

Artificial Intelligence(AI) with it's Applications

Dr. Dipikaben Umakant Thakar

Assistant Professor

Department of Computer Science

Shri C. J. Patel College of Computer Studies (BCA),

Sankalchand Patel University, Visnagar ,Dist- Mehsana, Gujarat – 384315.

Mrs. PL. Natchiammai

Assistant Professor

Department of Electronics and Communication Engineering

Mookambigai College of Engineering

Srinivasa Nagar, Kalamavur (PO), Keeranur, Pudukkottai - 622502

Dr. R. J. Kavitha

Assistant Professor (Sl. Gr.)

Department of Electronics and Communication Engineering

University College of Engineering Panruti

Kumbakonam Highway, Panikkankuppam, Panruti, Tamil Nadu - 607106

Mrs. P. Anandavalli

Adhoc Assistant Professor

Department of Electronics and Communication Engineering

University College of Engineering Panruti

Kumbakonam Highway, Panikkankuppam, Panruti, Tamil Nadu - 607106

Mr. D. Yobu

Department of Computer Science and Engineering(Cyber Security)

J.J College of Engineering and Technology

Trichy-620009



Title: Artificial Intelligence(AI) with it's Applications

Author's Name: Dr. Dipikaben Umakant Thakar
Mrs. PL. Natchiammai
Dr. R. J. Kavitha
Mrs. P. Anandavalli
Mr. D. Yobu

Published by: QUILL TECH PUBLICATIONS,
Namakkal, 637003, Tamil Nadu, India

Publisher's Address: QUILL TECH PUBLICATIONS,
Namakkal, 637003, Tamil Nadu, India

Printer's Details: QUILL TECH PUBLICATIONS,
Namakkal, 637003, Tamil Nadu, India

Edition Details (I,II,III): I

ISBN: 978-93-6786-379-4

Month & Year: March, 2025

Copyright @ Dr. Dipikaben Umakant Thakar
Mrs. PL. Natchiammai
Dr. R. J. Kavitha
Mrs. P. Anandavalli
Mr. D. Yobu

Pages: 282

Price: 850/-

Authors' Index



Dr. Dipikaben Umakant Thakar (B.C.A, P.G.D.C.A, M.C.A, Ph.D.) is an accomplished academician and researcher specializing in computer science, with over 14 years of experience in teaching and research. She currently serves as an Assistant Professor at Shri C.J. Patel College of Computer Studies (BCA), Sankalchand Patel University, Visnagar.

Her research expertise lies in **big data analytics, clustering algorithms, and AI-driven optimization techniques**. Her Ph.D. work focuses on enhancing clustering algorithms for big data applications, and she has contributed extensively to **UGC CARE-listed journals and international conferences**. Passionate about **AI, big data, and cloud computing**, she actively participates in research collaborations, faculty development programs, and technical training sessions.

Dr. Dipikaben Umakant Thakar is proficient in multiple programming languages, including **C, C++, Java, ASP.NET, and PHP**, and works with databases such as **Oracle, SQL Server, and MongoDB**. As a dedicated mentor, she enjoys guiding students in research and fostering innovative approaches to AI applications. Her extensive experience in academia and technology fuels her commitment to advancing AI education and its real-world applications.

In this book, "**Artificial Intelligence (AI) with Its Applications**," Dr. Dipikaben Umakant Thakar explores the foundational concepts, methodologies, and diverse applications of AI, bridging the gap between theory and practical implementation.



Mrs. PL. Natchiammai is working as an Assistant Professor, Department of ECE, Mookambigai College of Engineering - Pudukkottai. She has 16 years of teaching experience. Her area of interest includes Image Processing, Embedded Systems, IoT and Machine Learning.



Dr. R. J. Kavitha working as an Assistant professor (Sl. Gr.) in the department of Electronics and Communication Engineering at the University College of Engineering Panruti (A Constituent college of Anna University, Chennai). She completed her B. E., ECE in GCE, Salem. M. E., Optical Communication in ACCET, Karaikudi, PhD in Anna University, Chennai. She has 25 years of teaching experience. She has published more than 50 publications in the national, international journals and conferences. Her area of interests are Wireless networks, Electronic devices & Circuits and Artificial intelligence.



Mrs. P. Anandavalli working as an Adhoc Assistant Professor in the department of Electronics and Communication Engineering at University College of Engineering Panruti. She completed B. E., ECE and M. E., Applied Electronics. She is pursuing her Ph. D., in Anna University. She has 18 years of teaching experience. She has published 10 national, international and conference papers. Her area of interests are Wireless networks, VLSI chip and Design and Artificial intelligence.



Mr. D. Yobu is an Assistant Professor in the Department of Computer Science and Engineering (Cyber Security) at J.J. College of Engineering and Technology, Tiruchirappalli. With over 12 years of experience in academia, he has made significant contributions to research and education. He is an actively engaging in knowledge sharing and academic collaborations.

ACKNOWLEDGEMENT

The completion of this book on Artificial Intelligence has been a fulfilling and enlightening journey, made possible through the support, encouragement, and contributions of many individuals and communities. We extend our deepest gratitude to all those who have played a role in bringing this work to fruition.

First and foremost, we express our sincere appreciation to our friends, colleagues, and mentors, whose invaluable guidance, insightful feedback, and unwavering encouragement have greatly enriched the depth and scope of this book. Their expertise has helped shape a comprehensive exploration of AI, covering both fundamental concepts and advanced applications.

We are profoundly grateful to our families for their unconditional love, patience, and support throughout this endeavor. Their belief in our work and their sacrifices have provided the strength and motivation needed to complete this book.

Our heartfelt thanks also go to the broader research and academic communities, whose groundbreaking work in artificial intelligence has laid the foundation for the knowledge presented in this book. Their continuous innovations and discoveries have inspired us to delve deeper into the subject and present it in a meaningful and accessible manner.

Furthermore, we acknowledge the role of AI itself in enhancing research, facilitating knowledge discovery, and streamlining the writing process. AI-driven tools have played a crucial role in refining content, analyzing data, and ensuring clarity and coherence, demonstrating the very impact of the field we seek to explore.

Finally, we give thanks to Almighty God for His wisdom, guidance, and blessings throughout this journey. His grace has been our sustaining force in overcoming challenges and ensuring the successful completion of this book on Artificial Intelligence.

We hope this book serves as a valuable resource for students, educators, researchers, and professionals, helping them navigate and contribute to the ever-evolving field of artificial intelligence.

Dr. Dipikaben Umakant Thakar
Mrs. PL. Natchiammai
Dr. R. J. Kavitha
Mrs. P. Anandavalli
Mr. D. Yobu

Index

Chapter No.	Topics	Page No.
1	Introduction to AI and Production Systems	1 - 71
1.1	Introduction to AI	1
1.2	Formulation and Definition of the Problem	9
1.3	System of Production	18
1.4	Methods of Search	23
1.5	Strategy for Control	35
1.6	Heuristic Search	35
1.7	Hill Climbing Search	48
1.8	Issues with Constraint Satisfaction	52
2	Representation of Knowledge	72 - 100
2.1	Game playing	72
2.2	Knowledge Representation	78
2.3	Understanding using Predicate Logic	86
2.4	Overview of Predicate Calculus	92
2.5	Resolution	95
2.6	Structured Representation of Knowledge	99
3	Structured Knowledge Representation, Handling Uncertainty and Bayesian Theory	101 - 161
3.1	Overview of Structured Knowledge Representation Techniques	101
3.2	Weak - Slot and Filler Structures	102
3.3	Strong Slot and Filler Structures	120
3.4	CYC	134
3.5	Schema	137
3.6	Introduction to Handling Uncertainty	138
3.7	Probabilistic Reasoning	144

4	Planning and Machine Learning	162 - 184
4.1	Basic Plan Generation Systems	162
4.2	Strips	167
4.3	Advanced Plan generation systems	170
4.4	K strips	177
4.5	Learning	178
5	Expert System	185 - 230
5.1	Expert System	185
5.2	Expert System Architecture	189
5.3	Expert System Shell	196
5.4	Explanation	197
5.5	Knowledge Engineering	202
5.6	Building Expert System	220
5.7	Expert System Case Study	223
6	Applications of AI in Various Fields	231 - 282
6.1	Heart Disease Prediction Using Logistic Regression	231
6.2	Credit Card Fraud Detection	237
6.3	Autism Prediction	245
6.4	Wine Quality Prediction	264
6.5	Calories Burnt Prediction	274

CHAPTER-1

Introduction to AI and Production Systems

1.1 Introduction to AI

Introduction

Many human mental activities such as developing computer programs, working out mathematics, engaging in common sense reasoning, understanding languages and interpreting it, even driving an automobile are said to demand "intelligence". Several computer systems have been built that can perform tasks such as these. Also there are specially developed computers systems that can diagnose disease, solve quadratic equations, understand human speech and natural language text.

We can say that all such systems possess certain degree of artificial intelligence. The central point of all such activities and systems is that "How to think" OR rather "How to make system think". The process of thinking has various steps like perceive, understand, predict and manipulate a world that is made up of tiny complex things or situations.

The field of AI not just attempts to understand but also it builds intelligent entities.

Various Definitions of AI

- ❖ AI may be defined as the branch of computer science that is concerned with the automation of intelligent behaviour. (Luger - 1993)
- ❖ Systems that thinks like human.
- ❖ The exciting new effort to make computers think ... machines with minds, in the full and literal sense. (Hallgeland - 1985)
- ❖ "The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning ..." (Bellman - 1978)
- ❖ Systems that act like humans.
- ❖ "The art of creating machines that perform functions that require intelligence, when performed by people". (Kurzweil - 1990)
- ❖ "The study of how to make computers do things at which, at the moment, people are better". (Rich and Knight - 1991).

Artificial Intelligence(AI) with it's Applications

- ❖ Systems that think rationally.
- ❖ The study of mental faculties through the use of computational models. (Charniak and McDermott - 1985)
- ❖ "The study of the computations that make it possible to perceive, reason and act". (Winston - 1992)
- ❖ Systems that act rationally.
- ❖ "Computational intelligence is the study of the design of intelligent agents". (Poole et al - 1998)
- ❖ "AI is concerned with intelligent behaviour in artifacts". (Nilsson - 1998)

These definitions vary along two main dimensions. First dimension is the thought process and reasoning and second dimension is the behaviour of the machine. The first seven definitions are based on comparisons to human performance where as remaining definitions measure success against an ideal concept of intelligence, which we call rationality. A system is rational if it does the "right thing" given what it knows. Historically, there are four approaches that are followed in AI. These four approaches are Acting Humanly, Thinking Humanly, Thinking Rationally and Acting Rationally.

Let us consider four approaches in detail.

1. Acting Humanly

Turing test

For testing intelligence Alan Turing (1950) proposed a test called as Turing test. He suggested a test based on common features that can match with the most intelligent entity - human beings.

Computer would need to possess following capabilities:

- ❖ Natural language processing - To enable it to communicate successfully in English.
- ❖ Knowledge representation to store what it knows, what it hears.
- ❖ Automated reasoning to make use of stored information to answer questions being asked and to draw conclusions.
- ❖ Machine learning to adapt to new circumstances and to detect and make new predictions by finding patterns.

Turing also suggested to have physical interaction between interrogator and computers. Turing test avoids this but Total Turing Test includes video signal so that the interrogator can test the subject's perceptual abilities, as well as the opportunity for the interrogator to pass the physical objects "through the hatch".

Artificial Intelligence(AI) with it's Applications

To pass total turing test in addition, computer will need following capabilities.

- ❖ Computer vision to perceive objects.
- ❖ Robotics to manipulate objects.

2. Thinking Humanly

As we are saying that the given program thinks like human it we should know that how human thinks. For that, the theory of human minds needs to be explored. There are two ways to do this: through introspection i.e. trying to catch our own thoughts as they go by and through psychological experiments.

If computer programs, I/O and timing behaviours matches corresponding human behaviours, that is, we can say that some of the program's mechanisms could also be operating in human. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology that try to construct precise and testable theories of the workings of human mind.

3. Thinking Rationally - the "laws of thought approach"

The concept of "Right thinking" was proposed by Aristotle. This idea provided patterns for argument structures that always yielded correct conclusions when given correct premises.

For example,

"Ram is man",
"All men are mortal",
"Ram is mortal".

These laws of thought were supposed to govern the operation in the mind; their study initiated the field called logic which can be implemented to create intelligent systems.

4) Acting Rationally

An agent (Latin agere-to do) is something that acts. But computer agents are expected to have more other attributes that distinguish them from just the "programs", because they need to operate under autonomous control, perceiving their environment, persisting over a prolonged time period, adapting to change and being capable of taking on another goals. A rational agent is expected to act so as to achieve the best outcome or when there is uncertainty to achieve best expected outcome.

The laws of thought emphasis on correct inference which should be incorporated in rational agent.

The AI Foundation

Now we discuss the various disciplines that contributed ideas, viewpoints and techniques to AI.

Philosophy provides base to AI by providing theories of relationship between physical brain and mental mind, rules for drawing valid conclusions. It also provides information about knowledge origins and the knowledge leads to action.

Mathematics gives strong base to AI to develop concrete and formal rules for drawing valid conclusions, various methods for data computation and techniques to deal with uncertain information.

Economics support AI to make decisions so as to maximize payoff and make decisions under uncertain circumstances.

Neuroscience gives information which is related to brain processing which helps AI to develop data processing theories.

Psychology provides strong concepts of how humans and animals think and act which helps AI for developing process of thinking and actions.

The Strong and Weak AI

After taking brief look at various disciplines that contribute towards AI, now let us look at the concept of strong and weak AI which also gives basic foundation for developing automated systems.

Strong AI

This concept was put forward by John Searle in 1980 in his article, "Minds, Brains and Programs". Strong form AI provides theories for developing some form of computer based AI that can truly reason and solve problems. A strong form of AI is said to be sentient or self-aware.

Strong AI can be categorized as,

- ❖ Human-like AI - In which the computer program thinks and reasons much like a human-mind.
- ❖ Non-human-like AI - In which the computer program develops a totally non-human sentience, and a non-human way of thinking and reasoning.

Artificial Intelligence(AI) with it's Applications

Weak AI

Weak artificial intelligence research deals with the creation of some form of computer based AI that cannot truly reason and solve problems. They can reason and solve problems only in a limited domain. Such a machine would, in some ways, act as if it were intelligent, but it would not possess true intelligence.

There are several fields of weak AI, one of which is natural language. Much of the work in this field has been done with computer simulations of intelligence based on predefined sets of rules. Very little progress has been made in strong AI. Depending on how one defines one's goals, a moderate amount of progress has been made in weak AI.

The History of AI

The early work that is now generally recognized as AI was done in the period of 1943 to 1955. The first AI thoughts were formally put by men McCulloch and Walter Pitts (1943). Their idea of AI was based on three theories, firstly basic phsycology (the function of neurons in the brain), secondly formal analysis of propositional logic and third was Turing's theory of computation.

Later Donald Hebb in 1949 demonstrated simple updating rule for modifying the connection strengths between neurons. His rule now called Hebbian learning which is considered to be great influencial model in AI.

There were huge early day work that can be recognized as AI but Alan Turing who first articulated a complete vision of AI in his 1950 article named "Computing Machinery and Intelligence".

Real AI birth year is 1956 where in John McCarthy held workshop on automata theory, neural nets and study of intelligence where other researchers also presented their papers and they come out with new field in computer science called AI.

From 1952 to 1969 large amount of work was done with great success.

Newell and Simon's presented General Problem Solver (GPS) within the limited class of puzzles it could handle. It turned out that the order in which the program considered subgoals and possible actions was similar that in which humans approached the same problems. GPS was probably the first program which has "thinking humanly" approach.

Herbert Gelernter (1959) constructed the Geometry Theorem Prover which was capable of proving quite tricky mathematics theorem.

At MIT, in 1958 John McCarthy made major contributions to AI field: development of HLL LISP which has became the dominant AI programing language.

In 1958, McCarthy published a paper entitled Programs with Common Sense, in which he described the Advice Taker, a hypothetical program that can be seen as the first complete AI

Artificial Intelligence(AI) with it's Applications

system. Like the Logic Theorist and Geometry Theorem Prover. McCarthy's program was designed to use knowledge to search for solutions of problems.

The program was also designed so that it could accept new axioms in the normal course of operation, thereby allowing it to achieve competence in new areas without being reprogrammed. The Advice Taker thus embodied the central principles of knowledge representation and reasoning.

Early work building on the neural networks of McCulloch and Pitts also flourished. The work of Winograd and Cowan (1963) showed how a large number of elements could collectively represent an individual concept, with a corresponding increase in robustness and parallelism. Hebb's learning methods were enhanced by Bernie Widrow (Widrow and Hoff, 1960; Widrow, 1962), who called his networks adalines, and by Frank Rosenblatt (1962) with his perceptrons. Rosenblatt proved the perceptron convergence theorem, showing that his learning algorithm could adjust the connection strengths of a perception to match any input data, provided such a match existed.

In 1965, Weizenbaum's ELIZA program appeared to conduct a serious conversation on any topic by basically borrowing and manipulating the sentences given by a human. None of the programs developed so far, had complex domain knowledge and were called 'weak' methods. Researchers realized that it was necessary to use more knowledge for more complicated, larger reasoning tasks.

The DENDRAL program was developed by Buchanan in 1969 and was based on these principles. It was a unique program that effectively used domain specific knowledge in problem solving. In the mid-1970's, MYCIN, a program developed to diagnose blood infections. It used expert knowledge to diagnose illnesses and prescribe treatments. This program is also known as the first program, which addressed the problem of reasoning with uncertain or incomplete information.

Within a very short time a number of knowledge representation languages were developed such as predicate calculus, semantic networks, frames and objects. Some of them are based on mathematical logic such as PROLOG. Although PROLOG goes back to 1972, it did not attract wide spread attention until a more efficient version was introduced in 1979.

As the real, useful strong works on AI were put forward by researchers, AI emerged to be a big Industry.

In 1981, Japanese announced 5th generation project a 10 -year plan to build intelligent computers running PROLOG. US also formed the Micro-electronics and Computer Technology Corporation (MCC) for research in AI.

Overall the AI industry boomed from few million dollars in 1980 to billions of dollars in 1988. But soon after that AI industry had huge setback as many companies suffered as they failed to deliver on extra vagant promises.

Artificial Intelligence(AI) with it's Applications

In late 1970s more research were done by psychologists on neural networks which continued in 1980s.

In 1990s AI emerged as a science. In terms of methodology AI has finally come firmly under the scientific method. In recent years approaches based on Hidden Markov Models (HMMS) have come to dominate the AI field. This model is based on two aspects one is rigorous mathematical model theory and second is, these models are generated by a process of training on a large corpus real speech data.

Judea Pearl's (1988) Probabilistic Reasoning in Intelligent Systems led to a new acceptance of probability theory in AI. Later Bayesian network was invented which can represent uncertain knowledge along with reasoning support.

Judea Pearl, Eric Hovitz and David Hackerman in 1986 promoted the idea of normative expert systems that can act rationally according to the laws of decision theory.

Similar but slow revolution have occurred in robotics, computer vision and knowledge representation.

In 1987 a complete agent architecture called SOAR was work out by Allan Newell, John Laired and Paul Rosenbloom. Many such agents were developed to work in big environment "Internet". AI systems have become so common in web based applications that the "- bot" suffix has entered in everyday language.

AI technologies underlie many Internet tools, such as search engines, recommender systems and website.

While developing complete agents it was realized that previously isolated subfields of AI need to reorganize when their results are to be tied together.

Today, In particular it is widely appreciated that sensory systems (vision, sonar, speech-recognition, etc.) cannot deliver perfectly reliable information about the environment. Hence reasoning and planning systems must be able to handle uncertainty. AI has been draw in to much closer contact with other fields such as control theory and economics, that also deal with agents.

What AI can do Today ?

Autonomous Planning and Scheduling

NASA's Remote Agent program became the first on-board autonomous planning program to control the scheduling of operations for spacecraft. Such remote agents can do task of detecting, diagnosing and recovering from problems as they occurred.

Game Playing

A computer chess program by IBM named as Deep Blue defeated world chess champion Garry Kasparov in exhibition match in 1997. Such type of gaming programs can be developed using AI techniques.

Autonomous Control

The ALVINN computer vision system was trained to steer car to keep it following a lane. It was made to travel 2850 miles in which 98% of the time control was with the system and only 2% of the time human took over. AI can give more theories to develop such systems.

Diagnosis

Heckerman (1991) describes a case where a leading expert on lymph node pathology scoffs at a program's diagnosis of an difficult case. The machine can explain the diagnosis. The machine points out the major factors influencing its decision and explain interaction of several of the symptoms in this case. If such diagnostic programs are developed using AI then highly accurate diagnosis can be made.

Logistic Planning

In 1991 during the Persian Gulf Crisis U.S. forces deployed a dynamic analysis and replanning tool name DART for automated logistics planning and scheduling for transportation.

Robotics

For doing complex and critical tasks systems can be developed using AI techniques. For e.g. Surgeons can use robot assistants in microsurgery which can generate 3D vision of patients internal anatomy.

Language Understanding and Problem Solving

PROVERB is computer program which expert in solving crossword puzzles. It can make use of constraints or possible word fillers, a large database of past puzzles and variety of information sources including dictionaries and online databases. Such as a list of movies and the actors that appears in them.

AI does not generate magic or science fiction but rather it can develops science, engineering and mathematics system.

Recent progress in understanding the theoretical basis for intelligence has gone hand in hand with improvements in the capabilities of real systems. The subfields of AI have become more integrated and AI has found common ground with other disciplines.

1.2 Formulation and Definition of the Problem

What is Problem Solving?

For solving any type problem (task) in real world one needs formal description of the problem.

One should have clear understanding of following aspects of the problem,

1. What is the Explicit Goal of the Problem

Goals help to organize behaviour of systems by limiting the objectives that the agent is trying to achieve. Goal formulation is based on the current situation and the agent's performance measure. It is first step towards problem solving.

2. What is Implicit Criteria for Success

That is how success is defined. That will be the ultimate thing system needs to achieve, which is the problem solution's output.

3. What is the Initial Situation

It means that what is going to be the start state of problem being solved.

4. Ability to Perform

It tells how agents transforms from one situation to another, how operations and rules are specified which change the states of the problem during solution process.

Well Defined Problems

Problem formulation is the process of deciding what actions and states to consider, given a goal.

A problem can be defined formally by four components.

1. Initial state that the agent starts in

For example

Consider an agent program Indian Traveller developed for travelling from Pune to Chennai travelling through different states. The initial state for this agent can be described as In (Pune).

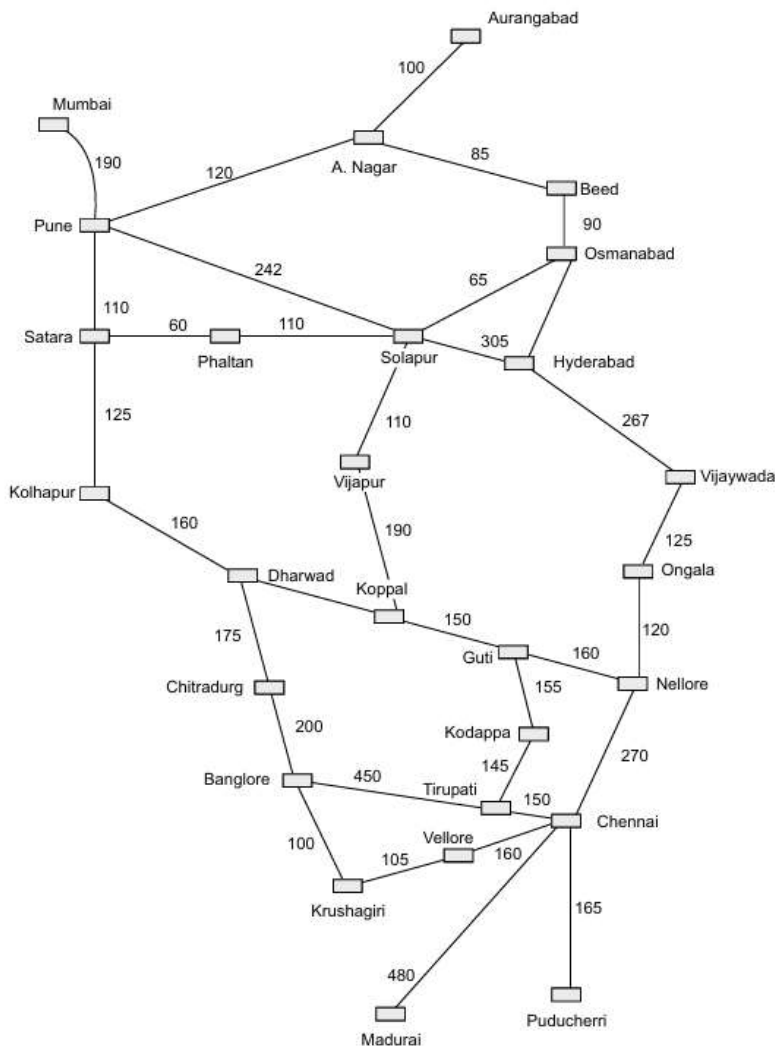


Fig. 1.1 Indian traveller map.

2. A description of the possible actions available to the agent

The most common formulation uses a successor function. Given a particular state x , SUCCESSOR Function (X) returns a set of $\langle \text{action}, \text{successor} \rangle$, ordered pairs, where each action is one of the legal actions in state x and each successor is a state that can be reached from x by applying the action.

For example

From the state In (Pune), the successor function for Indian Traveller problem would return.

{ $\langle \text{Go (Mumbai)}, \text{In (Mumbai)} \rangle$

$\langle \text{Go (AhemdNagar)}, \text{In (AhemdNagar)} \rangle$

< Go (Solapur), In (Solapur)>

< Go (Satara), In (Satara)>

}

Together, the initial state and successor function implicitly define the state space of the problem - which is the set of all states reachable from the initial state.

The state space forms a graph in which the nodes are states and the arcs between nodes are actions.

A path in the state space is a sequence of states connected by a sequence of actions.

3. The goal test, which determines whether a given state is goal (final) state. In some problems we can explicitly specify a set of goals. If a particular state is reached we can check it with set of goals and if a match is found success can be announced.

For example

In Indian Traveller problem the goal is to reach chennai i.e. it is a singleton set {In (Chennai)}.

In certain types of problems we can not specify goals explicitly. Instead, goal is specified by an abstract property rather than an explicitly enumerated set of states.

For example

In chess, the goal is to reach a state called "Checkmate" where the opponent's king is under attack and can't escape. This "Checkmate" situation can be represented using various state spaces.

4. A path cost function that assigns a numeric cost (value) to each path. The problem-solving agent is expected to choose a cost-function that reflects its own performance measure.

For Indian - Traveller agent we can have time required as cost for path-cost function. It should consider length of each road being travelled.

In general step-cost of taking action 'a' to go from state x to state y is denoted by $c(x, a, y)$.

The above 4 elements define a problem and can be put together in single data structure which can be given as input to a problem-solving algorithm.

A solution to the problem is a path from the initial state to a goal state.

We can measure quality of solution by the path cost function. We can have multiple solutions

to the problem. The optimal solution will be the one with lowest path cost among all the solutions.

Problem Formulation Types

There are two main kinds of problem formulation,

1. Incremental formulation
2. Complete-state formulation.

Depending upon problem requirements and specification one can decide which one to go for.

Incremental formulation

- ❖ It involves operators that augment the state description, starting with an empty state.
- ❖ It generates many sequences.
- ❖ Memory requirements is less as all states are not explored (exploration will be done till the goal is found).

For example

For the 8-queens problem, incremental formulation states that, each action adds a queen to the state. In this formulation we have $64 \times 63 \times \dots \times 57 \approx 3 \times 10^{14}$ possible sequences to investigate.

Complete state formulation

- ❖ In this initially we will have some basic configuration represented in initial state.
- ❖ Here while doing any action first the conditions on the actions will be checked so that the configuration state after the action will be same legal state.
- ❖ It takes up large memory as complete state space is generated. This formulation reduces number of sequences generated.

For example

In 8-queen problem initially all the queens will be arranged on the board. The action will be 'move a queen to the next square such that it is not attacking'.

This complete state formulation reduces state space from 3×10^{14} (which is for incremental formulation) to just 2,057 and solutions are easy to find.

Example of incremental formulation and complete-state formulation

Consider 8-queen problem,

Incremental formulation

1. States
 - ❖ Arrangement of upto 8 queens on the board.
-

2. Initial state
 - ❖ Empty board.
3. Successor function (operators)
 - ❖ Add a queen to any square.
4. Goal test
 - ❖ All queens on board
 - ❖ No queen attacked.

Properties: 3×10^{14} possible sequences.

Complete state formulation

1. States
 - ❖ Arrangement of 8-queens on the board.
2. Initial state
 - ❖ All 8 queens on board.
3. Successor function (operators)
 - ❖ Move a queen to a different square.
4. Goal test
 - ❖ No queen attacked.

Properties: Good strategies can reduce the number of possible sequences which are considerable.

Solving the Problem

Finding the solution of a problem is procedure which involves following phases,

1. **Problem definition:** Wherein detailed specification of inputs and what constitutes an acceptable solution is described.
2. **Problem analysis:** Wherein problem is studied through various view points like inputs, to the problem, environment of the problem, expected outputs.
3. **Knowledge representation:** Wherein the known data about the problem and various expected stimuli from environment is represented in particular format which is helpful for taking actions.
4. **Problem solving:** Wherein the selection of best suited techniques for problem solutions are thought of and finalized.

Problem Solving Agents

Approach of Problem Solving Agent

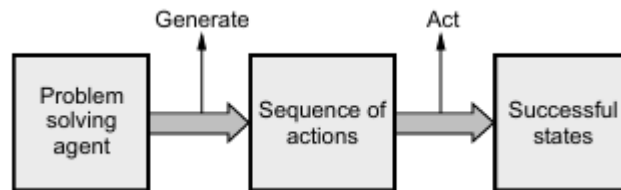


Fig. 1.2 Steps in problem solving.

- ❖ Goal based agents are also called as problem solving agent.
- ❖ Problem solving agent adapt to the task environment understand goal and achieve success
- ❖ Problem solving agents determine sequence of actions which generate successful state.
- ❖ Problem solving agent can be aimed at maximizing performance measure there by developing intelligent problem solving agent.

Steps in Problem Solving

Problem solving agent achieves success by taking following approach to problem solution,

Step 1: Goal setting

Agent set the goal by considering the environment.

Step 2: Goal formulation

The goals set in step 1 are formalized in the frame work. The key activity in goal formulation is

1. To observe current state.
2. To tabulate agents performance measures.

Step 3: Problem formulation

After formulating goal, it is required to find out what will be the sequence of actions which generate goal state.

Problem formulation is a way of looking at actions and states generated because of actions, which leads to success.

Step 4: Search in unknown environment

If the task environment is unknown then agent first tries different sequence of actions and gathers knowledge (i.e. learning). Then agent gets known set of actions which leads to goal state. Thus agent search for describable sequence of actions this process is called as searching process.

With knowledge of environment and goal state we can design a search algorithm. A search algorithm is a procedure which takes problem as input and return its solution which represented in the form of action sequence.

Step 5: Execution phase

Once the solution is given by the search algorithm then the actions suggested by the algorithm are executed. This is the execution phase. Solution guides agent for doing the actions. After executing the actions agent again formulate new goal.



Fig. 1.3 Problem solving agent.

Algorithm

Procedure or method: Problem solving agent (unknown space, percept).

Results: An action.

Input: $P \rightarrow$ percept (Environment perception)

Static:

1. $A \rightarrow$ An action sequence, initially with null value.
2. $S \rightarrow$ State - current state.
3. $G \rightarrow$ Goal - A goal initially null.

4. $P \rightarrow$ Problem - A real world situation.

State - update state (State, percept)

If (s) is empty then do

$g \leftarrow$ Formulate goal (s)
 $P \leftarrow$ Formulate problem (s, g)
 $S \leftarrow$ Search (p)
 $G \leftarrow$ First (s)
 $S \leftarrow$ Rest (s)

Example problems

1. General Route Finding Problem

Problem Statement: Route-finding problem is defined in terms of specified locations and transitions along links between them. Route-finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning, and air line travel planning systems.

1. **States:** Locations
2. **Initial state:** Starting point
3. **Successor function (operators):** Move from one location to another.
4. **Goal test:** Arrive at a certain location.
5. **Path cost:** May be quite complex, which can involve factors like, Money, time, travel, comfort, scenery, ... etc.

Simplified Example of Route Finding Problem [Airline Travelling Problem]

Problem Statement: Starting from initial location one has to reach at the specified destination by some prespecified time.

1. **States:** Each state is represented by a location (e.g. an airport) and the current time.
 2. **Initial state:** This is specified by the problem.
 3. **Successor function:** This returns the states resulting from taking any scheduled flight (perhaps further specified by seat class and location), leaving later than the current time plus the time within-airport transit time, from the current airport to another.
 4. **Goal test:** Are we at the destination by some prespecified time ?
-

5. **Path cost:** This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

2. VLSI Layout Problem

Problem Statement: A VLSI layout problem requires positioning millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.

1. States

- ❖ Positions of components, wires on a chip.

2. Initial state

- ❖ Incremental: No component placed.
- ❖ Complete-state: All components place (e.g. randomly, manually).

3. Successor function (operators)

- ❖ Incremental: Place components, route wire.
- ❖ Complete-state: Move component, move wire.

4. Goal test

- ❖ All components placed.
- ❖ Components connected as specified.

5. Path cost

- ❖ May be complex.

One can consider distance, capacity, number of connections per component, for path cost computation.

Detail description of VLSI layout problem

When logical design is made, the layout problem come and the problem is divided into two parts

- 1) Cell layout 2) Channel routing as below,

1. Cell layout

- ❖ The basic components of circuit are grouped into cells, each cells perform some recognition. The cells of standard size are connected to each of the other cell.
 - ❖ The overlapping between the cell component is avoided. Some sort of room is provided for connecting wires.
-

2. Channel routing

- ❖ The cell components are fixed on a chip.
- ❖ The channel routing task is to search a particular route for each wire. The functional task is achieved through the gaps between the cells.
- ❖ The complex algorithm are designed to perform the channel routing. The degree of complexity is so high, but it is always possible to solve problem with specific algorithm.

1.3 System of Production

Since, search forms the core of many intelligent processes, production system is useful to structure AI programs in a way that facilitates describing and performing the search process.

Production systems provide such structures. If one adopts a system with production rules and a 'rule interpreter' then the system is known as a 'production system'.
or "One convenient representational form of an action function is a production system".

A production system is a model of computation that provides pattern-directed search control using a set of production rules, a working memory, and a recognize-act cycle.

- ❖ The productions are rules of the form $C \rightarrow A$, where the LHS is known as the condition and the RHS is known as the action. These rules are interpreted as follows: given condition, C , take action A . The action part can be any step in the problem solving process. The condition is the pattern that determines whether the rule applies or not.
- ❖ There is a set of rules, each consisting of a left side (a pattern) that determines the applicability of the rule and a right side that describes the operation to be performed, if the rule is applied. These rules are called production rules or simply productions. Each rule is written in the form $c_i \rightarrow a_i$, where c_i is the condition part and a_i is the action part.
- ❖ A production system consists of a list of such rules -

$$\begin{aligned} c_1 &\rightarrow a_1 \\ c_2 &\rightarrow a_2 \\ &: \\ c_i &\rightarrow a_i \\ &: \\ c_m &\rightarrow a_m \end{aligned}$$

In general the condition part of a rule can be any binary-valued (0,1) function of the features resulting from perceptual processing of line sensory inputs. Often, it is a monomial - a conjunction of Boolean literals. To select an action, the rules are processed as follows,

Artificial Intelligence(AI) with it's Applications

1. Starting with the first rule, namely $c_1 \rightarrow a_1$; we look for the first rule is the ordering whose condition part evaluates to 1 and select the action part of that rule. The action part can either be primitive action, a called to another production system, or a set of action to be executed simultaneously.
2. One or more knowledge/database that contain whatever information is appropriate for the particular task. Some parts of the database may be permanent, while other parts of it may pertain only to the solution of the current problem. The information in these databases may be structured in any appropriate way.
3. A control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.
4. A rule applier-One major issue to be solved in a production system is 'conflict resolution'. Conflict resolution arises when there are more than one rule that can be fired in a situation and the rule interpreter is to decide which is to be fired, what is the order of triggering and whether to apply all that are applicable or to be only selective.

Following are a few strategies by which conflicts can be resolved.

These are termed as conflict resolution strategies. Some of the conflict resolution strategies are,

i) Perform the first - In this method, the system chooses the first rule that matches.

ii) Sequencing techniques - Adopt the rules in the sequence they are.

iii) Perform the most specific - If there are two matching rules and one is more specific than the other, activates the most specific rule.

iv)Most recent policy - It is generally believed that a newly added rule is more knowledgeable than existing ones. Hence, if a system is adopting this method, it should fire the most recent rule. However, there is slight overhead associated with this strategy in that the system has to keep track of which rule came in at what time, which rules were modified etc.

- ❖ Working memory contains a description of the current state of the world in the problem-solving process. The description is matched against the conditions of the production rules. When a conditions matches, its action is performed. Actions are designed to alter the contents of working memory.
- ❖ The recognize-act cycle is the control structure. The patterns contained in working memory are matched against the conditions of the production rules, which produces a subset of rules known as the conflict set, whose conditions match the contents of working memory. One (or more) of the rules in the conflict set is selected (conflict

resolution) and fired, which means its action is performed. The process terminates when no rules match the contents of working memory.

Background

- ❖ Post (1943) proposed the production rule model as a formal theory of computation. It is equivalent in power to a Turing machine.
- ❖ Newell and Simon (1960s) system, General Problem Solver used production system as a model of human cognition. Production rules represent problem-solving skills stored in a person's long-term memory. The working memory represents the person's current focus of attention. Choosing rules to apply, occurs through unconscious pattern matching.
- ❖ Production systems have been developed for a wide variety of problems, ranging from algebra word problems, mathematical and logical proofs, physics problems and games.
- ❖ John Anderson (1983) proposed a production system known as ACT* as a model of human cognition and learning.
- ❖ Newell, Rosenbloom and Laird (1990) proposed a production system known as SOAR (Symbols, Operators AND Rules) as a model of human cognition and learning.
- ❖ The OPS (Official Production System) languages are based on the production system model.
- ❖ 1980s : Production systems were the basis of rule-based expert systems.
- ❖ Production system is model for Human Problem Solving (SOAR, ACT*)

Purpose of Production System and Various Types of Production Systems

Production system is a good way to describe the operations that can be performed in a search for a solution to a problem.

Production systems like problems can be described by a set of characteristics that shed some light on how they can easily be implemented?

As we have following classes of production systems,

1. A monotonic production system
2. A non-monotonic production system
3. A partially commutative production system
4. A commutative production system

Artificial Intelligence(AI) with it's Applications

A monotonic production system: It 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 non-monotonic production system: It is a production system in which the application of a rule prevents the later application of another rule that could also have been applied at the time the first rule was selected.

A partially commutative production system: This system states that if the application of a particular sequence of rules transforms state x into state y , then any permutation of those rules that is allowable also transforms state x into state y .

A commutative production system: It is a production system that is both monotonic and partially commutative.

There exist an infinite number of production systems that describes ways to find solutions. For any solvable problem i.e., any problem can be solved by any production system. Or generally we cannot define a relationship between kinds of production and kinds of production system.

Since, all problems can be solved by all kinds of systems.

Table. 1.1 shows four categories of production systems.

	Monotonic	Non-monotonic
Partially commutative	Theorem proving	Robot navigation
Not partially commutative	Chemical synthesis	Bridge

Table. 1.1 The four categories of production systems.

Partially commutative, monotonic production systems are useful for solving ignorable problems as shown in left upper corner. Ignorable problems are one for which a natural formulation leads to solution steps that can be ignored. Such a natural formulation will then be a partially commutative, monotonic system. These problems have some creative processes and ignore changing old ones. Theorem proving is one of the example of such a creative process. These systems are useful from implementation standpoint as it can be implemented without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed.

Non-monotonic, partially commutative systems are useful for the problems in which changes occur but can be reversed and in which order of operations is not critical.

This is usually the case in physical manipulation problems, such as robot navigation on a flat plane.

Consider following operators:

N - to move north

E - to move east

W - to move west

S - to move south.

To reach its goal, it does not matter whether the robot executes $N - N - E$ or $N - E - N$ depending on how the operators are chosen.

8 -puzzle and block world problems are also of same kind problem. Not partial commutative are useful for many problems in which irreversible changes occur. For example, for making a particular chemical compound, there is fixed process for it. Say a chemical has to be added to B at certain temperature then C chemical should be added to mixture to get the required compound. This would not be possible if we add C to A first then B to it. So, in this type of processes order is very important i.e., make correct decision the first time. In this system, it is less likely to produce the same node many times in the search process.

Advantages and Disadvantages of Production System

Advantages of Production Systems

- ❖ Production systems provide an excellent tool for structuring AI programs.
- ❖ Production systems are highly modular because the individual rules can be added, removed or modified independently.
- ❖ The production rules are expressed in a natural form, so the statements contained in the knowledge base should be the recording of an expert thinking out loud.

Disadvantage of Production Systems

One important disadvantage is the fact that it may be very difficult to analyse the flow of control within a production system because the individual rules don't call each other.

Search Control in Production System

- ❖ Data driven ($C \rightarrow A$): If C matches, then perform A , changing WM .
- ❖ Goal driven ($C \rightarrow A$): If A matches a goal in WM, then add C (subgoal) to WM.
- ❖ The order of the rules or clauses affects search control.
- ❖ Heuristic knowledge can be built into the rules.
- ❖ Conflict resolution: refraction - If rule $C \rightarrow A$ has fired, don't refire it until the conditions, C , have changed in WM. (prevents looping)
- ❖ Conflict resolution recency - fire rules whose conditions match the newer additions to WM. (Focused reasoning)
- ❖ Conflict resolution: specificity - prefer rules that have more conditions and hence match with fewer WM patterns.

Important Characteristics of Production System Model

- ❖ There is separation of knowledge (the rules) and control (recognize-act cycle) in the system.
- ❖ Natural mapping onto state space search (data or goal driven) is done.
- ❖ There is modularity of production rules (rules represent chunks of knowledge).
- ❖ Pattern-directed control (more flexible than algorithmic control) is done in the system.
- ❖ Opportunities for heuristic control can be built into the rules.
- ❖ Tracing and explanation (Simple control, informative rules) is possible in the system.
- ❖ The systems are language independent.
- ❖ It has been used as model for human problem solving (SOAR, ACT*).

1.4 Methods of Search

At every stage in state space generating algorithms we need to apply searching procedure so as to reach to goal state. Search is a systematic examination at states to find path from the root state (initial state) to the goal state. The output of this procedure is the solution (goal) state.

There are 2 Basic Types of Search Strategies

1. Uninformed search (Blind search)

They have no additional information about states other than provided in the problem definition.

They can only generate successors and distinguish between goal state and non-goal state.

2. Informed search (Heuristic search)

This can decide whether one non-goal state is more promising than another non-goal state.

Key Point: All the search strategies are distinguished by the order in which the nodes are expanded.

Uninformed Search Strategies

- ❖ B.F.S.
- ❖ D.F.S.
- ❖ Depth limited search
- ❖ Iterative deepening DFS

- ❖ Bidirectional search
- ❖ Uniform cost search

In the following section we will discuss each uninformed searching in detail. For each searching technique we will see its basic methodology, its algorithmic implementation and its performance evaluation. Performance evaluation will be done on the basis of 4 criteria's we have seen previously namely,

1. Completeness
2. Optimality
3. Time complexity
4. Space complexity.

Generally it does happen that space and time complexity depends on some factors hence we will discuss them combinely.

Breadth-First Search

The procedure: In BFS root node is expanded first, then all the successor of root node are expanded and then their successor and so on. That is the nodes are expanded level wise starting at root level.

The implementation: BFS can be implemented using first, in first out queue data structure where fringe will be stored and processed. As soon as node is visited it is added to queue. All newly generated nodes are added to the end of the queue, which means that shallow nodes are expanded before deeper nodes.

The performance evaluation

1. **Completeness:** BFS is complete because if the shallowest goal node is at some finite depth d , BFS will eventually find it and will generate solution. (assuming that branching factor b is finite).
2. **Optimality:** The shallowest goal node is not necessarily optimal. BFS will yield optimal solution only when all actions have the same cost, let it be at any depth.
3. **Time and space complexity:** As the level of search tree grows more time is incurred. In general if search tree is at level d then $O(b^{d+1})$ time is required, where b is the number of nodes generated from each node, starting at root node i.e. root generates b nodes and each b node generates b more and so on shown below.

$$b + b^2 + b^3 + \dots + b^d + (b^{d+1} - b)$$
$$= O(b^{d+1})$$

Artificial Intelligence(AI) with it's Applications

Every node generated should remain in memory till its exploration. If branching factor 'b' is more then more memory will be required.

For example

If branching factor $b = 10$ at some level $d = 6$. If we assume 10,000 nodes will be generated per second and per node 1000 bytes are required for storage.

Then total generated nodes = 10^7 which will take 19 minutes but it will take 10 Gigabytes for storing them.

We can conclude that for BFS space complexity is major issue concerned than time complexity. Time complexity is critical issue when depth of tree increases.

Key Point: We can use uninformed methods for exponential-complexity search only when problems have smallest instances.

Algorithm BFS (G, n)

//Breadth first search of G

{

for $i := 1$ to n do // Mark all vertices unvisited

visited $[i] := 0$;

for $i := 1$ to n do

if (visited $[i] = 0$) then BFS (i);

}

BFS

❖ Time complexity - $O(b^{d+1})$

❖ Space complexity - $O(b^{d+1})$

BFS

D and J are goal state.

[D is found through the path] $\rightarrow$ (A-B-D) (shallowest goal)

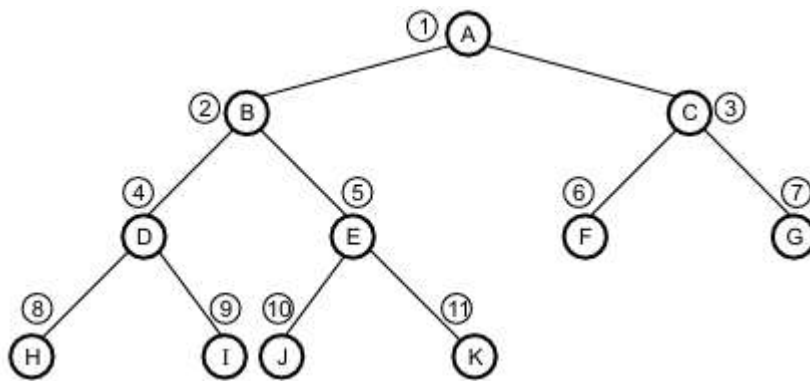


Fig. 1.4 Breadth-first search.

Note: The numeric value beside each node indicates the order in which nodes are visited (reached).

Uniform Cost Search

The procedure: Uniform cost search expands the node ' n ' with the lowest path cost. Uniform cost search does not care about the number of steps a path has, instead it considers their total cost. Therefore, there is a chance of getting stuck in an infinite loop if it ever expands a node that has a zero-cost action leading back to the same state.

The implementation: We can use queue data structure for storing fringe as in BFS. The major difference will be while adding the node to queue, we will give priority to the node with lowest path cost. (So the data structure will be a priority queue).

The performance evaluation

1. **Completeness:** Uniform cost search guarantees completeness provided the cost of every step is greater than or equal to some small positive constant " ϵ ".
2. **Optimality:** If cost of every step is greater than or equal to some small positive constant then uniform cost search will yield optimal solution, by reaching the goal state which has lowest path cost.
3. **Time and space complexity:** Uniform-cost search does not care about the number of steps a path has, but only about their total cost. Therefore, it will get stuck in an infinite loop if it ever expands a node that has a zero-cost action leading back to the same state.

Uniform-cost search is guided by path costs rather than depths, so its complexity cannot easily be characterized in terms of b and d . Instead, let C^* be the cost of the optimal solution, and assume that every action costs at least ϵ . Then the algorithm's worst-case time and space complexity is $O(b^{\lceil C^*/\epsilon \rceil})$, which can be much greater than b^d .

Uniform cost search

- ❖ Time complexity - $O(2^{C/\epsilon})$
- ❖ Space complexity - $O(C/\epsilon)$

Where, C is the cost of optimal solution.

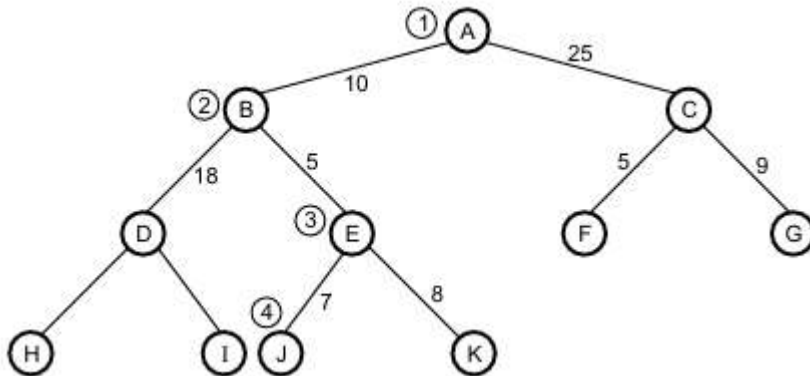


Fig. 1.5 Uniform cost search.

Assuming J and D are both goal state.

$$A - B - E - J$$

Note: The costs are associated with each edge.

Note: The numeric value beside each node indicates the order in which nodes are visited (reached).

Depth-First Search

The procedure: Depth-first search always expands the deepest node in the current unexplored node set (fringe) of the search tree. The search goes in to depth until there is no more successor node. As these nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next shallowest node (previous level node) that still has unexplored successors.

The implementation: DFS can be implemented with stack (LIFO) data structure which will explore the latest added node first, suspending exploration of all previous nodes on the path. This can be done using recursive procedure that calls itself on each of the children in turn.

The performance evaluation

1. **Completeness:** As DFS explores all the nodes hence it guarantees the solution.

2. **Optimality:** As DFS reach to deepest node first, it may ignore some shallow node which can be goal state. Therefore optimality is expected only when all states have same path cost.
3. **Time and space complexity:** DFS requires some moderate amount of memory as it needs to store single path from root to some node to a particular level, along with unexpanded siblings. When same node gets fully explored, its complete branch (its all descendants and itself) will get removed from memory.

With branching factor ' b ' and maximum depth d, dfs requires storage of - $bd+1$ nodes.

Algorithm DFS (v)

//Given an undirected (OR directed) graph $G = (V, E)$ with

//n vertices and an array visited initially set

// to zero, this algorithm visits all vertices

//reachable from v. G and visited [] are global.

{

visited [v]:= 1;

for each vertex w adjacent from v do

{

if (visited [w] = 0 then DFS (w);

}

}

DFS [Depth First Search]

❖ Time complexity - $O(b^d)$.

❖ Space complexity - $O(b^d + 1)$.

Goal state path - (A - B - D)

where D is a goal state.

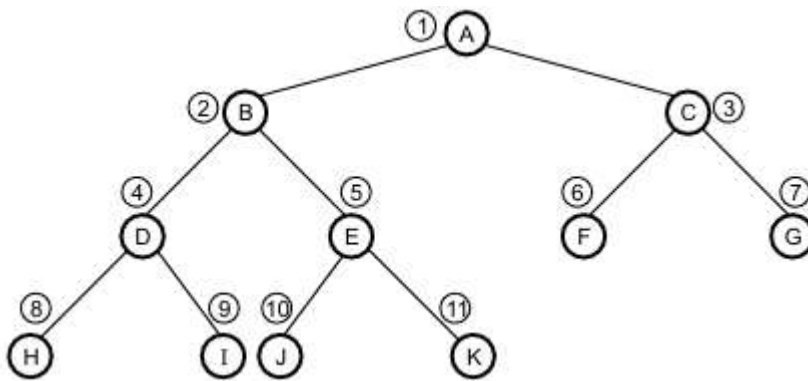


Fig. 1.6 Depth First Search.

Note: The numeric value beside each node indicates the order in which nodes are visited (reached).

A special case DFS → back tracking search: In this only one successor is generated at a time rather than all the successors and each partially expanded node remembers which successor to generate next.

This search technique uses less memory than DFS as only 'd' nodes (maximum depth) are required to store.

Drawback of DFS: DFS can make a wrong choice and get stuck going down a very long (even infinite) path when a different choice would lead to a solution near the root of the search tree.

Depth-Limited Search (DLS)

The procedure: If we can limit the depth of search tree to a certain level then searching will be more efficient. This removes the problem of unbounded trees. Such a kind of depth first search where in the depth is limited at a certain level is called as depth limited search. It solves infinite path problem.

The implementation: Same as DFS but limited to depth level l . DLS will terminate with two kinds of failure. The standard failure value indicates no solution and the cut-off value indicates no solution with in the depth limit.

The performance evaluation

1. **Completeness:** DLS suffers from completeness because if we choose l (levels to be searched) $< d$ (actual levels), when shallowest goal is beyond the depth limit l . It generally happens when 'd' is unknown.
2. **Optimality:** DLS is non-optimal if we choose l (levels to be searched) $> d$ (actual levels) because shallowest goal state may be ignored while reaching to depth level l .

3. **Time and space complexity:** Its time complexity is $O(b^l)$ and space complexity is $O(bl)$. If we have knowledge of the problem, then we can decide depth limits. If we can find better depth limit then we can achieve more efficiency. This better depth limit is termed as diameter of state space. If problem is too complex then we are unable to find diameter of state space.

Algorithm for Depth Limited Search

DLS (node, goal, depth)

```
{  
if(depth >= 0)  
{  
if(node == goal)  
return node  
for each child expand(node)  
DLS (child, goal, depth-1)  
}  
}
```

General steps for DLS

1. Determine the node where the search should start and assign the maximum search depth.
2. Check if the current node is the goal state.
 - ❖ IF not: Do nothing
 - ❖ IF yes: Return
3. Check if the current node is within the maximum search depth.
 - ❖ IF not: Do nothing
 - ❖ IF yes: a) Expand the vertex and save all of its successors in stack.
b) Call DLS recursively for all nodes of the stack and go back to step 2.

