfully to the correct answer, then there is no reason to go back and check another path to lead the solution.

1.3.1 Theorem proving through AI

AI is being used to solve problems such as intelligent game playing and theorem proving using a computer system. In intelligent game playing, a computer is programmed to play a game such as chess and tic-tac-toe in the same way as human beings play. The chess game— developed by Arthur Samuel— was the first game in which AI was used for intelligent game playing. Mathematical theorems were proved using AI. The Theorem Prover system developed by Gelernter uses AI to prove geometrical theorems. Computer researchers and software developers consider that computers can be easily used with AI for intelligent game playing and theorem proving because computers are fast and can explore a large number of solution paths. After exploring the solution paths, the computers can also efficiently select the most suitable solution path for solving a problem.

In the area of decision-making, AI has been used for common sense reasoning in which reasoning about physical objects and their relationships with each other is done. Common sense reasoning also includes reasoning about actions and their consequences. AI is also used to develop software for vision processing and speech recognition. In addition, it helps to solve the problem of natural language understanding and for problem solving in specialized areas such as medical diagnosis and chemical analysis. There are also various specialized areas, such as engineering design, scientific discovery and financial planning, in which it is necessary to obtain expertise. AI can be used to create complex programs for solving problems in these specialized areas. It is easier to learn perpetual, linguistic and common sense skills than expert skills. As a result, currently AI is being used to solve problems related to areas in which only expert skills are required instead of common sense skills. Table 1.1 shows various tasks for which AI is being used.

Table 1.1 *Functions of AI*



- Perception
  - Vision
  - Speech
- Natural Language
  - Understanding
  - Generation
  - Translation
- Common sense Reasoning
- Robot Control
- Games
  - Chess
  - Backgammon
  - Checkers
  - Go
- Mathematics
  - Geometry
  - Logic
  - Integral Calculus
  - Proving properties of programs
- Engineering
  - Design
  - Fault finding
  - Manufacturing planning
- Scientific analysis
- Medical diagnosis
- Financial analysis

There are certain questions related to problem-solving through AI that need to be addressed. These questions are as follows:

- What are the AI techniques useful for solving problems?
- To model human intelligence what should be the level of detail in artificial intelligence?

- How will a developer know if an intelligent program has been developed successfully?

NOTES

Check Your Progress

1. What is ontology?
2. What are the benefits of expert systems?
3. Who created the general problem solver?
4. Define the term tree.
5. Name four problems that can be solved through AI.

1.4 APPLICATION AREAS OF AI

There are various application areas in which AI is being used these days. These application areas are as follows:

- Games
- Natural language processing
- Vision processing
- Speech processing
- Robotics
- Expert system
- Search knowledge
- Abstraction
- Learning automation systems
- Neural network

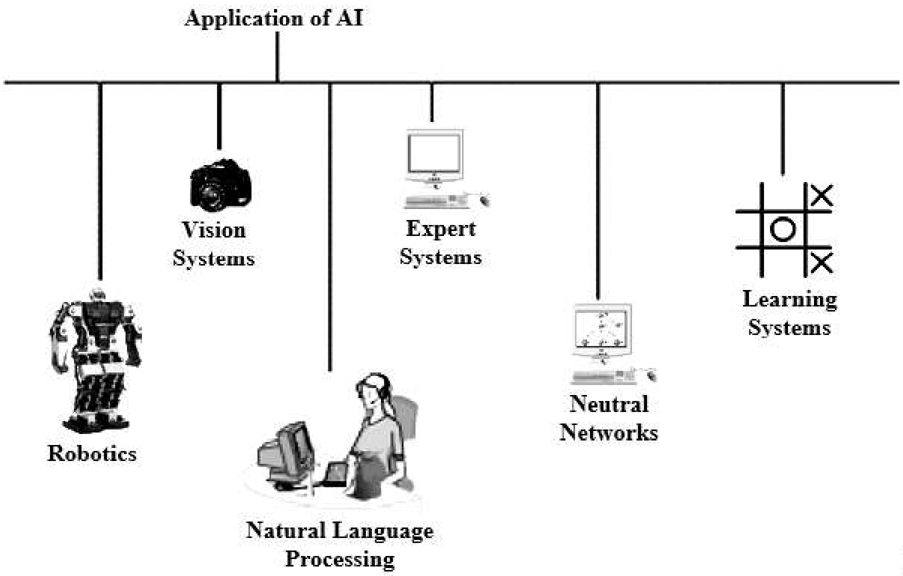


Fig. 1.1 Shows the Various Application Areas of AI

1.4.1 Games

In the game application area, a software program is created using AI, which makes the computer intelligent for playing a game. To create software programs for intelligent game playing, the developers first analyse various options and then use computers to select the best option.

In intelligent game playing, it is not necessary that a computer will always win; sometimes, the humans can also win. For example, the Deep Blue software program is created using AI that makes a computer intelligent for playing chess with humans. When a computer in which Deep Blue was installed played a game of chess against the world champion Garry Kasparov, he defeated it. However, the next time they played, the computer won the game.

1.4.2 Natural Language Processing

A natural language is a language, which is written and spoken by human beings for communication. Natural languages are different from computer programming languages because they have evolved naturally while computer programming languages have been developed by human beings. There are basically two systems, natural language generation system and natural language understanding system. The natural language generation involves conversion of information from computer databases into normal human language.

Basics of Natural Language Processing

Natural Language Processing (NLP) provides a method for interaction between computers and human beings. The interaction between human beings and computers occurs through a user interface, which is a part of the software. NLP is used to study the problem of automated generation. NLP is needed for the following reasons:

- To synthesize and recognize natural language speech.
- To make the understandable form of a natural language.
- To generate appropriate responses related to unpredictable inputs of a natural language.

Problems of Natural Language Processing

Natural language understanding involves conversion of human language into formal representations, which can be easily manipulated by computer programs. Natural language understanding is also called an AI-complete problem, because recognition of natural language requires extensive knowledge about the world and the ability to manipulate this language.

Natural language consists of many words having the same meaning and semantics. In other words, these have ambiguity at different aspects of a language, such as speech, grammar, meaning and pronunciation. Human beings can easily understand the use of ambiguous words, such as income, input and intake; however, a NLP machine cannot understand the use of these three words separately because the basic meaning of all these words is the same. This is the main problem with NLP. Given below are factors that create problems for NLP:

- The NLP system must have complete knowledge about the subjects of the sentences to be processed. For example, fruit apples can have bruises but not acts and snow falls but cities never fall.
- The NLP system should have the ability to understand the flow of different sentences involved in a story.
- The NLP system must have the ability to understand the meaning of every sentence in the correct sense.

Keyword Analysis

Keyword analysis is the checking of keywords that a visitor is using or will be using while searching for a specific web site. Natural language processors can serve as powerful AI tools for keyword analysis. These processors are nothing but algorithms that help in understanding keywords used by the visitors. Some important aspects involved in keyword analysis are morphology, synonyms, grammar and syntax. Morphology considers the different shapes a word can take, as seen in verb tense. At times, it is easy to use changes in verbs, such as come, came and coming. However, there are irregular changes, such as tell, told, bring and brought, where analyzing the keywords becomes very difficult.

Synonyms are words that have similar meaning in a given context. Many natural language processors have the capability to identify synonyms and know how to use them. The concept of syntax and grammar is also important in keyword analysis. The syntax of a keyword helps in determining the pattern of the word, while its grammar determines when the keyword is used or the context in which it is used.

Processes in Natural Language Processing

The main goal of NLP is to design and build a computer system that can be used to understand, analyze and generate natural languages, which is understood by human beings. The various processes included in NLP are as follows:

- Text-To-Speech (TTS)
- Natural Language Generation
- Machine Translation Software (MTS)
- Information Retrieval
- Text Simplification
- Automatic Summarization
- Information Extraction (IE)
- Question Answering (QA)
- Optical Character Recognition (OCR)

Text-To-Speech

A Text-To-Speech (TTS) system helps to convert text written in a specific language to speech. This process of converting language text into speech is called speech synthesis. Speech synthesis is carried out using a computer system, which is called as a speech synthesizer. Synthesized speech can be created by adding various parts of recorded speech stored in a database. The quality of the speech synthesizer is measured based on its similarity with human voice and the ability of the speech synthesizer to understand the natural language. TTS is made up of two parts, front-end and back-end. The front-end helps to convert raw text consisting of numbers and abbreviations into equivalent words. It, then, assigns different phonetic transcriptions to each word. The front-end, then, divides the text into prosodic units, such as sentences and clauses. The back-end also known as the synthesizer, then, converts these prosodic units into sound waves.

Natural Language Generation

The natural language generation can be defined as a task, which is used for generating a natural language from a machine representation system, such as a knowledge base. In natural language generation, the natural language generation system is responsible for making decisions about how to put a concept into words. The stages that occur in natural language generation are:

- Content Determination
- Discourse Planning
- Sentence Aggregation
- Lexicalization
- Referring Expression Generation
- Syntactic and Morphological Realization
- Orthographic Realization

Machine Translation Software

Machine Translation Software (MTS) checks the computer software, which is used to translate the text and speech from a given natural language into other natural languages. Machine translation substitutes atomic words of a given natural language by words used in other natural languages. MTS allows customization that helps to improve the output by limiting the scope of allowable substitutions. In machine translation process, initially the meaning of the source text is decoded. This decoded meaning is then re-encoded in the target natural language. The different types of machine translation are as follows:

- Dictionary-based machine translation
- Statistical machine translation
- Example-based machine translation
- Interlingual machine translation

Information Retrieval

Information retrieval is a process of searching data contained in documents. It also helps search metadata that is used to describe documents. The information retrieval system is related to data object and user query. The object can be defined as an entity, which stores information in the database. User queries are formal statements that contain information about the database for an object.

Text Simplification

The text simplification is an operation that is used to modify and enhance the human-readable text in such a manner that the grammar and structure of the text is simplified, while the meaning and the information remains the same. Text simplification is an important area of research since natural languages contain complex structures, which cannot be processed through the automation system.

Automatic Summarization

Automatic summarization is the process of creating a shortened version of the text with the help of a computer program. The most common type of automatic summarization is multidocument summarization. Multidocument summarization helps to extract information from multiple texts written on a single topic. It helps to create reports that are both concise and comprehensive.

Information Extraction

Information Extraction (IE) is used to extract the structured information from unstructured machine-readable documents. The main application of IE is to scan the set of documents written in the natural language and store the extracted information in a database. The different types of subtasks done by IE are as follows:

- **Named Entity Recognition**: This task is performed to recognize numerical expressions, place names, entity names and temporal expressions.
- **Co-Reference**: This task is performed to identify noun phrases, which represent a single object.
- **Terminology Extraction**: This task is performed to find the relevant terms for large and structured sets of text used to do statistical analysis.

Question Answering

Question Answering (QA) is a type of information retrieval in which a system called QA system is used to retrieve answers for questions, which are written in a natural language. QA is the most complex NLP technique as compared to other techniques. The QA systems use text documents as their knowledge source. It adds different natural language techniques to create a single processing technique and then uses the newly developed technique to search answers for the questions written in the natural language. The QA system contains a question classifier module that is used to determine the types of questions and answers.

Optical Character Recognition

Optical Character Recognition (OCR) is a computer software that helps to convert handwritten text images into machine-editable text. Text generated by OCR is provided as input to the text search databases. OCR is mainly used by libraries, businesses and government agencies to create text-searchable files for digital collections. OCR can also be used in processing cheque and credit card slips.

1.4.3 Vision Processing

In vision processing, AI is used to create software programs which allow computers to perform tasks such as mobile robot navigation, complex manufacturing tasks, analysis of satellite images and medical image processing. For vision processing, a video camera is used, which provides a computer with a visual image. The visual image is represented as a two-dimensional grid of intensity levels. Each pixel in the visual image contains either a single bit or multiple bits of information. A visual image captured through a camera consists of thousands of pixels. In vision processing, there are various operations that can be performed through a software program based on AI for the processing of visual images. These operations are as follows:

NOTES

- **Signal processing**: It allows you to enhance images and provide them as input to a vision processing software program based on AI.
- **Measurement analysis**: It allows you to determine the two-dimensional aspect for single object images.
- **Pattern recognition**: It allows you to classify a single object image into a category.
- **Image understanding**: It allows you to locate an object, classify it and develop a three-dimensional mode for images containing many objects.

In vision processing, the task of understanding an image is the most difficult task. AI researchers are currently performing research for understanding an image. Although, the main operations performed for understanding an image include pattern recognition and measurement analysis, still there are various difficulties encountered in understanding the images. These difficulties are as follows:

- Loss of information when a two-dimensional image is converted into a three-dimensional image. Due to the loss of information, understanding of the image becomes difficult.
- An image can contain multiple objects and there may be some objects, which hide other objects. This makes the understanding of the image difficult, as the AI software program for understanding the image may not know how many objects are hidden in an image.
- The value of a pixel may be affected by various image aspects such as the color of the object, source of light and distance of the camera. It is a difficult task to determine these effects on the value of a pixel.

A large amount of knowledge such as the shadows and textures of objects in an image is required to understand a low-level image. Knowledge related to the motion of objects in the image is also required for understanding an image. AI software programs for understanding images may also require knowledge about how multiple views of an object in an image are obtained. Multiple views of an object can be obtained through the use of two or more cameras and this process of getting multiple views is called stereovision. Another method of obtaining multiple views of an object is by moving objects or cameras. Information related to an image can also be obtained through the laser rangefinder, which is a device that returns an array of distance measures. However, the laser rangefinder is an expensive device so a method of integrating visual and range data can be used to acquire information related to an image. AI software programs for image understanding also require high-level knowledge about an image for interpreting visual data.

1.4.4 Speech Processing

Another important application area of AI is speech processing, which involves the processing of the spoken language. In various AI software programs such as the programs for natural language understanding, providing input through typing is not sufficient. These programs may require the users to provide data verbally. As a result, AI has been used to create software programs that make the computer intelligent enough to recognize the voice of a human being. Various software programs have been developed using AI to recognize the voice of a human being. These software programs have the following limitations:

- **Speaker dependence versus speaker independence**: Many speech recognition AI software programs are developed to recognize the voice of only a specific speaker. These programs can be modified to recognize the voice of other speakers also, but it takes a long time as these programs are complex. By using the speaker independent AI software program, the computer can be made intelligent to recognize the voice of any speaker and translate the voice command into a written text. However, it is easier to develop speaker dependent AI software programs for speech recognition instead of speaker independent programs because speaker independence is difficult to achieve due to variations in pitch and accent.
- **Continuous versus isolated word speech**: The various speech recognition AI software systems are developed to interpret an isolated word speech instead of a continuous word speech. In an isolated word speech, a speaker, who is a human being, has to pause between words while speaking. In a continuous word speech, the speaker can speak words continuously without pausing. It is easier for a human being to speak in continuous word speech instead of isolated word speech.
- **Real-time versus offline processing**: In various speech recognition AI software programs, the processing of the speech is not done when the input is being provided, but the processing is performed after some time. This is called offline processing. However, when the processing of the data is done at the same time when the input is being provided, then it is called real-time processing. In some cases, it may be required that AI software programs for speech recognition perform real-time processing. However, it is difficult to achieve real-time processing because it requires high knowledge in the AI software programs.

1.4.5 Robotics

Robotics is defined as the study of robots that helps in designing automated mechanisms, which are capable of replacing humans in certain jobs such as bolting and fitting automobile parts. The robotics system can be categorized into six types, which are similar to the six-way division of the human body functions. Table 1.2 shows the relationship between the robotics system and the human system.

***Table 1.2** Relationship between the Robotics System and the Human System*

SL No.	The Human System	The Robotics System	Functions
1	Brain	Processors	Controls the operations. Examples include computer chips and software.
2	Skin, nose, ears, taste buds	Sensors	Captures information that is recognized by human sensory organs.
3	Eyes	Vision system	Provides the ability to visualize by working with optical signals. Examples include TV cameras.
4	Hands, arms	Effectors	Provides tools for manipulating and supporting, such as hammers, poles and drills.
5	Feet	Transportation systems	Provides the mechanism of movement using wheels, propellers, etc.
6	Scientifically unidentified	Communication systems	Provides the long distance operations and communications. Examples include telephone, fax.

NOTES

A robot is composed of a series of links and joints. AI is used in robotics to create programs for instructing robots to perform repetitive tasks such as picking and placing an object. However, there is also a limitation in robotics because AI software programs are used to instruct robots to perform only a single specific task. Research is being conducted to create AI software programs that can instruct robots to perform multiple tasks.

A robot is an automatic machine, which can be programmed to perform various complex tasks. A machine must satisfy certain features to qualify as a robot. These features are as follows:

- It must have the ability to sense the environment.
- It must be capable of taking some decisions depending upon the changes in the environment.
- It must be able to move in one or more directions.
- It must obtain energy from some source to remain charged for doing work. The source of energy can either be solar or electrical.

Robotics requires in-depth practical knowledge of electronics, mechanics and computers. The manner in which a robot works can be divided into three parts: sensing, processing and action. A robot after sensing its environment processes the sensed information and then decides the further action to be taken. Sensing the environment involves detecting obstacles, pattern recognition, etc., to decide its positioning and orientation to carry out its intended task. The intelligence of sensing the environment and deciding any further action by robots has been made possible with the help of AI. Robots can be programmed to carry out heavy mechanical work, thus reducing the effort of human beings.

1.4.6 Expert Systems

Feigenbaum, one of the earliest developers of expert systems, defines an expert system as ‘An intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution’. As shown in Figure 1.4 a typical expert system has the following components:

1. **Input/Output Interface:** The input/output interface enables the users to communicate with the system using selection menus or a restricted language which is close to the natural language. This requires a system to have special prompts or specialized vocabulary including the terminology of the given domain of expertise.

   For example, MYCIN can recognize many medical terms in addition to various common words needed to communicate. For this purpose, MYCIN has vocabulary of some 2000 words.

2. **Explanation Module:** When a user requests for an explanation of the reasoning process by way of a how or why query, the explanation module provides him the answer. This brings in transparency in the reasoning process and enables the user to decide whether he agrees or disagrees with the reasoning steps presented. If he does not, then the same can be changed using the editor.

3. **Editor:** The editor is used by developers to create new rules for addition to the knowledge base, to delete outdated rules and/or to modify existing rules in some way. Some of the more sophisticated expert systems’ editors also enable the users to perform consistency tests for newly created rules, to add missing conditions to rules and/or to reformat newly created rules.

   TEIRESUIS (Davis, 1982) is an example of an intelligent editor developed to assist users in building a knowledge base, directly, without the need of an intermediary knowledge engineer.

4. **Inference Engine:** User input queries and responses to questions are accepted by the inference engine through the I/O interface. The inference engine then uses this dynamic information with the static knowledge stored in the knowledge base to arrive at inferences.

5. **Working Memory:** The execution of rules may result in placement of some new facts in working memory, a request for additional information from the user or simply stopping the search process. When appropriate knowledge is stored in the knowledge base and all required parameters values are provided by the user, conclusions are found and reported to the user. The chaining continues as long as new matches can be found between clauses in the working memory and rules in the knowledge base. When no more new rules can be placed in the conflict sets, the process is stopped.

6. **Knowledge Base:** The knowledge base contains facts and rules about some specialized knowledge domain.
7. **Learning Module:** This module uses learning algorithms to learn from usages and experience, saved in case history files. These algorithms themselves determine to a large extent how successful a learning system will be.

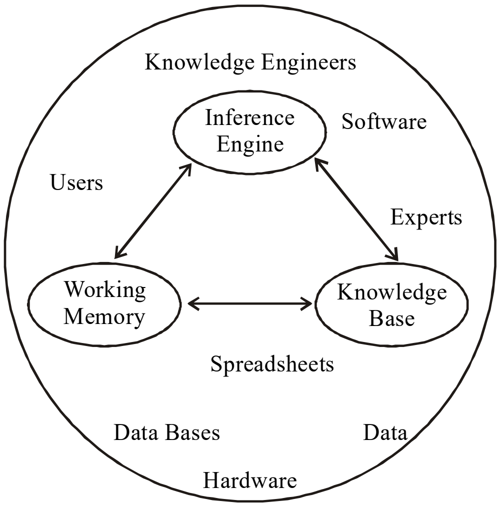


Fig. 1.4 *Simple Expert System*

Expert Systems vs. Conventional Computer Programs

Expert systems differ fundamentally from conventional computer programming systems as they treat the knowledge and inference procedures, separately. They also represent a more powerful implementation of knowledge and are able to give the end user, explanatory information on different operations or paths. Table 1.3 shows the significant differences between expert systems and conventional computer programs.

Table 1.3 *General Distinction between Expert Systems and Conventional Computer Programs*

Expert system	**Conventional Program**
Makes decisions	Calculates results.
Based on reasoning	Based on algorithms
Conducive to change	More difficult to change
Can handle uncertainty	Can not handle uncertainty
Can work with partial information, incon partial beliefs	Requires complete information
Can provide explanations of results	Gives results without explanation
Symbolic reasoning	Numerical calculations
Primarily declarative	Primarily procedural
Control and knowledge separated	Control and knowledge interlaced

Until the mid 1980's, expert systems were primarily developed using the Lisp and Prolog artificial intelligence languages. However, since these languages required long development time of about ten years, their usage has eventually decreased to a large extent. The systems developed now generally make use of expert system shell programs.

Need and Justification of Expert Systems

Nowadays, expert systems are applied to diverse fields. The need for these systems is rising mainly due to the following reasons:

1. Human beings get tired from physical or mental workload but expert systems are diligent.
2. Human beings can forget crucial details of a problem, but expert systems are programmed to take care of the minutest detail.
4. Human beings may sometimes be inconsistent or biased in their decisions, but expert systems always follow logic.
5. Human beings have limited working memory and are therefore unable to comprehend large amounts of data quickly, but expert systems can store, manipulate and retrieve large amount of data in seconds.

The various advantages, which justify the huge costs associated with experts systems, are as follows:

1. Expert systems reproduce the knowledge and skills possessed by experts. This reproduction enables wide distribution of the expertise, making it available at a reasonable cost.
2. Expert systems are always consistent in their problem-solving abilities, providing uniform answer at all times. There are no emotional or health considerations that can vary their performance.
3. Expert systems provide (almost) complete accessibility. They work 24 hours all days including weekends and holidays. They are never tired, nor do they, ever take rest.
4. Expert systems also help in preserving expertise in situations where the turnover of employees or experts is very high.
5. Expert systems are capable of solving problems even where complete or exact data do not exist. This is an important feature because complete and accurate information on a problem is rarely available in the real world.

The applications of expert systems can be categorized into the following seven major classes:

1. **Diagnosis and Troubleshooting Devices:** Expert systems can be used to deduce faults and suggest corrective actions for malfunctioning devices or processes.
2. **Planning and Scheduling:** Expert systems are used to set goals and determine a set of actions to achieve those goals. Such systems are widely used for airline scheduling of flights, manufacturing job-shop scheduling and manufacturing process planning.

3. **Configuration of Manufactured Objects from Subassemblies:** One of the most important expert system applications includes configuration, whereby a solution to a problem is synthesized from a given set of elements related by a set of constraints. The configuration technique is used in different industries like modular home building, manufacturing and complex engineering design and manufacturing.
4. **Financial Decision Making:** Expert system techniques are widely used in the financial services industry. These programs assist the bankers in determining whether to make loans to businesses and individuals. Insurance companies also use these systems to assess the risk presented by the customers and determine a price for the insurance. Expert systems are used in foreign exchange trading.
5. **Knowledge Publishing:** The primary function of expert systems used in this area is to deliver knowledge that is relevant to the user's problem. The two most widely distributed expert systems which are used for knowledge publishing are as follows: One is an advisor which counsels a user on appropriate grammatical usage in a text; the second one is a tax advisor that accompanies a tax preparation program and advises the user on tax strategy, tactics and individual tax policy.
6. **Process Monitoring and Control:** Expert systems can also be used to analyze real-time data from physical devices and notice anomalies, predict trends and control optimality and failure correction. These systems can be found in the steel making and oil refining industries.
7. **Design and Manufacturing:** These systems assist in the design of physical devices and processes, starting from high-level conceptual design of abstract entities to factory floor configuration of manufacturing processes.

Stages of Expert Systems

As shown in Figure 1.5 the development of expert systems, generally, involves the following stages:

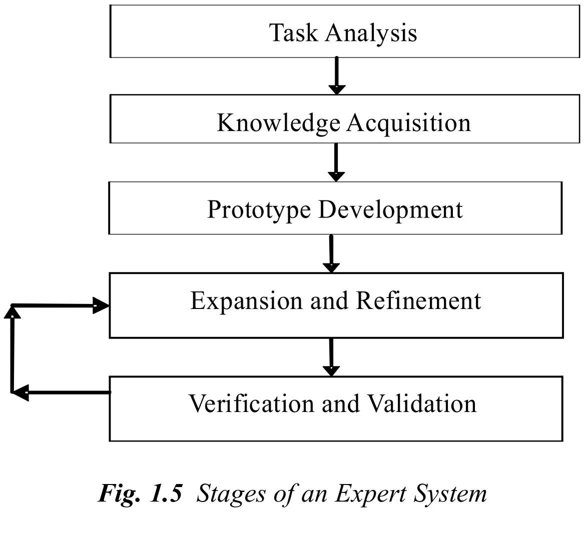


Fig. 1.5 Stages of an Expert System

1. **Task Analysis:** The first stage of developing an expert system includes identification and analysis of the problem to be solved by the knowledge engineers.
2. **Knowledge Acquisition:** The second stage involves acquiring and organizing the knowledge needed to develop an expert system. The goal of knowledge acquisition and representation is the transfer and transformation of problem-solving and decision-making expertise from a knowledge source to a form useful for developing an expert system.
3. **Prototype Development:** In this stage, knowledge expertise is transformed into a computer program. As the overall system is developed in increments, prototypes are developed for different segments or modules of the system. Only the most critical factors and most basic relationships are selected while developing prototypes.
4. **Expansion and Refinement:** In this stage, the expert adds more knowledge expertise from interviews, field observation and research publications. The prototype is reviewed repeatedly and rapidly until a sufficiently satisfactory prototype is achieved.
5. **Verification and Validation:** In this stage, the performance of the systems is evaluated. This involves testing the system in terms of effectiveness, accuracy, performance, ease of use, adaptability, adequacy, reliability and credibility. The system is also compared to the expert's prediction of the final results. This is known as validation of the system.

Expert System Architecture

The general architecture of an expert system involves two principal components; a problem dependent set of data declarations called the knowledge base or rule base and a problem independent program which is called the inference engine. The two main categories of expert system architectures are production and non-production system architectures.

Production System Architecture

One of the most common examples of the system architecture of expert system is production system. In this type of system, knowledge is represented in the form of IF-THEN-ELSE production rules. For example, IF antecedent, THEN take the consequent. The following example is taken from the knowledge base of one of the expert systems available for marketing analysis.

If: The person has good communication and written communication.

Then: The person will be considered as having ability to work as a teacher.

Each production rule in such a system represents a single piece of knowledge and sets of related production rules are used to achieve a goal. Expert systems of this type conducting a session where the systems attempt to find the best goal using information supplied by the user. The sequence of events comprises a question and answer session. The two main methods of reasoning used in this architecture are as follows:

1. **Forward Chaining:** This method involves checking the condition part of a rule to determine whether it is true or false. If the condition is true, then the action part of the rule is also true. This procedure continues until a solution is found or a dead-end is reached. Forward chaining is commonly referred to as data-driven reasoning
2. **Backward Chaining:** This is the reverse of forward chaining. It is used to backtrack from a goal to the paths that lead to the goal. It is very useful when all outcomes are known and the number of possible outcomes is not large. In this case, a goal is specified and the expert system tries to determine what conditions are needed to arrive at the specified goal. Backward chaining is thus also called goal-driven.

Non-Production System Architecture

The non production system architecture of certain expert systems do not have rule representation scheme. These systems employ more structure representation schemes like frames, decision trees or specialized networks like neural networks. Some of these architectures are discussed below.

Frame Architecture

Frames are structured sets of closely related knowledge, which may include object's or concept's names, main attributes of objects, their corresponding values and possibly some attached procedures. These values are stored in specified slots of the frame and individual frames are usually linked together.

Decision Tree Architecture

Expert system may also store information in the form of a decision tree, that is, in a top to bottom manner. The values of attributes of an object determine a path to a leaf node in the tree which contains the objects identification. Each object attribute corresponds to a non terminal node in the tree and each branch of the decision tree corresponds to a set of values. New nodes and branches can be added to the tree when additional attributes are needed to further discriminate among new objects.

Black Board System Architecture

Black board architecture is a special type of knowledge based system which uses a form of opportunistic reasoning. H. Penny Nii (1986) has aptly described the blackboard problem solving strategy through the following analogy.

'Imagine a room with a large black board on which a group of experts are piecing together a jigsaw puzzle. Each of the experts has some special knowledge about solving puzzles like border expert, shapes experts, colour expert etc. Each member examines his or her pieces and decides if they will fit into the partially completed puzzle. Those members having appropriate pieces go up to

NOTES

the black board and update the evolving solution. The whole puzzle can be solved in complete silence with no direct communication among members of the group. Each person is self activating, knowing when to contribute to the solution. The solution evolves in this incremental way with expert contributing dynamically on an opportunistic basis, that is, as the opportunity to contribute to the solution arises.

The objects in the black board are hierarchically organized into levels which facilitate analysis and solution. Information from one level serves as input to a set of knowledge sources. The sources modify the knowledge and place it on the same or different levels.'

Black boards system is applied on WEARSAY family of projects, which are speech understanding systems developed to analyse complex scenes and model the human cognitive processes.

Analogical Reasoning Architecture

Expert systems based on analogical architectures solve problems by finding similar problems and their solutions and applying the known solution to the new problem, possibly with some kind of modification.

These architectures require a large knowledge base having numerous problem solutions. Previously encountered situations are stored as units in memory and are content-indexed for rapid retrieval.

1.4.7 Search Knowledge

A large amount of knowledge is required to solve the problems related to AI. If the knowledge which is available for solving these AI problems is not enough, then a search has to be made for obtaining more knowledge in a knowledge base. The knowledge base must be systematically represented in order to efficiently search for knowledge in it. Knowledge can be represented in the form of facts in a knowledge base. A mechanism called search control knowledge can be used to control the knowledge search. In the search control knowledge, the knowledge about different paths, which can lead to a goal are obtained and reasoned. After this, the best possible path is selected to achieve the goal state.

1.4.8 Abstraction

In abstraction, some details related to AI problems are eliminated to find a solution for a problem. This process of eliminating details is continued until a solution is found. Abstraction is mainly used to solve the hard problems of an AI. Abstraction basically means hiding the unnecessary details of an AI problem. For example, the predefined sort function of the C program is used to calculate the sequence square root of a number. A programmer need not know the implementation of details of the sort function in order to use it in the C program. As a result, the implementation of details of the sort function is hidden, which is the concept of abstraction.

Check Your Progress

6. Name some application areas of AI.
7. Define an expert system.
8. Name some of the fields in which expert systems are used.
9. What is abstraction in relation to AI?

NOTES

1.5 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Ontology is the philosophical study of the nature of being existence or realing in general.
2. The benefits of expert systems are as follows:
   - Reducing the skill level needed to operate complex devices
   - Diagnostic advice for device repair
   - Interpretation of complex data
   - 'Cloning' of scarce expertise
   - Capturing knowledge of expert who is about to retire
   - Combining knowledge of multiple experts
   - Intelligent training
3. The general problem solver was created by Simon and Newell.
4. A tree is a graph in which any two vertices are connected by exactly one path.
5. The four problems that can be solved through AI are intelligent game playing, theorem proving, natural language understanding and visual image understanding.
6. Some of the application areas of AI include:
   - Natural language processing
   - Speech processing
   - Robotics
   - Expert system
   - Learning automation systems
   - Neural network
7. An expert system is a set of programs, which helps to manipulate knowledge to solve problems in specialized fields that may normally require human intelligence and expertise.
8. The various fields in which expert systems are currently being used are:
   - Aerospace
   - Military operations
   - Finance

- Banking
- Meteorology
- Geology
- Geophysics

9. Abstraction refers to the hiding of unnecessary details of an AI problem.

1.6 SUMMARY

- Artificial Intelligence (AI) is the branch of computer science that deals with the creation of computers with human skills. It recognizes its surrounding and initiates actions that maximize its change of success.
- The term AI is used to describe the 'intelligence' that the system demonstrates. Tools and insights from fields, including linguistics, psychology, computer science, cognitive science, neuroscience, probability, optimization and logic are used.
- The telephone is one of the most marvellous inventions of the communications' era. It helps in conquering the physical distance instantly.
- The development of communication systems began two centuries ago with wire-based electrical systems called telegraph and telephone. Before that human messengers on foot or horseback were used. Egypt and China built messenger relay stations.
- The concept of AI as a true scientific pursuit is very new. It remained a plot for popular science fiction stories over centuries. Most researchers associate the beginning of AI with Alan Turing.
- Perception is defined as 'the formation, from a sensory signal, of an internal representation suitable for intelligent processing'.
- AI has applications in all fields of human study, such as finance and economics, environmental engineering, chemistry, computer science and so on.
- Newell and Simon presented the Physical Symbol System Hypothesis, which lies in the heart of the research in artificial intelligence.
- Artificial intelligence research during the last three decades has concluded that Intelligence requires knowledge.
- To solve the problem of playing a game, we require the rules of the game and targets for winning as well as representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state.
- The problem solved by using the production rules in combination with an appropriate control strategy, moving through the problem space until a path from an initial state to a goal state is found.
- Solutions can be good in different ways. They can be good in terms of time or storage or in difficulty of the algorithm. In case of the travelling

salesman problem, finding the best path can lead to a significant amount of computation.

- A state-space demonstration is a mathematical structure of a physical system as a set of input, output and state variables associated by first-order differential equations. To abstract from the number of inputs, outputs and states, the variables are denoted as vectors and the differential and algebraic equations are depicted in matrix form.'
- Iterative deepening carries out repetitive depth-limited searches, beginning from zero and increasing once every time. Consequently, it has the space-saving advantages of depth first search.
- Bidirectional search is an algorithm that makes use of two searches that occur at the same time to arrive at a target goal. Bidirectional search usually seems to be an effective graph search as rather than carrying out a search through a large tree, one search is performed backwards from the goal while one search is performed forward from the beginning.
- A heuristic is a method that improves the efficiency of the search process. These are like tour guides. There are good to the level that they may neglect the points in general interesting directions; they are bad to the level that they may neglect points of interest to particular individuals.
- In the game application area, a software program is created using AI, which makes the computer intelligent for playing a game. To create software programs for intelligent game playing, the developers first analyse various options and then use computers to select the best option.
- Natural Language Processing (NLP) provides a method for interaction between computers and human beings.
- In vision processing, AI is used to create software programs which allow computers to perform tasks such as mobile robot navigation, complex manufacturing tasks, analysis of satellite images and medical image processing.
- This module uses learning algorithms to learn from usages and experience, saved in case history files. These algorithms themselves determine to a large extent how successful a learning system will be.
- Expert systems are capable of solving problems even where complete or exact data do not exist. This is an important feature because complete and accurate information on a problem is rarely available in the real world.
- A programmer need not know the implementation of details of the sort function in order to use it in the C program.

1.7 KEY TERMS

- **Ontology:** It is the philosophical study of the nature of being, existence or reality in general.

- **Epistemology:** It is the branch of philosophy concerned with the nature and scope of knowledge.
- **Heuristics:** It is a method that improves the efficiency of the search process.
- **Robotics:** The study of robots that helps in designing automated mechanisms capable of replacing human beings in certain jobs.
- **Robot:** An automatic machine programmed to perform many complex tasks.
- **Chaining:** Chaining refers to a technique of teaching, which consists of breaking a task down into small steps and thereafter teaching each step within the sequence by itself.

1.8 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are intelligent communication systems? Give some examples.
2. How do you think AI can help in authorizing financial transactions?
3. What problems are faced by AI, in general?
4. How AI can solve the problem of intelligent game playing?
5. What is of natural language processing?
6. Define the term OCR.
7. What are the operations used in vision processing?
8. What are the features that a machine needs to possess in order to qualify as a robot?
9. List the fields in which expert systems can be used.
10. What is forward and backward chaining?
11. Give the concept of abstraction.

Long-Answer Questions

1. Discuss the events which have led to the development AI with the help of examples.
2. Discuss briefly about the games with the help of examples.
3. Explain briefly about the problems of natural language processing. Give appropriate examples.
4. Explain the vision processing application area of AI with the help of examples.
5. Briefly explain about the speech processing with the help of examples and limitation.
6. Describe the robotics application area of AI.
7. Discuss the component of expert system with the help of examples.
8. Explain briefly about the abstraction. Give appropriate examples.

1.9 FURTHER READING

Russell, Stuart J. and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*, 3rd Edition. New Jersey: Prentice Hall.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco (California): Morgan Kaufmann Publishers, Inc.

Knight Kevin, Elaine Rich and B. Nair. *Artificial Intelligence (SIE)*, 3rd Edition. New Delhi: Tata McGraw-Hill.

Sivanandam, S.N. and M. Paulraj. 2009. *Introduction to Artificial Neural Networks*. New Delhi: Vikas Publishing House Pvt. Ltd.

Rich, E. and K. Knight, *Artificial Intelligence*. New York: McGraw-Hill Book Company, 1991.

LiMin, Fu. 2003. *Neural Networks in Computer Intelligence.* New Delhi: Tata McGraw-Hill.

NOTES

UNIT 2 PROBLEM SPACE, SEARCH AND KNOWLEDGE REPRESENTATION

Structure

2.0 INTRODUCTION

State space search is a process used in the field of computer science, including Artificial Intelligence (AI), in which successive configurations or states of an instance are considered, with the intention of finding a goal state with the desired property. Problems are often modelled as a state space, a set of states that a problem can be in. The set of states forms a graph where two states are connected if there is an operation that can be performed to transform the first state into the second. State space search often differs from traditional computer science search methods because the state space is implicit: the typical state space graph is much too large to generate and store in memory. Instead, nodes are generated as they are explored, and typically discarded thereafter. A solution to a combinatorial search instance may consist of the goal state itself, or of a path from some initial state to the goal state. Production systems provide appropriate structures for performing and describing search processes.

A heuristic is a method that improves the efficiency of the search process. These are like tour guides. There are good to the level that they may neglect the points in general interesting directions; they are bad to the level that they may neglect points of interest to particular individuals.

Branch and Bound (BB) is a general algorithm that is used to find optimal solutions of different optimization problems, particularly in discrete and combinatorial optimization. It contains a systematic detail of each candidate solution, in which big subsets of candidates giving no results are rejected in groups, by making use of the higher and lower approximated limits of the quantity that are undergoing optimization. A.H. Land and A.G. Doig in 1960 were the first ones to propose the method for linear programming.

Constraint satisfaction is a usual problem the goal of which is finding values for a set of variables which would satisfy a given set of constraints. It is the centre of several applications in AI, and has witnessed its implementation in several domains. These domains include planning and scheduling. Due it's usually, maximum AI researchers must be able to gain from possessing sound knowledge of methods in this field. Means End Analysis (MEA) is a strategy which is brought into use in Artificial Intelligence to control search in problem solving computer programs. It has been in use since the 1950s as a creativity tool.

Knowledge representation and reasoning is the field of Artificial Intelligence (AI) dedicated to representing information about the world in a form that a computer system can use to solve complex tasks such as diagnosing a medical condition or having a dialog in a natural language. Knowledge representation incorporates findings from psychology about how humans solve problems and represent knowledge in order to design formalisms that will make complex systems easier to design and build.

In this unit, you will learn about the search space control, production system, heuristic search, heuristic search techniques, best first search, branch and bond, problem reduction, constraint satisfaction, means end analysis, basics of knowledge representation, representation and mapping, approaches to knowledge representation, issues in knowledge representation and the frame problem.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Interpret the search space control
- Elaborate on the production system
- Define the heuristic search
- Understand the techniques of heuristic search and best first search
- Discuss about the branch and bound algorithm
- Analyse the problem reduction technique
- Explain about the constraint satisfaction problems
- Illustrate the mean end analysis strategy
- Discuss the basic concept of knowledge representation
- Elaborate on the frame problem

2.2 SEARCH SPACE CONTROL

The word 'search' refers to the search for a solution in a *problem space*.

- Search proceeds with different types of *'**search control strategies**'*.
- A strategy is defined by picking the order in which the nodes expand.

The search strategies are evaluated along the following dimensions: completeness, time complexity, space complexity and optimality.

Algorithm's Performance and Complexity

Performance of an algorithm depends on the folllowing internal and external factors.

- Time required to run
- Size of input to the algorithm
- Space (memory) required to run
- Speed of the computer
- Quality of the compiler
- *Complexity* is a measure of the performance of an algorithm. *Complexity* measures the internal factors, usually in time than space.

Computational Complexity

It is the measure of resources in terms of time and space.

- If A is an algorithm that solves a decision problem f, then run-time of A is the number of steps taken on the input of length n.
- *Time Complexity* $T(n)$ of a decision problem f is the run-time of the 'best' algorithm A for f.
- *Space Complexity* $S(n)$ of a decision problem f is the amount of memory used by the 'best' algorithm A for f.

'Big - O' Notation

The Big-O, theoretical *measure of the execution of an* algorithm, usually indicates the time or the memory needed, given the problem size n, which is usually the number of items. It is used to give an approximation to the *run-time- efficiency of an algorithm;* the letter 'O' is for order of magnitude of operations or space at run-time.

The *Big-O* of an Algorithm A

- If an algorithm A requires time proportional to $f(n)$, then algorithm A is said to be of order $f(n)$, and it is denoted as $O(f(n))$.
- If algorithm A requires time proportional to $n2$, then the order of the algorithm is said to be $O(n2)$.
- If algorithm A requires time proportional to n, then the order of the algorithm is said to be $O(n)$.

The function $f(n)$ is called the algorithm's *growth-rate function*. In other words, if an algorithm has performance complexity $O(n)$, this means that the run-time t should be directly proportional to n, ie t • n or $t = k\ n$ where k is constant of proportionality. Similarly, for algorithms having performance complexity $O(log2(n))$, $O(log\ N)$, $O(N\ log\ N)$, $O(2N)$ and so on.

Example

Determine the *Big-O* of an algorithm:

Calculate the sum of the *n* elements in an integer array *a*[*0..n-1*].

Line no.	Instructions	No. of executions steps
line 1	sum	1
line 2	for (i = 0; i < n; i++)	n + 1
line 3	sum += a[i]	n
line 4	print	sum 1
	Total	2n + 3

Thus, the polynomial (2*n* + 3) is dominated by the 1st term as *n* while the number of elements in the array becomes very large.

- In determining the *Big-O*, ignore constants such as 2 and 3. So the algorithm is of order *n.*
- So the *Big-O* of the algorithm is *O*(*n*).
- In other words the run-time of this algorithm increases roughly as the size of the input data *n*, e.g., an array of size *n*.

Example

Determine the *Big-O* of an algorithm for finding the largest element in a square 2-D array *a*[*0 . . . . . n-1*] [*0 . . . . . n-1*]

Line no	Instructions	No of execution
line 1	max = a[0][0]	1
line 2	for (row = 0; row < n; row++)	n + 1
line 3	for (col = 0; col < n; col++)	n*(n+1)
line 4	if (a[row][col] > max) max = a[row][col].	n*(n)
line 5	print max	1
	Total	$2n^2 + 2n + 3$

Thus, the polynomial ($2n^2 + 2n + 3$) is dominated by the 1st term as *n2* while the number of elements in the array becomes very large.

- In determining the *Big-O*, ignore constants such as *2, 2* and *3*. So the algorithm is of order *n2*.
- The *Big-O* of the algorithm is *O*(*n2*).
- In other words, the run-time of this algorithm increases roughly as the square *of the size of the input data* which is *n2*, e.g., an array of size *n* x *n*.

Example: Polynomial in *n* with degree *k.*

The number of steps needed to carry out an algorithm is expressed as

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n^1 + a0$$

Then $f(n)$ is a polynomial in *n* with degree *k* and $f(n) \in O(n^k)$.

- To obtain the order of a polynomial function, use the term, which is of the highest degree and disregard the constants and the terms which are of lower degrees.

The *Big-O* of the algorithm is $O(n^k)$. In other words, the run-time of this algorithm increases exponentially.

Example: Growth rates variation

Problem: If an algorithm requires *1 second run-time* for a *problem of size 8,* then find the run-time for that algorithm for the *problem of size 16*?

Solutions: If the Order of the algorithm is $O(f(n))$ then the calculated execution time $\boldsymbol{T(n)}$ of the algorithm as problem size increases are as below.

$O(f(n))$		*Run-time T(n) required as problem size increases*
$O(1)$	⇒	$T(n) = 1$ *second ;* Algorithm is *constant time*, and independent of the size of the problem.
$O(log2n)$	⇒	$T(n) = (1*log216) / log28 = 4/3$ *seconds ;* Algorithm is of *logarithmic time*, increases slowly with the size of the problem.
$O(n)$	⇒	$T(n) = (1*16) / 8 = 2$ *seconds* Algorithm is *linear time*, increases directly with the size of the problem.
$O(n*log2n)$	⇒	$T(n) = (1*16*log216) / 8*log28 = 8/3$ *seconds* Algorithm is of *log-linear time*, increases more rapidly than a linear algorithm.
$O(n2)$	⇒	$T(n) = (1*162) / 82 = 4$ *seconds* Algorithm is *quadratic time*, increases rapidly with the size of the problem.
$O(n3)$	⇒	$T(n) = (1*163) / 83 = 8$ *seconds* Algorithm is *cubic time* increases more rapidly than quadratic algorithm with the size of the problem.
$O(2n)$	⇒	$T(n) = (1*216) / 28 = 28$ *seconds* $= 256$ *seconds* Algorithm is *exponential time*, increases too rapidly to be practical.

Tree Structures used in Searching Algorithms

Tree is a way of organizing objects, related in a hierarchical fashion.

- Tree is a type of data structure in which each *element* is attached to one or more elements directly beneath it.
- The connections between elements are called *branches*.
- Tree is often called *inverted trees* because it is drawn with the *root* at the top.

- The elements that have no elements below them are called *leaves*.
- A *binary tree* is a special type: each element has only two branches below it.

Properties

The various properties of trees are as follows:

- Tree is a special case of a *graph*.
- The topmost node in a tree is called the *root node*.
- At root node all operations on the tree begin.
- A node has at most one parent.
- The topmost node (root node) has no parents.
- Each node has zero or more *child nodes*, which are below it .
- The nodes at the bottommost level of the tree are called *leaf nodes*.

  Since *leaf nodes* are at the bottom most level, they do not have children.
- A node that has a child is called the child's *parent node*.
- The *depth of a node **n*** is the length of the path from the root to the node.
- The root node is at depth zero.

Stacks and Queues

The *Stacks* and *Queues* are data structures that maintain the order of *last-in, first-out* and *first-in, first-out* respectively. Both *stacks* and *queues* are often implemented as linked lists, but that is not the only possible implementation.

Stack - Last In First Out (LIFO) lists

- An ordered list; a sequence of items, piled one on top of the other.
- The insertions and deletions are made at one end only, called Top.
- If Stack S = (a[1], a[2], ... a[n]) then a[1] is bottom most element
- Any intermediate element (a[i]) is on top of element a[i-1], 1 < i <= n.
- In Stack all operation take place on Top.

The Pop operation removes item from top of the stack.

The Push operation adds an item on top of the stack.

Queue - First In First Out (FIFO) lists

- An ordered list; a sequence of items; there are restrictions about how items can be added to and removed from the list. A queue has two ends.
- All insertions (enqueue) take place at one end, called Rear or Back
- All deletions (dequeue) take place at other end, called Front.
- If Queue has a[n] as rear element then a[i+1] is behind a[i] , 1 < i <= n.
- All operation takes place at one end of queue or the other.

The Dequeue operation removes the item at Front of the queue.

The Enqueue operation adds an item to the Rear of the queue.

NOTES

Search

Search is the systematic examination of *states* to find path from the *start / root state* to the *goal state*.