DLS [Depth-Limited Search]

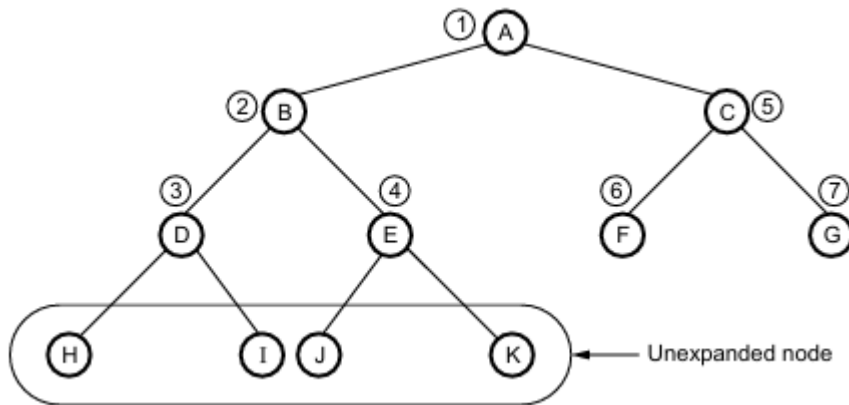


Fig. 1.7 Depth-Limited Search.

If depth limit = 2 then nodes on level 2 are not expanded.
D found (A-B-D)

(D, E, F, G are generated but not expanded).
J though better goal than D could not reached because of algorithm technique.

Note: The numeric value beside each node indicates the order in which nodes are visited (reached).

Iterative-Deepening Depth-First Search (IDDFS)

The procedure: In iterative deepening depth-first search, dfs is applied along with the best depth limit. In each step gradually it increases the depth limit until the goal is found. It increases depth limit from level 0, then level 1, then level 2 till the shallowest goal is found at certain depth 'd'.

The implementation: Iterative deepening depth first search can be implemented similar to BFS (where queue is used for storing fringe) because it explores a complete layer of new nodes at each iteration before going on to the next layer.

If we want to avoid memory requirements which are incurred in BFS then IDDFS can be implemented like uniform-cost search.

The key point is to use increasing path-cost limits instead of increasing depth limits, is if we have such a implementations it is termed as iterative lengthing search.

The performance evaluation

1. **Completeness:** It guarantees completeness as the search does not stop until goal node is found.

2. **Optimality:** As iterative deepening depth first search stops when the first goal node is reached, it is not necessary that it is the optimal. (That is, the shallowest goal reached is the final. Iterative deepening depth first search will not further search for optimal solution).
3. **Time and space complexity:** Iterative deepening depth first search has very moderate space complex which is $O(bd)$.

Its time complexity depends on branching factor (b) and the bottom most level that is depth (d). It is $O(b^d)$.

Algorithm for Iterative-Deepening Depth First Search

//In IDDFS algorithm we are using DLS algorithm from earlier section IDDFS (root, goal)

```
{  
depth = 0  
while (no solution)  
{  
solution = DLS(root, goal, depth)  
depth = depth +1  
}  
return solution  
}
```

Example - IDDFS (Iterative Deepening Depth-First Search]

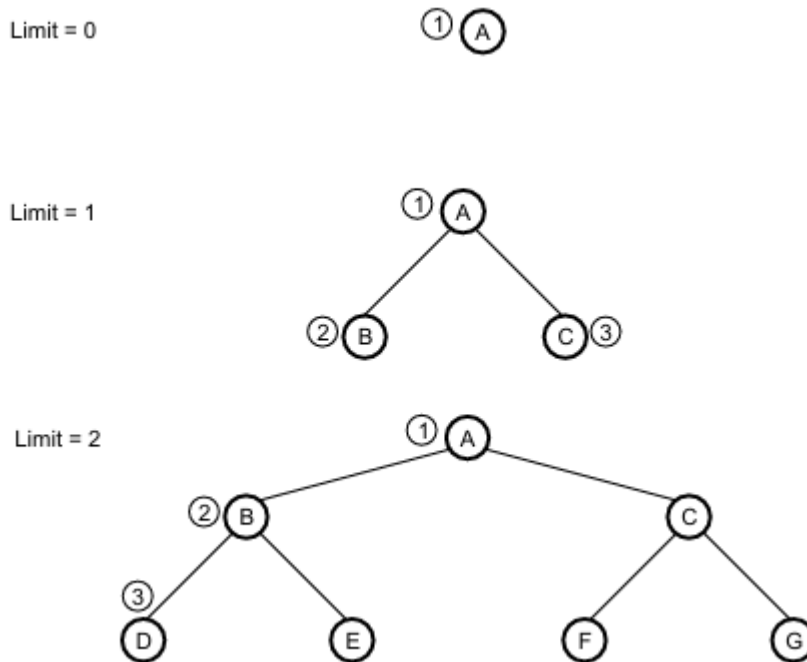


Fig. 1.8 Iterative Deepening Depth-First Search.

Shallowest Goal Node D Found:

Note: The numeric value beside each node indicates the order in which nodes are visited (reached).

Bidirectional Search

The procedure: As the name suggests bi-directional that is two directional searches are made in this searching technique. One is the forward search which starts from initial state and the other is the backward search which starts from goal state. The two searches stop when both the searches meet in the middle. [As shown in below diagram].

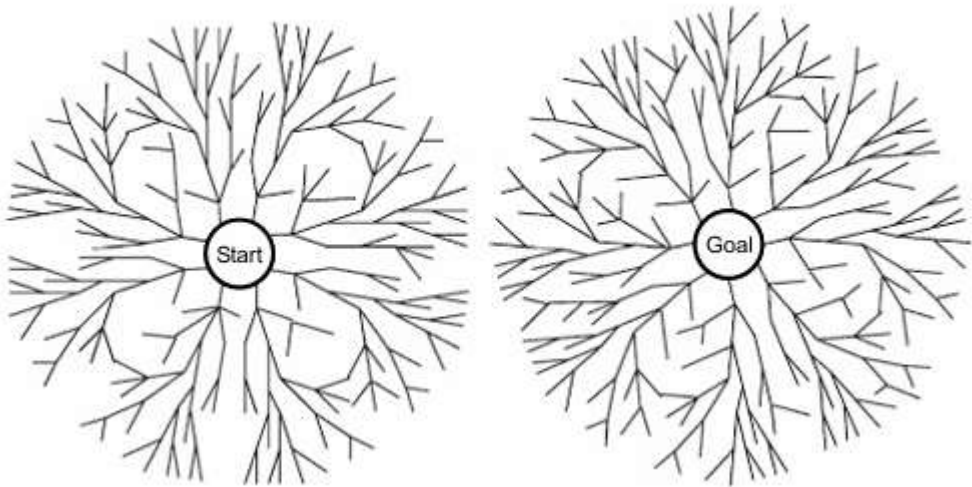


Fig. 1.9 Schematic view of bidirectional search.

The implementation: Bidirectional search is implemented by having one or both of the searches check, each node before it is expanded, is examined to see if it is in the fringe of the other search tree. If so solution is found. Fringe can be maintained in queue data structure, like BFS.

The performance evaluation

1. **Completeness:** Bidirectional search is complete if branching factor **b** is finite and both directions searches use BFS.
2. **Optimality:** It is optimal as the goal is being searched from both directions. So guaranteed to find optimal (best) goal state.
3. **Time and space complexity:** Bidirectional search has time complexity as $O(b^{d/2})$, where 'b' is branching factor. It is the important noticable point in case of backward a search because, we need to get predecessors of the node. The easiest case is when all the actions in the state space are reversible.

When one goal state is there, the backward search is very much like the forward search (like in 8 puzzle problem). If there are several explicitly listed goal states (like in part picking robot) then it needs to construct a new dummy goal state whose immediate predecessors are all then actual goal states. The worst case in bidirectional search is when the goal test gives only implicit description of some possibly large set of goal states.

For example, in the game of chess 'checkmate' is the goal state which will have many possible description.

The memory requirement for bidirectional search is also moderate which is $O(b^{d/2})$ where b = Branching factor, d = Depth, where at least one of the search tree is maintained in memory.

Bidirectional search

- ❖ Space complexity - $O(b^{d/2})$
- ❖ Time complexity - $O(b^{d/2})$

1.5 Strategy for Control

Control strategy is a strategy by which one come to know which rule is to be applied next during the process of reaching for a solution to problem.

Control strategies help to overcome the abnormal situations, when there are more than one rules or fewer than one rule will have its left side match the current state.

A good control strategy must have the following requirements (characteristics).

- ❖ **It should cause motion:** A control strategy should always cause motion as only such control strategy can lead to solution. If there is no motion that means there is no change of state and if state is not changed, then one will never proceed from the initial state. So reaching to a goal state will become a dream or impossible.

For example, in water jug problem if one chooses first rule every time i.e., fill the 4-gallon jug full then one would never solve the problem.

One would continue indefinitely filling the 4 -gallon jug with water. So a control strategy should always cause motion.

- ❖ **It should be systematic:** A control strategy that causes motion but is not systematic is also not good. As in this type of strategy one can reach to a solution but it may pass from one state many times or many explore a particular sequence of operators several times unnecessarily and may use many more steps than are necessary.

1.6 Heuristic Search

Generate and Test

Generate-and-Test is a search algorithm which uses depth first search technique. It assures to find solution in systematic way. It generates complete solution and then the testing is done. A heuristic is needed so that the search is improved.

Following is the algorithm for Generate-and-Test

1. Generate a possible solution which can either be a point in the problem space or a path from the initial state.

2. Test to see if this possible solution is a real (actual) solution by comparing the state reached with the set of goal states.
3. If it is real solution then return the solution otherwise repeat from state 1.

Following diagram illustrates the algorithm steps

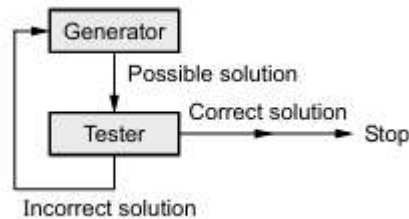


Fig. 1.10 Generate-and-Test.

Generate-and-Test is acceptable for simple problems whereas it is inefficient for problems with large spaces.

Introduction to Informed Search

Informed search strategy is the search strategy that uses problem-specific knowledge beyond the definition of the problem itself.

The general, method followed in informed search is best first search. Best first search is similar to graph search or tree search algorithm where in node expansion is done based on certain criteria.

Best First Search Technique (BFS)

- ❖ Search will start at root node.
- ❖ The node to be expanded next is selected on the basis of an evaluation function, $f(n)$.
- ❖ The node having lowest value for $f(n)$ is selected first. This lowest value of $f(n)$ indicates that goal is nearest from this node (that is $f(n)$ indicates distance from current node to goal node).

Implementation: BFS can be implemented using priority queue where fringe will be stored. The nodes in fringe will be stored in priority queue with increasing value of $f(n)$ i.e. ascending order of $f(n)$. The high priority will be given to node which has low $f(n)$ value.

As name says "best first" then, we would always expect to have optimal solution. But in general BFS indicates that choose the node that appears to be best according to the evaluation function. Hence the optimality based on 'best-ness' of evaluation function.

Algorithm for best first search

- ❖ Use two ordered lists OPEN and CLOSED.
- ❖ Start with the initial node ' n_1 ' and put it on the ordered list OPEN.
- ❖ Create a list CLOSED. This is initially an empty list.
- ❖ If OPEN is empty then exit with failure.
- ❖ Select first node on OPEN. Remove it from OPEN and put it on CLOSED. Call this node n .
- ❖ If ' n ' is the goal node exit. The solution is obtained by tracing a path backward along the arcs in the tree from ' n ' to ' n_1 '.
- ❖ Expand node ' n '. This will generate successors. Let the set of successors generated, be S . Create arcs from ' n ' to each member of S .
- ❖ Reorder the list OPEN, according to the heuristic and go back to step 4.

Consider the following 8-puzzle problem

Here the heuristic used could be "number of tiles not in correct position" (i.e. number of tiles misplaced). In this solution the convention used is, that smaller value of the heuristic function ' f ' leads earlier to the goal state.

Step 1

$$f(n) = 4$$

2	8	3
1	6	4
7	*	5

This is the initial state where four tiles (1,2,6,8) are misplaced so value of heuristic function at this node is 4.

Step 2

This will be the next step of the tree.

$$f(n) = 3$$

2	8	3
1	*	4
7	6	5

$$f(n) = 5$$

2	8	3
1	6	4
7	5	*

$$f(n) = 5$$

2	8	3
1	6	4
*	7	5

Heuristic function gives the values as 3, 5 and 5 respectively.

Step 3

Next, the node with the lowest $f(n)$ value 3 is the one to be further expanded which generates 3 nodes.

$$f(n) = 3$$

2	*	3
1	8	4
7	6	5

$$f(n) = 4$$

2	8	3
1	4	*
7	6	5

$$f(n) = 3$$

2	8	3
*	1	4
7	6	5

Step 4

Here there is tie for $f(n)$ value. We continue to expand node.

$$f(n) = 2$$

*	2	3
1	8	4
7	6	5

$$f(n) = 4$$

2	3	*
1	8	4
7	6	5

Step 5

$$f(n) = 1$$

1	2	3
*	8	4
7	6	5

Step 6

Goal state

1	2	3
8	*	4
7	6	5

$$f(n) = 2$$

1	2	3
7	8	4
*	6	5

In this algorithm a depth factor g is also added to h .

Heuristic Function

There are many different algorithms which employ the concept of best first search. The main difference between all the algorithms is that they have different evaluation function. The central component of these algorithms is heuristic function - $h(n)$ which is defined as, $h(n)$ = Estimate cost of the cheapest path from node 'n' to a goal node.

Note

- ❖ Heuristic function is key component of best first search. It is denoted by $h(n)$ and $h(n)$ = The shortest and cheapest path from initial node to goal node.
- ❖ One can give additional knowledge about the problem to the heuristic function.
For example - In our problem of Pune-Chennai route we can give information about distances between cities to the heuristic function.
- ❖ A heuristic function guide the search in an efficient way.
- ❖ A heuristic function $h(n)$ consider a node as input but it rely only on the state at that node.
- ❖ $h(n) = 0$, if n is goal state.

Greedy Best First Search (GBFS)

Greedy best first search expand the node that is closest to the goal, expecting to get solution quickly,

It evaluates node by using the heuristic function $f(n) = h(n)$.

It is termed as greedy (asking for more) because at each step it tries to get as close to the goal as it can.

GBFS resembles DFS in the way that it prefers to follow a single path all the way to the goal. It back tracks when it comes to dead end that is, to a node from which goal state cannot be reached.

Choosing minimum $h(n)$ can lead to bad start as it may not yield always a solution. Also as this exploration is not leading to solution, therefore unwanted nodes are getting expanded.

In GBFS, if repeated states are not detected then the solution will never found.

Performance measurement

1. Completeness: It is incomplete as it can start down an infinite path and never return to try other possibilities which can give solution.
2. Optimal: It is not optimal as it can initially select low value $h(n)$ node but it may happen that some greater value node in current fringe can lead to better solution. Greedy strategies, in general, suffers from this, "looking for current best they loose on future best and in turn finally the best solution" !
3. Time and space complexity: Worst case time and space complexity is $O(b^m)$ where ' m ' is the maximum depth of the search space.

The complexity can be reduced by devising good heuristic function.

Condition at which A^* becomes, i) BFS, ii) Best First Search and iii) No Search is Required

According to A^* algorithm

- ❖ The heuristic function that estimates the merits of node we generate. This will enable the algorithm to search more promising path first.
- ❖ Call this function f' (to indicate that it is an approximation to a function f that gives the true evaluation of the node).
- ❖ For many applications, it is convenient to define this function as the sum of two components that we call g and h' .
- ❖ The function g is a measure of the cost of getting from the initial state to the current node. Note that g is not an estimate of anything it is known to be the exact sum of the cost of applying each of the rules that were applied along the best path to the node.
- ❖ The function h' is an estimate of the additional cost of getting from the current node to a goal state.
- ❖ This is the place where knowledge about the problem domain is exploited.
- ❖ The combined function f' , then represents an estimate of the cost of getting from the initial state to a goal state along the path that generated the current node.
- ❖ There are conditions through which algorithm A^* will become -
 - i) Breadth-first search
 - ii) Best-first search
 - iii) No search is required.

i) Breadth-first search - If the value of g is always 1, then the search will be known as breadth-first search.

ii) Best-first search - If the value of h' is 0, then the search will be controlled by g , then the search is known as best first.

iii) No search is required - If h' is a perfect estimator of h then A^* will converge immediately to the goal without searching. Hence, search is not required.

Memory Bounded Heuristic Search

If we use idea of Iterative deepening search then we can reduce memory requirements of A^* . From this concept we device new algorithm.

Iterative deepening A^*

- ❖ Like IDDFS uses depth as cutoff value in IDA* f -cost ($g + h$) is used as cutoff rather than the depth.

- ❖ It reduces memory requirements, incurred in A^* , thereby putting bound on memory hence it is called as memory bounded algorithm.
- ❖ IDA* suffers from real value costs of the problem.
- ❖ We will discuss two memory bounded algorithm -
 1. Recursive breadth first search.
 2. MA* (Memory bounded A^*).

Recursive Best First Search (RBFS)

- ❖ It works like best first search but using only linear space.
- ❖ Its structure is similar to recursive DFS but instead of continuing indefinitely down the current path it keeps track of the F value of the best alternative path available from any ancestor of the current node.
- ❖ The recursion procedure is unwinded back to the alternative path if the current node crosses limit.
- ❖ The important property of RBFS is that it remembers the f-value of the best leaf in the forgotten subtree (previously left unexpanded). That is, it is able to take decision regarding re-expanding the subtree.
- ❖ It is reliable, cost effective than IDA but its critical problem is excessive node generation.

Performance measure

1. **Completeness:** It is complete.
2. **Optimality:** Recursive best-first search is optimal if $h(n)$ is admissible.
3. **Time and space complexity:** Time complexity of RBFS depends on two factors -
 - a) Accuracy of heuristic function.
 - b) How often (frequently) the best path changes as nodes are expanded.

RBFS suffers from problem of expanding repeated states, as the algorithm fails to detect them.

It's space complexity is $O(bd)$.

It's suffers from problem of using too little (very less) memory. Between iterations, RBFS maintains more information in memory but it uses only $O(bd)$ memory. If more memory is available then also RBFS cannot utilize it.

MA*

RBFS under utilizes memory. To overcome this problem MA* is devised.

Its more simplified version called as simplified MA* proceeds as follows -

- ❖ If expands the best leaf until memory is full.
- ❖ At this point it cannot add new node to the search tree without dropping an old one.
- ❖ It always drops the node with highest f-value.
- ❖ If goal is not reached then it backtracks and go to the alternative path. In this way the ancestor of a forgotten subtree knows the quality of the best path in that subtree.
- ❖ While selecting the node for expansion it may happen that two nodes are with same f-value. Some problem arises, when the node is discarded (multiple choices can be there as many leaves can have same f-value).

The SMA* generates new best node and new worst node for expansion and deletion respectively.

Performance measurement

1. **Completeness:** SMA* is complete and guarantee solution.
2. **Optimality:** If solution is reachable through optimal path it gives optimal solution. Otherwise it returns best reachable solution.
3. **Time and space complexity:** There is memory limitations on SMA*. Therefore it has to switch back and forth continually between a set of candidate solution paths, only small subset of which can fit in memory. This problem is called as thrashing because of which algorithm takes more time.

Extra time is required for repeated regeneration of the same nodes, means that the problem that would be practically solvable by A* (with unlimited memory) became intractable for SMA*.

It means that memory restrictions can make a problem intractable from the point of view of computation time.

Problem Reduction with AO* Algorithm (AND-OR Graphs Algorithm)

When a problem can be divided in a set of sub problems, where each sub problem can be solved separately and a combination of these will be a solution. AND-OR graphs or AND-OR trees are used for representing the solution.

AND-OR graphs

- ❖ The decomposition of the problem or problem reduction generates AND arcs.
- ❖ One AND arc may point to any number of successor nodes.
- ❖ All these must be solved so that the arc will give rise to many arcs, indicating several possible solutions. Hence the graph is known as AND-OR instead of AND.
- ❖ AO* is a best-first algorithm for solving problems represented as a cyclic AND/OR

graphs problems.

- ❖ An algorithm to find a solution in an AND-OR graph must handle AND area appropriately.

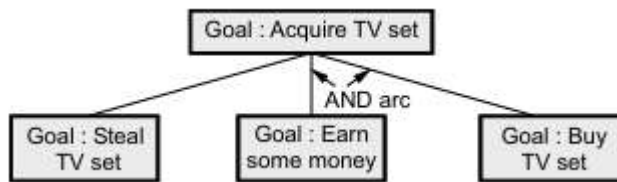


Fig. 1.11 AND-OR graph example.

The comparative study of A* and AO*

- ❖ Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G.
- ❖ G represents the part of the search graph generated so far.
- ❖ Each node in G points down to its immediate successors and upto its immediate predecessors, and also has with it the value of 'h' cost of a path from itself to a set of solution nodes.
- ❖ The cost of getting from the start nodes to the current node 'g' is not stored as in the A* algorithm. This is because it is not possible to compute a single such value since there may be many paths to the same state.
- ❖ AO* algorithm serves as the estimate of goodness of a node.
- ❖ A* algorithm cannot search AND-OR graphs efficiently.
- ❖ AO* will always find minimum cost solution.

The algorithm for performing a heuristic search of an AND-OR graph is given below.

AO* algorithm

- ❖ Initialize the graph to start node.
- ❖ Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved.
- ❖ Pick any of these nodes and expand it and if it has no successors call this value as, FUTILITY, otherwise calculate only 'f' for each of the successors.
- ❖ If 'f' is 0 then mark the node as SOLVED.
- ❖ Change the value of 'f' for the newly created node to reflect its successors by back propagation.

- ❖ Wherever possible use the most promising routes and if a node is marked as SOLVED then mark the parent node as SOLVED.
- ❖ If starting node is SOLVED or value greater than FUTILITY, stop, else repeat from 2.

How to Search Better?

We have seen many searching strategies till now, but as we can see no one is really the perfect. How can we make our AI agent to search better?

We can make use of a concept called as meta-level state space. Each state in a meta-level state space captures the internal (computational) state of a program that is searching in an object-level state space such as in Indian Traveller Problem.

Consider A* algorithm

- ❖ In A* algorithm it maintains internal state which consists of the current search tree.
- ❖ Each action in the metalevel state space is computation step that alters the internal state.

For example - Each computation step in A* expands a leaf node and adds its successors to the tree.

- ❖ As the level increases the sequence of larger and larger search trees is generated.

In harder problem there can be mis-steps taken by algorithm. That is algorithm can explore unpromising unuseful subtrees. Metalearning algorithm overcomes these problems.

The main goal of metalearning is to minimize the total cost of problem solving.

Heuristic Functions and Their Nature

Accuracy of Heuristic Function

More accurate the heuristic function more is the performance.

The quality of heuristic function can be measured by the effective branching factor b^* .

Consider A* that generates N nodes and 'd' is depth of a solution, then b^* is the branching factor that a uniform tree of depth 'd' would have so as to contain 'N + 1' nodes.

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

If A* finds a solution at depth 5 using 52 nodes, then the effective branching factor is 1.92.

A well designed heuristic function will have a value of b^* close to 1 , allowing fairly large problems to be solved.

An Example of Designing Heuristic Function

Consider 8-puzzle problem

The objective of the puzzle is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration.

Start state

7	2	4
5		6
8	3	1

Goal state

	1	2
3	4	5
6	7	8

- ❖ The average solution cost for a randomly generated 8 -puzzle instance is about 22 steps.
- ❖ The branching factor is about 3 (when the empty tile is in the middle, there are four possible moves, when it is in a corner there are two, and when it is along an edge there are three).
- ❖ An exhaustive search to depth 22 would look at about $3^{22} \approx 3.1 \times 10^{10}$ states.
- ❖ If we keep track of repeated states, we could cut this down by a factor of about 1,70,000, because there are only $9!/2 = 1,81,440$ distinct states that are reachable.
- ❖ If we use A^* , we need a heuristic function that never overestimates the number of steps to the goal.
- ❖ We can use following heuristic function,
 - h_1 = The number of misplaced tiles. All of the eight tiles are out of position, so the start state would have $h_1 = 8$ · h_1 , is an admissible heuristic, because it is clear that any tile that is out of place must be moved at least once.
 - h_2 = The sum of the distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of the horizontal and vertical distances.

The Domination of Heuristic Function

If we could design multiple heuristic function for same problem then we can find that which one is better.

Consider two heuristic functions h_1 and h_2 . From their definitions for some node n , if $h_2(n) \geq h_1(n)$ then we say that h_2 dominates h_1 .

Domination directly points to accuracy. A^* using h_2 will never expand more nodes than A^* using h_1 (except for some node having $f(n) = c^*$).

Admissible Heuristic Functions

- ❖ A problem with fewer restrictions on actions is called a relaxed problem.
- ❖ The cost of an admissible solution to a relaxed problem is an admissible heuristic for the original problem. The heuristic function is admissible because the optimal solution in the original problem is, also a solution in the relaxed problem, and therefore must be at least as expensive as the optimal solution in the relaxed problem.
- ❖ As the derived heuristic is an exact cost for the relaxed problem, therefore it is consistant.
- ❖ If problem definitions are written in formal languages then it is possible to construct relaxed problem automatically.
- ❖ Admissible heuristic function can also be derived from the solution cost of a subproblem of the given problem. The cost of optimal solution of this subproblem would be the lower bound on the cost of the complete problem.
- ❖ We can store the exact solution costs for every possible subproblem instance in a database. Such a database is called as pattern database.

The pattern database is constructed by searching backwards from the goal state and recording the cost of each new pattern encountered.

- ❖ The cost of the entire problem is always greater than sum of cost of two subproblems. Hence it is always better to derive disjoint solution and then sum up all solutions to minimize the cost.

Learning Heuristic from Experience

- ❖ A heuristic function $h(n)$ is supposed to estimate the cost of a solution begining from the state at node n . Therefore it is really difficult to design $h(n)$.
- ❖ One solution is to devise relaxed problems for which an optimal solution can be found.
- ❖ Another solution is to make agent program that can learn from experience.

- "Learn from experience" means solving similar problem again and again (i.e. practicing).
- Each optimal solution will provide example from which ' $h(n)$ design' can be learned.
- One can get experience of which $h(n)$ was better.
- Each example consists of a state from the solution path and the actual cost of the solution from that point.
- From such solved examples an inductive learning algorithm can be used to construct the function $h(n)$ that can predict solution costs for another states that have arised during search. [A lucky agent program will get the prediction early.]
- For developing inductive algorithms we can make use of techniques like neural nets, decision trees, etc.
- If inductive learning methods have knowledge about features of a state that are relevant to evaluation of algorithms then inductive learning methods give best output.

1.7 Hill Climbing Search

This algorithm generally moves up in the direction of increasing value that is-uphill. It breaks its “moving up loop” when it reaches a “peak” where no neighbour has a higher value.

It does not maintain a search tree. It stores current node data structure. This node records the state and its objective function value. Algorithm only look out for immediate neighbours of current state.

It is similar to greedy local search in a sense that it considers a current good neighbour state without thinking ahead.

Greedy algorithm works very well as it is very easy to improve bad state in hill climbing.

Algorithm for Hill Climbing

The algorithm for hill climbing is as follows:

- ❖ Evaluate the initial state. If it is goal state quit, otherwise make current state as initial state.
- ❖ Select a new operator that could be applied to this state and generate a new state.
- ❖ Evaluate the new state. If this new state is closer to the goal state than current state make the new state as the current state. If it is not better, ignore this state and proceed with the current state.

- ❖ If the current state is goal state or no new operators are available, quit. Otherwise repeat from 2.

Problems with Hill Climbing

1. Local maxima - can't see higher peak.
- 2) Shoulder - can't see the way out.

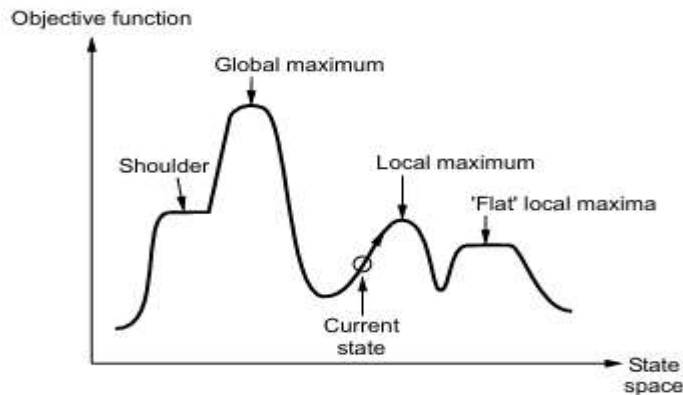


Fig. 1.12 Various stages in hill climbing search.

Local maxima - It is a state where we have climbed to the top of the hill, and missed on better solution.

It is the mountain - A state that is better than all of its neighbours, but not better than some other states further away. [Shown in **Fig. 1.13**]



Fig. 1.13 Local maxima.

Plateau: It is a state where everything around is about as good as where we are currently. In other words a flat area of the search space in which all neighbouring states have the same value. [Shown in **Fig. 1.14**]

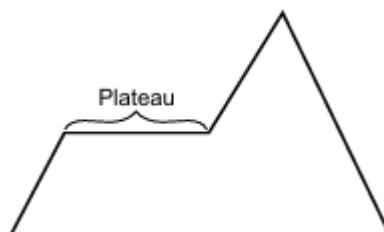


Fig. 1.14 Plateau.

Ridges: In this state we are on a ridge leading up, but we can't directly apply an operator to improve the situation, so we have to apply more than one operator to get there. [Shown in Fig. 1.15]

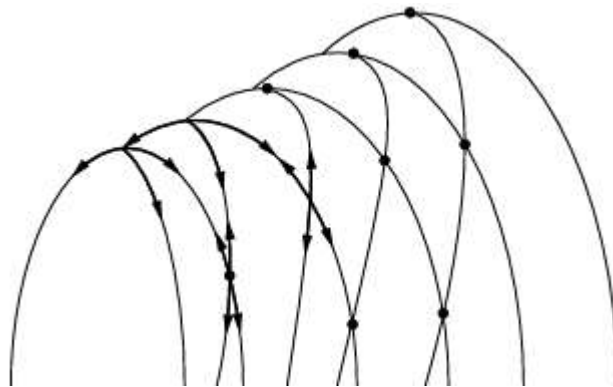


Fig. 1.15 Ridges.

Illustration of ridges: The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequences of local maxima that are not directly connected to each other. From each local maximum all the available actions point downhill.

Solving Problems Associated with Hill Climbing

All the above discussed problems could be solved using methods like backtracking, making big jumps (to handle plateaus or poor local maxima), applying multiple rules before testing (helps with ridges) etc. Hill climbing is best suited to problem where the heuristic gradually improves, the closer it gets to the solution; it works poorly where there are sharp drop-offs. It assumes that local improvement will lead to global improvement.

Example for Local Search

Consider the 8-queens problem

A complete-state formulation is used for local search algorithms. In 8-queens problem, each state has 8 -queens on the board one per column. There are two functions related with 8 -queens.

1. **The successor function:** It is function which returns all possible states which are generated by a single queen move to another cell in the same column. The total successor of the each state $8 \times 7 = 56$.
2. **The heuristic cost function:** It is a function ' h ' which hold the number of attacking pair of queens to each other either directly or indirectly. The value is zero for the global minimum of the function which occurs only at perfect solutions.

Advantages of Hill Climbing

- ❖ Hill climbing is an optimization technique for solving computationally hard problems.
- ❖ It is best used in problems with the property that the "state description itself contains all the information needed for a solution".
- ❖ The algorithm is memory efficient since it does not maintain a search tree. It looks only at the current state and immediate future states.
- ❖ In contrast with other iterative improvement algorithms, hill-climbing always attempts to make changes that improve the current state. In other words, hill-climbing can only advance if there is a higher point in the adjacent landscape.
- ❖ It is often useful when combined with other methods, getting it started right in the immediate general neighbourhood.

Variations of Hill Climbing

Many variants of hill-climbing have been invented as discussed below.

- ❖ **Stochastic hill climbing:** Chooses at random from among the uphill moves; the probability of selections can vary with the steepness of the uphill move.
- ❖ **First choice hill climbing:** Implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g. thousands) of successors.
- ❖ **Random restart hill climbing:** Adopts the well known saying "If at first you don't succeed, try, try again". It conducts a series of hill climbing searches from randomly generated initial states, stopping when a goal is found.

The hill climbing algorithms described so far are incomplete because they often fail to find a goal when surely goal exist. This can happen because these algorithms can get stuck on local maxima. Random restart hill is complete with probability approaching to 1. This algorithm do not stop until it reaches to goal.

The success of hill climbing depends very much on the shape of the state-space landscape. If there are few local maxima and plateaux, random-restart hill climbing will find a good solution very quickly.

- ❖ **Steepest ascent hill climbing:** This algorithm differs from the basic hill climbing algorithm by choosing the best successor rather than the first successor that is better. This indicates that it has elements of the breadth first algorithm.

Steepest ascent hill climbing algorithm

- ❖ Evaluate the initial state.

- ❖ If it is goal state then quit otherwise make the current state this initial state and proceed.
- ❖ Repeat set target to be the state that any successor of the current state can better; for each operator that can be applied to the current state apply the new operator and create a new state evaluate this state.
- ❖ If this state is goal state then quit. Otherwise compare with target. If better set target to this value. If Target is better than current state set current state to target. Until a solution is found or current state does not change.

Both the basic and this method of hill climbing may fail to find a solution by reaching a state from which no subsequent improvement can be made and this state is not the solution. Local maximum state is a state which is better than its neighbours but is not better than states faraway. These are often known as foothills. Plateau states are states which have approximately the same value and it is not clear in which direction to move in order to reach the solution. Ridge states are special types of local maximum states. The surrounding area is basically unfriendly and makes it difficult to escape from, in single steps, and so the path peters out when surrounded by ridges. Escape relies on: backtracking to a previous good state and proceed in a completely different direction involves keeping records of the current path from the outset; making a gigantic leap forward to a different part of the search space perhaps by applying a sensible small step repeatedly, good for plateau; applying more than one rule at a time before testing, good for ridges. None of these escape strategies can guarantee success.

1.8 Issues with Constraint Satisfaction

Constraint satisfaction problems are problems whose states and goal test conform to a standard, structured and very simple representation. Search algorithm can be defined that take advantage of the structure of states and can use general-purpose rather than problem-specific heuristics to enable the solution of large problems.

Constraint Satisfaction Problems - The Concept

- ❖ Constraint satisfaction problem has various states and goal test, a traditional problem has been converted into standard structured and very simple "representation".
- ❖ The general-purpose routines can be used to access a special representation which has more benefit than problem-specific heuristics. These routines combined with special structure can find solution of large problems.
- ❖ The structure of the problem is represented in different form such as, standard representation of the goal test.
- ❖ The revealed structured is very efficient in many ways such as,

- Problem decomposition.
- To understand structure of problem and the difficulty of solving it and their connection.
- ❖ If the problem is treated as CSP we have many advantages, as discussed below.
 - As the representation of states have standard pattern [that is a set of variables with assigned values], we can design successor function and goal test in generic way that will apply to all CSPs.
 - We can develop effective generic heuristic that require no additional domain specific expertise.
 - The structure of the constraint graph can be used to simplify the solution process in some cases giving exponential reduction in complexity.

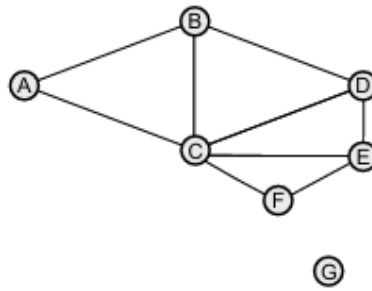
Formal Definition of Constraint Satisfaction Problems

- ❖ A constraint satisfaction problem is defined by a set of variables $X_1, X_2, \dots, X_n$ and a set of constraints $C_1, C_2, \dots, C_m$.
 - ❖ Each variable X_1 has a nonempty domain D_1 of possible values.
 - ❖ Each constraint C_1 involves some subset of the variables and specifies the allowable combinations of values for that subset.
 - ❖ A state of the problem is defined by an assignment of values to some or all of the variables $\{X_i = V_i, X_j = V_j, \dots\}$.
 - ❖ An assignment that does not violate any constraints is called a consistent or legal assignment.
 - ❖ A complete assignment is one in which every variable is maintained and a solution to a CSP is, a complete assignment that satisfies all the constraints.
4. Some CSPs also require a solution that maximizes an objective function.

Consider following graph colouring problem. Constraints are -

1. We have three colours for colouring a vertex.
2. No two adjacent vertices have same colour.

Given three colours-(Red, Green, Blue). Allowable combination for A, B vertices would be, $\{(R, G), (R, B), (G, R), (G, B), (B, R), (B, G)\}$



Examples of CSP

Map Colouring Problem

- ❖ The problem is to colour the regions of a given map such that no 2 adjacent regions have the same colour. [Refer **Fig. 1.17**]
- ❖ The regions in the map are the variables and the set of possible colours for the regions is the domain.
- ❖ The constraint is that "no two adjacent regions should have the same colour."

Formal Representation of Map Colouring Problem

1. **Variables:** {N, NW, NE, M, MW, ME, S, SE}
2. **Domains:** $D = \{\text{red, green, blue}\}$
3. **Constraints:** Adjacent regions must have different colors.

Example: $N \neq NW$

Note:

- ❖ Typically, such a problem has many solutions.
- ❖ We sometimes represent map colouring as a graph coloring (constraint graph) problem.
- ❖ The topology of a constraint graph can sometimes be used to identify solutions easily.

Other Examples of CSP

1. N-queens puzzle.
2. Job shop scheduling.
3. Scene labelling.
4. Circuit board layout.
5. Map colouring problem.
6. Sudoku

7. Boolean satisfiability.

Some real-world problems

1. Assignment problems.
2. Transportation scheduling.
3. Hardware configuration.
4. Spreadsheets.
5. Factory scheduling.
6. Floor planning.

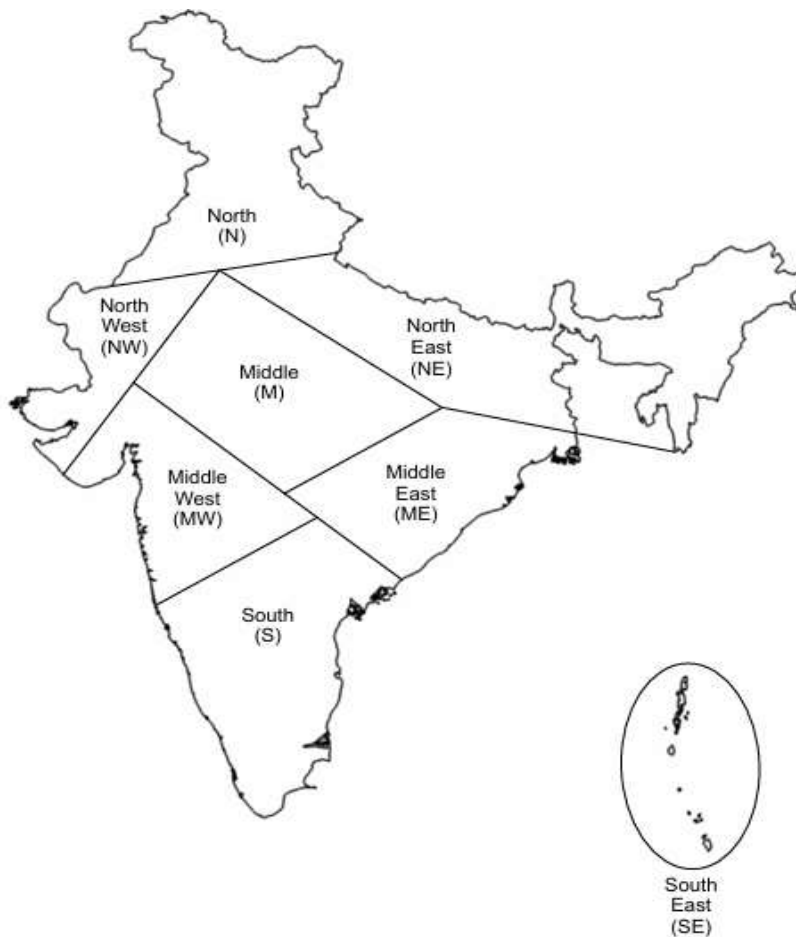


Fig. 1.17 Map colouring problem.

Incremental Formulation for CSP

It is fairly easy to see that a CSP can be given an incremental formulation as a standard search problem as follows,

i) Initial state

The empty assignment {}, in which all variables are unassigned.

ii) Successor functions

A value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.

iii) Goal test

The current assignment is complete.

iv) Path cost

A constant cost for every step.

Searching for Goal State in CSP

1. Every solution must be a complete assignment and therefore appears at depth n if there are n variables.
2. The search tree extends only to depth n .
3. Depth-first search algorithms are popular for CSPs.
4. The path by which a solution is reached is irrelevant.
5. We can also use a complete-state formulation, in which every state is a complete assignment that might or might not satisfy the constraints.
6. Local search methods work well for this formulation.

Variations in CSPs

- ❖ The simplest kind of CSP involves variables that are discrete and have finite domains. Graph-coloring problems are of this kind. The 8 -queens problem described can also be viewed as finite-domain CSP, where the variables $Q_1, \dots, Q_8$ are the positions of each queen in columns 1, ..., 8 and each variable has the domain $\{1, 2, 3, 4, 5, 6, 7, 8\}$. If the maximum domain size of any variable is a CSP in d , then the number of possible complete assignments are $O(d^n)$ that is exponential in the number of variables.
- ❖ Finite-domain CSPs include Boolean CSPs, whose variables can be either true or false. Boolean CSPs include, as special cases, some NP-complete problems, such as 3SAT.
- ❖ In most practical applications, however, general-purpose CSP algorithms can solve problems of orders of magnitude larger than those solvable via the general-purpose search algorithms that we saw.

- ❖ Discrete variables can also have infinite domains-for example, the set of integers or the set of strings.
- ❖ Constraints satisfaction problems with continuous domains are very common in the real world and are widely studied in the field of operations research.

For example, the scheduling of experiments on the Hubble Space Telescope requires very precise timing of observations; the start and finish of each observation and man ever are continuous-valued variables that must obey a variety of astronomical, precedence and power constraints. The best-known category of continuous-domain CSPs is that of linear programming problems where constraints must be linear inequalities.

Constraints in CSPs

Properties of Constraints

Constraints are used to guide reasoning of everyday common sense. The constraints have following properties.

- ❖ Constraints may specify partial information; constraint need not uniquely specify the values of its variables.
- ❖ Constraints are non-directional, typically a constraint on (say) two variables V_1 , V_2 can be used to infer a constraint on V_1 given a constraint on V_2 and vice versa.
- ❖ Constraints are declarative; they specify what relationship must hold without specifying a computational procedure to enforce that relationship.
- ❖ Constraints are additive; the order of imposition of constraints does not matter, all that matters at the end is that the conjunction of constraints is in effect.
- ❖ Constraints are really independent; typically constraints in the constraint store (i.e. collection of constraints) share variables.

Types of Constraints in CSPs

Unary constraint

Which restricts the value of a single variable. Every unary constraint can be eliminated simply by preprocessing the domain of the corresponding variable to remove any value that violates the constraint.

For example - Constraint can be that, Vertex A can't be coloured with blue colour.

Binary constraint

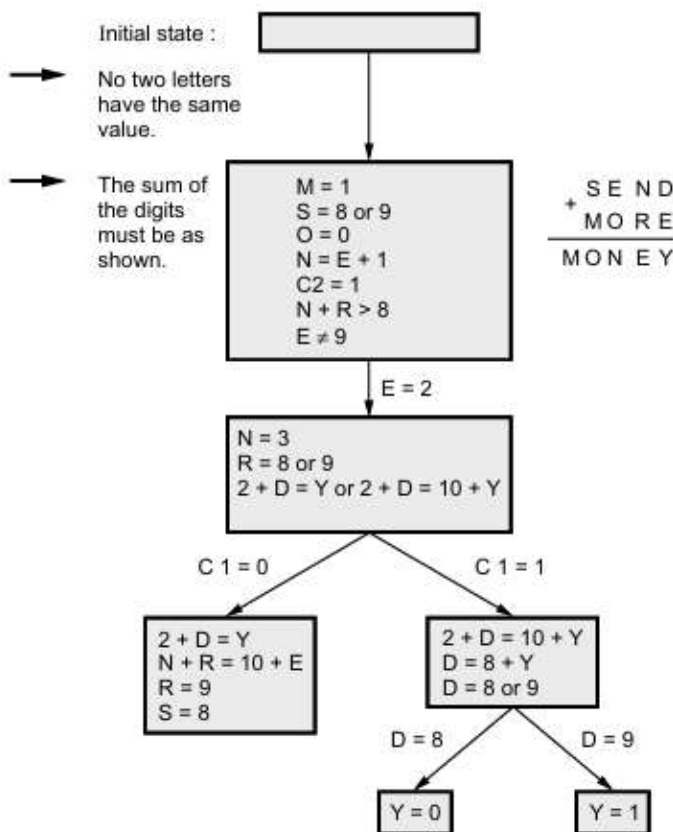
Relates two variables. A binary CSP is one with only binary constraints; it can be represented as a constraint graph.

For example - In graph colouring problem two adjacent vertices can't have same colour.

Higher-order constraints

Involves three or more variables. A familiar example is provided by crypto arithmetic puzzles. It insists that each letter in a crypto arithmetic puzzle represent a different digit. Higher-order constraints can be represented in a constraint hypergraph. Such as shown below,

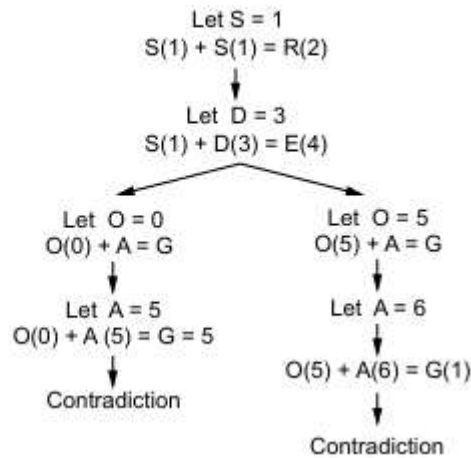
Example 1 - The crypto arithmetic problem. (A CSP problem having high order constraints)



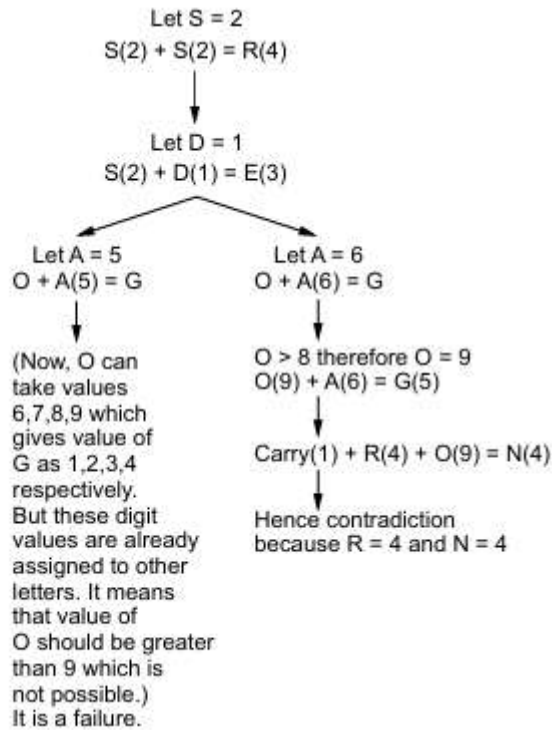
Example 2 - Explain the constraint satisfaction procedure to solve the crypto arithmetic problem.

Solution: (At each step minimum possible value is selected.)

Solution (i)



Solution (ii)



Solution (iii)

Now let $S = 3$
 $S(3) + S(3) = R(6)$
 $\downarrow$
 Let $D = 1$
 $S(3) + D(1) = E(4)$
 $\downarrow$
 Let $O = 2$
 $A + O(2) = G$
 $\downarrow$
 Let $A = 5$
 $\downarrow$
 $A(5) + O(2) = G(7)$
 $\downarrow$
 $R(6) + O(2) = N(8)$
 $\downarrow$
 Let $C = 9$
 $C(9) + R(6) = A(5)$

Solution is,

	carry = 0	carry = 0	carry = 0	carry = 0	
	C(9)	R(6)	O(2)	S(3)	S(3)
	R(2)	O(2)	A(5)	D(1)	S(3)
D(1)	A(5)	N(8)	G(7)	E(4)	R(6)

General Algorithm for Finding Solution in CSP

- ❖ Propagate available constraints
 - Open all objects that must be assigned values in a complete solution.
 - Repeat until inconsistency or all objects are assigned valid values: Select an object and strengthen as much as possible, the set of constraints that apply to object. If set of constraints are different from previous set, then open all objects that share any of these constraints. Remove selected object.
- ❖ If union of constraints discovered above defines a solution then return solution.
- ❖ If union of constraints discovered above defines a contradiction then return failure.
- ❖ Make a guess in order to proceed. Repeat until a solution is found or all possible solutions exhausted;
 - Select an object with a no assigned value and try to strengthen its constraints.
 - Recursively invoke constraint satisfaction with the current set of constraints plus the selected strengthening constraint.

CSP Representation as a Constraint Graph

The CSP can be represented in terms of the constraint graph. A constraint graph has,

- ❖ Nodes as variables (For example - In graph-colouring problem the vertex is variable which needs to be coloured. This vertex will be a node in constraint graph.)
- ❖ Arcs as constraints (For example - In graph-colouring problem the arc will denote that no two adjacent vertices will have same colour.)

Advantages of Constraint Graph

- ❖ The organization of constraint graph is very much useful for simplification of CSP's solutions.
- ❖ It reduces complexity in exponential manner.

Backtracking Search

If we apply BFS to generic CSP problems then we can notice that the branching factor at the top level is ' d ', because any of ' d ' values can be assigned to any of ' n ' variables. At the next level, the branching factor is ' $(n - 1)d$ ' and so on for n levels. Therefore a tree with ' $n! * d^n$ ' leaves is generated even though there are only ' d^n ' possible complete assignments.

Basic Steps in Backtracking Search

- ❖ Depth-first search selects values for single variable at a given time.
- ❖ Apply constraint to the variable when variable is selected.
- ❖ Backtracking action-performed if a variable has no legal values left to assign.
- ❖ Backtracking search is basic uninformed algorithm for CSPs.

Backtracking Search Algorithm

- ❖ Consider a CSP problem.
 - ❖ Apply backtracking search. If backtracking search successful returns a solution else, a failure state which return a procedure recursive-backtracking.
 - ❖ Procedure recursive-backtracking starts with empty set and takes input as CSP problem.
If complete assignment possible or assignment done then return actual assignment.
 - ❖ Variable is assigned a specific value.
 - ❖ The relative constraint is a set which is taken as input.
 - ❖ If value is complete and consistent according to constraints then assign value to variable, add that to list.
 - ❖ Call the recursive backtracking until result or failure is reached.
-

- ❖ Every time recursive-backtracking sustains result (i.e. assignment.)
- ❖ If result is failure then remove variable with specific value from assignment. It will return failure status.

Limitations of Backtracking

- ❖ A success function is generated with above algorithm. But backtracking is not effective for very large problems.
- ❖ The general performance is not so good. Domain specific heuristic function combined with uninformed search algorithm can generate better searching result.

Improvement in Backtracking Search

We can also solve CSP without knowing the domain-specific knowledge. Here we need to design a module which resolve following queries:-

- ❖ Which variable is assigned in next step and what order values can be tried?
- ❖ Impact of current variable assignments on unassigned variables.
- ❖ Avoidance of known failed path in the potential paths.

Commutative Problem

A problem is commutative if the order of application of any given set of actions has no effect on the outcome. This is the case for CSPs because, when assigning values to variables, we reach the same partial assignment, regardless of order. Therefore, all CSP search algorithms generate successors by considering possible assignments for only a single variables at each node in the search tree.

For example -

In graph-colouring problem, if vertex 1 is coloured with red then vertex 2 (adjacent to vertex 1) will have two choices either green or blue but never red.

Selection of Next Unassigned Variable

Any general purpose method for solving CSP suffers from making a choice of next unassigned variable.

Variable and Value Ordering

- ❖ Choosing a variable is critical to performance.
- ❖ The efficiency of search algorithms depends considerably on the order in which variables are considered for instantiations.
- ❖ This ordering affects the efficiency of the algorithm.

- ❖ There exist various heuristics for dynamic or static ordering of values and variables.

Techniques for selecting next best unassigned variable

1. Minimum Remaining Values (MRV)
2. It is also called as "Most constrained variable or fail-first heuristic".
3. Backtracking combined with MRV gives better performance.
4. Rule: Choose a variable with the fewest legal moves.
5. It answers-Which variable shall we try first?

Degree heuristic

6. It helps to choose next better state.
7. Rule: Select variable involved in highest number constraints on other unassigned variable.
8. Degree heuristic is very useful as a tie breaker among MRV variable.
9. It answers-In what order should variable values can be tried?

Least constraining value

10. The least constraining value is also effective method in some application.
11. Rule: Choose the least constraining value from many variable i.e. the one that leaves the maximum flexibility for subsequent variable assignment.
12. It can be combined with MRV for fast selection.

Passing Information through Constraints

While selecting unassigned variable for computation if we look at some of the constraints earlier in the search (or even before search begins), we can drastically reduce search space. Following are techniques which are helpful for earlier constraints checks.

Forward Checking (FC)

- ❖ Forward checking is one of the potential technique which uses constraints more effectively during search.
- ❖ It removes values in neighbouring unassigned variables domain that conflict with assigned variable.
- ❖ Forward checking uses MRV to select assigned variable.
- ❖ Backtracking search is performed if failure occurs.
- ❖ Search terminates when any variable has no legal values.
- ❖ Generic method,
 - Assigns variable X .

- Forward checking remarks unassigned variable Y connected to X .
- Remove values from D , where value is inconsistent with X .

Constraint Propagation

- ❖ A combined approach of heuristic plus forward checking gives more reliable, accurate and efficient results than a singular approach.
- ❖ The forward checking propagates information from assigned to unassigned variables but can't avoid or detect all failure.
- ❖ Constraint propagation repeatedly enforces constraints locally.
- ❖ The idea of arc consistency provides a fast method of constraint propagation that is substantially stronger than forward checking. Here "arc" refers to a directed arc in the constraint graph.