- Search usually results from a lack of knowledge.
- Search explores knowledge alternatives to arrive at the best answer.
- Search algorithm output is a solution, that is, a path from the initial state to a state that satisfies the goal test.

For general-purpose problem-solving – 'Search' is an approach.

- Search deals with finding *nodes* having certain properties in a *graph* that represents search space.
- Search methods explore the search space 'intelligently', evaluating possibilities without investigating every single possibility.

Examples

- For a Robot this might consist of PICKUP, PUTDOWN, MOVEFORWARD, MOVEBACK, MOVELEFT, and MOVERIGHT—until the goal is reached.
- Puzzles and Games have explicit rules: e.g., the 'Tower of Hanoi' puzzle.
- This puzzle involves a set of rings of different sizes that can be placed on three different pegs.
- The puzzle starts with the rings arranged as shown in Figure 2.1(a).
- The goal of this puzzle is to move them all as to Figure 2.1(b).
- Condition: Only the top ring on a peg can be moved, and it may only be placed on a smaller ring, or on an empty peg.

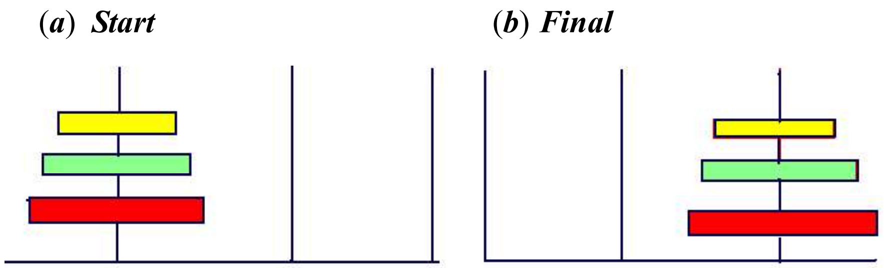


Fig. 2.1 *Tower of Hanoi Puzzle*

In this *Tower of Hanoi* puzzle, situations encountered while solving the problem are described as *states* and set of all possible configurations of rings on the pegs is called *'problem space*'.

States

A state is a representation of elements in a given moment.

A problem is defined by its *elements* and their *relations*.

At each instant of a problem, the elements have specific descriptors and relations; the *descriptors* indicate how to select elements?

Among all possible states, there are two special states called:

- *Initial state* – The start point
- *Final state* – The goal state

State Change: Successor Function

A '*successor function*' is needed for state change. The successor function moves one state to another state.

Successor Function:

- It is a description of possible actions; a set of operators.
- It is a transformation function on a state representation, which converts that state into another state.
- It defines a relation of accessibility among states.
- It represents the conditions of applicability of a state and corresponding transformation function.

State Space

A state space is the set of all states reachable from the initial state.

- A state space forms a graph (or map) in which the nodes are states and the arcs between nodes are actions.
- In a state space, a path is a sequence of states connected by a sequence of actions.
- The solution of a problem is part of the map formed by the state space.

Structure of a State Space

The structures of a state space are trees and graphs.

- A tree is a hierarchical structure in a graphical form.
- A graph is a non-hierarchical structure.
- A tree has only one path to a given node;

  i.e., a tree has one and only one path from any point to any other point.
- A graph consists of a set of nodes (vertices) and a set of edges (arcs). Arcs establish relationships (connections) between the nodes; i.e., a graph has several paths to a given node.
- The operators are directed arcs between nodes.

  A search process explores the state space. In the worst case, the search explores all possible paths between the initial state and the goal state.

Problem Solution

In the state space, a solution is a path from the initial state to a goal state or, sometimes, just a goal state.

- A solution cost function assigns a numeric cost to each path; it also gives the cost of applying the operators to the states.
- A solution quality is measured by the path cost function; and an optimal solution has the lowest path cost among all solutions.

- The solutions can be any or optimal or all.
- The importance of cost depends on the problem and the type of solution asked.

Problem Description

A problem consists of the description of the following:

- The current state of the world.
- The actions that can transform one state of the world into another.
- The desired state of the world.

  The following action are taken to describe the problem:

  - o State space is defined explicitly or implicitly

    A state space should describe everything that is needed to solve a problem and nothing that is not needed to solve the problem.

  - o Initial state is start state
  - o Goal state is the conditions it has to fulfill.

  The description by a desired state may be complete or partial.

  - o Operators are to change state
  - o Operators do actions that can transform one state into another;
  - o Operators consist of; Preconditions and Instructions

    Preconditions provide partial description of the state of the world that must be true in order to perform the action, and

    Instructions tell the user how to create the next state.

- Operators should be as general as possible, so as to reduce their number.
- *Elements of the domain* has relevance to the problem
  - o Knowledge of the starting point.
- *Problem solving* is finding a solution
  - o Find an ordered sequence of operators that transform the current (start) state into a goal state.
- *Restrictions* are solution quality any, optimal, or all
  - o Finding the shortest sequence, or
  - o finding the least expensive sequence defining cost, or
  - o finding any sequence as quickly as possible.

This can also be explained with the help of algebraic function as given below.

Algebraic Function

A function may take the form of a set of ordered pair, a graph or a equation. Regardless of the form it takes, a function must obey the condition that, no two of its ordered pairs have the same first member with different second members.

Relation: A set of ordered pair of the form (x, y) is called a relation.

Function: A relation in which no two ordered pairs have the same *x-value* but different *y-value* is called a function. Functions are usually named by lower-case letters such as *f, g,* and *h*.

For example, *f* = {(*-3, 9*), (*0, 0*), (*3, 9*)},   g = {(*4, -2*), (*2, 2*)}

Here **f** is a function, *g* is not a function.

Domain and Range: The *domain* of a function is the set of all the first members of its ordered pairs, and the *range* of a function is the set of all second members of its ordered pairs.

if function *f* = {(*a, A*), (*b, B*), (*c, C*)},then its domain is {*a, b, c* } and its range is {*A, B, C*}.

Function and Mapping: A function may be viewed as a mapping or a pairing of one set with *elements* of a second set such that each element of the first set (called *domain*) is paired with exactly one *element* of the second set (called *co domain*)., e.g., if a function *f* maps {*a, b, c*} into {*A, B, C, D*} such that *a* ’! *A* (read ‘ *a* is mapped into *A*’), *b* ’! *B, c* ’! *C* then the domain is {*a, b, c*} and the co domain is {*A, B, C, D*}. Since *a* is paired with *A* in *co domain*, *A* is called the image of *a*. Each element of the *co domain* that corresponds to an element of the domain is called the image of that element.

The set of image points, {*A, B, C*}, is called the range. Thus, the range is a subset of the *co domain*.

Onto Mappings: *Set A* is mapped onto *set B* if each element of *set B* is image of an element of a *set A*. Thus, every function maps its domain onto its range.

Describing a Function by an Equation: The rule by which each *x-value* gets paired with the corresponding *y-value* may be specified by an equation. For example, the function described by the equation *y* = *x* + *1* requires that for any choice of *x* in the domain, the corresponding range value is *x* + *1*. Thus, *2* ’! *3, 3* ’! *4*, and *4* ’! *5*.

Restricting Domains of Functions: Unless otherwise indicated, the domain of a function is assumed to be the largest possible set of real numbers. Thus: The domain of *y* = *x* / (*x2* - *4*) is the set of all real numbers except ± *2* since for these values of ***x*** the denominator is *0*.

The domain of *y* = (*x* - *1*)*1/2* is the set of real numbers greater than or equal to *1* since for any value of ***x*** less than *1*, the root radical has a negative radicand so the radical does not represent a real number.

Example: Find which of the relation describe function?

(a) *y* = *x1/2* , (b) *y* = *x3* , (c) *y* > *x* , (d) *x* = *y2*

Equations (a) and (b) produce exactly one value of ***y*** for each value of ***x***. Hence, equations (a) and (b) describe functions.

The equation (c), *y* > *x* does not represent a function since it contains ordered pair such as (1, 2) and (1, 3) where same value of x is paired with different values of *y*.

The equation (*d*), *x* = *y2* is not a function since ordered pair such as (4, 2) and (4, -2) satisfy the equation but have the same value of *x* paired with different values of *y.*

Function Notation: For any function *f,* the value of *y* that corresponds to a given value of *x* is denoted by *f*(*x*).

If $y = 5x - 1$, then $f(2)$, read as *'f of 2'*, represents the value of y.
When $x = 2$, then $f(2) = 5 . 2 - 1 = 9;$
when $x = 3$, then $f(3) = 5 . 3 - 1 = 14;$
In an equation that describes function f, then $f(x)$ may be used in place of y, for example, $f(x) = 5x - 1$. *If* $y = f(x)$, then y is said to be a function of x.
Since the value of ***y*** depends on the value of ***x, y*** is called the dependent variable and ***x*** is called the independent variable.

2.2.1 Defining Problem as a State Space Search

To solve the problem of playing a game, we require the rules of the game and targets for winning as well as representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state. Moves from initial state to other states leading to the goal state follow legally. However, the rules are far too abundant in most games— especially in chess, where they exceed the number of particles in the universe. Thus, the rules cannot be supplied accurately and computer programs cannot handle easily. The storage also presents another problem but searching can be achieved by hashing.

The number of rules that are used must be minimized and the set can be created by expressing each rule in a form as possible. The representation of games leads to a state space representation and it is common for well-organized games with some structure. This representation allows for the formal definition of a problem that needs the movement from a set of initial positions to one of a set of target positions. It means that the solution involves using known techniques and a systematic search. This is quite a common method in Artificial Intelligence.

2.2.2 State Space Search

A state space represents a problem in terms of states and operators that change states.
A state space consists of the following:

- A representation of the states the system can be in. For example, in a board game, the board represents the current state of the game.
- A set of operators that can change one state into another state. In a board game, the operators are the legal moves from any given state. Often the operators are represented as programs that change a state representation to represent the new state.
- An initial state.
- A set of final states; some of these may be desirable, others undesirable. This set is often represented implicitly by a program that detects terminal states.

The Water Jug Problem

In this problem, we use two jugs called four and three; four holds a maximum of four gallons of water and three a maximum of three gallons of water. How can we get two gallons of water in the four jug?

The state space is a set of prearranged pairs giving the number of gallons of water in the pair of jugs at any time, i.e., (four, three) where four = 0, 1, 2, 3 or 4 and three = 0, 1, 2 or 3.

NOTES

The start state is (0, 0) and the goal state is (2, n) where n may be any but it is limited to three holding from 0 to 3 gallons of water or empty. Three and four shows the name and numerical number shows the amount of water in jugs for solving the water jug problem. Table 2.1 lists the major production rules for solving this problem.

Table 2.1 *Production Rules for the Water Jug Problem*

Initial condition	Goal comment
1. (four, three) if four < 4	(4, three) fill four from tap
2. (four, three) if three< 3	(four, 3) fill three from tap
3. (four, three) If four > 0	(0, three) empty four into drain
4. (four, three) if three > 0	(four, 0) empty three into drain
5. (four, three) if four + three<4	(four + three, 0) empty three into four
6. (four, three) if four + three<3	(0, four + three) empty four into three
7. (0, three) If three > 0	(three, 0) empty three into four
8. (four, 0) if four > 0	(0, four) empty four into three
9. (0, 2)	(2, 0) empty three into four
10. (2, 0)	(0, 2) empty four into three
11. (four, three) if four < 4	(4, three-diff) pour diff, 4-four, into four from three
12. (three, four) if three < 3	(four-diff, 3) pour diff, 3-three, into three from four and a solution is given below four three rule

Table 2.2 shows one solution to the water jug problem.

Table 2.2 *One Solution to the Water Jug Problem*

Gallons in Four Jug	Gallons in Three Jug	Rules Applied
0	0	-
0	3	2
3	0	7
3	3	2
4	2	11
0	2	3
2	0	10

The problem solved by using the production rules in combination with an appropriate control strategy, moving through the problem space until a path from an initial state to a goal state is found. In this problem solving process, search is the fundamental concept. For simple problems it is easier to achieve this goal by hand but there will be cases where this is far too difficult.

2.2.3 Design of Search Programs and Solutions

Solutions can be good in different ways. They can be good in terms of time or storage or in difficulty of the algorithm. In case of the travelling salesman problem, finding the best path can lead to a significant amount of computation. The solution of such problems is only possible by using heuristics. In this type of problem, a path is found of a distance of 8850 miles and another one of 7750. It is clear that the second is better than the first but is it the best? Infinite time may be needed and usually heuristics are used to find a very good path in finite time.

Each search process can be considered to be a tree traversal. The object of the search is to find a path from the initial state to a goal state using a tree. The number of nodes generated might be huge; and in practice many of the nodes would not be needed. The secret of a good search routine is to generate only those nodes that are likely to be useful, rather than having a precise tree. The rules are used to represent the tree implicitly and only to create nodes explicitly if they are actually to be of use.

The following issues arise while searching:

- The tree can be searched forward from the initial node to the goal state or backwards from the goal state to the initial state.
- To select applicable rules, it is critical to have an efficient procedure for matching rules against states.
- How to represent each node of the search process? This is the knowledge representation problem or the frame problem. In games, an array suffices; in other problems, more complex data structures are needed.

Finally in terms of data structures, considering the water jug as a typical problem do we use a graph or tree? The breadth-first structure does take note of all nodes generated but the depth-first one can be modified.

For checking duplicate nodes follow these steps:

1. Observe all nodes that are already generated, if a new node is present.
2. If it exists add it to the graph.
3. If it already exists, then
   a. Set the node that is being expanded to the point to the already existing node corresponding to its successor rather than to the new one. The new one can be thrown away.
   b. If the best or shortest path is being determined, check to see if this path is better or worse than the old one. If worse, do nothing.

Better save the new path and work the change in length through the chain of successor nodes if necessary.

Example: Tic-Tac-Toe

State spaces are good representations for board games such as Tic-Tac-Toe. The position of a game can be explained by the contents of the board and the player whose turn is next. The board can be represented as an array of 9 cells, each of which may contain an X or O or be empty.

State:

- Player to move next: X or O.
- Board configuration:

X		0
	0	
X		X

Operators: Change an empty cell to X or O.

Start State: Board empty; X's turn.

Terminal States: Three X's in a row; Three O's in a row; All cells full.

Search Tree

The sequence of states formed by possible moves is called a *search tree*. Each level of the tree is called a *ply.*

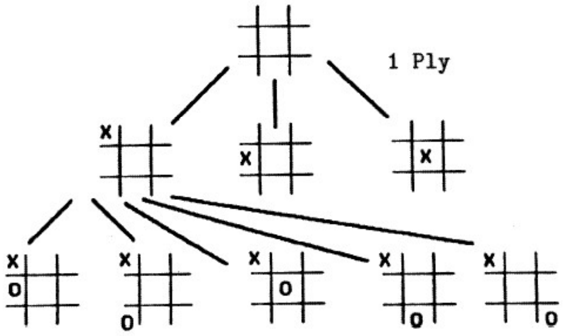


Fig. 2.2 State Space Tree

Since the same state may be reachable by different sequences of moves, the state space may in general be a graph. It may be treated as a tree for simplicity, at the cost of duplicating states.

Solving Problems using Search

- Given an informal description of the problem, construct a formal description as a state space:
  - Define a data structure to represent the *state*.
  - Make a representation for the *initial state* from the given data.
  - Write programs to represent *operators* that change a given state representation to a new state representation.
  - Write a program to detect *terminal states*.
- Choose an appropriate search technique:
  - How large is the search space?
  - How well structured is the domain?
  - What knowledge about the domain can be used to guide the search?

2.3 PRODUCTION SYSTEMS

Production systems provide appropriate structures for performing and describing search processes. A production system has the following four basic components:

- A set of rules each consisting of a left side that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.
- A database of current facts established during the process of inference.
- A control strategy that specifies the order in which the rules will be compared with facts in the database and also specifies how to resolve conflicts in selection of several rules or selection of more facts.
- A rule firing module.

The production rules operate on the knowledge database. Each rule has a precondition—that is, either satisfied or not by the knowledge database. If the precondition is satisfied, the rule can be applied. Application of the rule changes the knowledge database. The control system chooses which applicable rule should be applied and ceases computation when a termination condition on the knowledge database is satisfied.

Example: Eight Puzzle (8-Puzzle)

The 8-puzzle is a 3 × 3 array containing eight square pieces, numbered 1 through 8, and one empty space. A piece can be moved horizontally or vertically into the empty space, in effect exchanging the positions of the piece and the empty space. There are four possible moves, UP (move the blank space up), DOWN, LEFT and RIGHT. The aim of the game is to make a sequence of moves that will convert the board from the start state into the goal state:

2	3	4
8	6	2
7		5

Initial State

1	2	3
8		4
7	6	5

Goal State

This example can be solved by the operator sequence UP, RIGHT, UP, LEFT, DOWN.

Example: Missionaries and Cannibals

The Missionaries and Cannibals problem illustrates the use of state space search for planning under constraints:

Three missionaries and three cannibals wish to cross a river using a two-person boat. If at any time the cannibals outnumber the missionaries on either side of the river, they will eat the missionaries. How can a sequence of boat trips be performed that will get everyone to the other side of the river without any missionaries being eaten?

State Representation:

1. BOAT position: original (T) or final (NIL) side of the river.
2. Number of Missionaries and Cannibals on the original side of the river.
3. Start is (T 3 3); Goal is (NIL 0 0).

Operators:

Table 2.3 lists the operators and their descriptions

Table 2.3 *Operators and Descriptions*

Operators	Descriptions
`(MM 2 0)`	Two Missionaries cross the river.
`(MC 1 1)`	One Missionary and one Cannibal.
`(CC 0 2)`	Two Cannibals.
`(M 1 0)`	One Missionary.
`(C 0 1)`	One Cannibal.

Missionaries/Cannibals Search Graph

Figure 2.3 shows the missionaries/ cannibals search graph.

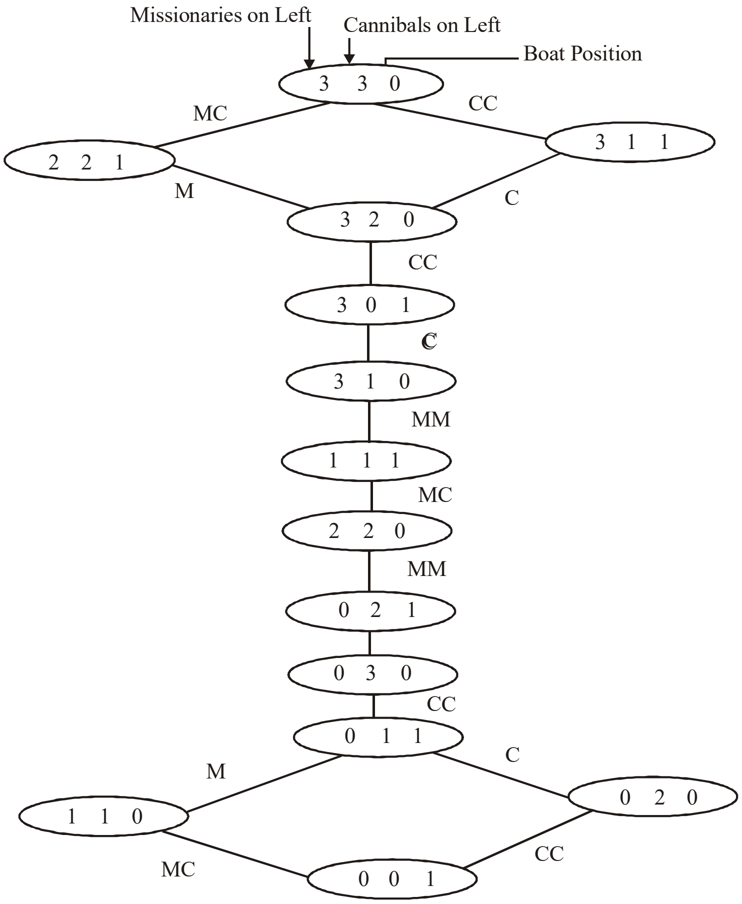


Fig. 2.3 *Missionaries/Cannibals Search Graph*

NOTES

Characteristics of Production Systems

Production systems provide us with good ways of describing the operations that can be performed in a search for a solution to a problem.

At this time, the following two questions may arise:

- Can production systems be described by a set of characteristics? How can they be easily implemented?
- What relationships are there between the problem types and the types of production systems well suited for solving the problems?

To answer these questions, first consider the following definitions of classes of production systems:

- A monotonic production system is a production system in which the application of a rule never prevents the later application of another rule that could also have been applied at the time the first rule was selected.
- A nonmonotonic production system is one in which this is not true.
- A partially communicative production system is a production system with the property that if the application of a particular sequence of rules transforms state P into state Q, then any combination of those rules that is allowable also transforms state P into state Q.
- A commutative production system is a production system that is both monotonic and partially commutative.

Is there any relationship between classes of production systems and classes of problems? For any solvable problems, there exist an infinite number of production systems that show how to find solutions. Any problem that can be solved by any production system can be solved by a commutative one, but the commutative one is practically useless. It may use individual states to represent entire sequences of applications of rules of a simpler, non-commutative system. In the formal sense, there is no relationship between kinds of problems and kinds of production systems since all problems can be solved by all kinds of systems. But in the practical sense, there is definitely such a relationship between the kinds of problems and the kinds of systems that lend themselves to describing those problems.

Partially commutative, monotonic productions systems are useful for solving ignorable problems. These are important from an implementation point of view without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed. Both types of partially commutative production systems are significant from an implementation point; they tend to lead to many duplications of individual states during the search process.

Production systems that are not partially commutative are useful for many problems in which permanent changes occur.

2.4 HEURISTIC SEARCH

A heuristic is a method that improves the efficiency of the search process. These are like tour guides. There are good to the level that they may neglect the points in general interesting directions; they are bad to the level that they may neglect points of interest to particular individuals. Some heuristics help in the search process without sacrificing any claims to entirety that the process might previously had. Others may occasionally cause an excellent path to be overlooked. By sacrificing entirety it increases efficiency. Heuristics may not find the best solution every time but guarantee that they find a good solution in a reasonable time. These are particularly useful in solving tough and complex problems, solutions of which would require infinite time, i.e., far longer than a lifetime for the problems which are not solved in any other way.

Heuristic Search

To find a solution in proper time rather than a complete solution in unlimited time heuristics is used. 'A heuristic function is a function that maps from problem state descriptions to measures of desirability, usually represented as numbers'. Heuristic search methods use knowledge about the problem domain and choose promising operators first. These heuristic search methods use heuristic functions to evaluate the next state towards the goal state. For finding a solution, by using the heuristic technique, one should carry out the following steps:

Step 1. Add domain—specific information to select what is the best path to continue searching along.

Step 2. Define a heuristic function h(n) that estimates the 'goodness' of a node n. Specifically, h(n) = estimated cost(or distance) of minimal cost path from n to a goal state.

The term, heuristic means 'serving to aid discovery' and is an estimate, based on domain specific information that is computable from the current state description of how close we are to a goal.

Finding a route from one city to another city is an example of a search problem in which different search orders and the use of heuristic knowledge are easily understood.

1. State: The current city in which the traveller is located.
2. Operators: Roads linking the current city to other cities.
3. Cost Metric: The cost of taking a given road between cities.
4. Heuristic information: The search could be guided by the direction of the goal city from the current city, or we could use airline distance as an estimate of the distance to the goal.

2.4.1 Heuristic Search Techniques

For complex problems, the traditional algorithms, presented above, are unable to find the solution within some practical time and space limits. Consequently, many special techniques are developed, using heuristic functions.

- Blind search is not always possible, because it requires too much time or Space (memory).
- Heuristics are rules of thumb; they do not guarantee a solution to a problem.
- Heuristic Search is a weak technique but can be effective if applied correctly; it requires domain specific information.

Characteristics of Heuristic Search

The characteristics of heuristic search are as follows:

- Heuristics are knowledge about domain, which help search and reasoning in its domain.
- Heuristic search incorporates domain knowledge to improve efficiency over blind search.
- Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.
  - Heuristics might (for reasons) *underestimate* or *overestimate* the merit of a state with respect to goal.
  - Heuristics that underestimate are desirable and called admissible.
- Heuristic evaluation function estimates likelihood of given state leading to goal state.
- Heuristic search function estimates cost from current state to goal, presuming function is efficient.

Heuristic Search Compared with Other Search

The Heuristic search is compared with Brute force or Blind search techniques shown is Table 2.4.

Table 2.4 *Comparison of Algorithms*

Brute force / Blind search	**Heuristic search**
Can only search what it has knowledge about already	Estimates 'distance' to goal state through explored nodes
No knowledge about how far a node	Guides search process toward goal node from goal state
	Prefers states (nodes) that lead close to and not away from goal state

Example: Travelling Salesman Problem

A salesman has to visit a list of cities and he must visit each city only once. There are different routes between the cities. The problem is to find the shortest route between the cities so that the salesman visits all the cities at once.

Suppose there are N cities, then a solution would be to take N! possible combinations to find the shortest distance to decide the required route. This is not efficient as with N=10 there are 36,28,800 possible routes. This is an example of *combinatorial explosion*.

There are better methods for the solution of such problems: one is called *branch* and *bound*.

First, generate all the complete paths and find the distance of the first complete path. If the next path is shorter, then save it and proceed this way avoiding the path when its length exceeds the saved shortest path length, although it is better than the previous method.

Algorithm: Travelling salesman problem

1. Select a city at random as a starting point
2. Repeat
3. Select the next city from the list of all the cities to be visited and choose the nearest one to the current city, then go to it,
4. until all cities are visited.

This produces a significant development and reduces the time from order N! to N. Our goal is to find the shortest route that visits each city exactly once. Suppose the cities to be visited and the distance between them are as shown in Table 2.5

Table 2.5 *Cities and the Distance Between Them*

	Hyderabad	Secunderabad	Mumbai	Bangalore	Chennai
Hyderabad	-	15	270	780	740
Secunderabad	15	-	280	760	780
Mumbai	270	280	-	340	420
Bangalore	780	760	340	-	770
Chennai	740	780	420	770	-

* (Distance in Kilometres)

One situation is the salesman could start from Hyderabad. In that case, one path might be followed as shown in Figure 2.4.

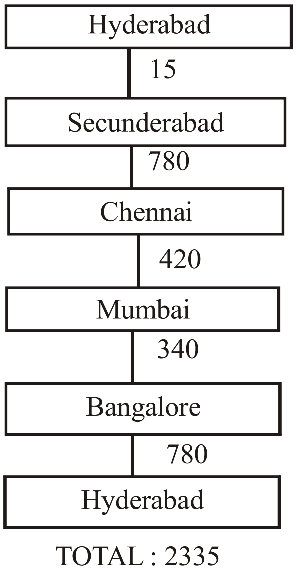


Fig. 2.4 *Travelling Salesman Problem*

Here the total distance is 2335 km. But this may not be a solution to the problem, maybe other paths may give the shortest route.

It is also possible to create a *bound* on the error in the answer, but in general it is not possible to make such an error bound. In real problems, the value of a particular solution is trickier to establish, but this problem is easier if it is measured in miles, and other problems have vague measures.

Although heuristics can be created for unstructured knowledge, producing coherent analysis is another issue and this means that the solution lacks reliability. Rarely is this an optimal solution, since the required approximations are usually in sufficient.

Although heuristic solutions are bad in the worst case, the problem occurs very infrequently.

Formal Statement

Problem solving is a set of statements describing the desired states expressed in a suitable language; e.g., *first-order logic.*

The solution of many problems (like chess, crosses) can be described by finding a sequence of actions that lead to a desired goal.

- Each action changes the state, and
- The aim is to find the sequence of actions that lead from the initial (start) state to a final (goal) state.

A well-defined problem can be described by the example given below:

Example

- *Initial State:* (***S***)
- *Operator or successor function:* for any state ***x*** , returns ***s***(***x***), the set of states reachable from ***x*** with one action.
- *State space:* all states reachable from the initial one by any sequence of actions.
- *Path:* sequence through state space.
- *Path cost:* function that assigns a cost to a path; cost of a path is the sum of costs of individual actions along the path.
- *Goal state:* (***G***)
- *Goal test:* test to determine if at goal state.

Search Notations

Search is the systematic examination of states to find *path* from the *start / root state* to the *goal state*.

The notations used for this purpose are given as follows:

- Evaluation function $f(n)$ **e**stimates the least cost solution through node n
- Heuristic function $h(n)$

  Estimates least cost path from node n to goal node
- Cost function $g(n)$ estimates the least cost path (Refer Figure 2.5) from start node to node n

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

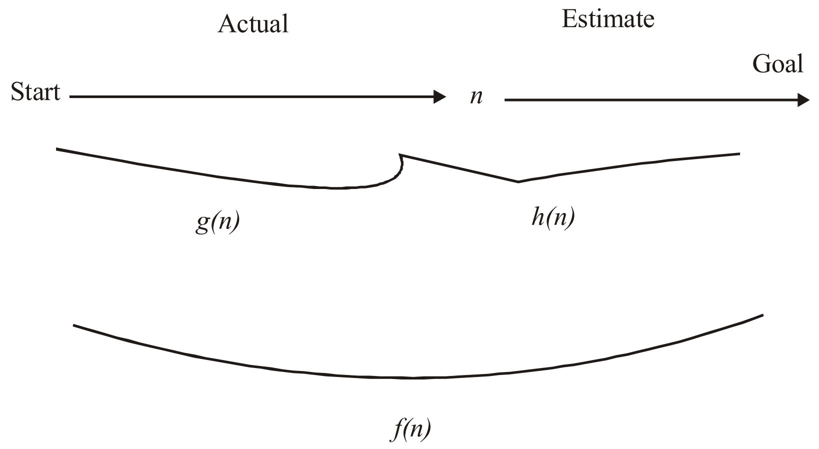


Fig. 2.5 *Estimate of the Least Cost Path*

- The notations ^*f*, ^*g*, ^*h,* are sometimes used to indicate that these values are estimates of *f* , *g* , *h*

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

- If *h*(*n*) d• *actual cost* of the shortest path from node *n* to *goal,* then *h*(*n*) is an *underestimate*.

  AI Search and Control of Strategies

- **Estimate cost function *g****

The estimated least cost path from *start node* to *node* ***n*** is written as ***g**(*n*)**.

- ***g**** is calculated as the actual cost, so far, of the explored path.
- ***g**** is known exactly by summing all path costs from *start* to *current* state.
- If search space is a *tree*, then ***g* = g***, because there is only one path from start node to current node.
- In general, the search space is a graph.
- If search space is a *graph*, then ***g* e" g***,
  - o ***g**** can never be less than the cost of the optimal path; it can only over estimate the cost.
  - o ***g**** can be equal to ***g*** in a graph if chosen properly.
- **Estimate heuristic function *h****

The estimated least cost path from *node* ***n*** to *goal node* is written ***h**(*n*)**

- ***h**** is heuristic information, represents a guess at: 'How hard it is to reach from current node to goal state ?'.
- ***h**** may be estimated using an evaluation function ***f*(*n*)** that measures 'goodness' of a node.
- ***h**** may have different values; the values lie between ***0 d" h*(n) d" h(n)***; they mean a different search algorithm.
- If ***h* = h***, it is a perfect heuristic; means no unnecessary nodes are ever expanded.

NOTES

2.4.2 Best First Search

Best First Search or **BFS** is a search algorithm which searches a graph by expanding the most favorable node selected according to a specified rule. Judea Pearl described best first search as "*Estimating the promise of node n by a heuristic evaluation function f*(*n*) *which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain*".

Some authors have specifically used the term best first search for searching using a heuristic search that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called greedy best first search. Efficient selection of the current best candidate for extension is typically implemented using a priority queue. The A* search algorithm is an example of best first search, as is B*. Best first algorithms are often used for path finding in combinatorial search.

Algorithm

The algorithm for best first search must be correct in order to work efficiently. Consider the following algorithm which is not correct, i.e., it does not always find a possible path between two nodes, even if there is one. For example, it gets stuck in a loop if it arrives at a dead end, i.e., a node with the only successor being its parent. It would then go back to its parent, add the dead end successor to the OPEN list again, and so on.

```
OPEN = [initial state]
while OPEN is not empty or until a goal is found
do
 1. Remove the best node from OPEN, call it n.
 2. If n is the goal state, backtrace path to n (through
recorded parents) and return path.
 3. Create n's successors.
 4. Evaluate each successor, add it to OPEN, and record
its parent.
done
```

The following description extends the algorithm to use an additional CLOSED list, containing all nodes that have been evaluated and will not be looked at again. As this will avoid any node being evaluated twice, it is not subject to infinite loops.

```
OPEN = [initial state]
CLOSED = []
while OPEN is not empty
do
 1. Remove the best node from OPEN, call it n, add it to
CLOSED.
 2. If n is the goal state, backtrace path to n (through
recorded parents) and return path.
 3. Create n's successors.
 4. For each successor do:
a. If it is not in CLOSED and it is not in OPEN: evaluate
it, add it to OPEN, and record its parent.
```

```
b. Otherwise, if this new path is better than previous
one, change its recorded parent.
    i.  If it is not in OPEN add it to OPEN.
    ii. Otherwise, adjust its priority in OPEN using this
new evaluation.
done
```

Best first search in its most universal form is a simple heuristic search algorithm. 'Heuristic' here refers to a general problem solving rule or set of rules that never assure the best solution or even any solution, but functions as a suitable controller for problem solving. Best first search is a graph based search algorithm (Dechter and Pearl, 1985), specifying that the search space can be represented as a series of nodes connected by paths.

Typically, the term 'Best First' refers to the method of exploring the node with the best 'Score' first. An evaluation function is used for assigning a score to each candidate node. The algorithm maintains two lists, one containing a list of candidates yet to explore (OPEN) and the other containing a list of visited nodes (CLOSED). Since all unvisited successor nodes of every visited node are included in the OPEN list, the algorithm is not restricted to only exploring successor nodes of the most recently visited node. Alternatively, the algorithm always selects the best of all unvisited nodes that have been graphed, rather than being restricted to only a small subset, such as immediate neighbours.

The best first search algorithm proceeds in the following manner:

Step 1: Start with OPEN holding the initial state.

Step 2: Repeat.

Step 3: Pick the best node on OPEN.

Step 4: Generate its successors.

The first step is to define the OPEN list with a single node, the starting node. The second step is to check whether or not OPEN is empty. If it is empty, then the algorithm returns failure and exits. The third step is to remove the node with the best score, *n*, from OPEN and place it in CLOSED. The fourth step "expands" the node *n*, where expansion is the identification of successor nodes of *n*. The fifth step then checks each of the successor nodes to see whether or not one of them is the goal node. If any successor is the goal node, the algorithm returns success and the solution, which consists of a path traced backwards from the goal to the start node. Otherwise, the algorithm proceeds to the sixth step. For every successor node, the algorithm applies the evaluation function, *f*, to it and then checks to see if the node has been in either OPEN or CLOSED. If the node has not been in either, it gets added to OPEN. Finally, the seventh step establishes a looping structure by sending the algorithm back to the second step. This loop will only be broken if the algorithm returns success in step five or failure in step two.

The algorithm is represented as follows in the form of pseudo-code:

1. Define a list, OPEN, consisting solely of a single node, the start node, *s*.
2. IF the list is empty, return failure.
3. Remove from the list the node *n* with the best score (the node where *f* is the minimum), and move it to a list, CLOSED.

4. Expand node *n*.
5. IF any successor to *n* is the goal node, return success and the solution (by tracing the path from the goal node to *s*).
6. FOR each successor node:
   - Apply the evaluation function, *f*, to the node.
   - IF the node has not been in either list, add it to OPEN.
7. Loop the structure by sending the algorithm back to the second step.

In addition, best first search is an algorithm that traverses a graph in search of one or more goal nodes. For example, a maze is a special illustration of the mathematical object known as a 'Graph'. The defining characteristic of this search is that, best first search uses an evaluation function called a 'heuristic' search to determine which object is the most promising and then examines this object. This 'best first' behaviour is implemented with a PriorityQueue. The algorithm for best first search is as follows:

```
Best-First-Search( Maze m )
    Insert( m.StartNode )
    Until PriorityQueue is empty
        c <- PriorityQueue.DeleteMin
        If c is the goal
            Exit
        Else
            For each neighbor n of c
                If n "Unvisited"
                    Mark n "Visited"
                    Insert( n )
            Mark c "Examined"

End procedure
```

The objects which will be stored in the PriorityQueue are maze cells and the heuristic search will be the cell's 'Manhattan distance' from the exit. The Manhattan distance is a fast-to-compute and surprisingly accurate measurement of how likely a MazeCell will be on the path to the exit. Geometrically, the Manhattan distance is distance between two points if only allowed to walk on paths that were at 90 degree angles from each other.

Check Your Progress

1. Name the structures of a state space.
2. Define search tree.
3. What is the function of production systems?
4. What kind of production systems are useful for solving ignorable problems?
5. What is heuristic function?
6. How does the best first search algorithm proceed?

2.5 BRANCH AND BOUND

Branch and Bound (BB) is a general algorithm that is used to find optimal solutions of different optimization problems, particularly in discrete and combinatorial optimization. It contains a systematic detail of each candidate solution, in which big subsets of candidates giving no results are rejected in groups, by making use of the higher and lower approximated limits of the quantity that are undergoing optimization.

A.H. Land and A.G. Doig in 1960 were the first ones to propose the method for linear programming.

General Description

For certainty, presume that the aim is of finding the least value of a function $f(x)$, in which x is in the range of over a certain set S of permissible or candidate solutions (the search space or feasible region). Remember that finding the highest value of $f(x)$ is possible by finding the minimum of $g(x) =$ '$f(x)$. (For instance, one could assume S to be the set of all probable trip schedules for a bus fleet, and $f(x)$ could be the aniticipated revenue for schedule x.)

A branch-and-bound process needs two tools. The first one is a *splitting* process in which, given a set S of candidates, it gives back two or more smaller sets S_1, S_2...the union of which would cover S. Remember that the minimum of $f(x)$ over S is min $\{v_1, v_2, ...\}$, where each v_i is the minimum of $f(x)$ inside S_i. This step is known as branching, as its recursive application determines a tree structure (the search tree) the nodes of which are the subsets of S.

One more tool is a process which calculates the upper and lower limits for the minimum value of $f(x)$ inside a given subset S. This step is known as bounding.

The main idea of the BB algorithm is that, in case the lower limit for some tree node (set of candidates) A is higher as compared to the upper limit for another node B, then it is safe to discard A from the search. This step is known as pruning, and is generally applied with the help of a global variable m (shared among all nodes of the tree) which is maintained, that documents the minimum upper limit seen among all subregions investigated so far. Any node the lower limit of which is greater than m can be discarded.

The recursive process ceases once the current candidate set S is decreased to a single element; or even when the upper limit for set S matches the lower limit. In any way, any element of S will be a minimum of the function within S.

Effective Subdivision

The effectiveness of the technique is strongly dependent on the node-splitting process and on the upper and lower limit estimators. Everything else being equal, it is most advisable to select a splitting method that gives non-overlapping subsets.

Typically the process terminates when either pruning or solving of the search nodes is done. At that stage, all non-pruned subregions will have their upper and lower bounds as equivalent to the global minimum of the function. Practically the procedure is frequently stopped after a prescribed time. At that point, the minimum lower and upper bound, amongst all non-pruned sections, determine a range of values that has the global minimum. Optionally, inside an overriding time constraint, it is possible to terminate the algorithm whenever any error criterion, like (*max* – *min*)/(*min* + *max*), is below a precribed value.

NOTES

The effectiveness of the technique relies crucially on the efficiency of the branch and bound algorithms used; it is possible for bad choices to result in repeated branching, with no pruning, till the sub-regions become extremely small. In this case, the method will be decreased to a fully comprehensive details of the domain, that is generally unrealistically big. A global bounding algorithm that would provide a solution for all problems does not exist. Moreover, there is less possibility of ever finding one. Therefore, the normal approach has to be applied distinctly for every application, including branch and bound algorithms that are particularly designed for it.

It is possible to categorize branch and bound techniques as per the bounding methods and as per the methods of creating/inspecting the search tree nodes.

The branch and bound design technique is much like backtracking in which a state space tree is utilized for problem solving. The dissimilarities are that this method (1) does not restrict us from any specific method of traversing the tree and (2) is utilized solely for optimization problems.

This method by nature gives itself to be applied both in a parallel and distributed manner, see, e.g., the travelling salesman problem article.

Applications

The branch and bound algorithm is used for the resolution of following problems:

- Knapsack Problem
- Integer Programming
- Nonlinear Programming
- Traveling Salesman Problem (TSP)
- Quadratic Assignment Problem (QAP)
- Maximum Satisfiability Problem (MAX-SAT)
- Nearest Neighbor Search (NNS)
- Cutting Stock Problem
- False Noise Analysis (FNA)

Branch and bound could even be a base of several heuristics. For instance, one can desire to terminate the branching, once the space between the upper and lower bounds gets smaller as compared to a particular threshold. This is made use of when the solution is 'appropriate enough for realistic objectives' and can greatly reduce the computations required. This kind of solution can be mainly applied when the cost function brought into use is loud or is the outcome of statistical estimates. Therefore, one does not know specifically. Instead, one only knows to be inside a range of values with a particular possibility. An instance of its implementation here is in biology while cladistic analysis is being performed for evaluating developing relationships between organisms, wherein the data sets are frequently unrealistically huge with no heuristics.

This causes branch and bound techniques to be frequently used in game tree search algorithms, most significantly through the utilization of alpha-beta pruning.

Branch and Bound Algorithm Technique

NOTES

Branch and bound is yet another algorithm technique that will be presented in the multi-part article series including algorithm design patterns and techniques. Branch and bound is one of the most complicated techniques and for sure is difficult to be discussed as a whole in one article. Thus, the focus will be on the A* algorithm which is the most distinct branch and bound graph search algorithm.

By now, you should be able to understand that the most vital techniques, like, backtracking, the greedy strategy, divide and conquer, dynamic programming, and even genetic programming have all been covered. It is extremely helpful in understanding the dissimilarities branch and bound algorithms.

Branch and bound is an algorithm technique that is frequently applied to find the optimal solutions when optimization problems arise. It is chiefly brought into use for combinatorial and discrete global optimizations of problems. In short, this technique is the best option when the domain of probable candidates is extremely big and all the other algorithms prove unsuccessful. This technique is grounded on the group removal of the candidates.

The tree structure of algorithms must already be known to you. Among the techniques learned, both backtracking and divide-and-conquer travel through the tree in its depth, though they adopt opposite routes. The greedy strategy takes up a single route and does not bother about the others. Dynamic programming is known to approach this in a kind of Breadth First Search Variation (BFS).

Now, in case the decision tree of the problem that you plan to solve has really an unlimited depth, then, according to definition, the backtracking and divide and conquer algorithms are eliminated. Greedy strategy cannot be relied upon since it is problem-dependent and does not ensure of delivering a global optimum, unless mathematically proved otherwise.

As a final resort, you can also consider dynamic programming. The fact is that possibly the problem can actually be solved with the help of dynamic programming. However, the implementation will not be an effective approach; moreover, its implementation will be difficult. Therefore, you can understand, that in case there is a complicated problem in which many parameters will be required to explain the solutions of sub-problems, dynamic programming will prove to be ineffective.

In case a real-world proof is still required, the fifteen puzzle exists. One of the most direct implementations of dynamic programming will need sixteen different parameters for representing the optimum values of the solutions of every sub-problem. This means a 16-dimensional array. Thus, the reason for dynamic programming is eliminated.

This can be summarized in a list form as follows:

- Suppose you are attempting to find the minimum of a certain function f(x) in a certain range x[x1;x2]. You do not want to simply repeat all the values of x—you need optimizations.

- BB algorithm first divides the [x1;x2] range into a number of sub-ranges.
- After that BB algorithm makes an estimate for the lower and upper bounds for the lowest value of f(x) for every sub-range; this is the stage where optimization takes place— rather than all the values inside a sub-range, just two evaluations of f(x) for each sub-range can be performed.

A comparison of all the sub-ranges is done, with the one with the lowest upper bound for the minimal f(x) value being repeated for finding the universal minimum of f(x). Rather than repeating the final sub-range, it can even be divided into more sub-ranges for implementing the branch and bound algorithm once again.

Branch and bound algorithm appears like a typical tradeoff case. You may obtain speed (while sub-ranges are rough, i.e., involve several data points).You may even obtain accuracy (e.g., while each sub-range has only three data-points, such that it is difficult to miss a local minimum).

However, there there in no proof to show that implementing branch and bound algorithm to one set of data points with differing criteria for sub-range sizes can enable accomplishing greater accuracy (by reducing chances of leaving out a local minimum) while retaining unproportionally greater speed for coarse sub-ranges.

2.6 PROBLEM REDUCTION

If you are searching for a series of actions for achieving a certain goal, the method of state-space search can be used, in which every node in your search space is a state of the world. You are also looking for a series of actions that help you reach from an initial state to a final state. Another method is considering the various methods with the help of which the goal state can be broken down into uncomplicated subgoals. For instance, while you plan a visit to Mumbai, you possibly do not desire to look at all the probable series of actions that may help you reach Mumbai. It is more likely that you break down the problem into uncomplicated ones, like, reaching the station, then getting a train to Mumbai. More than one probable method of breaking down the problem—an optional technique may be arriving at the airport and taking a flight to Mumbai. These various possible plans may have varied expenses (and benefits) related with them, and you may need to select the best plan.

The simpler state-space search techniques can be depicted with the help of a tree in which each successive node denotes an *option* action to be adopted. The graph structure being looked for is denoted as an *OR graph*. To show problem-reduction techniques, an AND-OR graph/tree needs to be used. Here, one can have *and* nodes the successors of which should *all* be accomplished, and *or* nodes in which *one* among the successors need to be accomplished (i.e., they are options). This enables you to depict both instances wherein it is imperative to satisy ALL of a set of subgoals to accomplish a certain goal, and wherein optional subgoals exist, out of any one can accomplish the goal. Some of the alternatives for making a travel plan to Mumbai could be with the help of AND-OR tree. In this sets of goals *each one* of which has to be satisfied are shown with the help of a line that connects all the components.

The nodes that succeed AND nodes depict goals that have to be accomplished in a combined manner. Successors of OR nodes depict various methods to achieve a goal. Certain AND-OR trees contain levels involving both AND and OR nodes. However, this has a tendency for less clarity.

To search for a method of getting to Mumbai you have to search this tree for finding a set of uncomplicated goals, the method of satisfying which, you casually know. Possibly, ordering a taxi is a primary goal—it could be broken down into 'lift the phone, dial number ..'. However, this will not prove to be very useful. In any case, you have to implement your fundamental notions of tree/graph search to searching AND-OR trees/graphs.

There are several possible methods to search AND-OR graphs. One method is tranforming the graph back into OR graphs in an effective manner, in which every node depicts an entire set of goals that need to be achieved. Therefore, as far as our search algorithm is concerned, every item on the list (/Open node list) is a set of goals. Searching a successor to an item on the list includes lifting a non-primitive goal (say G) from this set of goals and to find probable subgoals for that goal. If the node (that corresponds to G) is an AND node then there would be just one successor. This is a set of goals, with the goal G replacing its subgoals. In case the node was an OR node then there would be many possible nodes succeeding it, each one happening to be a set of goals with apossible successor replacing G. A last 'goal state' according to this will be a set of goals/actions that can be executed directly or are primary. The node lists would be lists of goals, in which every sublist will signify a possible plan which has only been partially developed. An example node list may be the following:

open = [[cycle-to-QS, get-train],[walk-to-QS, get-train], [get-to-airport, get-plane, get-tube]]

It becomes a little more complicated when you want to make use of heuristic search. In that case, you need to assess the benefits of an entire set of goals. In case every goal has a related (estimate) expense, then the expenditure of a set of goals will only be the total of these costs.

Notice that AND-OR graph (or tree) search techniques are required if you want to use backward chaining to prove something with a set of rules of the form 'IF A AND B AND ... THEN C'. The problem of proving (for example) C is being decreased to the problems of proving A and proving B; etc. In reality, the Prolog's built-in search strategy can be proved to implement a simple AND-OR tree search.

Problem Reduction Search

Certain times problems merely appear difficult to be solved. A problem which is possible to be broken A down into several uncomplicated problems is known as a difficult problem. Moreover, when every single uncomplicated problem is resolved, it means the difficult problem has also been worked out. This is the fundamental thinking that underlies the technique of problem reduction. The classical problem that is made use of to depict problem reduction search is the Tower of Hanoi problem, an extremely refined solution can be found for this problem making use of this technique. The story that is classically cited for describing the Tower of

Hanoi problem shows the specific problem that the priests of Brahmah face. In case you do not know this story, the crux of it is that 64-size ordered disks fill up one of 3 pegs and should be shifted to any one of the other peg. However,it is possible to move only one disk at a time; and it is not possible to place a bigger disk over a smaller disk.

Instead of dealing with the 64-disk problem that the priests face, only three disks will be considered which is the least needed to help the problem be of more interest and helpful to the objective, that of, depicting problem-reduction search. Figure 2.6(*a*) depicts the state space related to a 3-disk Tower of Hanoi Problem. The problem includes shifting from a state in which the disks are arranged on one of the pegs and shifting them to have them arranged on another peg. In this instance, the state on the top of the diagram the *starting state* will be considered. In this instance, all three disks happen to be on the left-most peg. The state at the bottom right will be considered to be the *goal state*. In this state, the arrangement of the three is on the right-most peg.

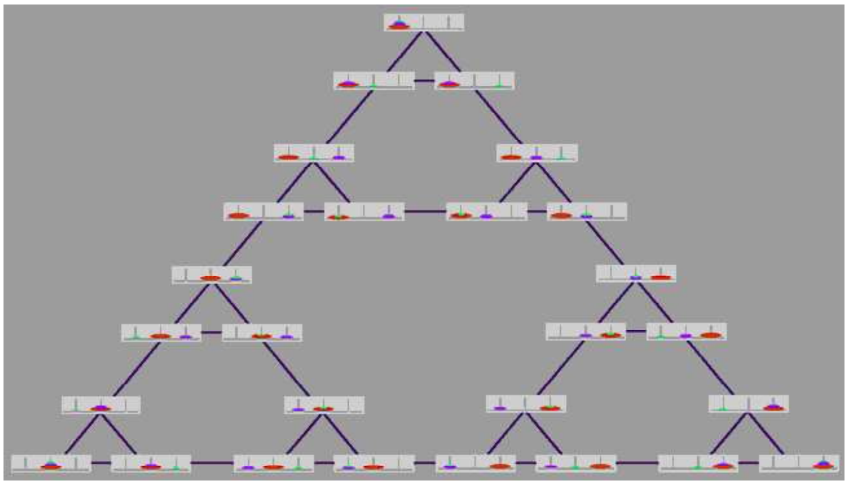


Fig. 2.6(a) *State Space for 3-Disk Tower of Hanoi Problem*

Recollect that in state space search the generators relate to shifts in the state space. Therefore, the two states below the top state in the triangle of states correspond to the shifts of the smallest disk either to the rightmost peg or to the middle peg. The simplest solution to this problem seems to correspond to the path down the right side of the state space. This solution is shown in Figure 2.6(b).

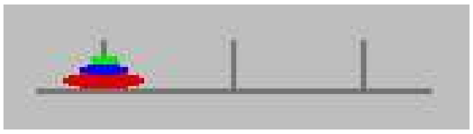


Fig. 2.6(b) *The Solution to the 3-Disk Tower of Hanoi Problem*

The problem space in problem reduction search contains an AND/OR graph of (partial) state pairs. Such pairs are known as (*sub*)*problems*. The pair's initial element is the starting state of the (sub)problem and the pair's second element is the goal state (sub)problem. Two kinds of generators exist: non-terminal rules and terminal rules. Non-terminal rules break down a problem pair, <s0, g0> into an

AND set of problem pairs {<si,gi>, <sj,gj>, ...>. The presumption being the set of subproblems happen to be in a certain sense uncomplicated problems as compared to the problem itself. An AND set is used to refer to the set since it is assumed that the solution of all the subproblems indicates that a solution to the problem has been found. Note that all of the subproblems need to be solved for solving the main problem. It is possible to decompose any subproblem into a number of subproblems. However, for this technique to be successful, all subproblems should finally end in primitive subproblems. A primitive subproblem is one that cannot be broken down (i.e., no such non-terminal exists that can be applied to the subproblem) with its solution being uncomplicated or straightforward. The terminal rules function as those that recognize primary subproblems.

The state space' symmetry depicted in Figure could have resulted in making you conclude that it is possible to solve the Tower of Hanoi problem with the help of the method of problem decomposition. The AND tree that solves the 3-disk problem is shown in Figure 2.7.

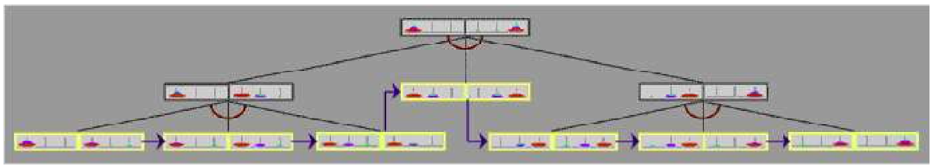


Fig. 2.7 AND Tree Used for 3-Disk Tower

AND Tree Showing the Problem Reduction Solution: Give a number to the state space solution depicted in the state space above 1 through 8 so that states can be referred to by number. 1 matches with the topmost or starting state and 8 to the right corner or goal state. These two states are the first and second element in the problem depicted as the root node in the AND tree above. The arc indicates that the node is an AND node.

The problem is further broken down into three subproblems. The left subproblem contains states 1 and 4; the middle subproblem contains states 4 and 5; and the right contains states 5 and 8. Notice that the left and the right subproblems match to the top and bottom nodes of the upper and lower triangles respectively. The middle subproblem matches to the shift that connects these two triangles of states. Note that this central subproblem does not have any further decomposition. It is a primary problem corresponding to shifting the large disk from the first to the third peg. The border helps in depicting primitive or terminal subproblems. The left and right subproblems are not ancinet subproblems and both of them are broken down further. The three subproblems for every subproblem are primitive corresponding to the first three and the last three moves of the solution.

This example shows only the AND nodes An OR node matches with the case in which more than one various non-terminal rules are implemented to a specific subproblem. For an OR node, a minimum one of the OR nodes should be solved to solve the (sub)problem (Refer Figure 2.8).

NOTES

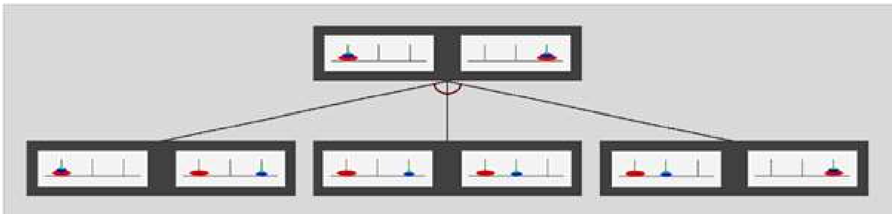


Fig. 2.8 Implementing OR NODE in 3-Disk Tower

Notice that according to syntax, a break down is merely a set of (partial) state descriptions. Therefore, several probable non-terminal reduction rules exist which can be defined for this 3-disk Tower of Hanoi Problem shown in Figure 2.9 depicts another probable non-terminal rule regarding this problem. In this case, the first subproblem is shifting the middle and small sized disks to the right peg. The next subproblem is to shift these disks from the right peg to the centre peg. This is found to be corresponding to shifting down the left side of the triangle in the state space above and then going over to the right side. Though this includes unwanted moves, it results in a relevant solution when combined with the suitable further rules of decomposition.

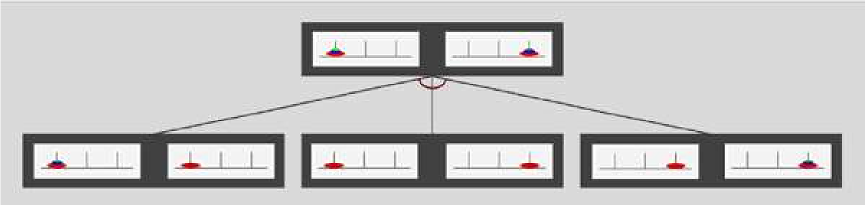


Fig. 2.9 Non -Terminal Rules

Notice that four states in the state space exist satisfying the goal of the first subproblem and also four satisfying the goal of the second subproblem. This observation forces one to understand that indispensable while using non-terminal rules for problem reduction is the presumption that there exists a path in the state space that will be found out once the partial state descriptions are bound to a suitable state space.

Order of Problem Solving and Order of Problem Execution

One more vital aspect of problem reduction rules is that using them enables the *order of problem solving* to be different from the o*rder of problem execution*. In the state space mentioned these two are evidently the same In the given case, the problem solver can select any of the subproblems to work on initially.

In the Tower of Hanoi problem no evident benefit exists of solving the problem in an order that is different from the order of problem execution. However, in certain problems one may note that some subproblems are not as difficult as others and their solution can make the solution of the remaining subproblems simpler. A Cryptarithmetic Problem has been evolved to illustrate this point. In this case, the subproblems correspond to the various equations that are formed in the algebraic depiction of the problem. Notice that, in this case the state space has 10! states and finding a solution for it it via state space search is not possible for any sensible human mind.

2.7 CONSTRAINT SATISFACTION

NOTES

Constraint satisfaction is a usual problem the goal of which is finding values for a set of variables which would satisfy a given set of constraints. It is the centre of several applications in AI, and has witnessed its implementation in several domains. These domains include planning and scheduling. Due its usuality, maximum AI researchers must be able to gain from possessing sound knowledge of methods in this field.

Constraint Satisfaction Problems

Constraint satisfaction problems or CSPs are mathematical problems defined as a set of objects whose state must satisfy many constraints or limitations. CSPs help in representing the entities in a problem as a uniform collection of finite limitations over variables, that can be solved by constraint satisfaction techniques. CSPs are the topic of intensive research in both AI and operations research, as their customary formulation offers a general base for analysing and solving problems of a number of unrelated families. CSPs frequently show great complexity, that requires a coupling of heuristics and combinatorial search techniques to be solved within a rational time.

Examples of problems that can be modelled as a CSP are as follows:

- Eight-queens puzzle
- Map coloring problem
- Sudoku
- Boolean satisfiability

Formal Definition

Formally, a CSP can be defined as a triple (X, D, C), in which *X* is a set of variables, *D* is a domain of values, and *C* is a set of constraints. Each constraint is in turn a pair (t, R) where *t* is a tuple of variables and *R* is a set of tuples of values. All these tuples have the same number of elements; the result is that *R* is a relation. An assessment of the variables is a function from variables to values, $v : X \rightarrow D$. This kind of an assessment is known to satisfy a constraint $<(x_1 ..., x_n)R>$ if $(v(x_1), ..., v(x_n)) \varepsilon R$. A solution is an assessment that is known to satisfy all constraints.

Resolution of CSPs

CSPs on fixed domains are classically solved with the help of a form of search. The most used methods are types of backtracking, constraint propagation and local search.

Backtracking is a recursive algorithm. It is known to maintain an incomplete assignment of the variables. In the beginning, none of the variables are assigned. At every step, a variable is selected, with all possible values being assigned to it in turn. For every value, a checking of the consistency of the incomplete assignment with the constraints is performed. In case of consistency, a recursive call is carried out. When each value has been tried, the algorithm backtracks. In this basic backtracking algorithm, consistency is defined as the satisfaction of all constraints the variables of which are all assigned. Different types of backtracking exist. Backmarking enhances the effectiveness to check consistency. Backjumping enables a portion of the search to be saved by backtracking 'more than one variable' in certain instances. Constraint learning deduces and saves new constraints which can be made use of later to keep away from performing a part of the search. Look-ahead is also frequently made use of in backtracking to try to predict the effects of selecting a variable or a value. Thus, it sometimes determines in advance whether a subproblem can be solved or not.

Constraint propagation techniques are made use of for modifying a CSP. More accurately, they are techniques that impose a kind of local consistency, which are conditions associated with the consistency of a group of variables and/or constraints. Constraints propagation has several uses. First, it turns a problem into one that is equivalent but is generally uncomplicated to solve. Second, they can prove wether a problem can be solved or not. It is not sure whether this would take place in general. However, it always occurs for certain kinds of constraint propagation and/or for certain types of problems. The most popular types of local consistency that is widely used are arc_consistency, hyper-arc consistency, and path consistency. The most widely known constraint propagation technique is the AC-3 algorithm, which imposes arc consistency.

Local search methods are partial satisfiability algorithms. It is not sure if they can find solution to a problem or not. They function by repeatedly enhancing an entire assignment over the variables. At every stage, a few variables change value, with the whole objective being that to increase the number of constraints satisfied by this assignment. The min-conflicts algorithm is a local search algorithm that are CSPs specific and grounded on that principle. In reality, local search seems to function effectively when these changes are also affected by random choices. Combination of search with local search has been developed, resulting in hybrid algorithms.

Theoretical Aspects of CSPs

CSPs are also a subject of study in computational complexity theory and finite model theory. A vital question is whether for each set of relations, the set of all CSPs that can be represented using only relations chosen from that set is either in PTIME or otherwise NP-complete (assuming P = NP). If this kind of a dichotomy is true, then CSPs offer one of the largest known subsets of NP. This evades problems that are neither polynomial time solvable nor NP-complete, whose existence was shown by Ladner. Dichotomy results are famous for CSPs where the domain of values is of size 2 or 3, but the general case remains open.

Maximum classes of CSPs which are known to follow instructions are those where the hypergraph of constraints has bounded treewidth (with no limitations on the set of constraint relations), or where the constraints have a random kind but there exist mainly non-unary polymorphisms of the set of constraint relations.

Each CSP may even be considered as a conjunctive query containment problem.

Types of CSPs

The typical model of CSP outlines a model of stationary, rigid constraints. This inflexible model is a fault which makes the representation of problems difficult. Many proposals about making changes to the primary CSP definition have been put forth, so that the models adapts to a broad range of problems.

Dynamic CSPs

Dynamic CSPs (DCSPs) are helpful when the original design of a problem is modified in a certain way, mainly due to the set of constraints being considered undergoes evolution because of the environment. DCSPs are seen as a series of static CSPs, with each one being a modification of the earlier one wherein it is possible to add (restriction) or remove (relaxation) variables and constraints. Information seen in the originial formulations of the problem may be made use of for refining the subsequent ones. It is possible to classify the solving method as per the method in which transfer of information takes place. The solving methods are as follows:

- **Oracles**: The solution found to earlier CSPs in the series are made use of as heuristics to direct the resolution of the present CSP from the beginning.
- **Local repair**: Every CSP is computed beginning from the incomplete solution of the earlier one and making repairs to the varying constraints with local search.
- **Constraint recording**: New constraints are defined at every step of the search for representing the learning of varying group of decisions. Those constraints are taken over to the new CSP problems.

Flexible CSPs

Typical CSPs handle constraints as being rigid. This means that they are extremely important (every solution should satisfy all of them) and nonflexible (meaning that they should be fully satisfied otherwise they are totally violated). Changeable CSPs help in relaxing those presumptions with partial relaxation of the constraints and permitting non-complaince of the solution with all of them. Some kinds of flexible CSPs are as follows:

- MAX-CSP, in which violation of several constraints is permitted, with the quality of a solution being measured by the number of satisfied constraints.
- Weighted CSP, a MAX-CSP in which every constraint being violated is weighed as per a pre-defined preference. Therefore, satisfying constraint with greater weight takes preference.
- Fuzzy CSP model constraints as fuzzy relations in which the satisfaction of a constraint is a constant function of its variables' values, moving from completely satisfied to completely violated.

Specifying Constraint Problems

Just like many successful AI techniques, constraint solving is all about find solutions to problems. By any means define the intelligent task as a problem, then rub it into a CSP's , place it inside a constraint solver and check if you find a solution. The parts that CSPs contain are as follows:

- A set of variables X = {x_1, x_2, ..., x_n}
- A finite set of values that each variable can take. This is known as the domain of the variable. The domain of variable x_i is written as $D_{i..}$
- A set of constraints that denotes the values that variables can assume simultaneously

Depending on the solver that you are using, constraints are frequently expressed as relationships between variables, e.g., $x_1 + x_2 < x_3$. However, to discuss constraints in a more formal manner, the following notation is used:

A constraint C_{ijk} specifies the tuples of values variables x_i, x_j and x_k that are permitted to take simultaneously. In simple terms, a constraint usually speaks about things which can not happen, but formally, tuples (v_i, v_j, v_k) are being considered that x_i, x_j and x_k can take simultaneously. As a simple example, suppose you have a CSP with two variables x and y, and that x can take values {1,2,3}, while y can take values {2,3}. Then the constraint that x=y will be expressed as:

C_{xy}={(2,2), (3,3)},

and the constraint that x<y will be expressed as

C_{xy} = {(1,2),(1,3),(2,3)}

A solution to a CSP is assigning of values, one to every variable in a manner that no constraint breaks. It is dependent on the existing problem, but the user may want to be aware about the existence of a solution, i.e., the user will adopt the first given answer. Optionally, they may need all the solutions to the problem, or they may like to know that there is no solution to the problem. Certain times, the aim of the exercise is finding the optimum solution on the basis of a certain measure of worth. Certain times, this can be done without all the solutions being enumerated. However, at other times, it would be essential to find all solutions, then assess the one which would be the optimum. In the high-IQ problem, a solution is merely a set of lengths, one per square.

Binary Constraints

Unary constraints denote that a specific variable can adopt certain values, which primarily limits the domain for that variable, and hence needs to be taken care of while the CSP is being specified. Binary constraints associate two variables, and binary constraint problems are particular CSPs involving only binary constraints. Binary CSPs have a distinct place in the theory since all CSPs can be denoted as binary CSPs. Moreover, binary CSPs can be represented using graphs and matrices, which can make them easier to comprehend.

Binary constraint graphs, such as the one shown in Figure 2.10 represent the constraint problems clearly. Here, the nodes are the variables and the edges denote the constraints on the variables between the two variables that the edge joins (note that the constraints indicate the values that can be assumed at the same time).

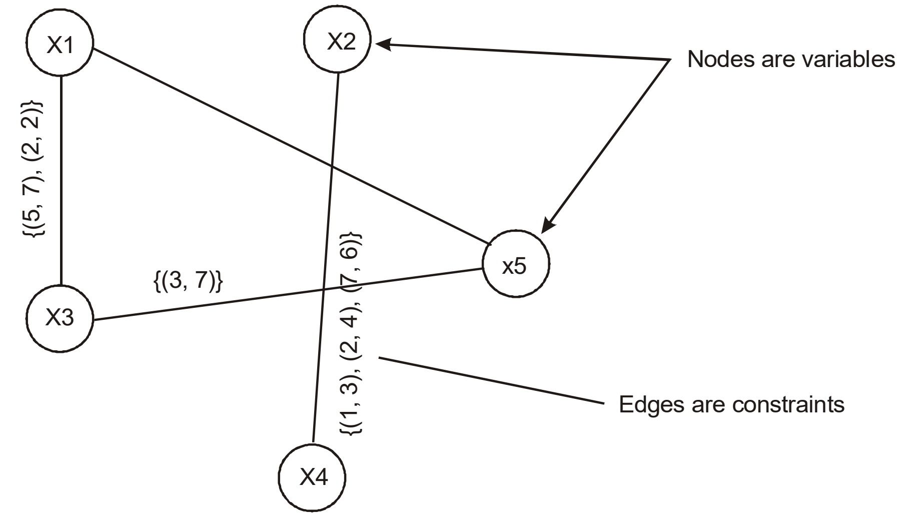


Fig. 2.10 *Binary Constraints Tower*

Matrices can also be used to represent binary constraints, with one matrix for every constraint. For instance, in the above constraint graph, the constraint between variables x_4 and x_5 is $\{(1,3),(2,4),(7,6)\}$. Table 2.6 represents this.

Table 2.6 *Matrices*

C	1	2	3	4	5	6	7
1			*				
2				*			
3							
4							
5							
6							
7						*	

In this the asterixes signify the entry (i,j) in the table such that variable x_4 can take value i at the same time that variable x_5 assumes value j. Since it possible to write all CSPs as binary CSPs, the artificial generation of random binary CSPs as a set of matrices is frequently used to evaluate the relative capabilities of constraint solvers. However, it is important to note that in actual world constraint problems, much more structure to the problems exist as compared to what you obtain from such random constructions.

A common example of CSP, is the 'n-queens' problem, which is the problem of positioning *n* queens on a chess board in a manner that there is no threat to one another along the vertical, horizontal or diagonal. There are several probabilities to represent this as a CSP (in fact, to find the best specification of a problem such that a solver is able to get the answer as soon as possible is an extremely skilled art). One likelihood is to have the variables represent the rows and the values they can assume to represent the columns on the row that a queen was located on. Take a look at the following solution to the 4-queens problem shown in Figure 2.11.

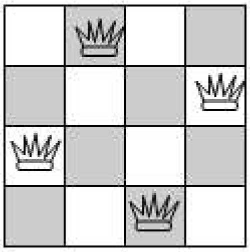


Fig. 2.11 *4-Queens Problem*

Then, if you count rows from the top downwards and columns from the left, the solution can be denoted as: X_1=2, X_2=4, X_3=1, X_4=3. This is due to the fact that the queen on row 1 is in column 2, the queen in row 2 is in column 4, the queen in row 3 is in column 1 and the queen in row 4 is in column 3. The constraint between variable X_1 and X_2 will be:

$$C_{1,2} = \{(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)\}$$

For an exercise, try working out precisely that which the above constraint is trying to say.

Arc Consistency

There have been several advances in the methodology of constraint solvers searching for solutions (note that this denotes an assignment of a value to every variable in a manner that none of the constraint is violated). You first look at a pre-processing stage which helps in highly improving the effectiveness by pruning the search space, namely arc-consistency.

The pre-processing routine for binary constraints called arc-consistency includes calling a pair (x_i, x_j) an arc and taking note that this is an ordered pair, i.e., it is not similar to (x_j, x_i). Every arc is related with one constraint C_{ij}, that constrains variables x_i and x_j. It can be said that the arc (x_i, x_j) is consistent if, for all values *a* in D_i, there is a value *b* in D_j in a manner that the assignment x_i=a and x_j=b satisfies constraint C_{ij}. Remember that (x_i, x_j) being consistent does not essentially signify that (x_j,x_i) too is consistent. To utilize this in a pre-processing manner, each pair of variables needs to be taken and made arc-consistent. That is, each pair (x_i,x_j) needs to be taken and the variables removed from D_i which make it inconsistent, till it becomes consistent. This helps in the effective removal of values from the domain of variables. Hence it prunes the search space making it possible for the solver to succeed (or be unsuccessful in finding a solution) faster.

To show the value of carrying out an arc-consistency check before beginning a search for a solution, an example from Barbara Smith's tutorial. Imagine that you have four tasks to complete, namely, A, B, C and D, and you are trying to schedule them. They are subjected to constraints which are as follows:

- Task A is known to last for 3 hours and precedes tasks B and C.
- Task B is known to last for 2 hours and precedes task D.
- Task C is known to last for 4 hours and precedes task D.
- Task D is known to last for 2 hours.

This problem will be modelled with a variable for every task start times, namely *startA, startB, startC and startD*. You will even have a variable for the overall start time, namely, *start*, and a variable for the overall finishing time, namely, *finish*. You may consider that {0} is the domain for the variable *start*. However, the domains for all the other variables is {0,1,...,11}, since the sum of the duration of the tasks is 3 + 2 + 4 + 2 = 11. The English specification of the constraints can be translated into our formal model. Therefore, you begin with an intermediate start:

- *start d” startA*
- *startA + 3 d” startB*
- *startA + 3 d” startC*
- *startB + 2 d” startD*
- *startC + 2 d” startD*
- *startD + 2 d” finish*

Then, by considering the values that each pair of variables can take in a simultaneous manner, the constraints can be written as follows:

- $C_{start,startA}$ = {(0,0), (0,1), (0,2), ..., (0,11)}.
- $C_{startA,start}$ = {(0,0), (1,0), (2,0), ..., (11,0)}.
- $C_{startA,startB}$ = {(0,3), (0,4), ..., (0,11), (1,4), (1,5), ..., (8,11)}, etc.

Now, it will be verified if every arc is arc-consistent, and if not, the values from the domains of variables will be removed consistency is attained. You now need to first consider the arc (*start, start A*) which is associated with the constraint {(0,0), (0,1), (0,2), ..., (0,11)} above. You need to verify if there is any value, P, in D_{start} that is without a corresponding value, Q, in a manner that (P,Q) satisfies the constraint, i.e., can be seen in the set of assignable pairs. As D_{start} is only {0}, there is no cause for worry. You then start looking at the arc (*startA, start*), and verify if there is any value in D_{startA}, P, which does not have a corresponding Q such that (P,Q) is in $C_{startA,\ start}$. Again, there is no cause for worry, since all the values in D_{startA} appear in $C_{startA,\ start}$.

If you now look at the arc (startA, startB), then the constraint in question is: {(0,3), (0,4), ..., (0,11), (1,4), (1,5), ..., (8,11)}. It can be seen that their is no pair of the form (9,Q) in the constraint, likewise, no pair of the form (10,Q) or (11,Q). Hence, this arc is not arc-consistent, and it is important to eliminate the values 9, 10 and 11 from the domain of *startA* to bring consistency to the arc. This is correct, since you know that, if task B will start after task A, with duration of 3 hours, and they will all start by the eleventh hour, then it is not possible for task A to begin after the eighth hour. Therefore, –it is possible to remove the values 9, 10 and 11 from the domain of *startA*.

This technique of eliminating values from domains is extremely effective. The domains become very small, as shown in the scheduling network shown in Figure 2.12.

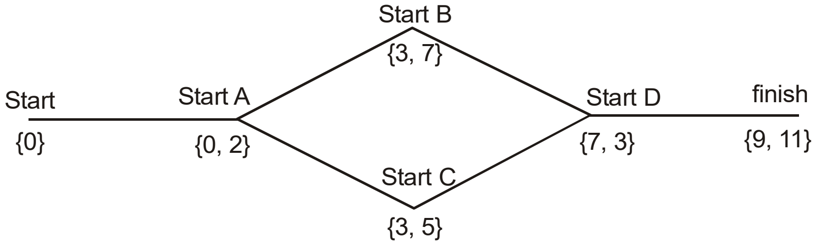


Fig. 2.12 *Scheduling Network for Specific Domains*

You can see that the largest domain size contains only 5 values. This denotes that a majority of the search space has been pruned. In reality, to eliminate as many variables as possible in a CSP that depends on precedence constraints. You need to work backwards, i.e., see the start time of the task, T, which should take place in the end, then make every arc of the form (*startT, Y*) consistent for every variable *Y*. Subsequently, go on to the task that needs to take place second to last, etc. In CSPs only involving precedence constraints, arc-consistency is sure to eliminate all values that cannot appear in a solution to the CSP. In general, however, such a guarantee cannot be made, but arc-consistency generally affects the beginning specification of a problem in some way or the other.

Search Methods and Heuristics

The question of the method of constraint solvers searching for solutions now arises—constraint that preserve assignments of values to variables—to the CSPs they are given. The most evident approach is using a depth first search: assigning a value to the first variable and checking that this assignment does not violate any constraints. Then, go on to the next variable, assigning it a value and checking that this does not violate any constraints, then go on to the next variable, so on and so forth. In case an assignment violates a constraint, select another value for the assignment till one is found that satisfies the constraints. In case, it is difficult to find one, then this is when the search should backtrack. In this condition, the earlier variable is once again looked at, with the next value for it being tried. In this manner, all probable sets of assignments will be tried, with a solution being found. The search diagram shown in Figure 2.13 taken from Smith's tutorial paper—denotes how the search for a solution to the 4-queens problem progresses until it comes across a solution.

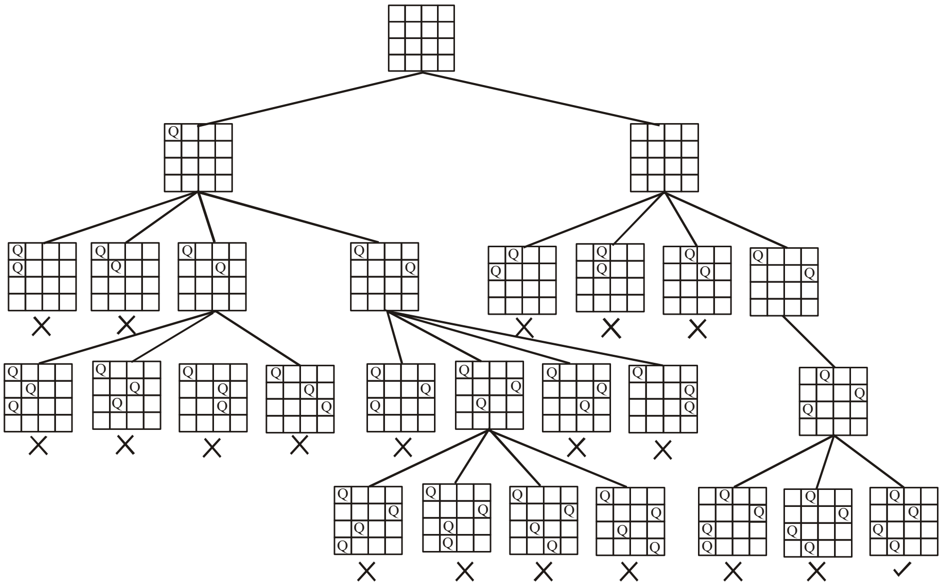


Fig. 2.13 *Search Diagram in 4-Queens Problem*

You can see that the first time that it backtracks is after it has failed to place a queen in row three given queens in positions (1,1) and (2,3). In this case, it backtracked and moved the queen in (2,3) to (2,4). Ultimately, this did not work out either, so it was forced to backtrack further to move the queen in (1,1) to (1,2). This helped in reaching the solution much faster.

A technique called forward checking is used by constraint solvers for adding some sophistication t the search method. The common notion is to work similar to a backtracking search. However, while conformance with constraints is checked after a value to a variable ha been assigned, the agent even will check if this assignment will break constraints with future variable assignments. That is, if V_c has been allocated to the present variable c, then for every unassigned variable x_i, (temporarily) eliminate all values from D_i which, combined with V_c break a constraint. It is quite possible that while doing this D_i becomes empty. This denotes that the selection of V_c for the present variable is not too good—it will be unable to find its way into a solution to the problem, since there is no method of assigning a value to x_i without a constraint getting broken. In this kind of a situation, although the assigning of V_c may not break any constraints with previously assigned variables, a new value is selected (or backtracking takes place in case there are no values left), since it is known know that V_c is not a good assignment.

The diagram (again, taken from Smith's tutorial) shown in Figure 2.14 depicts the method of forward checking improving the search for a solution to the 4-queens problem.

NOTES

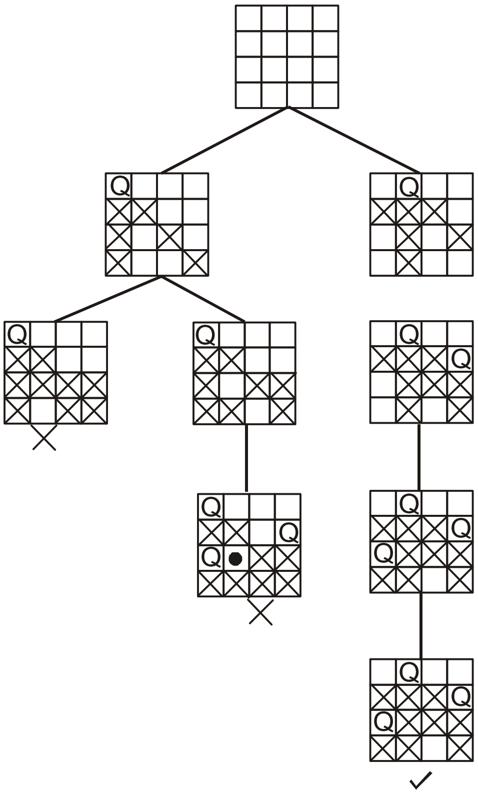


Fig. 2.14 *Forward Checking Method*

In addition to forward checking to enhance the intelligence of the constraint solving agent, certain other possibilities for a heuristic search exist. First, there is a need to check the sequence in which it looks at the variables, for example, in the 4-queens problem, it may attempt at putting a queen in row 2, then one in row 3, one in row 1 and ultimately one in row 4. A solver who takes such extra care is known to be making use of a variable-ordering heuristic. The sequencing of variables can be carried out before the beginning of a search. These variables must be strictly followed at the time of the search. This may sound like a pleasing idea in case there is some more knowledge regarding the problem, for example, a specific variable must be given a value as soon as possible. Optionally, the dynamic sequencing of the variables can be done, as a way of responding to certain collected information regarding the way the search progresses while the search procedure is going on.

One such dynamic ordering procedure is known as 'fail-first forward checking'. The notion is taking maximum advantage of the information collected when performing a forward check during search. In instances where forward checking is known to highlight the fact that a future domain has been made empty in an effective manner, it signifies that it is time to change the present assignment. However, usually, the domain of the variable will be decreased but not essentially made empty. Imagine that among all the future variables, x_f has the maximum values eliminated from D_f. The fail-first approach is known to specify that assigning values to x_f next must be selected. The logic for this is that, with lesser possible assignments for x_f as compared to the other future variables, you will be able to find out very fast whether you are going towards a dead-end. Hence, a better term for this approach will be to 'find out if it is a dead end quickest'. However, this is not as interesting a phrase as 'fail-first'.

An option/addition to a variable ordering is value ordering. Again, the sequence in which values must be assigned to variables must be specified beforehand. This form of alteration of the problem specification can greatly enhance search time. You can also carry out value ordering in a dynamic manner. Imagine that it is possible to allocate values V_c, V_d and V_e to the present variable. Further imagine that, while looking at all the future variables, the entire number of values in their respective domains decreases to 300, 20 and 50 for V_c, V_d and V_e respectively. It can then be specified that ns V_c is assigned at this stage in the search, since it has been successful in retaining the maximum number of values in the future domains. This is dissimilar from variable ordering in two vital methods:

- In case this is a dead end then you will finally be going through all the values for this variable in any case. Therefore, fail-first carries no meaning for values. Instead, try to have your options open as far as possible. This will prove to be useful in case there is a solution ahead of you.
- Not like the variable ordering heuristics, this heuristic is known to carry an additional cost on top of forward checking, since the decrease in domain sizes of future variables for each assignment of the present variable has to be verified. Therefore, it is quite likely that this type of value ordering will make the process gradual. In reality, this is what happens for arbitrarily built binary CSPs. Sometimes, however, employing dynamic value ordering can be extremely helpful.

Applications and Hot Topics of Constraint Solving

Constraint solving is one of the most incredible stories signifying achievement in AI. There have existed several mathematical implementations of CSP techniques. Examples include finding solutions to algebraic existence problems and numerical problems, such as, finding least Golomb rulers: On a ruler mark the integer places, such that no two pairs of marks are at the same distance from each other. The question that arises here is given a specific number of marks, what is the smallest Golomb ruler which can accomodate all of them.

Initially, it was correct that constraint solving was classified as a type of 'mediocre' method. You can obtain reasonably good results for an entire range of problems with the help of constraint solving. The main benefit of constraint solving is the flexibility of using solvers to obtain answers to questions. In the high-IQ problem, it did not consume a lot of time for specifying the problem and getting an answer. Therefore, for non-expert users, the constraint solving method is frequently the first choice. This is the cause for wide implementation of constraint solving in industries. These include making schedule for sporting events, i.e., determining when different teams will play each other in a league. This may sound easy but is extremely complicated (not to forget that a lot of money goes into it). They also involve bin-packing difficulties, for instance, how to fit a particular number of crates into a ship. Constraint solving is growingly performing a crucial role in other sciences, most notably bioinformatics.

A great amount of research is still being carried out to find out the uses of constraint solving. One limitation is the making of CSPs in the first place. As stated earlier, this is an extremely skilled job—the right selection of variables/values/constraints for representing the problem and the sequencing of the variables and values can turn an insolvable problem into a solvable one. Hiring of experts helps in accurately specifying constraint problems in industrial settings. Given this requirement to accurately specify a CSP, some attempts have been made recently to have a software agent to *automatically* remake CSPs. One good method of doing this is adding some implied constraints to the problem specification. These are extra constraints that can be proved for following from the original specification, and therefore can be added with no loss of generality (no solutions will be lost by adding these constraints). Optionally, in cases where you only want to find one solution, it is possible to automatically specialize CSPs, with the expectation that a solution to the specialized solution is easier to find.

Another important topic is detecting the symmetry, and breaking it. For example, in case one can demonstrate that two variables always assume the same values (a symmetry), then the removal of one of these is possible from the problem specification (breaking that symmetry). More such symmetries subtler than this exist, which man is an expert in finding and breaking. Research on how to get a search strategy to automatically find and break symmetry before and during a CSP search is in still going on.

2.8 MEAN END ANALYSIS

Means End Analysis (MEA) is a strategy which is brought into use in Artificial Intelligence to control search in problem solving computer programs. It has been in use since the 1950s as a creativity tool.

It is also a technique used at least since the 1950s as a tool for creativity, most often stated in engineering books on design techniques. MEA is also a method of clarifying one's thoughts when one comes across a mathematical proof.

MEA is a technique that was ianitially made use of in Newell and Simon's General Problem Solver (GPS). GPS is a problem-solving method in which comparisons are drawn between the present state and the goal state. The difference between them is categorized into subgoals for achieving the goal state using the operators at hand. MEA is one among the several weak search techniques that have been used in both cognitive architectures and more general artificial intelligence research.

The following architectures enable the utilization of MEA:

- Planning and Learning Architecture (Prodigy)
- Problem Space Architecture (Soar)
- Modular, Integrated Architecture (ICARUS)

Planning and Learning Architecture (Prodigy)

This is a system which has been designed to unify problem solving, planning and multiple learning techniques in a combined architecture. The Prodigy architecture assumes the form of a general problem solver in which six learning modules are coupled in a tight manner. The Prodigy architecture can be best depicted with the help of a graph.

The problem solver is a search engine that looks over a problem space defined by the present domain and operators. As it performs its search, a problem-solving trace is made by it. This incorporates every step of the search (including paths that were later dropped) and its own logic at the time that is about the present state of the search. This trace can be made use of by the learning modules.

Prodigy's problem solver makes use of a MEA for solving problems. Differences are decided by drawing comparisons between the current state and the goal state. Not like maximum MEA, Prodigy can have many goals that exist simultaneously, that would need to be considered. The system initially decides the goal that needs to be achieved and then produces differences for a single goal as soon as this determination has been made. These differences can be easily calculated as all knowledge is denoted in PDL. This is both uniform and can be penetrated. Once the differences are calculated for the current goal, operators are proposed on the basis of their capability to decrease the dissimilarities. As Prodigy makes use of STRIPS-like operators, this step matches with the current operators' add-lists being scanned for determining whether an operator exerts any predicates which, post binding, will lead to the elimination of a difference (the assertion of a goal state conjunct). Choice among the operators is intervened by control rules. When the control rules are absent, in keeping with Prodigy's general commitment technique, the intervention between operators defaults to a depth-first consideration of each.

When the control rules are absent, the search defaults to depth-first MEA. Nodes in the problem space are determined as the set of goals and the state of the world, both indicated in first-order predicate logic. The search goes on through the problem space till a node is found that achieves the top-level goal, with the help of the following algorithm:

- Decision Phase
    - o Determining the node for expanding next (by either control rules or DFS).
    - o Determining new goal from this node.
    - o Selecting operator for achieving the goal.
    - o Binding operator parameters.
- Expansion Phase
    - o If it is possible to apply the operator, apply it. Or else, create subgoal on non-matching preconditions.

Control Rules: Control rules are made use of for the following three purposes:

1. To improve the search efficiency
2. To improve the solution quality
3. To direct the problem solver along normally-unexplored paths

In order to search, Prodigy presumes that the search will be directed by exclusive control knowledge to make crucial decisions. This presumption is known as the casual commitment technique. It denotes that the problem solver will not try refined conduct when that kind of control knowledge is absent.

NOTES

Control rules can be categorized into a left-hand side that is temporarily found to be matching variables against the present state axioms. Another category is the right-hand side action that denotes whether to SELECT, REJECT or PREFER a specific candidate rule. A control decision is made on the basis of these indications with the determination of a new candidate node.

Problem Space Architecture (Soar)

Soar was made use of to imagine the reality of a universal weak method, an approach that stated that the search technique must occur as a result of the communication between the structure of the agent and the projected task. The search strategy selected is presumed as being *weak*; i.e., the agent does not possess much knowledge regarding the task environment. Therefore, any of the weak methods might occur in Soar when this universal weak method and the task interact with each other. The benefit of this kind of an approach is that it refrains from program synthesis: the behavior results knowledge and task interact instead of being programmed in a precise and detailed manner.

A summary of certain weak methods that have been demonstrated in Soar with the help of the idea of a universal weak technique is as follows:

- Heuristic Search
- Operator Subgoaling
- Waltz Constraint Propagation
- Means-End Analysis
- Generate and Test
- Breadth First Search
- Depth first search
- Look-ahead Search
- Simple and Steepest Ascent Hill Climbing
- Progressive Deepening
- Mini-Max
- Alpha-Beta Pruning
- Iterative Deepening
- Branch and Bound
- Best first search
- Macro-operators

Modular Integrated Architecture (ICARUS): Daedalus

Daedalus makes use of a different type of MEA for generating plans. This constituent calls for Labyrinth for retrieving suitable operators or stored plan on the basis of the problem's preconditions, postconditions or the variations that it decreases. In case, Daedalus happens to detect a loop or a dead end, it is known to recede and retrieve a separate operator, that produces a heuristic depth first search with the help of means-ends space. In case the Labyrinth gives back an entire plan, Daedalus performs a kind of derivational analogy, to check the validity of the problem at hand.

This algorithm of priority-first execution will appear to open the likelihood of starvation of lower-priority tasks. The explanation is not very clear about this issue, therefore no presumptions can be drawn either way.

Problem-Solving as Search

A vital feature of wise behaviour as performed in studies in AI is *goal-based* problem solving. It is an underlying structure in which it is possible to give an explanation of the solution of a problem by looking for a series of *actions* leading to a desired goal. A goal-seeking system is to be connected to its external environment with the help of sensory channels which help it to receive information regarding the environment and the motor channels with the help of which it influences the environment (the word 'afferent' describes 'inward' sensory flows, and 'efferent' describes 'outward' motor commands). Moreover, the system contains a certain method to store in a *memory* information regarding the *condition* of the environment (afferent information) and information regarding actions (efferent information). The capacity of achieving goals is dependent on developing relationships, simple or complex, between specific changes in states and specific actions that will result in these changes. Search is the method of discovery and assembly of a series of actions which will lead from a given state to a desirable state. While this technique can suit machine learning and problem solving, it is not always recommended for humans (e.g., cognitive load theory and its implications).

Working Methodology of MEA

The MEA technique is an approach that is used for controlling search in problem solving. Given a current state and a goal state, an action is chosen that would reduce the *difference* between the two. The action is carried out on the current state to give rise to a new state, and the process is implemented in recursion to this new state and the goal state.

Notice that, for MEA to be effective, the goal-seeking system should possess a method to associate with any form of noticeable difference, those actions which are valid for decreasing that difference. It should also have the means to detect the progress that is being made (the changes in the differences between the real and the desirable state), since certain tried series of actions might prove unsuccessful and, hence, certain optional series can be tried.

When knowledge is obtainable that concerns the vitality of differences, the most vital difference is selected first to further enhance the average performance of MEA over other brute-force search techniques. However, even when differences are not ordered as per the importance, MEA enhances over other search heuristics (again in the average case) by focussing the problem solving on the real differences between the present state and that of the goal.

Check Your Progress

7. What is branch and bound?
8. Name any three problems for which branch and bound is used.
9. State a method to search AND-OR graphs.
10. What is the fundamental principle of the technique of problem reduction?
11. What is constraint satisfactions?
12. What are the parts of the CSPs?
13. Name the architectures that enable the utilization of MEA.
14. What is a problem solver?

2.9 BASIC CONCEPT OF KNOWLEDGE REPRESENTATION

Knowledge representation can be understood as follows:

(x): animal (x) fi **eat (x, fruit) eat (x, meat)**

- What is knowledge?
- How do we search through knowledge?
- How do we get knowledge?
- How do we get a computer to understand knowledge?

What AI researchers call 'knowledge' appears as data at the level of programming. Data becomes knowledge when a computer program represents and uses the meaning of some data. Many knowledge-based programs are written in the LISP programming language, which is designed to manipulate data as symbols.

Knowledge may be declarative or procedural. Declarative knowledge is represented as a static collection of facts with a set of procedures for manipulating the facts. Procedural knowledge is described by an executable code that performs some action. Procedural knowledge refers to 'how-to' do something. Usually, there is a need for both kinds of knowledge representation to capture and represent knowledge in a particular domain.

First-Order Predicate Calculus (FOPC) is the best-understood scheme for knowledge representation and reasoning. In FOPC, knowledge about the world is represented as objects and relations between objects. Objects are real-world things that have individual identities and properties, which are used to distinguish the things from other objects. In a first-order predicate language, knowledge about the world is expressed in terms of sentences that are subject to the language's syntax and semantics.

Knowledge in AI

The core concepts of research in Artificial Intelligence (AI) are knowledge representation and knowledge engineering. Objects, categories, properties and relations between objects; situations, states, events and time; causes and effects;

knowledge about what other people know; and other less factors less researched are some issues that AI must represent.

In artificial intelligence research, commonsense knowledge is the collection of facts and information that an ordinary person is expected to know. The commonsense knowledge problem is the ongoing project in the field of knowledge representation (a sub-field of artificial intelligence) to create a commonsense knowledge base: a database containing all the general knowledge that most people possess, represented in a way that it is available to artificial intelligence programs that use natural language or make inferences about the ordinary world.

The following listed are some of the most difficult problems in knowledge representation:

- **Default Reasoning:** Working assumptions can be defined as the things that people know. For instance, during an after-dinner conversation, if people talk about a football match, people typically visualize large crowds, screaming and cheering fans, aggression between players on the field. Now, this may not be true for a community-level match played for charity. This problem of assumptions was identified by John McCarthy in 1969 and termed the qualification problem. That is, for any 'common sense' rule identified by AI researchers, there exists a considerable portion of exceptions. McCarthy said that nothing can be absolutely true or false in the way required by abstract logic. Several solutions to this problem have been brought up in the course of AI research.
- **Range of Common Sense Knowledge:** The intensity and breadth of common sense knowledge is immense in AI research and hence considered an issue to be dealth with. Any job that warrants common sense knowledge is considered 'AI complete', which means it is to be done as efficiently as done by a human being. This means the machine is to be made as intelligent as a human being. Such tasks can include machine translation, object recognition and text mining. In order to be able to do these tasks as good as a human being, the machine has to be aware ('know') of what the text is talking about or what objects it may be looking at. This is generally not possible unless the machine is familiar with all the same concepts known by a human being.
- **Sub-Symbolism in Common Sense Knowledge:** Facts or statements do not always make up what people know and could actually say out loud. For instance, Vishwanathan Anand, Indian chess player, might avoid a certain chess position because it is 'too exposed' or a museum curator may take one look at a coin collection and realize it is fake. These are intuitions (gut feelings) represented in the brain at the sub-conscious and sub-symbolic level. It is this intuition that informs, supports and gives a context for symbolic and conscious knowledge. However, there are issues that exist with sub-symbolic reasoning, and it is anticipated that AI will come up with ways to represent this knowledge.

  Many times, the success and development of an AI system is attributed to the volume and quality of world knowledge provided to it. Several highly structured knowledge bases have come up whose collaborative nature has caused a considerable increase in the volume and quality of world knowledge that can be used in AI applications.

Here, knowledge is another term for data, but knowledge is more than data or information. Data is raw facts derived at by discovering and gathering of raw information and researching on that information. This data is converted into information by organizing it such that it can be made easy to draw conclusions.