Constraint Propagation using Arc Consistency

- ❖ It is fast method of constraint propagation.
- ❖ $X \rightarrow Y$ is consistent if (for every value of X there is some allowed value Y . For example $[V_2 \rightarrow V_1]$, is consistent iff $V_1 = \text{Red}, V_2 = \text{Blue}$ (i.e. for every value of x in X there is some allowed value y in Y). This is directed property example $-[V_1 = V_2]$

$V_1 = V_2$ is consistent if

$V_1 = \text{Red and } V_2 = \text{Blue}$

- ❖ As directed arcs between variables represent the domains of specified variables, they are consistent with each other.
- ❖ Constraint propagation can be applied as preprocessing or propagation step.
 - Before search-Preprocessing.
 - After search-Propagation.
- ❖ The procedure for maintaining arc consistency, can be applied repeatedly.

Constraint Propagation using K-consistency

- ❖ Arc consistency is not capable of detecting all inconsistencies. Partial assignments $[V_1 = \text{Red}, V_2 = \text{Red}]$ are inconsistent.
- ❖ K-consistency is very strong form of constraint propagation.
- ❖ A CSP is K-consistency if for any set of $K - 1$ variables and for any consistent assignment to those variable a consistent value can always be assigned to any K^{th} variable.

Local Search for CSP

- ❖ It is most powerful search for CSP's.
- ❖ Backtrack search require more time in dynamic environment which can be reduced by local search.
- ❖ It uses complete state formulation as follows : -
 - a) The initial state which assigns value to every variable.
 - b) Successor function alters value of one variable for each instance.
 - c) For example - 8-queen problem

Initial state: A random configuration of 8-queens in 8 columns.

Successor function:

Function 1: It picks one queen and moved to elsewhere in its columns.

OR

Function 1: Each column can have queen in a permutations of the 8 rows.

1 MIN-CONFLICTS Algorithm for Solving CSPs by Local Search

1. Min-conflict heuristic select new value that result in a minimum number of conflicts with the other variable.
2. The initial state may be choosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn.
3. The conflicts procedure that counts the number of constraints violated by a particular value, given the rest of the current assignments.

The Algorithm

MIN-CONFLICTS

Inputs: CSP, a constraint satisfaction problem.

Max-steps, the number of steps allowed before giving up.

Output: A solution or failure.

Current-An initial complete assignment for CSP.

for i = 1 to max-steps do

 If current is a solution for CSP then return current.

 var - a randomly chosen conflicted variable from VARIABLE [CSP].

 value - the value v for var that minimizes CONFLICTS (var, v, current, csp).

 set var = value in current.

return failure.

Advantages of Min-conflict

- ❖ The key feature is that the runtime required for min-conflict is independent of problem size.
For example - It can solve million - queen problem in an average 50 steps.
- ❖ It also works for hard-problems.
- ❖ Local search is also used in an online setting.

For Example - in scheduling problem like weekly outline schedule.

- ❖ Example - of Min conflict

Consider 4-queen example shown in **Fig. 1.18** in 3 parts (a), (b), (c).

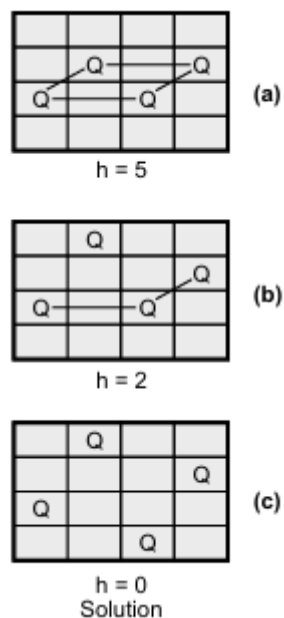


Fig. 1.18 Min conflict example.

[Figure depicts how ' h ' value changes per step]

Dealing with Special Constraints

Realistic problem has very special type (real in nature) of constraints which occurs frequently.
Example -

- ❖ All-diff constraints: It enforces the rule that all the variable in state space must have distinct values (as in the crypto arithmetic problem).
- ❖ Almost constraints or resource constraints:
 - These are most important higher order constraints.

- It bounds propagation for large value domains which are widely used in practical constraint problem.

Intelligent Backtracking

We observed in forward checking if inconsistency or failure occurs then we need to apply backtracking.

If backtracking is done efficiently then we can reduced total time.

Chronological Backtracking

1. It is one of the standard form (i.e. try different value for preceding variable).
2. In chronological backtracking most latest decision points are always revisited.
3. It has potential to back, up to preceding variable.
4. A more intelligent approach to backtracking is to go all the way back to one of the set of variables that caused the failure. This set is called the conflict set; for example, in map colouring problem, the conflict set for M is {N, NW, NE}, where N, MW, NE are already coloured regions. In general, the conflict set for variable X is the set of previously assigned variables that are connected to X by constraints.
5. The back jumping method backtracks to the most recent variable in the conflict set.

If no legal value is found, it should return the most recent element of the conflict set along with the failure indicator.

Structure of a Problem (Which can be used for Finding Quick Solution)

A technique to represent a problem in simple way, is to decompose the problem in to many sub -problems.

Sub-problems can be independent or they can be connected. If the sub-problems are totally independent then we have very easy way to solve the problem in totality. We can solve each sub-problem independently and then combine the solutions.

Many practical CSP sub-problems are connected. The simple case is when the constraint graph forms a tree. Any two variables are connected by almost one path.

Algorithm for Solving Tree-Structured CSP in Linear Time

- ❖ Choose any variable as the root of the tree, and order the variables from the root to the leaves in such a way that every node's parent in the tree precedes it in the ordering.
- ❖ Label the variables $X_1, \dots, X_n$ in order. Now, every variable except the root has exactly one parent variable.

Consider following example,

a) The constraint graph of a tree-structure CSP is,

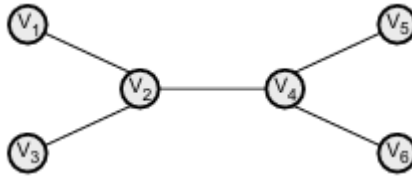


Fig. 1.19 Constraint graph.

b) A linear ordering of the variables consistent with the tree with V_1 as the root. Refer **Fig. 1.20**.



Fig. 1.20 Linear ordering of variables.

3. For j from n down to 2, apply arc consistency to the arc (X_i, X_j) where X_i is the parent of X_j , removing values from domain $[X_i]$ as necessary.

4. For j from 1 to n , assign any value for X_j consistent with the value assigned for X_i , where X_i is the parent of X_j .

Note: The complete algorithm runs in time $O(nd^2)$.

Graph Structured CSP

If we can reduce graph to a tree then solving graph structured CSP would be better in terms of time.

Reducing a graph to tree can be done in two ways.

1. Remove the nodes.
2. Collapse the nodes.

Remove the Nodes Approach (Cutset Conditioning)

- ❖ Choose a subset S from VARIABLES [csp] such that constraint graph becomes a tree after removal of S . S is called as a cycle cutset.
- ❖ For each possible assignment to the variables in S that satisfies all constraints on S ,
 - i) Remove from the domains of the remaining variables any values that are inconsistent with the assignment for S .

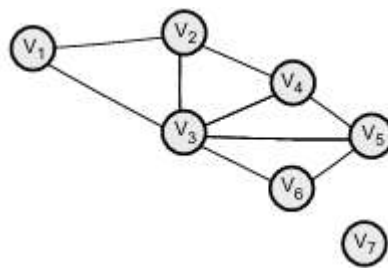
ii) If the remaining CSP has a solution, return it together with the assignment for S.

Note:

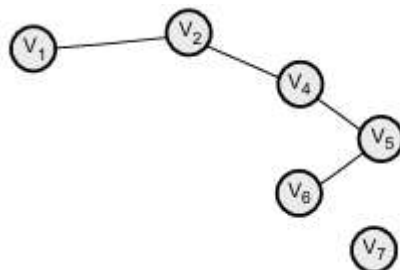
- ❖ If the cycle cutset has size C , then the total runtime is $O(d^c, (n - c)d^2)$.
- ❖ If the graph is "nearly a tree" then c will be small and the savings over straight backtracking will be huge.

For example –

Consider the following graph,



If we remove V_3 then the constraint graph is,



Collapse the Node (Tree Decomposition) Approach

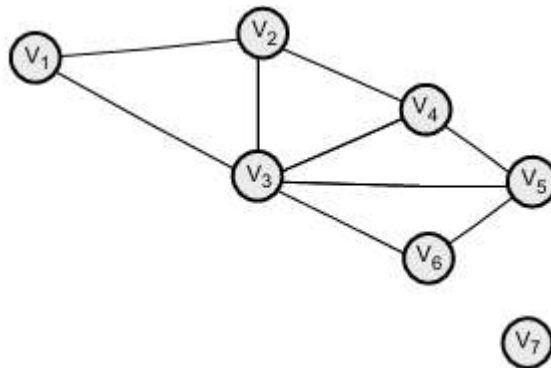
- ❖ The approach is based on constructing a tree decomposition of the constraint graph into a set of connected sub-problems.
- ❖ Each sub-problem is solved independently and the resulting solutions are then combined.
- ❖ A tree decomposition must satisfy the following three requirements:
 - Every variable in the original problem appears in at least one of the sub-problems.
 - If two variables are connected by a constraint in the original problem, they must

appear together (along with the constraint) in at least one of the sub-problems.

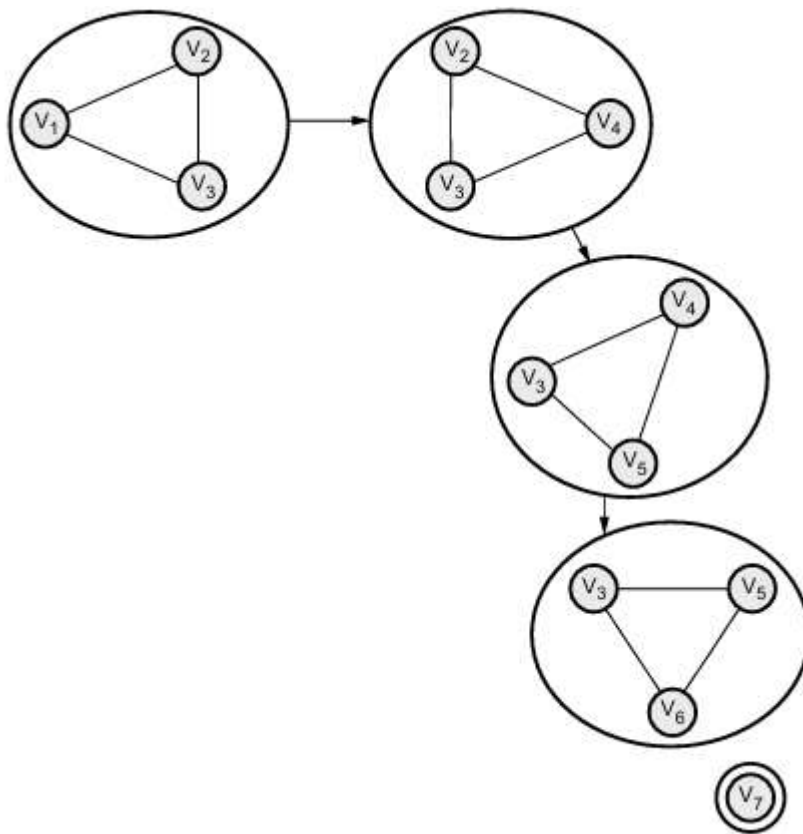
- If a variable appears in two sub-problems in the tree, it must appear in every sub-problem along the path connecting those sub-problems.
- ❖ A given constraint graph admits many tree decompositions. In choosing a decomposition, the aim is to make the sub-problems as small as possible.
- ❖ The tree width of a tree decomposition of a graph is one less than the size of the largest sub-problem; the tree width of the graph itself is defined to be the minimum tree width among all its tree decompositions.
- ❖ CSPs with constraint graphs of bounded tree width, are solvable in polynomial time. Unfortunately, finding the decomposition with minimal tree width is NP-hard, but there are heuristic methods that work well in practice.

For example –

Consider the following graph,



A tree decomposition of above graph is,



Representation of Knowledge

2.1 Game playing

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game. Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS(Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. Game playing in AI is an active area of research and has many practical applications, including game development, education, and military training. By simulating game playing scenarios, AI algorithms can be used to develop more effective decision-making systems for real-world applications.

The most common search technique in game playing is Minimax search procedure. It is depth-first depth-limited search procedure. It is used for games like chess and tic-tac-toe.

Mini-Max Algorithm

- ❖ Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- ❖ Mini-Max algorithm uses recursion to search through the game-tree.
- ❖ Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- ❖ In this algorithm two players play the game, one is called MAX and other is called MIN.
- ❖ Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- ❖ Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- ❖ The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- ❖ The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Pseudo-code for MinMax Algorithm:

function minimax(node, depth, maximizingPlayer) is

if depth == 0 or node is a terminal node then

return static evaluation of node

if MaximizingPlayer then // for Maximizer Player

maxEva= -infinity

for each child of node **do**

eva= minimax(child, depth-1, false)

maxEva= max(maxEva,eva) //gives Maximum of the values

return maxEva

else // for Minimizer player

minEva= 3+infinity

for each child of node **do**

eva= minimax(child, depth-1, true)

minEva= min(minEva, eva) //gives minimum of the values

return minEva

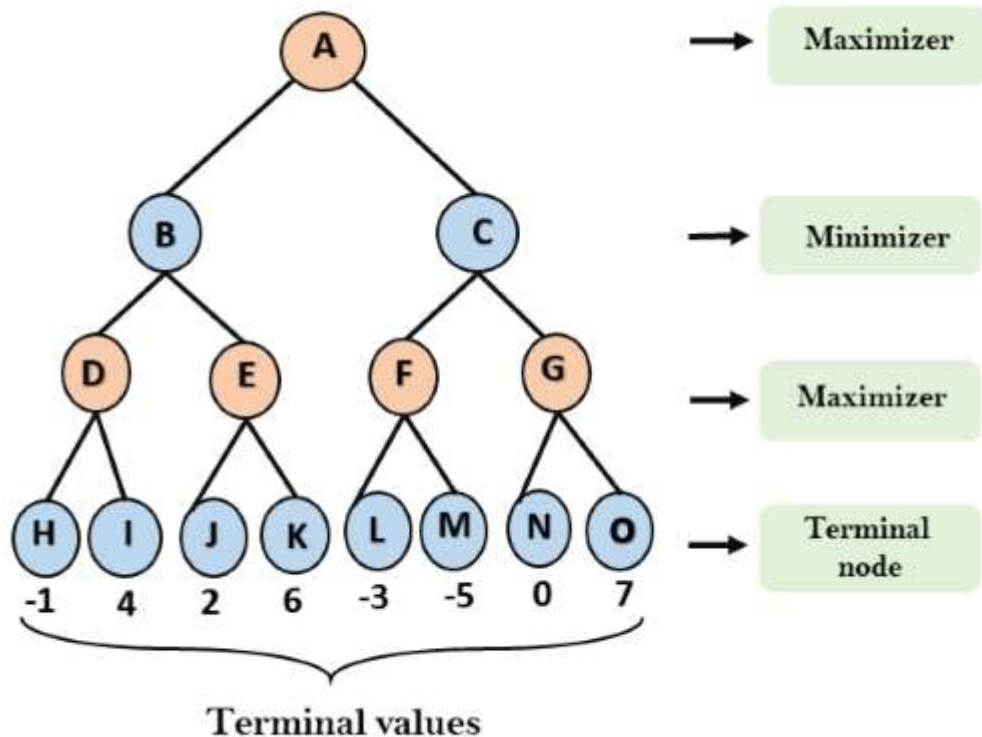
Initial call:

Minimax(node, 3, true)

Working of Min-Max Algorithm:

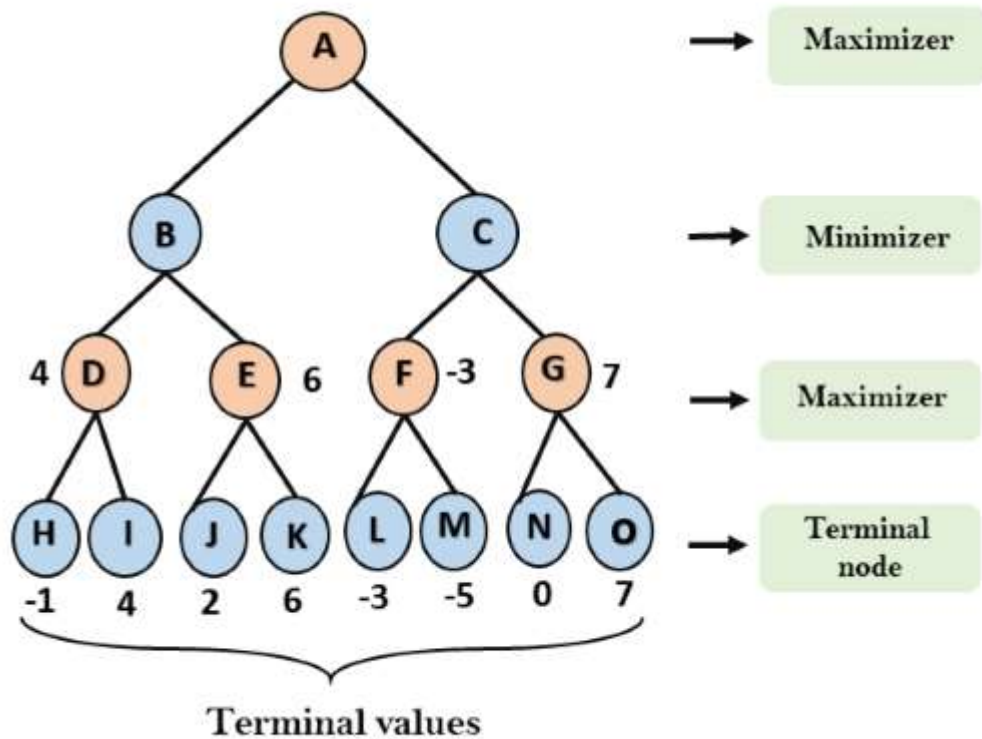
- ❖ The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- ❖ In this example, there are two players one is called Maximizer and other is called Minimizer.
- ❖ Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- ❖ This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- ❖ At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



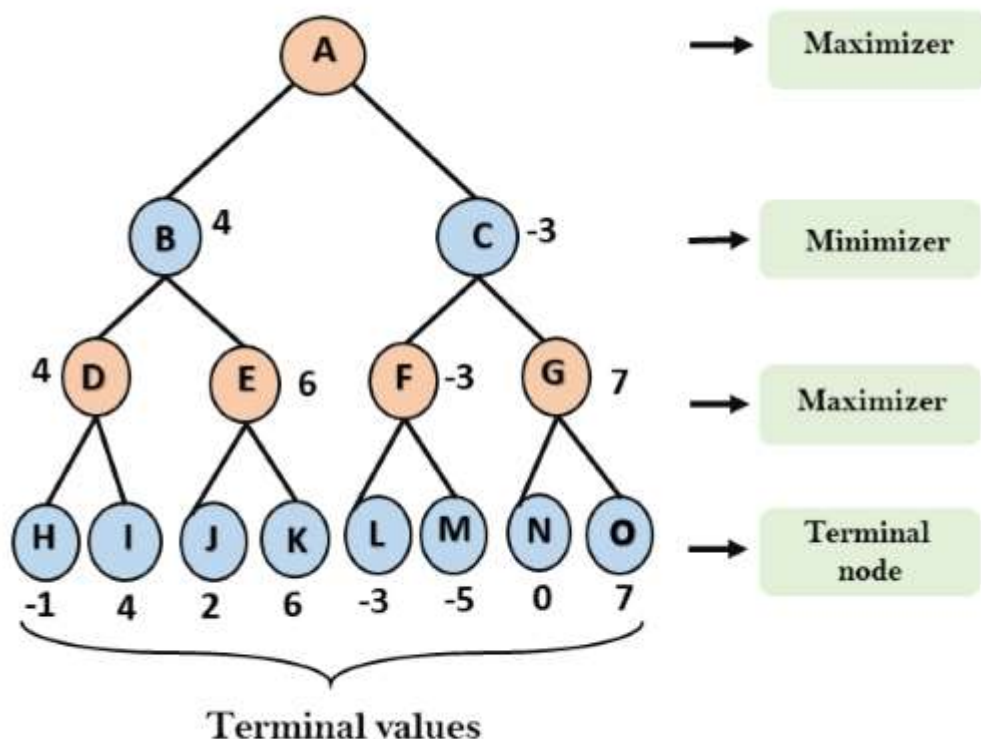
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is $-\infty$, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- ◆ For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$
- ◆ For Node E $\max(2, -\infty) \Rightarrow \max(2, 6) = 6$
- ◆ For Node F $\max(-3, -\infty) \Rightarrow \max(-3, -5) = -3$
- ◆ For node G $\max(0, -\infty) = \max(0, 7) = 7$



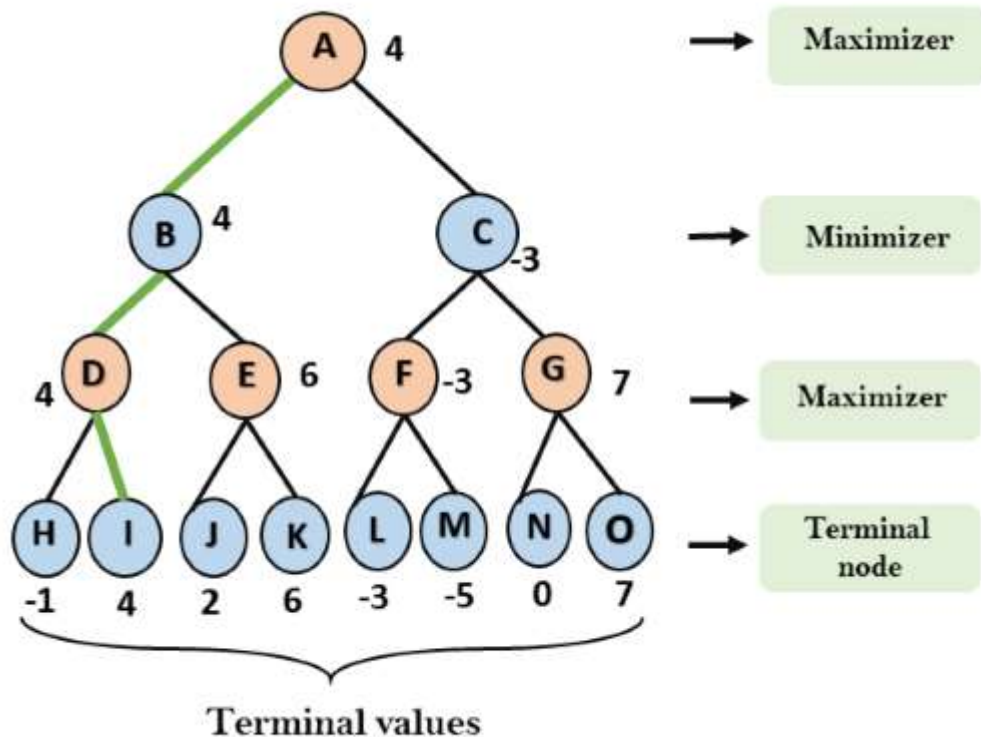
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with $+\infty$, and will find the 3rd layer node values.

- ◆ For node B= $\min(4, 6) = 4$
- ◆ For node C= $\min(-3, 7) = -3$



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- ◆ For node A $\max(4, -3) = 4$



That was the complete workflow of the minimax two player game.

Properties of Mini-Max algorithm:

- ❖ **Complete**- Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- ❖ **Optimal**- Min-Max algorithm is optimal if both opponents are playing optimally.
- ❖ **Time complexity**- As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- ❖ **Space Complexity**- Space complexity of Mini-max algorithm is also similar to DFS which is $O(bm)$.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning**

Advantages of Game Playing

1. **Advancement of AI:** Game playing has been a driving force behind the development of artificial intelligence and has led to the creation of new algorithms and techniques that can be applied to other areas of AI.
2. **Education and training:** Game playing can be used to teach AI techniques and algorithms to students and professionals, as well as to provide training for military and emergency response personnel.
3. **Research:** Game playing is an active area of research in AI and provides an opportunity to study and develop new techniques for decision-making and problem-solving.
4. **Real-world applications:** The techniques and algorithms developed for game playing can be applied to real-world applications, such as robotics, autonomous systems, and decision support systems.

Disadvantages of Game Playing in Artificial Intelligence

1. **Limited scope:** The techniques and algorithms developed for game playing may not be well-suited for other types of applications and may need to be adapted or modified for different domains.
2. **Computational cost:** Game playing can be computationally expensive, especially for complex games such as chess or Go, and may require powerful computers to achieve real-time performance.

2.2 Knowledge Representation

Knowledge Representation in AI refers to the way in which artificial intelligence systems store, organize, and utilize knowledge to solve complex problems. It is a crucial aspect of AI, enabling machines to mimic human understanding and reasoning. Knowledge representation involves the creation of data structures and models that can efficiently capture information about the world, making it accessible and usable by AI algorithms for decision-making, inference, and learning.

Humans are best at understanding, reasoning, and interpreting knowledge. Human knows things, which is knowledge and as per their knowledge they perform various actions in the real world. **But how machines do all these things comes under knowledge representation and reasoning.** Hence we can describe Knowledge representation as following:

- ❖ Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- ❖ It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.

Artificial Intelligence(AI) with it's Applications

- ❖ It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

What to Represent:

Following are the kind of knowledge which needs to be represented in AI systems:

- ❖ **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- ❖ **Events:** Events are the actions which occur in our world.
- ❖ **Performance:** It describe behavior which involves knowledge about how to do things.
- ❖ **Meta-knowledge:** It is knowledge about what we know.
- ❖ **Facts:** Facts are the truths about the real world and what we represent.
- ❖ **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB.



Fig. 2.1 Types of Knowledge.

1. Declarative Knowledge:

- ❖ Declarative knowledge is to know about something.
- ❖ It includes concepts, facts, and objects.
- ❖ It is also called descriptive knowledge and expressed in declarative sentences.
- ❖ It is simpler than procedural language.

2. Procedural Knowledge

- ❖ It is also known as imperative knowledge.
- ❖ Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- ❖ It can be directly applied to any task.
- ❖ It includes rules, strategies, procedures, agendas, etc.
- ❖ Procedural knowledge depends on the task on which it can be applied.

3. Meta-knowledge:

- ❖ Knowledge about the other types of knowledge is called Meta-knowledge.

4. Heuristic knowledge:

- ❖ Heuristic knowledge is representing knowledge of some experts in a field or subject.
- ❖ Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

5. Structural knowledge:

- ❖ Structural knowledge is basic knowledge to problem-solving.
- ❖ It describes relationships between various concepts such as kind of, part of, and grouping of something.
- ❖ It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

AI knowledge cycle:

An Artificial intelligence system has the following components for displaying intelligent behavior:

- ❖ Perception
- ❖ Learning
- ❖ Knowledge Representation and Reasoning
- ❖ Planning
- ❖ Execution

The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are given below:

1. Simple relational knowledge:

- ❖ It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- ❖ This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- ❖ This approach has little opportunity for inference.

Example: The following is the simple relational knowledge representation.

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

Table. 2.1 Simple relational knowledge.

2. Inheritable knowledge:

- ❖ In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- ❖ All classes should be arranged in a generalized form or a hierarchal manner.
- ❖ In this approach, we apply inheritance property.
- ❖ Elements inherit values from other members of a class.
- ❖ This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- ❖ Every individual frame can represent the collection of attributes and its value.
- ❖ In this approach, objects and values are represented in Boxed nodes.
- ❖ We use Arrows which point from objects to their values.

Example:

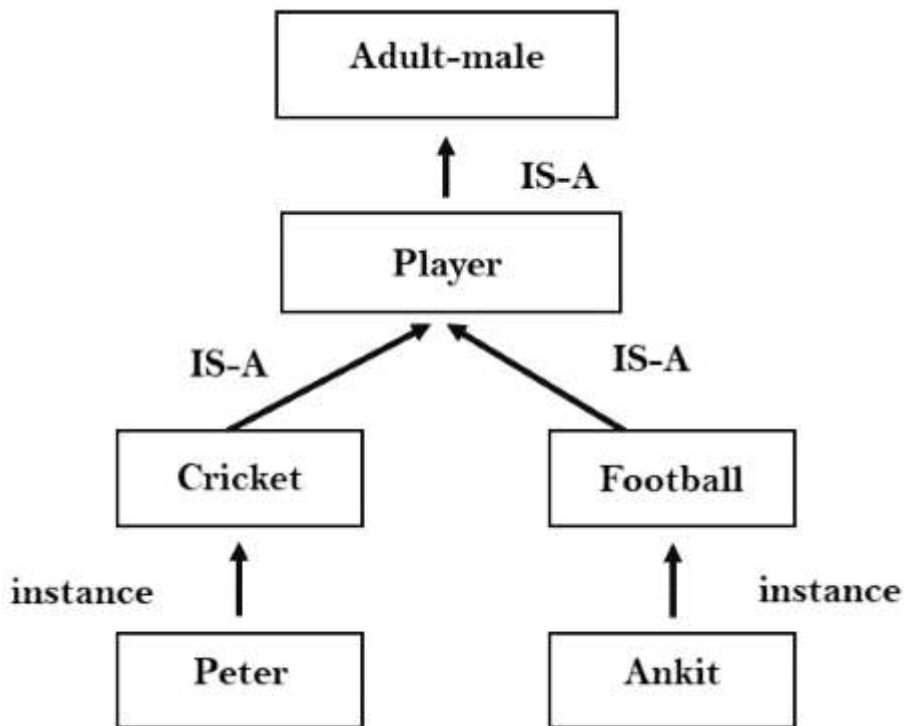


Fig. 2.2 Inheritable knowledge Example.

3. Inferential knowledge:

- ❖ Inferential knowledge approach represents knowledge in the form of formal logics.
- ❖ This approach can be used to derive more facts.
- ❖ It guaranteed correctness.

Artificial Intelligence(AI) with it's Applications

Example: Let's suppose there are two statements:

- ❖ Marcus is a man
- ❖ All men are mortal
- ❖ Then it can represent as;

man(Marcus)

$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$

4. Procedural knowledge:

- ❖ Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- ❖ In this approach, one important rule is used which is **If-Then rule**.
- ❖ In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- ❖ We can easily represent heuristic or domain-specific knowledge using this approach.
- ❖ But it is not necessary that we can represent all cases in this approach.

Relationship between Knowledge and Intelligence

Knowledge as a Foundation: Knowledge provides the necessary information, facts, and skills that intelligence uses to solve problems and make decisions.

Intelligence as Application: Intelligence is the ability to learn, reason, and adapt, using knowledge to perform tasks and solve complex problems.

Interdependence: Knowledge without intelligence is static, while intelligence without knowledge lacks the raw material to function effectively.

Synergy: Effective AI systems require a balance of both knowledge (the "what") and intelligence (the "how") to operate successfully.

Cycle of Knowledge Representation in Artificial Intelligence

The AI Knowledge Cycle is an ongoing process where AI systems continually acquire, process, utilize, and refine knowledge to enhance performance.

It consists of these key stages:

1. **Knowledge Acquisition:** Gathering data and information from various sources, including databases, sensors, and human input.
 2. **Knowledge Representation:** Organizing and structuring this knowledge using techniques like ontologies and semantic networks for effective processing.
 3. **Knowledge Utilization:** Applying the structured knowledge to perform tasks, make decisions, and solve problems through reasoning and inference.
 4. **Knowledge Learning:** Continuously updating the knowledge base by learning from new data and outcomes using machine learning algorithms.
-

5. **Knowledge Validation and Verification:** Ensuring the accuracy, consistency, and reliability of the knowledge through validation against real-world outcomes.
6. **Knowledge Maintenance:** Regularly updating the knowledge base to stay relevant and accurate as the environment or information changes.
7. **Knowledge Sharing:** Distributing the knowledge to other systems or users, making it accessible and usable beyond the original AI system.

Key Techniques in Knowledge Representation

1. First-Order Logic (FOL)

First-Order Logic is a formal system used in mathematics, philosophy, and computer science to represent and reason about propositions involving objects, their properties, and their relationships. Unlike propositional logic, FOL allows the use of quantifiers (like "forall" and "exists") to express more complex statements.

FOL is widely used in AI for knowledge representation and reasoning because it allows for expressing general rules and facts about the world. For example, FOL can be used to represent statements like "All humans are mortal" and "Socrates is a human," enabling AI systems to infer that "Socrates is mortal." It provides a powerful and flexible framework for representing structured knowledge and supports various forms of logical reasoning.

2. Fuzzy Logic

Fuzzy Logic is an approach to knowledge representation that deals with reasoning that is approximate rather than exact. It allows for the representation of concepts that are not black and white, but rather fall along a continuum, with degrees of truth ranging from 0 to 1.

Fuzzy Logic is particularly useful in domains where precise information is unavailable or impractical, such as control systems, decision-making, and natural language processing. For example, in a climate control system, fuzzy logic can be used to represent concepts like "warm," "hot," or "cold," and make decisions based on the degree to which these conditions are met, rather than relying on strict numerical thresholds.

3. Description Logics

Description Logics are a family of formal knowledge representation languages used to describe and reason about the concepts and relationships within a domain. They are more expressive than propositional logic but less complex than full first-order logic, making them well-suited for representing structured knowledge.

Description Logics form the foundation of ontologies used in the Semantic Web and are key to building knowledge-based systems that require classification, consistency checking, and inferencing. For example, they can be used to define and categorize different types of products in an e-commerce system, allowing for automated reasoning about product features,

relationships, and hierarchies.

4. Semantic Web Technologies

Semantic Web Technologies refer to a set of standards and tools designed to enable machines to understand and interpret data on the web in a meaningful way. Key technologies include Resource Description Framework (RDF), Web Ontology Language (OWL), and SPARQL, which are used to represent, query, and reason about knowledge on the web.

These technologies are essential for building intelligent applications that can access, share, and integrate data across different domains and systems. For example, Semantic Web Technologies are used in search engines, recommendation systems, and data integration platforms to provide more relevant and accurate results by understanding the context and meaning of the data. They enable AI systems to perform tasks like semantic search, data linking, and automated reasoning over distributed knowledge bases.

Challenges in Knowledge Representation

While knowledge representation is fundamental to AI, it comes with several challenges:

Complexity: Representing all possible knowledge about a domain can be highly complex, requiring sophisticated methods to manage and process this information efficiently.

Ambiguity and Vagueness: Human language and concepts are often ambiguous or vague, making it difficult to create precise representations.

Scalability: As the amount of knowledge grows, AI systems must scale accordingly, which can be challenging both in terms of storage and processing power.

Knowledge Acquisition: Gathering and encoding knowledge into a machine-readable format is a significant hurdle, particularly in dynamic or specialized domains.

Reasoning and Inference: AI systems must not only store knowledge but also use it to infer new information, make decisions, and solve problems. This requires sophisticated reasoning algorithms that can operate efficiently over large knowledge bases.

Applications of Knowledge Representation

Knowledge representation is applied across various domains in AI, enabling systems to perform tasks that require human-like understanding and reasoning. Some notable applications include:

Expert Systems: These systems use knowledge representation to provide advice or make

decisions in specific domains, such as medical diagnosis or financial planning.

Natural Language Processing (NLP): Knowledge representation is used to understand and generate human language, enabling applications like chatbots, translation systems, and sentiment analysis.

Robotics: Robots use knowledge representation to navigate, interact with environments, and perform tasks autonomously.

Semantic Web: The Semantic Web relies on ontologies and other knowledge representation techniques to enable machines to understand and process web content meaningfully.

Cognitive Computing: Systems like IBM's Watson use knowledge representation to process vast amounts of information, reason about it, and provide insights in fields like healthcare and research.

2.3 Understanding using Predicate Logic

Logic is a formal system in which the formulas or sentences have true or false values.

Logics are of different types:

Propositional logic, Predicate logic, Temporal logic, Modal logic, Description logic etc; They represent things and allow more or less efficient inference.

- ❖ Propositional Logic is the study of statements and their connectivity.
- ❖ Predicate Logic is the study of individuals and their properties. We need languages that allow us to describe properties (predicates) of objects, or a relationship among objects represented by the variables. Predicate logic satisfies the requirements of a language. Predicate logic is powerful enough for expression and reasoning. Predicate logic is built upon the ideas of propositional logic. Predicate Logic can represent Objects and Quantification. Theorem proving is semidecidable. Proposition is a declarative sentence whose value is either true or false

Basics of logic symbols:

“ $\rightarrow$ ” Material Implication “ $\neg$ ” Not

“ $\vee$ ” Or

“ $\wedge$ ” and “ $\forall$ ” for all

“ $\exists$ ” There exists

Representing simple facts in Logic

- ❖ Proposition logic is simple where a decision procedure exists.
- ❖ Represent Real-World facts as logical propositions, written as well – formed formulas in propositional logic as shown in figure below:

Example: Conclusion

- ❖ The fact “It is raining“ the fact it is not sunny
- ❖ Limitations of propositional logic are overridden.

If we want to represent the fact stated by the classical sentence. It is raining RAINING

Its is sunny SUNNY It is windy WINDY

If it is raining, then it is not sunny $\text{RAINING} \rightarrow \neg \text{SUNNY}$

Socrates is a man We could write: SOCRATESMAN

But if we also wanted to represent Plato is a man We would have to write something such as: PLATOMAN Which would be a totally separate assertion, and we would not be able to draw any conclusions about similarities between Socrates and Plato. It would be much better to represent these facts as:

MAN (SOCRATES) MAN (PLATO)

Since, now the structure of the representation reflects the structure of the knowledge itself. But to do that, we need to be able to use predicates applied to arguments. We are in even more difficulty if we try to represent the equally classic sentence All men are mortal. We could represent this as: MORTALMAN To capture the relationship between any individual being a man and that individual being a mortal. To do this, we need variable and quantification unless willing to write separate statements about the mortality of every known man.

Predicate Logic

This logic is one way of representing knowledge because it permits representations of things that cannot reasonably be represented in propositional logic.

- ❖ If we use logical statements as a way of representing knowledge, than we have good way of reasoning with that knowledge.
- ❖ Determining the validity of a propositional logic is easy but may be computationally hard.
- ❖ Predicate logic provides a good way of reasoning with knowledge.
- ❖ It provides a way of deducing new statements from old ones.
- ❖ It does not possess a decision procedure, even an exponential one. There exist

procedures that will find a proof of a proposed theorem if indeed it is a theorem. First order predicate logic is semi decidable. Such a procedure is to use the rules of inference to generate theorems from the axioms in some fashion, testing each to see if it is the one for which a proof is sought.

Examples of Predicate Logic

Let's now explore the use of predicate logic as a way of representing knowledge by looking at a specific example. Consider the following set of sentences:

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

The facts described by these sentences can be represented as a set of wff's in predicate logic as follows:

1. Marcus was a man

Man (Marcus)

Although this representation fails to represent the notion of past tense (which is clear in the English sentence), it captures the critical fact of Marcus being a man. Whether this omission is acceptable or not depends on the use to which we intend to put the knowledge.

2. Marcus was a Pompeian

Pompeian (Marcus)

3. All Pompeians were Romans

$\forall x : \text{Pompeian}(x) \rightarrow \text{Roman}(x)$

4. Caesar was a ruler

Ruler (Caesar)

Since many people share the same name, the fact that proper names are often not references to unique individuals, overlooked here. Occasionally deciding which of several people of the same name is being referred to in a particular statement may require a somewhat more amount of knowledge and logic

5. All Romans were either loyal to Caesar or hated him

$\forall x: \text{Roman}(x) \rightarrow \text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})$

In English, the word “or” sometimes means the logical inclusive-or and sometimes means the logical exclusive-or (XOR). Here we have used the inclusive interpretation. Soome people will argue, however, That this English sentence is really stating an exclusive-or. To express that, we would have to write:

$\forall x: \text{Roman}(x) \rightarrow [(\text{loyalto}(x, \text{Caesar}) \vee \text{hate}(x, \text{Caesar})) \vee \neg (\text{loyalto}(x, \text{Caesar}) \wedge \text{hate}(x, \text{Caesar}))]$

6. Everyone is loyal to someone

$\forall x: \rightarrow \exists y: \text{loyalto}(x, y)$

The scope of quantifiers is a major problem that arises when trying to convert English sentences into logical statements. Does this sentence say, as we have assumed in writing the logical formula above, that for each person there exists someone to whom he or she is loyal, possibly a different someone for everyone? Or does it say that there is someone to whom everyone is loyal?

7. People only try to assassinate rulers they are not loyal to

$\forall x : \forall y : \text{person}(x) \wedge \text{ruler}(y) \wedge \text{tryassassinate}(x, y) \rightarrow \neg \text{loyalto}(x, y)$

Like the previous one this sentence too is ambiguous which may lead to more than one conclusion. The usage of “try to assassinate” as a single predicate gives us a fairly simple representation with which we can reason about trying to assassinate. But there might be connections as try to assassinate and not actually assassinate could not be made easily.

8. Marcus tried to assassinate Caesar

To produce a Formal proof, reasoning backward from the desired goal:

$\neg \text{loyalto}(\text{Marcus}, \text{Caesar})$

To prove the goal

- ❖ Use the rules of inference to transform it into another goal that can be transformed and so on, until there are no unsatisfied goals remaining.
- ❖ This process requires the search of an AND-OR graph when there is an alternative way of satisfying individual goals. We show only a single path for simpler way.
- ❖ The below figure shows an attempt to produce a proof of the goal by reducing the set of necessary but as yet unattained goals to the empty set. If attempt fails, there is no way to satisfy the goal person with the available statements. Problem: Marcus was a man, need to add the representation of another fact to our system.

9.All men are people

$\forall x : \text{man}(x) \rightarrow \text{person}(x)$

Now we can satisfy the last goal and produce a proof that Marcus was not loyal to Caesar.

-loyalto (Marcus, Caesar)

↑ (7,substitution)

person (Marcus) ^ ruler (Caesra) ^ tryassassinate (Marcus, Caesar)

↑ (4)

person (Marcus) tryassassinate (Marcus, Caesar)

↑ (8)

person (Marcus)

Computable functions and predicates

Some of the computational predicates like Less than, Greater than used in knowledge representation. Simple facts were expresses as combinations of individual predicates, such as: Tryassassinate (Marcus, Caesar) If Number of facts is not large or if the facts are unstructured then there is little alternative.

If we want to express simple facts, like greater than and less than

relations:gt(1,0)

lt(0,1)

gt(2,1)

lt(1,2)

gt(3,2)

lt(2,3)

There is infinite representation of each of these fact. But if we consider the finite number of them that can be represented, using a single machine word per number, extremely inefficient to store explicitly large a set of statements. Instead, we compute each one as we need it.

Functions and predicate use

Consider the following set of facts Again involving Marcus:

1. Marcus was a man

Man (Marcus) Again we ignore the issue of tense.

2. Marcus was a pompeian

Pompeian (Marcus)

Marcus was born in 40 A.D

Born (Marcus, 40) For simplicity, we will not represent A.D. explicitly, just as we normally omit it in everyday discussions. If we ever need to represent dates B.C., then we will have to decide on a way to do that such as by using negative numbers.

2. All men are mortal

$\forall x: \text{men}(x) \rightarrow \text{mortal}(x)$

3. erupted(volcano,79) $\forall x$:[pompeian(x) $\rightarrow$ died(x, 79)]

This sentence clearly asserts the two facts represented above. It may also assert another that we have not shown, namely that the eruption of the volcano caused the death of the Pompeians. People often assume causality between concurrent events if such causality seems possible.

4. No mortal lives longer than 150 years

$\forall x: \forall t1: \forall t2: \text{mortal}(x) \wedge \text{born}(x,t1) \wedge \text{gt}(t2-t1,150) \rightarrow \text{dead}(x,t2)$

There are several ways that the content of this sentence could be expressed.

For example, we could introduce a function age and assert that its value is never greater than 150.

5. It is Now 1991

Now=1991

Here we exploit the idea of equal quantities that can be substituted for each other. Now suppose we want to answer the question “Is Marcus alive?” A quick glance through the statements we have suggests that there may be two ways of deducing an answer. Either we can show that Marcus is dead because he was killed by the volcano or we can show that he must be dead because he would otherwise be more than 150 years old, which we know is not possible.

As soon as we attempt to follow either of those paths rigorously, however, we discover, just as we did in the last example, that we need some additional knowledge. For example, our statements talk about dying, but they say nothing that relates to being alive, which is what the question is asking. So we add the following facts:

6. Alive means not dead

$$\forall x: \forall t: [\text{alive}(x,t) \rightarrow \neg \text{dead}(x,t)] \cup [\neg \text{dead}(x,t) \rightarrow \text{alive}(x,t)]$$

This is not strictly correct, since $\neg \text{dead}$ implies alive only for animate objects. This is an example of the fact that rarely do two expressions have truly identical meanings in all circumstances.

7. If someone dies then he is dead at all later times

$\forall x: \forall t1: \forall t2: \text{died}(x,t1) \wedge t(t2,t1) \rightarrow \text{dead}(x,t2)$ This representation says that one is dead in all years after the one in which one died. It ignores the question of whether one is dead in the year in which one died.

2.4 Overview of Predicate Calculus

Predicate Calculus deals with predicates, which are propositions containing variables.

Predicate

A predicate is an expression of one or more variables defined on some specific domain. A predicate with variables can be made a proposition by either assigning a value to the variable or by quantifying the variable.

Consider the following statement.

- ❖ Ram is a student.

Now consider the above statement in terms of Predicate calculus.

- ❖ Here "is a student" is a predicate and Ram is subject.
- ❖ Let's denote "Ram" as x and "is a student" as a predicate P then we can write the above statement as $P(x)$.
- ❖ Generally a statement expressed by Predicate must have at least one object associated with Predicate. In our case, Ram is the required object with associated with predicate P .

Statement Function

Earlier we denoted "Ram" as x and "is a student" as predicate P then we have statement as $P(x)$. Here $P(x)$ is a statement function where if we replace x with a Subject say Sunil then we'll be having a statement "Sunil is a student."

Thus a statement function is an expression having Predicate Symbol and one or multiple variables. This statement function gives a statement when we replaced the variables with objects. This replacement is called substitution instance of statement function.

Quantifiers

The variable of predicates is quantified by quantifiers. There are two types of quantifier in predicate logic – Universal Quantifier and Existential Quantifier.

Universal Quantifier

Universal quantifier states that the statements within its scope are true for every value of the specific variable. It is denoted by the symbol $\forall$.

$\forall x P(x)$ is read as for every value of x , $P(x)$ is true.

Example – "Man is mortal" can be transformed into the propositional form $\forall x P(x)$ where $P(x)$ is the predicate which denotes x is mortal and $\forall x$ represents all men.

Existential Quantifier

Existential quantifier states that the statements within its scope are true for some values of the specific variable. It is denoted by the symbol $\exists$.

$\exists x P(x)$ is read as for some values of x , $P(x)$ is true.

Example – "Some people are dishonest" can be transformed into the propositional form $\exists x P(x)$ where $P(x)$ is the predicate which denotes x is dishonest and $\exists x$ represents some dishonest men.

Predicate Formulas

Consider a Predicate P with n variables as $P(x_1, x_2, x_3, \dots, x_n)$. Here P is n -place predicate and $x_1, x_2, x_3, \dots, x_n$ are n individuals variables. This n -place predicate is known as atomic formula of predicate calculus. For Example: $P()$, $Q(x, y)$, $R(x, y, z)$

Well Formed Formula

Well Formed Formula (wff) is a predicate holding any of the following –

- ❖ All propositional constants and propositional variables are wffs
- ❖ If x is a variable and Y is a wff, $\forall x Y$ and $\exists x Y$ are also wff
- ❖ Truth value and false values are wffs
- ❖ Each atomic formula is a wff
- ❖ All connectives connecting wffs are wffs

Free and Bound variables

Consider a Predicate formula having a part in form of $(\exists x) P(x)$ or $(x)P(x)$, then such part is called x -bound part of the formula. Any occurrence of x in x -bound part is termed as bound occurrence and any occurrence of x which is not x -bound is termed as free occurrence. See the examples below -

- ❖ $(\exists x) (P(x) \wedge Q(x))$
- ❖ $(\exists x) P(x) \wedge Q(x)$

In first example, scope of $(\exists x)$ is $(P(x) \wedge Q(x))$ and all occurrences of x are bound occurrences. Whereas in second example, scope of $(\exists x)$ is $P(x)$ and last occurrence of x in $Q(x)$ is a free occurrence.

Universe of Discourse

We can limit the class of individuals/objects used in a statement. Here limiting means confining the input variable to a set of particular individuals/objects. Such a restricted class is termed as Universe of Discourse/domain of individual or universe. See the example below:

Some cats are black.

- ❖ $C(x)$: x is a cat.
- ❖ $B(x)$: x is black.
- ❖ $(\exists x)(C(x) \wedge B(x))$

If Universe of discourse is $E = \{ \text{Katy, Mille} \}$ where katy and Mille are white cats then our third statement is false when we replace x with either Katy or Mille where as if Universe of discourse is $E = \{ \text{Jene, Jackie} \}$ where Jene and Jackie black cats then our third statement stands true for Universe of Discourse F.

2.5 Resolution

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.

Clause: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.

Conjunctive Normal Form: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

The resolution rule for first-order logic is simply a lifted version of the propositional rule. Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

$$l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee m_n$$

$$\text{SUBST}(\theta, l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \dots \vee m_n)$$

Where l_i and m_j are complementary literals.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

$$[\text{Animal}(g(x) \vee \text{Loves}(f(x), x)] \quad \text{and} \quad [\neg \text{Loves}(a, b) \vee \neg \text{Kills}(a, b)]$$

Where two complimentary literals are: **Loves (f(x), x)** and **$\neg$ Loves (a, b)**

Artificial Intelligence(AI) with it's Applications

These literals can be unified with unifier $\theta = [a/f(x), \text{ and } b/x]$, and it will generate a resolvent clause:

[Animal (g(x) $\vee$ $\neg$ Kills(f(x), x)).

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

1. John likes all kind of food.
 2. Apple and vegetable are food
 3. Anything anyone eats and not killed is food.
 4. Anil eats peanuts and still alive
 5. Harry eats everything that Anil eats.
- Prove by resolution that:**
6. John likes peanuts.

Step-1: Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.

- a. $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$
 - b. $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
 - c. $\forall x \forall y: \text{eats}(x, y) \wedge \neg \text{killed}(x) \rightarrow \text{food}(y)$
 - d. $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$.
 - e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$
 - f. $\forall x: \neg \text{killed}(x) \rightarrow \text{alive}(x)$
 - g. $\forall x: \text{alive}(x) \rightarrow \neg \text{killed}(x)$
 - h. $\text{likes}(\text{John}, \text{Peanuts})$
- } added predicates.

Step-2: Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

❖ Eliminate all implication ($\rightarrow$) and rewrite

- ◆ $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- ◆ $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- ◆ $\forall x \forall y \neg [\text{eats}(x, y) \wedge \neg \text{killed}(x)] \vee \text{food}(y)$
- ◆ $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- ◆ $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
- ◆ $\forall x \neg [\neg \text{killed}(x)] \vee \text{alive}(x)$
- ◆ $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
- ◆ $\text{likes}(\text{John}, \text{Peanuts})$.

❖ Move negation ($\neg$)inwards and rewrite

- ◆ $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- ◆ $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- ◆ $\forall x \forall y \neg \text{eats}(x, y) \vee \text{killed}(x) \vee \text{food}(y)$
- ◆ $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- ◆ $\forall x \neg \text{eats}(\text{Anil}, x) \vee \text{eats}(\text{Harry}, x)$
- ◆ $\forall x \neg \text{killed}(x) \vee \text{alive}(x)$
- ◆ $\forall x \neg \text{alive}(x) \vee \neg \text{killed}(x)$
- ◆ $\text{likes}(\text{John}, \text{Peanuts})$.

❖ Rename variables or standardize variables

- ◆ $\forall x \neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- ◆ $\text{food}(\text{Apple}) \wedge \text{food}(\text{vegetables})$
- ◆ $\forall y \forall z \neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
- ◆ $\text{eats}(\text{Anil}, \text{Peanuts}) \wedge \text{alive}(\text{Anil})$
- ◆ $\forall w \neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- ◆ $\forall g \neg \text{killed}(g) \vee \text{alive}(g)$
- ◆ $\forall k \neg \text{alive}(k) \vee \neg \text{killed}(k)$
- ◆ $\text{likes}(\text{John}, \text{Peanuts})$.

❖ Eliminate existential instantiation quantifier by elimination.

In this step, we will eliminate existential quantifier $\exists$, and this process is known as **Skolemization**. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

❖ Drop Universal quantifiers.

In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

- ◆ $\neg \text{food}(x) \vee \text{likes}(\text{John}, x)$
- ◆ $\text{food}(\text{Apple})$
- ◆ $\text{food}(\text{vegetables})$
- ◆ $\neg \text{eats}(y, z) \vee \text{killed}(y) \vee \text{food}(z)$
- ◆ $\text{eats}(\text{Anil}, \text{Peanuts})$
- ◆ $\text{alive}(\text{Anil})$
- ◆ $\neg \text{eats}(\text{Anil}, w) \vee \text{eats}(\text{Harry}, w)$
- ◆ $\text{killed}(g) \vee \text{alive}(g)$
- ◆ $\neg \text{alive}(k) \vee \neg \text{killed}(k)$
- ◆ $\text{likes}(\text{John}, \text{Peanuts})$.

❖ Distribute conjunction $\wedge$ over disjunction $\vee$.

This step will not make any change in this problem.

Step-3: Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as $\neg \text{likes}(\text{John}, \text{Peanuts})$

Step-4: Draw Resolution graph:

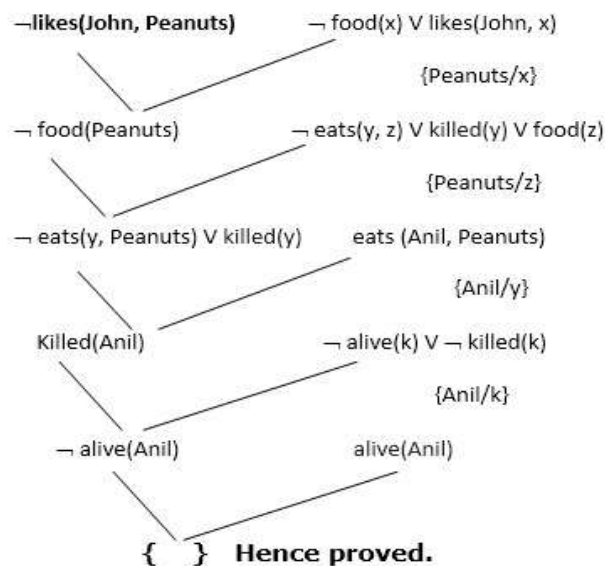


Fig. 2.3 Resolution graph.

Hence the negation of the conclusion has been proved as a complete contradiction with the given set of statements.