The distinction between knowledge and data will be made clear from the following example. A scientist working on a disease-causing organism works using both knowledge and data. The data is the organism's record, that is, its history, response to certain drugs, treatment regimen and so on. Knowledge is what the scientist has, usually gathered and learned as a student, during internship, specialization and practice. Knowledge will include facts, beliefs, prejudices and heuristic knowledge, which makes knowledge dynamic, as opposed to information, which is static. Knowledge involves introducing changes, either by laying the ground for action or by making an individual suitable for different or effective action.

Knowledge lays the background for two possible outcomes:

- Gather possible sources of actions to ascertain if a particular course of action will give the desired result.
- Use this judgment to determine how the action will be carried out.

The theme of intelligence is knowledge. It is needed for the following purposes:

- Use, or understand, a language.
- Take decisions.
- Recognize objects.
- Interpret situations.
- Plan strategies.

An AI system must be capable of

- Storing knowledge,
- Applying knowledge that is preserved for solving problems and
- Acquiring knowledge with the help of experience.

AI systems have three components. They are as follows:

(i) Representation

(ii) Reasoning

(iii) Learning

The two roles played by knowledge in AI programs are as follows:

(i) **Essential Knowledge:** AI programs define the search and criteria to determine a solution to a problem.

(ii) **Heuristic Knowledge:** AI programs make the reasoning procedure more efficient by optimizing the procedure to look for the best solution.

Knowledge is often taken through transcribed content or artifacts, derived from knowledge from other people, such as facts, concepts, processes, procedures and principles. Thus, the purpose of artifacts is to create knowledge in the learning process. In turn, knowledge creates new artifacts. There are five basic types of artifacts of knowledge:

(i) Facts

(ii) Concepts

(iii) Processes

(iv) Procedures

(v) Principles

(i) **Facts:** These are specific and unique data or instances.

(ii) **Concepts:** These are classes of items, words or ideas identified by a common name, and include multiple specific examples and share common features. Concepts are of types: concrete and abstract.

(iii) **Processes:** These are sequences of events or activities that describe the working of things, rather than their functioning. Processes are of two types: business processes, which describe the workflow, and technical processes, which describe how things work. In general, processes determine how anything works.

(iv) **Procedures:** These are a series of step-by-step actions and decisions that cause a task to be achieved. Actions are of two types: linear and branched.

(v) **Principles:** These are guidelines, rules and parameters that determine the artifacts of knowledge. They encompass both what should be done as well as what should not be done. Principles enable one to make predictions and deduce implications. They are the fundamental building blocks of causal and/or theoretical models.

These five artifacts are, in turn, used in the knowledge creation process to create two types of knowledge: declarative and procedural.

Declarative Knowledge

Descriptive knowledge (also known as declarative knowledge or propositional knowledge) is the category of knowledge expressed in declarative sentences or indicative propositions. This distinguishes descriptive knowledge from what is commonly known as "know-how", or procedural knowledge (the knowledge of how, and especially how best, to perform some task), and "knowing of", or knowledge by acquaintance (the knowledge of something's existence).

Declarative knowledge is knowledge of facts, for example, New Delhi is the capital of India. This is as opposed to procedural knowledge that involves the possession of 'how-to' knowledge, such as how to cook pasta or how to reverse a car. Declarative models are representations of objects and events and their relation with other objects and events. They focus on the 'why' rather than the 'how'. It allows us to think and talk about the world.

Take the example of a mobile robot that works as security backup in a building. An AI system based on declarative knowledge will contain a map of the building and information about the fundamental actions that can be done by the robot (for instance, moving back and forth, turning around and stopping) and use this information into a planning algorithm to use these actions to achieve the required goals.

The special features of declarative knowledge are as follows:

- It involves the knowledge or possession of information that is objective: either true or false.
- It is assertion oriented.
- It identifies objects and events by specifying its characteristic properties.
- It lays stress upon the properties and not to the actions needed to obtain a result.
- Its models include propositions and schemas.

Declarative knowledge is further divided into:

- **Episodic Knowledge:** It refers to memory for certain episodes, such as the context of where, when, with whom, usually measured by accuracy measures.
- **Semantic Knowledge:** It refers to the memory for worldly knowledge, facts, meaning of words and so on. For instance, semantic knowledge refers to knowing that alphabetically the first month of the year is April but chronologically it is January.

Procedural Knowledge

Procedural knowledge involves knowledge of the following aspects:

- Formal language
- Symbolic representations
- Knowledge of rules, algorithms, and procedures

Procedural knowledge, also known as **know-how, involves** how to perform a **task**. Know-how differs from other kinds of knowledge, such that it can be applied directly to a task. This knowledge focuses on jobs that must be performed to attain a certain target or goal. This knowledge is often difficult to put into so many words than declarative knowledge.

The emphasis in procedural knowledge is on production-based hierarchical or information processing approaches. A combination of productions creates production systems. By productions, we mean the building blocks of procedural knowledge that consist of a condition and an action, more like an IF and THEN statement. For example, IF the sky turns grey and cloudy, THEN it will rain. IF the traffic light turns red, THEN apply brake. IF you drink hot tea, THEN your tongue will get scalded. Productions are conditional IF-THEN branches. In a production system whenever an OR condition in the system is satisfied, the system is allowed to execute or perform a specific action which may be specified under that rule. If the rule is not fulfilled, it may perform another action. This can be put simply as follows:

WHEN (condition) is satisfied, PERFORM (action)

The advantages of procedural knowledge are as follows:

- Involvement of hands-on experience
- Practice at solving problems

- Understanding the limitations of a specific solution
- Representation of heuristic knowledge
- Facilitation of default reasoning
- Enable modeling of side effects of actions

The disadvantages of procedural knowledge are as follows:

- Not all cases may be represented.
- Not all deductions may be correct.
- Changes in knowledge base might have far-reaching effects.
- Controlling information is cumbersome.
- Its job dependence nature tends to be less general than declarative knowledge. For example, computer experts proficient in more than one language can use their knowledge about a computer algorithm in multiple languages while a programmer proficient only Visual Basic might know about a specific implementation of that algorithm written only in Visual Basic.

Approaches to AI Goals

There are four approaches to the goals of AI: (1) Computer systems that act like humans; (2) Programs that simulate the human mind; (3) Knowledge representation and mechanistic reasoning; and (4) Intelligent or rational agent design. The first two approaches focus on studying humans and how they solve problems, while the latter two approaches focus on studying real-world problems and developing rational solutions regardless of how a human would solve the same problems.

Programming a computer to act like a human is a difficult task and requires that the computer system be able to understand and process commands in natural language, store knowledge, retrieve and process that knowledge in order to derive conclusions and make decisions, learn to adapt to new situations, perceive objects through computer vision and have robotic capabilities to move and manipulate objects. Although this approach was inspired by the Turing Test, most programs have been developed with the goal of enabling computers to interact with humans in a natural way rather than passing the Turing Test.

Some researchers focus instead on developing programs that simulate the way in which the human mind works on problem-solving tasks. The first attempt to imitate human thinking was the Logic Theorist and the General Problem Solver programs developed by Allen Newell and Herbert Simon. Their main interest was in simulating human thinking rather than solving problems correctly. Cognitive science is the interdisciplinary field that studies the human mind and intelligence. The basic premise of cognitive science is that the mind uses representations that are similar to computer data structures and computational procedures that are similar to computer algorithms that operate on those structures.

Other researchers focus on developing programs that use logical notation to represent a problem and use formal reasoning to solve a problem. This is called the ‘logicist approach’ to developing intelligent systems. Such programs require huge computational resources to create vast knowledge bases and to perform complex reasoning algorithms. Researchers continue to debate whether this strategy will lead to computer problem solving at the level of human intelligence.

Still other researchers focus on the development of 'intelligent agents' within computer systems. Russell and Norvig defines these agents as 'anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.' The goal for computer scientists working in this area is to create agents that incorporate information about the users and the use of their systems into the agents' operations.

Fundamental System Issues

A robust AI system must be able to store knowledge, apply that knowledge to the solution of problems and acquire new knowledge through experience. Among the challenges that face researchers in building AI systems, there are three that are fundamental: knowledge representation, reasoning and searching and learning.

How do we represent what we know?

- **Knowledge** is a general term. It is a progression that starts with *data* which is of limited utility. By organizing or analysing the data, we understand what the data means, and this becomes **information**. The interpretation or evaluation of information yield **knowledge**. An understanding of the principles within the knowledge is **wisdom**.

  An answer to the question, *'how to represent knowledge'*, requires an analysis to distinguish between knowledge 'how' and knowledge 'that'.

  - knowing *'How to do something'.*

    e.g., 'How to drive a car' is procedural knowledge.
  - knowing *'that something is true or false'.*

    e.g., 'That is the speed limit for a car on a motorway' is declarative knowledge.
- **Knowledge and representation** are distinct entities that play central but distinguishable roles in the intelligence system.
  - Knowledge is a description of the world. It determines a system's competence by what it knows.
  - Representation is the way knowledge is encoded. It defines the performance of a system in doing something.
- Different types of knowledge require different kinds of representation.

  Knowledge Representation **models/mechanisms** are often based on:

  ▪ Logic ▪ Rules ▪ Frames▪ Semantic net
- Different types of knowledge require different kinds of ***reasoning.***

Knowledge Progression

Knowledge progression, shown in Figure 2.15, and its related concept are discussed as follows:

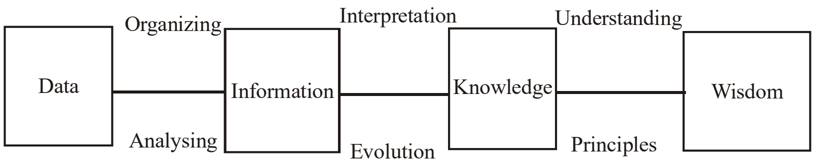


Fig. 2.15 *Knowledge Progression*

- **Data** is viewed as a collection of *disconnected facts.*

  E.g.: It is raining.

- **Information** emerges when relationships among facts are established and understood. Providing answers to 'who', 'what', 'where', and 'when' gives the relationships among facts.

  E.g.: The temperature dropped 15 degrees and it started raining**.**

Knowledge emerges when relationships among patterns are identified and understood. Answering to 'how' gives the relationships among patterns.

E.g.: If the humidity is very high, temperature drops substantially, and then atmosphere holds the moisture, so it rains.

Wisdom is the understanding of the principles of relationships that describe patterns. Providing the answer for 'why' gives understanding of the interaction between patterns.

E.g.: Understanding of all the interactions that may happen between raining, evaporation, air currents, temperature gradients and changes.

Let's look at various kinds of knowledge that might need to represent AI systems:

- **Objects**

  Objects are defined through facts.

  E.g.: Guitars with strings and trumpets are brass instruments.

- **Events**

  Events are actions.

  E.g.: Vinay played the guitar at the farewell party.

- **Performance**

  Playing the guitar involves the behaviour of the knowledge about how to do things.

- **Meta-knowledge**

  Knowledge about what we know. To solve problems in AI, we must represent knowledge and must deal with the entities.

- **Facts**

  Facts are truths about the real world on what we represent. This can be considered as knowledge level.

Knowledge Model

Knowledge model defines that as the level of 'connectedness' and 'understanding' increases, our progress moves from data through information and knowledge to wisdom (Refer Figure 2.16).

The model represents transitions and understanding.

- The transitions are from data to information, information to knowledge, and finally knowledge to wisdom;
- The support of understanding is the transitions from one stage to the next stage.

The distinctions between data, information, knowledge and wisdom are not very discrete. They are more like shades of gray, rather than black and white.

Data and information deal with the past and are based on gathering facts and adding context.

Knowledge deals with the present and that enables us to perform.

Wisdom deals with the future vision for what will be rather than for what it is or it was.

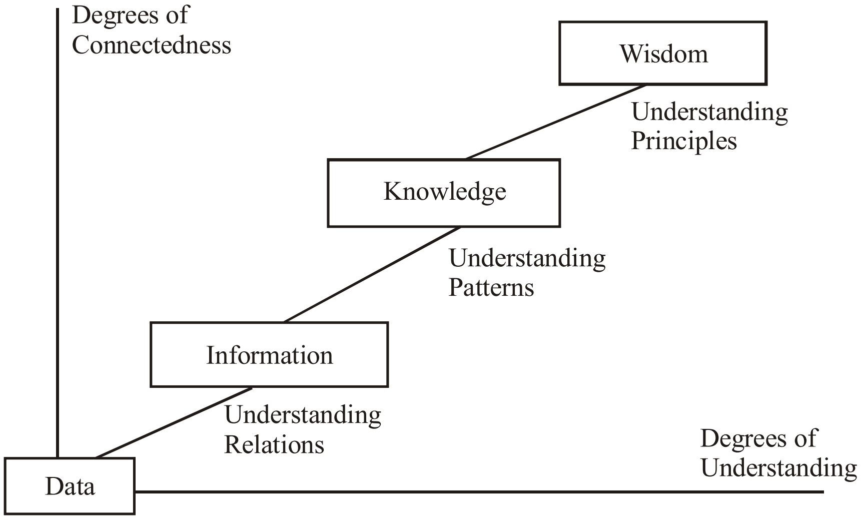


Fig. 2.16 *Knowledge Model*

Knowledge Typology Map

According to the topology map:

1. Tacit knowledge derives from the experience, doing action and subjective insight.
2. Explicit knowledge derives from principle, procedure, process and concepts.
3. Information derives from data, context and facts.
4. Knowledge conversion derives from internalization, socialization, wisdom, combination and externalization.

Finally knowledge derives from the tacit and explicit knowledge and information.

Let's see the contents of the topology map (Refer Figure 2.17):

- **Facts** are data or instance that is specific and unique.
- **Concepts** are classes of items or words or ideas that are known by a common name and share common features.
- **Processes** are flow of events or activities that describes how things will work rather than how to do things.
- **Procedures** are a series of step-by-step actions and decisions that give the result of a task.
- **Principles** are guidelines, rules and parameters that are going to rule or monitor.

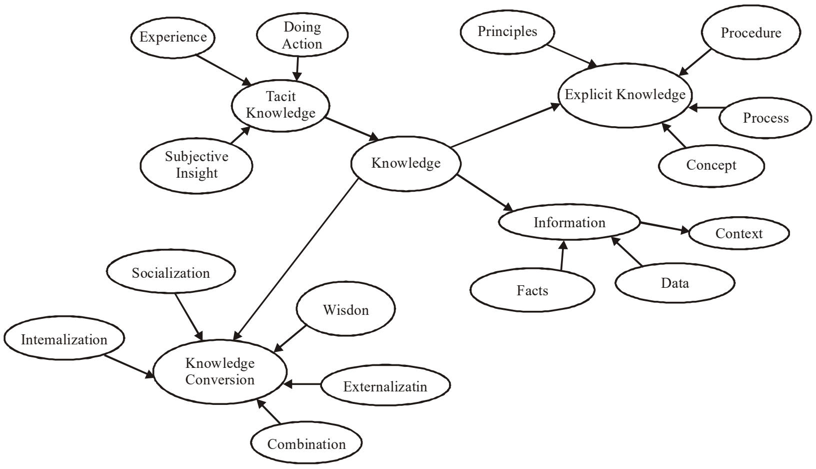


Fig. 2.17 *Knowledge Typology Map*

Principles are the basic building blocks of theoretical models and allow for making predictions and drawing implications. These artifacts are supported in the knowledge creation process for creating two of knowledge types: declarative and procedural, which are explained below.

- Knowledge Type

  Cognitive psychologists sort knowledge into **declarative** and **procedural** categories and some researchers add **strategic** as a third category.

Procedural knowledge	Declarative knowledge
• *Examples:* Procedures, rules, agendas, models	• *Example:* Concepts, objects, strategies, facts, propositions, assertions, semantic nets, logic and descriptive models
• Focuses on tasks that must be performed to reach a particular objective or goal	• Refers to representations of objects and events; knowledge about facts and relationships
• Knowledge about *'how* to do something'; *e.g.,* to determine if Peter or Robert is older, first find their ages	• Knowledge about *'that something* if *is true or false'. e.g.,* A car has four tyres; Peter is older than Robert

Note : About procedural knowledge, there is some disparity in views. One view is that it is close to tacit knowledge; it manifests itself in the doing of something, yet cannot be expressed in words; e.g., we read faces and moods.

Another view is that it is close to declarative knowledge; the difference is that a task or method is described instead of facts or things. All declarative knowledge is explicit knowledge; it is knowledge that can be and has been articulated.

Strategic knowledge is considered to be a subset of declarative knowledge.

Knowledge Representation and Reasoning

Knowledge Representation (KR) is basically a replacement for an actual thing. It enables an entity to decide the end result by thinking rather than acting. KR is a set of answers to questions. It is a medium for pragmatically efficient computation.

According to John Sowa, in *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, 'knowledge representation is a multidisciplinary subject that applies theories and techniques from three other fields: 1. Logic provides the formal structure and rules of inference. 2. Ontology defines the kinds of things that exist in the application domain. 3. Computation supports the applications that distinguish knowledge representation from pure philosophy.'

According to David Poole, Alan Mackworth and Randy Goebel, in *Computational Intelligence: A Logical Approach*, 'in order to use knowledge and reason with it, you need what we call a Representation and Reasoning System (RRS). A representation and reasoning system is composed of a language to communicate with a computer, a way to assign meaning to the language, and procedures to compute answers given input in the language. Intuitively, an RRS lets you tell the computer something in a language where you have some meaning associated with the sentences in the language, you can ask the computer questions, and the computer will produce answers that you can interpret according to the meaning associated with the language. . . . One simple example of a representation and reasoning system . . . . is a database system. In a database system, you can tell the computer facts about a domain and then ask queries to retrieve these facts. What makes a database system into a representation and reasoning system is the notion of semantics. Semantics allows us to debate the truth of information in a knowledge base and makes such information knowledge rather than just data.'

Knowledge representation and reasoning is a domain in Artificial Intelligence (AI). Its scope involves how to formally 'think'. This means using a symbol system to represent that which can be discussed about, and functions that is, or is not, within the domain of discourse that permits formal reasoning regarding objects within the domain of discourse to take place. Usually, some amount of logic is employed to provide formal semantics of the functioning of reasoning to symbols in the domain of discourse, and to supply quantifiers and modal operators that give meaning to the sentences in the logic.

Overview

Rules, frames, semantic networks and tagging are techniques of representation that have come up from human information processing theories. Knowledge representation aims to represent knowledge such that it can facilitate the drawing of conclusions from knowledge. The issues that arise in knowledge representation from the viewpoint of AI are as follows:

- Representation of knowledge by people
- Nature of knowledge and its representation
- Representation schemes vis-à-vis a particular or a general-purpose domain
- Nature of expression of a representation scheme or formal language
- Declarative or procedural scheme

Scanty discussion and research exists in knowledge representation–related issues. Some of the well-known problems that exist are as follows:

- **Spreading Activation:** Deals with issues in navigating a network of nodes
- **Subsumption:** Deals with selective inheritance
- **Classification:** Deals with classification of a product under both genre and sub-genre

Knowledge representation refers to representations aimed at information processing by modern computers, and specifically, for representations that consist of explicit objects (the class of all chimpanzees or Johnny as a specific entity), and of claims regarding them (Johnny is a chimpanzee, or all chimpanzees are cute and know tricks). In this case, representing knowledge so explicitly enables computers to arrive at conclusions from already-stored knowledge (Johnny is cute and knows tricks).

Glimpse into the History of Knowledge Representation and Reasoning

- Knowledge representation methods, such as heuristic knowledge, neural networks and theorem proving, were tried in the 1970s and early 1980s, with varying degrees of success. Also, various medical diagnoses and games such as chess were major application areas.
- In 1972, Prolog was developed. It represented propositions and rudimentary logic that could derive conclusions from established premises.
- The 1980s witnessed the birth of formal computer knowledge, languages and systems. Major projects, such as the 'Cyc' project, which is ongoing, tried to encode large bodies of general knowledge. This project went through a large encyclopedia, encoding the information needed by readers to understand basic physics, notions of time, causality, motivation, commonplace objects and classes of objects.
- Around the same time, much larger databases of language information were being built in computational linguistics, and these coupled with faster processing speed and capacity made intense knowledge representation more feasible.
- In the 1980s, KL-ONE targeted at knowledge representation itself.
- In 1995, the Dublin Core standard of metadata was conceived. Languages were developed to represent document structure, such as SGML (from which HTML descended) and later XML. These enabled the retrieval of information and efforts in data mining.
- Web development has included development of XML-based knowledge representation languages and standards, including RDF, RDF Schema, DARPA Agent Markup Language (DAML) and Web Ontology Language (OWL).
- Various and notations are being developed to represent knowledge, which have their foundation in logic and mathematics.

Properties of Knowledge Representation Systems

A knowledge representation system must possess the following properties:

- **Representational Adequacy:** It denotes the ability to represent the requisite knowledge.

- **Inferential Adequacy:** It refers to the simplicity with which inferences can be drawn using represented knowledge.
- **Inferential Efficiency:** It refers to the ability to make complex deductions based on knowledge in a sophisticated language. It is generally believed that the more complex the deductions, the less efficient will be the entity doing the reasoning.
- **Acquisitional Efficiency:** It refers to the ability to acquire knowledge with the help of automatic methods rather than depend on human intervention.

Issues in Knowledge Representation

The following issues must be considered when using a knowledge representation technique:

- **Important Attributes:** Are there any attributes that occur in many different types of problem? If yes, we need to ensure they are handled appropriately in each of the mechanisms we propose.
- **Relationships:** What about the relationship between the attributes of an object, such as, inverses, existence, techniques for reasoning about values and single valued attributes?
- **Granularity:** It indicates the level at which knowledge should be represented and the primitives.

Methods of Knowledge Representation

The main methods of knowledge representation are as follows:

- Logic
- Semantic network
- Frames
- Production rules
- Scripts

Knowledge Representation Using Logic

Logic, a sub-division of philosophy, is one of the oldest methods of knowledge representation. Some of the most common definitions of logic are as follows:

- It is a reasoning process that has its foundations in science and is used to distinguish correct reasoning from incorrect reasoning.
- It is a process that reasons the problem to derive a conclusion.

Logic can be divided into two types:

1. Propositional logic calculus
2. Predicate logic calculus

1. Propositional Logic Calculus

A propositional statement is either true or false, in which logic is used to arrive at the solution. Calculus is used for computation of such problems.

Propositional logic uses symbols for computing, such as, logic gates for Boolean algebra. The following are the logic gates:

- NOT gate
- AND gate
- OR gate

2. Predicate Logic Calculus

Predicate logic calculus is used in AI programming. Its rules and concepts are the similar to that used by propositional logic calculus. Predicate programming is similar to the function prototype programming in C and C++. In AI, declarative programming languages employ predicate logic calculus rather than propositional logic calculus. A declarative program is a combination of facts and rules. Rules are relationships of facts.

2.9.1 Representation and Mappings

When we collect knowledge we are faced with the problem of how to record it. And when we try to build the knowledge base we have the similar problem of how to represent it.

We could just write down what we are told but as the information grows, it becomes more and more difficult to keep track of the relationships between the items.

Let us start with the observation that we have no perfect method of knowledge representation today.

The **knowledge representation** problem concerns the mismatch between human and computer 'memory', i.e., how to encode knowledge so that it is a faithful reflection of the expert's knowledge and can be manipulated by the computer.

We call these representations of knowledge **knowledge bases**, and the manipulative operations on these knowledge bases, **inference engine** programs.

What to Represent?

- **Facts:** Truths about the real world and what we represent. This can be regarded as the base knowledge level.
- **Representation of the facts:** That which we manipulate. This can be regarded as the symbol level since we usually define the representation in terms of symbols that can be manipulated by programs.
- **Simple Representation:** Simple way to store facts. Each fact about a set of objects is set out systematically in columns. Little opportunity for inference. Knowledge basis for inference engines.

Framework of Knowledge Representation

A computer requires a well-defined problem description to process and provide a well-defined acceptable solution. To collect fragments of knowledge, we first need to formulate a description in our spoken language and then represent it in formal language so that the computer can understand. The computer can then use an algorithm to compute an answer. This process is shown in Figure 2.18.

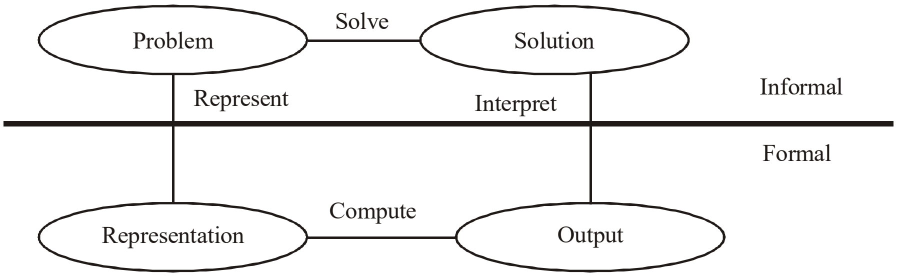


Fig. 2.18 *Knowledge Representation Framework*

The steps are:

- The informal formalism of the problem takes place first.
- It is then represented formally and the computer produces an output.
- This output can then be represented in a informally described solution that the user understands or checks for consistency.

Problem solving requires the following:

- Formal knowledge representation
- Conversion of informal (implicit) knowledge to formal (explicit) knowledge

Knowledge and Representation

Problem solving requires a large amount of knowledge and some mechanism for manipulating that knowledge. Knowledge and representation are distinct entities and play a central but distinguishable roles in the intelligence system.

- Knowledge is a description of the world; it determines a system's competence by what it knows.
- Representation is the way knowledge is encoded; it defines the system's performance in doing something.

In simple words, we:

- Need to know about things we want to represent, and
- need some means by which we can manipulate things.
  - **Objects** - Facts about objects in the domain.
  - **Events** - Actions that occur in the domain.
  - **Performance** - Knowledge about how to do things.
- Know things to represent
  - **Meta-knowledge -** Knowledge about what we know.
- Need means to manipulate.
  - **Requires some formalism -** To what we represent.

Thus, knowledge representation can be considered at two levels:

(i) *Knowledge level* at which facts are described, and

(ii) *Symbol level* at which the representations of the objects, defined in terms of symbols, can be manipulated in the programs.

Note: A good representation enables fast and accurate access to knowledge and understanding of the content.

Representation of Facts

Representations of facts are those which we manipulate. This can be regarded as the symbol level since we normally define the representation in terms of symbols that can be manipulated by programs. We can structure these entities in two levels (Refer Figure 2.19):

Knowledge Level: At which the facts are going to be described.

The Symbol Level: At which representations of objects are going to be defined in terms of symbols that can be manipulated in programs.

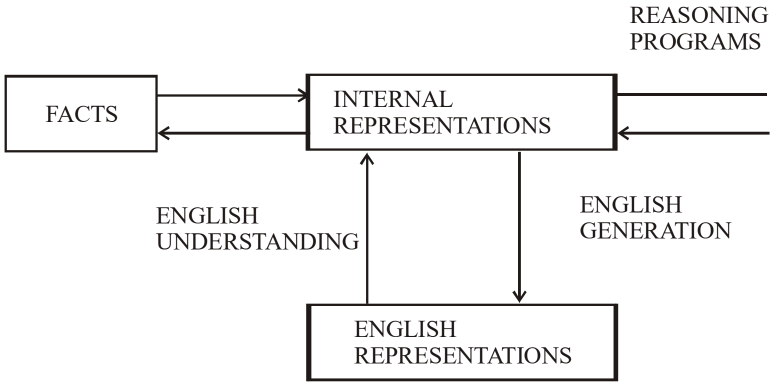


Fig. 2.19 *Two Entities in Knowledge Representation*

Natural language (or English) is the way of representing and handling the facts. Logic enables us to consider the following fact:

spot is a dog represented as *dog*(*spot*)

We infer that all dogs have tails

: *dog*(*x*) *has_a_tail*(*x*)

According to logical conclusion

has_a_tail(*Spot*)

Using a backward mapping function

Spot has a tail can be generated.

The available functions are not always one to one but are many to many which are a characteristic of English representations. The sentences *All dogs have tails* and *every dog has a tail* – both say that each dog has a tail, but from the first sentence, one could say that each dog has more than one tail and try substituting teeth for tails. When an AI program manipulates the internal representation of facts these new representations can also be interpretable as new representations of facts.

Consider the classic problem of the mutilated chessboard. The problem in a normal chessboard, the opposite corner squares, have been eliminated. The given task is to cover all the squares on the remaining board by dominoes so that each domino covers two squares. Overlapping of dominoes is not allowed. Consider three data structures:

The first two data structures are shown in Figure 2.20 and the third data structure is the number of black squares and the number of white squares. The first diagram loses the colour of the squares and a solution is not easy. The second preserves the colours but produces no easier path because the numbers of black and white squares are not equal. Counting the number of squares of each colour, giving black as 32 and the number of white as 30, gives the negative solution as a domino must be on one white square and one black square; thus the number of squares must be equal for a positive solution.

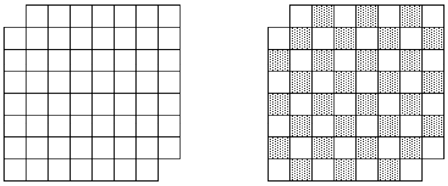


Fig. 2.20 *Mutilated Checker*

Using Knowledge

We have briefly discussed above where we can use knowledge. Let us consider how knowledge can be used in various applications.

Learning

Acquiring knowledge is learning. It simply means adding new facts to a knowledge base. New data may have to be classified prior to storage for easy *retrieval,* interaction and inference with the existing facts and has to avoid redundancy and replication in the knowledge. These facts should also be updated.

Retrieval

Using the representation scheme shows a critical effect on the efficiency of the method. Humans are very good at it. Many AI methods have tried to model humans.

Reasoning

Get or infer facts from the existing data.

If a system only knows that:

- Ravi is a jazz musician.
- All jazz musicians can play their instruments well.

If questions are like this

Is Ravi a jazz musician? *OR*

Can jazz musicians play their instruments well?

then the answer is easy to get from the data structures and procedures.

However, questions like

Can Ravi play his instrument well?

require reasoning. The above are all related. For example, it is fairly obvious that learning and reasoning involve retrieval; *etc.*

2.9.2 Approaches to Knowledge Representation

We discuss the various concepts relating to approaches to knowledge representation in the following sections.

Knowledge Representation Using Natural Languages

The processing of natural languages provides the ability to read and understand the languages spoken by humans to machines. Researchers believe that an effective natural language processing system would be able to gain knowledge on its own, by reading the existing matter available on the Internet. A few applications of natural language processing include information retrieval and machine translation.

Natural languages are very expressive and almost everything that can be expressed symbolically can also be expressed in natural languages. It is the most expressive knowledge representation formalism that humans use. However, there are various limitations also to natural languages. Its reasoning is very complex and it is hard to maneuver. It is also very ambiguous and most people do not understand the concepts of syntax and semantics. Also, there is very little uniformity in the sentence structures.

The association between natural language and knowledge representation focuses on the two following points:

- Study of hybrid logic (propositional, first-order and higher-order) and implementation of efficient proof methods.
- Investigation of other logic that are of relevance to natural languages and knowledge representation like memory logic, dedicated planning method and the Discourse Representation Theory (DRT).

Properties for Knowledge Representation Systems

Knowledge representation systems possess the following properties:

- Representational adequacy
- Inferential adequacy
- Inferential efficiency
- Acquisitional efficiency
- Well-defined syntax and semantics
- Naturalness
- Frame problem

Representational Adequacy

A knowledge representation scheme must be able to actually represent the knowledge appropriate to our problem.

Inferential Adequacy

A knowledge representation scheme must allow us to make new inference from old knowledge. It means the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original. It must make inferences that are as follows:

- Sound – The new knowledge actually does follow from the old knowledge.
- Complete – It should make all the right inferences.

Here soundness is usually easy, but completeness is very hard.

Example: Given Knowledge

Tom is a man.

All men are mortal.

The inference –

James is mortal

Is not sound, whereas

Tom is mortal

is sound.

Inferential Efficiency

A knowledge representation scheme should be tractable, i.e., make inferences in reasonable time. Unfortunately, any knowledge representation scheme with interesting expressive power is not going to be efficient; often, the more general knowledge representation schemes are less efficient. We have to use knowledge representation schemes tailored to problem domain, i.e., less general but more efficient. The ability to direct the inferential mechanisms into the most productive directions by storing guides.

Acquisitional Efficiency

Acquire new knowledge using automatic methods wherever possible rather than on human intervention. Till today no single system optimizes all the properties. Now we will discuss some of the representation schemes.

Well-defined syntax and semantics

It should be possible to tell:

- Whether any construction is 'grammatically correct'.
- How to read any particular construction, i.e., no ambiguity.

Thus a knowledge representation scheme should have well-defined syntax. It should be possible to precisely determine for any given construction, exactly what its meaning is. Thus a knowledge representation scheme should have well-defined semantics. Here syntax is easy, but semantics is hard for a knowledge representation scheme.

Naturalness

A knowledge representation scheme should closely correspond to our way of thinking, reading and writing. A knowledge representation scheme should allow a knowledge engineer to read and check the knowledge base. Simply we can define knowledge representation as knowledge representation is the problem of representing what the computer knows.

Frame Problem

A frame problem is a problem of representing the facts that change as well as those that do not change. For example, consider a table with a plant on it under a window. Suppose we move it to the centre of the room. Here we must infer that plant is now in the center but the window is not. Frame axioms are used to describe all the things that do not change when an operator is applied in one state to go to another state say n + 1.

Simple Relational Knowledge

Simple relational knowledge involves the following. It is shown in Figure 2.21.

- Simple way to store facts.
- Each fact about a set of objects is set out systematically in columns.
- It presents little opportunity for inference.
- Knowledge basis for inference engines.

Musician	Style	Instrument	Age
Ravi	Jazz	Trumpet	36
John	Avant Garde	Saxophone	35
Robert	Rock	Guitar	43
Krishna	Jazz	Guitar	47

Fig. 2.21 Simple Relational Knowledge

We ask things like:

- Who is dead?
- Who plays jazz/trumpet; *etc.*?

This sort of representation is popular in database systems.

Inheritable Knowledge

Inheritable knowledge is shown in Figure 2.22.

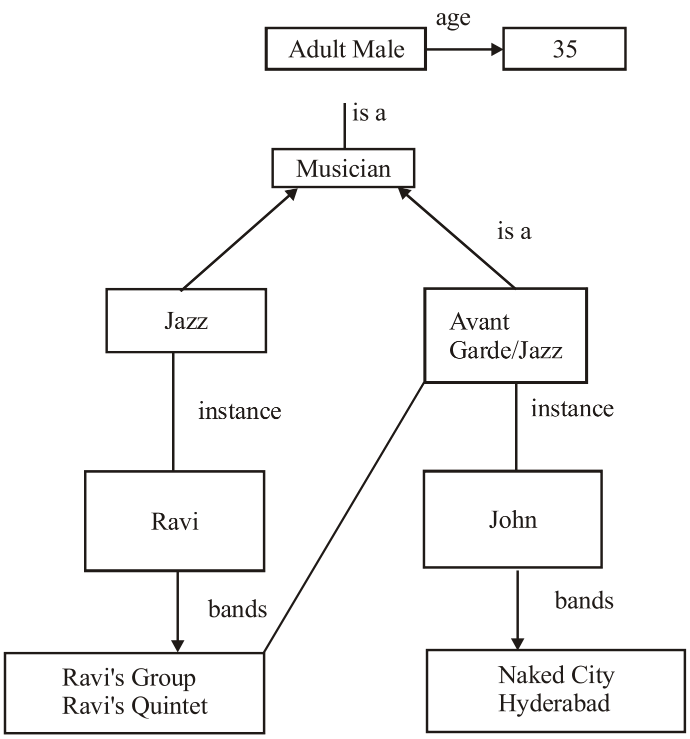


Fig. 2.22 *Property Inheritance Hierarchy*

Relational knowledge is made up of objects consisting of

- Attributes.
- Corresponding associated values.

We can extend the base by allowing inference mechanisms:

- Property inheritance
  - Elements inherit values from being members of a class.
  - Data must be organized into a hierarchy of classes.
- Boxed nodes – objects and values of attributes of objects.
- Values can be objects with attributes, and so on.
- Arrows – point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

The algorithm retrieves a value for an attribute in an instance object:

Algorithm: Property Inheritance

1. Find the object in the knowledge base.
2. If there is a value for the attribute report it.
3. Otherwise look for a value of instance if none fail.
4. Otherwise go to that node and find a value for the attribute and then report it.
5. Otherwise search through using *isa* until a value is found for the attribute.

Inferential Knowledge

Represent knowledge as ***formal logic***:

All dogs have tails: $dog(x) \quad has\_a\_tail(x)$

Advantages:

- A set of strict rules.
  - Can be used to derive more facts.
  - Truths of new statements can be verified.
  - Guaranteed correctness.
- Many inference procedures available to implement standard rules of logic.
- Popular in AI systems. Ex. Automated theorem proving.

Procedural Knowledge

The basic idea of procedural knowledge is the knowledge encoded in some procedures. Small programs know how to do specific things and how to proceed. For example, a parser in a natural language understander has the knowledge that a *noun phrase* may contain articles, adjectives and nouns. It is represented by calls to routines that know how to process articles, adjectives and nouns.

Advantages:

- Heuristic or domain specific knowledge can be represented.
- Extended logical inferences, such as default reasoning facilitated.
- Side effects of actions may be modelled. Some rules may become false in time. Keeping track of this in large systems may be tricky.

Disadvantages:

- Completeness - not all cases may be represented.
- Consistency - not all deductions may be correct.
- Modularity is sacrificed.
- Cumbersome control information.

2.9.3 Issues in Knowledge Representation

The issues in knowledge representation are discussed as follows:

Important Attributes and Relationships

There are two main attributes of knowledge representation: *Instance* and *Isa.* They are important because each supports property inheritance.

What about the relationship between the attributes of an object, such as inverses, existence, techniques for reasoning about values and single valued attributes. We consider an example of an inverse in

band(*John, Naked City*)

This can be treated as John plays in the band *Naked City* or John's band in *Naked City*.

Another representation is *band = Naked City*

Band-members = John, Robert,

The relationship between the attributes of an object is independent of specific knowledge they encode, and may hold properties like: inverses, existence in an isa hierarchy, techniques for reasoning about values and single valued attributes.

Inverses

This is about consistency check, while a value is added to one attribute. The entities are related to each other in many different ways. The figure shows attributes (*isa, instance and team*), each with a directed arrow, originating at the object being described and terminating either at the object or its value.

Existence in an Isa Hierarchy

This is about generalization-specialization, like classes of objects and specialized subsets of those classes, their attributes and specialization of attributes.

Ex.: The *height* is a specialization of general attribute; physical-size is a specialization of physical-attribute. These generalization-specialization relationships are important for attributes because they support inheritance.

Techniques for Reasoning About Values

This is about *reasoning values of attributes* not given explicitly. Several kinds of information are used in reasoning. Like *height* must be in a unit of length, *age* of person cannot be greater than the age of person's parents. The values are often specified when a knowledge base is created.

Single-Value Attributes

This is about a specific attribute that is guaranteed to take a unique value. For example, a guitar player can be only a single and be a member of only one team. KR systems take different approaches to provide support for single-value attributes.

Granularity

At what level the knowledge represents and what are the primitives? Choosing the granularity of representation primitives are fundamental concepts, such as holding, seeing, playing. English is a very rich language with over half a million words is clear and we will find difficulty in deciding upon which words to choose as our primitives in a series of situations.

If *Syam feeds a dog* then it could be: *feeds*(*Syam, dog*)

If *Syam gives the dog a bone* will be: *gives* (*Syam, dog, bone*)

Are these the same?

In any sense does giving an object food constitute feeding?

If *give*(*x*, *food*) *feed*(*x*) then we are making progress. But we need to add certain inferential rules.

In the famous program on relationships

Ravi is Sai's cousin.

How do we represent this?

Ravi = daughter (*brother or sister* (*father or mother* (*Sai*)))

Suppose it is *Sree* then we do not know whether *Sree* is a male or female and then *son* applies as well.

Clearly the separate levels of understanding require different levels of primitives and these need many rules to link together apparently similar primitives.

Obviously there is a potential storage problem and the underlying question must be what level of comprehension is needed.

Representing Set of Objects

There are certain properties of objects that are true as member of a set but not as individual; for example, consider the assertion made in the sentences:

'There are more *sheep* than *people* in India', and '*English* speakers can be found all over the world.'

To describe these facts, the only way is to attach assertion to the sets representing *people, sheep* and *English*.

The reason to represent sets of objects is if a property is true to all or most of the elements of a set, then it is more efficient to associate with the set rather than to associate it explicitly with every elements of the set. This is done

- In logical representation through the use of a universal quantifier, and
- In hierarchical structure where nodes represent sets and inheritance propagate a set level assertion down to the individual.

  For example: assertion *large* (*elephant*), remember to make clear distinction between
- Whether we are asserting some property of the set itself, means, *the set of elephants is large*, or
- Asserting some property that holds for individual elements of the set means *any thing that is an elephant is large*.

There are three ways in which sets may be represented:

(a) Name, for example the node Musician and the predicates Jazz and Avant Garde in logical representation (see Figure 2.22).

(b) Extensional definition is to list the numbers, and

(c) Intentional definition is to provide a rule that returns true or false depending on whether the object is in the set or not.

Finding Right Structure

For accessing the right structure we can describe a particular situation. This requires selecting an initial structure and then revising the choice. While doing so, it is necessary to solve the following problems:

- How to perform an initial selection of the most appropriate structure?
- How to fill in appropriate details from the current situations?
- How to find a better structure if the one chosen initially turns out not to be appropriate?
- What to do if none of the available structures is appropriate?
- When to create and remember a new structure?

There is no good and general-purpose method for solving all these problems. Some knowledge representation techniques solve some of them.
Let's consider two problems:

1. How to select initial structure?
2. How to find a better structure?

Selecting an initial structure: There are three important approaches for the selection of an initial structure.

(a) Index the structures directly by specific English words. Let each verb have own its structure that describeds the meaning, when disadvantages are as follows:

(i) Many words may have several different meanings.

E.g.: 1. John flew to Delhi.

2. John flew the kite.

'flew' here had different meaning in two different contexts.

(ii) It is useful only when there is an English description of the problem.

(b) Consider a major concept as a pointer to all of the structures, when disadvantages are given as follows:

Example:

1. The concept, Steak might point to two scripts, one for the restaurant and the other for the super market.
2. The concept bill might point to as restaurant script or a shopping script. We take the intersection of these sets; get the structure that involves all the content words.
   - (i) If the problem contains extraneous concepts then the intersection will result as empty.
   - (ii) It may require a great deal of computation to compute all the possible sets and then to intersect them.

(c) Locate the major clue and use that to select an initial structure, when disadvantages are given as follows:

(i) We cannot identify a major clue in some situations.

(ii) It is difficult to anticipate which clues are important and which are not.

Revising the Choice When Necessary: Once we find a structure and if it doesn't seem to be appropriate then we would opt for another choice. The different ways in which this can be done are:

1. Select the fragments of the current structure and match them against the other alternatives available. Choose the best one.
2. Make an excuse for the failure of the current structures and continue to use that. There are heuristics such as the fact that a structure is more appropriate if a desired feature is missing rather than having an inappropriate feature.

   Example: A person with one leg is more plausible than a person with a tail.
3. Refer to specific links between the structures in order to follow new directions to explore.
4. If the knowledge is represented in a 'isa' hierarchy then move upward until a structure is found and they should not conflict with evidence. Use this structure if it provides the required knowledge or create a new structure below it.

Knowledge Representation Using Semantic Network

A semantic network is a tool used in knowledge representation that consists of a structure of semantic terms. The aim is to permit a definition of those words through their relationships.

The semantic web is aimed at converting the Internet into a universal semantic network by using markup languages and advanced metadata and permitting it to become machine readable to enhance the knowledge management of computers. Information is extremely important in the success of an enterprise. Ranging from a one-man set-up to a billion-dollar global enterprise, the importance of business intelligence cannot be stated enough. However, mere possession of information is not sufficient to ensure business intelligence. It is more essential to convert information into valuable data and use it in strategizing.

Businesses collect large volumes of commercial data on a daily basis, in the form of feedback from customer to market research, sales and supply chain data. Since, it is not possible for a business to devote all resources to examine this data closely, it is important that some method of data analysis exists to highlight patterns and trends that may be useful in the achievement of objectives.

A semantic network symbolizes semantic relations between entities and concepts. These networks are used in knowledge representation and are a directed, or undirected, graph that has vertices, which represent concepts and edges.

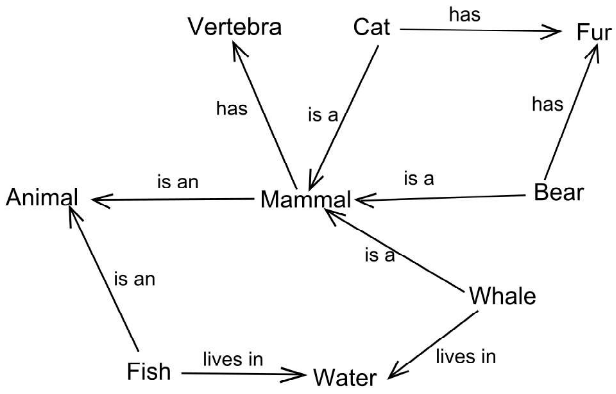


Fig. 2.23 *Example of a Semantic Network*

Conceptual Graph

A conceptual graph is a graphical notation for logic based on semantic networks of artificial intelligence. These graphs state meaning in a form that is precise, readable and tractable. Since these graphs are directly mapped to language, they act as an intermediate language for translating computer-oriented formalisms to and from natural languages. CGs are applied in various projects, such as, information retrieval, database design, expert systems and natural language processing.

Mind Map

A mind map is a diagram that represents words, ideas, tasks or other items linked to and arranged around a main keyword or idea. Mind maps generate, visualize, structure and classify ideas; act as a study aid; help to organize; solve problems; make decisions and formulate and write. The elements of a mind map are arranged as per the importance of concepts and classified into branches or areas, with the goal of representing semantic or other connections between portions of information. Mind maps facilitate a brainstorming approach to planning and organizational tasks, which encourages people to enumerate and connect concepts.

Semantic Networks in Relation to Knowledge Representation

Semantic networks enable enterprises to rapidly convert vast volumes of information into valuable data. Computer-based semantic networks use metadata (data describing data) to accurately interpret the meaning of information. Semantic networks permit computers to recognize the meaning of data held within a data warehouse (or other information storage medium), our computer searches for specific information become immeasurably more effective.

Semantic networks are also applied in content management such that the sum total of data held within an organization can be centrally held. This way all data assigned can be stored in a single, cost-effective and easy-to-manage database that can be used by any application. This approach also lends itself to data integration, which is a solution to problems faced by content management people posed by incompatible data infrastructures.

A semantic Web also changes the way information is stored on the Internet. Rather than search for information containing our targeted keywords we will be able to search the semantic meaning of the content, allowing search engines to return vastly more targeted information to us. Enterprises can also adopt a semantic model on a corporate Website. These Websites can recognize synonyms and are more flexible and intuitive that typical syntactic sites.

2.9.4 The Frame Probolem

The frame problem is the challenge of representing the effects of action in logic without having to represent explicitly a large number of intuitively obvious non-effects. Philosophers and AI researchers' frame problem is suggestive of a wider epistemological issue, namely whether it is possible or in principle, to limit the scope of the reasoning required to derive the consequences of an action.

The Frame Problem in Logic

Using mathematical logic, how is it possible to write formulae that describe the effects of actions without having to write a large number of accompanying formulae that describe the mundane, obvious non-effects of those actions? Let us look at an example. The difficulty can be illustrated without the full apparatus of formal logic, but it should be borne in mind that the devil is in the mathematical details. Suppose we write two formulae, one describing the effects of painting an object and the other describing the effects of moving an object.

1. *Colour*(x, *c*) holds after *paint*(*x, c*)
2. *Position*(x, *p*) holds after *move*(*x, p*)

   Now, suppose we have an initial situation in which *colour*(*A, Red*) and *position*(*A, House*). According to the machinery of deductive logic, what then holds after the action *paint*(*A, Blue*) followed by the action *move* (*A, Garden*)? Intuitively, we would expect *colour*(*A, Blue*) and *position* (*A, Garden*) to hold. Unfortunately, this is not the case. If written out more formally in classical predicate logic, using a suitable formalism for representing time and action such as the situation calculus, the two formulae above gives the conclusion that *position*(*A, Garden*) holds. This is because they do not rule out the possibility that the colour of *A* gets changed by the *Move* action.