Explanation of Resolution graph:

- ❖ In the first step of resolution graph, $\neg \text{likes}(\text{John}, \text{Peanuts})$, and $\text{likes}(\text{John}, x)$ get resolved(canceled) by substitution of $\{\text{Peanuts}/x\}$, and we are left with $\neg \text{food}(\text{Peanuts})$
- ❖ In the second step of the resolution graph, $\neg \text{food}(\text{Peanuts})$, and $\text{food}(z)$ get resolved (canceled) by substitution of $\{\text{Peanuts}/z\}$, and we are left with $\neg \text{eats}(y, \text{Peanuts}) \vee \text{killed}(y)$.
- ❖ In the third step of the resolution graph, $\neg \text{eats}(y, \text{Peanuts})$ and $\text{eats}(\text{Anil}, \text{Peanuts})$ get resolved by substitution $\{\text{Anil}/y\}$, and we are left with $\text{Killed}(\text{Anil})$.
- ❖ In the fourth step of the resolution graph, $\text{Killed}(\text{Anil})$ and $\neg \text{killed}(k)$ get resolve by substitution $\{\text{Anil}/k\}$, and we are left with $\neg \text{alive}(\text{Anil})$.
- ❖ In the last step of the resolution graph $\neg \text{alive}(\text{Anil})$ and $\text{alive}(\text{Anil})$ get resolved.

2.6 Structured Representation of Knowledge

Representing knowledge using logical formalism, like predicate logic, has several advantages. They can be combined with powerful inference mechanisms like resolution, which makes reasoning with facts easy. But using logical formalism complex structures of the world, objects and their relationships, events, sequences of events etc. can not be described easily.

A good system for the representation of structured knowledge in a particular domain should posses the following four properties:

Representational Adequacy:- The ability to represent all kinds of knowledge that are needed in that domain.

Inferential Adequacy :- The ability to manipulate the represented structure and infer new structures.

Inferential Efficiency:- The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.

Acquisitional Efficiency :- The ability to acquire new information easily, either by direct insertion or by program control.

The techniques that have been developed in AI systems to accomplish these objectives fall under two categories:

Declarative Methods:- In these knowledge is represented as static collection of facts which are manipulated by general procedures. Here the facts need to be stored only one and they

Artificial Intelligence(AI) with it's Applications

can be used in any number of ways. Facts can be easily added to declarative systems without changing the general procedures.

Procedural Method:- In these knowledge is represented as procedures. Default reasoning and probabilistic reasoning are examples of procedural methods. In these, heuristic knowledge of

“How to do things efficiently “can be easily represented.

In practice most of the knowledge representation employ a combination of both. Most of the knowledge representation structures have been developed to handle programs that handle natural language input. One of the reasons that knowledge structures are so important is that they provide a way to represent information about commonly occurring patterns of things . such descriptions are some times called schema. One definition of schema is

“Schema refers to an active organization of the past reactions, or of past experience, which must always be supposed to be operating in any well adapted organic response”.

By using schemas, people as well as programs can exploit the fact that the real world is not random. There are several types of schemas that have proved useful in AI programs. They include

Frames:- Used to describe a collection of attributes that a given object possesses (eg: description of a chair).

Scripts:- Used to describe common sequence of events
(eg:- a restaurant scene).

Stereotypes :- Used to described characteristics of people.

Rule models:- Used to describe common features shared among a set of rules in a production system.

Frames and scripts are used very extensively in a variety of AI programs. Before selecting any specific knowledge representation structure, the following issues have to be considered.

The basis properties of objects , if any, which are common to every problem domain must be identified and handled appropriately.

The entire knowledge should be represented as a good set of primitives.

Mechanisms must be devised to access relevant parts in a large knowledge base.

CHAPTER-3

Structured Knowledge Representation, Handling Uncertainty and Bayesian Theory

3.1 Overview of Structured Knowledge Representation Techniques

Following are various structured knowledge representation techniques. All of these techniques, in different ways, involve hierarchical representation of data.

- ❖ Lists - Linked lists are used to represent hierarchical knowledge
- ❖ Trees and Graphs - These are the techniques which represent hierarchical knowledge. LISP, the programming language of AI, was developed to process lists and trees.
- ❖ Semantic networks - These structure have nodes and links stored as propositions.
- ❖ Schemas - These are used to represent common sense or stereotyped knowledge.
 - Frames - Frames describe objects and consist of a cluster of nodes and links manipulated as a whole. Knowledge is organised in slots. Frames are hierarchically organised.
 - Scripts - Scripts describe event rather than objects. They consist of stereotypically ordered causal or temporal chain of events.
- ❖ Rule-based representations - This technique is used in specific problem-solving contexts. It involves production rules containing if-then or situation-action pairs.
- ❖ Logic-based representations - This may use deductive or inductive reasoning. It contains -
 - Facts and premises.
 - Set of rules of propositional logic (Boolean - dealing with complete statements)
 - Set of rules of predicate calculus (It allows use of additional information about objects in the proposition, use of variables and functions of variables.)
 - Measures of certainty - may involve Certainty Factors (eg. If symptom then (CF) diagnosis) which could be derived from expert estimation or from statistical data; Bayesian probability or Fuzzy logic (in which the concepts or information itself

has some associated certainty value).

3.2 Weak - Slot and Filler Structures

This data structure enables attribute values to be retrieved quickly. Properties of the relations are easy to describe in this data structures. It allows ease of consideration as it embraces aspects of object oriented programming.

A slot is an attribute value pair in its simplest form. A filler is a value that a slot can take which could be a numeric, string (or any data type) value or a pointer to another slot. A weak slot and filler structure does not consider the content of the representation.

We will study two types of weak slot and filler structures namely semantic nets and frames.

Knowledge Representation and Inferencing using Semantic Net

Semantic networks are an alternative to predicate logic as a form of knowledge representation. The idea is that we can store our knowledge in the form of a graph, with nodes representing objects in the world, and arcs representing relationships between those objects. The physical attributes of a person can be represented as in **Fig. 3.1**.

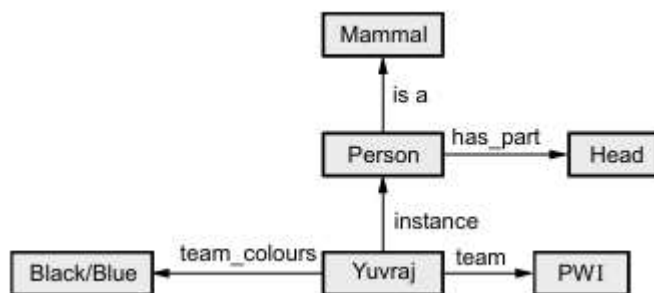


Fig. 3.1 A semantic network.

These values can also be represented in logic as: is a (person, mammal), instance (Yuvraj, person) team (Yuvraj, PWI). We have already seen how conventional predicates such as lecturer (Poonam) can be written as instance (Poonam, lecturer). Recall that isa and instance represent inheritance and are popular in many knowledge representation schemes. But we have a problem: how we can have more than 2 place predicates in semantic nets? For e.g. score (PWI, India, 20). Solution is that, create new nodes to represent new objects either contained or alluded to in the knowledge, game and fixture in the current example. Relate information to nodes and fill up slots.

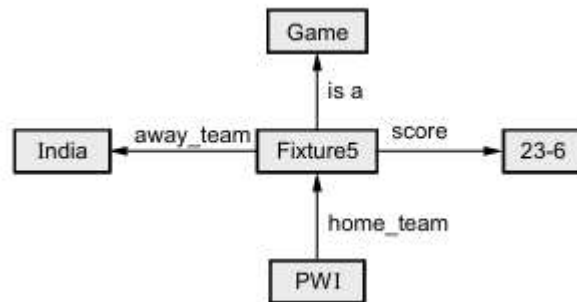


Fig. 3.2 A semantic network for n-place predicate.

As a more complex example consider the sentence: Ram gave Shyam the book. Here we have several aspects of an event.

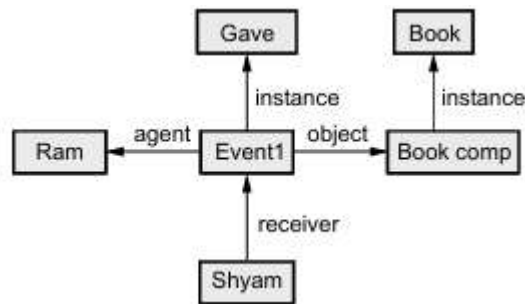


Fig. 3.3 A semantic network for a sentence.

Semantic Networks or Semantic Net

- ❖ It provides graphical aids for visualizing a knowledgebase and efficient algorithms for inferring properties of an object on the basis of its category membership.
- ❖ Semantic network is graph which has nodes representing objects and their categories and arc representing relationships between objects.
- ❖ A typical graphical notation displays object or category names in oval or boxes and connects them with the labelled arcs.
- ❖ Inheritance association (is a relationship) can be described using semantic network. It is useful for inferencing a query on the object. It can happen that a particular object is subset of more than one category (multiple inheritance association) then inferencing algorithm might find two or more conflicting values answering the query for that object.

For example: whale is a mammal, Ram is a person.

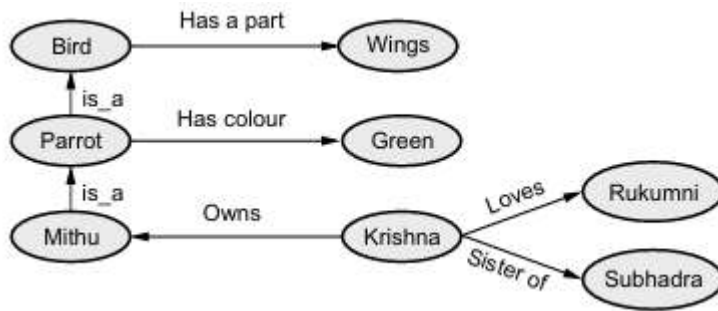


Fig. 3.4 Semantic network.

- ❖ Inverse links is one common inferencing technique used in semantic network.

For example:

$\forall P, B \text{ HasBrother } (P, B) \Leftrightarrow \text{BrotherOf } (B, P)$

Here relationship is two way

- ❖ Semantic network provide direct indexing for objects, categories and the links between them.
- ❖ Semantic network can represent default values for categories.

For example:

If there is category WildAnimals with following descriptions,

Category (WildAnimals)

Then, we specify default value for legs of WildAnimals, $\forall x x \in \text{WildAnimals} \Rightarrow \text{Legs } (x, 4)$ [It means that if any animal who is WildAnimals will have four legs. We need not specify again].

$\text{Tiger} \in \text{Wild Animals}$ [It means that tiger has four legs].

The semantic network would like as shown in **Fig. 3.5**.

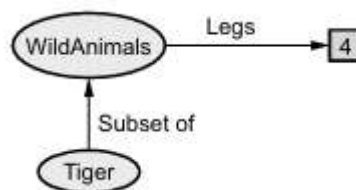


Fig. 3.5 Semantic network.

- ❖ The major drawback of semantic network is, it represents binary relationships. Solution for this is, we need to convert n-ary associations into binary by reifying the propositions as events.
- ❖ If some things are unable to expressed in semantic network then semantic network system provides procedural routines attached with the network of objects. Procedure attachment is a technique whereby a query about a certain relation result in a call to a

special procedure designed for that relation.

Inference in a Semantic Net

Basic inference mechanism: follow links between nodes. Two methods are there to do this:

Intersection search

One of the early ways that semantic nets were used was to find relation among objects by spreading activation out from each of the two nodes.

The notion that spreading activation out of two nodes and finding their intersection finds relationships among objects. This is achieved by assigning a special tag to each visited node. Many advantages are there including entity-based organisation and fast parallel implementation. However very structured questions need highly structured networks.

For e.g., in the **Fig. 3.6** below intersections search can be used to answer the question like, "What is the connection between India and blue?". To answer more structural questions, the networks themselves must be highly structured.

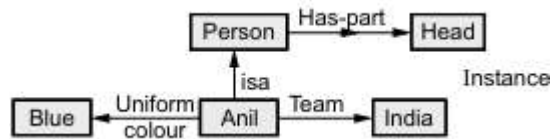


Fig. 3.6 Intersection search in semantic network.

❖ Representing non-binary predicates

Semantic nets are used to represent relationship that appear as ground instances of binary predicates in predicate logic.

Arcs in the **Fig. 3.7** can be represented in logic as: home_team (Cricket, India)

Instance (Cricket, game)

Visiting_team (Cricket, India)

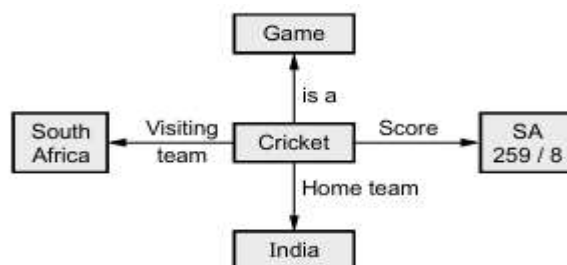


Fig. 3.7 Representing non-binary predicates in semantic network.

❖ Representing some important distinctions

There should be a difference between the link that defines a new entity and one that relates two existing entities:

Both nodes represent an object that exists independently of their relationships to each other. But suppose we want to represent the fact like "Anil" is taller than Raj".

The nodes H_1 and H_2 represent Anil's and Raj's height respectively. Sometimes it is useful to introduce the arc value to make the distinction clear. This is shown in the following **Fig. 3.10**.

The procedure that operates on nets such as this can be used to exploit the fact that some arcs such as height define new entities whereas some arcs like greater than and value merely describes the relationship among existing entities.

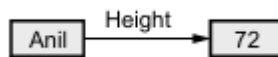


Fig. 3.8 Semantic network.

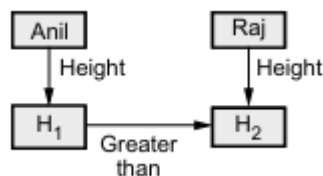


Fig. 3.9 Semantic network.

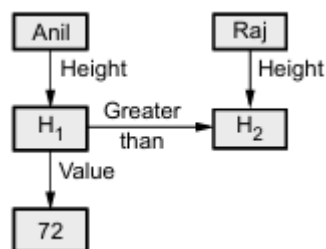


Fig. 3.10 Representation distinction in semantic network.

Inheritance

AI researchers have refined the notion of inheritance. It is called a specialized inferencing technique for representing properties of classes, exceptions to inherited properties, multiple super classes and structured concepts with specific relationships among the structural elements.

Artificial Intelligence(AI) with it's Applications

The isa and instance representation provide a mechanism to implement this. Inheritance also provides a means of dealing with default reasoning. Eg. we could represent:

Emus are birds.

Typically birds fly and have wings.

Emus run. in the following Semantic net:

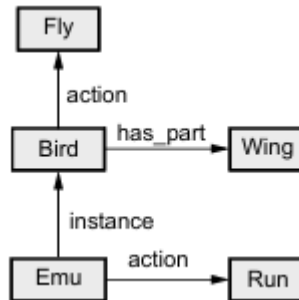


Fig. 3.11 A semantic network for a default reasoning.

In making certain inferences we will also need to distinguish between the links that defines a new entity and holds its value and the other kind of link that relates two existing entities. Consider the example shown where the height of two people is depicted and we also wish to compare them.



Fig. 3.12 Two heights.

We need extra nodes for the concept as well as its value. Special procedures are needed to process these nodes, but without this distinction the analysis would be very limited.

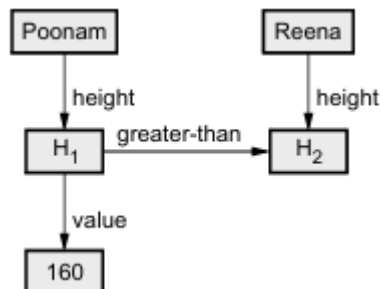


Fig. 3.13 Comparison of two heights.

Extending Semantic Nets

Here we will consider some extensions to semantic nets that overcome a few problems or extend their expression of knowledge. Hendrix developed the partitioned semantic network. It was designed to represent the difference between the description of an individual object or process. It is also used to present description of a set of objects. These set of descriptions involves quantification.

Partitioned networks: Partitioned semantic networks allow for propositions to be made without commitment to truth and ! expressions to be quantified. Basic idea: Break network into spaces which consist of groups of nodes and arcs and regard each space as a node, Consider the following: Anil believes that the earth is flat. We can encode the proposition the earth is flat in a space and within it have nodes and arcs represent the fact (**Fig. 3.14**). We can have nodes and arcs to link this space and rest of the network to represent Anil's belief.

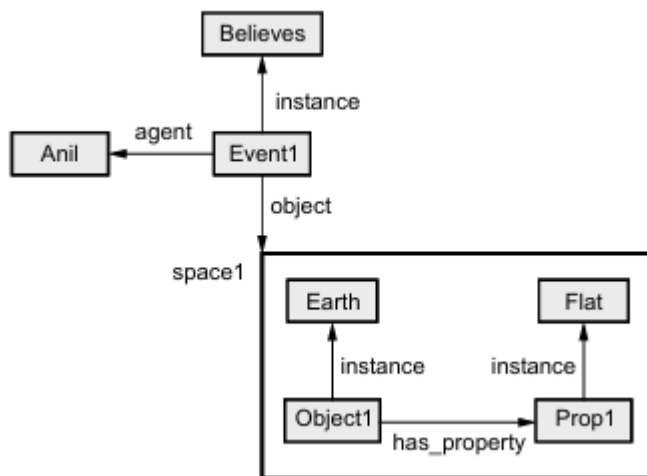


Fig. 3.14 Partitioned network.

Now consider the quantified expression: Every parent loves their child To represent this we: ! Create a general statement, GS, special class. ! Make node g an instance of GS. Every element will have at least 2 attributes: - a form that states which relation is being asserted. - one or more forall () or exists () connections -- these represent universally quantifiable variables in such statements e.g. x, y in $\text{parent}(x) : \text{child}(y) \text{ loves } (x, y)$. Here we have to construct two spaces one for each x, y . Note that we can express variables as existentially qualified variables and express the event of love having an agent p and receiver b for every parent p which could simplify the network. Also If we change the sentence to every parent loves child then the node of the object being acted on (the child) lies outside the form of the general statement. Thus it is not viewed as an existentially qualified variable whose value may depend on the agent.

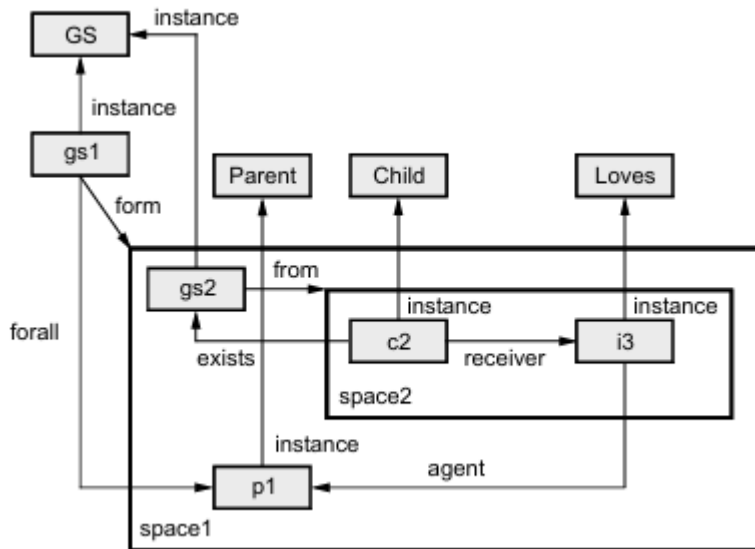


Fig. 3.15 Partitioning network.

So we could construct a partitioned network as in above figure.

In general, to represent

- ❖ Create general statement GS. This is a special class.
- ❖ Make node 'g' an instance of GS.
- ❖ Every element will have atleast 2 attributes -
 - a form that states which relations is being asserted.
 - for 'all $\forall$ ' and 'exists $\exists$ ' connections, that represent universally quantified variables and sentences.

Suppose now we consider statement, "Every dog has bitten a postman".

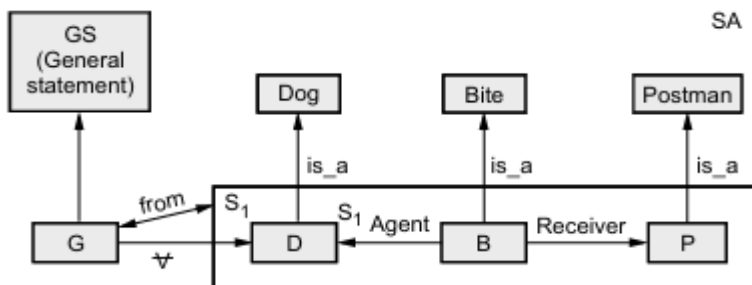


Fig. 3.16 (a) "Every dog has bitten a postman" - Partitioned semantic network.

Above partitioned semantic network has two partitions S_1 and SA . Node G is an instance of the special class of general statements about the world, comprising link statement form and one universal quantifier $\forall$.

Suppose we wish to make a specific statement, "The dog Tipya has bitten the postman Tukaram". Then following is the partitioned semantic net for the statement.

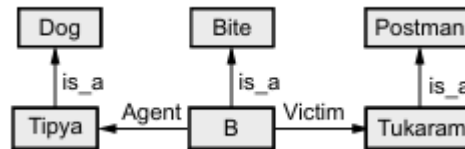


Fig. 3.16 (b) Partitioned semantic net.

The partitioning of semantic network makes the semantic network logically more adequate. Because of partitioning one can distinguish between individuals and sets of individuals.

Semantic network is used in building natural language front-ends for databases and for programs to deduct information from databases.

Parts of Semantic Network

- ❖ Lexical: Symbols are allowed to represent vocabulary.
- ❖ Structural: Symbols can be arranged in certain form.
- ❖ Procedural: Edit, create, delete procedure can be applied.
- ❖ Semantic: It is a way of associating meaning which semantic nets can depict.

Advantages of Semantic Nets

- ❖ Semantic network can represent default values of different categories.
- ❖ Semantic networks are simple and easy to understand.
- ❖ Semantic networks are easy to translate in to prolog.
- ❖ Semantic network arcs represent relationship between nodes.
- ❖ In semantic networks the relationships are handled by pointers.
- ❖ Semantic networks provide good visualization. Being diagrammatic representation they are easy to view.

Disadvantages of Semantic Networks

- ❖ Semantic networks represent relationships between the objects which are only binary relations.

- ❖ Nodes and arcs may not have perfect values always.

Limitations of Semantic Networks

- ❖ Lack of link names standards, make it difficult to understand the net meaning.
- ❖ Even the nodes naming is not standard. If a node is labelled "car" this may mean
 - The class of a car
 - A specific car
 - The concept of a car
- ❖ Answering negative query like "is XYZ a car" takes a very long time.
- ❖ Semantic nets are logically inadequate because they cannot define knowledge in a way logic can.
- ❖ Semantic nets can be used better in representing binary relations, but not all types of relations.
- ❖ Logic enhancements have been made and heuristic enhancements have been tried by attaching procedures to the nodes in the semantic nets. The procedures will be executed when the node is activated.

Knowledge Representation and Inferencing using Frames

Natural language understanding requires inference i.e., assumptions about what is typically true of the objects or situations under consideration. Such information can be coded in structures known as frames.

Frames can also be regarded as an extension to Semantic nets. Indeed it is not clear where the distinction between a semantic net and a frame ends. Semantic nets was initially used to represent labelled connections between objects. As tasks became more complex the representation needs to be more structured. The more structured the system it becomes more beneficial to use frames. A frame is a collection of attributes or slots and associated values that describe some real world entity.

Frames on their own are not particularly helpful but frame systems are a powerful way of encoding information to support reasoning. Set theory provides a good basis for understanding frame systems. Each frame represents: a class (set), or an instance (an element of a class).

Need of frames

Frame is a type of schema used in many AI applications including vision and natural language processing. Frames provide a convenient structure for representing objects that are typical to a stereotypical situations. The situations to represent may be visual scenes, structure of

Artificial Intelligence(AI) with it's Applications

complex physical objects, etc. Frames are also useful for representing common sense knowledge. As frames allow nodes to have structures they can be regarded as three-dimensional representations of knowledge.

A frame is similar to a record structure and corresponding to the fields and values are slots and slot fillers. Basically it is a group of slots and fillers that defines a stereotypical object. A single frame is not much useful. Frame systems usually have collection of frames connected to each other. Value of an attribute of one frame may be another frame.

A frame for a book is given below.

Slots	Fillers
publisher	Technical
Title	Artificial Intelligence
Author	Ravindra W.
Edition	Second
Year	2012
Pages	480

The above example is simple one but most of the frames are complex. Moreover with filler slots and inheritance provided by frames, powerful knowledge representation systems can be built.

Frames can represent either generic or frame. Following is the example for generic frame.

Slot	Fillers
Name	Computer
specialization_of	a_kind_of machine
Types	(desktop, laptop,mainframe,super) if-added: Procedure ADD_COMPUTER
Speed	default : faster if-needed: Procedure FIND_SPEED
Location	(home,office,mobile)
under_warranty	(yes, no)

The fillers may have values such as computer in the name slot or a range of values as in types slot. The procedures attached to the slots are called procedural attachments. There are mainly three types of procedural attachments: if-needed, default and if-added. As the name implies if-needed types of procedures will be executed when a filler value is needed. Default value is taken if no other value exists. Defaults are used to represent common sense knowledge. Common sense is generally used when no more situation specific knowledge is available.

The if-added type is required if any value is to be added to a slot. In the above example, if a new type of computer is invented ADD_COMPUTER procedure should be executed to add

that information. An if-removed type is used to remove a value from the slot.

Frame knowledge representation

Consider the example

Person	
isa :	Mammal
cardinality :	6,000,000,000
* handed :	Right
Adult-Male	
isa :	Person
cardinality :	2,000,000,000
* height :	5-10
Cricket-Player	
isa :	Adult-Male
cardinality :	624
* height :	6-1
* bats :	equal to handed
* batting-average :	90
* team :	
* uniform-color :	
Batsman	
isa :	Cricket player
cardinality :	376
* batting-average :	90
Cricket-Team	
isa :	Team
cardinality :	25
* team-size :	11
* manager :	

Fig. 3.17 A simple frame system.

Here the frames Person, Adult-Male, Cricket-Player and Cricket-Team are all classes and the frames Sachin-Mumbai and Cheteshwar-Gujarath are instances.

Distinction between Sets and Instances

It is important that this distinction is clearly understood. Cheteshwar-Gujrat can be thought of as a set of players or as an instance of a Cricket-Team. If Cheteshwar-Gujrat were a class then

- ❖ Its instances would be players.

- ❖ It could not be a subclass of Cricket-Team otherwise its elements would be members of Cricket-Team which we do not want. Instead we make it a subclass of Cricket-Player and this allows the players to inherit the correct properties enabling us to let the Cheteshwar-Gujrat to inherit information about teams. This means that Cheteshwar-Gujrat is an instance of Cricket-Team. BUT There is a problem here:
- ❖ A class is a set and its elements have properties.
- ❖ We wish to use inheritance to bestow values on its members.
- ❖ But there are properties that the set or class itself has such as the manager of a team. This is why we need to view Cheteshwar-Gujrat as a subset of one class players and an instance of teams. Solution is to create MetaClasses. The basic metaclass is Class, and this allows us to define classes which are instances of other classes, and (thus) inherit properties from this class. Inheritance of default values occurs when one element or class is an instance of a class.

Slots as objects

How can we represent the following properties in frames?

- ❖ Having attributes such as weight, age be attached and make sense.
- ❖ Putting constraints on values such as age being less than a hundred.
- ❖ Assigning default values
- ❖ Defining rules for inheritance of values such as children inheriting parent's names.
- ❖ Defining rules for computing values.
- ❖ Having many values for a slot.

A slot is a relation that maps from its domain of classes to its range of values. A relation is a set of ordered pairs so one relation is a subset of another. Since slot is a set, the set of all slots can be represented by a metaclass called Slot, say.

NOTE the following: (Features of frame)

- ❖ Instances of SLOT are slots.
- ❖ Associated with SLOT are attributes that each instance will inherit.
- ❖ Each slot has a domain and range.
- ❖ Range is split into two parts, one the class of the elements and the other is a constraint which is a logical expression, if it is absent it is taken to be true.
- ❖ If there is a value for default then it must be passed on unless an instance has its own value.
- ❖ The to-compute attribute involves a procedure to compute its value. E.g. in Position where we use the dot notation to assign values to the slot of a frame.

- ❖ Transfers through the lists other slots from which values can be derived from inheritance.

Interpreting frames

A frame system interpreter must be capable of the following in order to exploit the frame slot representation:

- ❖ Consistency checking - when a slot value is added to the frame relying on the domain attribute and that the value is legal using range and range constraints.
- ❖ Propagation of definition values along isa and instance links.
- ❖ Inheritance of default values along isa and instance links.
- ❖ Computation of value of slot as needed.
- ❖ Checking that only correct number of values are computed.

Reasoning using frame

- ❖ A frame can be attached to another frame, thereby resulting in a network of frames.
- ❖ The main task of action frames is to provide facility for procedural attachment and help transforming from initial to goal state. It also helps in breaking the entire problem into sub-tasks which can be described as top-down methodology.
- ❖ A task can be represented using these action frames.
- ❖ Reasoning using frames is done by instantiation.
 - In instantiation process a given situation is matched with frames that are already in existence.
 - The reasoning process tries to match the frame with the situation and latter fills up slots for which values must be assigned.
 - A particular situation is depicted by assigning the values to the slots and by this, the reasoning process try to move toward a goal.
- ❖ Generally, if a given slot characteristic is not present, the slot provides a default values for that characteristics.
- ❖ But mostly situations are not static and there is a deviation in characteristics, in such cases the values of the corresponding slots are updated that reflect clear current situation.
- ❖ Reasoning using frames allows one to move from one frame to another to match the current situation. The process builds up a wide network of frames, thereby facilitating one to build a knowledge base for representing knowledge about common-sense.

Converting: Semantic Network and Frames

a) Semantic network

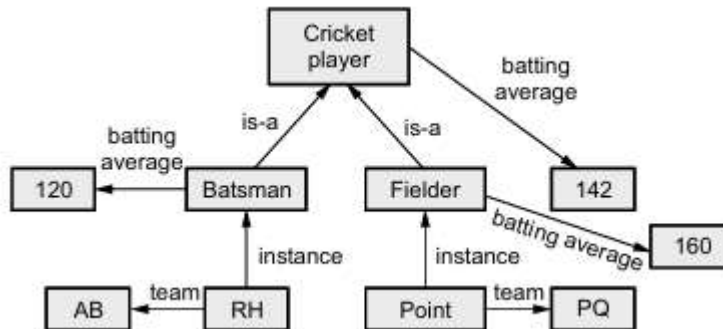


Fig. 3.18 Semantic network.

b) Frame

Cricket player	Fielder	Point
is-a : male	is-a : Cricket player	Instance : Cricket player
Batting avg. : 120	Batting avg. : 160	Team : PQ
Batting : RH		
Team : AB		
:		

Access Procedure for Frames

- ❖ Class constructor: Direct super-class
- ❖ Instance constructor: Instance is connected to class by is-a slot.
- ❖ Slot writer: Installs slot value.
- ❖ Slot reader: Retrieves slot value.

Advantages of Frames

- ❖ Frames are easy to understand.
- ❖ It is easy to represent new properties and relations in frames.
- ❖ Default information can be included.
- ❖ Missing values can be detected in frames.

Artificial Intelligence(AI) with it's Applications

- ❖ A frame collects information about an object at a single place in an organized fashion.
- ❖ Frames provide a way of associating knowledge with objects.
- ❖ By relating slots to other kinds of frames, a frame can represent typical structure involving an object; these can be very important for reasoning based on limited information.
- ❖ Frames allow data that are stored, computed and to be treated in a uniform manner, (E.g., Age might be stored, or might be computed from BIRTHDAY.)
- ❖ Frames are a relatively efficient way of implementing A.I. applications.

Disadvantages of Frames

- ❖ Frames are very general and do not have well-formed rule for construction and specification.
- ❖ Frame building procedure takes long time.
- ❖ Frames encourage baroque representations; little guide to good structuring of a domain.
- ❖ Some things that can be represented in logic cannot be represented well in frames.
- ❖ It is not possible to quantify over slots. For example, there is no way to represent "Some student made 100 in the exam".
- ❖ Inheritance can cause trouble.
- ❖ Slot fillers must be "real" data. For example, it is not possible to say that Shamu is a butcher or a baker, since there is no way to deal with a disjunction in a slot filler.
- ❖ Inference tends to become complex and ill-structured.
- ❖ It is necessary to repeat the same information to make it usable from different viewpoints, since methods are associated with slots or particular object types. (For example, it may be easy to answer "whom does Krishna love?" but hard to answer "who loves Radha?")
- ❖ There is more than one way to break down the world into taxonomies.
- ❖ Frames are good for representing static situations; they are not so good for representing fluents and things that change over time.

Declarative and Procedural Frames

A frame that just contains only description about objects is called a declarative type / factual / situation frame.

For example, a computer centre and each equipment of the centre that as shown below.

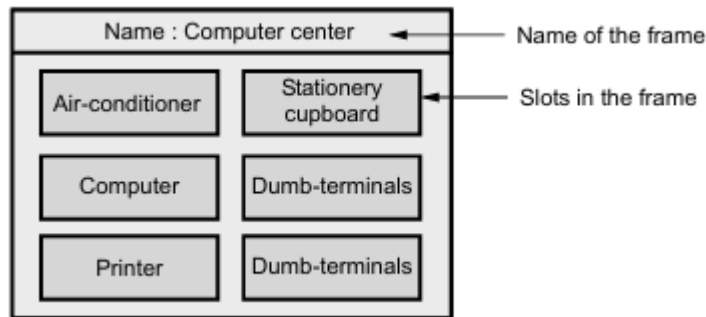


Fig. 3.19 A sample frame of a computer centre.

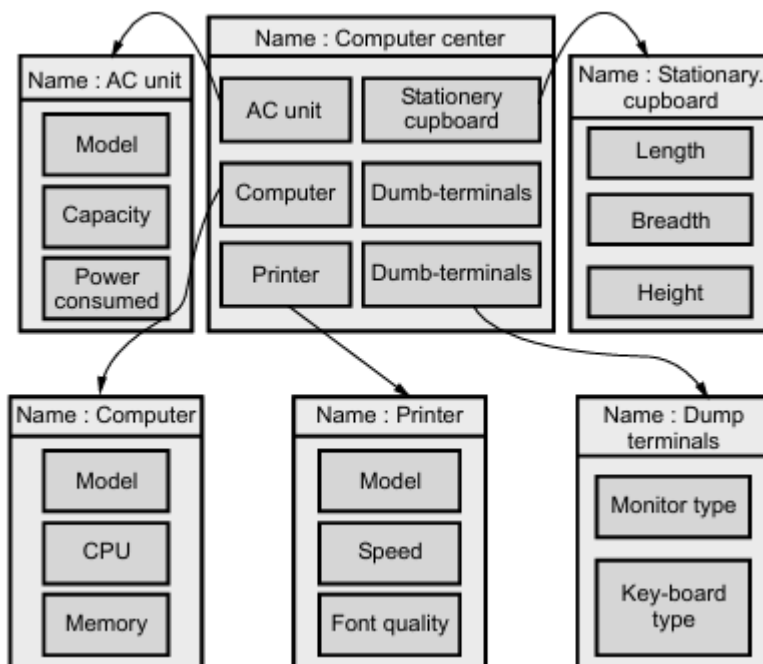


Fig. 3.20 Linking of sub frames.

Apart from the declarative part in a frame, it is possible to attach slots which explain how to perform things. Or it is possible to have procedural knowledge represented in a frame.

Such frames which have procedural knowledge embedded in it are called action procedure frames.

The action-frame has the following slot,

Actor slot - Which holds information about who is performing the activity.

Object slot - This frame has information about the item to be operated on.

Source slot - Source slot holds information from where the action has to begin.

Destination slot - Holds information about the place where the action has to end.

Task slot - This generates the necessary sub-frames required to perform the operation.

For example, consider **Fig. 3.21** showing a procedural frame, for cleaning the jet of a carburettor in a scooter.

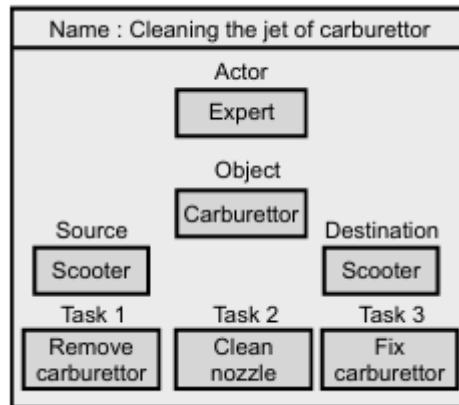


Fig. 3.21 A procedural frame.

This frame merely describes that, the expert in order to clean the nozzle of the scooter has to merely perform, the following operations,

- ❖ Removing the carburettor from the scooter.
- ❖ Opening it up to expose all parts.
- ❖ Cleaning the nozzle.
- ❖ Refixing it in the scooter.

Since, entire operation is done on a scooter, the source and destination slots are the same viz. the scooter.

Fig. 3.22 shows that for each task slot, it is possible to have other frames (procedural) of how to do it.

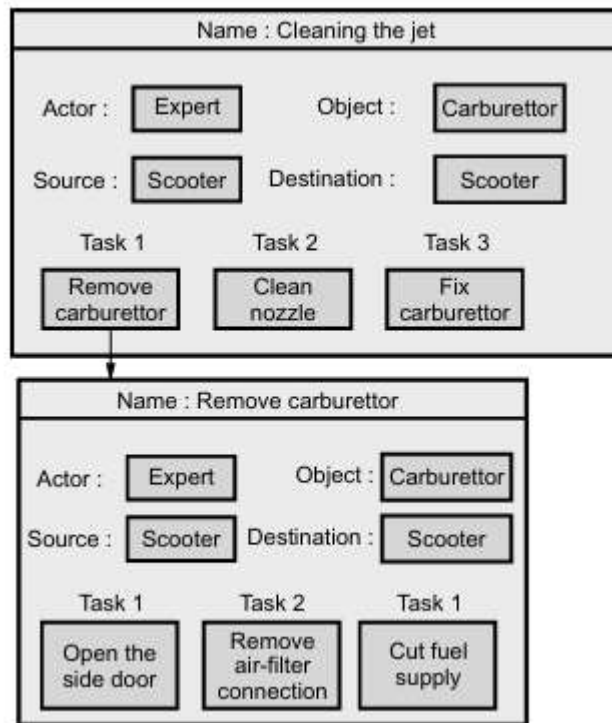


Fig. 3.22 Linking of procedure sub-frames.

3.3 Strong Slot and Filler Structures

To avoid the difficulties with Frames and Nets, Schank and Rieger offered network-like representations that would have implied uses and built-in semantics conceptual dependencies and scripts. These representations are termed as strong slot and filler structures.

Strong Slot and Filler Structures typically represent links between objects according to more rigid rules.

The specific notions of what types of object and relations between them, are provided in this structure.

These structures represent knowledge about common situations.

Knowledge Representation and Rule based Inferencing using Conceptual Dependency (CD)

A number of authors in AI have addressed the question of the 'concept'-based organization of knowledge. To know this idea, we will consider a verb-oriented organization of knowledge proposed by Schank which is Conceptual Dependency Grammar and then the Conceptual graphs.

Artificial Intelligence(AI) with it's Applications

Conceptual dependency Grammar is a powerful representation formalism, which has been used in a number of Virtual Reality (VR) Systems. In a VR system,

- ❖ The end user of the system can interact with the system in a manner analogous to interacting with another human being at a 'physical' level.
- ❖ A VR end user can 'bounce' a ball on a VR screen, and can catch a ball thrown by the system.
- ❖ A VR end user can ride a bike.

Conceptual dependency was originally developed to represent knowledge acquired from natural language input.

The goals of this theory are,

- ❖ To be independent of the words used in the original input. It has been argued that the representation (CD) is independent of the language in which the sentences were originally stated.
- ❖ To help in the drawing of inference from sentences.
- ❖ That is to say: For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.

Conceptual Dependency originally developed to represent knowledge acquired from natural language input. Conceptual dependency was derived as a form of semantic network that would have specific types of links to be used for representing specific pieces of information in English sentences, the action of the sentence, the objects affected by the action or that brought about the action and the modifiers of both actions and objects.

CD has been used by many programs that portent to understand English (some programs are MARGIE, SAM, PAM). CD was developed by Schank et al..

CD provides

- ❖ A structure into which nodes representing information can be placed a specific set of primitives at a given level of granularity.
- ❖ The sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.
- ❖ The agent and the objects are represented
- ❖ The actions are built up from a set of primitive acts which can be modified by tense.

Primitives Acts in CDs

CD defined 11 primitive actions, called ACTs

Artificial Intelligence(AI) with it's Applications

Every possible action can be categorized as one of these 11 primitives.

An ACT would form the center of the CD, with links attaching the objects and modifiers.

Examples of Primitive Acts are,

ATRANS

- ❖ Transfer of an abstract relationship. e.g. give.

PTRANS

- ❖ Transfer of the physical location of an object. e.g. go.

PROPEL

- ❖ Application of a physical force to an object. e.g. push.

MTRANS

- ❖ Transfer of mental information. e.g. tell.

MBUILD

- ❖ Construct new information from old. e.g. decide.

SPEAK

- ❖ Utter a sound. e.g. say.

ATTEND

- ❖ Focus a sense on a stimulus. e.g. listen, watch.

MOVE

- ❖ Movement of a body part by owner. e.g. punch, kick.

GRASP

- ❖ Actor grasping an object. e.g. clutch.

INGEST

- ❖ Actor ingesting an object. e.g. eat.

EXPEL

- ❖ Actor getting rid of an object from body. e.g. sweat, ????

Six primitive conceptual categories provide building blocks which are the set of allowable dependencies in the concepts in a sentence:

PP - Real world objects.
ACT - Real world actions.
PA - Attributes of objects.
AA - Attributes of actions.
T - Times.
LOC - Locations.

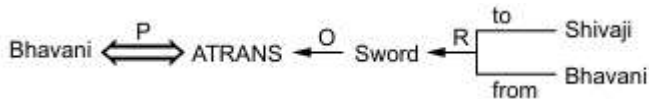
Connecting all these things together,

Consider the example,

For example: Consider statement,

Bhavani gave shivaji a sword.

- ❖ Following is CD representation of above statement.



- ❖ Arrows indicate the direction of dependency.
- ❖ Letters above indicate certain relationships which are as follows -
 - P - Indicates past tense.
 - O - Object.
 - R - Recipient-donor.
- ❖ Double arrows ($\Leftrightarrow$) indicate two-way links between the actor (PP) and action (ACT).
- ❖ The actions are built from the set of primitive acts (see above).
 - These can be modified by tense etc.
- ❖ The use of tense and mood in describing events is extremely important and Schank introduced the following modifiers :
- ❖ **p** - past
- ❖ **f** - future
- ❖ **t** - transition
 - t_s - start transition
 - t_f - finished transition
- ❖ **k** - continuing
- ❖ **?** - interrogative
 - /-negative

delta - timeless

- ❖ c - conditional the absence of any modifier implies the present tense.
- ❖ I - instrument e.g. eat with a spoon.
- ❖ D - destination e.g. going home.

Example: So the present tense of above statement in example 1 becomes.

- ❖ Bhawani gives Shivaji a sword and the CD representation is,
- ❖ The $\Leftrightarrow$ has an object (actor), PP and action, ACT. That is $PP \Leftrightarrow ACT$.
- ❖ The triple arrow ($\Leftrightarrow$) is also a two link but between an object, PP and its attribute, PA . I.e. $PP \Leftrightarrow PA$. It represents is a type dependencies.
- ❖ e.g. Dave $\Leftrightarrow$ lecturer, Dave is a lecturer.

Rules used in CD

- ❖ Primitive states are used to describe many state descriptions such as height, health, mental state, physical state.
- ❖ You can also specify things like the time of occurrence in the relationship.
- ❖ There are many more physical states than primitive actions. They use a numeric scale.
- ❖ The rules that are used for representing the knowledge in conceptual dependency are shown in **Fig. 3.23**.
- ❖ Here, the first column contains the rules, the second contains examples of their use; and the third contains an English version of each example.

The rules shown in **Fig. 3.23** can be interpreted as follows -

1. **Rule 1** - This relation describes the relationship between an actor and the event he or she causes. It is a two way dependency since neither actor nor event can be considered primary. The letter P above the dependency link indicates past tense.

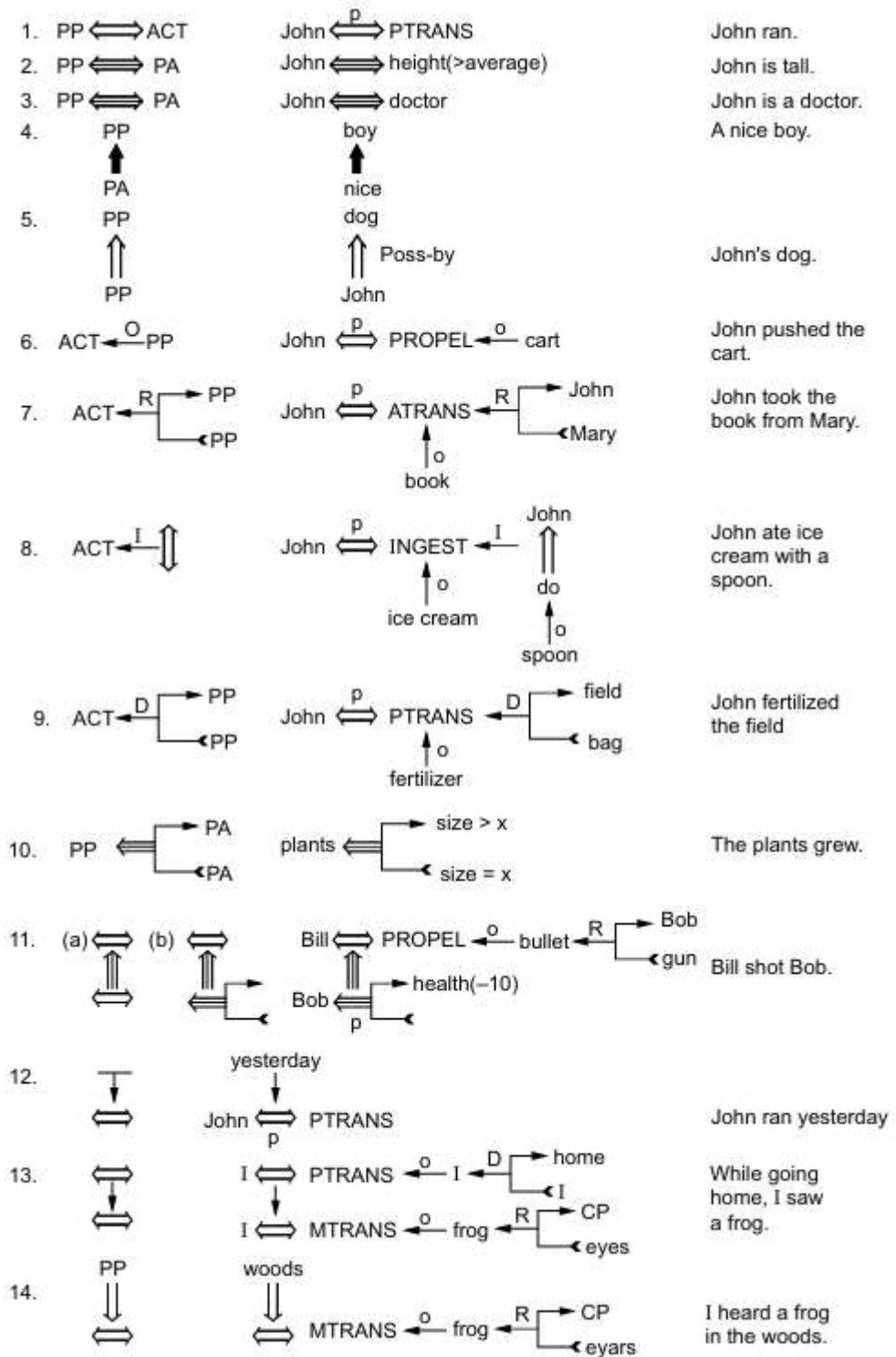


Fig. 3.23 CD dependencies.

2. **Rule 2** - This rule describes the relationship between PP and PA. Height are represented in CD as numeric scales. Here by showing that height of John is greater than average we indicate that "John is tall".
3. **Rule 3** - This rule describes the relationship between two PPs. One or which belongs to the set defined by the other.
Here, it is shown that John is a doctor.
4. **Rule 4** - This rule shows a relationship between a PP and a PA, i.e., between a PP and an attribute that has already been predicated of it. The direction of arrow is toward the PP.
Here, it is show that "boy is nice".
5. **Rule 5** - This rule gives the relationship between two PPs, one of which provides a particular kind of information about the other. The three most common types of information to be provided in this way are possession (shown as POSS-BT). Again direction of arrow is towards PP i.e., dog as in example, it is shown "John's dog".
6. **Rule 6** - This rule describes the relationship between ACT and PP i.e., the object of that ACT. The direction of the arrow is toward the ACT, since, the contest of the specific ACT determines the meaning of the object relation.
It is shown here, "John pushed the cart".
7. **Rule 7** - It describes the relationship between an ACT and the source and the recipient of the ACT.
In the example,
"John took the book from Mary".
Mary is source, book is an object and John is recipient.
8. **Rule 8** - This rule describes the relationship between an ACT and the instrument with which it is performed.
The instrument must always be a full conceptualization (i.e., it must contain an ACT), not just a single physical object.
Here, the example is
"John ate ice cream with a spoon".
i.e., John's action is eating and instrument used is spoon.
9. **Rule 9** - This rule describes the relationship between an ACT and its physical source and destination.
In the example, it is shown that the statement "John fertilized the fields" Here, John is physical source, fertilize is an act and field is the destination.
10. **Rule 10** - It represents live relationship between a PP and a state in which it started and another in which ended.
As here, "The plants grow".
One PP (object) is plant and there are its two states i.e., size = x and size $> x$ i.e., showing that initially size of plant was x but now in new state, its size is greater than x , so plants grow.

- ❖ **Rule 11** - This describes the relationship between one conceptualization and another that causes it. Notice that the arrows indicate dependency of one conceptualization on another and so point in the opposite direction of the implication arrows. The two forms of the rule describes the cause of an action and the cause of a state change.
- ❖ **Rule 12** - It describes the relationship between a conceptualization and the time at which the event it describes occurred.
For example "John ran yesterday" in which it is shown that John run, but past is shown by P so, John ran and day is indicated above arrow yesterday.
- ❖ **Rule 13** - Describes the relationship between one conceptualization and another that is the time of the first.
The example, "While going home I saw a frog" shows how CD exploits a model of the human information processing system, see is represented as the transfer of information between the eyes and the conscious processor.
- ❖ **Rule 14** - This rule describes the relationship between a conceptualization and the place at which it occurred.

Examples

Ram $\Leftrightarrow$ height(+10). Ram is the tallest.
Ram $\Leftrightarrow$ height(< average). Ram is short.
Ravana $\Leftrightarrow$ health(-10). Ravana is dead Monster.
Ravana $\Leftrightarrow$ mental_state(-10). Ravana is sad.
Vase $\Leftrightarrow$ physical_state(-10) The vase is broken

Following are some of the examples

1. PP $\Leftrightarrow$ ACT

Raj $\Leftrightarrow$ PTRANS
Raj walk

2. PP $\Leftrightarrow$ PA

Raj $\Leftrightarrow$ Scientist
Raj is a scientist

3. PP Boy

↑ ↑
A handsome boy
PA handsome

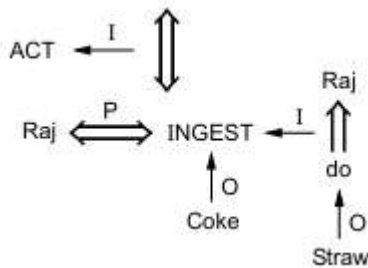
4. PP parrot

↑ ↑ Raj's parrot.
PP Raj

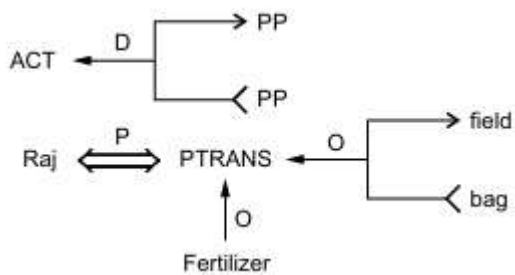
5. $ACT \xleftarrow{O} PP$

$Raj \xrightleftharpoons{P} PROPEL \xleftarrow{O} door$ Raj pulled the door.

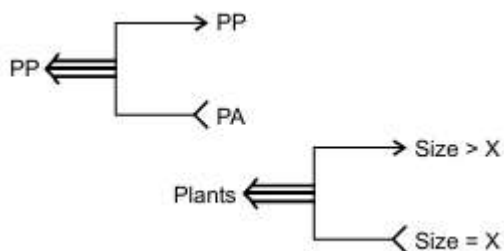
6. Raj drunk coke with a straw.



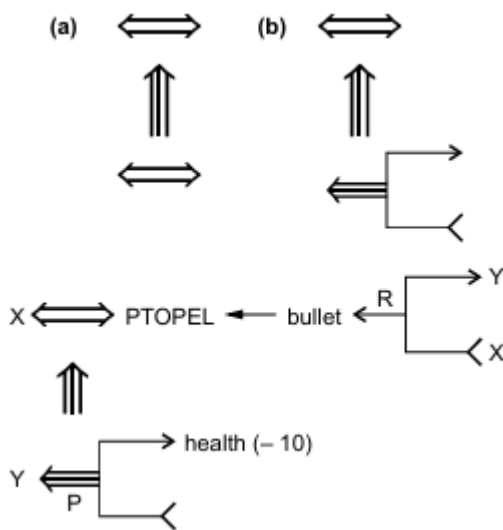
7. Raj fertilized the field



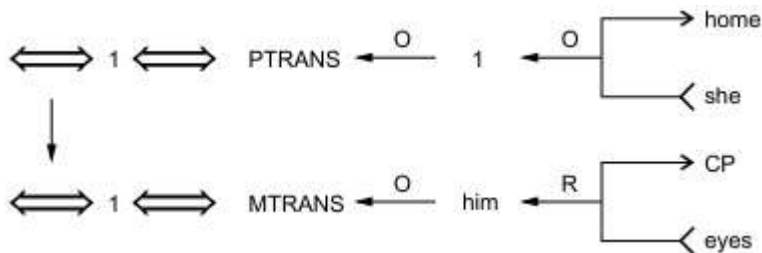
8. The plants grow



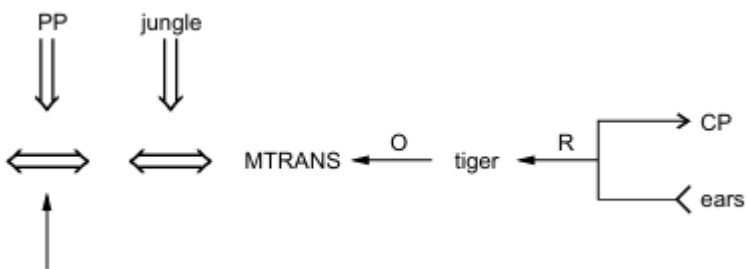
9. X shot y



10. While going home she saw him



11. I heard tiger in the jungle



CD can describes the relationship between a conceptualization and the place of which it occurred.

Now let us consider a more complex sentence:

Since smoking can kill you, I stopped. Let's look at how we represent the inference that smoking can kill:

- ❖ Use the notion of one to apply the knowledge to.
- ❖ Use the primitive act of INGESTing smoke from a cigarette to one.
- ❖ Killing is a transition from being alive to dead. We use triple arrows to indicate a transition from one state to another.
- ❖ Have a conditional, c causality link. The triple arrow indicates dependency of one concept on another.

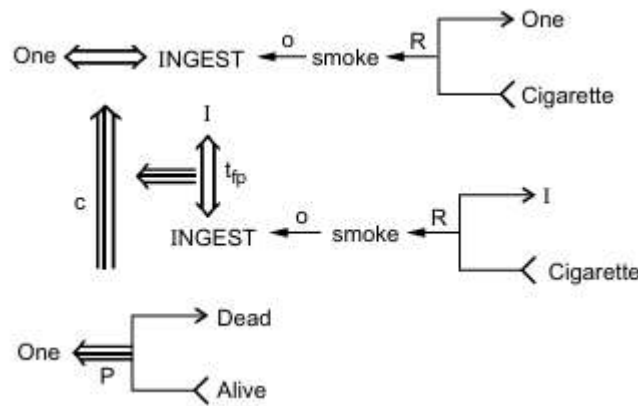


Fig. 3.24 “Since smoking can kill you, I stopped”.

To add the fact that I stopped smoking

- ❖ Use similar rules to imply that I smoke cigarettes.
- ❖ The qualification t_{fp} attached to this dependency indicates that the instance INGESTing smoke has stopped.

Advantages of CD

- ❖ Using these primitives involves fewer inference rules.
- ❖ Many inference rules are already represented in CD structure.
- ❖ The holes in the initial structure help to focus on the points still to be established.
- ❖ The ACT primitives help in representing wide knowledge in a succinct way. In CD representation the verbs that use to transfer mental information such as, see, hear, learn, inform, remember, all can be represented by a single ACT primitive MTRANS.
- ❖ The main goal of CD representation is to make explicit of what is implicit. That is why every statement that is made has not only the actors and objects but also time and location, source and destination.
- ❖ CD brought forward the notion of language independence because all ACTs are language - independent primitives.

CD is a special purpose extension of semantic network in which specific primitives were used in building representations.

Disadvantages of CD

- ❖ Knowledge must be decomposed into fairly low level primitives.
- ❖ Impossible or difficult to find correct set of primitives.
- ❖ A lot of inference may still be required.
- ❖ Representations can be complex even for relatively simple actions. Consider :

Dave bet Frank five pounds that Wales would win the Rugby World Cup.
Complex representations require a lot of storage

Applications of CD

MARGIE

(Meaning Analysis, Response Generation and Inference on English) - model natural language understanding.

SAM

(Script Applier Mechanism) - Scripts to understand stories. See next topic.

PAM

(Plan Applier Mechanism) - Scripts to understand stories.

Schank et al. developed all of the above.

Knowledge Representation and Interfacing using Scripts

The other structured representation developed by Schank (along with Abelson) is the script.

A script is a structure that prescribes a set of circumstances (stereotyped sequences) which could be expected to follow on from one another.

It is similar to a thought sequence or a chain of situations which could be anticipated. It could be considered to consist of a number of slots or frames but with more specialized roles.

Scripts are useful and handy because,

- ❖ In general the events tend to occur in known runs or patterns.
- ❖ Scripts represent existing causal relationships between events.
- ❖ Scripts can represent entry conditions which exist and which allow an event to take place.
- ❖ There are certain prerequisites exist upon events taking place. e.g. when a child progresses through a kindergarten scheme or when a girl purchases a dress.

The components of a script

Entry Conditions

- ❖ These must be satisfied before events in the script can occur.

Results

- ❖ Conditions that will be true after events in script occur.

Props

- ❖ Slots representing objects involved in events.

Roles

- ❖ Persons involved in the events.

Track

- ❖ Variations on the script. Different tracks may share components of the same script.

Scenes

- ❖ The sequence of events that occur. Events are represented in conceptual dependency form.

Features of Scripts

- ❖ For a particular script when applying it, it must be activated and the activation depends on its significance.
- ❖ When certain topic is mentioned in passing, a pointer to that script could be held.
- ❖ When the topic is important, then the script should be opened.
- ❖ The negative aspect is that if too many active scripts exist then one might have too many windows open on the screen or too many recursive calls in a program.
- ❖ Provided that events follow a known trail we can use scripts to represent the actions involved and use them to answer detailed questions.
- ❖ Different trails may be allowed for different outcomes of Scripts (e.g. The bank robbery goes wrong).

Advantages of Scripts

- ❖ Scripts have ability to predict events.
- ❖ Scripts provide an ability for default reasoning when information is not available that directly states that an action occurred.
- ❖ In scripts a single coherent interpretation may be build up from a collection of

observations.

Disadvantages of Scripts

- ❖ Scripts are less general than frames.
- ❖ Scripts are might not be suitable to represent all kinds of knowledge.

Script Example

Construct a script for going to the movie from the viewpoint of movie goer.

SCRIPT : MOVIE WATCHING

TRACK : THEATRE

PROPS : CHAIR MOVIE HALL

TICKET

MONEY

TICKET WINDOW

MOVIE = M

CHANGE

ROLES : S = Movie goes

Tc = Ticket checker

Tg = Ticket giver

O = Owner

entry conditions : S is interested in the movie

S has money.

Results :

S has less money.

O has more money.

S is pleased.

S is not pleased.

Scene 1 entering

S PTRANS S to Theatre
S ATTEND eyes to ticket window
S MBUILD himself in the queue.
S PTRANS S towards ticket window as queue moves.
S MOVES S to buy ticket.

Scene 2 Buying ticket

S MBUILD choice to Tg.
Tg PTRANS TICKET to S.
S ATRANS MONEY to Tg
Tg ATRANS CHANGE to S
S PTRANS S towards Movie hall.

Scene 3 Watching movie

TCMTRANS S for ticket
S PTRANS Ticket to Tc
Tc PTRANS Ticket to S
S PTRANS S in the movie hall.
Tc MTRANS S about his chair.
S PTRANS S in the chair
S MTRANS MOVIE

Scene 4 exiting the theatre

S PTRANS S towards movie hall door
S PTRANS S to out of movie hall

3.4 CYC

The Concept

- ❖ CYC is a big prospective attempt to form a very large knowledgebase aimed at capturing common sense reasoning.
- ❖ The initial goal of CYC was to capture knowledge from a hundred randomly selected articles in the Encyclopaedia Britannica.
- ❖ In CYC both Implicit and Explicit knowledge is encoded.

- ❖ CYC puts emphasis on study of underlying information (assumed by the authors) but not needed to tell to the readers.
Example: Suppose we read that Wellington learned of Napoleon's death. Then we (humans) can conclude Napoleon never knew that Wellington had died.

How do human do this?

Humans require special implicit knowledge or common sense such as,

- ❖ We only die once.
- ❖ You stay dead.
- ❖ You cannot learn of anything when dead.
- ❖ Time cannot go backwards.

Why build large knowledge bases?

Brittleness

- ❖ The specialised knowledge bases are brittle. It is hard to encode new situations and non-graceful degradation in performance. Common sense based knowledge bases should have a firmer foundation.

Form and Content

- ❖ Such type of knowledge representation may not be suitable for AI. Common sense strategies could point out where difficulties in content may affect the form.

Shared Knowledge

- ❖ There should be greater communication among systems with common bases and assumptions.

CYC Coding

CYC can be coded simply by hand. There are special CYCL language such as LISP which can be used for coding. Frame based coding, Multiple inheritance, Generalised inheritance (where any link is not just isa and instance), Slots which are fully fledged objects, can also be used for CYC coding.

For Example: Construct a script for going to the movie from the view point of movie goer.

SCRIPT : MOVIE WATCHING
TRACK : THEATRE
PROPS : CHAIR MOVIE HALL
 TICKET
 MONEY
 TICKET WINDOW
 MOVIE = M
 CHANGE
ROLES : S = Movie goes
 Tc = Ticket checker
 Tg = Ticket giver
 O = Owner
entry conditions : S is interested in the movie
 S has money.

Results :

S has less money.
O has more money.
S is pleased.
S is not pleased.

Scene 1 entering

S PTRANS S to Theatre
S ATTEND eyes to ticket window
S MBUILD himself in the queue.
S PTRANS S towards ticket window as a queue moves.
S MOVES S to buy ticket.

Scene 2 Buying ticket

S MBUILD choice to Tg.
Tg PTRANS TICKET to S.
S ATRANS MONEY to Tg
Tg ATRANS CHANGE to S
S PTRANS S towards Movie hall.

Scene 3 Watching movie

TCMTRANS S for ticket
S PTRANS Ticket to Tc
Tc PTRANS Ticket to S
S PTRANS S in the movie hall.
Tc MTRANS S about his chair.
S PTRANS S in the chair
S MTRANS MOVIE

Scene 4 exiting the theatre

S PTRANS S towards movie hall door
S PTRANS S to out of movie hall

3.5 Schema

- ❖ The overall design of the knowledge database is called the knowledge base schema. Schema are changed infrequently, if at all.
- ❖ A database schema corresponds to the programming language type definition. A variable of a given type has a particular value at given instance which corresponds to an instance of a database schema.
- ❖ Knowledge-base systems have several schemas, partitioned according to the levels of abstraction. At the lowest level is the physical schema, at the intermediate level is the logical schema and at the highest level is a subschema. In general, knowledge-base system support one physical schema, one logical schema and several subschemas.
- ❖ In order for a program to be able to participate intelligently in a dialogue it must be able to participate not only its own beliefs about the world, but also its knowledge of the other dialogue participant's belief about the world, that person's belief, about the computer's belief and so forth.
- ❖ To make computational models of belief, it is useful to divide the issue into two parts - those beliefs that can be assumed to be shared among all the participants in a linguistic event and those that cannot.
- ❖ Shared beliefs can be modeled without and explicit notion of belief in the knowledge base. All that we need to do is to represent the shared beliefs as facts and they will be accessed whenever knowledge about anyone's belief is needed. The representation technique used here to represent shared belief is known as schema.
- ❖ Information-base contains a catalog having many schemas. The presence of multiple schemas allows different applications and different users to work independently. If

schema is not set explicitly, a default schema associated with the user identifier is used, so different users see their own schemas.

- ❖ Another use of schemas is to allow multiple versions of an application - one a production version and other versions being tested to be used on the same knowledge-base.

3.6 Introduction to Handling Uncertainty

A agent working in real world environment almost never has access to whole truth about its environment. Therefore, agent needs to work under uncertainty.

Earlier agents make the epistemological commitment that either the facts (expressed as prepositions) are true, false or else they are unknown. When an agent knows enough facts about its environment, the logical approach enables it to derive plans, which are guaranteed to work.

But when agent works with uncertain knowledge then it might be impossible to construct a complete and correct description of how its actions will work. If a logical agent can't conclude that any particular course of action achieves its goal, then it will be unable to act.

The right thing logical agent can do is, take a rational decision. The rational decision depends on following things,

- ❖ The relative importance of various goals.
- ❖ The likelihood and the degree to which, goals will be achieved.

Dealing with Uncertain Knowledge and Probability Theory

An agent would possess some early basic knowledge of the world (Assume that knowledge is represented in first order logic sentence). Using first order logic to handle real word problem domains fails for three main reasons as discussed below,

Laziness

It is too much work to list the complete set of antecedents or consequents needed to ensure an exception less rule and too hard to use such rules.

Theoretical ignorance

A particular problem may not have complete theory for the domain.

Practical ignorance

Even if all the rules are known, particular aspects of problem are not checked yet or some details are not considered at all (missing out the details).

Artificial Intelligence(AI) with it's Applications

The agent's knowledge can provide it with a degree of belief with relevant sentences. To this degree of belief probability theory is applied. Probability assigns a numerical degree of belief between 0 and 1 to each sentence.

Probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance.

Assigning probability of 0 to a given sentence corresponds to an unequivocal belief saying that sentence is false. Assigning probability of 1 corresponds to an unequivocal belief saying that the sentence is true. Probabilities between 0 and 1 correspond to intermediate degree of belief in the truth of the sentence.

The beliefs completely depends on precepts of agent at particular time. These precepts constitute the evidence on which probability assertions are based. Assignment of probability to a proposition is analogous to saying that whether the given logical sentence (or its negation) is entailed by the knowledge base rather than whether it is true or not. When more sentences are added to knowledge base the entailment keeps on changing. Similarly the probability would also keep on changing with additional knowledge.

All probability statements must therefore, indicate the evidence with respect to which the probability is being assessed. As the agent receives new precepts, its probability assessments are updated to reflect the new evidence. Before the evidence is obtained, we talk about prior or unconditional probability; after the evidence is obtained, we talk about posterior or conditional probability. In most cases, an agent will have some evidence from its precepts and will be interested in computing the posterior probabilities of the outcomes it cares about.

Uncertainty and rational decisions

The presence of uncertainty drastically changes the way an agent makes decision. At particular time an agent can have various available decisions, from which it has to make a choice. To make such choices an agent must have a preferences between the different possible outcomes, of the various plans.

A particular outcome is completely specified state, along with the expected factors related with the outcome.

For example: Consider a car driving agent who wants to reach at airport by a specific time say at 7.30 pm.

Here factors like, whether agent arrived at airport on time, what is the length of waiting duration at the airport are attached with the outcome.

Utility Theory

Utility theory is used to represent and reason with preferences. The term utility in current context is used as "quality of being useful".

Artificial Intelligence(AI) with it's Applications

Utility theory says that every state has a degree of usefulness called as utility. The agent will prefer the states with higher utility.

The utility of the state is relative to the agent for which utility function is calculated on the basis of agent's preferences.

For example: The pay-off functions for games are utility functions. The utility of a state in which black has won a game of chess is obviously high for the agent playing black and low for the agent playing white.

There is no measure that can count test or preferences. Someone loves deep chocolate icecream and someone loves chocochip icecream. A utility function can account for altruistic behavior, simply by including the welfare of other as one of the factors contributing to the agent's own utility.

Decision theory

Preferences as expressed by utilities are combined with probabilities for making rational decisions. This theory, of rational decision making is called as decision theory.

Decision theory can be summarized as,

Decision theory = Probability theory + Utility theory.

❖ The principle of Maximum Expected Utility (MEU)

Decision theory says that the agent is rational if and only if it chooses the action that yields highest expected utility, averaged over all the possible outcomes of the action.

❖ Design for a decision theoretic agent

Following algorithm sketches the structure of an agent that uses decision theory to select actions.

The algorithm

Function: DT-AGENT (percept) returns an action.

Static: belief-state, probabilistic beliefs about the current state of the world. Action, the agent's action.

-  Update belief-state based on action and percept
-  Calculate outcome probabilities for actions, given actions descriptions and current belief-state
-  Select action with highest expected utility given probabilities of outcomes and utility information
-  Return action.

A decision therotic agent that selects rational actions.

The decision theoretic agent is identical, at an abstract level, to the logical agent. The primary difference is that the decision theoretic agent's knowledge of the current state is uncertain; the agent's belief state is a representation of the probabilities of all possible actual states of the world.

As time passes, the agent accumulates more evidence and its belief state changes. Given the belief state, the agent can make probabilistic predictions of action outcomes and hence select the action with highest expected utility.

The Basic Probability Notation

The probability theory uses propositional logic language with additional expressiveness.

The probability theory uses represent prior probability statements, which apply before any evidence is obtained. The probability theory uses conditional probability statements which include the evidence explicitly.

Propositions

- ❖ The propositions (assertions) are attached with the degree of belief.
- ❖ Complex proposition can be formed using standard logical connectives.

For example: $[(\text{Cavity} = \text{True}) \wedge (\text{Toothache} = \text{False})]$ and $[(\text{Cavity} \wedge \neg \text{Toothache})]$ both are same assertions.

The random variable

- ❖ The basic element of language is random variable.
- ❖ It refers to a "part" of the world whose "status" is initially unknown.
For example: In toothache problem 'cavity' is a random variable which can refer to my left wisdom tooth or right wisdom tooth.
- ❖ Random variables are like symbols in propositional logic.
- ❖ Random variables are represented using capital letters. Whereas unknown random variable can be represented with lowercase letter.
For example: $P(a) = 1 - P(\neg a)$
- ❖ Each random variable has a domain of values that it can take on. That is domain is, set of allowable values for random variable.
For example: The domain of cavity can be $\langle \text{true}, \text{false} \rangle$
- ❖ A random variable's proposition will assert that what value is drawn for the random variable from its domain.
For example: $\text{Cavity} = \text{True}$ is proposition.
Saying that "there is cavity in my lower left wisdom tooth".

Artificial Intelligence(AI) with it's Applications

- ❖ Random variables are divided into three kinds, depending on their domain. The types are as follows.
 - Boolean random variables: These are random variables that can take up only boolean values.
For example: Cavity, it takes value either true or false.
 - Discrete random variables: They take values from countable domain. They also include boolean domain. The values in the domain must be mutually exclusive and exhaustive (finite).
For example: Weather, it has domain $\langle \text{sunny, rainy, cloudy, cold} \rangle$
 - Continuous random variables: They take values from real numbers. The domain can be either entire real line or subset of it like intervals $[2, 3]$.
For example : $X = 4.14$ asserts that X has exact value 4.14
Propositions having continuous random variable can have inequalities like $X \leq 4.14$.

Atomic Events

- ❖ An atomic event is a complete specification of the state of the world about which agent is uncertain.
- ❖ They are represented as variables. These variables are assigned values from the real world.

For example: If the world is consists of cavity and Toothache then there are four distinct atomic events,

- a) $\text{Cavity} = \text{False} \wedge \text{Toothache} = \text{True}$
- b) $\text{Cavity} = \text{False} \wedge \text{Toothache} = \text{False}$
- c) $\text{Cavity} = \text{True} \wedge \text{Toothache} = \text{False}$
- d) $\text{Cavity} = \text{True} \wedge \text{Toothache} = \text{True}$
- e) Properties of atomic events

- ❖ They are mutually exclusive - That is at most one can actually be the case.

For example: $(\text{Cavity} \wedge \text{Toothache})$ and $(\text{Cavity} \wedge \neg \text{Toothache})$ can not both be the case.

- ❖ The set of all possible atomic events is exhaustive out of which at least one must be the case. That is, the disjunction of all atomic events is logically equivalent to true.
- ❖ Any particular atomic event entails the truth or falsehood of every proposition, whether simple or complex.

For example -

The atomic event

$(\text{Cavity} \wedge \neg \text{Toothache})$ entails the truth of cavity and the Falsehood of (cavity $\rightarrow$

toothache).

- ❖ Any proposition is logically equivalent to the disjunction of all atomic events that entail the truth of the proposition.

For example: The proposition cavity is equivalent to disjunction of the atomic events (cavity $\wedge$ toothache) and (cavity $\wedge \neg$ toothache).

Prior Probability (Unconditional Probability)

- ❖ The prior (unconditional) probability is associated with a proposition 'a'.
- ❖ It is the degree of belief accorded to a proposition in the absence of any other information.
- ❖ It is written as $P(a)$.

For example: The probability that, Ram has cavity = 0.1, then prior probability is written as, $P(\text{Cavity} = \text{true}) = 0.1$ or $P(\text{Cavity}) = 0.1$

- ❖ 4) It should be noted that as soon as new information is received, one should reason with the conditional probability of 'a' depending upon new information.
- ❖ 5) When it is required to express probabilities of all the possible values of a random variable, then a vector of values is used. It is represented using $\mathbf{P(a)}$. This represents values for the probabilities of each individual state of the 'a'.
For example: $\mathbf{P(\text{Weather})} = \langle 0.7, 0.2, 0.08, 0.02 \rangle$ is representing four equations

$$P(\text{Weather} = \text{Sunny}) = 0.7$$

$$P(\text{Weather} = \text{Rain}) = 0.2$$

$$P(\text{Weather} = \text{Cloudy}) = 0.08$$

$$P(\text{Weather} = \text{Cold}) = 0.02$$

- ❖ The expression $P(a)$ is said to be defining prior probability distribution for the random variable 'a'.
- ❖ To denote probabilities of all random variables combinations, the expression $P(a_1, a_2)$ can be used. This is called as joint probability distribution for random variables a_1, a_2 . Any number of random variables can be mentioned in the expression.
- ❖ A simple example of joint probability distribution is,
 $\rightarrow \mathbf{P} < \text{Weather}, \text{Cavity} >$ can be represented as, 4×2 table of probabilities.
(Weather's probability) (Cavity probability)
- ❖ A joint probability distribution that covers the complete set of random variables is called as full joint probability distribution.

- ❖ A simple example of full joint probability distribution is,

If problem world consists of 3 random variables, whether, cavity, toothache then full joint probability distribution would be,

$P < \text{Weather, Cavity, Toothache} >$

It will be represented as, $4 \times 2 \times 2$, table of probabilities.

- ❖ Prior probability for continuous random variable

i) For continuous random variable it is not feasible to represent vector of all possible values because the values are infinite. For continuous random variable the probability is defined as a function with parameter x , which indicates that random variable takes some value x .

For example: Let random variable x denotes the tomorrow's temperature in Chennai. It would be represented as,

$$P(X = x) = U[25 - 37](x).$$

This sentence express the belief that X is distributed uniformly between 25 and 37 degrees celcius.

ii) The probability distribution for continuous random variable has probability density function.

3.7 Probabilistic Reasoning

Probability based reasoning is the same as inferring directly from knowledge that can be given a probability rating based on the amount of uncertainty present. The uncertainties can arise from an inability to predict outcomes due to unreliable, vague incomplete or inconsistent knowledge. The probability of an uncertain event is a measure of the degree of likelihood of occurrence of that event.

When one takes it for granted that certain parameters exist or known with certainty, solutions can be found out. Belief revision takes place whenever one encounters a piece of information that decreases the belief of the fact.

The source for all uncertainties is the real world. The following sources constitute the major causes of uncertainties,

- ❖ Most information is not obtained personally but got from sources on which one has a lot of belief. Uncertainty prevails when the source does not send information in time or the information provided is not understood.
- ❖ In laboratory, experiments one takes it for granted that the equipment is properly

calibrated and error free. If the equipment is faulty, the picture one gets about the situation is not correct which leads to uncertainty.

- ❖ Experimental errors like parallax errors are also sources of uncertainty.
- ❖ Imprecision in natural language is also a source of uncertainty.
- ❖ A random event occurring is a major source of uncertainty.
- ❖ In judicial courts one may have seen or heard delivering judgements acquitting the accused for,
 - Lack of evidence
 - Lack of certainty in evidence thereby giving the benefit of doubt to the accused.

To handle these uncertain data, probability is the oldest technique available. Probabilistic reasoning is sometimes used when outcomes are unpredictable. For example, when a physician examines a patient, the patient's history, symptoms and test results provide some, but not conclusive evidence of possible ailments. This knowledge, together with the physician's experience with previous patients, improves the likelihood of predicting the unknown event, but there is still much uncertainty in most diagnoses.

One way to express confidence about such event is through probability. It is a mere number that expresses the chance of something happening or not happening.

Representing knowledge in uncertain domain

Independence and conditional independence relationships among variables can greatly reduce the number of probabilities that need to be specified in order to define the full joint distribution. There is technique called as Bayesian Network which can be used to represent the dependencies among variables and to give a concise specification of any full joint probability distribution.

Bayesian Network

Definition

It is a data structure which is a graph, in which each node is annotated with quantitative probability information.

The nodes and edges in the graph are specified as follows,

- ❖ A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.

Artificial Intelligence(AI) with it's Applications

- ❖ A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y , then X is said to be a parent of Y .
- ❖ Each node X_i , has a conditional probability distribution $P(X_i \mid \text{Parents}(X_i))$ that quantifies the effect of the parents on the node.
- ❖ The graph has no directed cycles (and hence is a directed, acyclic graph, or DAG).

The set of nodes and links is called as topology of the network. The topology of the network specifies the conditional independence relationships that hold in the domain.

The intuitive meaning of an arrow between two nodes X and Y , in a properly constructed network, is that " X has direct influence on Y ". The domain expert is a person who can identify such influence associations.

Once Bayesian network topology is specified then conditional probability distribution for each variable is specified. For specifying continuous probability distribution for a variable, its parent information is required.

The combination of topology and the conditional distributions is sufficient to specify the full joint distribution for all the variables.

Consider our example of simple world consisting of variables toothache, cavity, catch and weather. Weather is surly independent of other variables. Toothache and Catch are conditionally independent if given the information of cavity.

This relationships are represented by Bayesian network as shown below,

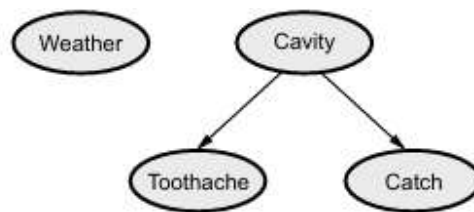


Fig. 3.25 A simple Bayesian network in which Weather is independent of other three variables and Toothache and Catch are conditionally independent, given the cavity.

Note

- ❖ The conditional independence of toothache and catch given the cavity is indicated by the absence of a link between toothache and catch.
- ❖ The network represents the fact that cavity is a direct cause of toothache and catch.
- ❖ No direct causal relationship exists between toothache and catch.

Consider another example of burglar alarm installed at A's home. This alarm notifies in case of burglary and can occasionally respond to minor earthquakes. There are two neighbours to A, M and J. M and J promised to call A in office when they hear alarm. J always calls A when

he hears alarm, but many times he is confused between telephone ring and burglar alarm. Many a times misses alarm because he keeps on listening to loud music. Given the evidence of who has or has not called, we are to estimate the probability of burglary.

The Bayesian network for above problem along with the associated probabilities is shown below,

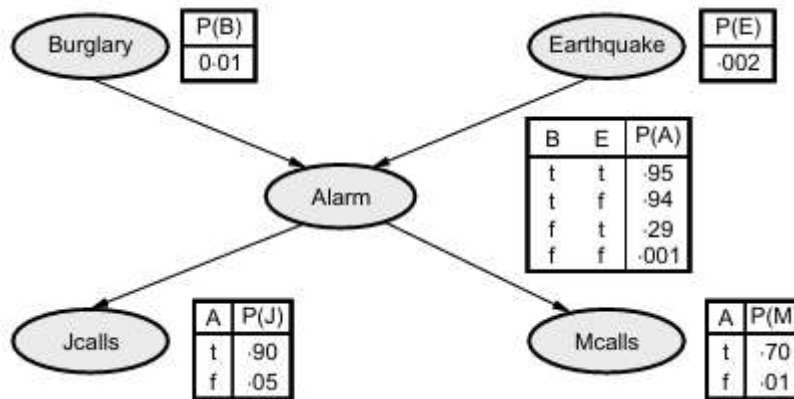


Fig. 3.26 A Typical Bayesian network, showing both the topology and the conditional probability tables (CPTs).

Note: In the figure, in the CPTs, the letters B, E, A, J and M stand for Burglary, Earthquake, Alarm, Johncalls, Marycalls respectively.

- ❖ In the figure, each distribution is shown as conditional probability table. Each row in the table contains the conditional probability of each node value for a conditioning case. A conditioning case is just a possible combination of values for the parent nodes. Each row must sum upto 1 because the entries represent an exhaustive set of cases for the variable.
- ❖ For boolean variable if it is true and its probability is known to be 'P' then when it is false its probability is $1 - P$ and it is omitted from the table (because it can be deduced from 'P').
- ❖ A table for a boolean variable with k boolean parents contains 2^k independently specifiable probabilities. A node with no parents has only one row, representing the prior probabilities of each possible value of the variable.

The Semantics of Bayesian Networks

There are two ways through which semantic (meaning) of Bayesian network can be understood.

One way is to view network as a representation of the joint probability distribution. This view helps to construct networks. Second way is to view a network as an encoding of a collection

of conditional independence statements. This view helps in designing inference procedure semantically both views are equivalent.

Understanding semantic of Bayesian network method,

Representing the full joint distribution

- ❖ Every entry in the full joint probability distribution (hereafter abbreviated as "joint") can be calculated from the information in the network.
- ❖ A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1 \wedge \dots \wedge X_n = x_n)$
- ❖ The notation $P(x_1, \dots, x_n)$ is used as an abbreviation for this.
- ❖ The value of this entry is given by the formula.

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \text{parents}(X_i)) \quad (1)$$

Where, $\text{parents}(x_i)$ denotes the specific values of the variables in parents (x_i) .

- ❖ Thus, each entry in the joint distribution is represented by the product of the appropriate elements of the Conditional Probability Tables (CPTs) in the Bayesian network. The CPTs therefore, provide a decomposed representation of the joint distribution.
We can calculate the probability that the alarm has sounded, but neither a burglary nor an earthquake has occurred and both J and M call. We use single letter names for the variables:

$$\begin{aligned} P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) &= P(j \mid a)P(m \mid a)P(a \mid \neg b \wedge \neg e)P(\neg b)P(\neg e) \\ &= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062 \end{aligned}$$

- ❖ Remember that the full joint distribution can be used to answer any query about the domain.
If a Bayesian network is a representation of the joint distribution, then it too can be used to answer any query, by summing all the relevant joint entries.

A method for constructing Bayesian network

- ❖ We rewrite the joint distribution in terms of a conditional probability, using the product rule.

$$P(x_1, \dots, x_n) = P(x_n \mid x_{n-1}, \dots, x_1)P(x_{n-1}, \dots, x_1).$$

- ❖ Then the process is represented, reducing each conjunctive probability to a conditional probability and a smaller conjunction. We end up with one big product.
 - ❖
$$\begin{aligned} P(x_1, \dots, x_n) &= P(x_n \mid x_{n-1}, \dots, x_1)P(x_{n-1} \mid x_{n-2}, \dots, x_1) \dots P(x_2 \mid x_1)P(x_1) \\ &= \prod_{i=1}^n P(x_i \mid x_{i-1}, \dots, x_1) \end{aligned}$$
-

- ❖ Above identity holds true for any set of random variables and is called the Chain rule. Comparing it with equation (1). We see that the specification of the joint distribution is equivalent to the general assertion that, for every variable X_i in the network,

$$\mathbf{P}(X_i | X_{i-1}, \dots, X_1) = \mathbf{P}(X_i | \text{Parents}(X_i))$$

provided that $\text{parents}(X_i) \subseteq \{X_{i-1}, \dots, X_1\}$. This last condition is satisfied by labeling the nodes in any order that is consistent with the partial order implicit in the graph structure.

4) Bayesian network is correct representation of the domain only if each node is conditionally independent of its predecessors in the node ordering, given its parents.

5) In order to construct a Bayesian network with the correct structure for the domain, we need to choose parents for each node such that this property holds. Intuitively, the parents of node X_i should contain all those nodes in $X_1, \dots, X_{i-1}$ that directly influence X_i .

For example : Suppose we have completed the network in **Fig. 3.26** except for the choice of parents for M calls, M calls is certainly influenced by whether there is a burglary or an earthquake, but not directly influenced. Intuitively, our knowledge of the domain tells us that these events influence M's calling behaviour only through their effect on the alarm. Also given the state of the alarm, whether J calls has no influence on M's calling. Formally speaking, we believe that the following conditional independence statement holds,

$$\mathbf{P}(\text{M Calls} | \text{J Calls, Alarm, Earthquake, Burglary}) = \mathbf{P}(\text{M calls} | \text{Alarm}).$$

Compactness and node ordering

Bayesian network are compact and they possess a property of being locally structured (also called as sparse systems). In a locally structured system, each sub component interacts directly with only a bounded number of other components, regardless of the total number of components.

- ❖ Local structure is usually associated with linear rather than exponential growth in complexity.
- ❖ We assume ' n ' boolean variables for simplicity, then the amount of information needed to specify each conditional probability table will be at most 2^k numbers. Where each random variable is influenced by ' k ' other variables.
- ❖ The complete network can be specified by $n2^k$ numbers.
- ❖ With some values of ' n ' the joint distribution contains 2^n numbers.
- ❖ To make this concrete, suppose we have $n = 30$ nodes, each with five parents ($k = 5$). Then the Bayesian network requires 960 numbers, but the full joint distribution requires over a billion.

Ordering of nodes in Bayesian network

- ❖ The correct order in which to add nodes is to add the "root causes" first, then the variables they influence, and so on, until we reach the "leaves", which have no direct causal influence on the other variables.
- ❖ If wrong order is chosen we get more complicated network.

For example: Consider network shown in following diagram.

The network construction process goes as follows,

- ❖ Adding M calls: No parent.
- ❖ Adding J calls: If M calls, that probably means the alarm has gone off, which of course would make it more likely that J calls. Therefore J calls needs M calls as a parent.
- ❖ Adding alarm: Clearly, if both call, it is more likely that the alarm has gone off than if just one or neither call, so we need both M calls and J calls as parent.
- ❖ Adding burglary: If we know the alarm state, then the call from J or M might give us information about phone ringing or M's music, but not about burglary : $P(\text{Burglary} | \text{Alarm}, \text{J calls}, \text{M calls}) = P(\text{Burglary} | \text{Alarm})$.

Hence we need just alarm as parent.

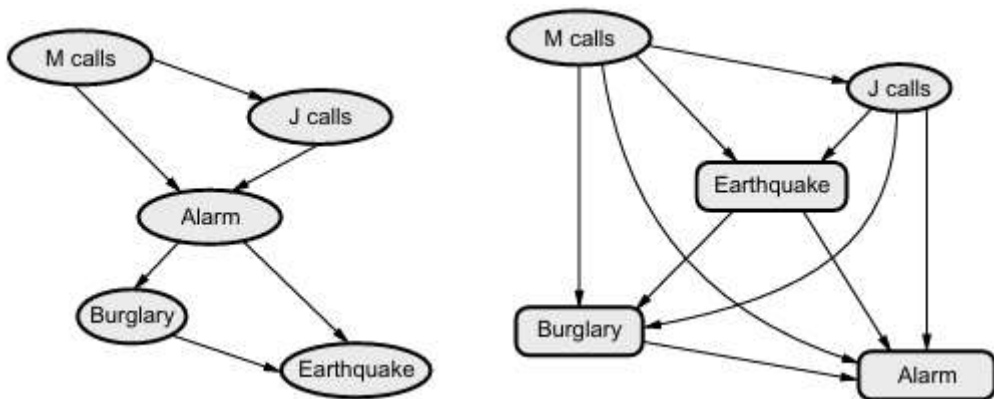


Fig. 3.27 and Fig. 3.28 Network structure depends on order of introduction. In each network nodes are added from top to bottom.

- ❖ Adding earthquake: If the alarm is on, it is more likely that there has been an earthquake. (The alarm is an earthquake detector of sorts). But if we know that there has been a burglary, then that explains the alarm, and the probability of an earthquake would be only slightly above normal. Hence, we need both Alarm and Burglary as parents. The network is shown below.

Consider example of a very bad node ordering as shown in the **Fig. 3.28**

M calls, J calls, Earthquake, Burglary, Alarm are nodes. This network requires 31 distinct probabilities to be specified exactly the same as the full joint distribution. It is important to realize, however, that any of the three networks can represent exactly the same joint distribution. The last two versions simply fail to represent all conditional independence relationships and hence end up specifying a lot of unnecessary numbers instead.

Conditional Independence Relations in Bayesian Networks

One can start from a "topological" semantics that specifies the conditional independence relationships encoded by the graph structure, and from these we can derive the "numerical" semantics. The topological semantics is given by either of the following specifications, which are equivalent.

1. A node is conditionally independent of its non-descendants, given its parents. For example, in **Fig. 3.26** J calls is independent of Burglary and Earthquake, given the value of Alarm.
2. A node is conditionally independent of all other nodes in the network, given its parents, children, and children's parents-that is, given its Markov blanket. For example: Burglary is independent of J calls and M calls given Alarm and Earthquake.

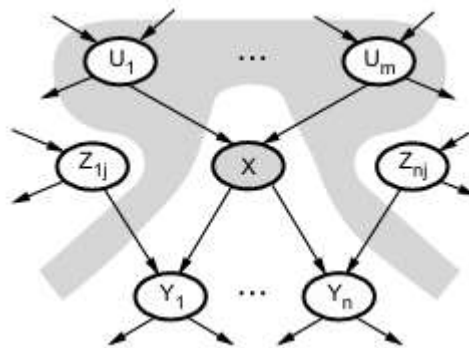


Fig. 3.29 (a) A node x is conditionally independent of its non-descendants. (e.g., the Z_{ij} s) given its parents (the U_i s shown in the gray area).

This specifications are illustrated in following figures. From these conditional independence assertions and the CPTs, the full joint distribution can be reconstructed; thus the "numerical" semantics and the "topological" semantics are equivalent.

A node X is conditionally independent of its non-descendants (example - The Z_{1j} s) given its parents (the U_i is shown in the gray area).

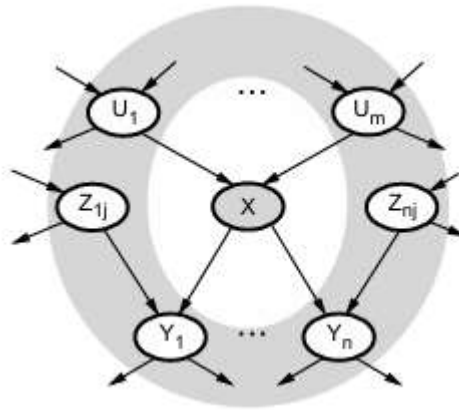


Fig. 3.29 (b) A node x is conditionally independent of all other nodes in the network given its markov blanket (the gray area).

A node X is conditionally independent of all other nodes in the network given its Markove blanket (the gray area).

Efficient representation of conditional representation

If the maximum number of parents k is smallish, filling in the CPT for a node, would require up to $O(2^k)$ numbers.

To avoid this big relationship number canonical distribution is used. In this a complete table is specified by naming the patterns supplied with parameters, which can describe relationship which are necessary. A deterministic node represents canonical distribution. A deterministic node is a node whose value is exactly specified by the value of its parents with no uncertainty.

The uncertain relationships are often represented by noisy-OR relationships. [Noisy-OR is generalization of logical OR]. The noisy-OR model allows uncertainty about the parent to cause the child to be true i.e. the causal relationship between parent and child would be inhibited.

For example: A patient could have cold (parent) but not exhibit a fever (child).

For noisy-OR model it is required that all the possible effects of parent must be listed. It should be clear that inhibition of one parent is independent of other parent.

The Hybrid Bayesian Network

A network with both discrete and continuous variables is called as hybrid Bayesian network. For representing continuous variable its discretization is done in terms of intervals (because it can have infinite values).

To specify a hybrid network, we have to specify two new kinds of distributions. The conditional distribution for a continuous variable given discrete or continuous parents; and the conditional distribution for a discrete variable given continuous parents.

Consider the simple example in following diagram, in which a customer buys some fruit depending on its cost, which depends in turn on the size of the harvest and whether the government's subsidy scheme is operating. The variable Cost is continuous and has continuous and discrete parents; the variable Buys is discrete and has a continuous parent.

For the cost variable, we need to specify $P(\text{Cost} \mid \text{Harvest}, \text{Subsidy})$. The discrete parent is handled by explicit enumeration that is, specifying both $P(\text{Cost} \mid \text{Harvest}, \text{Subsidy})$ and $P(\text{Cost} \mid \text{Harvest}, \neg \text{Subsidy})$. To handle Harvest, we specify how the distribution over the cost 'c' depends on the continuous value 'h', of Harvest. In other words, we specify the parameter of the cost distribution as a function of 'h'.

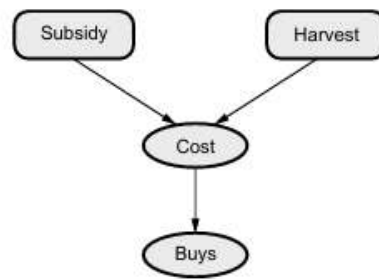


Fig. 3.30 A simple network with discrete variables (Subsidy and Buys) and continuous variables (Harvest and Cost).

Inferencing in Bayesian Networks

Exact Inference in Bayesian Networks

For inferencing in probabilistic system, it is required to calculate posterior probability distribution for a set of query variables, where some observed events are given. [That is we have some values attached to evidence variables].

Notation Revisited

The notation used in inferencing is same as the one used in probability theory.

X : Query variable.

E : The set of evidence variables $E_1, \dots, E_m$ and 'e' is the particular observed event.

Y : The set of non-evidence variables $Y_1, Y_2, \dots, Y_k$ [Non-evidence variables are also called as hidden variables].

X : It the complete set of all the types of variables, where $X = \{X\} \cup E \cup Y$.

Artificial Intelligence(AI) with it's Applications

Generally the query requires the posterior probability distribution $P(X | e)$ [assuming that query variable is not among the evidence variables, if it is, then posterior distribution for X simply gives probability 1 to the observed value].

[Note that query can contain more than one variable. For study purpose we are assuming single variable].

Example: In the burglary case, if the observed event is $J_{calls} = \text{true}$ and $M_{calls} = \text{true}$.
The query is 'Has burglary occurred?'

The probability distribution for this situation would be,

$$P(\text{Burglary} \mid J_{calls} = \text{true}, M_{calls} = \text{true}) = \langle 0.284, 0.716 \rangle$$

Inference by Enumeration

A Bayesian network gives a complete representation of the full joint distribution. These full joint distributions can be written as product of conditional probabilities from the Bayesian network.

A query can be answered using Bayesian network by computing sums of products of conditional probabilities from the network.

The algorithm

The algorithm ENUMERATE-JOINT-ASK gives inference by enumerating on full joint distribution.

Characteristics of algorithm

- ❖ It takes input a full joint distribution P and looks up values in it. [The same algorithm can be modified to take input as Bayesian network and looking up in joint entries by multiplying the corresponding conditional probability table entries from Bayesian network.
- ❖ The ENUMERATION-JOINT-ASK uses ENUMERATION-ASK (EA) algorithm which evaluate expression using depth-first recursions. Therefore, the space complexity of EA is only linear in the number of variables. The algorithm sums over the full joint distribution without ever constructing it explicitly. The time complexity for network with ' n ' boolean variables is always $O(2^n)$ which is better than the $O(n2^n)$ required in simple inferencing approach (using posterior probability).
- ❖ The drawback of the algorithm is, it keeps on evaluating repeated sub expression which results in wastage of computation time.

The enumeration algorithm for answering queries on Bayesian network.

The algorithm

Function ENUMERATION-ASK (X, e, bn) returns a distribution over X .

Inputs: X , the query variable

e , observed values for variables E .

bn , a Bayes net with variables $\{X\} \cup E \cup Y$ / $*y = \text{Hidden variable} *$

$Q(x) \leftarrow$ A distribution over X , initially empty for each value x_i of X do extend e with value x_i for X .

$Q(x_i) \leftarrow \text{ENUMERATE-ALL}(\text{VARS}[bn] \ e) \text{ return } \text{NORMALIZE}(Q(x)).$

Function $\text{ENUMERATE-ALL}(\text{vars}, e)$ returns a real number.

if $\text{EMPTY} ? (\text{vars})$ then return 1.0

$Y \leftarrow \text{FIRST}(\text{vars})$

If Y has value y in e .

Then return $P(y \mid \text{parents}(Y)) \times \text{ENUMERATE-ALL}(\text{REST}(\text{vars}), e)$

else return $\sum_y P(y \mid \text{parents}(Y)) \times \text{ENUMERATE-ALL}(\text{REST}(\text{vars}), e_y)$

where, e_y is e extended with $Y = y$.

Example

Consider query,

$P(\text{Burglary} \mid J \text{ calls} = \text{true}, M \text{ calls} = \text{true})$

Hidden variables in the queries are $\rightarrow$ Earthquake and Alarm.

using the query equation.

$P(\text{Burglary} \mid j, m) = \alpha P(\text{Burglary}, J, M) = \alpha \sum_e \sum_a P(\text{Burglary}, e, a, j, m)$

The semantics of Bayesian networks (equation 8.2.1) then gives us an expression in terms of CPT entries. For simplicity, we will do this just for $\text{Burglary} = \text{true}$.

$$P(b \mid j, m) = \alpha \sum_e \sum_a P(b)P(e)P(a \mid b, e)P(j \mid a)P(m \mid a)$$

- ❖ To compute this expression, we have to add four terms, each computed by multiplying five numbers.
- ❖ Worst case, where we have to sum out almost all the variables, the complexity of the algorithm for a network with n boolean variables is $O(n2^n)$.
- ❖ An improvement can be obtained from the following simple observations. The $P(b)$ term is a constant and can be moved outside the summations over a and e , and the $P(e)$ term can be moved outside the summation over a . Hence, we have

$$P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e)P(j \mid a)P(m, a) \quad (2)$$

This expression can be evaluated by looping through the variables in order, multiplying CPT entries as we go. For each summation, we also need to loop over the variable's possible values. The structure of this computation is shown in following diagram. Using the numbers from **Fig. 3.26**, we obtain $P(b \mid j, m) = \alpha \times 0.00059224$. The corresponding computation for $\neg b$ yields $\alpha \times 0.0014919$;

Hence,

$$P(B \mid j, m) = \alpha < 0.00059224, 0.0014919 > \\ \approx < 0.284, 0.716 >$$

That is, the chance of burglary, given calls from both neighbours is about 28%.

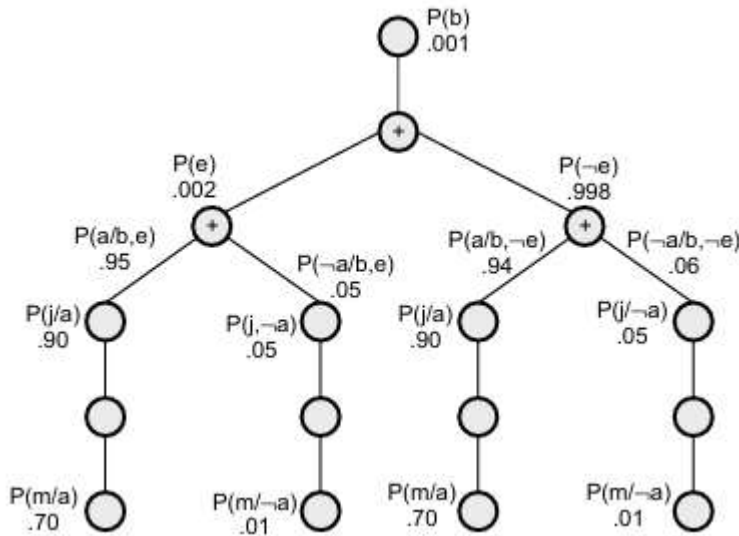


Fig. 3.31 The structure of the expression shown in equation (2).

Note: In the Fig. 3.31, the evaluation proceeds top to down, multiplying values along each path and summing at the "+" nodes. Observe that there is repetition of paths for j and m .

The Variable Elimination Algorithm

The enumeration algorithm can be improved substantially by eliminating calculations of repeated sub expression in tree. Calculation can be done once and save the results for later use. This is a form of dynamic programming.

Working of variable elimination algorithm

1. It works by evaluating expressions such as $[P(b \mid j, m) = \alpha P(b) \sum_e P(e) \sum_a P(a \mid b, e) P(j \mid a) P(m \mid a)]$ in right-to-left order. $\sum_a P(a \mid b, e) P(j \mid a) P(m \mid a)]$ in right-to-left order.

2. Intermediate results are stored and summations over each variable are done only for those portions of the expression that depends on the variable.

For example: Consider the Burglary network.

We evaluate the expression :

$$P(B | j, m) = \underbrace{\alpha P(B)}_B \sum_e P_E^{P(e)} \sum \underbrace{P(a | B, e)}_A \underbrace{P(j | a)}_E \underbrace{P(m | a)}_M$$

3. **Factors:** Each part of the expression is annotated with the name of the associated variable, these parts are called factors.

Steps in algorithm

- i) The factor for M, $P(m | a)$, does not require summing over M . Probability is stored, given each value of a, in a two-element factor,

$$f_M(A) = \begin{bmatrix} P(m | a) \\ P(m | \neg a) \end{bmatrix}$$

Note: f_M means that M was used to produce f .

- ii) Store the factor for J as the two-element vector $f_J(A)$.

- ii) The factor for A is $P(a | B, e)$ which will be a $2 \times 2 \times 2$ matrix $f_A(A, B, E)$.

- iv) Summing out A from the product of these tree factors. This will give 2×2 matrix whose indices range over just B and E . We put bar over A in the name of the matrix to indicate that A has been summed out.

$$\begin{aligned} f_{\bar{A}M}(B, E) &= \sum_a f_A(a, B, E) \times f_J(a) \times f_M(a) \\ &= f_A(a, B, E) \times f_J(a) \times f_M(a) + f_A(\neg a, B, E) \times f_J(\neg a) \times f_M(\neg a) \end{aligned}$$

The multiplication process used is called a pointwise product.

- v) Processing E in the same way (i.e.) sum out E from the product of $f_E(E)$ and $f_{\bar{A}JM}(B, E)$:

$$f_{\bar{A}JM}(B) = f_E(e) \times f_{\bar{A}JM}(B, e) + f_E(\neg e) \times f_{\bar{A}JM}(B, \neg e).$$

- vi) Compute the answer simply by multiplying the factor for B. (i.e.) $(f_B | B) = P(B)$, by the accumulated matrix (B) :

$$P(B | j, m) = \alpha f_B(B) \times f_{\bar{E}A}JM(B).$$

From the above sequence of steps it can be noticed that two computational operations are required.

- a) Pointwise product of a pair of factors.
- b) Summing out a variable from a product of factors.

a) Pointwise product of a pair of factors: The pointwise product of two factors f_1 and f_2 yields a new factor f , those variables are the union of the variables in f_1 and f_2 . Suppose the two factors have variables $Y_1, \dots, Y_k$. Then we have $f(X_1, \dots, X_j, Y_1, \dots, Y_k, Z_1, \dots, Z_l) = f_1(X_1, \dots, X_j, Y_1 \dots Y_k) f_2(Y_1, \dots, Y_k, Z_1, \dots, Z_l)$. If all the variables are binary, then f_1 and f_2 have 2^{j+k} and 2^{k+l} entries and the pointwise product has 2^{j+k+l} entries.

For example : Given two factors $f_1(A, B)$ and $f_2(B, C)$ with probability distributions shown below, the pointwise product $f_1 \times f_2$ is given as $f_1(A, B, C)$.

A	B	$f_1(A, B)$	B	C	$f_2(B, C)$	A	B	C	$f_3(A, B, C)$
T	T	.3	T	T	.2	T	T	T	$.3 \times .2$
T	F	.7	T	F	.8	T	T	F	$.3 \times .8$
F	T	.9	F	T	.6	T	F	T	$.7 \times .6$
F	F	.1	F	F	.4	T	F	F	$.7 \times .4$
						F	T	T	$.9 \times .2$
						F	T	F	$.9 \times .8$
						F	F	T	$.1 \times .6$
						F	F	F	$.1 \times .4$

b) Summing out a variable from a product of factors: It is a straight forward computation. Any factor that does not depend on the variable to be summed out can be moved outside the summation process.

For example:

$$\sum_e f_E(e) \times f_A(A, B, e) \times f_J(A) \times f_M(A) = f_J(A) \times f_M(A) \times \sum_e f_E(e) \times f_A(A, B, e).$$

Now, the pointwise product inside the summation is computed and the variable is summed out of the resulting matrix.

$$f_J(A) \times f_M(A) \times \sum_e f_E(e) \times f_A(A, B, e) = f_J(A) \times f_M(A) \times f_{\overline{EA}}(A, B).$$

Matrices are not multiplied until we need to sum but a variable from the accumulated product. At that point, multiply those matrices that include the variable to be summed out.

The procedure for pointwise product and summing is as follows, the variable elimination algorithm is as follows:

Function ELIMINATION-ASK (X, e, bn) returns a distribution over X

Inputs: X, the query variable

```
e, evidence specified as an event
bn, a Bayesian network specifying joint distribution  $P(X_1, \dots, X_n)$ .
Factors  $\leftarrow [ ] : \text{vars} \leftarrow \text{REVERSE}(\text{VARS}[\text{bn}])$ 
for each var in vars do
  Factors  $\leftarrow [\text{MAKE-FACTOR}(\text{var}, e) \text{ factors}]$ 
if var is a hidden variable then
  Factors  $\leftarrow \text{SUM-OUT}(\text{var}, \text{factors})$ 
returns  $\text{NORMALIZE}(\text{POINTWISE-PRODUCT}(\text{factors}))$ .
```

$P(J \text{ calls}, \text{Burglary} = \text{True})$
The first step is to write out the nested summation.

$$P(J | b) = \alpha P(b) \sum_e P(e) \sum_a P(a | b, e) P(J | a) \sum_m P(M | a).$$

Evaluating this expression from right to left, $\sum_m P(M | a)$ is equal to 1.

Note: The variable M is irrelevant to this query. Result of the query $P(J \text{ calls}/\text{Burglary} = \text{True})$ is unchanged if we remove M calls from the network. We can remove any leaf node which is not a query variable or an evidence variable. After its removal, there may be more leaf nodes and they may be irrelevant. Eventually we find that every variable that is not an ancestor of a query variable or evidence variable is, irrelevant to the query. A variable elimination algorithm can remove all these variables before evaluating the query.

The Complexity Involved in Exact Inferencing

The variable elimination algorithm is more efficient than enumeration algorithm because it avoids repeated computations as well as drops irrelevant variables.

The variable elimination algorithm constructs the factor, deriving its operation. The space and time complexity of variable elimination is directly dependant on size of the largest factor constructed during the operation. Basically the factor construction is determined by the order of elimination of variables and by the structure of the network; which affects both space and time complexity.