   The most obvious way to augment such formalization so that the right common sense conclusions fall out is to add a number of formulae that explicitly describe the non-effects of each action. These formulae are called *frame axioms*. For example, we need a pair of frame axioms.

3. colour(x ,c) holds after Move(x, p) if colour(x ,c) held beforehand
4. Position(x, p) holds after Paint(x, c) if Position(x, p) held beforehand

In other words, painting an object will not affect its position, and moving an object will not affect its colour. With the addition of these two formulae, all the desired conclusions can be drawn. However, this is not at all a satisfactory solution. Since *most* actions do not affect *most* properties of a situation, in a domain comprising *M* actions and *N* properties we will, in general, have to write out almost *MN* frame axioms. Whether these formulae are destined to be stored explicitly in a computer's memory, or are merely part of the designer's specification, this is an unwelcome burden.

The challenge is to find a way to capture the non-effects of actions more successively in formal logic. What we need is some way of declaring the general rule-of-thumb that an action can be assumed not to change a given property of a situation *unless* there is evidence to the contrary. This default assumption is known as the *common sense law of inertia*. The technical frame problem can be viewed as the task of formalizing this law.

The main obstacle to doing this is the *monotonic* of classical logic. In classical logic, the set of conclusions that can be drawn from a set of formulae always *increases* with the addition of further formulae. This makes it impossible to express a rule that has an open-ended set of exceptions, and the common sense law of inertia is just such a rule.

Researchers in logic-based AI have put a lot of effort into developing a variety of *non-monotonic* reasoning formalisms, such as *circumscription*, and investigating their application to the frame problem. None of this has turned out to be at all straightforward. One of the most troublesome barriers to progress was highlighted in the so-called *Yale shooting problem*, a simple scenario that gives rise to counter-intuitive conclusions if naively represented with a non-monotonic formalism. To make matters worse, a full solution needs to work in the presence of concurrent actions, actions with non-deterministic effects, continuous change and actions with indirect ramifications. In spite of these subtleties, a number of solutions to the technical frame problem now exist that are adequate for logic-based AI research. Although improvements and extensions continue to be found, it is fair to say that the dust has settled, and that the frame problem, technically, is more or less solved.

Check Your Progress

15. What is the best understood scheme for knowledge representation and reasoning?
16. What are the core concepts of AI?
17. How will you define the five basic categories of artifacts of knowledge?
18. When was Prolog developed?
19. What do you understand by 'Acquisitional Efficiency' of knowledge representation system?
20. Define the term learning.
21. What is a frame problem?
22. What are the two main attributes of knowledge representation?
23. What is a semantic network?
24. Define conceptual graph.
25. What is a mind map?
26. What does the common sense law of inertia state?

2.10 ANSWERS TO 'CHECK YOUR PROGRESS'

1. The structures of a state space are trees and graphs where:
   - A tree is a hierarchical structure in a graphical form.
   - A graph is a non-hierarchical structure.
2. The sequence of states formed by possible moves is called a search tree. Each level of the tree is called a ply.
3. The function of production systems is to provide appropriate structures for performing and describing search processes.
4. Partially commutative, monotonic productions systems are useful for solving ignorable problems.
5. A heuristic function is a function that maps from problem state descriptions to measures of desirability, usually represented as number.
6. The best first search algorithm proceeds in the following manner:

   Step 1: Start with OPEN holding the initial state.

   Step 2: Repeat.

   Step 3: Pick the best node on OPEN.

   Step 4: Generate its successors.
7. Branch and Bound (BB) is a general algorithm that is used to find optimal solutions of different optimization problems, particularly in discrete and combinatorial optimization.
8. The branch and bound algorithm is used for the following problems:
   - Knapsack problem
   - Integer programming
   - Nonlinear programming
9. One method is transforming the graph back into OR graphs in an effective manner, in which every node depicts an entire set of goals that need to be achieved.
10. A difficult problem can be reduced to several uncomplicated problems. Moreover, when every single simple problem is solved, the difficult problem is solved as well.
11. Constraint satisfaction is a usual problem the goal of which is finding values for a set of variables which would satisfy a given set of constraints.
12. The parts that CSPs contain are as follows:
    - A set of variables X = $\{x_1, x_2, ..., x_n\}$
    - A finite set of values that each variable can take.
    - A set of constraints that denotes the values that variables can assume simultaneously

13. The following architectures enable the utilization of MEA:
    - Planning and Learning Architecture (Prodigy)
    - Problem Space Architecture (Soar)
    - Modular Integrated Architecture (ICARUS)
14. The problem solver is a search engine that looks over a problem space defined by the present domain and operators. As it performs its search, a problem-solving trace is made by it.
15. First-Order Predicate Calculus (FOPC) is the best-understood scheme for knowledge representation and reasoning.
16. The core concepts of research in Artificial Intelligence (AI) are knowledge representation and knowledge engineering.
17. The five basic types of artifacts of knowledge are as follow:
    (i) Facts
    (ii) Concepts
    (iii) Processes
    (iv) Procedures
    (v) Principles
18. Prolog was developed in 1972.
19. Acquisitional efficiency refers to the ability of the knowledge representation system to acquire knowledge with the help of automatic methods rather than depend on human intervention.
20. Acquiring knowledge is learning. It simply means adding new facts to a knowledge base.
21. A frame problem is a problem of representing the facts that change as well as those that do not change.
22. There are two main attributes of knowledge representation: Instance and Isa.
23. A semantic network is a tool used in knowledge representation that consists of a structure of semantic terms.
24. A conceptual graph is a graphical notation for logic based on semantic networks of artificial intelligence.
25. A mind map is a diagram that represents words, ideas, tasks or other items linked to and arranged around a main keyword or idea.
26. According to the common sense law of inertia an action can be assumed not to change a given property of a situation unless there is evidence to the contrary.

2.11 SUMMARY

- The word 'search' refer to the search for a solution in a problem space.
- Search is the systematic examination of states to find path from the start/root state to the goal state.

- A successor function needed for state change. The successor function moves one state to another state.
- A set of ordered pair of the form (x, y) is called a relation.
- The domain of a function is the set of all the first members of its ordered pairs, and the range of a function is the set of all second members of its ordered pairs.
- To solve the problem of playing a game, we require the rules of the game and targets for winning as well as representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state.
- The problem solved by using the production rules in combination with an appropriate control strategy, moving through the problem space until a path from an initial state to a goal state is found.
- Production systems provide appropriate structures for performing and describing search processes.
- Production systems provide us with good ways of describing the operations that can be performed in a search for a solution to a problem.
- A heuristic is a method that improves the efficiency of the search process. These are like tour guides. There are good to the level that they may neglect the points in general interesting directions; they are bad to the level that they may neglect points of interest to particular individuals.
- A salesman has to visit a list of cities and he must visit each city only once. There are different routes between the cities. The problem is to find the shortest route between the cities so that the salesman visits all the cities at one.
- Depth first is recommended since it is possible to find a solution without calculating all nodes and breadth first is recommended as it is not possible to trap it in dead ends. The best first search permits the switching between paths thus benefiting from both the approaches.
- Branch and Bound (BB) is a general algorithm that is used to find optimal solutions of different optimization problems, particularly in discrete and combinatorial optimization. It contains a systematic detail of each candidate solution, in which big subsets of candidates giving no results are rejected in groups, by making use of the higher and lower approximated limits of the quantity that are undergoing optimization.
- Constraint satisfaction is a usual problem the goal of which is finding values for a set of variables which would satisfy a given set of constraints.
- One of the core concepts of research in AI is knowledge representation. While declarative knowledge is represented as a static collection of facts with a set of procedures for manipulating the facts, procedural knowledge is described by an executable code that performs some action.
- Declarative knowledge is knowledge of facts.
- Procedural knowledge involves knowledge of formal language, symbolic representations and knowledge of rules, algorithms, and procedures.

- There are four approaches to the goals of AI; computer systems that act like humans, programs that simulate the human mind, knowledge representation and mechanistic reasoning and intelligent or rational agent design.
- Data is viewed as a collection of disconnected facts.
- Information emerges when relationships among facts are established and understood.
- Knowledge emerges when relationships among patterns are identified and understood.
- Wisdom is the understanding of the principles of relationships that describe patterns.
- The Knowledge Model defines that as the level of 'connectedness' and 'understanding' increases, our progress moves from data through information and knowledge to wisdom.
- Knowledge Representation (KR) is basically a replacement for an actual thing. It enables an entity to decide the end result by thinking rather than acting.
- Rules, frames, semantic networks and tagging are techniques of representation that have come up from human information processing theories.
- Processing of natural languages provides the machines the ability to read and understand the languages spoken by humans.
- There are two main important attributes in Knowledge Representation: Instance and Isa.
- A semantic network is a tool used in knowledge representation that consists of a structure of semantic terms.
- The frame problem is the challenge of representing the effects of action in logic without having to represent explicitly a large number of intuitively obvious non-effects.

2.12 KEY TERMS

- **State:** It represents the position of the solution at a mentioned step of the problem-solving process.
- **Depth first search:** It is a search strategy that extends the current path as far as possible before back-tracking to the last choice point and trying the next alternative path.
- **Constraint satisfaction problems:** These are mathematical problems defined as a set of objects whose state must satisfy many constraints or limitations.
- **Knowledge:** It is a progression that starts with data that is of limited utility and by organizing or analysing the data, we understand the meaning of

data, and this becomes information. The interpretation or evaluation of information yield knowledge.

- **Information:** It emerges when relationships among facts are established and understood. Providing answers to 'who', 'what', 'where' and 'when' gives the relationships among facts.
- **Wisdom:** It is the understanding of the principles of relationships that describe patterns. Providing the answer for 'why' gives understanding of the interaction between patterns.
- **Frame problem:** It is a problem of representing the facts that change as well as those that do not change.

2.13 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is problem space?
2. Give the definition of production system.
3. Define heuristic search.
4. What do you understand by the travelling salesman problem?
5. List the characteristics of heuristic search.
6. State the best first search.
7. What is branch and bound?
8. Define problem reduction.
9. Write a short note on dynamic constraint satisfaction problems.
10. How will you define the mean end analysis?
11. What role is played by knowledge in AI programs?
12. Write a short note on 'artifacts of knowledge'.
13. Write a short note on 'knowledge progression in AI systems'.
14. What do you understand by inferential efficiency and naturalness of knowledge representation systems?

Long-Answer Questions

1. Explain how Big-O notation is used for measuring complexity of an algorithm.
2. Discuss briefly about the production systems. Give characteristics with the help of examples.
3. What do you mean by heuristic search? Discuss heuristic search techniques with the help of examples.
4. Describe the types of AI search techniques with the help of suitable example.
5. Describe the branch and bound algorithm technique with the help of suitable example.

6. Explain briefly about the problem reduction. Give appropriate examples.
7. What is constraint satisfaction? Discuss briefly about the types of CSPs.
8. Give the name of architectures that enable the utilization of MEA.
9. Discuss the most difficult problems associated with knowledge representation in AI.
10. What are the differences between procedural and declarative knowledge? Give appropriate examples.
11. What do you understand by knowledge representation? Describe the various methods used to represent knowledge in AI systems.
12. What are the advantages of using semantic networks in enterprises?

2.14 FURTHER READING

Russell, Stuart J. and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*, 3rd Edition. New Jersey: Prentice Hall.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco (California): Morgan Kaufmann Publishers, Inc.

Knight Kevin, Elaine Rich and B. Nair. *Artificial Intelligence* (*SIE*), 3rd Edition. New Delhi: Tata McGraw-Hill.

Sivanandam, S.N. and M. Paulraj. 2009. *Introduction to Artificial Neural Networks*. New Delhi: Vikas Publishing House Pvt. Ltd.

Rich, E. and K. Knight, *Artificial Intelligence*. New York: McGraw-Hill Book Company, 1991.

LiMin, Fu. 2003. *Neural Networks in Computer Intelligence*. New Delhi: Tata McGraw-Hill.

UNIT 3 PREDICATE LOGIC AND RULE BASED SYSTEM

NOTES

Structure

3.0 INTRODUCTION

In Artificial Intelligence (AI), predicate logic provides the logic for dealing with complex real-life scenarios. Prior to predicate logic, the popular way of knowledge representation was propositional logic, which is suitable in situations where results are either true or false, but not both. This limitation of propositional logic gives way to the inception of predicate logic, using which you can not only represent the existing facts but also derive new facts from the existing ones. However, the facts, which are represented using predicate logic need to be true, otherwise the results derived from the facts will be wrong. In classical logic, modus ponendo ponens (Latin term for the way that affirms by affirming) is abbreviated as MP or Modus Ponens. It is a valid and simple argument form and is also referred to as affirming the antecedent or the law of detachment. It is closely related to another valid form of argument, modus tollens.

Resolution is a procedure in which the statements are converted into a standard form. Here, proofs are attempted to validate using refutation. One inference procedure using resolution is known as refutation. Refutation is also known as proof by contradiction and an absurdum. In other words, the resolution attempts to prove a statement by showing that the negation of that statement produces a contradictable result to the known statement. Natural deduction methods perform deduction in a manner similar to reasoning used by humans, e.g., in proving mathematical theorems. Forward chaining and backward chaining are natural deduction methods. Backtracking is used in many AI applications to solve a number of schemes to improve its efficiency.

Rule-based system, also known as expert system or production system has immense importance in the building of knowledge system. In these systems, the domain expertise is encoded in the form of 'if–then' rules. This enables a modular portrayal of the knowledge, which facilitates its updating and maintenance, we learn about forward reasoning, backward reasoning, conflict resolution and the use of non-backtracking. Forward reasoning, which is also known as forward chaining, breaks a task down into manageable and understandable steps. In case of backward chaining, the teaching process begins at the end of the sequence and moves to the beginning. It is used when it seems easier to teach a student a task from the last step instead of the first.

Matching is required between the current state and the preconditions of rules for better searching. This searching involves choosing from the rules, which can be applied at some particular point that can lead to a solution. Search control knowledge is defined as knowledge regarding different paths that are most likely to lead quickly to a goal state.

Backtracking and non-backtracking methods and algorithms are used by AI researchers to develop various applications of AI, such as game playing, expert system, robotics, etc. AI uses genetic programming, which is a technique for getting programs to solve a task by implementing the random List Processing Programming (LISP) programs and selecting the fittest in millions of generations.

In this unit, you will learn about the overview of predicate logic, modus ponens, resolution, natural deduction, dependency directed backtracking, rule based systems, procedural vs declarative knowledge, forward or backward reasoning, matching and conflict resolution and use of non-backtrack.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn about the predicate logic
- Represent simple facts in predicate logic
- Define instance and is-a relationship
- Discuss about the modus ponens
- Elaborate on the resolution
- Define natural deduction
- Explain the dependency-directed backtracking
- Learn about the rule based systems
- Illustrate the procedural vs declarative knowledge
- Analyse the forward vs backward reasoning
- Understand the matching and conflict resolution
- Discuss the use of non-back track

3.2 OVERVIEW OF PREDICATE LOGIC

Predicate logic is one of the important knowledge representation languages, which is concerned with deriving the real-world facts. In other words, predicate logic provides a way to represent the existing facts and allows you to derive new facts. The facts derived using predicate logic need to be true; otherwise, the conclusions drawn from the facts will go wrong.

Need for Predicate Logic

The need for predicate logic has been identified due to various limitations available in the propositional logic. Propositional logic provides a simple way of representing the real-world facts using Well-Formed Formulas (wff's). Figure 3.1 shows some examples of propositional logic.

Ram is running.
RUNNING

Ram is standing.
STANDING

Ram is sitting.
SITTING

If Ram is running, then Ram is not sitting.
RUNNING $\rightarrow \neg$ SITTING

Fig. 3.1 *Examples of Propositional Logic*

From the preceding example, it can be easily concluded that Ram is not sitting from the fact that Ram is running. Thus, it is seen that propositional logic is suitable in situations in which the results are either true or false, but not both. However, to the contrary of ease in use, propositional logic is not so powerful that it can represent all types of facts and assertions used in computer science and mathematics. Sometimes, it is also not ab1e to express certain types of relationships. To demonstrate the limitations of propositional logic, let us consider the following examples:

Let us consider the assertion, Tommy is a dog. In propositional logic, it can be represented as follows:

```
TOMMYDOG
```

A similar sentence of this type can be, Maxi is a Dog, which can be represented in propositional logic as follows:

```
MAXIDOG
```

However, from the above two representation of propositional logic, you cannot differentiate between TOMMY and MAXI. You can represent them in a better way as:

```
DOG(TOMMY)
DOG(MAXI)
```

Another limitation of propositional logic comes while representing the relationships in sentences. For example, the assertion—All dogs have tails—can be represented in propositional logic as follows:

```
TAILEDDOGS
```

However, from this representation, you cannot capture information about the relationship that whether it is talking about two dogs or dogs in general.

Consider the assertion, x is greater than 10, where x is a variable. However, from this assertion, you cannot ensure whether it is true or false, unless you know the value of x. Therefore, it is not a proposition and proposition logic cannot deal with such types of sentences.

Propositional logic also cannot capture the patterns involved in the logical equivalences in the following sentences:

1. Not all dogs are male, which is equivalent to, some dogs are not male.
2. Not all integers are odd, which is equivalent to, some integers are not odd.
3. Not all bikes are expensive, which is equivalent to, some bikes are not expensive.

In the preceding sentences, each of the propositions is considered independent of the other in propositional logic. For example, if P represents that not all dogs are male and Q represents that some dogs are not male, then there is no mechanism in propositional logic to represent the fact that P is equivalent to Q.

Thus, from the various examples representing the limitations in predicate logic, it can be inferred that the main limitations in propositional logic are the need of variables and quantification.

Basic Concepts of Predicate Logic

To overcome the limitations in propositional logic, you need to understand the various basic concepts of predicate logic. These include:

- Predicate
- Terms
- Quantifiers
- Free and bound variables

Predicate

A predicate can be defined as a relation, which enables you to bind two atoms together. For example, you can represent the assertion; Jimmy likes pastries, in predicate logic as follows:

```
LIKES(Jimmy, pastries)
```

Here, the predicate is LIKES, which binds two atoms, Jimmy and pastries. The two atoms represent the arguments for the predicate LIKES. In a generalized form, this predicate can be represented as follows:

```
LIKES(x, y)
```

Where, x and y are variables representing x likes y. Moreover, you can also use function as an argument of a predicate. For example, Jimmy's mother is Jack's mother, can be represented as follows:

```
MOTHER(mother(Jimmy), Jack)
```

Here, MOTHER is a predicate and mother (Jimmy) is a function, which indicates Jimmy's mother.

Terms

The terms are the arguments used in a predicate. For example, you can represent Jimmy is Jack's sister, in predicate logic as follows:

```
SISTER(Jimmy, Jack)
```

Where, Jimmy and Jack are terms for the predicate SISTER. You can also use a function as a term. For example, in predicate logic, Jack's sister is Sam's sister, can be represented asfollows:

```
SISTER(sister(Jack), Sam)
```

Here, sister (Jack) is a function that indicates Jack's sister. However, sister (Jack) is an argument of the predicate, SISTER and hence it is also a term. You can define the terms using the following rules:

- A constant is a term.
- A variable is a term.
- If f is a function and x1, x2 and x3 are terms, then f(x1, x2. x3) is also a term.

Quantifiers

A quantifier is defined as a symbol, which allows you to quantify a variable in a logical expression. In other words, a quantifier allows you to declare or identify the range or scope of the variables used in the logical expression. You can use several quantifiers in a logical expression including some, much, many, few, little, a lot, etc. However, basically in AI application, two primary quantifiers are used with predicate logic. These are:

- **Universal quantifier**: The universal quantifier in predicate logic allows you to formalize the concept that something is true for everything or every relevant thing. The symbol used to represent the universal quantifier is $\forall$. For example, if x is a variable, then you can read $\forall$x as any one of the followings:
    - o For all x
    - o For each x
    - o For every x
- **Existential quantifier**: The existential quantifier in predicate logic allows you to formalize the concept that something is true for something or for at least one relevant thing. The symbol used to represent the universal quantifier is $\exists$. For example, if x is a variable, then you can read $\exists$x as any one of the following:
    - o There exists a b
    - o For some b
    - o For at least one b

Table 3.1 shows the use of the universal and existential quantifiers in predicate logic.

Table 3.1 *Universal and Existential Quantifiers in Predicate Logic*

Expressions in Predicate Logic	Meaning
1. $\forall x\ [P(x)]$	For all x, P is true.
2. $\forall x\ [\neg P(x)]$	For all x, P is false
3. $\neg(\forall x\ [\neg P(x)])$	For some x, P is true
4. $\exists x\ [P(x)]$	For some x, P is true
5. $\exists x\ [\neg P(x)]$	For some x, P is false
6. $\neg(\exists x\ [\neg P(x)])$	For all x, P is true

Free and bound variables

A variable used in a formula of predicate logic is known as a free variable, if and only if the variable occurs outside the scope of the quantifier. For example, let us consider the following formula:

$\forall x\ (P(y) \rightarrow Q(y))$

In this formula, the quantifier $\forall$ is applied over the entire formula.

$P(y) \rightarrow Q(y)$

Therefore, the scope of the quantifier is $P(y) \rightarrow Q(y)$. This states that for all the values of x, P(y) implies Q(y), i.e., any change in the quantifier has no effect on P(y) and Q(y). Hence, the variable y is said to be a free variable.

A variable in a formula of predicate logic is known as a bound variable, if and only if the variable occurs within the scope of the quantifier. For example, let us consider the following formula:

$\forall x\ (P(x) \rightarrow Q(x))$

Here, the scope of the quantifier is $P(x) \rightarrow Q(x)$. This states that for all the values of x, P(x) implies Q(x), i.e., any change in the quantifier can affect both P(x) and Q(x). Hence, the variable y is said to be a bound variable.

A variable can also be either free or bound within a formula, if it occurs free at least for once. For example, let us consider the following formula:

$\forall x\ \exists y\ (P(x, y, z)\ \&\ \forall x\ (Q(y, z))$

Here, the variable z is free in the first part of the formula, while it is bound in the second part of the formula.

3.2.1 Representing Simple Facts in Predicate Logic

Predicate logic enables you to use variables and quantifiers for representing the real-world facts as statements, which are written as wff's. It is also regarded as the way of knowledge representation, as it allows you to represent those facts, which cannot reasonably be represented using propositional logic. To understand simple facts in predicate logic, let us consider the following sentences:

1. Pratap was a man.
2. Pratap was a Rajpoot.

3. All Rajpoots were Indian.
4. Rana was a ruler.
5. All Indians were either loyal to Rana or hated him.
6. Everybody is loyal to somebody.
7. People try to kill the rulers to whom they are not loyal.
8. Pratap tried to kill Rana.

In predicate logic, you can represent the facts described by these sentences as a set of wff's, depicted as follows:

```
1. Pratap was a man.
   man(Pratap)
2. Pratap was a Rajpoot.
   Rajpoot(Pratap)
3. All Rajpoots were Indian.
   ∀x: Rajpoot(x) → Indian(x)
4. Rana was a ruler.
   Ruler(Rana)
5. All Indians were either loyal to Rana or hated him.
   ∀x: Indian(x) → loyal to(x, Rana) V hate(Rana)
6. Everybody is loyal to somebody.
   ∀x:∃y: loyalto(x, y)
7. People try to kill the rulers to whom they are not
    loyal.
   ∀x:∃y: person(x) Λ ruler(y) Λ trykill(x, y) → ¬loyalto
     (x, y)
8. Pratap tried to kill Rana.
   trykill(Pratap, Rana)
```

Here, the predicate representation of the first sentence fails to state the concept of tense, which is cleared from the English sentence. However, it is able to capture the critical fact of Pratap being a man. In predicate logic, another problem arises in the scope quantifiers while converting English sentences to logical statements. This can be seen in the sixth sentence, where the logical representation, $\forall x{:}\exists y{:}$ loyal (x, y) has multiple meanings. In short, it means there exists someone to whom everyone is loyal. At the same time, it may mean that there exists someone to whom one is loyal, signifying a different someone for everyone.

In the preceding predicate logic representations, you may encounter another difficulty while trying to answer some specific questions. Consider that you need to know whether or not Pratap was loyal to Rana. For this, you can start reasoning backward from the desired goal represented in the predicate logic to get the answer of the question. Figure 3.2 shows how backward reasoning can be performed.

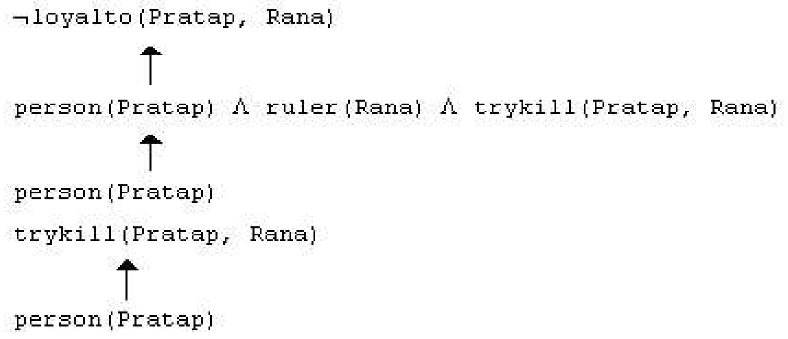


Fig. 3.2 *Backward Representations of Statements forAchieving the Goal*

You cannot put forward any statement to prove person (Pratap) as according to the given sentences, Man (Pratap). Therefore, you cannot prove that Pratap was not loyal to Rana. However, you can solve this problem by adding the following statement:

```
9. All men are people
∀x: man(x) → person(x)
```

Now, you can easily prove that Pratap was not loyal to Rana. This process of getting an answer by starting from the goal itself is known as backward chaining. The reverse of this process is called forward chaining.

To understand the concept of backward chaining more clearly, let us consider another example. Suppose, in this context, the following statements are available:

```
1. Jimmy likes all types of fruit.
   ∀x: fruit(x) → likes(Jimmy, x)
2. Mangoes are fruits.
   fruit(x)
3. Anything anyone eats is not killed by is a fruit.
   ∀x:∀y: eats(x, y) Λ ¬killedby(x, y) à fruit(y)
4. Jack eats an apple and is alive.
   Eats(Jack, apple) Λ ¬killedby(Jack, apple)
    (Here it is assumed that alive is same as not killed
by)
5. Sam eats everything Jack eats.
   ∀x: fruit(x) Λ eats (Jack, x) → eats(Sam, x)
```

Now, consider that you need to answer whether Jimmy likes apple. You can prove that Jimmy likes apple applying backward chaining, as shown in Figure 3.3.

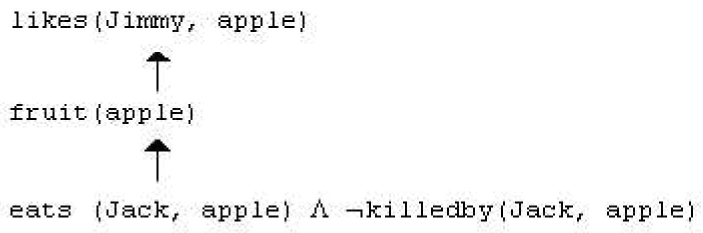


Fig. 3.3 *Example to Show the Use of Backward Chaining*

3.2.2 Representing Instance and is a Relationships

An instance is a binary predicate containing two arguments, where the first argument is an object of the class represented by the second argument. For example, consider the following predicate logic:

```
cat(Billy)
```

This statement represents a unary predicate containing a single argument and the meaning of this statement is that Billy is a cat. However, you can use an object of the class cat to represent this predicate as an instance, shown as follows:

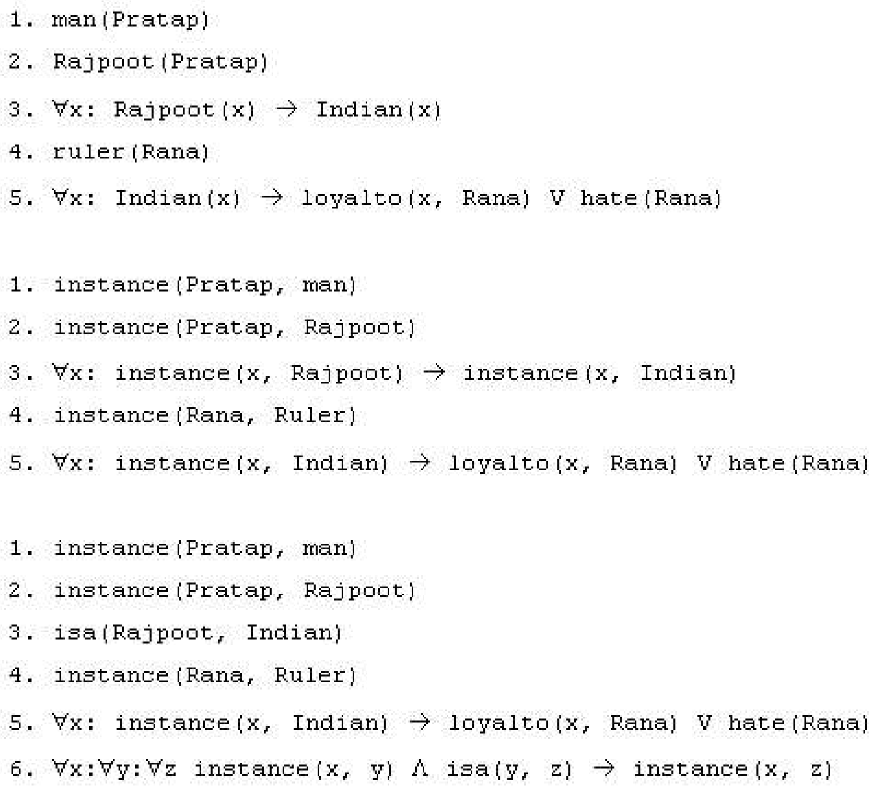
```
Instance(Billy, cat)
```

Here, Billy is the object of the class cat.

However, an instance does not directly represent an isa predicate, where the isa predicate is also a binary predicate used to simplify the logical representations. To understand the instance and isa predicates and their relationships, let us consider the first five statements of the first example representing simple facts in predicate logic.

- Pratap was a man.
- Pratap was a Rajpoot.
- All Rajpoots were Indians.
- Rana was a ruler.
- All Indians were either loyal to Rana or hated him.

Figure 3.4 shows the instance and isa predicates for the preceding statements and the relationship between the predicates.



```
1. man(Pratap)
2. Rajpoot(Pratap)
3. ∀x: Rajpoot(x) → Indian(x)
4. ruler(Rana)
5. ∀x: Indian(x) → loyalto(x, Rana) V hate(Rana)

1. instance(Pratap, man)
2. instance(Pratap, Rajpoot)
3. ∀x: instance(x, Rajpoot) → instance(x, Indian)
4. instance(Rana, Ruler)
5. ∀x: instance(x, Indian) → loyalto(x, Rana) V hate(Rana)

1. instance(Pratap, man)
2. instance(Pratap, Rajpoot)
3. isa(Rajpoot, Indian)
4. instance(Rana, Ruler)
5. ∀x: instance(x, Indian) → loyalto(x, Rana) V hate(Rana)
6. ∀x:∀y:∀z instance(x, y) Λ isa(y, z) → instance(x, z)
```

Fig. 3.4 *Relationship between Instance and is a Predicates*

NOTES

3.2.3 Modus Ponens

In classical logic, **modus ponendo ponens** (Latin term for *the way that affirms by affirming* is abbreviated as **MP** or **Modus Ponens.** It is a valid and simple argument form and is also referred to as **affirming the antecedent** or **the law of detachment**. It is closely related to another valid form of argument, *modus tollens*. *Modus ponens* is a common rule of inference and takes the following form:

If *P*, then *Q*.

P.

Therefore, *Q*.

Formal Notation: The *modus ponens* rule may be written in sequent notation as:

$$P \to Q, P \vdash Q$$

And in rule form:

$$\frac{P \to Q, P}{\therefore\ Q}$$

The argument form has two premises. The first premise is the 'if–then' or conditional claim that P implies Q. The second premise is that P, the antecedent of the conditional claim is true. These two premises can be logically concluded that Q which is the consequent of the conditional claim and is true. In Artificial Intelligence, *modus ponens* is also called forward chaining.

The following example is an argument that fits the form *modus ponens*:

If today is Monday, then I will go to work.

Today is Monday.

Therefore, I will go to work.

This is a valid argument but it has no bearing on whether any of the statements in the argument is true for *modus ponens* for a sound argument. The premises must be true for any true instances of the conclusion. An argument is valid but unsound if one or more premises are false. If an argument is valid and all the premises are true then the argument is termed sound argument. For example, one may go to work on Wednesday because the reasoning for going to work is unsound. The argument is only sound for Monday. A propositional argument that uses modus ponens is deductive.

The Curry-Howard correspondence between proofs and programs that are related to modus ponens function application: if *f* is a function of type $P \to Q$ and *x* is of type *P*, then *f x* is of type *Q*. The validity of *modus ponens* in classical two-valued logic can be clearly demonstrated by use of a truth table.

p	**q**	**p → q**
T	T	T
T	F	F
F	T	T
F	F	T

In instances of *modus ponens* it is assumed that premises p → q is true and p is true. Therefore, whenever p → q is true and p is true, q must also be true.

3.2.4 Resolution

NOTES

Resolution is a procedure in which the statements are converted into a standard form. Here, proofs are attempted to validate using refutation. One inference procedure using resolution is known as refutation. Refutation is also known as proof by contradiction and an absurdum. In other words, the resolution attempts to prove a statement by showing that the negation of that statement produces a contradictable result to the known statement.

Conversion to Clause Form

Suppose, we know that all Indians who know Amit hate ABC. We can represent this statement as follows:

$\forall x : [\mathrm{Indians}(x) \wedge \mathrm{Know}(x, \mathrm{Amit})] \rightarrow$

$[\mathrm{hate}(x, \mathrm{ABC}) \vee (\forall y : \exists z : \mathrm{hate}(y,z))$

This formula requires a complex matching process in a proof. It is very important to match the pieces with the formula. The process of matching the pieces with the formula would have been easier, if the formula were in a simpler form.

Basis of Resolution

The procedures in the resolutions are very easy, as the two clauses in the resolution procedures are known as the parent clauses. The parent clauses are compared, which gives rise to the new clauses. The new clause states ways in which the interaction between the parent clauses must take place. This can be explained with the help of the following example:

```
Summer V Winter
¬Summer V Hot
```

By recalling both the clauses we can state that they must be true.

From the example we can recall that at any point one of Summer and ¬Summer will be true. If the summer is true, then surely hot will be the truth of the second clause. With the help of these two clauses we can deduce the statement as:

```
Winter V Hot
```

Taking the two clauses each containing the same literal is the way the resolution operates. The literals in the resolution must appear in the positive and the negative forms in the respective clauses.

The contradiction in the resolutions can be found if the clauses are produced in the empty clause, such as:

```
Summer
¬Summer
```

These two clauses will give the result as an empty clause. If there is any contradiction in the clauses, then it can be found. If there is no contradiction, then the clauses in the procedures can never be found.

Resolution in Propositional Logic

To explain the working of the resolution, first the procedures of the resolutions for the propositional logic must be presented. To understand how the resolution works, consider the following example:

All Indians are Asians.

Aryabhatta is an Indian.

Therefore, Aryabhatta is an Asian.

Or, more generally:

∀X, P(X) implies Q(X).

P(a).

Therefore, Q(a).

Resolution in prepositional logic can also be explained with the help of an example. First, the axioms are converted in a clause form, shown as follows:

Given Axioms	**Converted to Clause Form**
A	A
(E Λ G)→ J	¬E V¬G V J
(O V Z)→ G	¬O V G
Z	¬Z V G
	T

After the axioms are converted in the clause form, then J is negated, producing ¬J which is in a clause form. Then the pair of the clauses is selected in order to resolve them together. All the pairs cannot be resolved except those which have the complimentary literals that will produce the empty clause.

Another way of viewing the resolution process is that all the clauses that are true are taken and the new classes are developed in place of the original clauses that are true.

Resolution in Predicate Logic

To prove things in predicate logic, you need two things. First, you need to determine what inference rules are valid and second, you need to know a good proof procedure that will allow proving the things with the inference rules in an efficient manner. You can explain the resolution in predicate logic with the help of the following example:

```
Man(Amit)
¬man(x1) V Amit(x1)
```

The literal man (Amit) can be unified with the literal man (x1) with the substitutions Amit/x1, determining that for x1=Amit, ¬man(Amit) is false. But the two literals cannot be cancelled out. Therefore, the resolutions in the predicates determine the importance of converting the variables into the clause form.

From the following example, you can understand how the resolution can be used to prove new things. You can use the resolutions to prove the things about Amit. First, the statements have to be converted into the clauses.

```
Loyal(Amit, ABC)
```

From this example, many more resolutions could have been generated. The actual goal of the preceding statement is to prove whether Amit hated ABC. In that case, the statement would have been:

```
¬hate(Amit, ABC)
```

NOTES

Question Answering

Question answering is basically a type of information retrieval in which a system called Question Answering (QA) system is used to retrieve the answers for the questions, which are written in the natural language. The question answering is the most complex natural language processing techniques as compared to other techniques. The QA system uses the text documents as their knowledge source. The QA system adds different natural language techniques to create a single processing technique and then uses the newly developed technique to search for answers to the questions written in the natural language. The QA system contains a question classifier module that is used to determine the types of questions and answers.

3.2.5 Natural Deduction

Natural deduction methods perform deduction in a manner similar to reasoning used by humans, e.g., in proving mathematical theorems. Forward chaining and backward chaining are natural deduction methods. These are similar to the algorithms described earlier for propositional logic, with extensions to handle variable bindings and unification.

Backward chaining by itself is not complete, since it only handles Horn clauses (clauses that have at most one positive literal). Not all clauses are Horn; for example, 'every person is male or female' becomes $\neg$ Person(x) $\vee$ Male(x) $\vee$ Female(x) which has two positive literals. Such clauses do not support backward chaining.

Splitting can be used with back chaining to make it complete. Splitting makes assumptions (e.g., 'Assume x is Male') and attempts to prove the theorem for each case.

3.2.6 Dependency

Conceptual dependency theory is used as a model of natural language understanding used in artificial intelligence systems. Schank developed the model for representing knowledge for natural language. Independent usage of words in the input, i.e., two sentences which are identical in meaning have a single representation. The system is also used to draw logical inferences.

The dependency model uses the following basic representational tokens:

- Real world objects, each with some attributes
- Real world actions, each with attributes
- Times
- Locations

A set of conceptual transitions then act on this representation as an ATRANS is used to represent a transfer, such as 'give' or 'take' while a PTRANS is used to act on locations, such as 'move' or 'go'. An MTRANS represents mental acts, such as 'tell', etc. For example, a sentence 'Mohan gave a pen to Sudhir' can be represented as the action of an ATRANS on two real world objects Mohan and Sudhir.

A dependency in the Unified Modeling Language (UML) always exists between two defined elements if a change in the definition results in a change to the other. In UML is indicated using a dashed line pointing from the dependent to the independent element.

If there are more than one dependent or independent participates in the dependency then the arrows with their tails on the dependent elements are connected to the tails of one or more arrows with their heads on the independent elements. A small dot is placed on the junction point along with a note on the dependency. Dependency is also a model-level relationship that describes the need to investigate the model definition of the dependent element for possible changes if the model definition of the independent element is changed.

A dependency is a semantic relationship that changes the independent modelling element which may affect the semantics of the dependent modeling element. It also identifies a set of model elements that needs other model elements for their specification or implementation. The arrow represents a dependency specifying the direction of a relationship and not the direction of a process.

3.2.7 Directed Backtracking

Backtracking is used in many AI applications to solve a number of schemes to improve its efficiency. Such schemes are termed dependency-directed backtracking, or sometimes intelligent backtracking and can be classified as follows:

Lookahead Schemes

These schemes are used to control that which variable to be instantiated next or what value to be selected from the consistent options.

- **Variable Ordering**: This approach helps to select a variable for making the rest of the problem easier for solving. Basically, this is done by selecting the variable involved in the most of the constraints.
- **Value Ordering**: A value is selected for maximizing the number of options available for future assignments.

Look-Back Schemes

In backtracking, look-back schemes are used to control the specific decisions of where and how to go back in case of dead-ends. Basically, there are two basic approaches:

- **Go Back to Source of Failure:** It changes only those past decisions that caused the error and leaves other past decisions unchanged.
- **Constraint Recording:** It record the 'reasons' for the dead-end so that they can be avoided in future search.

Dependency-directed backtracking is also used in truth-maintenance systems. It follows a variable that assigns some value and a justification for that value is recorded. A default value is then assigned to some other variable and justified. The system now checks whether the assignments have violate any constraints. If there is any then it is records as the two which are not simultaneously acceptable. This record is used for justifying the choice of some other variable and continues until a solution is found. Such systems never perform redundant backtracking.

Check Your Progress

1. How will you understand the simple facts in predicate logic?
2. Define modus pones.
3. How is the modus ponens rule written in sequent notation?
4. What is resolution?
5. State the natural deduction.
6. How will you define the dependency directed backtracking?
7. Who developed the model for representing knowledge for natural language?
8. How a dependency is indicated in Unified Modelling Language (UML)?
9. Why look-back schemes are used in backtracking?

3.3 RULE BASED SYSTEMS

The designing of rule-based systems is often repeated due to restrictions offered by some off-the-shelf libraries and systems and the lack of common software architecture. Thus, the architecture for rule-based systems is to be understood through design patterns, which aim to incorporate a design catalog, which designers can use to understand and build new rule-based systems; thus facilitating reuse in these systems. Besides, specific patterns are used in the design of intelligent tutoring system architecture. A *rule-based system* refers to a system of encoding an expert's wisdom in a relatively narrow area into an automated system. This function has a couple of advantages. The first advantage is that the human expert's wisdom, after this process, is available to a very huge range of people. The second advantage is that if one is able to record the expertise of an expert in a particular field, the individual's knowledge is not lost whenever he/she leaves the firm or retires.

Rule-based systems are not like object-oriented programs or standard procedural because there is no comprehensible order in which the code executes. Rather, the expert's knowledge or wisdom is tapped in a set of rules, wherein each rule encodes a small piece of knowledge. Here, each rule has a left-hand side and a right-hand side. The former side contains information of certain facts and objects that must hold true when the rule is applied on them, i.e., executed. Any rule, whose left-hand sides match in this way at a given time, are placed on an agenda. At random selection, one of the rules on the agenda is picked, its right-hand side is executed and thereafter it is taken off from the agenda. Then, using a special algorithm called the Rete algorithm, the agenda is updated and the next rule is chosen for execution. This process goes on until the agenda is emptied of rules.

The following is an example of a typical rule for a mortgage application:

```
IF
(number-of-30-day-delinquencies > 4)
AND (number-of-30-day-delinquencies < 8)
THEN
increase mortgage rate by 1 per cent.
```

Here, the aforementioned rule is just like an 'if-then-else' statement; however, unlike an 'if-then-else' statement, it is unique and does not execute any predetermined order related to other if-then-else statements.

A rule-based system can be considered as being similar to a multi-threaded system . In a multi-threaded system, one does not know which thread will execute next; similarly, one does not know which rule will execute next. However, all rule-based systems are often implemented like single-thread programs. This approach has some advantages, such as—opposed to a procedural approach, it emphasizes that if you have designed the system well, the expert's knowledge or wisdom can be kept very easily just by changing whichever rules need changes. In fact, many rule-based systems have a rules editor, which allows even the non-technical people to easily maintain rules.

A rules engine is used to implement the rules. It provides a fundamental framework for writing rules and running them in the aforementioned manner. Previously it was very hard to deal with rules engines as they used to be complex technologies, which were not compatible with the rest of the IT world. However, just a couple of years ago, great research has been done in making rules engines that are easily compatible with other technologies.

A rule-based system can be built by using a set of assertions, which are collectively called the 'working memory', and a set of rules that tells how to act on the assertion set. Rule-based systems extend support to the so-called 'expert systems', which are commonly used in many fields although they may seem simple systems consisting of a couple of 'if-then' statements. Rule-based systems are so simple that these models can be referred to solve any number of problems. As with any Artificial Intelligence (AI), a rule-based system has its advantages as well as disadvantages. These factors must always be referred to while choosing this technique for a given problem. Generally, you can apply rule-based systems only for problems in which any and all knowledge in the problem area can be represented in the form of if-then rules and in which this problem area is not huge. The system can become difficult to handle and may suffer a performance hit if there are too many rules.

In order to build a rule-based system for a specific problem, you must have the following:

- A set of facts that symbolizes the initial working memory. It could be something concerned with the beginning state of the system.

- A set of rules. It could involve nothing irrelevant but any and all actions, which are taken within the scope of a problem. This is because excessive number of rules in the system can affect its performance.
- A condition that certifies the discovery or not-finding of a solution. In this way you terminate some rule-based systems that otherwise find themselves in infinite loops.

Theory of Rule-Based Systems

The rule-based system uses a simple technique, as it begins with a rule-base, which contains all of the required knowledge encoded into If-Then (IF) rules, and a working memory, which may or may not, to begin with, contain any assertions, data or initially well-known information. The system analyses all the rule conditions (IF) and decides a subset, or the conflict set, of the rules whose conditions are matched based on the working memory. One of the rules of this conflict set is triggered (fired). Which one is triggered is based on a conflict resolution planning. When the rule is fired or triggered, any actions mentioned in its THEN clause are carried out. These actions can modify the rule-base itself, the working memory, or do anything else that the system programmer may decide to include. This cycle of firing rules and performing actions goes on till one of the two conditions are fulfilled — a rule is fired whose action specifies that the program should terminate or there are no more rules whose conditions are satisfied.

The conflict resolution strategy helps decide which rule is chosen to fire. The selection of strategy is decided by the problem or preference. In any case, it is significant as it controls which of the favourable rules are fired and in return, how the entire system behaves. Among various strategies, the major ones are as follows:

- **First Applicable:** If the rules fall in a certain order, firing the first applicable rule gives you control over the order in which rules fire. This is the straight forward strategy and can pose a big problem, i.e., an infinite loop on the same rule may crop up. If the working memory does not change, as does the rule-base, the conditions of the first rule also remain unchanged, and thus, it will fire again and again. To overcome this problem, a fired rule is suspended and prevented from re-firing till the data that satisfied the rule's conditions in working memory has changed.
- **Random:** This strategy has its unique advantages, although it never provides control over the first-applicable strategy. For example, its unpredictability is advantageous in some specific circumstances (i.e., games). This strategy just chooses a single random rule to fire from the conflict set. Another possibility of this strategy is a fuzzy rule-based system in which every rule has a probability such that some rules are more likely to fire than others.
- **Specific:** It is based on the number of conditions offered by rules. From the conflict set, the rule having maximum conditions is chosen. This is done on the assumption that it has the most relevance to the existing data if it has the most conditions.
- **Least Recently Used:** Every rule is followed by a time or step stamp that marks the last time it was used. This increases the number of individual

rules, which are fired at least once. If all rules are needed for providing a solution to a certain problem, this is an excellent strategy.

- **'Best' Rule:** For its working, every rule is given a 'weight,' which decides how much it should be weighted over the rest of the alternatives. The rule having the most preferable outcomes is selected based on this weight.

Unlike representing knowledge in a relatively static, declarative manner such as a bunch of things that are true, rule-based systems represent knowledge in the form of a group or set of rules that tell you what you could conclude in different situations or what you should do. A rule-based system consists of a bunch of IF rules, a set of facts, and some interpreters that control the application of the rules. Thus rule-based reasoning is a very specific type of reasoning that uses 'if-then-else' rule statements. These rules are the patterns and an inference engine searches for these set of patterns or patterns in the rules that match the patterns given in the data. Here, the 'if' means 'when the condition is true,' the 'then' means 'take action A' and the 'else' means 'when the condition is not true, take action B'.

Here is an example with the rule PROBABLE CAUSE:

IF robbery is TRUE

AND

suspect witness identification is TRUE

AND

suspect physical evidence is TRUE

AND

suspect lacks alibi is TRUE

THEN

probable cause is TRUE

ELSE

round up usual suspects.