For developing more efficient process we can construct singly connected networks which are also called as polytrees. In singly connected network, there is at most one undirected path between any two nodes in the networks. The singly connected networks have property that, the time and space complexity of exact inference in polytrees is linear in the size of the network. Here the size is defined as the number of CPT entries. If the number of parents of each node is bounded by a constant, then the complexity will also be linear in the number of nodes.

For example: The Burglary network shown in the Fig. 3.26 is a polytrees.
[Note that every problem may not be represented as polytrees].

In multiply networks [In this, their can be multiple undirected paths between any two nodes and more than one directed path between some pair of nodes], variable elimination takes exponential time and space complexity in the worst case, even when the number of parents per node is bounded. It should be noted that variable elimination includes inference in propositional logic as a special case and inference in Bayesian network is NP-hard. In fact it is strictly harder than NP-complete problem.

Clustering algorithm

1. Clustering algorithm (known as joint tree algorithms) in which inferencing time can be reduced to $O(n)$. In clustering individual nodes of the network are joint to form cluster nodes to such a way that the resulting network is a polytree.
2. The variable elimination algorithm is efficient algorithm for answering individual queries. Posterior probabilities are computed for all the variables in the network. It can be less efficient, in polytree network because it needs to issue $O(n)$ queries costing $O(n)$ each, for a total of $O(n^2)$ time, clustering algorithm, improves over it.

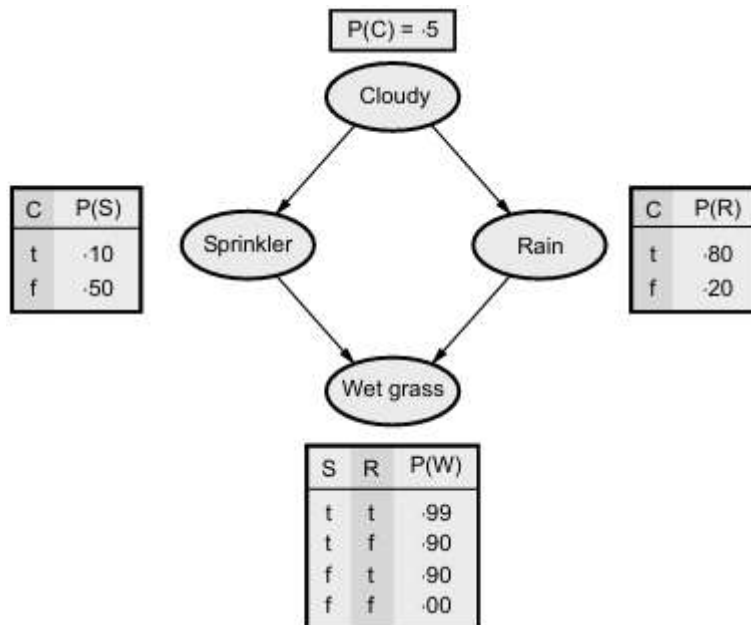


Fig. 3.32 (a) A multiply connected network with conditional probability tables.

For example: The multiply connected network shown in **Fig. 3.32 (a)** can be converted into a polytree by combining the Sprinkler and Rain node into a cluster node called Sprinkler + Rain, as shown in **Fig. 3.32 (b)**. The two Boolean nodes are replaced by a mega node that takes on four possible values: TT, TF, FT, FF. The mega node has only one parent, the Boolean variable, Cloudy, so there are two conditioning cases.

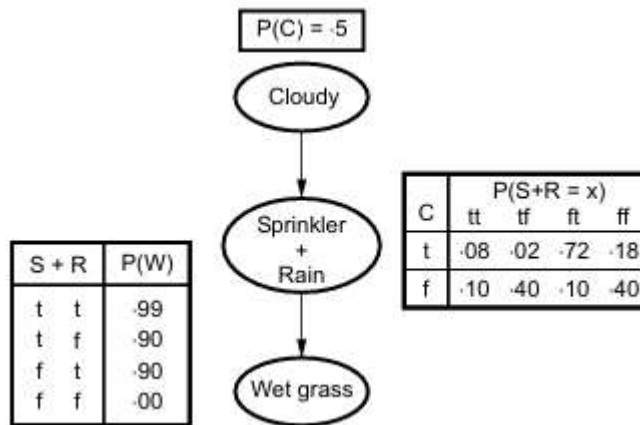


Fig. 3.32 (b) A clustered equivalent of the multiply connected network.

Peculiarities of algorithm

- ❖ Once the network is in polytree form, a special-purpose inference algorithm is applied. Essentially, the algorithm is a form of constraint propagation where the constraints ensure that neighbouring clusters agree on the posterior probability of any variables that they have in common.
- ❖ With careful book keeping, this algorithm is able to compute posterior probabilities for all the non-evidence nodes in the network in time $O(n)$, where n is now the size of the modified network.
- ❖ However, the NP-hardness of the problem continues: if a network requires exponential time and space with variable elimination, then the CPTs in the clustered network will require exponential time and space to construct.

Planning and Machine Learning

4.1 Basic Plan Generation Systems

A plan is considered a sequence of actions, and each action has its preconditions that must be satisfied before it can act and some effects that can be positive or negative.

So, we have **Forward State Space Planning (FSSP)** and **Backward State Space Planning (BSSP)** at the basic level.

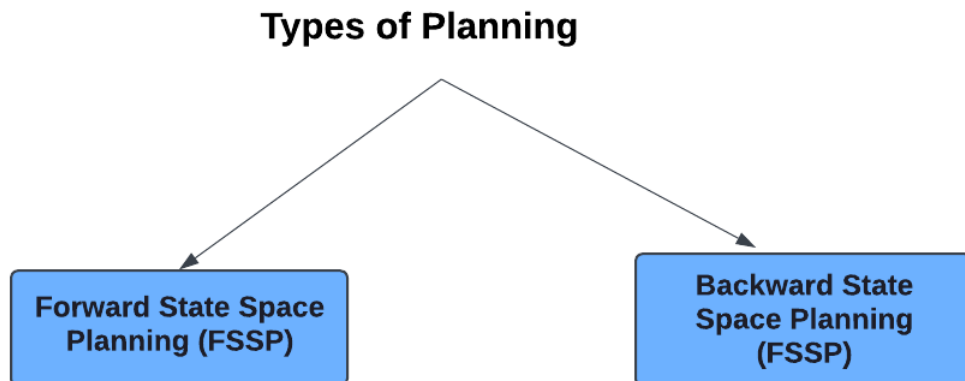


Fig. 4.1 Types of Planning.

1. Forward State Space Planning (FSSP)

FSSP behaves in the same way as forwarding state-space search. It says that given an initial state S in any domain, we perform some necessary actions and obtain a new state S' (which also contains some new terms), called a progression. It continues until we reach the target position. Action should be taken in this matter.

- ❖ **Disadvantage:** Large branching factor
- ❖ **Advantage:** The algorithm is Sound

2. Backward State Space Planning (BSSP)

BSSP behaves similarly to backward state-space search. In this, we move from the target state g to the sub-goal g , tracing the previous action to achieve that goal. This process is called regression (going back to the previous goal or sub-goal). These sub-goals should also be checked for consistency. The action should be relevant in this case.

- ❖ **Disadvantages:** not sound algorithm (sometimes inconsistency can be found)
- ❖ **Advantage:** Small branching factor (much smaller than FSSP)

Planning in artificial intelligence is about decision-making actions performed by robots or computer programs to achieve a specific goal.

Execution of the plan is about choosing a sequence of tasks with a high probability of accomplishing a specific task.

Block-world planning problem

- ❖ The block-world problem is known as the Sussmann anomaly.
- ❖ The non-interlaced planners of the early 1970s were unable to solve this problem. Therefore it is considered odd.
- ❖ When two sub-goals, $G1$ and $G2$, are given, a non-interleaved planner either produces a plan for $G1$ that is combined with a plan for $G2$ or vice versa.
- ❖ In the block-world problem, three blocks labeled 'A', 'B', and 'C' are allowed to rest on a flat surface. The given condition is that only one block can be moved at a time to achieve the target.

The start position and target position are shown in the following diagram.

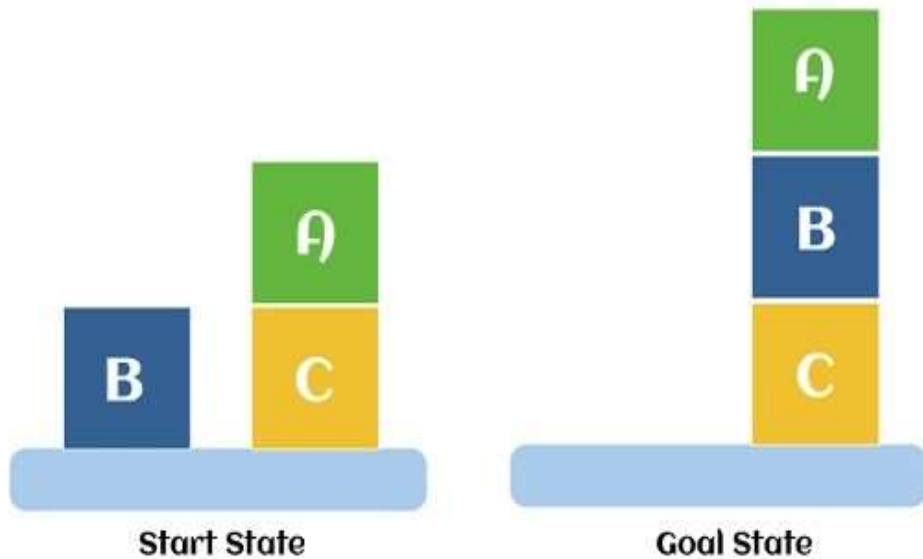


Fig. 4.2 Word Planning Problem.

Components of the planning system

The plan includes the following important steps:

- ❖ Choose the best rule to apply the next rule based on the best available guess.
- ❖ Apply the chosen rule to calculate the new problem condition.
- ❖ Find out when a solution has been found.
- ❖ Detect dead ends so they can be discarded and direct system effort in more useful directions.
- ❖ Find out when a near-perfect solution is found.

Target stack plan

- ❖ It is one of the most important planning algorithms used by STRIPS.
- ❖ Stacks are used in algorithms to capture the action and complete the target. A knowledge base is used to hold the current situation and actions.
- ❖ A target stack is similar to a node in a search tree, where branches are created with a choice of action.

The important steps of the algorithm are mentioned below:

1. Start by pushing the original target onto the stack. Repeat this until the pile is empty. If the stack top is a mixed target, push its unsatisfied sub-targets onto the stack.

2. If the stack top is a single unsatisfied target, replace it with action and push the action precondition to the stack to satisfy the condition.

iii. If the stack top is an action, pop it off the stack, execute it and replace the knowledge base with the action's effect.

If the stack top is a satisfactory target, pop it off the stack.

Non-linear Planning

This Planning is used to set a goal stack and is included in the search space of all possible sub-goal orderings. It handles the goal interactions by the interleaving method.

Advantages of non-Linear Planning

Non-linear Planning may be an optimal solution concerning planning length (depending on the search strategy used).

Disadvantages of Nonlinear Planning

It takes a larger search space since all possible goal orderings are considered.

Complex algorithm to understand.

Algorithm

1. Choose a goal 'g' from the goal set
2. If 'g' does not match the state, then
 - ❖ Choose an operator 'o' whose add-list matches goal g
 - ❖ Push 'o' on the OpStack
 - ❖ Add the preconditions of 'o' to the goal set.
3. While all preconditions of the operator on top of OpenStack are met in a state
 - ❖ Pop operator o from top of opstack
 - ❖ state = apply(o, state)
 - ❖ plan = [plan; o]

Importance of Planning in AI

Planning is essential in AI for several reasons:

- ❖ **Efficiency and Optimization:** Planning allows AI systems to choose the most efficient route to achieve a goal, minimizing resource usage and time. For instance, an AI logistics system can optimize delivery routes, reducing fuel consumption and costs.
-

Artificial Intelligence(AI) with it's Applications

- ❖ **Adaptability:** In real-world applications, AI systems often face uncertain environments. By incorporating planning, AI can adjust its course of action in response to new information or unforeseen obstacles. For instance, a self-driving car can re-route when encountering roadblocks.
- ❖ **Autonomy:** Planning is a key factor that enables AI systems to act autonomously. Whether it's a robotic arm in a manufacturing plant or a personal assistant like Siri or Alexa, the ability to plan gives AI the capability to perform tasks without constant human intervention.
- ❖ **Decision-Making:** AI systems that utilize planning can make informed decisions by considering different possible outcomes. This makes planning critical in applications like medical diagnosis systems, financial forecasting, or game AI.

Challenges in AI Planning

Despite its importance, AI planning presents several challenges:

- ❖ **Computational Complexity:** Planning, especially in complex environments, can be computationally expensive. Finding the optimal sequence of actions in large, dynamic systems can take a significant amount of processing power and time.
- ❖ **Handling Uncertainty:** In uncertain or unpredictable environments, creating a plan that can handle every possible outcome is challenging. Probabilistic and reactive planning methods aim to address this, but it remains a difficult problem.
- ❖ **Scalability:** As the size of the problem or task increases, so does the difficulty of planning. Scaling up planning algorithms to handle large datasets or environments with numerous variables is a technical hurdle.

Applications of AI Planning

The role of planning in AI is visible across various industries and applications, such as:

- ❖ **Robotics:** Planning allows robots to move efficiently in environments, avoid obstacles, and perform tasks autonomously. For example, a warehouse robot can plan its path to pick up items without collisions.
 - ❖ **Healthcare:** AI planning systems are used in treatment planning, where algorithms suggest optimal therapies for patients based on various factors like medical history, current health, and probability of success.
 - ❖ **Autonomous Vehicles:** Self-driving cars use planning to navigate roads, make turns, stop at traffic signals, and avoid collisions with pedestrians or other vehicles.
 - ❖ **Gaming:** In video games, AI uses planning to simulate intelligent behavior in non-player characters (NPCs). NPCs can plan their strategies in real-time, providing more challenging and unpredictable gameplay.
-

- ❖ **Supply Chain Management:** Planning in AI optimizes logistics, inventory, and transportation, helping businesses improve efficiency and reduce costs. AI can plan the most cost-effective routes for shipping goods or the best times to restock inventory.

4.2 Strips

STRIPS is a formal language used for expressing planning problems and was originally designed to control the actions of a robot in a manipulable environment. It is primarily concerned with the automatic generation of plans, which are sequences of actions that transition a system from its initial state to a desired goal state.

Key Components of STRIPS:

States: Defined by a set of logical propositions.

Goals: Specified as a set of conditions that describe the desired outcome.

Actions: Each action in STRIPS is characterized by three components:

- ❖ **Preconditions:** Conditions that must be true for the action to be executed.
- ❖ **Add Effects:** Conditions that become true as a result of executing the action.
- ❖ **Delete Effects:** Conditions that become false as a result of executing the action.

STRIPS in AI: Leveraging Heuristics and Symbols for Effective Problem Solving

Before going through the details, we must be familiar about the terms heuristics and symbols.

Heuristics: are techniques that helps individuals to solve problems in feasible amount of time. It aims to find approximate solution and in some cases may find the optimal solution as well.

Symbols: are representations that acts as a medium between human knowledge and the AI systems so that the systems can understand the knowledge that it receives and establish the relationships.

Basically STRIPS makes use of heuristics and symbols to solve a real world problem. The features of STRIPS are as follows:

- ❖ STRIPS makes use of symbols to represent knowledge so that AI systems can process the incoming data.
- ❖ STRIPS makes use of logical reasoning so that appropriate relationship is established among the symbols and that we get appropriate outputs based on the reasoning.
- ❖ STRIPS plans the execution of sub problems in a sequential fashion.

How STRIPS Works in AI?

The STRIPS algorithm operates by maintaining a database of predicates that describe the state of the world. Each action available to the system is defined in terms of its preconditions and its effects (both add and delete). The planning process in STRIPS involves searching through the space of possible actions to find a sequence that transitions the system from the initial state to the state where the goal predicates are satisfied.

Planning with STRIPS:

1. **Define the Initial State:** Where the system starts.
2. **Set the Goal State:** What the system should achieve.
3. **Develop Actions:** Defined by their preconditions and effects.
4. **Search for Solutions:** Using a strategy like backward chaining from the goal state to the initial state, identifying actions that satisfy the goal conditions.

Using STRIPS for Block Stacking in AI

Problem Statement: Given three blocks labeled A, B, and C, the objective is to stack Block A on Block B, and Block B on Block C.

Initial State Representation

In STRIPS, the initial state of the environment is crucial for defining the problem. For our scenario:

OnTable(A) indicates Block A is on the table.

OnTable(B) indicates Block B is on the table.

OnTable(C) indicates Block C is on the table.

The notation **OnTable(X)** is used in STRIPS to denote that block X is resting on the table.

Goal State

The goal state specifies the desired arrangement of blocks:

On(A,B) means Block A should be on top of Block B.

On(B,C) means Block B should be on top of Block C.

Using **On(X,Y)**, we denote that Block X is directly on top of Block Y.

Step-by-Step STRIPS Planning Approach

The solution involves a series of actions, each changing the state of the blocks to move towards the goal state. STRIPS formalizes this with actions defined by preconditions (what must be true before the action) and effects (what becomes true after the action).

1. Action: MoveBlock(A, B)

Preconditions:

- ❖ **OnTable(A)**: A is on the table.
- ❖ **Clear(B)**: B has no other blocks on it.

Effects:

- ❖ **On(A,B)**: A is now on B.
- ❖ **Clear(A)**: Top of A is clear.
- ❖ **¬OnTable(A)**: A is no longer on the table.
- ❖ **¬Clear(B)**: B is no longer clear since A is on it.

This action transitions Block A from the table to being on top of Block B, achieving our first sub-goal **On(A,B)**.

2. Action: MoveBlock(B, C)

Preconditions:

- ❖ **OnTable(B)**: B is on the table.
- ❖ **Clear(C)**: C has no other blocks on it.

Effects:

- ❖ **On(B,C)**: B is now on C.
- ❖ **Clear(B)**: Top of B is clear.
- ❖ **¬OnTable(B)**: B is no longer on the table.
- ❖ **¬Clear(C)**: C is no longer clear since B is on it.

With this action, Block B is moved onto Block C, achieving the second sub-goal **On(B,C)**.

Results

Combining the results of these actions, we find:

Final Configuration: Block A is on Block B, and Block B is on Block C. This configuration satisfies our goal state of **On(A,B)** and **On(B,C)**.

This methodical approach using STRIPS not only simplifies complex problems into manageable actions but also ensures that each step is logically sound, leading to a successful execution in environments such as robotics and automated planning systems.

Applications of STRIPS

STRIPS methodology is widely used in many fields. Some applications are as follows:

1. Manufacturing of automobiles

Manufacturing of automobiles requires sequential planning. Here STRIPS play a very important role since it is action-goal oriented. For instance, first the parts needs to be assembled. Then the the parts are embedded in the automobile one by one. Finally the automobiles are painted and lubricated followed by testing.

2. Robotic Systems

STRIPS mechanism is widely used in Robots. For example there are three blocks A, B, C lying on the ground. A robot arm is present and the goal is to place Block B on Block A and Block C on Block B. So the robot arm needs to plan its actions in such a way so that the

blocks are placed on one another and that the sequence is maintained while achieving the goal.

3. Knowledge Representation

Since STRIPS makes use of symbols, it is widely used in representing knowledge. It helps to represent the actions, states and goals. It also ensures that appropriate logic has been established for each action that has been taken.

Applications of STRIPS in AI

STRIPS has been fundamental in the development of AI systems across various domains, including:

- ❖ **Robotics:** STRIPS algorithms are used to plan the sequence of movements and tasks for robots, especially in manufacturing and assembly lines where precise and repetitive actions are required.
- ❖ **Video Games:** AI characters and non-player characters (NPCs) use planning algorithms derived from STRIPS to decide on actions based on player behavior and game dynamics.
- ❖ **Automated Reasoning:** STRIPS provides a basis for systems that need to reason about the effects of actions in various scenarios, such as automated theorem provers and problem solvers.

Limitations and Evolution

While STRIPS was revolutionary, it has limitations, primarily its assumption of a static world and the lack of support for actions with nondeterministic outcomes or concurrent actions. These limitations led to the development of more sophisticated planning languages like PDDL (Planning Domain Definition Language), which extend and generalize the concepts introduced by STRIPS to accommodate more complex planning scenarios

4.3 Advanced Plan generation systems

One of the earliest techniques is planning using goal stack. Problem solver uses single stack that contains

sub goals and operators both

sub goals are solved linearly and then finally the conjoined sub goal is solved.

Plans generated by this method will contain complete sequence of operations for solving one goal followed by complete sequence of operations for the next etc.

Problem solver also relies on

A database that describes the current situation.

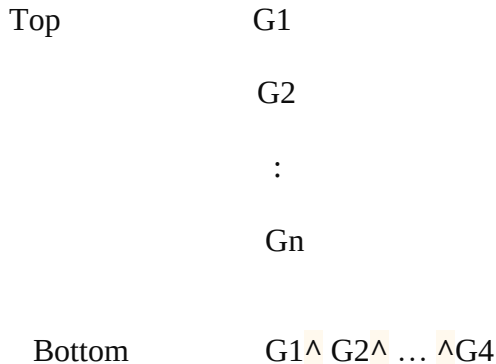
Artificial Intelligence(AI) with it's Applications

Set of operators with precondition, add and delete lists.

Let us assume that the goal to be satisfied is:

$$\text{GOAL} = G1 \wedge G2 \wedge \dots \wedge G_n$$

Sub-goals $G1, G2, \dots, G_n$ are stacked with compound goal $G1 \wedge G2 \wedge \dots \wedge G_n$ at the bottom.



At each step of problem solving process, the top goal on the stack is pursued.

Algorithm

Find an operator that satisfies sub goal $G1$ (makes it true) and replace $G1$ by the operator.

If more than one operator satisfies the sub goal then apply some heuristic to choose one.
In order to execute the top most operation, its preconditions are added onto the stack.

Once preconditions of an operator are satisfied, then we are guaranteed that operator can be applied to produce a new state.

New state is obtained by using ADD and DELETE lists of an operator to the existing database.

Problem solver keeps track of operators applied.

This process is continued till the goal stack is empty and problem solver returns the plan of the problem.

Goal Stack Example

With this example, let us explain the working method of Goal Stack Algorithm.

Artificial Intelligence(AI) with it's Applications



Initial State: $ON(B, A) \wedge ONT(C) \wedge ONT(A) \wedge ONT(D) \wedge CL(B) \wedge CL(C) \wedge CL(D) \wedge AE$

Goal State: $ON(C, A) \wedge ON(B, D) \wedge ONT(A) \wedge ONT(D) \wedge CL(C) \wedge CL(B) \wedge AE$

We notice that following sub-goals in goal state are also true in initial state.

$$ONT(A) \wedge ONT(D) \wedge CL(C) \wedge CL(B) \wedge AE$$

Represent for the sake of simplicity - TSUBG.

Only sub-goals $ON(C, A)$ & $ON(B, D)$ are to be satisfied and finally make sure that TSUBG remains true.

Either start solving first $ON(C, A)$ or $ON(B, D)$. Let us solve first $ON(C, A)$.

Goal Stack:

$ON(C, A)$

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge TSUBG$

To solve $ON(C, A)$, operation $S(C, A)$ could only be applied.
So replace $ON(C, A)$ with $S(C, A)$ in goal stack.

Goal Stack:

$S(C, A)$
 $ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge TSUBG$

$S(C, A)$ can be applied if its preconditions are true. So add its preconditions on the stack.

Goal Stack:

CL(A)
HOLD(C) Preconditions of STACK
 $CL(A) \wedge HOLD(C)$
S (C, A) Operator
ON(B, D)
 $ON(C, A) \wedge ON(B, D) \wedge TSUBG$

To do the $S(C, A)$ operation all preconditions should be true. In the given problem $CL(A)$ is not true. So, to make the state true, replace $CL(A)$ by $U(B, A)$ and write the preconditions of Unstack operator.

Goal Stack:

ON(B, A)
CL(B) Preconditions of UNSTACK
AE
 $ON(B, A) \wedge CL(B) \wedge AE$

US(B, A) Operator

HOLD(C) Preconditions of STACK
 $CL(A) \wedge HOLD(C)$

S (C, A) Operator

ON(B, D)

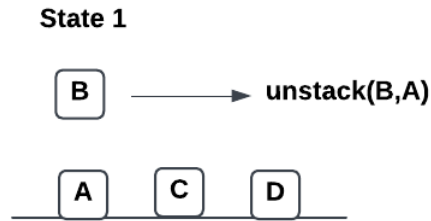
$ON(C, A) \wedge ON(B, D) \wedge TSUBG$

- ❖ $ON(B, A)$, $CL(B)$ and AE are all true in initial state, so pop these along with its compound goal.
- ❖ Next pop top operator $US(B, A)$ and produce new state by using its ADD and $DELETE$ lists.
- ❖ Add $US(B, A)$ in a queue of sequence of operators.

SQUEUE = US (B, A)

State_1:

$ONT(A) \wedge ONT(C) \wedge ONT(D) \wedge HOLD(B) \wedge CL(A) \wedge CL(C) \wedge CL(D)$



Goal Stack:

HOLD(C) Preconditions of STACK
 $CL(A) \wedge HOLD(C)$

S (C, A) Operator

ON(B, D)

$ON(C, A) \wedge ON(B, D) \wedge TSUBG$

To execute the S(C,A),all the preconditions of Stack operator should be true.But in this case HOLD(C) is not true .To make the state true use the operator S(B,D)

S(B,D) Operator

HOLD(C)
 $CL(A) \wedge HOLD(C)$ Preconditions of STACK

S (C, A) Operator

ON(B, D)

$ON(C, A) \wedge ON(B, D) \wedge TSUBG$

Write down the preconditions of S(B,D)

Goal Stack

$CL(D) \wedge HOLD(B)$ Preconditions of STACK

S(B,D) Operator

HOLD(C)
 $CL(A) \wedge HOLD(C)$ Preconditions of STACK

ON(B, D)

ON(C, A) $\wedge$ ON(B, D) $\wedge$ TSUBG

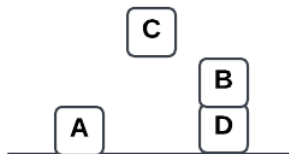
Here, all the preconditions of PU operator is true,so add PU(C) in a queue of sequence of operators.

SQUEUE = US (B, A), S (B, D),PU(C)

State_3:

ONT(A) $\wedge$ HOLD(C) $\wedge$ ONT(D) $\wedge$ ON(B, D) $\wedge$ CL(A) $\wedge$ CL(B)

State 3



Goal Stack:

HOLD(C)

CL(A)) $\wedge$ HOLD(C)

Preconditions of STACK

S (C, A)

Operator

ON(B, D)

ON(C, A) $\wedge$ ON(B, D) $\wedge$ TSUBG

Here all the preconditions of S(C,A) is true ,so add S(C,A) in queue

SQUEUE = US (B, A), S (B, D),PU(C),S(C,A)

State_4:

ONT(A) $\wedge$ ON(C, A) $\wedge$ ONT(D) $\wedge$ ON(B, D) $\wedge$ CL(C) $\wedge$ CL(B) $\wedge$ AE

Goal State



Finally, we reached goal state after S(C,A) using Goal Stack algorithm, so the plan for the given problem is,

UnStack (B, A)
Stack (B, D)
PickUp(C)
Stack(C,A)

4.4 K strips

Modal Operator K:

We are familiar with the use of connectives $\wedge$ and $\vee$ in logics. Thinking of these connectives as operators that construct more complex formulas from simpler components. Here, we want to construct a formula whose intended meaning is that a certain agent knows a certain proposition.

The components consist of a term denoting the agent and a formula denoting a proposition that the agent knows. To accomplish this, modal operator K is introduced.

For example, to say that Robot (name of agent) know that block A is on block B, then write, $K(\text{Robot}, \text{On}(A,B))$

The sentence formed by combining K with the term Robot and the formula $\text{On}(A,B)$ gets a new formula, the intended meaning of which is “Robot knows that block A is on block B”.

The words “knows” and “belief” is different in meaning. That means an agent can believe a false proposition, but it cannot know anything that is false.

Some examples,

$K(\text{Agent1}, K(\text{Agent2}, \text{On}(A,B)))$, means Agent1 knows that Agent2 knows that A is on B.

$K(\text{Agent1}, \text{On}(A,B)) \vee K(\text{Agent1}, \text{On}(A,C))$ means that either Agent1 knows that A is on B or it knows that A is on C.

$K(\text{Agent1}, \text{On}(A,B)) \vee K(\text{Agent1}, \neg \text{On}(A,B))$ means that either Agent1 knows whether or not A is on B.

Knowledge Axioms:

The operators $\wedge$ and $\vee$ have compositional semantics (depends on truth value), but the semantics of K are not compositional. The truth value of $K(\text{Agent1}, \text{On}(A,B))$ for example, cannot necessarily be determined from the properties of K, the denotation of Agent1 and the truth value of $\text{On}(A,B)$. K Operator is said to be referentially opaque.

Example in Planning Speech Action:

We can treat speech acts just like other agent systems. Our agent can use a plan-generating system to make plans comprising speech acts and other actions. To do so, it needs a model of the effects of these actions.

Consider for example, $\text{Tell}(A, \phi)$, where A is Agent and ϕ is true.
We could model the effects of that action by the STRIPS rule :

$\text{Tell}(A, \phi) :$

Precondition : $\text{Next_to}(A) \wedge \phi \wedge \neg K(A, \phi)$

Delete : $\neg K(A, \phi)$

Add : $K(A, \phi)$

The precondition $\text{Next_to}(A)$ ensures that our agent is close to agent A to enable communication.

The precondition ϕ is imposed to ensure that our agent actually believes ϕ before it can inform another agent about the truth.

The precondition $\neg K(A, \phi)$ ensure that our agent does not communicate redundant information.

4.5 Learning

Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning**.

A subset of artificial intelligence known as machine learning focuses primarily on the creation of algorithms that enable a computer to independently learn from data and previous experiences. Arthur Samuel first used the term "machine learning" in 1959. It could be summarized as follows:

Without being explicitly programmed, machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things.

Machine learning algorithms create a mathematical model that, without being explicitly programmed, aids in making predictions or decisions with the assistance of sample historical data, or training data. For the purpose of developing predictive models, machine learning brings together statistics and computer science. Algorithms that learn from historical data are either constructed or utilized in machine learning. The performance will rise in proportion to the quantity of information we provide.

How does Machine Learning work

A machine learning system builds prediction models, learns from previous data, and predicts the output of new data whenever it receives it. The amount of data helps to build a better model that accurately predicts the output, which in turn affects the accuracy of the predicted output.

Let's say we have a complex problem in which we need to make predictions. Instead of writing code, we just need to feed the data to generic algorithms, which build the logic based on the data and predict the output. Our perspective on the issue has changed as a result of machine learning. The Machine Learning algorithm's operation is depicted in the following block diagram:

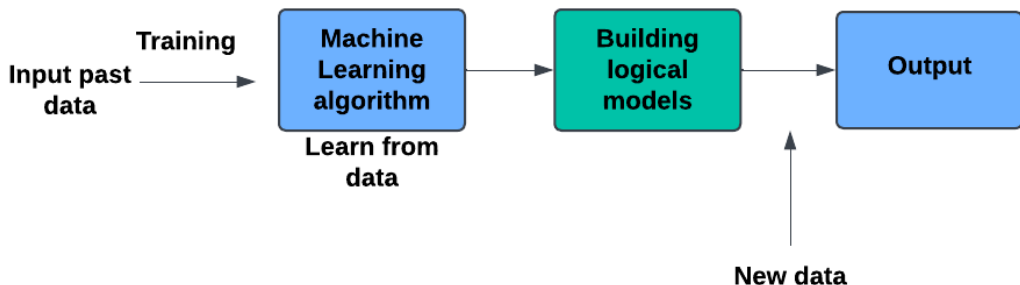


Fig. 4.3 Working of Machine Learning.

Features of Machine Learning:

- ❖ Machine learning uses data to detect various patterns in a given dataset.
- ❖ It can learn from past data and improve automatically.
- ❖ It is a data-driven technology.
- ❖ Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Need for Machine Learning

The demand for machine learning is steadily rising. Because it is able to perform tasks that are too complex for a person to directly implement, machine learning is required. Humans

Artificial Intelligence(AI) with it's Applications

are constrained by our inability to manually access vast amounts of data; as a result, we require computer systems, which is where machine learning comes in to simplify our lives.

By providing them with a large amount of data and allowing them to automatically explore the data, build models, and predict the required output, we can train machine learning algorithms. The cost function can be used to determine the amount of data and the machine learning algorithm's performance. We can save both time and money by using machine learning.

The significance of AI can be handily perceived by its utilization's cases, Presently, AI is utilized in self-driving vehicles, digital misrepresentation identification, face acknowledgment, and companion idea by Facebook, and so on. Different top organizations, for example, Netflix and Amazon have constructed AI models that are utilizing an immense measure of information to examine the client interest and suggest item likewise.

Following are some key points which show the importance of Machine Learning:

- ❖ Rapid increment in the production of data
- ❖ Solving complex problems, which are difficult for a human
- ❖ Decision making in various sector including finance
- ❖ Finding hidden patterns and extracting useful information from data.

Classification of Machine Learning

At a broad level, machine learning can be classified into three types:

1. **Supervised learning**
2. **Unsupervised learning**
3. **Reinforcement learning**

1) Supervised Learning

In supervised learning, sample labeled data are provided to the machine learning system for training, and the system then predicts the output based on the training data.

The system uses labeled data to build a model that understands the datasets and learns about each one. After the training and processing are done, we test the model with sample data to see if it can accurately predict the output.

The mapping of the input data to the output data is the objective of supervised learning. The managed learning depends on oversight, and it is equivalent to when an understudy learns things in the management of the educator. Spam filtering is an example of supervised learning.

Supervised learning can be grouped further in two categories of algorithms:

- ❖ **Classification**

- ❖ **Regression**

2) Unsupervised Learning

Unsupervised learning is a learning method in which a machine learns without any supervision.

The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.

In unsupervised learning, we don't have a predetermined result. The machine tries to find useful insights from the huge amount of data. It can be further classified into two categories of algorithms:

- ❖ **Clustering**

- ❖ **Association**

3) Reinforcement Learning

Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance. In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.

The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.

Adaptive Learning

Adaptive learning in Artificial Intelligence (AI) refers to the use of intelligent algorithms and data-driven techniques to tailor educational content and experiences to the needs, preferences, and learning pace of individual learners. This approach is revolutionizing traditional teaching and training methods by enabling a personalized learning journey that evolves based on the learner's interactions.

Key Features of Adaptive Learning in AI

1. Personalization:

- ❖ The system analyzes the learner's strengths, weaknesses, and preferences.
- ❖ Content is customized to match the learner's knowledge level, skills, and goals.
- ❖ Adjustments are made dynamically as the learner progresses.

2. Data-Driven Insights:

- ❖ AI collects data from various sources, such as quizzes, assignments, and real-time responses.
- ❖ The data helps identify patterns, predict outcomes, and make informed adjustments.

3. Real-Time Feedback:

- ❖ Learners receive instant feedback on their performance.
- ❖ The system provides hints, additional resources, or alternative explanations when needed.

4. Scalability:

- ❖ Adaptive learning platforms can handle large numbers of learners while maintaining a personalized approach.
- ❖ It's applicable in diverse fields, from K-12 education to corporate training.

5. Dynamic Content Delivery:

- ❖ The system can present different types of content, such as videos, interactive simulations, or textual materials, depending on the learner's preferred style.
- ❖ Challenges are adjusted based on learner competence to avoid frustration or boredom.

Technologies Behind Adaptive Learning

1. Machine Learning (ML):

- ❖ Learns from data on user interactions to predict optimal learning paths.
- ❖ Clustering and classification algorithms group learners with similar patterns for better recommendation systems.

2. Natural Language Processing (NLP):

- ❖ Understands and responds to user queries in human language.
- ❖ Used in chatbots, virtual tutors, and question-answer systems.

3. Reinforcement Learning:

- ❖ AI agents optimize learning strategies by receiving rewards for successful outcomes.
- ❖ Helps in devising strategies for student engagement and motivation.

4. Big Data Analytics:

- ❖ Analyzes vast amounts of educational data to uncover trends and insights.
- ❖ Enables predictive analytics for identifying at-risk students or improving curriculum design.

5. Cognitive Computing:

- ❖ Simulates human thought processes to adapt to complex learner behavior.
-

- ❖ Creates a more interactive and human-like learning experience.

Benefits of Adaptive Learning in AI

- 1. Enhanced Learning Outcomes:**
 - ❖ Personalized pacing ensures that learners fully understand concepts before moving forward.
- 2. Efficient Use of Resources:**
 - ❖ Teachers and trainers can focus on strategic intervention while AI handles routine content delivery.
- 3. Equity in Education:**
 - ❖ Ensures that all learners, regardless of background, receive an equal opportunity to excel.
- 4. Continuous Improvement:**
 - ❖ AI systems evolve over time, becoming more effective as they process more data.
- 5. Engagement and Motivation:**
 - ❖ By adapting to interests and providing challenges at the right level, learners stay motivated.

Applications of Adaptive Learning

- 1. Education:**
 - ❖ K-12 and higher education platforms like Khan Academy and Coursera use adaptive learning to cater to diverse learner needs.
- 2. Corporate Training:**
 - ❖ Customized training programs for employees in sectors like IT, healthcare, and finance.
- 3. Language Learning:**
 - ❖ Platforms like Duolingo adapt lessons based on user proficiency and errors.
- 4. Skill Development:**
 - ❖ Professional certification courses use adaptive tools to ensure mastery of concepts.
- 5. Special Education:**
 - ❖ Supports learners with disabilities by providing tailored assistance and resources.

Challenges and Limitations

- 1. Data Privacy and Security:**
 - ❖ Handling sensitive learner data responsibly is a critical concern.
 - 2. High Initial Investment:**
 - ❖ Developing adaptive learning systems requires significant time and financial resources.
 - 3. Algorithm Bias:**
 - ❖ Improperly trained models can perpetuate biases, leading to unequal learning opportunities.
-

4. **Dependence on Technology:**

- ❖ Requires access to devices and stable internet connectivity, which can be a barrier in underdeveloped regions.

Future of Adaptive Learning in AI

1. **Integration with AR/VR:**

- ❖ Immersive technologies will enhance engagement and create realistic learning scenarios.

2. **Cross-disciplinary Learning:**

- ❖ AI could enable seamless integration of topics, fostering a holistic understanding.

3. **Lifelong Learning:**

- ❖ Adaptive systems will evolve to support continuous skill development across careers and life stages.

4. **Global Access:**

- ❖ With advancements in AI and internet access, adaptive learning can reach even the most remote areas.

Expert System

5.1 Expert System

One of the largest area of applications of artificial intelligence is expert systems or knowledge based systems as they are often known. This type of systems seeks to exploit the specialised skills or information held by a group of people on specific areas. It can be thought of as a computerised consulting service. It can also be called an information guidance system. Such systems are used for prospecting medical diagnosis or as educational aids. They are also used in engineering and manufacture in the control of robots where they inter-relate with vision systems. The initial attempts to apply artificial intelligence to generalised problems did not achieve success and had limited progress.

Definition of Expert System

Following are various definitions of expert systems :-

Definition 1 - Expert system is a artificial intelligence based system that converts the knowledge of an expert in a specific subject into a software code. This code can be merged with other such codes based on the knowledge of other experts, and used for answering queries submitted through a computer.

Definition 2 - Expert system is a piece of software which uses databases of expert knowledge to offer advice or make decisions in such areas as medical diagnosis.

Definition 3 - Expert system is a computer program that contains a knowledgebase and a set of algorithms or rules that infer new facts from knowledge and from incoming data. An expert system is an AI application that uses a knowledgebase of human expertise to aid in solving problems.

Definition 4 - Expert system is a model and associated procedure that exhibits, within a specific domain, a degree of expertise in problem solving that is comparable to a human expert.

Definition 5 - Expert system is a computing system capable of representing and reasoning about some knowledge rich domain, which usually requires a human expert, with a view towards solving problems and/or giving advice. Its level of performance makes it "expert".

Definition 6 - Expert system = Knowledge + Inference engine.

All in all, an expert system contains knowledge acquired by interviewing human expertise in some domain. Expert system can not operate in situations that call for common sense. Thus, Expert system perform a task in limited domains that require human expertise like medical diagnosis, fault diagnosis, status monitoring, data interpretation, mineral exploration, tutoring, computer configuration, credit checking etc.

Common Sense Reasoning

Computers have an entirely deserved reputation for lacking common-sense.

A human expert has two main types of knowledge -

- i) Domain-specific knowledge
- ii) Common-sense knowledge

Domain-specific knowledge deals with all kinds of knowledge about a particular domain. All other pieces of knowledge that help in reasoning other than domain-specific knowledge are termed as common-sense knowledge.

Common-sense knowledge when coupled, with domain-specific knowledge increases the performance capability of the system.

There are number of techniques available that can be used to enable an AI program to represent and reason with common-sense knowledge.

As of yet, however, no program can match the common-sense reasoning powers of a five-years-old-child.

To understand the common-sense reasoning consider the following real life situation.

Ram enjoys shopping in the mall only when the stores are not crowded. He has agreed to accompany Shyam there on the following Wednesday evening since this is normally a time when few people at shop.

Before the given data, several of the larger stores announce a one-time, special sale starting on that Wednesday evening.

Ram fearing large crowds, now retracts the offer to accompany Shyam, promising to go on some future date.

On the Tuesday before the sale was to commence, whether forecasts predicted heavy snow.

Now believing the whether would discourage most shoppers, Ram once again agreed to join Shyam. But, unexpectedly, on the given Wednesday, the forecasts proved to be false; so Ram once again declined to go.

This anecdote illustrates that how one's beliefs can change in a dynamic environment. And while one's beliefs may not fluctuate as much as Ram's in most situations. This form of belief is not uncommon. Indeed it is common enough that we label it as a form of common-sense reasoning i.e. reasoning with uncertain knowledge.

Why to Use an Expert System ?

Expert System (ES) helps to preserve knowledge. ES can be well suitable and affordable when expertise is expensive or unavailable. Human can fail to work under pressure and time bound but system would not. ES can train new employees thereby saving expertise time.

ES can save cost, as it can perform same task repeatedly and thereby providing good quality and uniform performance. ES can provide rigid solution for large problems.

Is Expert System Feasible ?

The major factor to consider for feasibility is the cost. How much is the need of ES would justify the cost to be incurred ?

When human expertise is not available ES is best solution in all criterion and conditions.

The major problem faced by ES is what is the size and scope of the system. This is relative to resources and evolving technology.

ES implementation is structured and does not require any common sense reasoning and hence perform uniformly with respect to all situations.

Advantages and Disadvantages of Expert System

Advantages of expert system

1. A well functioning ES provides increased distribution of expertise.
2. ES gives broader job description for individual workers.
3. ES can work round the clock.
4. ES can be used by the user more frequently.
5. ES can be trained faster as compare to human, therefore training becomes less time and money consuming.
6. ES enables new services.
7. ES work as a new communication channel for knowledge.
8. ES holds and maintains significant levels of information.

9. ES has faster adaptation to changing conditions.
ES never "forgets" to ask a question, as a human might.
10. A multi-user expert system can serve more users at a time.

Disadvantages of expert system

1. ES lacks common sense needed in some decision making.
2. ES can not make creative responses as human expert would in unusual circumstances.
3. Domain experts may not be always able to explain their logic and reasoning.
4. When the problem is outside the domain of ES then ES can not respond reasonably well when no answer exists.
5. Errors may occur in the knowledgebase, and lead to wrong decisions by ES.
6. ES can not adapt to changing environments, unless knowledgebase is changed. That is ES can not be made to do self learning.
7. ES is dependent on symbolic input only and can not respond to sensory experience as human can do.

Limitations of Expert Systems (ES)

- ❖ Knowledge may not always be readily accessible.
- ❖ Extracting expertise from human experts can be challenging.
- ❖ There are often multiple valid solutions or assessments for a given problem.
- ❖ Processing under strict time constraints can be a challenge for ES.
- ❖ Users may have cognitive limitations that affect their interaction with the system.
- ❖ ES performs effectively only within a narrowly defined domain of expertise.
- ❖ Experts often lack independent mechanisms to validate the results produced by the system.
- ❖ The vocabulary used in the knowledge base can be limited and difficult to comprehend.
- ❖ Assistance from knowledge engineers is both hard to acquire and expensive.
- ❖ End-users may exhibit a lack of trust in the system.
- ❖ Biases can influence the process of transferring knowledge to the system.

Factors Contributing to the Success of Expert Systems

- ❖ The system must possess a sufficiently high level of knowledge.
-

- ❖ Expertise must be available from at least one domain specialist.
- ❖ The problem being addressed should involve some degree of uncertainty or ambiguity.
- ❖ The problem should be specific and focused rather than broad or general.
- ❖ The ES framework (shell) must be robust, effectively managing and manipulating knowledge.
- ❖ The user interface should be intuitive and user-friendly, even for non-expert users.
- ❖ The problem should be complex and significant enough to warrant the development of an expert system.
- ❖ Skilled developers with strong interpersonal and communication skills are crucial.
- ❖ The potential impact of the system should be carefully evaluated.
- ❖ The system's impact should result in positive outcomes.
- ❖ Support and backing from management are essential for success.

5.2 Expert System Architecture

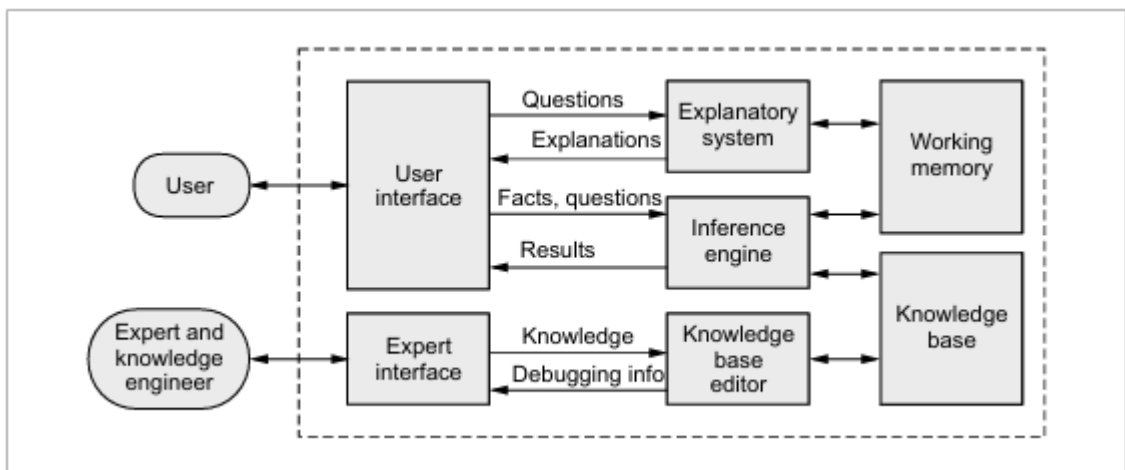


Fig. 5.1 Constitute of domain-specific and general knowledge section.

The user interacts with the system through a user interface which may use menus, natural language or any other style of interaction. Then an inference engine is used to reason with both the expert knowledge (extracted from the expert) and data specific to the particular problem being solved. The expert knowledge will typically be in the form of a set of IF-THEN rules. The case specific data includes both data provided by the user and partial conclusions (along with certainty measures) based on this data. In a simple forward chaining rule-based system the case specific data will be the elements in working memory.

Every expert system consists of two principle parts : The knowledge base; and the reasoning or inference engine. The knowledge base of expert systems contains both factual and heuristic

knowledge. Factual knowledge is that knowledge of the task domain that is widely shared, typically found in textbooks or journals, and commonly agreed upon by those knowledgeable in the particular field. Heuristic knowledge is the less rigorous, gained from experience, more judgemental knowledge of performance. In contrast to factual knowledge, heuristic knowledge is rarely discussed, and is largely individualistic. It is the knowledge of good practice, good judgement, and plausible reasoning in the field. It is the knowledge that underlies the "art of good guessing".

Knowledge representation formalizes and organizes the knowledge. One widely used representation is the production rule, or simply rule. A rule consists of an IF part and a THEN part (also called a condition and an action). The IF part lists a set of conditions in some logical combination. The piece of knowledge represented by the production rule is relevant to the line of reasoning being developed if the IF part of the rule is satisfied; consequently, the THEN part can be concluded, or its problem-solving action taken. Expert systems whose knowledge is represented in rule form are called rule-based systems.

Generally the expert systems also have an explanation subsystem, which allows the program to explain its reasoning to the user. Some systems also have a knowledge base editor which help the expert or knowledge engineer to easily update and check the knowledge base.

Another important feature of expert systems is the way they (usually) separate domain specific knowledge from more general purpose reasoning and representation techniques. The general purpose module (in the dotted box in the figure) is referred to as an expert system shell. As seen in the **Fig. 5.2**, the shell will provide the inference engine (and knowledge representation scheme), a user interface, an explanation system and sometimes a knowledge base editor. Given a new kind of problem to solve (say, TV set design), one can usually find a shell that provides the right sort of support for that problem, so what is needed to do is provide the expert knowledge. There are numerous commercial expert system shells, each one appropriate for a slightly different range of problems. (Expert systems work in industry includes both writing expert system shells and writing expert systems using shells). Using shells to write expert systems generally greatly reduces the cost and time development (compared with writing the expert system from scratch).

An ideal Expert System should include the following :-

- ❖ Extensive specific knowledge from the domain of interest.
- ❖ Extensive database interfaces.
- ❖ Application of several techniques.
- ❖ Support for heuristic analysis.
- ❖ Capacity to infer new knowledge from existing knowledge.
- ❖ Decisions under uncertainty.

- ❖ Decisions with unknowns.
- ❖ Symbolic processing.
- ❖ An ability to explain its own reasoning.

Types of Architecture

i) Rule Based Architecture of an Expert System

The most common form of architecture used in expert and other types of knowledge based systems is the production system or it is called rule based systems. This type of system uses knowledge encoded in the form of production rules i.e. if-then rules. The rule has a conditional part on the left hand side and a conclusion or action part on the right hand side. For example if: condition1 and condition 2 and condition3

Then: Take action 4

Each rule represents a small chunk of knowledge to the given domain of expertise. When the known facts support the conditions in the rule's left side, the conclusion or action part of the rule is then accepted as known. The rule based architecture of an expert system consists of the domain expert, knowledge engineer, inference engine, working memory, knowledge base, external interfaces, user interface, explanation module, database spreadsheets executable programs s mentioned in **Fig. 5.2**.

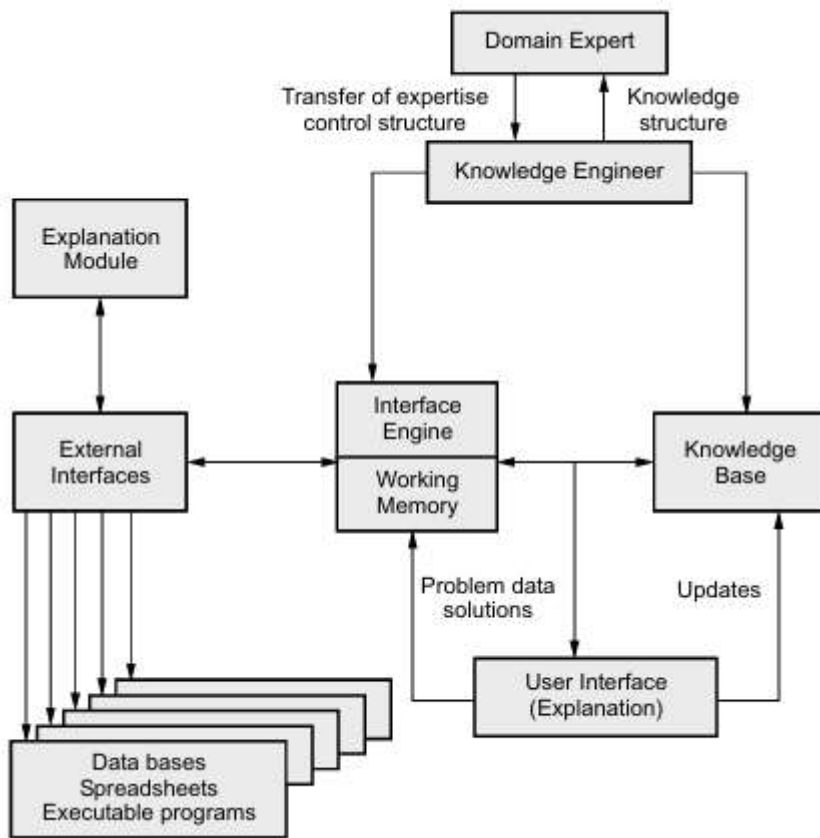


Fig. 5.2 Expert system : Rule based architecture

Integration of Expert Systems Components

The components of the rule based architecture are as follows.

1. **User Interface** : It is the mechanism by which the user and the expert system communicate with each other i.e. the user interacts with the system through a user interface. It acts as a bridge between user and expert system. This module accepts the user queries and submits those to the expert system. The user normally consults the expert system for following reasons.
 - a) To get answer of his/her queries.
 - b) To get explanation about the solution for psychological satisfaction.

The user interface module is designed in such a way that at user level it accepts the query in a language understandable by expert system. To make the expert system user friendly, the user interface interacts with the user in natural language. The user interface provides as much facilities as possible such as menus, graphical interfaces etc. to make the dialog user friendly and more attractive.

2. Explanation Module : The explanation module explains the reasoning of the system to a user. It provides the user with an explanation of the reasoning process when requested. The credibility of expert system will be established only when it is able to explain "how and why" a particular conclusion is drawn. This explanation increases the belief of user in the expert system.

a) Explanation (How) : To respond to a how query, the explanation module traces the chain of rules fired during a consultation with the user. This explanation mode can be activated once the process is over. It explains how a fact was deduced by the system and similarly how a rule was/wasn't used. The simplest way to specify this is to explain the rule which allows the deduction. For e.g.

If the system (S) will give information about the parent-child relationship to the user (U) then the followings can be possible.

S : My diagnosis is " A is the father of B "

U: How ?

S : The result was obtained by the addition of following facts and rules.

Fact no 11: A is the parent of Hari.

Fact no 15: A is a male.

Fact no 110: X is father of Y :

X is parent of Y, X is male.

So A is the father of B .

b) Explanation (Why)? To respond to a why query, the explanation module must be able to explain why certain information is needed by the inference engine to complete a step in the reasoning process. This mode of explanation is dynamic and is used when the system functions in backward chaining. If the user doesn't understand the reason behind the question. The system is capable of explaining why the question was asked. For example S: Is the following true ?

A is the father of B.

U:Why?

S:I need the fact:

A is the father of B to establish the following fact " B is the son of A ".

By using the rule no. 4

A is the father of B :

B is the son of A.

3. Working Memory : It is a global database of facts used by the rules.

4. Knowledge Engineering : The primary people involved in building an expert system are the knowledge engineer, the domain expert and the end user. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem solving sessions with the domain expert, he/she is ready to begin actually designing the system, selecting a way to represent the knowledge, determining the search strategy (backward or forward) and designing the user interface. After making complete designs, the knowledge engineer builds a prototype. The prototype should be able to solve problems in a small area of the domain. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its disadvantages.

5. Knowledge Base : In rule based architecture of an expert system, the knowledge base is the set of production rules. The expertise concerning the problem area is represented by productions. In rule based architecture, the condition actions pairs are represented as rules, with the premises of the rules (if part) corresponding to the condition and the conclusion (then part) corresponding to the action. Case-specific data are kept in the working memory. The core part of an expert system is the knowledge base and for this reason an expert system is also called a knowledge based system.

Expert system knowledge is usually structured in the form of a tree that consists of a root frame and a number of sub frames. A simple knowledge base can have only one frame, i.e. the root frame whereas a large and complex knowledge base may be structured on the basis of multiple frames.

Inference Engine : The inference engine accepts user input queries and responses to questions through the I/O interface. It uses the dynamic information together with the static knowledge stored in the knowledge base. The knowledge in the knowledge base is used to derive conclusions about the current case as presented by the user's input. Inference engine is the module which finds an answer from the knowledge base. It applies the knowledge to find the solution of the problem. In general, inference engine makes inferences by deciding which rules are satisfied by facts, decides the priorities of the satisfied rules and executes the rule with the highest priority. Generally inferring process is carried out recursively in 3 stages like match, select and execute. During the match stage, the contents of working memory are compared to facts and rules contained in the knowledge base. When proper and consistent matches are found, the corresponding rules are placed in a conflict set.

ii) Example Architecture : Blackboard System

A blackboard system is composed of an area of shared memory, referred to as the blackboard, which contains a problem to be solved and a number of different processes, referred to as knowledge sources that can access and modify the blackboard [Nii].

Artificial Intelligence(AI) with it's Applications

Each knowledge source will post a partial solution whenever doing so can contribute to the overall solution of the problem. These partial solutions cause other knowledge sources to update their portions of the solution on the blackboard until eventually an answer is found.

Blackboard systems are useful for tasks where different types of knowledge need to be brought to bear; e.g., natural language processing, speech processing, and perception.

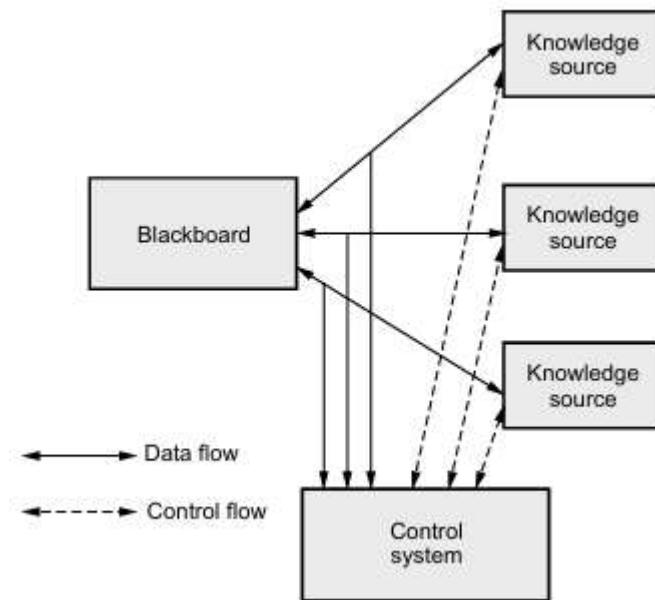


Fig. 5.3 Blackboard System.

Blackboard system application consists of three major components

1. The software specialist modules, which are called knowledge sources.
2. The blackboard, a shared repository of problems, partial solutions, suggestions, and contributed information.
3. The control shell, which controls the flow of problem-solving activity in the system.
4. Knowledge sources have a condition-action structure. Ordinarily the condition is a particular configuration of elements on the blackboard. The action ordinarily entails the creation or modification of solution elements on the blackboard.

The advantages of a blackboard include separation of knowledge into independent modules with each module being free to use the appropriate technology to arrive at the best solution with the most efficiency.

Famous examples are the Hearsay-II speech recognition system based on the BB1 architecture and Douglas Hofstadter's Copycat program. Other systems include HASP/SIAP that maintained surveillance of surface ships and submarines from sonar data; CRYSLIS, which was designed to infer the three-dimensional structure of protein molecules; and TRICERO, which monitored an area of airspace for traffic. The Intelligent Intermediary for Information Retrieval (IIR), developed by [Croft and Thompson], consists of a group of experts that communicate via a common data structure.

Two free blackboard systems are GBB (UMass blackboard system) and GEST (Generic Expert System Tool). BBTech Corporation sells blackboard systems, multi-agent organizations, and collaborating-software technology solutions.

5.3 Expert System Shell

Many expert systems are built with products called expert system shells. A shell is a piece of software which contains the user interface, a format for declarative knowledge in the knowledgebase and an inference engine. The knowledge and system engineers use these shells in making expert systems.

Earlier each expert system was built from scratch, usually in LISP. But, after several varieties of systems they had been built this way, it became clear that these systems often had a lot in common. In particular, since the systems were constructed as a set of declarative representations (mostly rules) combined with an interpreter for those representations, it was possible to create a system that would be used to construct new expert systems by adding new knowledge corresponding to the new problem domain. The resulting interpreters are called shell, example of such a shell is EMYCIN, which was derived from MYCIN.

Knowledge engineer uses shell to build a system for a particular domain. System engineer builds the user interface, designs the declarative format of the knowledgebase and implements the inference engine.

Several commercial shells are available which can work as base system for expert systems that are currently under developed. These shells provide rules frames, truth maintenance systems and a variety of other reasoning mechanisms. The expert system shells provide mechanisms for knowledge representation, reasoning and explanation. Later on knowledge acquisition tools were added to the shell. The ability of shell was not enough and new systems required more capabilities and tools. The shells were expected to integrate expert systems with other programs. Expert system need access to large databases and they are generally embedded within larger programs that use primarily conventional programming techniques. Therefore, expert system shell must provide an easy to use interface between an expert system that is written with the shell and a larger and more conventional programming environment.

5.4 Explanation

An important feature of expert system is the ability to explain itself. Given that the system knows which rules were used during the inference process, the system can provide those rules to the user as means for explaining the result. By looking at explanations, the knowledge engineer can see how the system is behaving and how the rules and data are interacting. This is very valuable diagnostic tool during development.

To make an effective expert system, people must be able to interact with it easily. To facilitate this interaction the expert system must have the following two capabilities in addition to the ability to perform its underlying task.

Explain its reasoning (Why, Why not and How explanations)

In many of the domains in which expert systems operate, people will not accept results unless they have been convinced of the accuracy of the reasoning process that produced those results. For example, in medical, a doctor must accept ultimate responsibility for a diagnosis, even if the diagnosis was arrived at with considerable help from a program. Thus it is important that reasoning process be used in such programs proceeds in understandable steps and that enough meta-knowledge is available so that explanations of those steps can be generated.

An expert system shell asks questions and provides advice and Justification (all via the user interface). Justification comes in two parts :

Why - explanation of the reasoning for asking a particular question.

How - explanation of the reasoning in providing a particular piece of advice.

Given that the system knows which rules were used during the inference process, it is possible for the system to provide those rules to the user as a means for explaining its current action and the obtained results.

Types of Explanation

a. Why a particular question is asked - usually by describing the rule that has as a condition the answer of the question.

b. How a particular conclusion has been reached - usually by listing the rules that have been applied.

c. Why not a given conclusion. The user put forward a hypothesis to be verified.

d. What if a particular symptom is present. Sometimes the user may want to see whether a particular symptom would prove a hypothesis or not.

e. Terminology - explains the meaning of system terminology to achieve better understanding by the user.

Acquiring new knowledge and modifying old knowledge

Expert systems derive their power from their richness of the knowledge bases they exploit, therefore it is extremely important that those knowledge bases should be as complete and as accurate as possible. But often there exists no standard codification of that knowledge, rather it exists only inside the heads of human experts. One way is to get this knowledge with the human experts, another way is to have the program learn experts behaviour from raw data.

Explanation mode allows the system to explain its conclusions and its reasoning process. This ability comes from the AND/OR trees created during the production system reasoning process. As a result most expert systems can answer the following why and how questions.

Why was a given fact used ?

Why was a given fact not used ?

How was a given conclusion reached ?

How was it that another conclusion was not reached?

Let us illustrate the action of an expert system by considering the behaviour of an actual one MYCIN, a medical expert system.

MYCIN has been informed that a patient has contracted Meningitis while in hospital. Doctors wish to interrogate MYCIN to check this diagnosis.

MYCIN then asks the doctors this question

MYCIN : Is Meningitis a hospital acquired illness ?

Doctor : why?

MYCIN : [2.0] This will aid in determining the organisms that might be causing the infection.

It has already established

[2.1] The infection that requires therapy is Meningitis.

[2.2] Organisms were not seen on the stain of this pending csf culture.

[2.3] The type of infection is bacterial.

[2.4] Neurosurgery is not one of the relevant items of patient 100.

[2.5] The age of patient is 100 is greater than 25.

Therefore if [2.6] the infection was acquired while the patient was in hospital then there is evidence that the organisms that might be causing the infection are E coli(0.75), staphylococcus-coag-pos(0.3), pseudomonas-aeruginosa(0.3), klebsiella-pneumoniae(0.5)

- ❖ If the doctors wanted to know how something was concluded they might ask Doctor:
How 2.3
The system then responds with :

MYCIN : The following rules concluded about the likelihood that the type of the infection is bacterial :

[2.7] RULE148 (0.19)

[2.8] RULE500 (0.51)

[2.9] RULE501 (0.95)

[3.0] RULE502 (0.97)

[3.1] RULE526 (0.98)

[3.2] RULE504 (0.97)

[3.3] RULE524 (0.98)

The system takes the doctors through the production system AND/OR tree to explain its conclusions. The numbers in brackets are called certainty factors because little can be absolutely certain. The certainty factors range from -1 to +1 where -1 means absolutely not and +1 absolutely so and 0 undecided either way.

A blackboard system

HEARSAY uses the concept of a blackboard system where each partition or subsystem is treated as a specialist. Each specialist is a set of production rules in its own right. It is illustrated below. HEARSAY is a speech recognition system that is about 90% accurate. The specialism for HEARSAY are also illustrated below.

Human experts communicate using TEIRESIAS with the performance program MYCIN to discover how well MYCIN is doing in responding to the requests for help that it receives. Through using TEIRESIAS the expert can coach MYCIN into producing better responses so that an element of learning is introduced.

TEIRESIAS recognizes that some of the knowledge required for the knowledge base is inexact or rough and it succeeds in offering advice for its correction. It also provides method for extracting the best performance out of an expert system such as MYCIN.

TEIRESIAS was the first program to support explanation and knowledge acquisition system of TEIRESIAS served as a front-end for the MYCIN expert system.

A crucial premise underlying TEIRESIAS's approach to explanation is that the behavior of a program can be explained simply by referring to a trace of the programs execution. There are ways in which this assumption limits the kinds of explanation that can be produced, but it does minimize the overhead involved in each explanation. To understand how TEIRESIAS generates explanation of MYCIN's behaviour, one need to know how that behaviour is structured. MYCIN solve its goal of recommending a therapy for a particular patient by first finding the cause of the patients illness. It uses its production rules to reason backward from goals to clinical observations. Initially, to solve the top-level diagnostic goal it looks for rules whose right sides suggest diseases. Later it uses the left sides of those rules (the preconditions)

to set-up sub-goals whose success would enable the rules to be involved. These sub-goals are again matched against rules and their preconditions are used to set-up additional sub-goals.

Whenever a precondition describes a specific piece of clinical evidence, MYCIN uses that evidence if it already has access to it. Otherwise, it asks the user to provide the information. In order that MYCIN's request for information will appear coherent to the user, the actual goals that MYCIN set-up are often more general than they need to satisfy the preconditions of an individual rule. For example, if a precondition specifies that the identity of an organism is X . MYCIN will set-up the goal 'infer identity'. This approach also means that if another rule mentions the organism's identity no further work will be required, since the identity will be known.

Consider the trace of TEIRESIAS - MYCIN's behaviour as shown in **Fig. 5.4**. The first question that the user asks is a 'WHY' question. Particularly for clinical tests that are either expensive or dangerous, it is important for the doctor to be convinced that the information is really needed before ordering the test. Because MYCIN is reasoning backward the question can easily be answered by examining the goal tree. Doing so provides two kinds of information :

- ❖ What higher level questions might be able to answer if it had the requested piece of information?
- ❖ What other information does the system already have makes it think that the requested piece of knowledge would help? (Here facts 2.1 to 2.4).

When TEIRESIAS provides the answer to the first of these questions, the user may be satisfied or may want to follow the reasoning process back even further. The user can do that by asking additional 'WHY' questions. When TEIRESIAS provides the answer to the second of these questions and tells the user what it already believes, the user may want to know the basis for those beliefs. The user can ask this with a 'How' question also can be answered by looking at the goal tree and chaining backward from the started fact to the evidence that allowed a rule that determined the fact to fire.

The production system model is very general and without restrictions, it is hard to support all kinds of explanations that design artifacts through a propose and revise strategy. SALT is capable of answering questions like WHY-NOT and WHAT-IF.

Is the patient's illness with ORGANISM – 1 a hospital acquired infection ?

WHY ?

[2.0] This will aid in determining the category of ORGANISM – 1. It has already been established that

[2.1] the gram stain of ORGANISM – 1 is gramneg

[2.2] the morphology of ORGANISM – 1 is rod and

[2.3] the aerobicity of ORGANISM – 1 is facultative. Therefore if,

[2.4] the infection with ORGANISM – 1 was not acquired while the patient was hospitalized then, there is strongly suggestive evidence(s) that the category of ORGANISM – 1 is enterobacteriaceae.

[Also : there is weakly suggestion evidence (1) that the identify of ORGANISM – 1 is pseudomonas – aeruginosa]

[RULE 037]

HOW ?

[i.e. How it was established that the aerobicity of ORGANISM – 1 is facultative.

The following were used :

[3.1] RULE 027 indicated that there is strongly suggestive evidence (8) that the aerobicity of ORGANISM – 1 is facultative.

[3.2] RULE 033 indicated there is weakly suggestive evidence (2) that the aerobicity of ORGANISM – 1 is not facultative. Since this gave a cumulative CF of (6) for facultative the evidence indicates that the aerobicity of ORGANISM – 1 is facultative.

Fig. 5.4 A portion of dialogue with TEIRESIAS.

Simple example of explanation

Most expert systems have explanation facilities that allow the user to ask why it asked some question, and how it reached some conclusion. These questions are answered by referring to the system goals, the rules being used and any existing problem data. To illustrate the sorts of facilities commonly provided, we'll elaborate on our simple

example above, starting with a simple example dialogue involving why and how questions and explanations.

System : Is there gas in the fuel tank ?

User: Yes.

System : Is it true that the engine turns over ?

User: No.

System : Is it true that the lights come on ?

User: Why ?

System : I am trying to prove : problem with battery.

It has been established that it is not the case that the engine turns over.
Therefore if it is not the case that the lights come on then there is a problem with battery.

Is it true that the lights come on ?

User: No.

System : I conclude that there is a problem with the battery.

User: How ?

System : This follows from Rule 2 :

IF NOT engine_turns_over AND ...

NOT engine_turns_over was given by the user.

NOT lights_come_on was given by the user.

Providing such an explanation facility involves, at least, recording what rules are used in drawing conclusions and using these records to compose explanations like the ones above.

Giving simple explanations like those above is not very difficult (the answers always have a simple set form), and is sometimes quite useful. However, explanation facilities in expert systems are often not widely used and where used not viewed as acceptable by their users.

There are a whole lot of reasons for this, motivating current research in the area. One reason is that the explanations just reference the "surface" knowledge encoded in the rules, rather than the "deep" knowledge about the domain which originally motivated the rules (but which is usually not represented). So, the system will say that it concluded X because of rule23, but not explain what rule23 is all about. (In the above example, maybe the user needs to understand that both the lights and the starter use the battery, which is the underlying rationale for the second rule in our example). Another stated reason for the frequent failure of explanation facilities is the fact that, if the user fails to understand or accept the explanation, the system can't re-explain in another way (as people can).

Explanation generation is a fairly large (and fascinating) area of research, concerned with effective communication : how do we present things so that people are really satisfied with the explanation and what implications does this have for how we represent the underlying knowledge.

5.5 Knowledge Engineering

Knowledge and its Meaning

EPISTEMOLOGY (theory of knowledge) is the branch of philosophy that studies the nature, methods, limitations, and validity of knowledge and belief. The main problems with which epistemology is concerned are the definition of knowledge and related concepts, the sources and criteria of knowledge, the kinds of knowledge possible and the degree to which each is certain, and the exact relation between the one who knows and the object known. Epistemology primarily addresses the following questions :- "What is knowledge ?", "How is knowledge acquired ?", and "What do people know ?"

❖ Knowledge and Metaknowledge

1. Knowledge is the fact or condition of knowing something with familiarity gained through experience or association.
2. Knowledge is acquaintance with or understanding of a science, art, or technique.

Knowledge is the range of one's information or understanding the circumstance or condition of apprehending truth or fact through reasoning : cognition. The fact or condition of having information or of being learned.

❖ Types of knowledge

- ❖ A priori knowledge comes before knowledge perceived through senses. It is considered to be universally true.
- ❖ A posteriori knowledge which is knowledge verifiable through the senses and may not always be reliable.
- ❖ A procedural knowledge is knowing how to do something.
- ❖ A declarative knowledge is knowing that something is true or false.
- ❖ A tacit knowledge is a knowledge not easily expressed by language.
- ❖ Meta-knowledge : Meta-knowledge is knowledge about knowledge and expertise. Most successful expert systems are restricted to as small a domain as possible. In an expert system, an ontology is the meta-knowledge that describes everything known about the problem domain. Wisdom is the meta-knowledge of determining the best goals of life and how to obtain them.

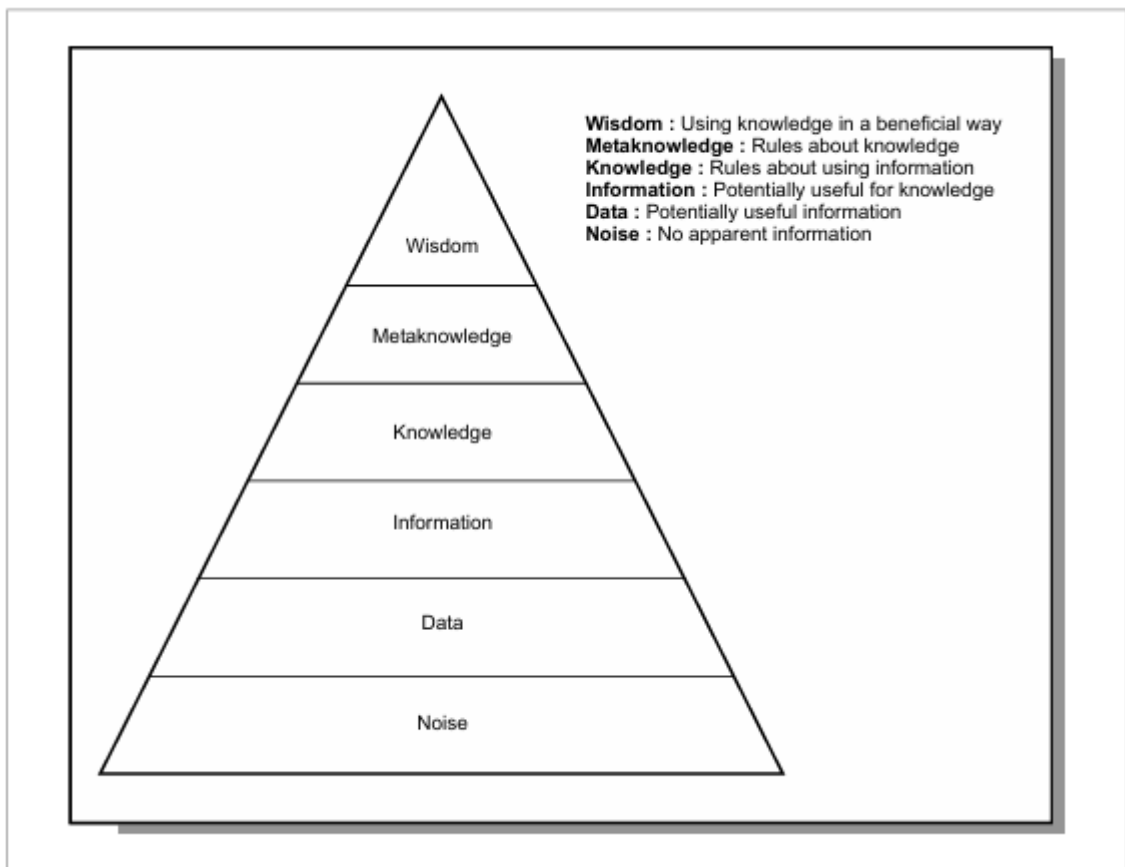


Fig. 5.5 Knowledge of Pyramid.

The following diagram depicts the core elements of knowledge engineering proess.

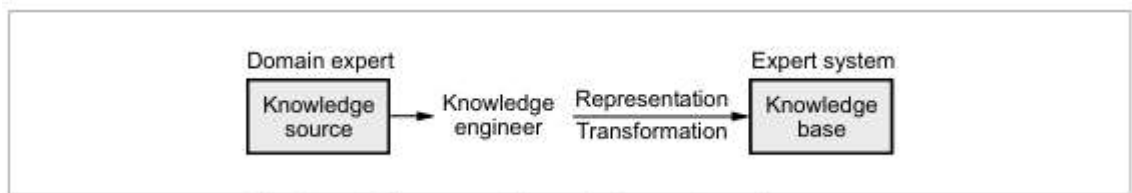


Fig. 5.6 Elements of knowledge Engineering Process.

Knowledge Acquisition

Knowledge Acquisition Process

The knowledge acquisition process begins in the specification phase and continues into the development phase. There are three kinds of cases the developer should discuss with expert which are namely, current cases, historical cases and hypothetical cases.

Artificial Intelligence(AI) with it's Applications

By observing the current cases expert performs the task. Historical cases helps expert to provide knowledge from past experiences and past data. Hypothetical cases serve as dummy situations for expert so as to describe the process and how to carry out the task.

The expert system developer looks for different types of knowledge which depends upon his need. The developer requires strategic knowledge which helps to create flow chart of the system. The judgemental knowledge helps to define the inference process and describes the reasoning process which used by expert. To describe the characteristics and important attributes of the objects in the system factual knowledge is required.

The developer must learn how the expert performs the task knowledge of acquisition in a variety of cases. The knowledge acquisition process, starts in the specification phase, continues into the development phase.

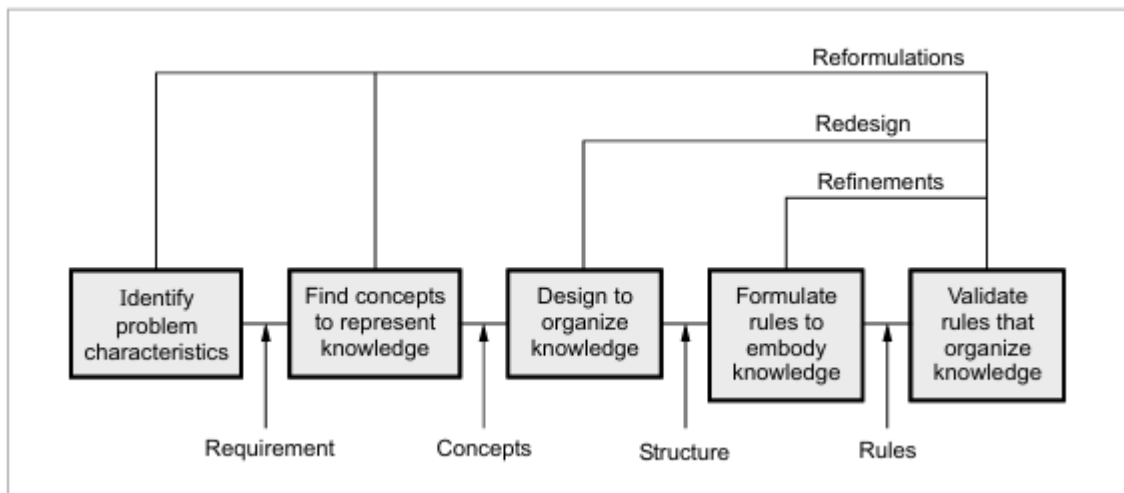


Fig. 5.7 Stages of Knowledge Aquisition.

There are basically three kinds of cases the developer should discuss with the expert : current, historical and hypothetical.

Current : By watching the expert perform a task.

Historical : By discussing with the expert a task that was performed in the past.

Hypothetical : By having the expert describe how a task should be performed in a hypothetical situation.

Knowledge acquisition includes the elicitation, collection, analysis, modeling and validation of knowledge.

Issues in Knowledge Acquisition

The important issues in knowledge acquisition are :

- ❖ knowledge is in the head of experts.
- ❖ Experts have vast amounts of knowledge
- ❖ Experts have a lot of tacit knowledge.

They do not know all that they know and use. Tacit knowledge is hard (impossible) to describe. Experts are very busy and valuable people. Single expert does not know everything. Knowledge has a "shelf life".

5.5.2.2 Knowledge Elicitation

Knowledge elicitation is a type of the knowledge acquisition where the only knowledge source is the domain expert.

Difficulties in Knowledge Elicitation

Technical nature of specialist fields that hinders knowledge elicitation by non-specialist knowledge engineers.

Experts tend to think less in terms of general principles and more in terms of typical objects and commonly occurring events.

Difficulties in searching for a good notation for expressing domain knowledge and a good framework for fitting it all together.

Techniques for knowledge Acquisition

The technique for acquiring, analyzing and modeling knowledge are :

Protocol-generation techniques, Protocol analysis techniques, Hierarchy-generation techniques, Matrix-based techniques, Sorting techniques, Limited-information and constrained-processing task, Diagram-based techniques.

- ❖ Protocol-generation techniques

This method includes many types of interviews (unstructured, semi-structured and structured), reporting and observational techniques.

- ❖ Protocol analysis techniques

This technique is used with transcripts of interviews or text-based information to identify basic knowledge objects within a protocol, such as goals, decisions, relationships and

attributes. These act as a bridge between the use of protocol-based technique and knowledge modeling techniques.

❖ Hierarchy-generation techniques

This technique involves creation, reviewing and modification of hierarchical knowledge. Hierarchy-generation techniques, such as laddering, are used to build taxonomies or other hierarchical structures such as goal trees and decision networks. The ladders are of various forms like concept ladder, attribute ladder, composition ladders.

❖ Matrix-based techniques

This technique has process of construction and filling-in a 2-D matrix (grid, table), indicating such things, as may be, for example, between concepts and properties attributes and values or between problem and solutions or between tasks and resources, etc. The elements within the matrix can contain : symbol (ticks, crosses, question marks), colors, numbers, text.

Mole

MOLE is a knowledge acquisition system which is used for heuristic classification problems, such as diagnosing diseases. It is used in conjunction with the cover and differentiate problem-solving method.

An expert system produced by MOLE accepts input data, generates the set of candidate explanations or classifications that cover the data and then uses differentiating knowledge to determine which one is best. The process is carried out interactively, because explanation needs to be justified, till ultimate causes are confirmed.

MOLE has interface with a domain expert to produce a knowledge - base. A system called MOLE-up (which enhances MOLE performance) is used to solve problems. The acquisition proceeds through following steps :

1. Construction of initial knowledge-base

- ❖ MOLE asks the expert to list common symptoms or complaints which are required for diagnosis. For each symptom, MOLE prompts for a list of possible explanations.
- ❖ Then MOLE iteratively seeks out higher-level explanation until it comes-up with a set of ultimate causes.
- ❖ Whenever an event has multiple explanations MOLE tries to determine the conditions under which one expiation is correct.
- ❖ The expert supplies covering knowledge, which is the knowledge about hypothesized event which might be the cause of a certain symptom. Then

MOLE tries to infer anticipatory knowledge, which says that if the hypothesized event does occur, then the symptom will definitely appear. This knowledge helps the system to rule out certain hypotheses on the basis that specific symptoms are absent.

2. Refinement of knowledge base :

- ❖ In this phase MOLE tries to identify the weakness of the knowledge base. One approach is to find holes (empty places where knowledge is lacking) and prompt the expert to fill them. It is difficult to know whether a knowledge base is complete so instead MOLE lets the expert watch MOLE-P. Solving sample problems, whenever MOLE-P makes an incorrect diagnosis, the expert adds new knowledge.
- ❖ MOLE is used to build systems that diagnose problems with car engines, problems in steel-rolling mills and inefficiency in coal-burning power plants.

However, for using MOLE it must be possible to pre-enumerate solutions or classifications. It should be practical to encode the knowledge in terms of covering and differentiating.

- ❖ Consider example of designing of artifacts of elevator system. It is no longer possible to pre-enumerate all solutions. Instead, we must assign values to a large number of parameters such as the width of the platform, the type of door, the cable weight and the cable strength. These parameters must be consistent with each other and they must result in a design that satisfies external constraints imposed by cost factors, the type of building involved and expected payloads.

A problem-solving method useful for designing such task is called propose and revise. Propose and revise systems builds solutions incrementally. Initially, the system proposes an extension to the current design. Then it checks whether the extension violates any global or local constraints. Constraints violations are then resolved and the process is repeated. Generally it happens that, domain experts are good at listing overall design constraints and at providing local constraints on individual parameters, but they are not so good at explaining how to arrive at global solutions.

SALT

The SALT program provides mechanisms for extracting the knowledge from the expert. Similar to MOLE, SALT builds a dependency network as it has conversation with the expert. Each node stands for a value of a parameter that must be gathered or generated. There are three kinds of links, 'contribute to', 'constraints' and 'suggests revision of'. 'Contribute to' link has associated procedures that allow SALT to generate a value for one parameter based on the value of another. The 'constraints' link rules out certain parameter values. 'Suggests revision of' points to ways in which a constraint violation can be resolved. SALT uses the following heuristics to guide the acquisition process :

1. Every non-input node in the network needs at least one 'contributes to' link coming into it. If links are missing, the expert is prompted to fill the min.
2. No 'contributes to' loops are allowed in the network. Without a value for at least one parameter in the loop, it is impossible to compute values for any parameter in that loop. If a loop exists, SALT tries to transform one of the 'contributes to' links into a 'constraints' link.
3. Constraining links should have suggestion for revision of links associated with them. These include constraints links that are created when dependency loops are broken. The control knowledge is also important in this case. It is critical that the system proposes extensions and revision that lead toward a design solution. SALT allows the expert to rate revisions in terms of how much trouble they tend to produce.

SALT compiles its dependency network into a set of production rules. Like the MOLE system, an expert can watch the production systems, solve problems and can override the systems decision. At that point, the knowledge base can be changed or the override can be logged for future inspection.

The process of interviewing a human expert to extract expertise presents a number of difficulties regardless of whether the interview is conducted by human or by a machine. Experts are surprisingly inarticulate when it comes to how they solve problems. They do not seem to have access to the low level details of what they do. Generally there are inadequate suppliers of any type of statistical information. Therefore, there is a great deal of interest in building systems that automatically induce their own rules by looking at sample problems and solutions with inductive techniques, an expert needs only to provide the conceptual framework for a problem and a set of useful examples.

META-DENDRAL was the first program to use learning techniques to construct rules for an expert system automatically. It built rules to be used by DENDRAL, where job was to determine the structure of complex chemical compounds. META-DENDRAL was able to induce its rules based on a set of mass spectrometry data. It was then able to identify molecular structures with very high accuracy. META-DENDRAL used the version-space learning algorithm. Another popular method used for automatically constructing expert system is decision trees.

Statistical techniques such as multivariate analysis provide an alternative approach for building expert level systems. But statistical methods fail to produce concise rules that humans can understand. Therefore, they lack the ability of explanation.

Knowledgebase (KB) and Building Knowledgebase

KB is important and central part of ES. The main task of KB is to provide the connections between ideas, concepts and statistical probabilities that allow the reasoning part of the system to perform an accurate evaluation of a potential problem. Reasoning can take place only on the basis of available KB .

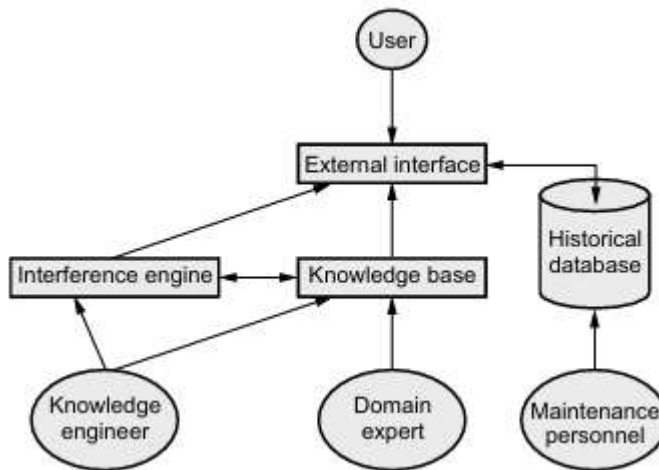


Fig. 5.8 Personnel involved in expert system development.

KB is a large system of "if-then" statements or may contain only associative relationships among different concepts or simply large databases of facts that can be compared to one another, based on simple conventions with respect to the ES. KB contains all the rules and most of the facts.

Knowledge Engineer

The task of the knowledge engineer is to extract the knowledge from the expert and represent it, so that it would be used in expert system. The knowledge engineer should be a expert and well versed in AI language and knowledge representation. Knowledge engineer assists the expert in determining the representation of their knowledge. Knowledge engineer should be able to select a suitable expert system and other tools for the project.

Knowledge engineer should possess the ability to extract the knowledge from the expert and implement the knowledge in a correct and efficient knowledgebase.

Knowledge engineer may not possess the initial domain knowledge of the application. Therefore, to extract the knowledge from expert knowledge engineer should get acquainted with basic domain knowledge.

Knowledge engineer should be perfect and skilled interviewer so as to analysed the problem domain and gather knowledge from expert.

Generally, expert set a series of example problems and will explain their reasoning in solving the problem. The knowledge engineer will abstract general rules from these explanations and check them with the expert. It is up to the knowledge engineer to capture the knowledge of the domain expert into a knowledgebase, which can be then used for an expert system.

Knowledge engineer needs to apply problem-solving techniques so that the process of troubleshooting should also be documented.

Domain Experts

- ❖ Experts are people who have uncommon expertise. To be useful, experts must have other qualities also. They should be able to
- ❖ Recognize and formulate problems.
- ❖ Explain.
- ❖ Organize knowledge (make connections).
- ❖ Determine relevance.
- ❖ Solve problems.
- ❖ We may regard such abilities as desirable in programs or systems which aim to be 'expert'.
- ❖ Ideal expert should possess following characteristics :
- ❖ Highly developed perceptual attention ability - experts can "see" what others cannot.
- ❖ Awareness of the difference between relevant and irrelevant information-experts know how to concentrate on what is important.
- ❖ Ability to simplify complexities - experts can "make sense out of chaos."
- ❖ A strong set of communication skills - experts know how to demonstrate their expertise to others.
- ❖ Knowledge of when to make exceptions - experts know when and when not to follow decision rules.
- ❖ A strong sense of responsibility for their choices - experts are not afraid to stand behind their decisions.
- ❖ Selectivity about which problems to solve - experts know which problems are significant and which are not.
- ❖ Outward confidence in their decision - experts believe in themselves and their abilities.
- ❖ Ability to adapt to changing task conditions - experts avoid rigidity in decision strategies.
- ❖ Highly developed content knowledge about their area - experts know a lot and keep up with the latest developments.
- ❖ Greater automaticity of cognitive processes - experts can do habitually what others have to work at.
- ❖ Ability to tolerate stress - experts can work effectively under adverse conditions.

- ❖ Capability to be more creative - experts are better able to find novel solutions to problems.
- ❖ Inability to articulate their decision processes - experts make decisions "on experience."
- ❖ Through familiarity with the domain, including task expertise built up over a long period of task performance, knowledge of the organizations that will be developing and using the ES, knowledge of the user community and knowledge of technical and technological alternatives.
- ❖ Knowledge and reputation such that if the ES is able to capture a portion of the expert's expertise, the system's output will have credibility and authority.
- ❖ Commitment of a substantial amount of time to the development of the system, including temporary relocation to the development site if necessary.
- ❖ Capability of communicating his or her knowledge, judgment and experience.
- ❖ Cooperative, easy to work with and eager to work on the project.
- ❖ Interest in computer systems, even if he or she is not a computer specialist.

Knowledge Representation

If an artificial agent is supposed to solve some problem related to the real world, it somehow needs to be able to represent knowledge. Facts are represented as data structures that can be manipulated by programs stored inside the computer. Knowledge representation is the process of describing and formally storing the acquired knowledge in certain format so that it can be used fluently in the expert system. The expert knowledge is mapped onto symbols and the meaning is attached to the syntax (rules).

The represented knowledge should possess following characteristics -

- ❖ Transparency
- ❖ Explicitness
- ❖ Naturalness
- ❖ Efficiency
- ❖ Adequacy
- ❖ Modularity

Knowledge representation methods

1. **Production rules** - It is frequently used method to formulate the knowledge in expert systems. A formal variation of this methodology is Backus-Naur Form (BNF). BNF comprises following -

- ❖ Meta-language for the definition of language syntax.
- ❖ A grammar is a complete, unambiguous set of production rules for a specific language.
- ❖ A parse tree is a graphic representation of a sentence in that language.

BNF provides only a syntactic description of the language therefore some sentences may not make sense.

Advantages of production rules

- ❖ They are simple and easy to understand.
- ❖ Grammars have straightforward implementation in computers.
- ❖ They are formal foundations for some other variants.

Problems associated with production rules

- ❖ Simple implementations of production rules are very inefficient.
- ❖ Some types of knowledge are not easily expressed in such rules.
- ❖ Large sets of rules become difficult to understand and maintain in the database.

2. Semantic Nets - A semantic network uses directed graphs to represent knowledge. A directed graph is made of vertices (nodes) and edges (arcs). A concept can be thought of as a set or a subset. For example, animal defines the set of all animals, horse defines the set of all horses and is a subset of the set animal. In a semantic network, relations are shown by edges. An edge can define a subclass relation, an instance relation attribute, an object (color, size, ...), or a property of an object. Semantic nets are classic representation technique for propositional information. Propositions are a form of declarative knowledge, stating facts (true/false). Propositions are called "atoms" which cannot be further subdivided. Semantic nets consist of nodes (objects, concepts, situations) and arcs (relationship between them).

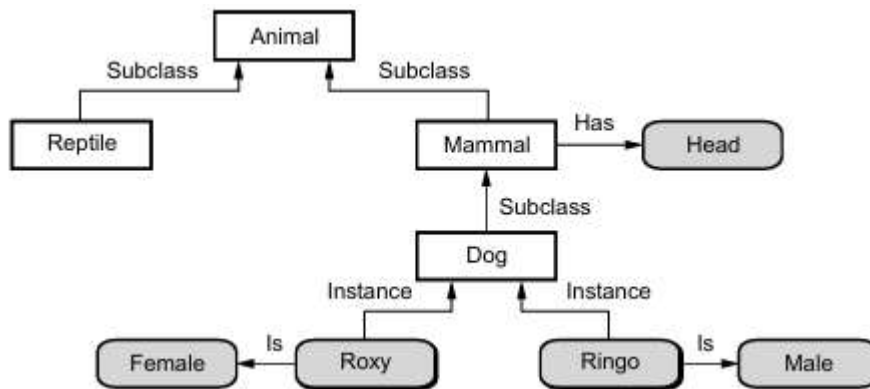


Fig. 5.9 Semantic Net.

Common types of links that are used in semantic nets are, IS-A - relates an instance or individual to a generic class and A-KIND-OF - relates generic nodes.

Problems associated with Semantic nets

- ❖ To represent definitive knowledge, the link and nodes names must be rigorously defined.
- ❖ A solution to this is extensible markup language (XML) and ontologies.
- ❖ Problems also include combinatorial explosion of searching nodes, inability to define knowledge the way logic can, and heuristic inadequacy.

3. Schemata - It is a knowledge structure which is an ordered collection of knowledge - not just data. Semantic Nets are shallow knowledge structure - all knowledge is contained in nodes and links. Schema is a more complex knowledge structure than a semantic net. In a schema, a node is like a record which may contain data, records, and/or pointers to nodes.

4. Frames - One type of schema is a frame (or script - time-ordered sequence of frames). Frames are useful for simulating common sense knowledge. Semantic nets provide 2-dimensional knowledge whereas the frames provide 3-dimensional. Frames represent related knowledge about narrow subjects having much default knowledge.

A frame is a group of slots and fillers that defines a stereotypical object that is used to represent generic/specific knowledge. Common sense knowledge is knowledge that is generally known. Prototypes are objects possessing all typical characteristic of whatever is being modeled.

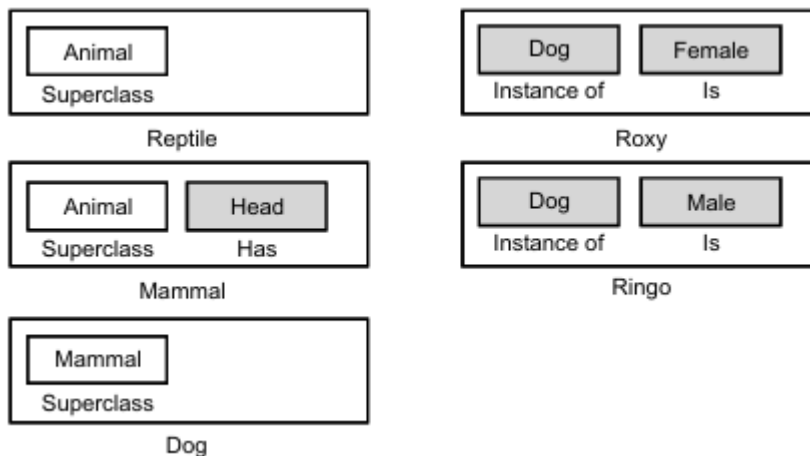


Fig. 5.10 Frame.

A node in a semantic network becomes an object in a set of frames, so an object can define a class, a subclass or an instance of a class. Edges in semantic networks are translated into slots-fields in the data structure. The name of the slot defines the type of the relationship and the value of the slot completes the relationship.

Problems with frames include allowing a unrestrained alteration or cancellation of slots.

5. Logic - Knowledge can also be represented by symbols of logic. Logic is the study of rules of exact reasoning - inferring conclusions from premises. Automated reasoning is the logic programming in the context of expert systems.

Earliest form of logic was based on the syllogism - developed by Aristotle. Syllogisms - have two premises that provide evidence to support a conclusion.

Example :

Premise : All cats are climbers.

Premise : Garfield is a cat.

Conclusion: Garfield is a climber.

The most common knowledge representation is predicate logic. Predicate logic can be used to represent complex facts. It is a well-defined language developed via a long history of theoretical logic.

Propositional logic is a language made up from a set sentences that can be used to carry out logical reasoning about the world. Propositional logic uses five operators which are $\wedge$ (AND), $\vee$ (OR), $\sim$ (Negation), $\rightarrow$ (IMPLICATION), $\leftrightarrow$ (DOUBLE IMPLICATION).

Inference Engine

The inference engine is the main processing element of the expert system. It is the software that provide the reasoning mechanism in an expert system. The rule based expert system, typically implements forward chaining strategies. The inference engine chooses rules from the agenda to fire. If there are no rules on the agenda, the inference engine must obtain information from the user in order to add more rules to the agenda. It makes use of knowledge base, in order to draw conclusion for situations. It is responsible for gathering the information from the user, by asking various questions and applying it wherever necessary.

The most crucial property of expert system is their capacity to make inferences or the drawing of conclusions from premises. This is precisely what makes an expert system intelligent.

Inferencing is to computers what reasoning is to humans. A computer expert system would need to decide which, and in what order, the rules should be selected for evaluation. To do this, an expert system uses an inference engine. The inference engine control overall execution of the rule. It searches through the knowledgebase, in order to draw conclusions. If a rule's antecedent is satisfied, the rule is ready to fire and placed in the agenda. When a rule is ready to fire it means that since the antecedent is satisfied, the consequent can be executed.

Following are the examples of IF ... THEN Rule

Example 1

Rule : Red_Light
IF the light is red
THEN stop

Rule : Yellow_Light
IF the light is yellow

THEN look

Rule : Green_Light
IF the light is green
THEN go

Example 2

A typical rule used by the MYCIN expert system is :-

IF the stain of the organism is gram negative

AND the morphology of the organism is rod

AND the aerobicity of the organism is anaerobic

THEN there is strongly suggestive evidence (0.8) that the class of the organism is ENTEROBACTERIACEAE

MYCIN Format

IF (AND (SAME CNTEXT GRAM GRAMNEG)

(SAME CNTEXT MORPH ROD)

(SAME CNTEXT AIR AEROBIC)

THEN (COCLUDE CNTEXT CLASS ENTEROBACTERIACEAE TALLY .8))

Two alternative strategies are available for inferencing which are namely Forward chaining and Backward chaining. A particular inference engine may adopt either or both. Forward chaining (data driven) and Backward chaining (hypothesis driven) techniques represent the fundamental reasoning approaches implemented in rule-based expert systems. In forward chaining, the expert system is given data and chains forward to reach a conclusion. Forward chaining systems are data-driven. In order to execute a rule-based expert system using the method of forward chaining we merely need to fire (or execute) actions whenever they appear on the action list of a rule whose conditions are true. This involves assigning values to attributes, evaluating conditions, and checking to see if all of the conditions in a rule are satisfied.

A general algorithm for this might be :

- ❖ While values for attributes remain to be input read value and assign to attribute.
- ❖ Evaluate conditions
- ❖ Fire rules whose conditions are satisfied

Several points about this require consideration. First, some conflict resolution strategy needs to be employed in order to decide which rules are fired first. Our method is to fire the rule which the system designer defined first. Also, we wish to cut down on computational time. To do this we must not do anything which does not absolutely need to be done. This means that conditions are only evaluated at the time they might change and that rules are checked (to see if all of their conditions are satisfied) only when they might be ready to be fired, not before. We shall do this, as attributes are assigned values and shall only consider rules and conditions affected by the new attribute assignment.

In backward chaining, the expert system is given a hypothesis and backtracks to check if it is valid. Backward chaining systems are generally goal-driven or 'goal-orientated' in the sense that it tries to prove a goal or rule conclusions by confirming the truth of all it's premises. Backward chaining starts with a list of goals (or hypothesis) and works backwards to see if there are data available that will support any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a Then clause that matches a desired goal. If the If clause of that inference rule is not known to be true, then it is added to the list of goals (in order for your goal to be confirmed you must also provide data that confirms this new rule).

Inference Engine

The inference engine is the main processing element of the expert system. It is the software that provide the reasoning mechanism in an expert system. The rule based expert system, typically implements forward chaining strategies. The inference engine chooses rules from the agenda to fire. If there are no rules on the agenda, the inference engine must obtain information from the user in order to add more rules to the agenda. It makes use of knowledge base, in order to draw conclusion for situations. It is responsible for gathering the information from the user, by asking various questions and applying it wherever necessary.

The most crucial property of expert system is their capacity to make inferences or the drawing of conclusions from premises. This is precisely what makes an expert system intelligent.

Inferencing is to computers what reasoning is to humans. A computer expert system would need to decide which, and in what order, the rules should be selected for evaluation. To do this, an expert system uses an inference engine. The inference engine control overall execution of the rule. It searches through the knowledgebase, in order to draw conclusions. If a rule's antecedent is satisfied, the rule is ready to fire and placed in the agenda. When a rule is ready to fire it means that since the antecedent is satisfied, the consequent can be executed.

Following are the examples of IF ... THEN Rule

Example 1

Rule : Red_Light
IF the light is red
THEN stop

Rule : Yellow_Light
IF the light is yellow

THEN look

Rule : Green_Light
IF the light is green
THEN go

Example 2

A typical rule used by the MYCIN expert system is :-

IF the stain of the organism is gram negative

AND the morphology of the organism is rod

AND the aerobicity of the organism is anaerobic

THEN there is strongly suggestive evidence (0.8) that the class of the organism is ENTEROBACTERIACEAE

MYCIN Format

IF (AND (SAME CNTEXT GRAM GRAMNEG)

(SAME CNTEXT MORPH ROD)

(SAME CNTEXT AIR AEROBIC)

THEN (COCLUDE CNTEXT CLASS ENTEROBACTERIACEAE TALLY .8))

Two alternative strategies are available for inferencing which are namely Forward chaining and Backward chaining. A particular inference engine may adopt either or both. Forward chaining (data driven) and Backward chaining (hypothesis driven) techniques represent the fundamental reasoning approaches implemented in rule-based expert systems. In forward chaining, the expert system is given data and chains forward to reach a conclusion. Forward chaining systems are data-driven. In order to execute a rule-based expert system using the method of forward chaining we merely need to fire (or execute) actions whenever they appear on the action list of a rule whose conditions are true. This involves assigning values to attributes, evaluating conditions, and checking to see if all of the conditions in a rule are satisfied.

A general algorithm for this might be :

- ❖ While values for attributes remain to be input read value and assign to attribute.
- ❖ Evaluate conditions
- ❖ Fire rules whose conditions are satisfied

Several points about this require consideration. First, some conflict resolution strategy needs to be employed in order to decide which rules are fired first. Our method is to fire the rule which the system designer defined first. Also, we wish to cut down on computational time. To do this we must not do anything which does not absolutely need to be done. This means that conditions are only evaluated at the time they might change and that rules are checked (to see if all of their conditions are satisfied) only when they might be ready to be fired, not before. We shall do this, as attributes are assigned values and shall only consider rules and conditions affected by the new attribute assignment.

In backward chaining, the expert system is given a hypothesis and backtracks to check if it is valid. Backward chaining systems are generally goal-driven or 'goal-orientated' in the sense that it tries to prove a goal or rule conclusions by confirming the truth of all it's premises. Backward chaining starts with a list of goals (or hypothesis) and works backwards to see if there are data available that will support any of these goals. An inference engine using backward chaining would search the inference rules until it finds one which has a Then clause that matches a desired goal. If the If clause of that inference rule is not known to be true, then it is added to the list of goals (in order for your goal to be confirmed you must also provide data that confirms this new rule).

5.6 Building Expert System

Developing an Expert System

There are following ten basic steps in developing any expert system

- 1. Identify the problem and need :** Like many computer programs, expert systems are, in a sense a solution looking for a problem. To justify the creation of an expert systems, there must be a real problem to solve or need to meet. For that reason, your first step in developing an expert system should be to examine your situation and clearly decide what the problem is or why such a system may be helpful. In attempting to achieve results, we often run into problems : time problems, productivity problems, or people problems.
- 2. Determine the suitability of the problem :** Once the problem is clearly identified, you can then examine it in more detail to see if it is right for an expert system solution.
- 3. Consider the alternatives :** Although your problem may fit the criterion for an expert system, don't overlook what might be simpler or equally as satisfactory alternative solutions. For example, certain kinds of employee performance problems might be corrected by training. Another solution is providing all employees with the information they need in a written manual or job aid. A non-computers solution may be not only the best solution, but also the simplest and least expensive. Consider other software alternatives. A database management system (DBMS) software package might better fit the problem than an expert system, by letting you store a significant amount of information and access it conveniently. The ability of the computer to reason may not be necessary. Finally, don't overlook a conventional software solution. A straightforward algorithmic solution might be better than a symbolic or AI solution. Having now looked at the alternatives, you can make your final decision. If an expert system is your choice, then you can begin the next steps.
- 4. Compute the return-on-investment :** If your choice is an expert system, your next step is to determine whether or not it is economically feasible. You must compute the Return On your Investment (ROI) by performing a cost/benefit analysis. Developing an expert system is not a simple job. It is going to cost a considerable amount not only in the purchase of software but in the hours it will take to create the system. Personnel must be available to do the work, and even the simplest of expert systems will take months to develop and cost a lot of money. Can the time and expense be justified?, is the problem to be solved, bad enough or important enough to make the investment.
- 5. Select a development tool :** An expert system development tool is a software package that allows you to enter an expert's knowledge into the computer without having to program. Most expert system development tools are rule based. Some tools permit the implementation of frames and semantic networks, but they are usually more expensive

and generally operate on larger computer.

- 6. Perform the knowledge engineering :** At this point in the development process, you are finally ready to do some real creative work. Development of an expert system begins with knowledge engineering : that is, acquiring the knowledge. As you know, knowledge comes in many forms. It can be standard textbook knowledge that you can dig out of books, articles, and other references quickly and easily. This knowledge is important, but it is usually not the best kind of knowledge for an expert system. The real knowledge will come from individuals who are experts in the subject.
- 7. Design the System :** Using the knowledge you have acquired and the tool you selected, you can now begin the design of the expert system. First, you will need to have create an outline, a hierarchical flow chart, a matrix, decision table, or other format that will help you organize and understand the knowledge. Using these aids, you will convert the knowledge in to IF-THEN rules. It is best to follow the specific procedures recommended by the software tool you are using. Once the basic design is complete, you can begin using the tool to create a prototype of one segment of the system. Translate a portion of the knowledge into rules and test the newly created segment. Your job here is to test the concept before going ahead with the entire program.
- 8. Complete the development :** Once you have satisfied yourself that the system is going to work satisfactorily you can begin to expand the prototype into the final system. The best way to go about this is to expand the prototype one segment at a time.
- 9. Test and debug the system :** After the expert system has been developed, you will need to spend sometime to testing and debugging it. No expert system will be perfect the first time, and a considerable amount of work will be required to validate it. User feedback will show you where to make final changes, corrections, and additions to achieve the desired performance.
- 10. Maintain the system :** An important part of expert system development is ongoing, maintenance, updating the system with new knowledge, removing knowledge is no longer applicable, and otherwise fine tune the system to keep it fully current and applicable to the problem.