Rules can be forward-chaining, also known as forward reasoning or data-driven reasoning, as they begin with data or facts and search for rules, which can be applied to the facts till a goal is reached. Rules can also be backward chaining, also known as backward reasoning or goal-driven reasoning, as they begin with a goal and search for rules which can be applied to that goal till a conclusion is drawn.

Origin of Rule-Based Systems

The phenomenon of AI has witnessed immense development and has been applied to various types of problems. Despite the wide range of goals and perspectives shown by diverse systems, there are many intermittent themes. The following is an analysis of those themes, and a conceptual framework through which most of the apparently disparate efforts can be seen, both in context of one another and other methodologies. Thus, the term *production system* is used in a broad sense. It is also meant to know how many systems that have used this term can fall into the framework. In order to provide a view of production systems characteristics in a broader context, they are compared to other methodologies. This is done not only with basic reference to procedurally based techniques, but also with reference to recent discoveries and researches in programming and the organization of knowledge and databases.

NOTES

Pure Production Systems

A pure production system generally comprises three fundamental components: a set of rules, a database and an interpreter for the rules. In the most generalized design, a rule is a sequenced pair of symbol strings with a Left-Hand Side (LHS) and a Right-Hand Side (RHS). The set of rules has a pre-decided, total ordering and the database is just an assemblage of symbols. In this simple design, the interpreter operates by scanning the LHS of each rule till one is found that can be correctly matched against the database. Now, the symbols matched in the database are exchanged with those found in the RHS of the rule, and further scanning either continues with the next rule or begins afresh. A rule can also be seen as a simple conditional statement and as the invocation of rules as an extension of actions chained by modus ponens.

Rules

Generally, one side of a rule is evaluated with regard to the database, and if this is correct (i.e., evaluates to TRUE in some sense) the action sought by the other side is carried out. Note that evaluate typically means 'an operation involving only matching and detection', or a passive operation of 'perception', while the action means one or more than one conceptually primitive operations.

Note that there is no specification of which side is to be matched, as either is possible. For example, the following is a grammar written in production rule form.

```
S~ABA
A-,A1
A~I
B~B0
B~0
```

When you match the LHS on a database that consists of the start symbol 'S' it provides a generator for strings in the language. On the other hand, matching on the RHS of the same set of rules provides a recognizer for the language. You can also change the process slightly to get a top-down recognizer by interpreting the elements in the LHS as goals to be achieved by the correct matching of elements in the RHS. In the aforementioned case the rules 'unwind.' Thus you can use the same set of rules in several manners. Note, however, that while doing so, you get many different systems having characteristically different control structures and behaviour. The organization and assessment of the set of rules is also a vital issue. The basic scheme is the fixed, i.e., total ordering; however, elaborations quickly grow more intricate. The term conflict resolution is used to refer to the process of selecting a rule.

Database

In the most basic production system the database is just a collection of symbols referring to the state of the world; however, the correct interpretation of these symbols wholly depends generally on the nature of the application. The database is interpreted as modelling the composition of a few memory mechanisms, i.e., Short-Term Memory (STM), wherein each symbol represents some 'chunk' of knowledge for those systems intended to explore symbol-processing aspects of human cognition.

Interpreter

The interpreter is the fountain of the whole variation that exists among different systems; however, it may be seen, in the fundamental terms, as a select-execute loop in which one rule, which is applicable to the current state of the database, is chosen and then executed. Its action brings modifications in the database and the select phase restarts. Although selection is at times a process of choosing the first rule that matches the current database, it becomes crystal clear why this cycle is often termed as a recognize-act, or situation-action, loop. This alternation between selection and execution is an essential characteristic of production system architecture, which is totally responsible for one of its most fundamental elements. By selecting each new rule for execution, based on the total contents of the database, you effectively carry out a complete re-evaluation of the control state of the system at every cycle. This is very different from the already set approaches in which control flow is generally dependent on just a small fraction of the total number of state variables, and is typically the decision of the process currently executing. Production systems are thus sensitive to any change that takes place in the entire environment, and highly responsive to such changes within the scope of a single execution cycle. Of course, the price of such responsiveness is the calculation time required for the re-evaluation.

Key Features of Rule-Based Systems

The following are the key features of the rule-based systems:

- Practical human experience and expertise can often be captured in the form of if/then rules.
- Instead of standard programming control strategies, rule-based systems are more flexible since those rules, which are appropriate in a certain situation, are dynamically chosen and combined.
- A rule-based system explains its results by recognizing the rules which resulted in a specific solution and describing the conditions which lead to that particular rule to be used.

Rule-Based System Architecture

The following is the rule-based system architecture:

- A collection of facts
- A collection of rules
- An inference engine

You might look forward to:

- See which new facts can be *derived at*, and
- Ask whether a fact is implicit in the knowledge base and already known facts (Refer Figure 3.5).

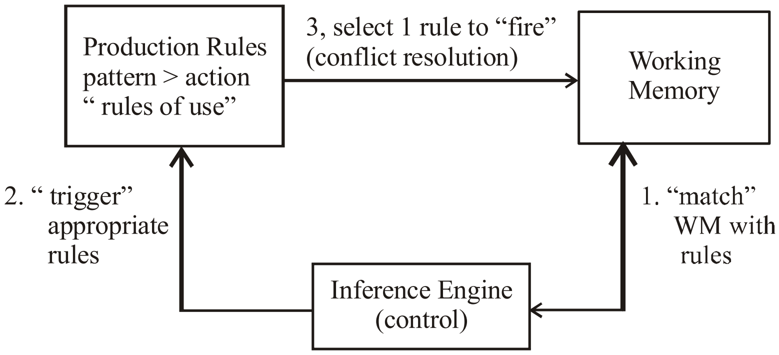


Fig. 3.5 *Rule-Based System*

- The inference engine addresses the following two problems:
  - o It must know the way to decide from where to begin. Rules and facts are found in a static knowledge base. Therefore, there has to be a way for the reasoning process to begin.
  - o The inference engine must be capable of resolving conflicts, which take place when alternative links of reasoning emerge. The system may reach a point where there may be more than a few rules ready to fire. At that point, the inference engine must select which rule to examine next.

Example 3.1

```
R1: IF hot AND smoky THEN fire
R2: IF alarm_beeps THEN smoky
R3: If fire THEN switch_on_sprinklers
F1: alarm_beeps [Given]
F2: hot [Given]
```

Example 3.2

```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers
F1: alarm_beeps [Given]
F2: hot [Given]
```

Example 3.3

```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers
F1: alarm_beeps [Given]
F2: hot [Given]
F3: smoky [from F1 by R2]
```

```
F4: fire [from F2, F4 by R1]
F5: switch_on_sprinklers [from F4 by R3]
```

Chaining and Rule-Based Systems

Chaining refers to a technique of teaching, which consists of breaking down a task into small steps and thereafter teaching each step within the sequence by itself. This process is very helpful when students are required to learn a routine task, which is repetitive. For example, the student may be required to comprehend all steps in the process of putting on a coat, using the bathroom or accomplishing a work task.

Chaining techniques are of two types: 'forward chaining' and 'backward chaining.' The forward chaining technique requires a student to handle the first part of the task until the end of the task. The backward chaining technique requires a student to handle the last part of the task until its beginning. The decision to use either 'forward chaining' or 'backward chaining' depends upon the task and the student. A teacher decides which chaining procedure is the best method of teaching the task through full analysis of the task and the students' ability level.

The chaining technique is a part of a bigger concept of behaviour intervention named as applied behavioural analysis. The task is indoctrinated to the student through the use of the behavioural technique of chaining and thereafter reinforced for completion. For the appropriate use of the chaining technique, the task, which the student is unable to complete, must first be broken down into small steps. This process of breaking those steps down is called 'task analysis.' Forward chaining systems are primarily data driven; however, backward chaining systems are goal driven.

3.3.1 Procedural vs Declarative Knowledge

A knowledge-based system represents, acquires and applies knowledge for a specific objective. There is a difference between the knowledge-based system and conventional system or data-oriented information processing system. The conventional system is a computer program that is described by the algorithmic processing of data. In the conventional system, knowledge deals with the execution of step-by-step instructions that is known as procedure. On the other hand, in the knowledge-based system, instructions are declared and knowledge is applied under a certain inference strategy. The main feature of a knowledge-based system is declarative representation rather than procedural representation.

Procedural Knowledge

In this approach of knowledge representation, knowledge is stored in the form of procedures. The procedures contain the logic in the form of a code, which specifies when and how the actions are to be performed. LISP is one of the languages used for writing procedures.

In procedural knowledge, the task is performed in a sequence such as reading the data, calculating the data and displaying the results. Procedural knowledge consists of a list of instructions that are organized into groups called functions. You can perform tasks using multiple functions. Figure 3.6 shows the structure of procedural knowledge.

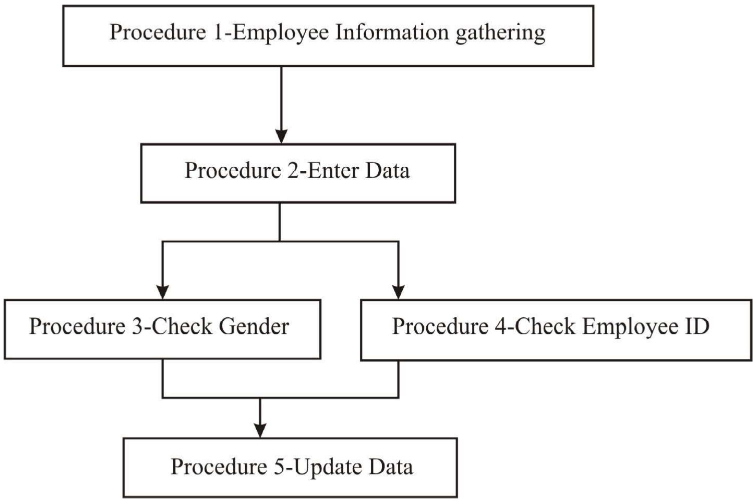

Procedure 1-Employee Information gathering
Procedure 2-Enter Data
Procedure 3-Check Gender
Procedure 4-Check Employee ID
Procedure 5-Update Data

Fig. 3.6 *Procedural Knowledge*

The problem with this kind of representation is that it is difficult to incorporate the inference mechanism in the procedure as the coding of logic, which consists of reasoning, becomes cumbersome. Moreover, acquisitional efficiency is also difficult to achieve as debugging and updating of large procedures is a tough task.

Declarative Knowledge

Declarative knowledge specifies what actions are to be performed. It consists of the description of facts and things or procedures. In declarative representation, knowledge is in a format that can be manipulated, decomposed and analysed, independent of its content. Declarative knowledge and explicit knowledge are treated as synonyms of each other. Declarative knowledge is an explicit knowledge that can be easily articulated, communicated and can be represented in formal languages.

3.3.2 Forward vs Backward Reasoning

Forward reasoning and backward reasoning depend upon the properties of the rule sets and the initial facts that you have set while solving a given problem. For logical reasoning tasks, you can either search forward from the initial stage (forward chaining) or you can search backward from the goal state (backward chaining).

Forward Reasoning

In AI, modus ponens is also called forward reasoning. Modus ponens is a common rule of inference, which is a function from the sets of formulae and it takes the following form:

If X, then Y.

X.

Therefore, Y.

The modus ponens can also be written as:

$$\frac{(X \to Y), X}{Y}$$

Here, the argument has two premises. The first premise is the if-then condition where X implies Y. In the second premise, X is the antecedent of the conditional claim, which is true. Thus, from the above two premises you can conclude that Y is the consequent of the conditional claim, which must be true.

For example, consider an argument, which has the modus ponens form:

If today is Wednesday, then I will go to the market.

Today is Wednesday.

Therefore, I will go to the market.

Here, the modus ponens concludes that if all the premises are true then the conclusion will also be true. Since the conclusion is true then the argument is sound. According to Rich and Knight, 'Begin by building a tree of move sequences that might be solutions by starting with the initial configuration(s) at the root of the tree. Generate the next level of the tree by finding all the rules whose *left* sides match the root node and using their right sides to create the new configurations. Generate the next level by taking each node generated at the previous level and apply to it all the rules whose left sides match it. Continue until a configuration that matches the goal state is generated.'

Backward Reasoning

Backward reasoning is an inference method that is used in AI. Backward reasoning is implemented in logic programming. According to Rich and Knight, 'Begin building a tree of move sequences that might be solutions by starting with the goal configuration(s) at the root of the tree. Generate the next level of the tree by finding all the rules whose *right* sides match the root node. These rules, if applied, would generate the state we wanted. Use the left sides of the rules to generate the nodes at this second level of the tree. Generate the next level of the tree by taking each node at the previous level and finding all the rules whose right sides match it. Then, using the corresponding left sides to generate the new nodes and continue until a node that matches the initial state is generated. This method of reasoning backward from the desired final state is often called *goal-directed reasoning*.'

Backward chaining is more suitable because the branching factor is significantly greater moving forward from the axioms than it is moving backward from the theorems to the axioms. For example, Prolog uses backward chaining.

3.3.3 Matching and Conflict Resolution

Matching is required between the current state and the preconditions of rules for better searching. This searching involves choosing from the rules, which can be applied at some particular point that can lead to a solution. Search control knowledge is defined as knowledge regarding different paths that are most likely to lead quickly to a goal state.

Matching Control Knowledge

Matching is used to provide a solution to a problem by searching. Matching between the current state and the preconditions of the rules is implemented by the following methods:

- Indexing
- Complex and approximate matching
- Conflict resolution

Indexing

Indexing involves selecting appropriate rules by simple searching. The searching is done through all the rules along with comparing each one's preconditions to the current state and then extracting all the rules that match these preconditions. Indexing involves two problems during searching that are as follows:

1. There may be a large number of rules in some problems; therefore, it is difficult to scan through all of them at every step of the search.
2. The preconditions of a rule may not be satisfied immediately by a specific state.

Complex and approximate matching

If the preconditions of a rule specify some properties that are not stated explicitly in the current state, then a complex matching process is required. A complex matching process requires a separate set of rules that should describe how some properties can be inferred from others. In some cases that include physical descriptions of the world, a more complex matching process is required, when the preconditions of the applied rules approximately match the current situation. The approximate matching process is difficult, because when the tolerance of the match is increased then, the number of rules are also increased that will match, and further the size of the main search process are increased.

Conflict resolution

The result of the matching process provides a list of rules whose antecedents have matched the description of the current state along with the variable bindings that were produced by the matching process. The search method decides on the order in which the rules will be applied. But, incorporating some of the decision-making into the matching process is sometimes useful. This phase of the matching process is known as conflict resolution. There are three approaches for the problem of conflict resolution in a production system, which are:

Preference based on rules

Preference based on objects

Preference based on states

Search Control Knowledge

A large amount of knowledge is required to solve the problems related to AI. If the knowledge, which is available for solving these AI problems, is not enough then a search has to be made for obtaining more knowledge in the knowledge base. The knowledge base must be systematically represented in order to efficiently search for knowledge in it. Knowledge can be represented in the form of facts in a knowledge base. A mechanism called search control knowledge can be used to control the knowledge search. In the search control knowledge, the knowledge about the different paths, which can lead to the goal, are obtained and reasoned. After this, the best possible path is selected to achieve the goal state.

3.3.4 Use of Non-Backtrack

Backtracking and non-backtracking methods and algorithms are used by AI researchers to develop various applications of AI, such as game playing, expert system, robotics, etc. AI uses genetic programming, which is a technique for getting programs to solve a task by implementing the random List Processing Programming (LISP) programs and selecting the fittest in millions of generations.

Backtracking

Backtracking is a systematic method to iterate through all the possible configurations of AI. It is a general technique which must be customized for each individual applications of artificial intelligence. It is a general algorithm used for finding all (or some) solutions to a computational problem. The algorithm works by incrementally building candidates to the solutions and abandoning each partial candidate *c* (backtracking) as soon as it determines that *c* cannot possibly be completed to a valid solution.

The eight queens puzzle is a classic example of the use of backtracking. It asks for all arrangements of eight queens on a standard chessboard so that no queen attacks any other. Here, the arrangements of *k* queens in the first *k* rows of the board, all in different rows and columns are referred to as partial candidates and any partial solution that contains two mutually attacking queens can be abandoned, since it cannot possibly be completed to a valid solution.

Backtracking is an important tool for solving constraint satisfaction problems, such as crosswords, verbal arithmetic and many other puzzles. It is often the most convenient (if not the most efficient) technique for parsing, for the knapsack problem and other combinatorial optimization problems. It is also the basis of the so-called logic programming languages such as Icon, Planner and Prolog.

Backtracking depends on user-given 'black box procedures' that define the problem to be solved, the nature of the partial candidates and how they are extended into complete candidates. It is therefore a metaheuristic rather than a specific algorithm, although, unlike many other metaheuristics, it is guaranteed to find all solutions to a finite problem in a limited amount of time.

Pseudocode

In order to apply backtracking to a specific class of problems, one must provide the data *P* for the particular instance of the problem that is to be solved, and six procedural parameters, *root*, *reject*, *accept*, *first*, *next* and *output*. These procedures should take the instance data *P* as a parameter and do the following:

- *Root*(*P*): Return the partial candidate at the root of the search tree.
- *Reject*(*P*,*c*): Return *true* only if the partial candidate *c* is not worth completing.

- *Accept*(*P*,*c*): Return *true* if *c* is a solution of *P*, and *false* otherwise.
- *First*(*P*,*c*): Generate the first extension of candidate *c*.
- *Next*(*P*,*s*): Generate the next alternative extension of a candidate, after the extension *s*.
- *Output*(*P*,*c*): Use the solution *c* of *P*, as appropriate to the application.

The backtracking algorithm reduces them to the call *bt*(*root*(*P*)), where *bt* is the following recursive procedure:

```
procedure bt(c)
  if reject(P,c) then return
  if accept(P,c) then output(P,c)
  $s \leftarrow first(P,c)$
  while $s \neq \Lambda$ do
    bt(s)
    $s \leftarrow next(P,s)$
```

Algorithm

The algorithm for backtracking method is defined in the following way:

```
bool finished = FALSE; /* found all solutions yet? */
backtrack(int a[], int k, data input)
{
int c[MAXCANDIDATES]; /* candidates for next position */
int ncandidates; /* next position candidate count */
int i; /* counter */
if (is_a_solution(a,k,input))
process_solution(a,k,input);
else {
k = k+1;
construct_candidates(a,k,input,c,&ncandidates);
for (i=0; i<ncandidates; i++) {
a[k] = c[i];
backtrack(a,k,input);
if (finished) return; /* terminate early */
}
}
}
```

Non-Backtracking System

Non-backtracking system is the opposite of backtracking system. It uses n-bit speed to find solutions to a computational problem. It supports one-sided error with fixed probability. The **uses of non-backtracking** are as follows:

- Search plays an important role in knowledge discovery in databases, such as Knowledge Discovery in Database (KDD) and data mining.

NOTES

- Non-backtracking system follows a search process to explore the useful knowledge from given data.
- It uses prune-search algorithm to determine the basic search techniques and highlight their performance and complexity.
- It is also used in systematic enumerative search methods, including best-first search, depth-first branch-and-bound and iterative deepening and neighborhood search methods, including gradient descent, artificial networks etc.
- By exploiting the dependency information, it recovers that information lost by backtracking and thus avoids the wasteful repetition of computation.
- The non-backtracking algorithm maintains an (extended) dependency set A = (U;D;F), which is defined in the algorithm, where U and F are sets of pairs of the form 's S' and D is a set of triplets of the form 's S'.
- It is used in functional data structures, which is also implemented in top-down algorithm.
- The non-backtracking Knuth-Morris-Pratt algorithm is frequently used in hash table to delegate each empty entry in the parsing table that is filled with a pointer to a special error routine. This algorithm provides significant speed over Brute-force attack. Brute-force string matching compares a given pattern with all substrings of a given text. Those comparisons between substring and pattern, proceed character by character unless a mismatch is found. Whenever a mismatch is found, the remaining character comparisons for that substring are dropped and the next substring is selected immediately. The brute-force searching usually involves automatic shifts upon mismatch to avoid unnecessary comparisons.
- The non-backtracking method indirectly supports in the processing of Augmented Transition Network (ATN) parsing. This network is used in transmitting the data especially data trafficking for virtual world, applications of multimedia, speech recognition, game playing etc.

Check Your Progress

10. What do you mean by the rule-based system?
11. What does conflict resolution strategy help to do?
12. Define the term chaining.
13. State the procedural knowledge.
14. Define the term forward reasoning.
15. What is matching?
16. Give an example of the use of backtracking algorithm.
17. Why is backtracking considered as a metaheuristic rather than a specific algorithm?

3.4 ANSWERS TO 'CHECK YOUR PROGRESS'

1. To understand simple facts in predicate logic, let us consider the following sentences:
   - Pratap was a man.
   - Pratap was a Rajpoot.

   In predicate logic, you can represent the facts described by these sentences as a set of wff's, depicted as follows:
   - Pratap was a man.

     man(Pratap)
   - Pratap was a Rajpoot.

     Rajpoot(Pratap)
2. In classical logic, modus ponendo ponens (Latin term for the way that affirms by affirming is abbreviated as MP or Modus Ponens. It is a valid and simple argument form and is also referred to as affirming the antecedent or the law of detachment. It is closely related to another valid form of argument, modus tollens.
3. The modus ponens rule may be written in sequent notation as:

   $$P \to Q, P \vdash Q$$

   And in rule form:

   $$\frac{P \to Q, P}{\therefore\ Q}$$
4. Resolution is a procedure in which the statements are converted into a standard form.
5. Natural deduction methods perform deduction in a manner similar to reasoning used by humans, e.g., in proving mathematical theorems. Forward chaining and backward chaining are natural deduction methods.
6. Backtracking is used in many AI applications to solve a number of schemes to improve its efficiency. Such schemes are termed dependency-directed backtracking.
7. Schank developed the model for representing knowledge for natural language.
8. In UML is indicated using a dashed line pointing from the dependent to the independent element.
9. Look-back schemes are used to control the specific decisions of where and how to go back in case of dead-ends.
10. A rule-based system refers to a system of encoding an expert's wisdom in a relatively narrow area into an automated system.
11. The conflict resolution strategy helps decide which rule is chosen to fire.

12. Chaining refers to a technique of teaching, which consists of breaking a task down into small steps and thereafter teaching each step within the sequence by itself.
13. In the procedural knowledge approach of knowledge representation, knowledge is stored in the form of procedures. The procedures contain the logic in the form of a code, which specifies when and how the actions are to be performed. LISP is one of the languages used for writing procedures.
14. In AI, modus ponens is also called forward reasoning. Modus ponens is a common rule of inference, which is a function from the sets of formulae.
15. Matching is required between the current state and the preconditions of rules for better searching. This searching involves choosing from the rules, which can be applied at some particular point that can lead to a solution. Search control knowledge is defined as knowledge regarding different paths that are most likely to lead quickly to a goal state.
16. The eight queens puzzle is a classic example of the use of backtracking algorithm.
17. Backtracking depends on user-given 'black box procedures' that define the problem to be solved, the nature of the partial candidates, and how they are extended into complete candidates; therefore, it is considered as a metaheuristic rather than a specific algorithm.

3.5 SUMMARY

- Predicate logic is one of the important knowledge representation languages, which is concerned with deriving the real-world facts.
- Predicate logic enables you to use variables and quantifiers for representing the real-world facts as statements, which are written as wff's.
- An instance is a binary predicate containing two arguments, where the first argument is an object of the class represented by the second argument.
- In classical logic, modus ponendo ponens (Latin term for the way that affirms by affirming is abbreviated as MP or Modus Ponens. It is a valid and simple argument form and is also referred to as affirming the antecedent or the law of detachment. It is closely related to another valid form of argument, modus tollens.
- The Curry-Howard correspondence between proofs and programs that are related to modus ponens function application: if f is a function of type $P \rightarrow Q$ and x is of type P, then $f\,x$ is of type Q.
- Resolution is a procedure in which the statements are converted into a standard form.
- The procedures in the resolutions are very easy, as the two clauses in the resolution procedures are known as the parent clauses.
- Natural deduction methods perform deduction in a manner similar to reasoning used by humans, e.g., in proving mathematical theorems. Forward chaining and backward chaining are natural deduction methods.

- Backtracking is used in many AI applications to solve a number of schemes to improve its efficiency. Such schemes are termed dependency-directed backtracking.
- Lookaheah schemes are used to control that which variable to be instantiated next or what value to be selected form the consistent options.
- A rule-based system refers to a system of encoding an expert's wisdom in a relatively narrow area into an automated system.
- A rule-based system can be considered as being similar to a multi-threaded system.
- Specific is the based on the number of conditions offered by rules. From the conflict set, the rule having maximum conditions is chosen. This is done on the assumption that it has the most relevance to the existing data if it has the most conditions.
- A pure production system generally comprises three fundamental components: a set of rules, a database and an interpreter for the rules.
- A knowledge-based system represents, acquires and applies knowledge for a specific objective.
- In this approach of knowledge representation, knowledge is stored in the form of procedures. The procedures contain the logic in the form of a code, which specifies when and how the actions are to be performed. LISP is one of the languages used for writing procedures.
- In AI, modus ponens is also called forward reasoning. Modus ponens is a common rule of inference, which is a function from the sets of formulae.
- Matching is required between the current state and the preconditions of rules for better searching. This searching involves choosing from the rules, which can be applied at some particular point that can lead to a solution. Search control knowledge is defined as knowledge regarding different paths that are most likely to lead quickly to a goal state.
- Backtracking and non-backtracking methods and algorithms are used by AI researchers to develop various applications of AI, such as game playing, expert system, robotics, etc.

3.6 KEY TERMS

- **Modus ponens:** In classical logic, modus ponendo ponens (Latin term for the way that affirms by affirming is abbreviated as MP or Modus Ponens. It is a valid and simple argument form and is also referred to as affirming the antecedent or the law of detachment. It is closely related to another valid form of argument, modus tollens.
- **Resolution:** Resolution is a procedure in which the statements are converted into a standard form.
- **Natural deductions:** Natural deduction is a kind of proof calculus in which logical reasoning is expressed by inference rules closely related to the 'natural' way of reasoning.

- **Rule-based system:** Rule-based system refers to a system of encoding an expert's wisdom in a relatively narrow area into an automated system.

NOTES

3.7 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is the difference between propositional logic and predicate logic?
2. Give some examples of propositional logic.
3. Why do you need predicate logic?
4. Define the term modus ponens.
5. What is resolution?
6. Name two methods of measuring used in natural deduction.
7. State the variable ordering.
8. What are the main features of rule-based systems?
9. How will you define the procedural vs declarative knowledge?
10. Differentiate between forward and backward reasoning.
11. What is matching control knowledge?
12. Define the term list processing programming.

Long-Answer Questions

1. Explain resolution from the point of view of propositional logic and predicate logic.
2. Give an example to represent simple facts with predicate logic.
3. Briefly explain about the modus pones. Give appropriate examples.
4. Discuss about the concept of resolution with the help of giving examples.
5. Explain briefly about the natural deduction. Give appropriate examples.
6. Explain the role of an interpreter in a rule-based system.
7. Differentiate between procedural and declarative knowledge with the help of example.
8. Illustrate the concept of forward and backward reasoning. Give appropriate examples.
9. Analyse the matching and conflict resolution with the help of examples.
10. Describe the use of non-back track with the help of giving examples.

3.8 FURTHER READING

Russell, Stuart J. and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*, 3rd Edition. New Jersey: Prentice Hall.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco (California): Morgan Kaufmann Publishers, Inc.

Knight Kevin, Elaine Rich and B. Nair. *Artificial Intelligence (SIE)*, 3rd Edition. New Delhi: Tata McGraw-Hill.

Sivanandam, S.N. and M. Paulraj. 2009. *Introduction to Artificial Neural Networks*. New Delhi: Vikas Publishing House Pvt. Ltd.

Rich, E. and K. Knight, *Artificial Intelligence*. New York: McGraw-Hill Book Company, 1991.

LiMin, Fu. 2003. *Neural Networks in Computer Intelligence.* New Delhi: Tata McGraw-Hill.

UNIT 4 STRUCTURED KNOWLEDGE REPRESENTATION AND SEMANTIC NET

NOTES

Structure

4.0 INTRODUCTION

A semantic network, or frame network is a knowledge base that represents semantic relations between concepts in a network. This is often used as a form of knowledge representation. It is a directed or undirected graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between concepts, mapping or connecting semantic fields. A semantic network may be instantiated as, for example, a graph database or a concept map. Typical standardized semantic networks are expressed as semantic triples. Semantic networks are used in natural language processing applications such as semantic parsing and word-sense disambiguation.

Frames are an artificial intelligence data structure used to divide knowledge into substructures by representing 'Stereotyped Situations'. They were proposed by Marvin Minsky in his 1974 article 'A Framework for Representing Knowledge'. Frames are the primary data structure used in artificial intelligence frame language; they are stored as ontologies of sets.

Slot and filler structures are types of data structures that are used to implement property inheritance. In these structures, knowledge is represented using objects and their attributes. Each object is connected with other objects or attributes using a relation. For example, a national-team is an object and player John, who is also an object, is a member of that team. In this example, the national-team and John are connected to each other using a relation 'is-a-member-of'. Conditional planning is a way to deal with uncertainty when planning. It is a planning method for managing bounded indeterminacy. It is a way to deal with uncertainty by checking what is actually happening in the environment at predetermined points in the plan.

Probabilistic reasoning involves the use of probability and logic to deal with uncertain situations. The result is a richer and more expressive formalism with a broad range of possible application areas. Probabilistic logics attempt to find a natural extension of traditional logic truth tables: the results they define are derived through probabilistic expressions instead. A difficulty with probabilistic logics is that they tend to multiply the computational complexities of their probabilistic and logical components. Other difficulties include the possibility of counter-intuitive results, such as those of Dempster–Shafer theory in evidence-based subjective logic. The need to deal with a broad variety of contexts and issues has led to many different proposals.

Certainty factors theory is an alternative to Bayesian reasoning which is used when reliable statistical information is not available or the independence of evidence cannot be assumed. Another methodology which is used to deal with reasoning is fuzzy logic which is a multi-valued logic derived from fuzzy set theory. In this unit, we will discuss these three tools of statistical reasoning in detail.

In this unit, you will learn about the semantic nets, frames, slot exceptions, slot values as object, handling uncertainties, probabilistic reasoning, use of certainty factor and fuzzy logic.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn about the semantic nets
- Explain about the frames
- Analysis the slot exceptions
- Elaborate on the handling uncertainties
- Discuss about the probabilistic reasoning
- Illustrate the use of certainty factor
- Define fuzzy logic

4.2 SEMANTIC NETS

Semantic nets help you to represent information using a set of nodes, which are connected to each other by arcs. Each arc is directed and labelled, which allows you to represent the relationship among nodes. Consider a network in which a person, John belongs to a football team. Using the semantic net, you can represent this network using nodes and arcs, as shown in Figure 4.1. In this figure, Is-a and instance relations are used to connect nodes. In addition, domain- specific relations such as Uniform-colour and has-part are also used.

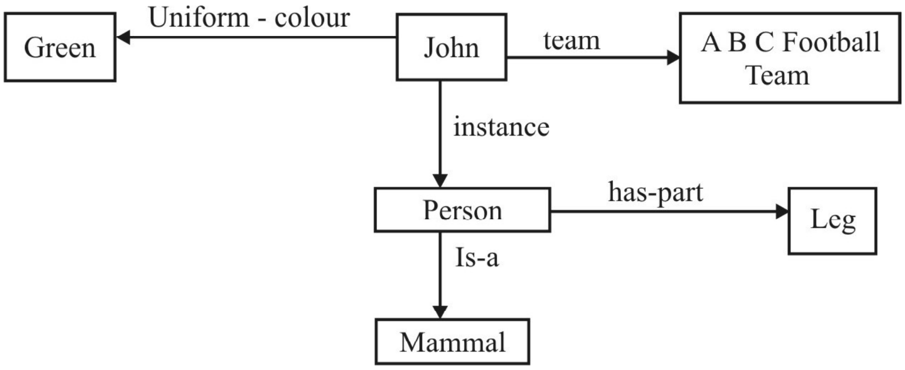


Fig. 4.1 *A Semantic Network*

Semantic networks can also represent the components of a declarative sentence. Consider the following statement:

'Michel gave the flower to john'

You can represent the sentence, Michel gave the flower to John, using the semantic net as shown in Figure 4.2. In this representation, EV5 is an event for which Michel is the agent, FLW1 is an object that belongs to class flower and John is the beneficiary.

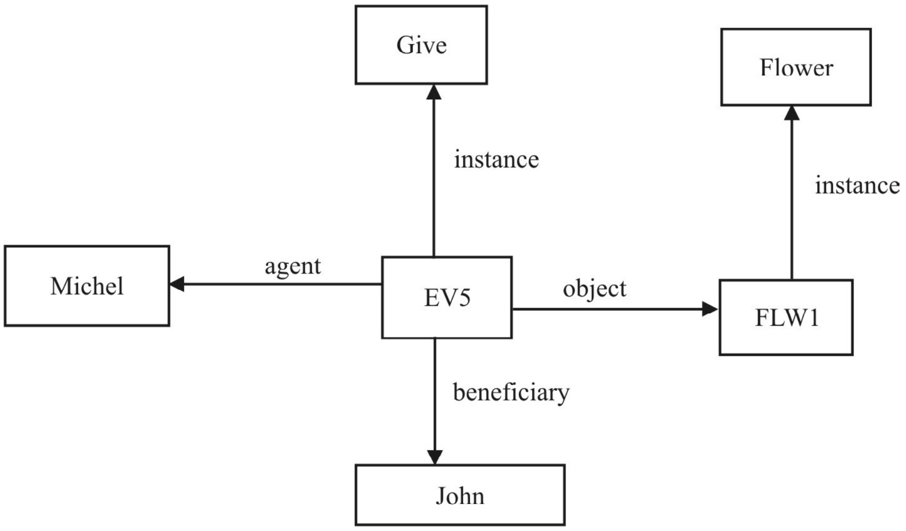


Fig. 4.2 *A Semantic Network representing a Declarative Sentence*

Partitioned Semantic Nets

Partitioned semantic nets are a type of semantic nets that allow you to represent quantified expressions using semantic nets. Quantified expressions are the expressions, which use quantifiers, existential (∃) and universal (∀). Consider the following statement:

"The dog bit the street hawker"

In this statement, various nodes will be dogs, bite and street-hawker, whereas d, b, s will be used to represent a particular dog, biting and a particular street hawker, respectively. Figure 4.3 shows the semantic net for the preceding statement.

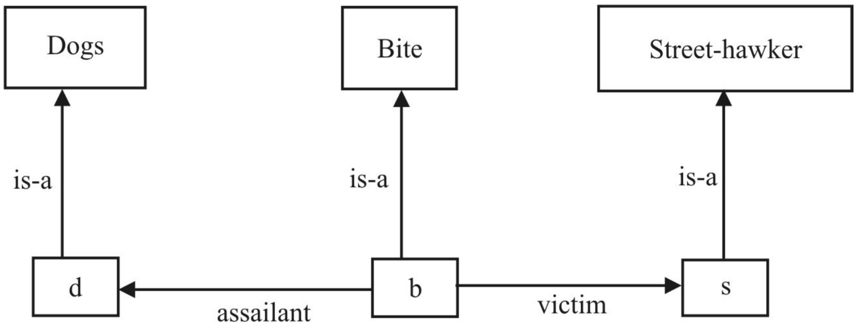


Fig. 4.3 *Semantic Net*

Now, consider the following sentence, which is:

'Every dog has bitten a street hawker' (2)

The equivalent quantifier expression for this statement is:

$\forall_x : \text{Dog}(x) = \exists y: \text{Street-hawker}(y) \wedge \text{Bite}(x, y)$

The above quantifier expression is represented by a partitioned semantic net. In this representation, an additional node (GS) is added to the semantic network. The g node is also added, which is an instance of node GS of a general statement about the world. Figure 4.4 shows the partitioned semantic net for statement (2).

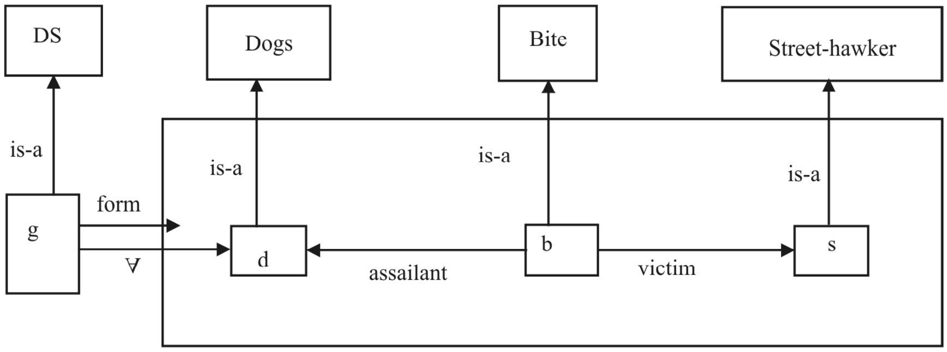


Fig. 4.4 *The Partitioned Semantic Net for Statement (2)*

Similarly, consider the following sentence in which the original sentence is modified:

'Every dog in the city has bitten the mail-carrier' (3)

For representing this statement, you need to consider three additional nodes, mail carriers, m, which is an instance of mail-carrier node and city-dogs. Figure 4.5 shows the semantic net for statement (3).

NOTES

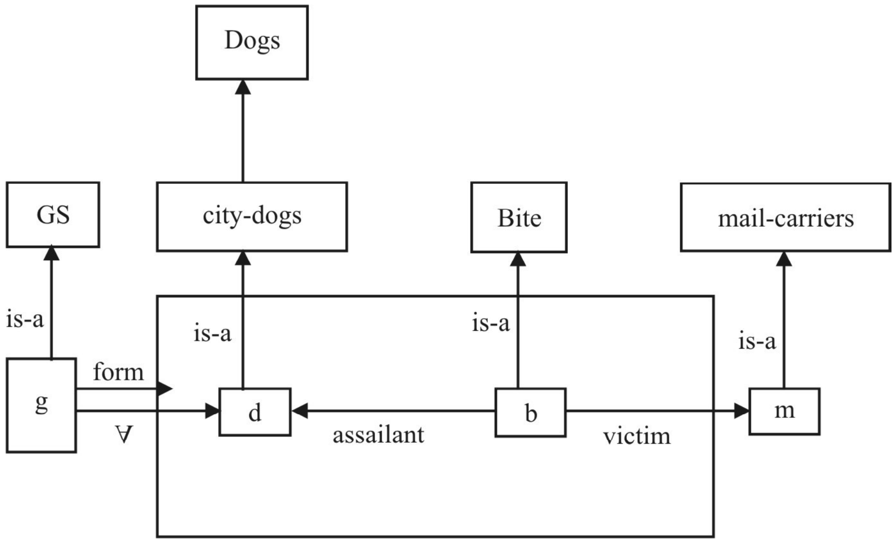


Fig. 4.5 *The Partitioned Semantic Net for Statement (3)*

Consider the following sentence in which an original sentence is also slightly modified:

'Every dog in the city has bitten every mail-carrier' (4)

For representing this statement, you need to apply a universal quantifier with two nodes d and m. Figure 4.6 shows the semantic net for statement (4).

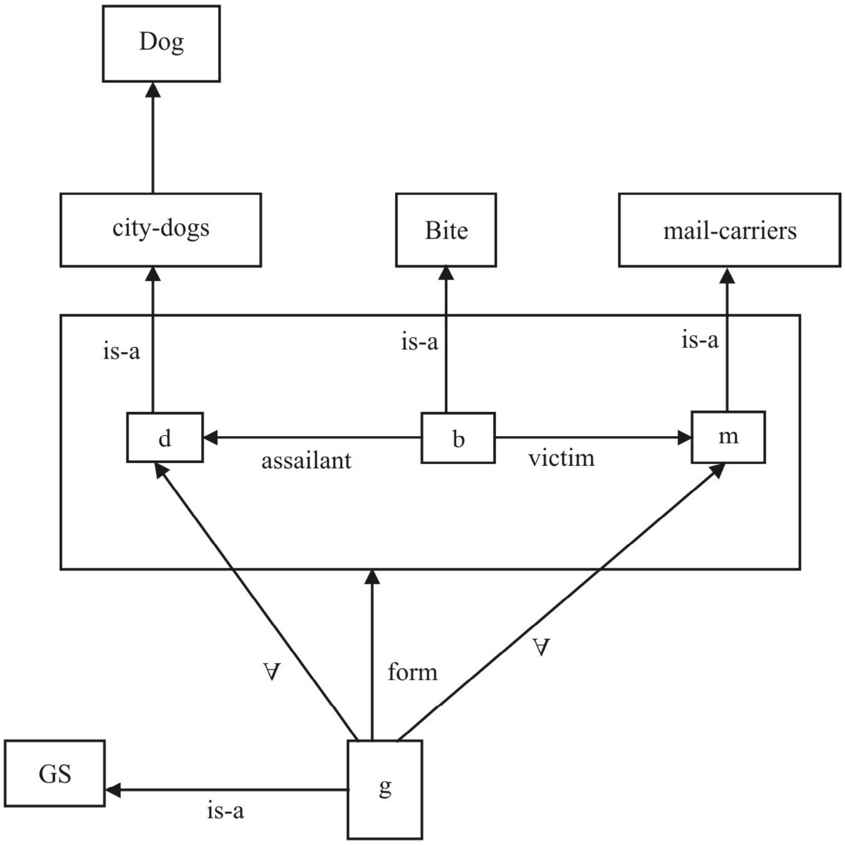
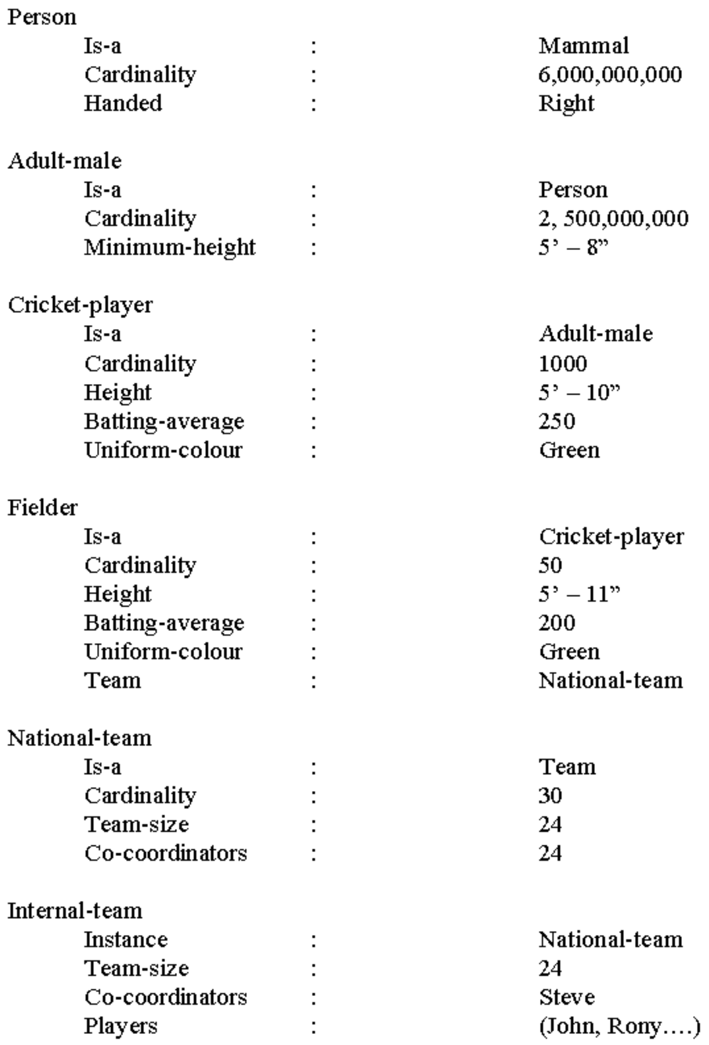


Fig. 4.6 *The Partitioned Semantic Net for Statement (4)*

4.2.1 Frames

Frames are also used to represent knowledge in the weak slot and filler structures. A frame is a collection of attributes and associated values to represent facts. Attributes in the frames are called slots and the associated values are used to define constraints, which are applied on the slots. A single frame is not sufficient enough to represent the fact. Therefore, you need to use a collection of frames forming the frame system in which the frames are connected to each other with associated values. Consider a frame system as shown in Figure 4.7. In this figure, Person, adult-male, cricket-players, cricket-team are all classes and frames John and internal-team are instances.



Person			
	Is-a	:	Mammal
	Cardinality	:	6,000,000,000
	Handed	:	Right
Adult-male			
	Is-a	:	Person
	Cardinality	:	2, 500,000,000
	Minimum-height	:	5' – 8"
Cricket-player			
	Is-a	:	Adult-male
	Cardinality	:	1000
	Height	:	5' – 10"
	Batting-average	:	250
	Uniform-colour	:	Green
Fielder			
	Is-a	:	Cricket-player
	Cardinality	:	50
	Height	:	5' – 11"
	Batting-average	:	200
	Uniform-colour	:	Green
	Team	:	National-team
National-team			
	Is-a	:	Team
	Cardinality	:	30
	Team-size	:	24
	Co-coordinators	:	24
Internal-team			
	Instance	:	National-team
	Team-size	:	24
	Co-coordinators	:	Steve
	Players	:	(John, Rony….)

Fig. 4.7 *A Frame System*

4.2.2 Slot Exceptions

Values associated with instances of a user-defined class are stored in slots. Each instance has a copy of the immediate class's set of slots, as well as any slots acquired through inheritance. The number of slots is limited by the amount of RAM available. With the exception of the keywords *isa* and *name*, which are reserved for usage in object patterns, the name of a slot can be any symbol.

The class precedence list for the instance's class is checked in order from most specific to most generic to determine the set of slots for an instance (left to right). A class's superclasses are less specific than its subclasses. With the exception of no-inherit slots, slots defined in any of the classes in the class precedence list are assigned to the instance. With the exception of composit, if a slot is inherited from multiple classes, the definition provided by the more specific class takes precedence.

For example,

```
(defclass A (is-a USER)
      (slot fooA)
      (slot barA))
(defclass B (is-a A)
    (slot fooB)
    (slot barB))
```

A has the following class precedence list: A USER OBJECT. There will be two slots in each instance of A: fooA and barA. B's class precedence list is as follows: B AN OBJECT OF A USER. There will be four slots in each instance of B: fooB, barB, fooA, and barA.

Facets make up slots in the same way that slots make up classes. Facets describe a slot's default value, storage, access, inheritance propagation, source of other facets, pattern-matching reactivity, visibility to subclass message-handlers, automatic creation of message-handlers to access the slot, the name of the message to send to set the slot, and constraint information. With the exception of shared slots, each object can still have its own value for a slot.

4.2.3 Slot Values as Object

In slot and filler structures, a slot is an object or attribute and filler is a value of any data type such as integer and string, which a slot can take. Consider an example of slot and filler structure, as shown in Figure 4.8. In this figure, Person, Adult-Male, Baseball-Player, Fielder, Pitcher, Pee-Wee-Reese and Three-Finger-Brown are objects. Whereas, Right, 6-11, 7-12, .352, Equal to Handed, .262, .106 are attributes. Attributes and objects are considered as slots in a slot and filler structure, which can be replaced by other values called fillers.

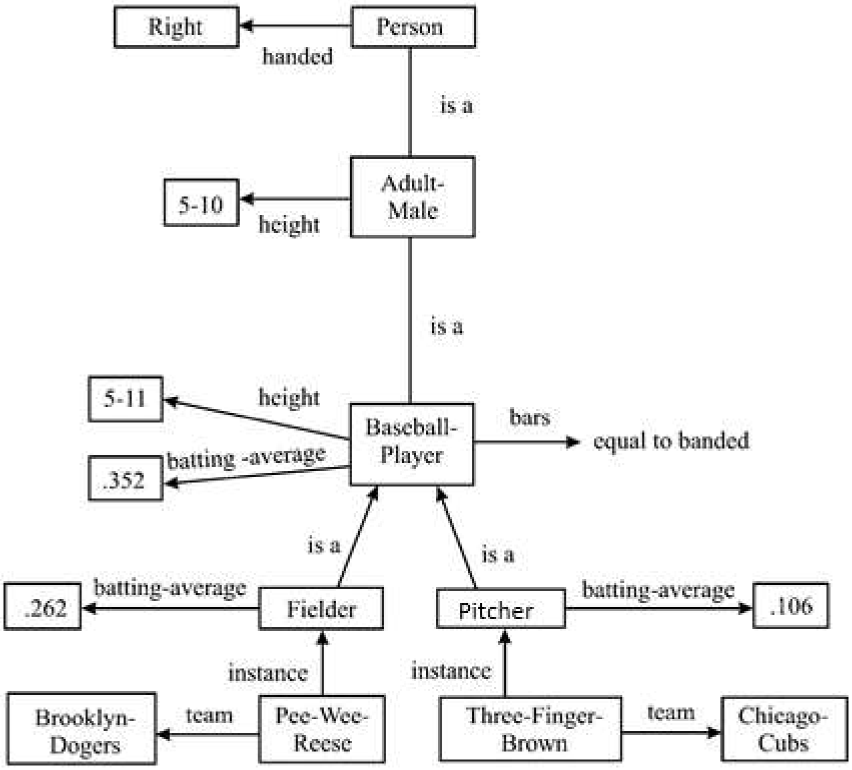


Fig 4.8 *Example of Slot and Filler Structure*

Slot and filler structures are divided into two categories, which are:

- **Weak Slot and Filler Structure**: A slot and filler structure that does not apply any rules on the content of the structure. Examples of weak and slot filler structure are semantic net and frame.
- **Strong Slot and Filler Structure**: A slot and filler structure in which links between objects are based on rigid rules. Examples of strong slot and filler structure are CD, scripts and CYCorp.

4.3 HANDLING UNCERTAINTIES

Uncertainty is a term specifically used in subtly different ways in a number of fields, including philosophy, physics, statistics, engineering, information science, and so on. The theory of uncertainty is concerned with predictions of future events or to the unknown. Uncertainty is referred as a state of having limited knowledge where it is impossible to exactly describe the existing state, a future outcome or more than one possible outcome. The uncertainty can be measured using a set of possible states or outcomes where probabilities are assigned to each possible state or outcome—this also includes the application of a probability density function to continuous variable.

Measures of Uncertainty: Shannon's Entropy

Entropy is a measure of unpredictability of information content. Let *X* be a discrete random variable taking a finite number of possible values $x_1, x_2, \ldots, x_n$ with

NOTES

probabilities $p_1, p_2, \ldots, p_n$, respectively, such that $p_i \geq 0, i = 1, 2, \ldots, n$ $\sum_{i=1}^{n} p_i = 1.$ We try to get a number that will measure the amount of uncertainty. Let *h* be a function defined on the interval (0, 1) and *h*(*p*) be interpreted as the uncertainty associated with the event $X = x_i, i = 1, 2, \ldots, n$ or the information conveyed by revealing that *X* has taken on the value x_i in a given performance of the experiment. For each *n*, we shall define a function H_n of the *n* variables $p_1, p_2, \ldots, p_n$. The function $H_n(p_1, p_2, \ldots, p_n)$ is to be interpreted as the average uncertainty associated with the event $\{X = x_i\}, i = 1, 2, \ldots, n$ given by

$$H_n(p_1, p_2, \ldots, p_n) = \sum_{i=1}^{n} p_i h(p_i).$$

Thus, $H_n(p_1, p_2, \ldots, p_n)$ is the average uncertainty removed by revealing the value of *X*. For simplicity we shall denote,

$$\Delta_n = \left\{ P = (p_1, p_2, \ldots, p_n) : p_i \geq 0, \sum_{i=1}^{n} p_i = 1 \right\}.$$

Following are some axiomatic characterizations of the measure of uncertainty $H_n(p_1, p_2, \ldots, p_n)$ which are used to get at its exact expression. For that, let *X* and *Y* be two independent experiments with *n* and *m* values, respectively. Let $P = (p_1, p_2, \ldots, p_n) \in \Delta_n$ be a probability distribution associated with *X* and $Q = (q_1, q_2, \ldots, q_m) \in \Delta_m$ be a probability distribution associated with *Y*. This lead us to write that, $H_{nm}(P * Q) = H_n(P) + H_m(Q),$ for all $P = (p_1, p_2, \ldots, p_n) \in \Delta_n$, $Q = (q_1, q_2, \ldots, q_m) \in \Delta_m$, and $P * Q = (p_1 q_1, \ldots, p_1 q_m, p_2 q_1, \ldots, p_2 q_m, \ldots, p_n q_1, \ldots,$ $p_n q_m) \in \Delta_{nm}$. Replacing $p_i h(p_i)$ by $f(p_i), \forall i = 1, 2, \ldots, n,$ we get the equation

$$H_n(P) = \sum_{i=1}^{n} f(p_i).$$

In most artificial intelligence applications, especially in expert system, it is required to make decisions which are based on uncertain data and uncertain models. As such, several methods have been developed for reasoning with different kinds of uncertainty.

We often have to take decisions based on uncertain knowledge. In our private life we have to take decisions, such as which job to take, which house to buy or where we should invest our money. Such types of decisions are purely based on uncertain knowledge. In professional activities also we have take decisions based on uncertain knowledge. Therefore, any reasoning method which tries to replicate human reasoning should be able to draw conclusions from uncertain models and uncertain data.

There are various kinds of uncertainty in all aspects of a reasoning system but the 'reasoning with uncertainty' or 'reasoning under uncertainty' research in Artificial Intelligence (AI) has been focused on the uncertainty of truth value, i.e., to allow and process truth values other than 'True' and 'False'.

In order to develop a system that reasons with uncertainty it is required to provide the following:

- A semantic explanation about the origin and nature of the uncertainty.
- A way to represent uncertainty in a formal language.
- A set of inference rules that derive uncertain, but well justified conclusions.
- An efficient memory control mechanism for uncertainty management.

Artificial Intelligence (AI) systems must have ability to reason under conditions of uncertainty. The reasons of uncertainties are given below:

- Incompleteness Knowledge
- Inconsistency Knowledge
- Changing Knowledge

Methods of Handling Uncertainty

In AI we must often represent and reason about uncertain information. There are multiple approaches to handle uncertainty, such as non-monotonic logic, probabilistic reasoning, fuzzy logic, etc. Among these probabilistic methods are most precise but it is often hard to apply. Bayesian probability is a popular interpretation on the concept of probability. Fuzzy logic is a form of many-valued logic which is used to deals with reasoning that is approximate rather than fixed and accurate.

The three main methods of handling uncertainties are given below:

- Non-Monotonic Logic
- Probabilistic Reasoning
- Fuzzy Logic

Non-Monotonic Logic

A non-monotonic logic is a formal logic whose consequence relation is not monotonic. If the truth of a proposition changes when new information (axioms) is added, the logic becomes non-monotonic. A non-monotonic logic allows a statement to be retracted. It is used to formalize reasonable reasoning. For example, let us consider two sentences: ‘Birds typically fly’ and ‘Tweety is a bird’. Now according to non-monotonic logic, it can be said that ‘Tweety (presumably) flies’. But if Tweety is a penguin, it is incorrect to conclude that Tweety flies. Thus it is seen that the conclusion of non-monotonic argument may not be correct.

All non-monotonic reasoning is concerned with consistency. Inconsistency is resolved, by removing the relevant conclusion(s) derived by default rules. The truth value (True or False), of propositions, such as ‘Tweety is a bird’ accepts default that is normally true, such as ‘Birds typically fly’. Conclusions derived from the above two sentence was ‘Tweety flies’. Thus it is seen that when an inconsistency is recognized, only the truth value of the last type is changed.

Probability Reasoning

Logic-based approaches assume that everything is either false or true. However, it is often useful to believe that something is probably true or true with probability.

This approach, which makes dealing with random, unpredictable and impractical problems easier, is known as probability reasoning.

Bayesian Probability Theory

Bayesian probability is one of the most popular interpretations of the concept of probability. It shows the relation between one conditional probability and its inverse; for example, the probability of a hypothesis given observed evidence and the probability of that evidence given the hypothesis. It is named after Reverend Thomas Bayes (1702–1761), who studied how to compute a distribution for the probability parameter of a binomial distribution. Ironically, Bayes was a minor figure in the history of science, who had little or no impact on the early development of statistics; it was the French mathematician Pierre-Simon Laplace (1749–1827) who pioneered and popularized what is now called Bayesian probability theory.

The Bayesian probability theory interprets the state of knowledge in two different ways. While the objectivist view justifies the rules of the theory by requirements of rationality and consistency and interprets them as an extension of logic, the subjectivist view, measures the state of knowledge as a 'personal belief'. Many modern machine learning methods are based on objectivist Bayesian principles.

The fundamental notion of the theory is that of conditional probability, P(H | E), which means that a hypothesis H is true given that evidence E for the hypothesis has been observed. The theorem, in the light of new evidences, adjusts the probabilities using the following formula:

$$P(H \mid E)\frac{P(E / H)P(H)}{P(E)}$$

Where,

- $P(H)$ is called the prior probability of H that was inferred before new evidence, E, became available.
- $P(E \mid H)$ is called the conditional probability of seeing the evidence E if the hypothesis H happens to be true. It is also called a likelihood function when it is considered as a function of H for fixed E.
- $P(E)$ is called the marginal probability of E. It is the a priori probability of witnessing the new evidence E under all possible hypotheses. It can be calculated as the sum of the product of all probabilities of any complete set of mutually exclusive hypotheses and corresponding conditional probabilities:

$$P(E) = \sum P(E|H_i)P(H_i)$$

- $P(H \mid E)$ is called the posterior probability of H given E.

$$P(E) = \sum P(E|H_i)P(H_i)$$

- $P(H \mid E)$ is called the posterior probability of H given E.

The factor $P(E \mid H)/P(E)$ represents the impact that the evidence has on the belief in the hypothesis. If it is likely that the evidence E would be observed when the hypothesis under consideration is true, but unlikely that E would have been the outcome of the observation, then this factor will be large. Multiplying the prior probability of the hypothesis by this factor would result in a larger posterior probability of the hypothesis given the evidence. Conversely, if it is unlikely that the evidence E would be observed if the hypothesis under consideration is true, but a priori likely that E would be observed, then the factor would reduce the posterior probability for H. Under Bayesian inference Bayes' theorem measures how much new evidence should alter a belief in a hypothesis.

To illustrate the use of the Bayes' theorem in probability theory, let us consider an example. Suppose there are two full bowls of cookies. While bowl one has 10 chocolate chips and 30 plain cookies, bowl two has 20 of each. A person is asked to pick a bowl at random and then pick a cookie at random. Assuming that he treats both the bowls and all the cookies similarly, let us suppose he picks a plain cookie. In that case, how probable is it that the person has picked it out of bowl one?

Intuitively, the answer should be more than a half, since there are more plain cookies in bowl one. However, Bayes' theorem can give a precise answer. Let H_1 correspond to bowl one and H_2 to bowl two. It is given that the bowls are identical from the person's point of view, thus $P(H_1) = P(H_2)$, and the two must add up to 1, so both are equal to 0.5. The event E is the observation of a plain cookie. From the contents of the bowls, we know that $P(E | H_1) = 30 / 40 = 0.75$ and $P(E | H_2) = 20 / 40 = 0.5$. Bayes' formula then yields

$$P(H_1|E) = \frac{P(E \mid H_1)\, P(H_1)}{P(E \mid H_1)\, P(H_1) + P(E \mid H_2)\, P(H_2)}$$

$$= \frac{0.75 \times 0.5}{0.75 \times 0.5 + 0.5 \times 0.5}$$

$$= 0.6.$$

Thus, the prior probability, $P(H_1)$ which was 0.5, now becomes $P(H_1 | E)$, which is 0.6.

The Bayesian probability calculus has been supported by several arguments, such as the Cox axioms, the Dutch book argument, arguments based on decision theory and de Finetti's theorem.

Richard T. Cox proved that Bayesian updating follows from several axioms, including two functional equations and the controversial hypothesis that probability is a continuous function.

The Dutch book argument was proposed by de Finetti and is based on betting. A Dutch book is made when a clever gambler places a set of bets that guarantee a profit, no matter what the outcome is of the bets. If a bookmaker follows the rules of the Bayesian calculus in the construction of his odds, a Dutch book cannot be made.

However, Ian Hacking noted that traditional Dutch book arguments did not specify Bayesian updating: they left open the possibility that non-Bayesian updating rules could avoid Dutch books. In fact, there are non-Bayesian updating rules that also avoid Dutch books. The additional hypotheses sufficient to specify (uniquely) Bayesian updating are substantial, complicated, and unsatisfactory, according to Bas van Fraassen's book *Laws and Symmetries*.

A decision-theoretic justification of Bayesian methods was given by Abraham Wald, who proved that every Bayesian procedure is admissible. Conversely, every admissible statistical procedure is either a Bayesian procedure or a limit of Bayesian procedures. Wald's result also established the Bayesian formalism as a fundamental technique in such areas of frequentist statistics as point estimation, hypothesis testing, and confidence intervals.

Bayesian methods have been used for hundreds of years, so there are many examples of Bayesian inference to scrutinize. Of the tens of thousands of papers published using Bayesian methods, few criticisms have been made of implausible priors in concrete applications. Such criticisms are themselves welcomed by Bayesian statisticians, as part of the inevitable revisions of science. Nonetheless, worries about the possible problems of Bayesian methods continue to appear. Concerns have been raised that a Bayesian view could be problematic for scientific judgements, since a Bayesian information processor tends to confirm already established views and to suppress controversial views. Such worries have not so far been accompanied by experimental evidence, nor have they published examples of implausible priors that have led to practical problems.

4.3.1 Probabilistic Reasoning

It is one of the problem solving systems that collects evidences of the problems and modify their behaviour on the basis of the evidence. In the probabilistic approach, PROSPECTOR, which is a representative of the system, is used to handle uncertainty. This approach uses the Bayes' theorem to solve the problem of uncertainty, as shown in the code:

$P(H|E)=P(E|H)P(H)/(\text{“}_{i}P(E/H_{i})P(H_{i}))$

In this code, P refers to the probability function.

You can combine the evidence under the assumption of conditional independence in the probability function. The formula that combines the evidence in the probability function is as follows:

$P(H|E1,E2)=P(E1|H)P(E2/H)P(H)/(\text{“}_{i}P(E1/H_{i})P(E2/H_{i})P(H_{i}))$

You can define odds in the probability function using the following formula:

$O(H)=P(H)/P(\neg H)=P(H)/(1-P(H))$

This code shows the odds of H in the probability function.

You can define a likelihood ratio of E with respect to H, as shown in the code:

$\lambda(E,H)=P(E|H)/(P(E|\neg H)$

From this equation, odds-likelihood formulation of Bayes' rule is derived, as shown in the code:

$O(H|E)= \lambda (E,H)O(H)$

You can combine evidence with the odds-likelihood formulation by using the following formula:

$O(H|E1,E2)= \lambda(E2,H) \lambda(E1,H)O(H)$

It is recommended to update odds than probabilities since it's easier. You can obtain the probability by odds easily using the following formula:

P(H)=O(H)/(1+O(H))

NOTES

When the information is transmitted using rules, the evidence derived from the rule's conclusion is uncertain. In this situation, the formula for probability with evidence would be:

P(H|E')=P(H|E)P(E|E')+P(H|¬E)P(¬E|E')

In the above code, `E'` refers to the observed evidence and `E` refers to the actual and absolute evidence.

`P(H|E')` is calculated using a linear interpolation between two extreme cases, `P(H|E)`, which is known to be true and `P(H|¬E)`, which is known to be false. The linear interpolation scheme uses three reference points:

When P(E|E')=0, P(H|E') = P(H|¬E)

When P(E|E')=P(E), P(H|E') = P(H)

When P(E|E')=1, P(H|E')=P(H|E)

Check Your Progress

1. What is a semantic net?
2. Define the term filler.
3. What is a slot?
4. Who gave a decision-theoretic justification of Bayesian methods?

4.4 USE OF CERTAINTY FACTOR

The Certainty Factor (CF) model is the technique used to handle the uncertainty in rule-based systems. The first CF model was developed by Shortliffe and Buchanan in 1975 for MYCIN. MYCIN is an expert system used to diagnose and treat meningitis and infections of the blood.

A MYCIN rule is given here:

If—

- The infection to be treated is meningitis.
- Any evidence of grave skin or soft tissue infection is not available.
- The germs were not observed in the culture.
- The infection is bacterial.

Then—

- It can be concluded that the causal agent is *Staphylococcus* (0.75).

The CF model is based on the following assumptions:

- Faults or hypotheses are mutually exclusive and exhaustive.
- Pieces of evidence are conditionally independent, given each fault or hypothesis.

In general, rule-based systems are based on rules such as 'if *e* then *h,'* where *e* is an evidence for the proposition *h.* The CF model associates only one CF to every rule to help an expert represent the uncertainty.

Any number from 1 to 0 linked to a condition or action of a rule is called a certainty factor. In other words, a certainty factor is attached to every component of a condition. For example, in case, there are two forms of a condition: A and B. There would be a CF for A and another for B.

A CF denotes the change in the hypothesis in accordance with the evidence as follows:

- A CF of 1 denotes high certainty for the proposition.
- A CF of 0 denotes that there is no information for the certainty or uncertainty of the proposition.
- A CF of – 1 denotes high uncertainty for the proposition.
- A CF from 0 to 1 denotes an increase in the certainty of the proposition.
- A CF from – 1 to 0 denotes a decrease in the certainty of the proposition.

Uncertainty arises from the following two sources:

- The rule itself
- The answers of a user

The CF model is used in modern rule-based expert systems. It is based on probability theory.

The main components of a CF model are as follows:

- **Production rules and attached certainty factors:** Expert systems are based on the production rule: 'if *e* then *h*' or '$e \rightarrow h$'. Where *e* is a Boolean set of conditions and *h* is a combination of conclusions or hypotheses. The production rule can be stated as 'if evidence *e* is true, then the hypothesis *h* is also true'.
- **Data supplied by the user:** The expert system asks a user to provide the actual data. The user has to associate a CF to each part of the data.
- **An inference engine:** It is the technique used to apply the production rules. It involves confirming or declining certain intermediate hypothesis. The CF attached to the intermediate hypothesis is computed on the basis of CF attached to the production rules.

Measure of Belief (MB)

Shortliffe and Buchanan who developed the concept of certainty factors got expert doctors express their level of certainty or uncertainty in MYCIN and then determined the corresponding CFs from these levels.

Let the measure of belief be MB. The production rule is:

If e then h.

Or

MB(h, e) = 1 if $p(h) = 1$

and

$$MB(h,e) = \frac{p(h,e) - p(h)}{1 - p(h)}$$

otherwise.

Here, the expert determines the level of the evidence, e, that increases his/her certainty or belief, which he/she would have if the evidence $[1 - p(h)]$ is absent. Consider the following cases:

- In case of a very weak evidence, $p(h,e) - p(h)$ is nearly zero, and the belief does not change.
- In case of a very strong evidence, $p(h, e) - p(h)$ will equal $1 - p(h)$ and the MB will equal 1, that is, there is no disbelief or uncertainty.

Measure of Disbelief (MD)

To estimate a disbelief, the formula is as follows:

$$\text{MD}(h, e) = 1 \text{ if } p(h) = 0$$

$$MB(h,e) = \frac{p(h) - p(h \mid e)}{p(h)}$$

otherwise.

The CF is calculated from MB and MD:

$$CF = \frac{MB - MD}{1 - \min(MB, MD)}$$

The preceding formula gives CFs from – 1.0 (total disbelief) to 1.0 (total belief). If MD is considered to be 0, CFs would range from 0 (complete disbelief) to 1 (complete belief). If MD is not used, then CF will equal MB.

McAllister Scheme

David McAllister developed a technique for ‘certainty factors’ to be used in an ‘expert system’.

The function of a CF in the McAllister scheme is to determine the accuracy or reliability of a hypothesis. A CF is neither a probability nor a truth value.

As per this scheme, a certainty factor is a number that varies from 0.0 to 1.0. A number 0.6 is assigned to a ‘suggestive evidence’. A number such as 0.8 is assigned to a ‘strongly suggestive evidence’.

McAllister scheme allows addition of latest evidences. A positive sum would increase certainty. The rule for addition of two positive certainty factors is as follows:

$$CFcombine\ (CF_a\ CF_b) = CF_a + CF_b\ (1 - Cf_a)$$

Where CF_a and CF_b are two certainty factors. The influence of the second certainty factor is decreased from the remaining uncertainty of the first, and the result is added to the certainty of the first.

Using the Model

In order to make the certainty factor model perform satisfactorily, the following guidelines must be followed:

- To minimize the occurrence of conflicting derivations for a single hypothesis, the condition parts of production rules drawing opposite conclusions, must be specified as 'mutually exclusive' as possible.
- The several pieces of evidence pertaining to a single hypothesis must be grouped in a way that the Boolean combinations of evidence mentioned in separate production rules are as independent as possible, and the atomic pieces of such a Boolean combination evidence within a production rule is as strongly correlated as possible.
- The production rules must be specified in such a way that a chain of rules that may arise while actually reasoning with the system is able to narrow the focus of attention.

Disadvantages of certainty factors

The CF model suffers from the following two demerits:

1. The concept of modeling human uncertainty by means of numerical certainty factors is controversial. Some people consider the formulae used for CF model invalid.
2. This model needs more work from the user than the binary logic mode. The user should assign a CF to every probable answer. In case the user ignores or forgets to assign a CF for a hypothesis, then the system would assume a default value of 0 (meaning 'do not know') for that hypothesis. This may or may not be a correct interpretation of the user's belief.

4.5 FUZZY LOGIC

Fuzzy logic is a variety of multi-valued logic that has been derived from the fuzzy set theory. It has been used to deal with reasoning that is estimated rather than precise. This is in contrast to crisp logic where binary sets have binary logic. On the other hand, fuzzy logic variables may contain a truth value that can range from 0 to 1 and that is not constrained to the two truth values of classic propositional logic. Also, on using linguistic variables, the given degrees can be managed by specific functions.

Fuzzy logic can be implemented in various kinds of systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems. The concept came about as a result of the proposal of the fuzzy set theory, made by Lotfi Zadeh in 1965 and has been used by scholars and researchers in many fields ranging from control theory to artificial intelligence. It is a powerful method of problem solving with a range of applications in embedded control and information processing. The concept provides a very simple way to derive definite conclusions from vague, ambiguous or imprecise information. In some ways, it is similar to the human decision-making process and is much faster.

Fuzzy logic is extremely different from classical logic which needs an in-depth understanding of a system, exact equations and precise numeric values. It uses an alternative way of thinking that allows the modeling of complex systems using a higher level of abstraction that originates from human knowledge and experience. This knowledge can be expressed with subjective concepts which are mapped into precise numeric ranges.

Characteristics of Fuzzy Logic

Fuzzy logic has a number of characteristics that are unique and these features are as follows:

- It is a robust system and does not need precise inputs and can be programmed to fail safely, if a feedback sensor quits or is destroyed.
- The fuzzy logic controller executes user-defined rules that rule the target control system. This can be altered to improve the performance of the system. A range of new sensors can be easily added to the system by generating rules of governance that are appropriate.
- It is not restricted by a few feedback inputs and one or two control outputs. It is also not necessary to measure or calculate rate-of-change parameters for it to be implemented.
- Fuzzy logic can also control systems that are non-linear and that would be impossible to control mathematically. This will allow the control of systems that would usually be considered to be unfeasible.

Fuzzy Rules

Decisions are made by humans on the basis of a set of rules and conventions. Even though we are unaware, these decisions are based on computer-like if-then statements. For example, if you are hungry now, then you may decide to eat a salad. If the salad does not taste good but a fruit does, then you will make a decision not to eat the salad but the fruit. Therefore, these rules tend to relate ideas and events.

Fuzzy machines that imitate the behaviour of human beings, work in the same manner. However, the decision and the way to come upon that decision are replaced by fuzzy sets and the rules are taken over by fuzzy rules. These fuzzy rules also work by using a range of if-then statements. Fuzzy rules define fuzzy patches, which is the essential concept in fuzzy logic.

Machines are made more efficient by using a concept designed by Bart Kosko, which is called the Fuzzy Approximation Theorem (FAT). This theorem states that a finite number of patches can cover a curve (see Figure 4.9) and if the patches are large, then the rules are sloppy. Conversely, if the patches are small then the rules are correct.

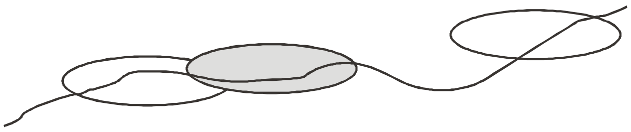


Fig. 4.9 Fuzzy Approximation Theorem

NOTES

Fuzzy Control

Fuzzy control makes use of the theory of fuzzy rules. By using a procedure that was developed by Ebrahim Mamdani, the three following steps are used to create a fuzzy controlled machine:

1. **Fuzzification:** This entails the use of membership functions to graphically describe a situation.
2. **Rule Evaluation:** This involves the application of fuzzy rules.
3. **Defuzzification:** It entails the process of obtaining the crisp or actual results.

Check Your Progress

5. What is MYCIN?
6. List the main components of a CF model.
7. Define the term fuzzy logic.
8. Who invented the fuzzy approximation theorem?

4.6 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Semantic nets help you to represent information using a set of nodes, which are connected to each other by arcs.
2. Filler is a value of a data type such as integer and string, which a slot can take.
3. Slot is an object or attribute, which contains general information about an object or attribute.
4. The decision-theoretic justification of Bayesian methods was given by Abraham Wald.
5. MYCIN is an expert system used to diagnose and treat meningitis and infections of the blood.
6. The main components of a Certainty Factor (CF) model are as follows:
   - Production rules and attached certainty factors
   - Data supplied by the user
   - An inference engine.
7. Fuzzy logic is a variety of multi-valued logic that has been derived from the fuzzy set theory.
8. The Fuzzy Approximation theorem was invented by Bart Kosko.

4.7 SUMMARY

NOTES

- Semantic nets help you to represent information using a set of nodes, which are connected to each other by arcs. Each arc is directed and labelled, which allows you to represent the relationship among nodes.
- Partitioned semantic nets are a type of semantic nets that allow you to represent quantified expressions using semantic nets.
- Frames are also used to represent knowledge in the weak slot and filler structures.
- A frame is a collection of attributes and associated values to represent facts.
- Attributes in the frames are called slots and the associated values are used to define constraints, which are applied on the slots.
- In slot and filler structures, a slot is an object or attribute and filler is a value of any data type such as integer and string, which a slot can take.
- It is one of the problem solving systems that collects evidences of the problems and modify their behaviour on the basis of the evidence. In the probabilistic approach, PROSPECTOR, which is a representative of the system, is used to handle uncertainty.
- The Certainty Factor (CF) model is the technique used to handle the uncertainty in rule-based systems.
- The first CF model was developed by Shortliffe and Buchanan in 1975 for MYCIN.
- MYCIN is an expert system used to diagnose and treat meningitis and infections of the blood.
- Shortliffe and Buchanan who developed the concept of certainty factors got expert doctors express their level of certainty or uncertainty in MYCIN and then determined the corresponding CFs from these levels.
- Fuzzy logic is a variety of multi-valued logic that has been derived from the fuzzy set theory. It has been used to deal with reasoning that is estimated rather than precise.
- Fuzzy logic variables may contain a truth value that can range from 0 to 1 and that is not constrained to the two truth values of classic propositional logic.
- Fuzzy logic can be implemented in various kinds of systems ranging from simple, small, embedded micro-controllers to large, networked, multi-channel PC or workstation-based data acquisition and control systems.
- Fuzzy logic can also control systems that are non-linear and that would be impossible to control mathematically. This will allow the control of systems that would usually be considered to be unfeasible.
- Fuzzy control makes use of the theory of fuzzy rules. By using a procedure that was developed by Ebrahim Mamdani.

4.8 KEY TERMS

- **Partitioned semantic nets:** They are a type of semantic nets that allow you to represent quantified expressions using semantic nets.
- **Frames:** They collections of attributes and associated values to represent facts.
- **Weak slot and filler structure:** It is a slot and filler structure, which does not apply any rules on the content of the structure.
- **Strong slot and filler structure:** It is a slot and filler structure in which the links between objects are based on rigid rules.
- **Fuzzification:** It is the process of transforming crisp values into grades of membership for linguistic terms of fuzzy sets.
- **Certainty Factor (CF) model:** Certainty Factor (CF) model is a technique used to handle the uncertainty in rule-based systems.

4.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. How will you define the semantic nets?
2. What are frames?
3. Define a slot and filler structure.
4. What is a weak slot and filler structure?
5. What is a strong slot and filler structure?
6. Write a short note on:
   (i) Logic programming
   (ii) Probabilistic approach using Bayes' theorem
7. What is Certainty Factor (CF)?
8. List the unique features of fuzzy logic.
9. What are the steps involved in the fuzzy approximation theorem?
10. How fuzzy controlled machines can be created?

Long-Answer Questions

1. Discuss briefly about the semantic nets with the help of relevant examples.
2. Elaborate on the frames. Give appropriate examples.
3. Briefly explain about the slot values as objects with the help of examples.
4. Explain with the help of an example, how Bayes' theorem is useful in probabilistic reasoning.
5. Describe the role of Certainty Factor (CF) model in statistical reasoning.

6. Explain the concepts of 'Measure of Belief' and 'Measure of Disbelief' with respect to the Certainty Factor (CF) model.

4.10 FURTHER READING

Russell, Stuart J. and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*, 3rd Edition. New Jersey: Prentice Hall.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco (California): Morgan Kaufmann Publishers, Inc.

Knight Kevin, Elaine Rich and B. Nair. *Artificial Intelligence (SIE)*, 3rd Edition. New Delhi: Tata McGraw-Hill.

Sivanandam, S.N. and M. Paulraj. 2009. *Introduction to Artificial Neural Networks*. New Delhi: Vikas Publishing House Pvt. Ltd.

Rich, E. and K. Knight, *Artificial Intelligence*. New York: McGraw-Hill Book Company, 1991.

LiMin, Fu. 2003. *Neural Networks in Computer Intelligence.* New Delhi: Tata McGraw-Hill.

NOTES

UNIT 5 LEARNING AND EXPERT SYSTEMS

Structure

5.0 INTRODUCTION

In learning, Machine learning (ML) is a scientific discipline related to the design and development of algorithms. Machine learning is very important in the research for Artificial Intelligence (AI), in which unsupervised learning helps to find patterns regarding the stream of input and supervised learning consists of both classification and numerical regression. Rote Learning is consists of simply storing of computed information. A lot of AI programs significantly improve their performance with the help of rote learning. Problem Solving Experience (Analogy) is involves remembering the manner in which a problem is solved. Hence, when the same problem re-occurs, you can solve it more efficiently.

Induction algorithms form another approach to machine learning. While neural networks are highly mathematical in nature, induction approaches involve symbolic data. Induction methods, which are characterized as 'Learning by example', begin with a set of observations. They construct rules to account for the observations and try to find general patterns that can fully explain the observations. Learning automation is supported by various algorithms and programs, which are based on future attempts or past actions and are useful in learning from established failures or successes.

A neural network is a system that can resolve paradigms that linear computing cannot. Traditionally, it is used to describe a network or circuit of biological neurons. It also refers to artificial neural networks that are made up of artificial neurons or nodes.

Expert systems are widely used today to solve real-world problems in the areas of medicine, law, construction and manufacturing. A successful expert system is able to almost accurately mimic the way an expert applies his problem-solving abilities while making a recommendation or drawing a conclusion with a high degree of accuracy. Expert systems differ significantly from other computer program architectures because they separate what is known about an application, called domain knowledge, from the logic that controls how the knowledge is used, known as inference procedures. Though expert systems cannot replace the experts, they can assist those who are less knowledgeable in the subject domain by using the knowledge of higher-level experts. These systems are also known as knowledge based systems or decision support systems. This unit will discuss the working of expert systems in detail.

In this unit, you will learn about the concept of learning, rote learning, learning by taking advice, learning in problem solving, learning by induction, explanation based learning, learning automation, learning in neural networks and expert systems.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basic concept of learning
- Explain the rote learning
- Discuss the learning by taking advice
- Analyse learning in problem solving
- Define learning by induction
- Learn about the learning automation
- Illustrate the expert system
- Understand the need and justification of expert systems
- Explain the representing and using domain knowledge

5.2 CONCEPT OF LEARNING

AI has applications in all fields of human study, such as finance and economics, environmental engineering, chemistry, computer science and so on. Some of the applications of AI are listed as follows:

- **Robotics:** Although industrial robots have been expensive, robot hardware can be cheap. The limiting factor in application of robotics is not the cost of the robot hardware itself.

  What is needed is perception and intelligence to tell the robot what to do; ‘blind’ robots are limited to very well-structured tasks (like spray painting car bodies).

- **Natural Language Processing:** Natural languages are human languages such as English. Making computers understand English enables programmers to use them with little training. Getting computers to generate human or natural languages can be called natural language generation. It is much easier than natural language understanding. A text written in one language and generated in another language by means of computers can be called machine translation.

- **Planning:** Planning attempts to make an order of actions for achieving goals. Planning applications include logistics, manufacturing scheduling and planning manufacturing steps to construct a desired product. Huge amounts of money can be saved through better planning.
- **Expert Systems:** Expert systems attempt to capture the knowledge of a human and present it through a computer system. There have been many successful and economically valuable applications of expert systems. The benefits of expert systems are as follows:
    - o Reducing the skill level needed to operate complex devices.
    - o Diagnostic advice for device repair.
    - o Interpretation of complex data.
    - o 'Cloning' of scarce expertise.
    - o Capturing knowledge of expert who is about to retire.
    - o Combining knowledge of multiple experts.
    - o Intelligent training.
- **Machine Learning:** Machine learning has been a central part of AI research from the beginning. In machine learning, unsupervised learning is a class of problems in which one seeks to determine how the data is organized. Unsupervised learning is the ability to find patterns in a stream of input. Supervised learning includes both classification (determines what category something belongs in) and regression (given a set of numerical input/output examples, discovers a continuous function that would generate the outputs from the inputs).
- **Theorem Proving:** Proving mathematical theorems might seem to be mainly of academic interest. However, many practical problems can be cast in terms of theorems. A general theorem prover can therefore be widely applicable.
- **Symbolic Mathematics:** Symbolic mathematics refers to manipulation of formula, rather than arithmetic on numeric values.

  Symbolic manipulation is often used in conjunction with ordinary scientific computations. Symbolic manipulation programs are an important component of scientific and engineering workstations.
- **Game Playing:** Games are good for research because they are well formalized, small and self-contained. They are good models for competitive situations, so principles discovered in game playing programs may be applicable to practical problems.

  Apart from these, the four following categories highlight application areas where AI technology is having a strong impact on industry and everyday life.

    1. **Authorizing Financial Transactions:** Credit card providers, telephone companies, mortgage lenders, banks etc. employ AI systems to detect fraud and expenditure financial transactions, with daily transaction volumes in billions. These systems first use learning

NOTES

algorithms to construct profiles of customer usage patterns and then use the resulting profiles to detect unusual patterns and take appropriate actions (e.g., disabling the credit card). Such automated oversight of financial transactions is an important component in achieving a viable basis for electronic commerce.

2. **Configuring Hardware and Software:** Systems that diagnose and treat problems, whether illness in people or problems in hardware and software, are now in widespread use. Diagnostic systems based on artificial intelligence technology are being built into photocopiers, computer operating systems and office automation tools to reduce service calls. Standalone units are being used to monitor and control operations in factories and office buildings. Artificial intelligence based systems assist physicians in many kinds of medical diagnosis, in prescribing treatments and monitoring patient responses. Microsoft's Office Assistant, an integral part of every office application, provides users with customized help by means of decision-theoretic reasoning.
3. **Scheduling for Manufacturing:** The use of automatic scheduling for manufacturing operations in exploring as manufacturers realize that remaining competitive demands on every more efficient use of resources. This technology of AI supporting rapid rescheduling up and down the 'supply chain' to respond to changing orders, changing markets, and unexpected events—has been shown to be superior to less adaptive systems based on older technology. This same technology has proven highly effective in other commercial tasks, including job shop scheduling and assigning airport gates and railway crews. It also has proven highly effective in military settings. DARPA reported that an AI based Logistics Planning Tool, DART, pressed into service for operations Dessert shield and Dessert storm, completely repaid its three decades of investments in AI research.
4. **The Future:** AI began as an attempt to answer some of the most fundamental questions about human existence by understanding the nature of intelligence, but it has grown into a scientific and technological field affecting many aspects of commerce and society.

Even as AI technology becomes integrated into the framework of everyday life, AI researchers remain focused on the grand challenges of automating intelligence. Artificial intelligence is used in fields of medical diagnosis, stock trading, robot control, law, scientific discovery, video games and toys. It can also be used for evaluation in specific problems such as small problems in chemistry, handwriting recognition and game-playing. For this Natural Language Processing (NLP) is used which interacts between computers and human (natural) languages. Natural language processing gives AI machines the ability to read and understand the languages that human beings speak. Some straightforward applications of natural language processing include information retrieval or text mining and machine translation. The pursuit of ultimate goals of AI—design of intelligent artifacts, understanding of human intelligence, abstract understanding of intelligence (possibly super human)—continues to have practical consequences in the form of new industries, enhanced functionality for existing systems, increased productivity in general and improvements in the quality of life. But the ultimate promises of AI are still decades away and the necessary advances in knowledge and technology will require a continuous fundamental research effort.

5.2.1 Explanation Based Learning

Machine learning or Explanation-Based learning is a scientific discipline related to the design and development of algorithms. It allows computers to evolve their behaviour, depending on empirical data, such as from sensor data or databases. Machine learning research mostly concentrates on automatic learning to identify complex patterns and make appropriate decisions. Machine learning is very important in the research for Artificial Intelligence (AI), in which unsupervised learning helps to find patterns regarding the stream of input and supervised learning consists of both classification and numerical regression. Classification helps to determine what category anything belongs to after examining various examples from different categories.

Regression consists of numerical input/output examples and it tries to establish a regular function that will generate output from the given input. In reinforcement learning, an agent gets a reward for good responses and punishment for the bad ones. These good or bad responses can be analysed by using decision theory and concepts, such as utility. The mathematical analysis of machine learning algorithms and their performance is a branch of theoretical computer science and this branch is known as computational learning theory.

Human Interaction

Certain machine learning systems attempt to remove the need for human intuition in data analysis, while other systems consider a collaborative approach between humans and machines. However, human intuition cannot be completely removed until the system's designer does not specify how data needs to be represented and the processes that should be used to search for data characterization.

Algorithm Types

Machine learning algorithms are organized in taxonomy, depending on the algorithm's outcome.

- **Supervised Learning**: It makes a process that relates inputs with desired outputs, like, in a classification problem the learner approximates a function, mapping a vector into classes by looking at the input–output parts of the function.
- **Unsupervised Learning:** It marks a set of inputs, for example, clustering.
- **Semi-Supervised Learning**: It combines both the labeled and the unlabeled examples to generate a correct function or classifier.
- **Reinforcement Learning**: It understands how to behave according to an observation of the world. Each and every action has an impact on the surroundings and that gives feedback in the manner of rewards which guide the learning algorithm.

- **Transduction**: It tries to forecast new outputs based on training inputs, training outputs and test inputs.
- **Learning to Learn**: It understands its own inductive bias based on the experiences of the past.

Theory

Computational learning theory is a part of theoretical computer science that deals with the computational analysis of machine learning algorithms and their execution. Along with the performance bounds, computational learning theorists also read about the complications of the time and the feasibility of learning. In the computational learning theory, a computation is thought to be affordable if it can be accomplished in polynomial duration. Complexity results are of two types:

- **Positive Results:** Positive results are depict that a specific class's functions can be understood in polynomial time.
- **Negative Results:** Negative results are show that some classes cannot be understood in polynomial time.

Approaches

- **Association Rule Learning:** This process determines the stimulating relation among variables in large databases.
- **Decision Tree Learning:** In this method, a decision tree is used as a predictive model that draws the observations about an item to conclude about the item's target value.
- **Artificial Neural Networks (ANNs):** Also known as Neural Network (NN), it is a mathematical or computational model that tries to simulate the structure and/or functional aspects of the biological neural networks. ANNs comprise an interconnected group of artificial neurons and process the information by using a connectionist approach towards computation. Modern neural networks refer to non-linear statistical data modeling tools mostly used for modeling difficult relationships between inputs and outputs or to find patterns in data.

Genetic Programming

Genetic Programming (GP) refers to an evolutionary algorithm-based methodology which is motivated by biological evolution for searching those computer programs which carry out user-defined tasks. Genetic Algorithms (GA) are special in the way that every identity is an individual computer program.

Inductive Logic Programming

Inductive Logic Programming (ILP) refers to the use of logic programming like a uniform representation. By giving an encoding of the known background knowledge and some examples that are depicted as a logical database of facts, an ILP system deduces a hypothesized logic program. This would entail the entire positive example but none of the negative examples.

Support Vector Machines

Support Vector Machines (SVMs) comprise a group of related supervised learning methods that are used for the purpose of classification and regression. Based on training examples, every SVM training algorithm, which is marked as one of the two categories, builds a model that foresees which category the relative new example comes under.

Clustering

Clustering refers to unsupervised learning. It is a general method for statistical data analysis. Cluster analysis or clustering refers to the formation of a set of observations into subsets such that the observations in the similar cluster are identical in certain aspects.

Bayesian Networks

A probabilistic graphical model that talks about a set of random variables and their conditional independencies through a Directed Acyclic Graph (DAG) is called a Bayesian network, belief network or directed acyclic graphical model.

Reinforcement Learning

Reinforcement learning describes how an agent should take action for maximizing certain notion for long-term reward. Reinforcement learning algorithms try to search for a new policy which would map states of the world with the actions that an agent needs to initiate in those states. Reinforcement learning is different from the supervised learning problem in the manner that correct input/output pairs can never be presented, nor can the sub-optimal actions be explicitly corrected.

The biggest problem with Artificial Intelligence (AI) is that a mechanism cannot be called intelligent until and unless it does new things and adapts to new situations. This ability of adapting to new surroundings and solving new problems is a very important characteristic in all 'intelligent' entities. Computers learn artificial intelligence through failure-driven learning and exploration.

Failure-driven learning depends on program creation that will learn with the help of mistakes that have been committed and eventually search for a solution so that these mistakes do not occur again. This is identical to the manner in which human beings also learn about things.

Figure 5.1 shows a graphical representation of a program that wants to put the 'a' block on top of the 'b' block. In the beginning, the program cannot do that as the 'c' block is on top of the 'a' block. Now, the program needs to find a solution so that it can lift the 'a' block. It finds a way for moving the 'c' block off the 'a' block. Once, the 'c' block is moved, it can place the 'a' block on top of the 'b' block and its motive will be completed.

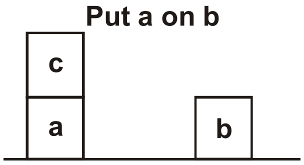


Fig. 5.1 *Graphical Representation of Failure-Driven Learning*

Learning by being told refers to the simple interaction between human and the AI student, but the interaction faces the problem of communication. As the teacher wants to teach in English, but AI does not understand English, so the communication problem occurs. A solution to it can be that the teacher puts the instructions into code. But, this is not preferable as lengthy instructions will need a lot of coding and it will even be time consuming.

Learning by exploration refers to gathering the information and not really pursuing any goal. It tries to search for interesting information so that it can store and learn from it.

Integrating the Approaches

The integrated approaches paradigm gives researchers the license to study isolated problems and search for solutions that are both verifiable and useful. A paradigm also gives researchers a mutual base to communicate with other fields, such as decision theory and economics.

Active learning approaches is less partial on a given situation, it is most likely to provide surprising results.

The knowledge-intensive approach was developed by relevant knowledge bases within the Strengths, Opportunities, Aspirations and Results (SOAR) framework. This framework provides a correct framework for representing and using the difficult information within a dynamic environment. Knowledge acquisition includes many different activities. Most of the learning activities are as follows:

- **Rote Learning**

It consists of simply storing of computed information. A lot of AI programs significantly improve their performance with the help of rote learning.

- **Problem Solving Experience (Analogy)**

It involves remembering the manner in which a problem is solved. Hence, when the same problem re-occurs, you can solve it more efficiently.

- **Learning form Examples (Induction)**

It is the manner of learning that involves stimuli without being given any explicit rules.

- **Deductive Learning**

It is done with the help of deductive inference steps. From these known facts, new facts and relationships are generally derived.

Inductive Learning

Inductive learning is a type of learning in which an agent tries to calculate or create an evaluation function. Most of the inductive learning can also be supervised learning, in which examples are created with the help of classifications.

Decision Trees

Decision tree is an inductive learning structure, where every internal node in the tree represents a test on one of those properties and the branches from the node are labeled with the possible outcomes of the test. Every leaf node is a Boolean classifier for the input instance.

Connectionist Learning

The connectionist learning approach has been taken from the human brain's model as an enormous parallel computer, in which small computational units feed simple, low-bandwidth signals to one another, and from which arises intelligence. It tries to copy this behaviour on an abstract level with *neural networks*.

Neural Networks

Neural networks are a general target representation for knowledge. These networks are encouraged by the neurons present in the brain but do not really fake neurons (Refer Figure 5.2). Artificial neural networks characteristically comprise various fewer than the approximately 10 neurons that are in the human brain, and the artificial neurons, called units, are much modest than their biological complements.

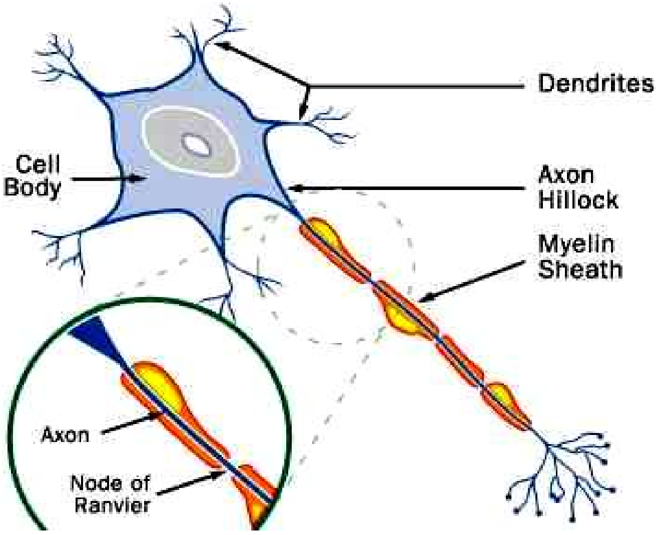


Fig. 5.2 *Basic Neuron Design*

A neural network comprises a set of *nodes* that match one node with another, and *weights* are associated with every link. Some of the nodes receive inputs through links while others receive them from the nature directly, and some of the nodes even send out the outputs out through the network. Mostly, all the nodes share identical activation function and threshold value and only the topology and weights change.

Network Structures

The two basic types of fundamental network structures are *feed-forward* network structures and *recurrent* network structures. Feed-forward network structures refer to directed acyclic graphs. Recurrent network structures consist of loops, and as a result it can represent state. All the connections in Hopfield networks are bidirectional with symmetric weights, all units have outputs of 1 or –1 and the activation function is the sign function. Feed-forward networks can be understood as flowed dense linear functions. The inputs feed into a layer of concealed units, which can feed into layers of more concealed units, which finally feed into the output layer. Each of the hidden units is a dense linear function of its inputs.

Neural networks of this kind can ensure as inputs any real numbers, and they ensure a real number as output. For reversion, it is characteristic for the output units to be a linear function of their inputs. For organization it is characteristic for the output to be a sigmoid function of its inputs. For the concealed layers, there is no argument in having their output is a linear function of their inputs as; accumulation of the additional layers gives no extra functionality. The output of each concealed unit is, therefore, a dense linear function of its inputs (Refer Figure 5.3).

Related with a network are the parameters for all of the linear functions. These parameters can be adjusted concurrently to diminish the forecast mistake on the training examples.

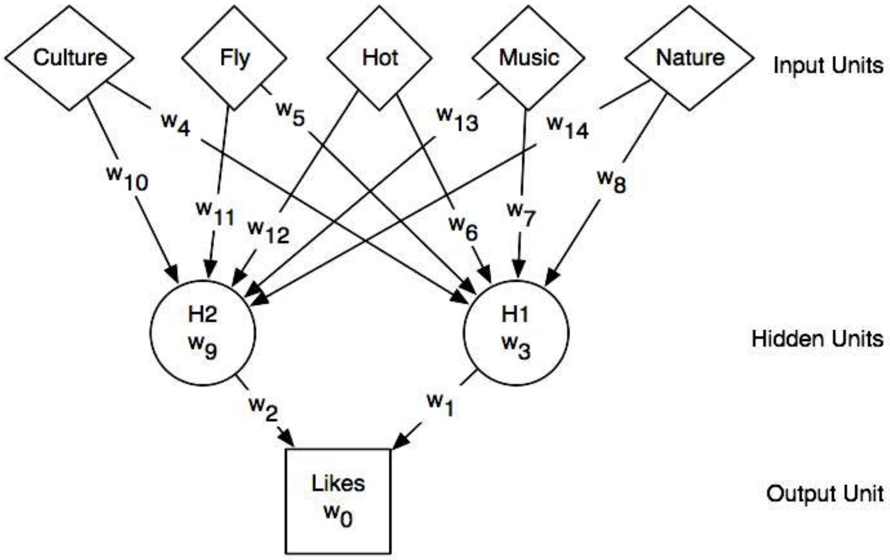


Fig. 5.3 *A Neural Network with One Hidden Layer*

The w_i are weights. The weight inside the nodes is the weight that does not depend on an input; it is the one multiplied by 1.

A major problem in building neural networks is deciding about the initial topology. Usually, c*ross-validation* techniques are used for determining when the network size is right.

Perceptrons

Perceptrons refer to single-layer, feed-forward networks that were initially studied in the 1950s. They can just learn *linearly separable* functions. Perceptrons are studied by updating the weights on their links as a response to the difference among their output value and the correct output value. The learning rule for each weight is as follows:

$$W_j \leftarrow W_j + A \times I_j \times \text{Err}$$

where, *A* is a constant called the *learning rate.*

Bayesian Learning in Belief Networks

Bayesian learning indicates a lot of presumptions about the data. Every hypothesis weighs its posterior probability whenever a prediction is made. The basic theme is that instead of having just one presumption, many should be entertained and calculated depending on their likelihoods.

Although, updating and using logic with a lot of hypotheses can be intractable, the most popular approximation is using a *most probable* hypothesis, i.e., an H_i of H that maximizes P(H_i | *D)*, where *D* is the data. This is often termed as the *Maximum Posteriori* (MAP) hypothesis H_{MAP}:

$$\mathbf{P}(X \mid D) \sim = \mathbf{P}(X \mid H_{MAP}) \text{ x } \mathbf{P}(H_{MAP} \mid D)$$

For finding H_{MAP}, you apply Baye's rule:

$$\mathbf{P}(H_i \mid D) = [\mathbf{P}(D \mid H_i) \text{ x } \mathbf{P}(H_i)] \,/\, \mathbf{P}(D)$$

Since, *P(D)* is fixed around the hypotheses, you just have to maximize the numerator. The first term depicts the probability which this data set might have, if H_i is the model. The second term refers to the prior probability that was assigned to the model.

Belief Network Learning Problems

Four belief networks are generally talked about, depending on the network's structure, whether it is unknown or known and whether the network variables are hidden or observable.

- **Known Structure (Fully Observable):** In it, the conditional probability tables can only be learned. These tables can be calculated by using the sample data set's statistics of the sample data set.
- **Unknown Structure (Fully Observable)**: In it, the major problem is reconstructing the network topology. The problem can be called as a search-through structure space and the fitting data for each structure reduces the fixed-structure problem.
- **Known Structure (Hidden Variables):** This is analogous to neural network learning.
- **Unknown Structure (Hidden Variables):** Whenever certain variables cannot be observed, it becomes problematic to apply the prior techniques for recovering structure of these variables, but these structures need averaging over all the probable values of the unknown variables.

Reinforcement Learning

Reinforcement learning occurs at places where the agent cannot compare the actions's results directly. Reinforcement learning needs to find a successful function by using such rewards. This form of learning is difficult than supervised learning because in it, the agent does not what the right action is and just knows whether it is doing well or poorly.

Following are the two basic types of information that an agent tries to learn:

- **Utility Function:** In it, the agent understands the utility of being in many states, and then chooses his actions to maximize the expected utility of their outcomes.

- **Action-Value:** In it, the agent learns an action-value function by giving the expected utility of performing an action in a given state. It is called *Q-learning* and is a *model-free* approach.

Passive Learning in a Known Environment

An agent learns slowly while observing a group of *training sequences* that consist of a set of state transitions followed by a reward. The main aim is using the reward information to learn the expected utility of every non-terminal state. A necessary simplifying assumption is that the sequence's utility refers to the sum of the rewards that have accumulated in the states of the sequence, i.e., the utility function is *additive.*

Temporal Difference Learning

Temporal difference learning assesses the difference in the utility values between successive states for adjusting them from one epoch to another. The key idea is using the observed transitions for adjusting the values of the observed states which would lead them to agree with the Approximate Dynamic Programming (ADP) constraint equations. Basically, this means updating the utility of the state *i* so that it agrees better with its successor *j*. This is accomplished with the Temporal Difference (TD) equation:

$$U(i) \leftarrow U(i) + a(R(i) + U(j) - U(i))$$

Where, *a* is the learning rate parameter.

Temporal difference learning refers to approximating the ADP constraint equations from every probable state without solving them. The general idea is defining those situations that keep a hold over local transition. It will lead U(i) to converge to the correct value if the learning rate parameter becomes less with the number of times a state has been visited.

Passive Learning in an Unknown Environment

As in reality, neither temporal difference learning nor Learning Management System (LMS) uses the model M of state transition probabilities. They might operate unchanged in the unknown environment. However, the ADP approach may update its proximate model of the unknown environment after every step and this model is used for revising the utility estimates.

The most important difference between TD and ADP is that TD can adjust a state to agree with the observed successor, whereas ADP makes the state agree to all the successors that might occur, based on their probabilities.

Active Learning in an Unknown Environment

The difference between active and passive agents is that passive agents learn a fixed policy, while active agents have to decide what action should be taken and how it will affect its rewards. For representing an active agent, the environment model M is extended to give the probability of a transition from state i to state j, given action *a*. Utility is the reward of the state plus the maximum utility that is expected, based upon the agent's action:

$$U(i) = R(i) + \max_a \times \text{SUMj MaijU(j)}$$

Learning Action-Value Functions

An action-value function refers to an expected utility for performing an action in a specific state. If Q(a, i) is the value of doing action a in state i, then

$$U(i) = \max_a Q(a, i)$$

The equations for Q-learning are identical to those for state-based learning agents. The only difference being that Q-learning agents do not need models of the world. The equilibrium equation that can be used directly is as follows:

$$Q(a, i) = R(i) + \text{SUMj Maij} \max_{a'} Q(a', j)$$

The temporal difference version does not need for a model to be learned; its update equation is as follows:

$$Q(a, i) = Q(a, i) + a(R(i) + \max_{a'} Q(a', j) - Q(a, i))$$

Learning with Knowledge

Various logical constraints are placed upon different types of knowledge-based learning and you can classify them more specifically.

- *Inductive learning* can be characterized with the help of following entailment constraint:

  Hypothesis ^ Descriptions = Classifications

- *Inductive learning* refers to generating classifications with the help of hypothesis and description of problem instances.

5.3 LEARNING BY INDUCTION

Induction algorithms form another approach to machine learning. While neural networks are highly mathematical in nature, induction approaches involve symbolic data. Induction methods, which are characterized as 'learning by example', begin with a set of observations. They construct rules to account for the observations and try to find general patterns that can fully explain the observations.

A large set of data comprising several input variables and one decision variable is given to the system, which recursively partitions the data set based on the variables that best distinguish between the data elements and construct a decision tree. It tries to partition the data in such a way that each partition contains the same valued data for a decision variable. It does this by selecting the input variables that divide the data set into homogeneous partitions, the best way. For example, consider Table 5.1, which contains the data set related to decisions made on loan applications.

Table 5.1 *Data Set for Making-Decisions on Loan Applications*

	Salary	Credit History	Current Assets	Loan Decision
(a)	High	Poor	High	Accept
(b)	High	Poor	Low	Reject
(c)	Low	Poor	Low	Reject
(d)	Low	Good	Low	Accept
(e)	Low	Good	High	Accept
(f)	High	Good	Low	Accept

Table 5.2 shows the rules that an induction algorithm would infer to explain the data presented in Table 5.1.

Table 5.2 *Rules Inferred by Induction Algorithm*

Rules Inferred
If the credit history is good, then accept the loan application
If the credit history is poor and current assets are high, then accept the loan application
If the credit history is poor and current assets are low, then reject the loan application

As you can understand by this example, an induction algorithm, by inducing rules that identify the general patterns in data, enables filtration of irrelevant or unnecessary attributes. In the given example, it pruned out the 'salary' which was irrelevant in terms of explaining the loan decision of the data set. Induction algorithms are often used for data-mining applications, such as marketing problems that help companies decide on the best market strategies for new product lines.

Inductive Learning

Inductive learning is essentially 'learning by example' as it involves drawing conclusions about previously unseen examples once the learning process is completed. Quinlan's decision trees, connectionism and decision list techniques are the most commonly used techniques for modelling the inductive learning process.

This process is however regarded as an imperfect technique. As Chalmers points out, 'An inductive inference with true premises [can] lead to false conclusions'. It is possible that the example set does not represent the true population completely but also shows inappropriate rules that are derived, which apply only to the example set.

For example, consider the set of bit-strings given in Table 5.3. Each bit string, herein, is noted as either a positive or negative example of some concept. The task is to infer a rule from this data to account for the given classification.

Table 5.3 *Bit Strings*

-	**1000101**	**-**	**1110100**	**+**	**0101**
+	**1111**	**+**	**10010**	**+**	**1100110**
-	**100**	**+**	**111111**	**-**	**00010**
-	**1**	**-**	**1101**	**+**	**101101**
+	**1010011**	**-**	**11111**	**-**	**001011**

A rule which can be induced from this data is that the strings with an even number of 1's are '+' and those with an odd number of 1's are '–'. This rule would allow the classification of previously unseen strings (that is, 1001 is '+').

Check Your Progress

1. What is the difference between association rule learning and decision tree learning?
2. Define reinforcement learning.
3. What is rote learning?
4. How will you define the learning in problem solving?
5. Give the techniques for modelling the inductive learning process.

5.4 LEARNING AUTOMATION

Artificial intelligence is the ability to think, imagine and create, memorize, understand, recognize patterns, make choices, adapt to change and learn from experience. This creates a non-organic machine-based entity that has all the given abilities of natural organic intelligence. That is why it is known as artificial intelligence. Programmers delve deeper into natural intelligence first and then they try to understand how cognition, comprehension and decision-making take place in the human mind.

The learning process approaches AI with the help of theorems and simulations. It works with the mapping function to transform the new problem from the given domain to the target domain. The learning process includes two mechanisms with reference to artificial intelligence: learning automation and environments. Both refer to the learning cycle, which starts with the stimulus-generating process from the environment. The automation receives stimulus to respond to the environment. The main function of learning automation is to control the response to the environment and offer a new stimulus, if requested. Stimulus here means current input. The learner then automatically adjusts the parameters. These parameters are also known as adapted parameters and are based on the current input and last response of the automation.

The other form of learning automation allows the learning AI programs as well as making good decisions. The AI program maintains software, which stores the results of all attempts, including all feedback received. The learning automation is supported by various algorithms and programs, which are based on future attempts or past actions and are useful in learning from established failures or successes. Figure 5.4 shows the layout of learning automation. The delay unit ensures that current stimulus and last response make the entry for learner concurrently and adapt parameters.

NOTES

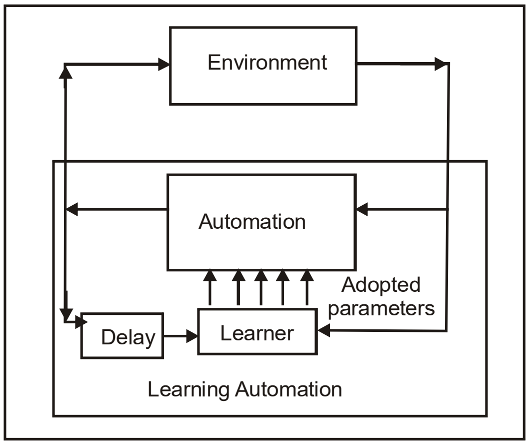


Fig. 5.4 *Layout of Learning Automation*

The following learning processes are implemented in learning automation and artificial intelligence:

(i) Reinforcement Learning

Reinforcement Learning (RL) is the process by which an agent improves his behavior in an environment via experience. Most RL learning researches have been confined to single-agent settings or multi-agent settings where agents have either positively correlated payoffs or totally negative correlated payoffs (zero sum games). This learning process is one of the most active research areas in AI.

RL refers to training by rewards and punishments. To accumulate a lot of rewards, the learning system must prefer the best experienced actions. This learning is widely and frequently applied to solve a diverse set of learning tasks, from board games to robot behaviours. In some of them, results have been very successful but some tasks present several characteristics that make the application of reinforcement learning harder to define. In the field of multi-robot learning, there are two important problems.

- The first one supports credit assignment, which refers to the question of how to define the reinforcement signal to each robot belonging to a cooperative team, depending on the results achieved by the whole team.
- The second one is working with large domains, where the amount of data can be large and different in each moment of a learning step.

The application of RL can be taken to move towards the desired place by the chess player. A master chess player makes a move. The choice can be taken either by planning or by immediate action. In planning, anticipating count replies settings are done whereas in immediate intuitive judgments of the desired and particular positions and movements can be done.

(ii) Temporal Difference Learning

The temporal difference learning technique is proposed by Richard S. Sutton and is used as the deciding factor for long-term future cost as a function of the current state. Basically, it is the field of RL. This learning approach is based on the Monte Carlo methods and dynamic programming.

Temporal difference learning methods can learn directly from raw experience without a model of the environment's dynamics. Examples are learning to play games, robot control, elevator control, network routing and animal learning. One of the applications has been developed as personalization travel support system and it provides travel information as per the user's interests. It applies reinforcement learning to analyse and learn customer behaviour and list the products that the customers wish to buy. If the system selects the right item that the customer wishes to buy, then it is given a reward by assigning a particular value for the state that a user selects to perform. On the other hand, if the system selects an item which the user does not wish to buy then it is penalized. In this way, the system learns the personal interests. By this process, the system acquires the knowledge of user behavior and interest which helps it to decide which information should be given to a particular user. It promotes customer satisfaction and increases the success rate of product promotion.

A view commonly adopted in the original setting is that the algorithm involves 'looking back in time and correcting previous predictions'. In this context, the eligibility vector keeps track of how the parameter vector should be adjusted in order to appropriately modify prior predictions when a temporal-difference is observed. To avoid the constraint equation as mentioned in the Negative-Index Metamaterial (NIM) game theory, the following equation is required for temporal difference learning:

$$U(i) \rightarrow U(i) + \alpha[R(i) + U(j) - U(i)]$$

where, α is the learning rate, which lies in the interval 0<<1. U(i) can be updated if transition is allowed to state *j* from state *i*, where U(j) >> U(i). This kind of learning is known as temporal difference learning. The statement U(j) >> U(i) suggest that U(j) keeps too large values. It also suggests that instantaneous value seems to be occasionally large.

(iii) Active Learning

Active learning is central to intelligence and hence refers to machine learning. The system is capable of acquiring and integrating the knowledge automatically. The capability to learn from experience, training, analytical observation and other means results in a system that can continuously self-improve and thereby exhibit efficiency and effectiveness (Refer Figure 5.5).

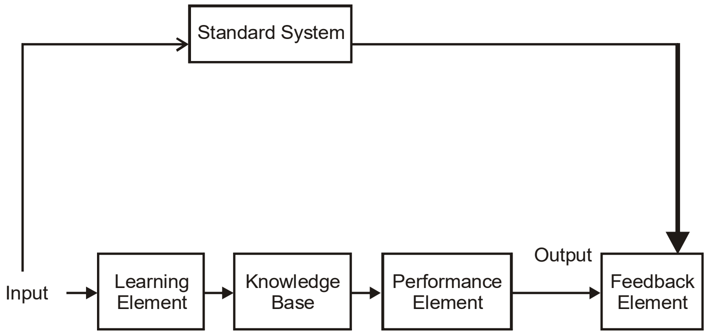


Fig. 5.5 *Learning System Model*

Figure 5.5 shows that the learning system model in which learning elements are used as input and the output appears as feedback element. The stimulus that has been input is passed through the knowledge base and performance elements. The role of performance elements is to push up the standard system that produces expected and desired output. Learning elements in Figure 5.5 receive and process the input obtained from the user by various means, such as magazines, journals, and so on.

Knowledge base refers to the database that contains some basic knowledge. Performance elements use the updated knowledge base to perform some tasks or solve problems and produce the corresponding output. The feedback element receives two inputs, one from the learning element and the other from the standard system. Basically, it identifies the differences between the two inputs. The feedback is used to determine what should be done in order to produce the correct output. The standard system is the heart of active learning in which a trained person or a computer program is able to produce the correct output. For a passive learner, M is considered to be a constant matrix but for an active learner, it is considered to be a variable matrix. The equation taken for an active learner is as follows:

$$\mathrm{U(i) = R(i) + Max}_a\ \Sigma_{\forall j}\ \mathrm{M}_{ij}^{n}\ \mathrm{U(j)}$$

where, denotes the probability of reaching state *j* with an action.

This action can be stated as ‘a performed at state *I*’. The role of the agent is to choose the action for. Therefore, probability results suggest that U(i) would be maximum.

(iv) Q-Learning

Q-Learning is a form of learning automation and can be used online. This is why it is very suitable for repeated games against an unknown opponent. Q-learning algorithms work to estimate the values of state-action pairs. The value Q(s, a) is defined to be the expected discounted sum of future payoffs obtained by taking action a from state s and following an optimal policy, thereafter. Once these values have been learned, the optimal action from any state is the one with the highest Q-value. The following algorithm is defined for estimating the Q-values:

- **Step 1:** From the current state (*s*), select an action (*a*). This will cause a receipt of an immediate payoff (*r*) and arrival at a next state (*s*′).
- **Step 2:** Update Q(s, a) by the following formula:

$$\mathrm{Q(s, a) = x[r + y max Q(s', b)–Q(s, a)]}$$

where, x is the learning rate and $0 < y < 1$ is the discount factor

- **Step 3:** Go back to Step 1.

This algorithm is guaranteed to converge to the correct Q-values with the probability if the environment is stationary and depends on the current state and the action taken in it.

In Q-learning, q-values are used. Q(a,i) is employed to denote the Q-value. Action (a) is worked with state (i). Utility and Q-values can be expressed by the following equation:

$$U(i) = \max_a Q(a, i).$$

You can construct another constraint equation at this stage. This equation holds at equilibrium if the Q-values are correct. The corresponding temporal difference updates the equation stated as follows:

$$Q(a, i) = R(i) + \Sigma M_{ij}^{n} .\max_{n} . Q(a', i)$$

If the given problem is not solved, then the following equation is evolved to solve the problem:

$$Q(a, i) \leftarrow Q(a, i) + \alpha[R(i) + \max(a', j) - Q(a, i)]$$

The given equation can be evaluated by every transition from state i to state j. Q-learning continues the given equation until the Q-value at each state i is generated with a steady value.

Inductive Logic Programming

Inductive Logic Programming (ILP) is formed at the intersection of machine learning and logic programming. ILP systems develop predicate descriptions from examples and background knowledge. The examples, background knowledge and final descriptions are all described as logic programs. It is used in refinement, least general generalization, inverse resolution and most specific corrections. Presently, successful application areas for ILP systems include the learning of structure-activity rules for drug design, finite-element mesh analysis design rules, primary-secondary prediction of protein structure and fault diagnosis rules for satellites.

ILP employs the inductive method in learning the automation process. It is done through the process of inverse resolution. It is also known as constructive induction because ILP predicts new solutions in learning automation. You will look at the resolution theorem that predicts logic. Let there be two clauses C_1 and C_2, which are expressed in the following way:

$$C_1\text{=Female(X)}\leftarrow\text{Girl(X)}$$

And, **C_2=Girl (Richa)**

Therefore, C1 can be expressed in the following way:

$$C_1 = \neg \text{ Girl (X)} \vee \text{Female (X)}$$

Let literal be $L_1 = \neg$ Girl (X) and $L_2 = \neg$ Girl (Richa).

Let unify the substitution θ = {Richa/X}. You will get the following equation:

$L_1\theta = \neg L_2\theta = \neg$ Girl (Richa).

After solving C1 and C2, the resolvent C is obtained and expressed in the following way:

C=Female (Richa).

Therefore, it is summed up that C is the union of $(C_1 - \{L_1\})\ \theta$ =Female (Richa) and $C_2 - \{L_2\})\ \theta$ =Female (Richa) and $C_2 - \{L_2\})\ \theta = \Phi$.

Application of Learning Automation: The NIM Game

The principle of learning automation is applied to the problems in the real world, for example, the Negative-Index Metamaterial (NIM) game. In this game theory, three sets of tokens are taken on aboard as shown in Figure 5.6.

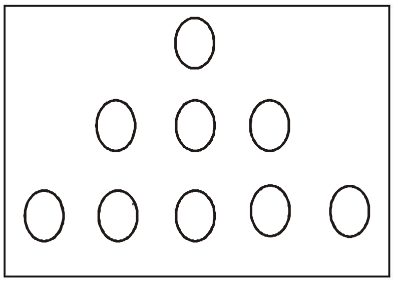


Fig. 5.6 *Token Setting in NIM Game*

Two players are required for this game. Each player has to remove one token and is also not able to access the token from more than one row. The player who loses the last token is the loser and the opponent is considered the winner. Let us consider the total calculation to compute the utility value of being the state. Suppose the utility value is high, let us say 1 but there must be utility value in other states as well. With the known starting, the computation is started with utility values. Let us assume that agent reaches the goal S_7 from S_1 via state S_2. Now, it is to be found how many tines S_2 is visited. If assumption is taken for 100 experiments, S_2 is visited 5 times and utility state can be calculated as 5/100 = 0.05. Therefore, agent moves from S_1 and S_2 or S_6 but not via S_5. It returns probability result as 0.5. If it is in S_5, it can move to S_2, S_4, S_6 with a probability result of 0.25. It is said that key of learning automation is used to update the utility values. AI uses adaptive dynamic programming and then utility, which is denoted by U (i) and is computed with state *i* by using the following expression:

$$U(i) = R(i) + \Sigma_{\forall j} M_{ij} U(j)$$

where, R(i) represents the reward of being state i, M_{ij} represents the probability of transition from state i to state j.

The given equation is said to be constraint equation. In practicality, several piles of sticks are taken in the NIM game. An example can be taken as a set of piles. The configuration of the piles is implemented by a monotone sequence of integers, for example (1, 3, 5, 7) or (2, 2, 3, 9, 110). In this example, a player may remove (in one turn) any number of sticks from one pile of as per choice. Thus, (1, 3, 5, 7) would become (1, 1, 3, 5), if the player were to remove 6 sticks from the last pile. The player who takes the last stick loses. After going through this example, now you can get the rules of the NIM game. The NIM game (1, 2, 2) can be presented by Figure 5.7.

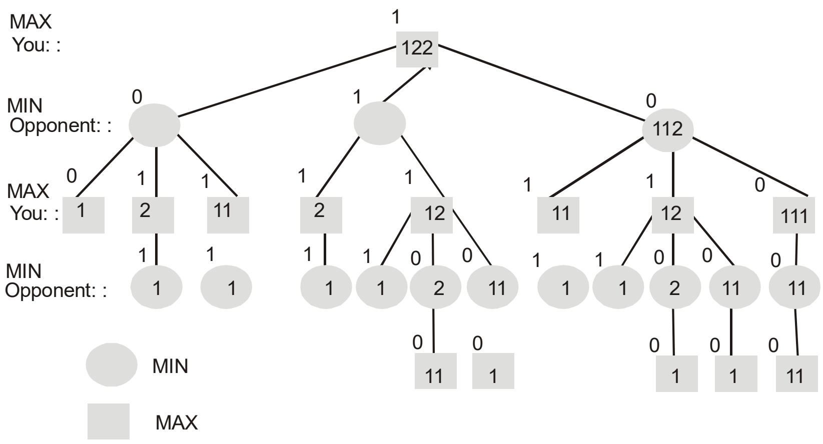


Fig. 5.7 *Game Tree for (1, 2, 2) NIM*

As shown in Figure 5.7, the number in the root consists of three sets, 1, 2, 2. Suppose you are the player who makes the first move. You may take one or two sticks. After moving from one position to the other, it is your opponent's turn and the numbers in the nodes represent the sticks on the left. Then the opponent moves one or two sticks and the status is shown in the next nodes and so on until there is one stick left. So, this is the total game scenario for the NIM tree.

5.5 LEARNING IN NEURAL NETWORKS

A neural network is a system that can resolve paradigms that linear computing cannot. Traditionally, it is used to describe a network or circuit of biological neurons. It also refers to artificial neural networks that are made up of artificial neurons or nodes. Therefore, the use of the term can be classified into the following:

- **Biological Neural Networks:** They are composed of real biological neurons that are connected in the peripheral or the central nervous system. Neuroscience identifies them as groups of neurons that fulfill a specific physiological function in laboratory analysis.
- **Artificial Neural Networks:** These networks are made up of interconnecting artificial neurons that are programming constructs which attempt to imitate the properties of biological neurons. These neural networks may be used either to understand biological neural networks or to solve problems related to artificial intelligence.

Artificial neural networks have been applied to the realm of Artificial Intelligence (AI) in matters, such as speech recognition, image analysis and adaptive control, construction of software or autonomous robots. A majority of the networks used for AI are based on statistical estimation, optimization and control theory. Inspite of the advancements in computing, there are some functions that a program made for a common microprocessor cannot fulfill. In such cases, software implementation of a neural network is made.

The advantages of a neural network are as follows:

- In case of failure of any element of the neural network, it can carry on working due to their parallel nature.

- There is no requirement for it to be reprogrammed.
- Any application can implement it.

Despite its advantages, it has a number of limitations also, which are as follows:

- The network needs to be trained in order to operate.
- Its architecture is different from that of microprocessors and, therefore, needs to be emulated.
- Large neural networks need high processing time.

Neuroscience is currently researching the question surrounding the amount of complexity and the characteristics that individual neural elements should have to reproduce a thing that resembles human intelligence.

Computers have evolved from von Neumann architecture, which was based on sequential processing and execution of the explicit instructions. Conversely, the origins of neural networks have been seen to be based on the attempts to model information processing in biological systems. This relies on parallel processing as well as implicit instructions to a large extent. These instructions are based on the recognition of sensory input patterns from external sources. Neural coding is related with how sensory and other information is represented in the brain by neurons. The major objective of studying neural coding is to determine the relationship between the stimulus and the individual. It also tries to gather neuronal responses and understand the link between electrical activities of the neurons. It is believed that neurons can encode both digital and analog information.

5.6 EXPERT SYSTEMS

Feigenbaum, one of the earliest developers of expert systems, defines an expert system as 'An intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution'. As shown in Figure 5.8, a typical expert system has the following components:

1. **Input/Output interface:** The input/output interface enables the users to communicate with the system using selection menus or a restricted language which is close to the natural language. This requires a system to have special prompts or specialized vocabulary including the terminology of the given domain of expertise.

   For example, MYCIN can recognize many medical terms in addition to various common words needed to communicate. For this purpose, MYCIN has vocabulary of some 2000 words.

2. **Explanation module:** When a user requests for an explanation of the reasoning process by way of a how or why query, the explanation module provides him the answer. This brings in transparency in the reasoning process and enables the user to decide whether he agrees or disagrees with the reasoning steps presented. If he does not, then the same can be changed using the editor.

NOTES

3. **Editor:** The editor is used by developers to create new rules for addition to the knowledge base, to delete outdated rules and/or to modify existing rules in some way. Some of the more sophisticated expert systems' editors also enable the users to perform consistency tests for newly created rules, to add missing conditions to rules and/or to reformat newly created rules.

   TEIRESUIS (Davis, 1982) is an example of an intelligent editor developed to assist users in building a knowledge base, directly, without the need of an intermediary knowledge engineer.

4. **Inference engine:** User input queries and responses to questions are accepted by the inference engine through the I/O interface. The inference engine then uses this dynamic information with the static knowledge stored in the knowledge base to arrive at inferences.
5. **Working memory:** The execution of rules may result in placement of some new facts in working memory, a request for additional information from the user or simply stopping the search process. When appropriate knowledge is stored in the knowledge base and all required parameters values are provided by the user, conclusions are found and reported to the user. The chaining continues as long as new matches can be found between clauses in the working memory and rules in the knowledge base. When no more new rules can be placed in the conflict sets, the process is stopped.
6. **Knowledge base:** The knowledge base contains facts and rules about some specialized knowledge domain
7. **Learning module:** This module uses learning algorithms to learn from usages and experience, saved in case history files. These algorithms themselves determine to a large extent how successful a learning system will be.

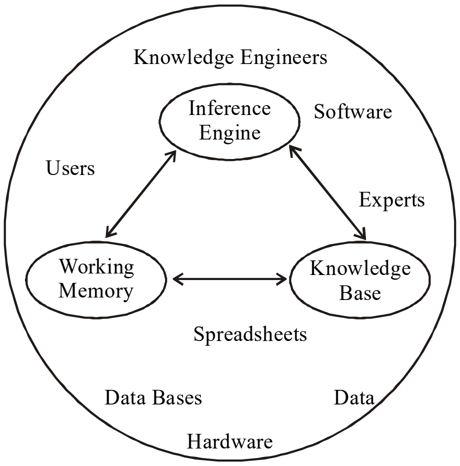


Fig. 5.8 *Simple Expert System*

Expert Systems vs. Conventional Computer Programs

Expert systems differ fundamentally from conventional computer programming systems as they treat the knowledge and inference procedures, separately. They also represent a more powerful implementation of knowledge and are able to give the end user, explanatory information on different operations or paths. Table 5.3

shows the significant differences between expert systems and conventional computer programs.

NOTES

Table 5.4 *General Distinction between Expert Systems and Conventional Computer Programs*

Expert system	**Conventional Program**
Makes decisions	Calculates results.
Based on reasoning	Based on algorithms
Conducive to change	More difficult to change
Can handle uncertainty	Can not handle uncertainty
Can work with partial information, incon partial beliefs	Requires complete information
Can provide explanations of results	Gives results without explanation
Symbolic reasoning	Numerical calculations
Primarily declarative	Primarily procedural
Control and knowledge separated	Control and knowledge interlaced

Until the mid 1980's, expert systems were primarily developed using the Lisp and Prolog artificial intelligence languages. However, since these languages required long development time of about ten years, their usage has eventually decreased to a large extent. The systems developed now generally make use of expert system shell programs.

5.6.1 Need and Justification of Expert Systems

Nowadays, expert systems are applied to diverse fields. The need for these systems is rising mainly due to the following reasons:

1. Human beings get tired from physical or mental workload but expert systems are diligent.
2. Human beings can forget crucial details of a problem, but expert systems are programmed to take care of the minutest detail.
4. Human beings may sometimes be inconsistent or biased in their decisions, but expert systems always follow logic.
5. Human beings have limited working memory and are therefore unable to comprehend large amounts of data quickly, but expert systems can store, manipulate and retrieve large amount of data in seconds.

The various advantages, which justify the huge costs associated with experts systems, are as follows:

1. Expert systems reproduce the knowledge and skills possessed by experts. This reproduction enables wide distribution of the expertise, making it available at a reasonable cost.
2. Expert systems are always consistent in their problem-solving abilities, providing uniform answer at all times. There are no emotional or health considerations that can vary their performance.

3. Expert systems provide (almost) complete accessibility. They work 24 hours all days including weekends and holidays. They are never tired, nor do they, ever take rest.
4. Expert systems also help in preserving expertise in situations where the turnover of employees or experts is very high.
5. Expert systems are capable of solving problems even where complete or exact data do not exist. This is an important feature because complete and accurate information on a problem is rarely available in the real world.

The applications of expert systems can be categorized into the following seven major classes:

1. **Diagnosis and troubleshooting devices:** Expert systems can be used to deduce faults and suggest corrective actions for malfunctioning devices or processes.
2. **Planning and scheduling:** Expert systems are used to set goals and determine a set of actions to achieve those goals. Such systems are widely used for airline scheduling of flights, manufacturing job-shop scheduling and manufacturing process planning.
3. **Configuration of manufactured objects from subassemblies:** One of the most important expert system applications includes configuration, whereby a solution to a problem is synthesized from a given set of elements related by a set of constraints. The configuration technique is used in different industries like modular home building, manufacturing and complex engineering design and manufacturing.
4. **Financial decision making:** Expert system techniques are widely used in the financial services industry. These programs assist the bankers in determining whether to make loans to businesses and individuals. Insurance companies also use these systems to assess the risk presented by the customers and determine a price for the insurance. Expert systems are used in foreign exchange trading.
5. **Knowledge publishing:** The primary function of expert systems used in this area is to deliver knowledge that is relevant to the user’s problem. The two most widely distributed expert systems which are used for knowledge publishing are as follows: One is an advisor which counsels a user on appropriate grammatical usage in a text; the second one is a tax advisor that accompanies a tax preparation program and advises the user on tax strategy, tactics and individual tax policy.
6. **Process monitoring and control:** Expert systems can also be used to analyze real-time data from physical devices and notice anomalies, predict trends and control optimality and failure correction. These systems can be found in the steel making and oil refining industries.
7. **Design and manufacturing:** These systems assist in the design of physical devices and processes, starting from high-level conceptual design of abstract entities to factory floor configuration of manufacturing processes.

5.6.2 Stages of Expert Systems

As shown in Figure 5.9, the development of expert systems, generally, involves the following stages:

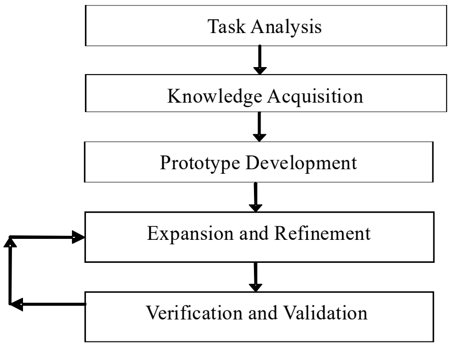


Fig. 5.9 *Stages of an Expert System*

1. **Task analysis:** The first stage of developing an expert system includes identification and analysis of the problem to be solved by the knowledge engineers.
2. **Knowledge acquisition:** The second stage involves acquiring and organizing the knowledge needed to develop an expert system. The goal of knowledge acquisition and representation is the transfer and transformation of problem-solving and decision-making expertise from a knowledge source to a form useful for developing an expert system.
3. **Prototype development:** In this stage, knowledge expertise is transformed into a computer program. As the overall system is developed in increments, prototypes are developed for different segments or modules of the system. Only the most critical factors and most basic relationships are selected while developing prototypes.
4. **Expansion and refinement:** In this stage, the expert adds more knowledge expertise from interviews, field observation and research publications. The prototype is reviewed repeatedly and rapidly until a sufficiently satisfactory prototype is achieved.
5. **Verification and validation:** In this stage, the performance of the systems is evaluated. This involves testing the system in terms of effectiveness, accuracy, performance, ease of use, adaptability, adequacy, reliability and credibility. The system is also compared to the expert's prediction of the final results. This is known as validation of the system.

Expert System Architecture

The general architecture of an expert system involves two principal components; a problem dependent set of data declarations called the knowledge base or rule base and a problem independent program which is called the inference engine. The two main categories of expert system architectures are production and non-production system architectures.

Production system architecture

One of the most common examples of the system architecture of expert system is production system. In this type of system, knowledge is represented in the form of IF-THEN-ELSE production rules. For example, IF antecedent, THEN take the consequent. The following example is taken from the knowledge base of one of the expert systems available for marketing analysis.

If: The person has good communication and written communication.

Then: The person will be considered as having ability to work as a teacher.

Each production rule in such a system represents a single piece of knowledge and sets of related production rules are used to achieve a goal. Expert systems of this type conducting a session where the systems attempt to find the best goal using information supplied by the user. The sequence of events comprises a question and answer session. The two main methods of reasoning used in this architecture are as follows:

1. **Forward chaining:** This method involves checking the condition part of a rule to determine whether it is true or false. If the condition is true, then the action part of the rule is also true. This procedure continues until a solution is found or a dead-end is reached. Forward chaining is commonly referred to as data-driven reasoning
2. **Backward chaining:** This is the reverse of forward chaining. It is used to backtrack from a goal to the paths that lead to the goal. It is very useful when all outcomes are known and the number of possible outcomes is not large. In this case, a goal is specified and the expert system tries to determine what conditions are needed to arrive at the specified goal. Backward chaining is thus also called goal-driven.

Non-production system architecture

The non production system architecture of certain expert systems do not have rule representation scheme. These systems employ more structure representation schemes like frames, decision trees or specialized networks like neural networks. Some of these architectures are discussed below.

Frame architecture

Frames are structured sets of closely related knowledge, which may include object's or concept's names, main attributes of objects, their corresponding values and possibly some attached procedures. These values are stored in specified slots of the frame and individual frames are usually linked together.

Decision tree architecture

Expert system may also store information in the form of a decision tree, that is, in a top to bottom manner. The values of attributes of an object determine a path to a leaf node in the tree which contains the objects identification. Each object attribute corresponds to a non terminal node in the tree and each branch of the decision tree corresponds to a set of values. New nodes and branches can be added to the tree when additional attributes are needed to further discriminate among new objects.

Black board system architecture

Black board architecture is a special type of knowledge based system which uses a form of opportunistic reasoning. H. Penny Nii (1986) has aptly described the blackboard problem solving strategy through the following analogy.

'Imagine a room with a large black board on which a group of experts are piecing together a jigsaw puzzle. Each of the experts has some special knowledge about solving puzzles like border expert, shapes experts, colour expert etc. Each member examines his or her pieces and decides if they will fit into the partially completed puzzle. Those members having appropriate pieces go up to the black board and update the evolving solution. The whole puzzle can be solved in complete silence with no direct communication among members of the group. Each person is self activating, knowing when to contribute to the solution. The solution evolves in this incremental way with expert contributing dynamically on an opportunistic basis, that is, as the opportunity to contribute to the solution arises.

The objects in the black board are hierarchically organized into levels which facilitate analysis and solution. Information from one level serves as input to a set of knowledge sources. The sources modify the knowledge and place it on the same or different levels.'

Black boards system is applied on WEARSAY family of projects, which are speech understanding systems developed to analyse complex scenes and model the human cognitive processes.

Analogical reasoning architecture

Expert systems based on analogical architectures solve problems by finding similar problems and their solutions and applying the known solution to the new problem, possibly with some kind of modification.

These architectures require a large knowledge base having numerous problem solutions. Previously encountered situations are stored as units in memory and are content-indexed for rapid retrieval.

5.6.3 Representing and Using Domain Knowledge

An expert system requires a knowledge base in the domain in which it is developed to solve the problems. This domain knowledge base must be such that the expert system is able to use it efficiently. The most commonly used representation of the knowledge base in the expert systems is done by defining a set of production rules. These production rules are usually united with a frame system, which provides a definition for the objects that occur in the rule. Different expert systems operate on the rules in different ways. For example, the following code shows a rule in English, which an expert system DEC VAX uses in a different version:

```
if: the most current and active context is distributing
mass-bus devices, and

there is a single-port disk drive, which has not been
assigned to a mass-bus, and

there are no unassigned dual-port disk drives, and

the number of devices that each mass-bus should support
is known, and
```

```
there is a mass-bus that has been assigned to at least one
disk drive and that should support the additional disk
drives, and
the type of cable needed to connect the disk drive to the
previous device on the mass-bus is known
then: assign the disk drive to the mass-bus.
```

The above program, called R1, has a knowledge domain that contains a set of actions to be taken for each circumstance. Also, it does not need to consider all the possible alternatives as it is responsible for doing design tasks and hence, does not require probabilistic information. Similarly, every expert system, designed for carrying out distinct tasks, has its own set of knowledge domain. These systems also make the use of the reasoning mechanism. Reasoning mechanism is required in order to apply their knowledge to a given problem. Since these systems are rule-based systems, they make the use of forward chaining, backward chaining or mixed chaining algorithms for reasoning.

5.6.4 Functioning of MYCIN and Rule Induction (RI)

MYCIN was an early backward chaining expert system that applies artificial intelligence. It is used to identify bacteria causing severe infections, such as bacteremia and meningitis. It recommend antibiotics, with the dosage adjusted according to patient's body weight. This system was also used for the diagnosis of blood clotting diseases. It was developed over five or six years in the early 1970s at Stanford University and written in Lisp as the doctoral dissertation of Edward Shortliffe under the direction of Bruce G. Buchanan, Stanley N. Cohen and others.

MYCIN operated using a fairly simple inference engine and a knowledge base of ~600 rules. It would query the physician running the program via a long series of simple yes/no or textual questions. At the end, it provided a list of possible culprit bacteria ranked from high to low based on the probability of each diagnosis, its confidence in each diagnosis' probability, the reasoning behind each diagnosis (that is, MYCIN would also list the questions and rules which led it to rank a diagnosis a particular way), and its recommended course of drug treatment. The MYCIN Expert System used backward chaining technology to diagnose infections based on symptoms and medical history and recommend treatment based on the data received.

Rule induction is an area of machine learning in which formal rules are extracted from a set of observations. The rules extracted may represent a full scientific model of the data, or merely represent local patterns in the data. Rule induction is a technique that creates "if–else–then"-type rules from a set of input variables and an output variable. A typical rule induction technique, such as Quinlan's C5, can be used to select variables because, as part of its processing, it applies information theory calculations in order to choose the input variables (and their values) that are most relevant to the values of the output variables. Therefore, the least related input variables and values get pruned and disappear from the tree. Once the tree is generated, the variables chosen by the rule induction technique can be noted in the branches and used as a subset for further processing and analysis. Remember that the values of the output variable (the outcome of the rule) are in the terminal (leaf) nodes of the tree. The rule induction technique also gives additional information about the values and the variables: the ones higher up in the tree are more general and apply to a wider set of cases, whereas the ones lower down are more specific and apply to fewer cases.

Check Your Progress

6. Which type of learning automation is used for repeated games against an unknown opponent?
7. What is a neural network?
8. What are the two main categories of expert system architecture?
9. How is systems based on analogical reasoning solve problems?

5.7 ANSWERS TO 'CHECK YOUR PROGRESS'

1. The difference between association rule learning and decision tree learning is that association rule learning is a process for finding out stimulating relation amongst variables in large databases whereas decision tree learning is used to conclude about an item's target value.
2. Reinforcement Learning (RL) is the process by which an agent improves its behaviour in an environment via experience.
3. Rote Learning is consists of simply storing of computed information. A lot of AI programs significantly improve their performance with the help of rote learning.
4. Learning in problem solving involves remembering the manner in which a problem is solved. Hence, when the same problem re-occurs, you can solve it more efficiently.
5. Inductive learning is essentially 'Learning by Example' as it involves drawing conclusions about previously unseen examples once the learning process is completed. Quinlan's decision trees, connectionism and decision list techniques are the most commonly used techniques for modelling the inductive learning process.
6. Q-learning is the learning automation used for repeated games against an unknown opponent.
7. A neural network is a system that can resolve paradigms that linear computing cannot.
8. The two main categories of expert system architectures are production and non-production system architectures.
9. Expert systems based on analogical architectures solve problems by finding similar problems and their solutions and applying the known solution to the new problem, possibly with some kind of modification.

5.8 SUMMARY

- An expert system is software that attempts to provide an answer to a problem, or clarify uncertainties where normally one or more human experts would need to be consulted.
- Common machine learning algorithms are supervised, unsupervised, semi-supervised, reinforcement, transduction and learning to learn.
- The main function of learning automation is to control the response of the environment and offer a new stimulus, if requested.
- The learning processes implemented in learning automation and artificial intelligence are reinforcement learning, temporal difference learning, active learning, Q-learning and inductive logic programming.
- The principle of learning automation is applied to the problems in the real world, for example, the Negative-Index Metamaterial (NIM) game.
- Induction methods, which are characterized as 'Learning by Example', begin with a set of observations, construct rules to account for these observations and try to find general patterns that can fully explain these observations.
- Artificial intelligence is the ability to think, imagine and create, memorize, understand, recognize patterns, make choices, adapt to change and learn from experience.
- Active learning is central to intelligence and hence refer to machine learning.
- Q-Learning is a form of learning automation and can be used online. This is why it is very suitable for repeated games against an unknown opponent.
- A neural network is a system that can resolve paradigms that linear computing cannot. Traditionally, it is used to describe a network or circuit of biological neurons.
- Biological neural networks are composed of real biological neurons that are connected in the peripheral or the central nervous system. Neuroscience identifies them as groups of neurons that fulfill a specific physiological function in laboratory analysis.
- Artificial neural networks are made up of interconnecting artificial neurons that are programming constructs which attempt to imitate the properties of biological neurons. These neural networks may be used either to understand biological neural networks or to solve problems related to artificial intelligence.
- Expert systems are widely applied to various industrial and commercial problems.
- The various stages of developing expert systems are task analysis, knowledge acquisition, prototype development, expansion and refinement and verification and validation.
- The two main categories of expert system architectures are production and non-production system architectures.

5.9 KEY TERMS

NOTES

- **Computational learning theory:** It is the part of theoretical computer science that deals with computational analysis of machine learning algorithms and their execution.
- **Artificial Neural Networks (ANN):** It is a mathematical or computational model that tries to simulate the structural and/or functional aspects of the biological neural networks.
- **Reinforcement Learning (RL):** It is the process by which an agent improves its behaviour in an environment via experience.
- **Q-learning:** It is a form of learning automation that can be used online.
- **Expert system:** It refers to software that attempts to provide an answer to a problem or clarify uncertainties where normally one or more human experts would need to be consulted.

5.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you know about Bayesian networks?
2. List the various approaches of learning.
3. What is reinforcement learning?
4. Define inductive learning.
5. List the steps of Q-learning algorithm.
6. What are the advantages of a neural network?
7. How expert systems are different from conventional computer programs?
8. How the huge costs associated with expert systems can be justified?
9. What is the process of building up an expert system?
10. Distinguish between the decision tree and analogical reasoning architecture of expert systems.

Long-Answer Questions

1. Discuss the theory and approaches of learning in detail.
2. Describe the various learning processes implemented in learning automation.
3. Write a short note on each of the following:
   (i) Learning by Induction
   (ii) Neural Networks
4. Explain the various components of a typical expert system.
5. Describe some applications of expert systems with me help of giving examples.

6. Elaborate on the production and non-production system architectures of expert systems.
7. Discuss the functioning of MYCIN and rule induction. Give appropriate examples.

5.11 FURTHER READING

Russell, Stuart J. and Peter Norvig. 2009. *Artificial Intelligence: A Modern Approach*, 3rd Edition. New Jersey: Prentice Hall.

Nilsson, Nils J. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco (California): Morgan Kaufmann Publishers, Inc.

Knight Kevin, Elaine Rich and B. Nair. *Artificial Intelligence* (SIE), 3rd Edition. New Delhi: Tata McGraw-Hill.

Sivanandam, S.N. and M. Paulraj. 2009. *Introduction to Artificial Neural Networks*. New Delhi: Vikas Publishing House Pvt. Ltd.

Rich, E. and K. Knight, *Artificial Intelligence*. New York: McGraw-Hill Book Company, 1991.

LiMin, Fu. 2003. *Neural Networks in Computer Intelligence.* New Delhi: Tata McGraw-Hill.