Early Expert System

Pioneer work			
1955	Herbert Simon, Allen Newell	Carnegie Mellon	Logic Theorist : Proved theorems using propositional logic
1960	Marvin Minsky John McCarthy Claude Shannon	MIT Dartmouth Bell Labs	LISP
1960	DENDRAL	Feigenbaum & Buchanan (Stanford)	Identify chemical constituents
1970	MYCIN	Stanford	Diagnosis of infectious diseases
1970	MACSYMA	MIT	Math expert system
1970	HEARSAY	Carnegie Mellon	Speech recognition
1977	PROSPECTOR	Stanford Research Inst Duda, Hart, Barnett	Mineral diagnosis
1978	XCON (Earlier called R1)	McDermott	Automated computer configuration system
1981	DART	Bennett and Hollander	Diagnoses fault in the hardware of computer system

Table. 5.1 Early Expert System.

5.7 Expert System Case Study

5.7.1 MYCIN Case Study

MYCIN is one of the earliest expert systems, and its design has strongly influenced the design of commercial expert systems and expert system shells. Mycin was an expert system developed at Stanford in the 1970s. Its job was to diagnose and recommend treatment for certain blood infections. To do the diagnosis "properly" involves growing cultures of the infecting organism. Unfortunately this takes around 48 hours, and if doctors waited until this was complete their patient might be dead! So, doctors have to come up with quick guesses about likely problems from the available data, and use these guesses to provide a "covering" treatment where drugs are given which should deal with any possible problem.

MYCIN was developed partly in order to explore how human experts make these rough (but important) guesses based on partial information. However, the problem is also a potentially importance in practical terms - there are lots, if junior or non-specialized doctors, who sometimes have to make such a rough diagnoses and if there is an expert tool available to help them then this might allow more effective treatment to be given. In fact, MYCIN was never actually used in practice. This wasn't because of any weakness in its performance - in tests it outperformed members of the Stanford medical school. It was as much because of ethical and legal issues related to the use of computers in medicine - if it gives the wrong diagnosis, who do you sue ?

MYCIN represented its knowledge as a set of IF-THEN rules with certainty factors. The following is an English version of one of MYCIN'S rules :

IF the infection is pimary-bacteremia

AND the site of the culture is one of the sterile sites

AND the suspected portal of entry is the gastrointestinal tract

THEN there is suggestive evidence (0.7) that infection is bacteroid.

The 0.7 is roughly the certainty that the conclusion will be true given the evidence. If the evidence is uncertain the certainties of the bits of evidence will be combined with the certainty of the rule to give the certainty of the conclusion.

MYCIN was written in Lisp, and its rules are formally represented as Lisp expressions. The action part of the rule could just be a conclusion an out the problem being solved, or it could be an arbitrary lisp expression. This allowed great flexibility, but removed some of the modularity and clarity of rule-based systems, so using this fascility had to be used with care.

MYCIN is a (primarily) goal-directed system, using the basic backward chaining reasoning strategy that we described above. However, MYCIN used various heuristics to control the search for a solution (or proof of some hypothesis). These were needed both to make the reasoning efficient and to prevent the user being asked too many unnecessary questions.

The other strategies relate to the way in which rules are invoked. The first one is simple : given a possible rule to use, MYCIN first checks all the premises of the rule to

see if any known to be false. If so there's not much point using the rule. The other strategies relate more to the certainty factors. MYCIN will first look at rules that have more certain conclusions, and will abandon a search once the certainties get below 0.2.

There were many other developments from the MYCIN project. For example, EMYCIN was really the first expert shell developed from MYCIN. A new expert system called PUFF was developed using EMYCIN in the new domain of heart disorders. And system called NEOMYCIN was developed for training doctors, which would take them through various example cases, checking their conclusions and explaining where they went wrong.

5.7.2 Prospector

PROSPECTOR aids geologists in their search for ore deposits. Given the field data about a geological region, it can determine the probability of discovering a range of ore deposits, including sandstone uranium, massive sulfide, carbonate lead/zinc etc.

Prospector is a consultation system to assists geologists working in mineral exploration. It is being developed by Hart and Duda of SRI international. This system attempts to represents to the knowledge and reasoning processes of experts in the geological domain. The user of this system is an exploration geologist in early stages of investigating a possible drilling site. Its expertise is based on geological rules which form models of ore deposits, and a database of known rocks and minerals. In 1980, PROSPECTOR analyzed geological data from a site near Mount Tolman in eastern Washington and predicted the existence of molybdenum (an ore) in a particular location. Subsequent drilling by a mining company **confirmed PROSPECTOR's prediction.**

Working of Prospector

- a) First, the geologists working on the prospector provide certain conclusions based on their study consisting of information like geologists setting structural controls, and kinds of rocks minerals and alteration products present or suspected.
- b) PROSPECTOR compares observations with stored models of core deposits.
- c) It then notes the similarities, differences and missing information.
- d) Prospector then assesses the potential of the prospect.

Features of Prospector

1. This ES talks to the geological expert in terms which are familiar to him.
2. It allows the user to see the information of the region at any given point of time during the consultation.
3. It chooses the hypothesis depending on core deposits and then asks the user relevant questions. Depending on the answer given by the user it can also change the hypothesis.

4. It checks for the inconsistencies in the answer and allows the user to change the answer if required.
5. It represents its conclusion and summary, explaining how it had reached the conclusion.

5.7.3 XCON / R1

The XCON system

The R1 (internally called XCON, for eXpert CONfigurer) program was a production-rule-based system written in OPS5 by John P. McDermott of CMU in 1978 to assist in the ordering of DEC's VAX computer systems by automatically selecting the computer system components based on the customer's requirements.

It is one of the most cited expert systems. It was developed by DEC (Digital Equipment Corporation) and was a system that ensured the customer was supplied with all the components and software that was required to make up the specified computer system that they had ordered.

Need of XCON

- ❖ DEC, Digital Equipment Corporation is a large computer manufacturer, started in 1957. Its catalogue has 40,000 different parts.
- ❖ Buyer (with Sales representatives) sends order, typically 100 parts in one order or so.
- ❖ The delivery and assembly was done by DEC personnel. Too often, the part collection does not allow installation and it was also found that installed computer does not meet requirements.
- ❖ The solution was to have a completely assembled and test system in factory. There was need of automated configuration system but early attempts with procedural languages were unsuccessful.
- ❖ XS approach started around 1980.

Significance

Building a bespoke computer system means that every system is different and, as such, needs different components and software. For example, if a customer orders a disc drive then DEC has to ensure that the customer also receives the disc controller and the relevant cables. As a single system can be built from thousands of separate components, it was not easy for a human to ensure that the customer had everything he/she needed to build the system. XCON/R1 was an expert system that did this job automatically. It saved DEC's millions of dollars every year by speeding the assembly process and increasing customer satisfaction. It raised the profile of expert system to a level that had not been seen before.

The data Of XCON

XCON has five data bases,

1. Components: Holds 30k parts, 25 to 125 attributes
2. Container : Extra data on backplanes and cabinets, size and slots.
3. Diagrams : 1200 ASCII arts for use in output diagrams
4. Questions : Motivate and ask data from user, eg. Do you already have ...?
5. Configuration : Consultation data like input and diagrams

R1's Evolution

In a period of less than a year, R1 went from an idea, to a demonstration system that had most of the basic knowledge required in the domain but lacked the ability to deal with complex orders, to a system that possesses true expertise. Its development was done in parallels, in many respects, in terms of the development of the several domain-specific systems engineered by Stanford University's Heuristic Programming Project. R1's implementation history divides basically into two stages. During the first stage, which began in December of 1978 and lasted for about four months, much of the time was spent in, read and reread the manuals that describe many of the VAX configuration constraints.

The second stage, which lasted for another four months, was spent in asking people who were expert in the VAX configuration task to examine R1's output, point out R1's mistakes, and indicate what knowledge R1 was lacking. R1 was sufficiently ignorant that finding mistakes was no problem. Given a criticism of some aspect of the configuration by an expert, all that was necessary in order to refine R1's knowledge was to find the offending rule, ask the expert to point out the problem with the condition elements in the rule, and then either modify the rule or split it into two rules that would discriminate between two previously undifferentiated states. During this stage, R1's domain knowledge got almost tripled.

5.7.4 DART

DART (Dynamic Analysis and Replanning Tool)

1. DART assists a technician in finding the faults in a computer system. (hardware and software).
2. DART is a joint project of the Stanford University and IBM that explores the application of artificial intelligence techniques to the diagnosis of computer faults.
3. DART uses a device-independent language for describing devices and device-independent inference procedure for diagnosis.
4. The primary goal of the DART Project is to develop programs that capture the special design knowledge and diagnostic abilities of these experts and to make them available to field engineers.
5. The practical goal of DART is the construction of an automated diagnostician capable of pinpointing the functional units responsible for observed malfunctions in arbitrary system configurations.

3. Personal Digital Assistant Expert System (PDAMUM)

- ❖ It is a system that help people to manage their devices as per need.
- ❖ PDA provides descriptive explanation that guide the user to recognize their PDA's problem and then provide a appropriate solution based on the accurate diagnosis.
- ❖ PDAMUM help to recognize the synchronization problem with the devices.
- ❖ PDAMUM uses several short questions and with the answers it reaches to conclusion with reasoning.
- ❖ PDAMUM responds fast to user input and generates a reliable diagnosis report.
- ❖ PDAMUM performs knowledge acquisition process in a cyclic manner.
- ❖ PDAMUM uses rule technique to code the knowledge in the knowledgebase. A rule is an IF/THEN structure that logically relates information contained in the IF part to other information contained in the THEN part.
- ❖ Following are the rules used in PDAMUM.

Rule 1

IF Cradle connects with power supply
AND Electrical source available
THEN Cradle is activated

Rule 2

IF PDA's battery power available
THEN PDA is working

Rule 3

IF PDA is working
THEN PDA's PIM activates

Rule 4

IF USB connector connects with PC
THEN PDA link with PC

Rule 5

IF Install Companion CD
THEN Active Sync available

Rule 6

IF Partnership Mode Menu available
AND Partnership Menu select
THEN Partnership Mode define

Rule 7

IF Partnership Mode define
_ THEN Partnership Mode activate

Rule 8

IF Install synchronization driver
THEN PDA communicates with PC

Rule 9

IF Cradle is activates
AND PDA's PIM activates
AND Active Sync available
AND PDA link with PC
AND PDA communicates with PC
AND Partnership Mode activate
THEN Active Sync performs synchronization
PDAMUM is more consistent than human. It may generate erreneous report but better as compare to human.

Expert system for travel recommendation and discuss its role

◆ Role of Travel recommendation expert system

1. The application of expert systems in tourism industry is no exception and they play a vital role in accumulation, synthesis and dissemination of knowledge.
2. Expert systems are knowledge-based system to embody expertise in a particular domain. The process of building a knowledge base in a particular domain helps to relieve man power and experts from answering routine questions which a computer based expert system can easily handle.
3. Expert systems are the most prominent product of Artificial Intelligence technology and they appear particularly suited for service industries such as tourism.
4. Travel recommendation expert system includes the features of both the travel router : An intelligent routing module for the tourism and personalization travel support agent.

5. Travel Router is an intelligent routing module designed for assisting the tourist in finding the shortest possible land route to reach the intended destination assisting the tourist in finding the shortest possible land route to reach the intended destination. By combining the features of both the travel router: An intelligent routing module for the tourism and personalization travel support system, one can give a more effective approach for the management of various tourist information.

◆ Design for expert system for travel recommendation

1. The expert system is built to recommend a suitable travel schedule that satisfies user input constraints such as time period, budget and preferences.

Tourist center managers need to answer similar set of queries in their day-to-day work which could be replaced by an expert system for tourist information management. There are many different tourist information such as activities and places which can be stored as similar data structure. The proposed knowledge structure would be flexible enough to cope with high volatility in some tourist information such as transportation routes.

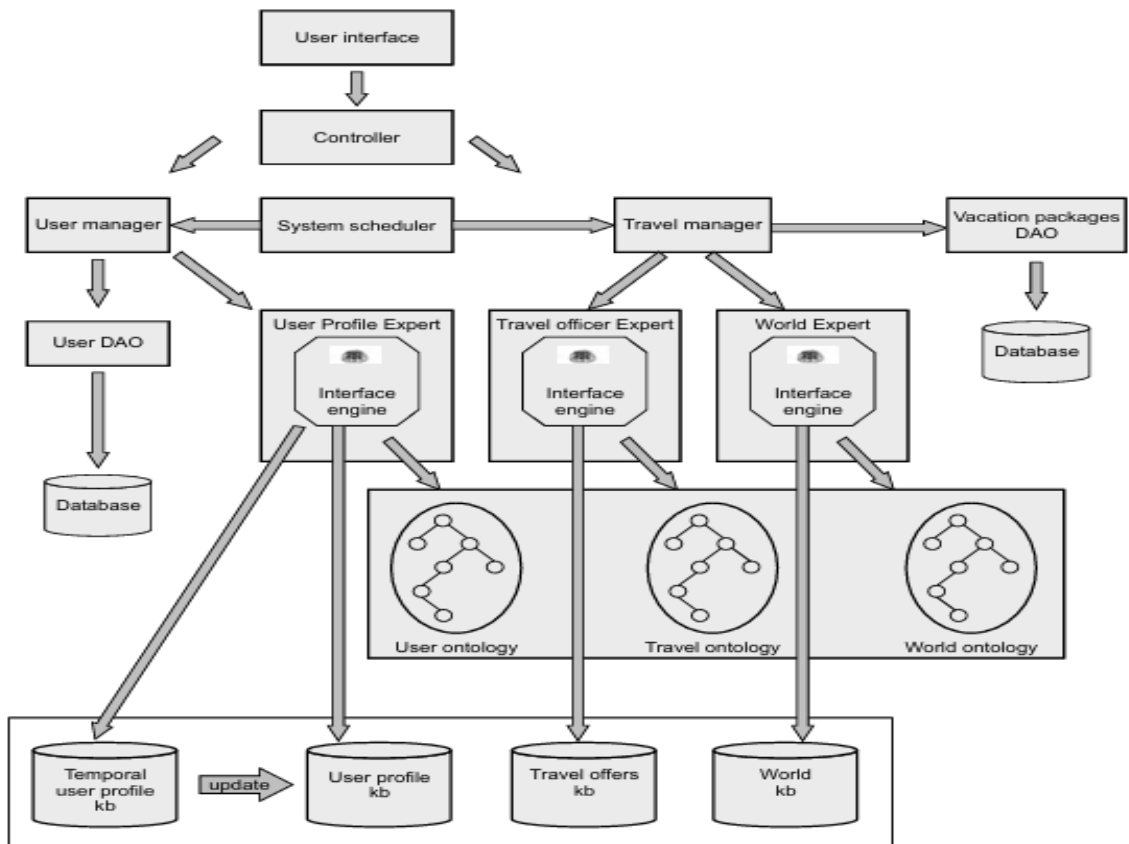


Fig. 5.11 Travel recommendation expert system architecture (kb is knowledge base).

2. Travel recommendation includes intelligent components that perform reasoning and have some built-in heuristics. It is based on a number of ontologies, which support the process of adding semantics to data and making them machine readable and understandable. In other words, Travel Guides illustrates how generating and using machine understandable data can

- ❖ Offer tourists a better service,
- ❖ Decrease maintenance efforts of tourist agents, and
- ❖ Increase interoperability of online tourism applications.

Ontologies enable semantic description of data and more importantly - reasoning. Specifically, in Travel recommendation system, ontologies enable inferring of user profile types based on user interests and activities. Likely, destination types are inferred based on the destinations data. During process of search, the destination types are matched with the relevant user profiles. Searching for a perfect destination in Travel recommendation is performed using a very simple form that is part of the Travel recommendation environment.

3. The expert system structure includes the followings

1. **Personalization Learner:** Is the process of learning and analyzing of user behavior to understand user's interest.
2. **Personalization Ranking:** Its function is to rank the trip information for the users. The work process is based on the initial weight of learning and the user's interests on each trip.
3. **User Profile Database:** This is the database of users, which is operated for travel management depending on the user's behaviors, the database will be processed in mapping the trip list to the user's requirements. Profile database is categorized into two types : User's properties data and User's behavior.
4. **Trip Features** : Trip features associate to user interests in tourist programs, they are as follows:
 - a) Trip Duration is numbers of days offering by each trip.
 - b) Trip Categories is type of trip including shopping, eco tour, and scuba diving and trekking etc.
 - c) Trip Lowest Price is the lowest prices for trip expenses.
 - d) Trip Highest Price is the lowest prices for trip expenses.
 - e) Trip Destination is the country of visitation.

CHAPTER-6

Applications of AI in Various Fields

6.1 Heart Disease Prediction Using Logistic Regression

Importing Necessary Libraries

```
import pandas as pd
import pylab as pl
import numpy as np
import scipy.optimize as opt
import statsmodels.api as sm
from sklearn import preprocessing
'exec(% matplotlib inline)'
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import seaborn as sns
```

Data Preparation

The dataset is from an ongoing cardiovascular study on residents of the town of Framingham, Massachusetts. The classification goal is to predict whether the patient has 10-year risk of future coronary heart disease (CHD). The dataset provides the patients' information. It includes over 4,000 records and 15 attributes.

Loading the dataset (save file name as sample_data.csv)

```
# dataset
disease_df = pd.read_csv("sample_data.csv")
disease_df.drop(['education'], inplace = True, axis = 1)
disease_df.rename(columns = {'male':'Sex_male'}, inplace = True)
```

Handling Missing Values

```
# removing NaN / NULL values  
disease_df.dropna(axis = 0, inplace = True)  
print(disease_df.head(), disease_df.shape)  
print(disease_df.TenYearCHD.value_counts())
```

Output

Sex_male	age	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp
1	39	0	0.0	0.0	0	0
0	46	0	0.0	0.0	0	0
1	48	1	20.0	0.0	0	0
0	46	1	23.0	0.0	0	0
0	61	1	30.0	0.0	0	1

diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
0	195.0	106.0	70.0	26.97	80.0	77.0	0
0	250.0	121.0	81.0	28.73	95.0	76.0	0
0	245.0	127.5	80.0	25.34	75.0	70.0	0
0	285.0	130.0	84.0	23.10	85.0	85.0	0
0	225.0	150.0	95.0	28.58	65.0	103.0	1

Splitting the Dataset into Test and Train Sets

```
X = np.asarray(disease_df[['age', 'Sex_male', 'cigsPerDay',  
'totChol', 'sysBP', 'glucose']])  
y = np.asarray(disease_df['TenYearCHD'])  
  
# normalization of the dataset  
  
X = preprocessing.StandardScaler().fit(X).transform(X)
```

```
# Train-and-Test -Split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size = 0.3, random_state = 4)

print ("Train set:", X_train.shape, y_train.shape)

print ("Test set:", X_test.shape, y_test.shape)
```

Output

Train set: (2625, 6) (2625,)

Test set: (1126, 6) (1126,)

Exploratory Data Analysis of Heart Disease Dataset

Ten Year's CHD Record of all the patients available in the dataset:

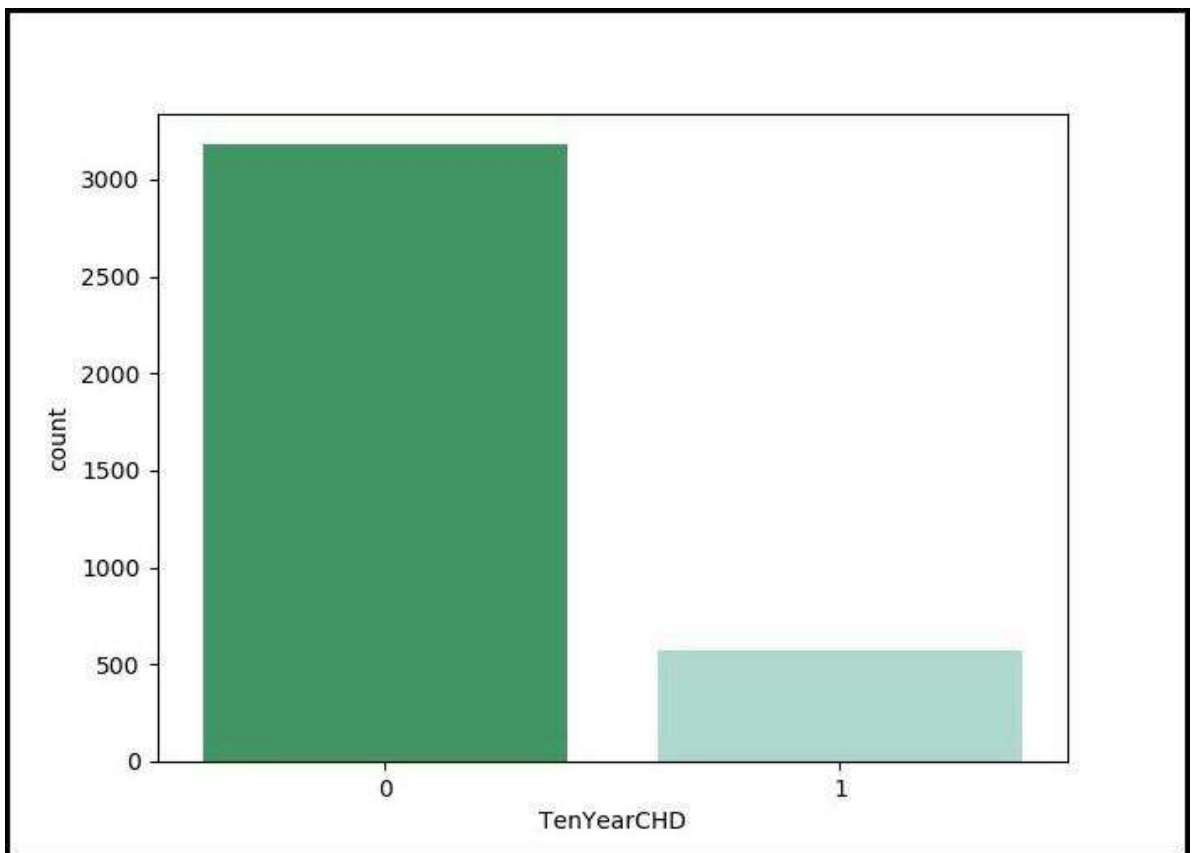
```
# counting no. of patients affected with CHD

plt.figure(figsize=(7, 5))

sns.countplot(x="TenYearCHD", data=disease_df,
              palette="BuGn_r")

plt.show()
```

Output

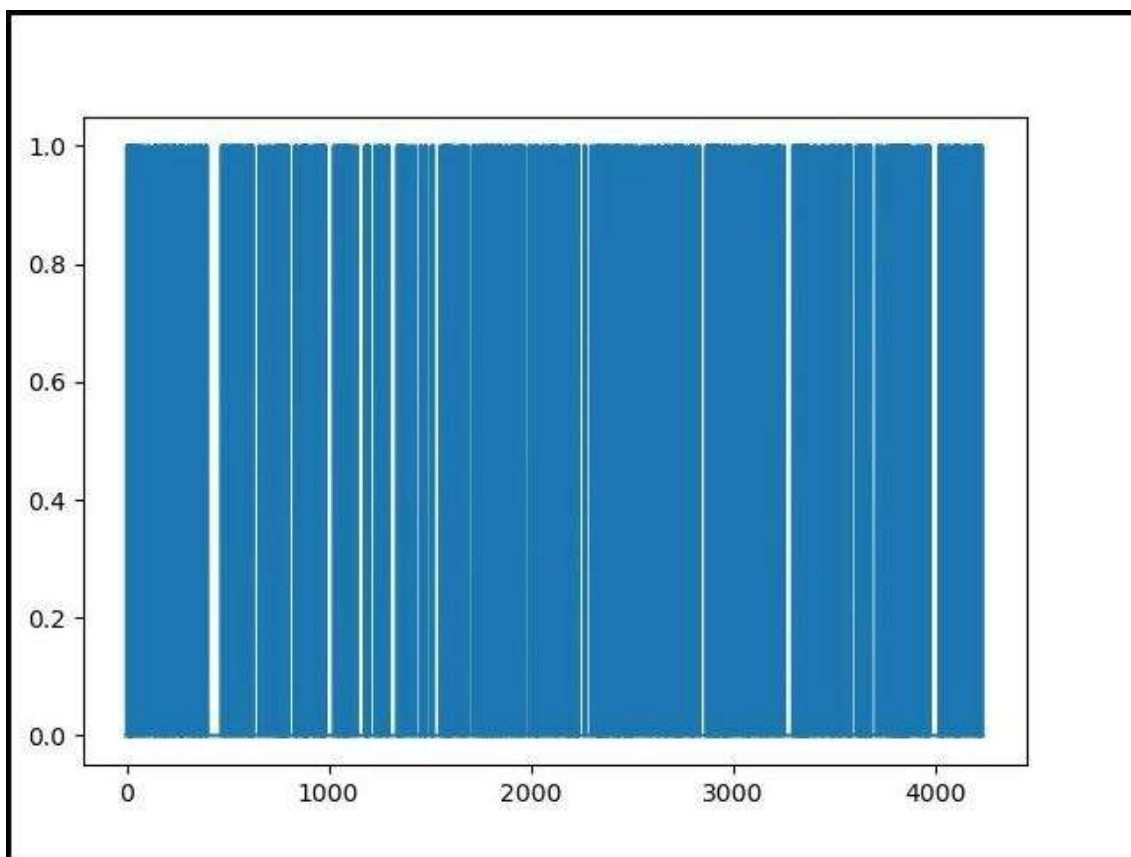


Counting number of patients affected by CHD where (0= Not Affected; 1= Affected)

```
laste = disease_df['TenYearCHD'].plot()
```

```
plt.show(laste)
```

Output



Fitting Logistic Regression Model for Heart Disease Prediction

```
from sklearn.linear_model import LogisticRegression
```

```
logreg = LogisticRegression()
```

```
logreg.fit(X_train, y_train)
```

```
y_pred = logreg.predict(X_test)
```

Evaluating Logistic Regression Model

```
# Evaluation and accuracy
```

```
from sklearn.metrics import accuracy_score
```

```
print('Accuracy of the model is =',
```

```
      accuracy_score(y_test, y_pred))
```

Output

Accuracy of the model is = 0.8490230905861457

Confusion Matrix

```
# Confusion matrix
```

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
cm = confusion_matrix(y_test, y_pred)
```

```
conf_matrix = pd.DataFrame(data = cm,  
                           columns = ['Predicted:0', 'Predicted:1'],  
                           index =['Actual:0', 'Actual:1'])
```

```
plt.figure(figsize = (8, 5))
```

```
sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap = "Greens")
```

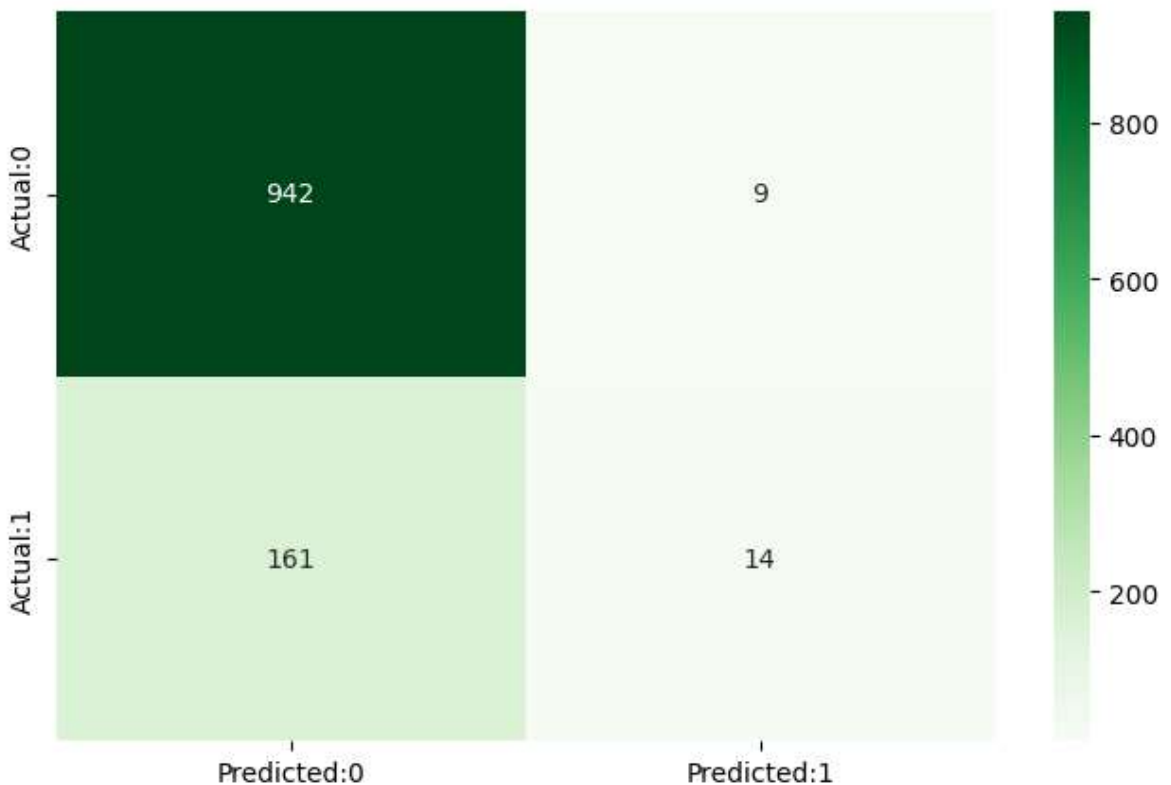
```
plt.show()
```

```
print('The details for confusion matrix is =')
```

```
print (classification_report(y_test, y_pred))
```

Output

Metric	Class 0	Class 1	Accuracy	Macro Avg	Weighted Avg
Precision	0.85	0.61	-	0.73	0.82
Recall	0.99	0.08	-	0.54	0.85
F1-Score	0.92	0.14	-	0.53	0.80
Support	951	175	1126	-	-



6.2 Credit Card Fraud Detection

Importing all the necessary Libraries

```
# import the necessary packages
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from matplotlib import gridspec
```

Loading the Data (File name as credit.csv)

```
# Load the dataset from the csv file using pandas
```

```
# best way is to mount the drive on colab and
```

```
# copy the path for the csv file
```

Artificial Intelligence(AI) with it's Applications

```
data = pd.read_csv("credit.csv")
```

Understanding the Data

```
# Grab a peek at the data
```

```
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098898	0.363787	0.090794	-0.551600	-0.817801	-0.991390	-0.311189
1	0.0	1.191857	0.286151	0.166480	0.448154	0.080018	-0.082361	-0.078603	0.085102	-0.255425	-0.166974	1.612727	1.085235	0.489095	-0.143772
2	1.0	-1.358354	-1.340183	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247876	-1.514654	0.207643	0.624501	0.088084	0.717293	-0.165946
3	1.0	-0.986272	-0.185226	1.792993	-0.863291	-0.010308	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

Describing the Data

```
# Print the shape of the data
```

```
# data = data.sample(frac = 0.1, random_state = 48)
```

```
print(data.shape)
```

```
print(data.describe())
```

Statistic	Time	V1	...	Amount	Class
Count	284807.000000	284807.000000	...	284807.000000	284807.000000
Mean	94813.859575	3.919560e-15	...	88.349619	0.001727
Std Dev	47488.145955	1.958696e+00	...	250.120109	0.041527
Min	0.000000	-5.640751e+01	...	0.000000	0.000000
25th Percentile	54201.500000	-9.203734e-01	...	5.600000	0.000000
50th Percentile (Median)	84692.000000	1.810880e-02	...	22.000000	0.000000
75th Percentile	139320.500000	1.315642e+00	...	77.165000	0.000000
Max	172792.000000	2.454930e+00	...	25691.160000	1.000000

Imbalance in the data

Time to explain the data we are dealing with.

```
# Determine number of fraud cases in dataset
```

```
fraud = data[data['Class'] == 1]
```

```
valid = data[data['Class'] == 0]
```

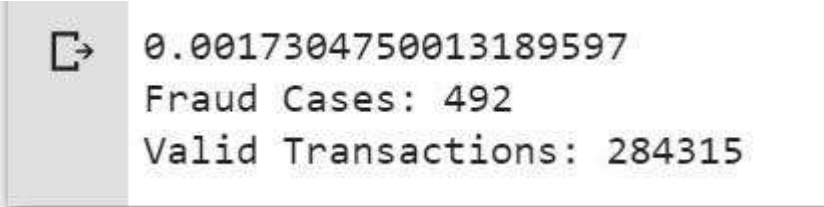
```
outlierFraction = len(fraud)/float(len(valid))
```

Artificial Intelligence(AI) with it's Applications

```
print(outlierFraction)

print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))

print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```



```
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
```

Only 0.17% fraudulent transaction out all the transactions. The data is highly Unbalanced. Let's first apply our models without balancing it and if we don't get a good accuracy then we can find a way to balance this dataset. But first, let's implement the model without it and will balance the data only if needed.

Print the amount details for Fraudulent Transaction

```
print("Amount details of the fraudulent transaction")

fraud.Amount.describe()
```

Output

Amount details of the fraudulent transaction

Statistic	Value
Count	492.000000
Mean	122.211321
Standard Deviation (Std)	256.683288
Minimum (Min)	0.000000
25th Percentile (Q1)	1.000000
Median (50th Percentile)	9.250000
75th Percentile (Q3)	105.890000
Maximum (Max)	2125.870000

Print the amount details for Normal Transaction

```
print("details of valid transaction")

valid.Amount.describe()
```

Output

Statistic	Value
Count	284315.000000
Mean	88.291022
Standard Deviation (Std)	250.105092
Minimum (Min)	0.000000
25th Percentile (Q1)	5.650000
Median (50th Percentile)	22.000000
75th Percentile (Q3)	77.050000
Maximum (Max)	25691.160000

As we can clearly notice from this, the average Money transaction for the fraudulent ones is more. This makes this problem crucial to deal with.

Plotting the Correlation Matrix

The correlation matrix graphically gives us an idea of how features correlate with each other and can help us predict what are the features that are most relevant for the prediction.

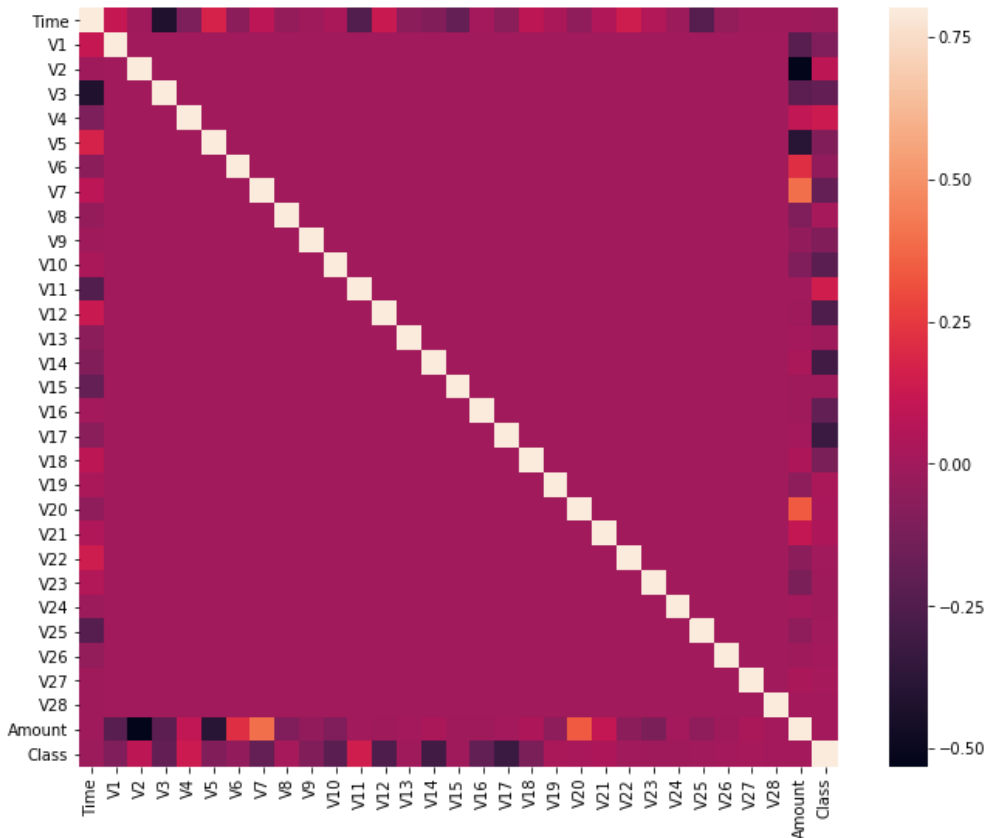
```
# Correlation matrix
```

```
corrmat = data.corr()
```

```
fig = plt.figure(figsize = (12, 9))
```

```
sns.heatmap(corrmat, vmax = .8, square = True)
```

```
plt.show()
```



In the HeatMap we can clearly see that most of the features do not correlate to other features but there are some features that either has a positive or a negative correlation with each other. For example, V2 and V5 are highly negatively correlated with the feature called *Amount*. We also see some correlation with V20 and *Amount*. This gives us a deeper understanding of the Data available to us.

Separating the X and the Y values

Dividing the data into inputs parameters and outputs value format

```
# dividing the X and the Y from the dataset
```

```
X = data.drop(['Class'], axis = 1)
```

```
Y = data["Class"]
```

```
print(X.shape)
```

```
print(Y.shape)
```

```
# getting just the values for the sake of processing
```

```
# (its a numpy array with no columns)
```

```
xData = X.values
```

```
yData = Y.values
```

Output

```
(284807, 30)
```

```
(284807, )
```

Training and Testing Data Bifurcation

We will be dividing the dataset into two main groups. One for training the model and the other for Testing our trained model's performance.

```
# Using Scikit-learn to split data into training and testing sets
```

```
from sklearn.model_selection import train_test_split
```

```
# Split the data into training and testing sets
```

```
xTrain, xTest, yTrain, yTest = train_test_split( xData, yData, test_size = 0.2, random_state = 42)
```

Building a Random Forest Model using scikit learn

```
# Building the Random Forest Classifier (RANDOM FOREST)
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
# random forest model creation
```

```
rfc = RandomForestClassifier()
```

```
rfc.fit(xTrain, yTrain)
```

```
# predictions
```

```
yPred = rfc.predict(xTest)
```

Building all kinds of evaluating parameters

```
# Evaluating the classifier
```

```
# printing every score of the classifier
```

```
# scoring in anything

from sklearn.metrics import classification_report, accuracy_score

from sklearn.metrics import precision_score, recall_score

from sklearn.metrics import f1_score, matthews_corrcoef

from sklearn.metrics import confusion_matrix


n_outliers = len(fraud)

n_errors = (yPred != yTest).sum()

print("The model used is Random Forest classifier")


acc = accuracy_score(yTest, yPred)

print("The accuracy is {}".format(acc))


prec = precision_score(yTest, yPred)

print("The precision is {}".format(prec))


rec = recall_score(yTest, yPred)

print("The recall is {}".format(rec))


f1 = f1_score(yTest, yPred)

print("The F1-Score is {}".format(f1))


MCC = matthews_corrcoef(yTest, yPred)

print("The Matthews correlation coefficient is {}".format(MCC))
```

Output

The model used is Random Forest classifier

The accuracy is 0.9995611109160493

The precision is 0.9866666666666667

The recall is 0.7551020408163265

The F1-Score is 0.8554913294797689

The Matthews correlation coefficient is 0.8629589216367891

Visualizing the Confusion Matrix

```
# printing the confusion matrix
```

```
LABELS = ['Normal', 'Fraud']
```

```
conf_matrix = confusion_matrix(yTest, yPred)
```

```
plt.figure(figsize =(12, 12))
```

```
sns.heatmap(conf_matrix, xticklabels = LABELS,
```

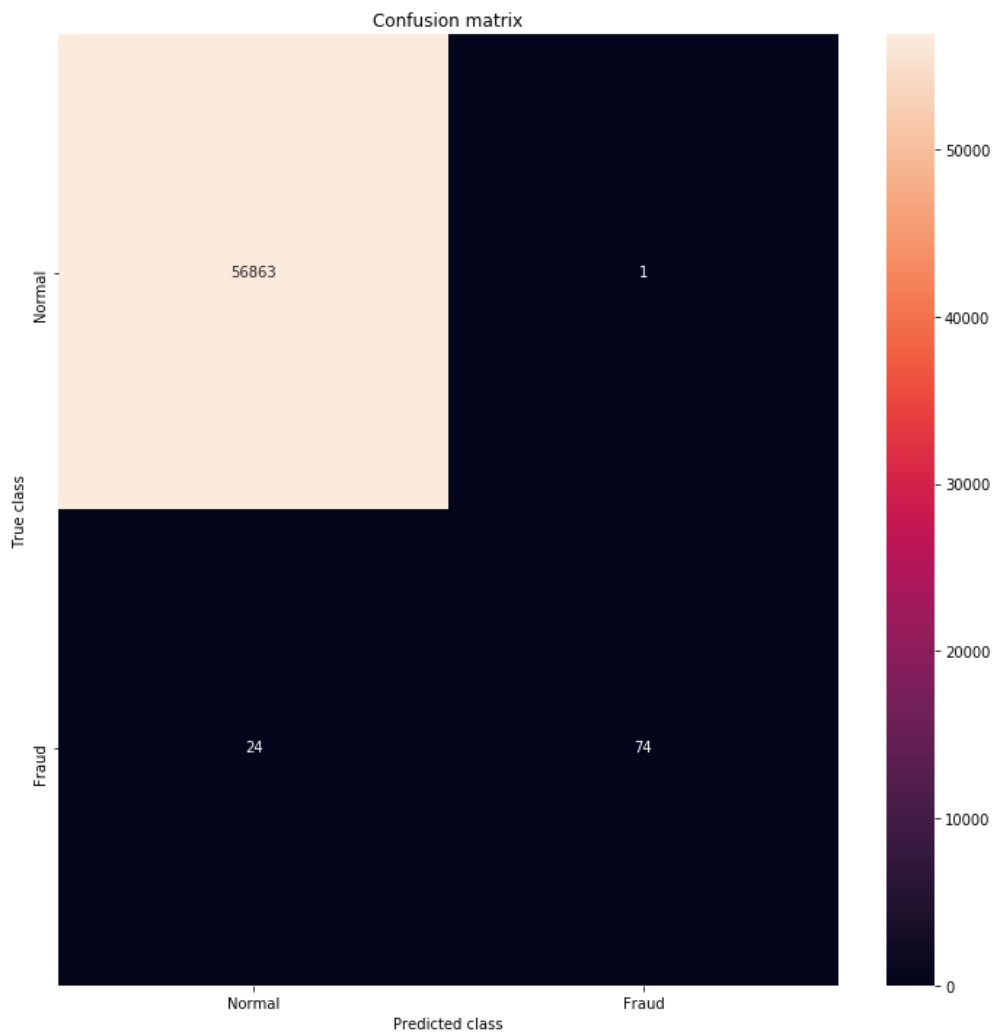
```
            yticklabels = LABELS, annot = True, fmt ="d");
```

```
plt.title("Confusion matrix")
```

```
plt.ylabel('True class')
```

```
plt.xlabel('Predicted class')
```

```
plt.show()
```



6.3 Autism Prediction

Importing Libraries and Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

Artificial Intelligence(AI) with it's Applications

```
from sklearn import metrics

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.linear_model import LogisticRegression

from imblearn.over_sampling import RandomOverSampler


import warnings

warnings.filterwarnings('ignore')
```

The dataset will now be loaded into a pandas data frame, and the first five rows will be printed. (Collect data and store the file name as dats.csv)

```
df = pd.read_csv('train.csv')
print(df.head())
```

Output

```
   ID  A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
0    1         1         0         1         0         1         0         1
1    2         0         0         0         0         0         0         0
2    3         1         1         1         1         1         1         1
3    4         0         0         0         0         0         0         0
4    5         0         0         0         0         0         0         0

   A8_Score  A9_Score  ...  gender  ethnicity  jaundice  austim  \
0         0         1  ...     f         ?         no     no
1         0         0  ...     m         ?         no     no
2         1         1  ...     m  White-European     no     yes
3         0         0  ...     f         ?         no     no
4         0         0  ...     m         ?         no     no

   contry_of_res  used_app_before  result  age_desc  relation  Class/ASD
0        Austria              no   6.351166  18 and more     Self         0
1         India              no   2.255185   18 and more     Self         0
2  United States              no  14.851484   18 and more     Self         1
3  United States              no   2.276617   18 and more     Self         0
4  South Africa              no  -4.777286   18 and more     Self         0

[5 rows x 22 columns]
```

Now let's check the size of the dataset,

```
df.shape
```

Output

```
(800, 22)
```

Let's check which column of the dataset contains which type of data,

```
df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 22 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ID                    800 non-null    int64
1   A1_Score              800 non-null    int64
2   A2_Score              800 non-null    int64
3   A3_Score              800 non-null    int64
4   A4_Score              800 non-null    int64
5   A5_Score              800 non-null    int64
6   A6_Score              800 non-null    int64
7   A7_Score              800 non-null    int64
8   A8_Score              800 non-null    int64
9   A9_Score              800 non-null    int64
10  A10_Score             800 non-null    int64
11  age                   800 non-null    float64
12  gender                800 non-null    object
13  ethnicity             800 non-null    object
14  jaundice              800 non-null    object
15  austim                800 non-null    object
16  contry_of_res         800 non-null    object
17  used_app_before       800 non-null    object
18  result                800 non-null    float64
19  age_desc              800 non-null    object
20  relation              800 non-null    object
21  Class/ASD             800 non-null    int64
dtypes: float64(2), int64(12), object(8)
memory usage: 137.6+ KB
```

As per the above information regarding the data in each column we can observe that there are no null values,

```
df.describe().T
```

Output

	count	mean	std	min	25%	50%	75%	max
ID	800.0	400.500000	231.084400	1.000000	200.750000	400.500000	600.250000	800.000000
A1_Score	800.0	0.560000	0.496697	0.000000	0.000000	1.000000	1.000000	1.000000
A2_Score	800.0	0.530000	0.499411	0.000000	0.000000	1.000000	1.000000	1.000000
A3_Score	800.0	0.450000	0.497805	0.000000	0.000000	0.000000	1.000000	1.000000
A4_Score	800.0	0.415000	0.493030	0.000000	0.000000	0.000000	1.000000	1.000000
A5_Score	800.0	0.395000	0.489157	0.000000	0.000000	0.000000	1.000000	1.000000
A6_Score	800.0	0.303750	0.460164	0.000000	0.000000	0.000000	1.000000	1.000000
A7_Score	800.0	0.397500	0.489687	0.000000	0.000000	0.000000	1.000000	1.000000
A8_Score	800.0	0.508750	0.500236	0.000000	0.000000	1.000000	1.000000	1.000000
A9_Score	800.0	0.495000	0.500288	0.000000	0.000000	0.000000	1.000000	1.000000
A10_Score	800.0	0.617500	0.486302	0.000000	0.000000	1.000000	1.000000	1.000000
age	800.0	28.452118	16.310966	2.718550	17.198153	24.848350	35.865429	89.461718
result	800.0	8.537303	4.807676	-6.137748	5.306575	9.605299	12.514484	15.853126
Class/ASD	800.0	0.201250	0.401185	0.000000	0.000000	0.000000	0.000000	1.000000

Data Cleaning

Data obtained from primary sources, often referred to as raw data, requires extensive preprocessing before it can be used for analysis or modeling. This process, known as data cleaning, involves several essential steps, including:

- ❖ **Outlier Removal:** Identifying and eliminating data points that deviate significantly from the dataset's overall pattern.
- ❖ **Null Value Imputation:** Handling missing values by filling them with appropriate estimates or removing them to ensure data integrity.
- ❖ **Resolving Discrepancies:** Addressing inconsistencies or errors in the data to maintain accuracy and reliability in analysis.

```
df['ethnicity'].value_counts()
```

Output

```
White-European    257
?                 203
Middle Eastern    97
Asian             67
Black             47
South Asian       34
Pasifika          32
Others            29
Latino            17
Hispanic          9
Turkish           5
others            3
Name: ethnicity, dtype: int64
```

In the above two outputs we can observe some ambiguity that there are '?', 'others', and 'Others' which all must be the same as they are unknown or we can say that null values have been substituted with some indicator.

```
df['relation'].value_counts()
```

Output

```
Self              709
?                 40
Parent            29
Relative          18
Others            2
Health care professional  2
Name: relation, dtype: int64
```

The same is the case with this column so, let's clean this data, and along with this let's convert 'yes' and 'no' to 0 and 1.

```
df = df.replace({'yes':1, 'no':0, '?':'Others', 'others':'Others'})
```

Now we have cleaned the data a bit to derive insights from it.

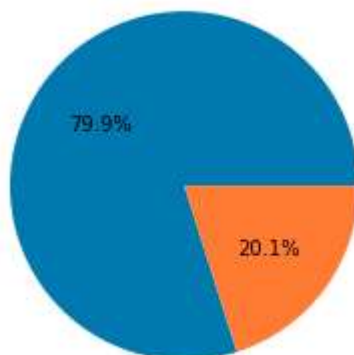
Exploratory Data Analysis

EDA is an approach to analyzing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations. Here we will see how to check the data imbalance and skewness of the data.

```
plt.pie(df['Class/ASD'].value_counts().values, autopct='%1.1f%%')
```

```
plt.show()
```

Output



The dataset we have is highly imbalanced. If we will train our model using this data then the model will face a hard time predicting the positive class which is our main objective here to predict whether a person has autism or not with high accuracy.

```
ints = [ ]
```

```
objects = [ ]
```

```
floats = [ ]
```

```
for col in df.columns:
```

```
    if df[col].dtype == int:
```

```
        ints.append(col)
```

```
    elif df[col].dtype == object:
```

```
        objects.append(col)
```

```
    else:
```

```
        floats.append(col)
```

Artificial Intelligence(AI) with it's Applications

The 'ID' column will contain a unique value for each of the rows and for the column 'Class/ASD' we have already analyzed its distribution so, that is why they have been removed in the above code.

```
# Convert the data to long-form using melt
```

```
df_melted = df.melt(id_vars=['ID', 'Class/ASD'], value_vars=ints, var_name='col',  
value_name='value')
```

```
plt.subplots(figsize=(15,15))
```

```
for i, col in enumerate(ints):
```

```
    plt.subplot(5,3,i+1)
```

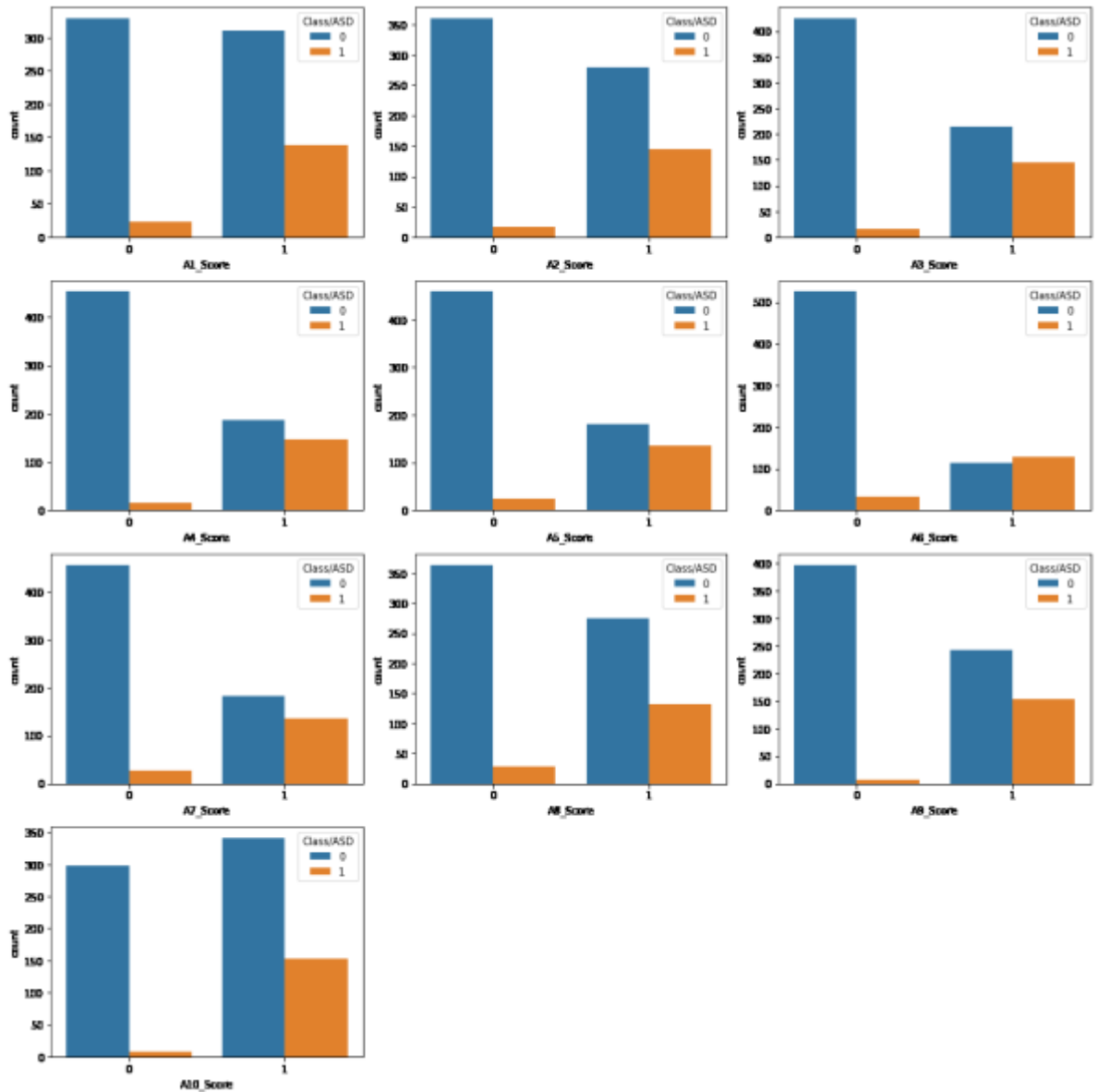
```
    # Use the melted DataFrame and specify x and hue
```

```
    sb.countplot(x='value', hue='Class/ASD', data=df_melted[df_melted['col'] == col])
```

```
plt.tight_layout()
```

```
plt.show()
```

Output



From these plots, we can conclude that if the score of some indicator is 0 then the chances of that person not having autism is quite high except in the case of A10_Score.

```
plt.subplots(figsize=(15, 30))
```

```
for i, col in enumerate(objects):
```

```
plt.subplot(5, 3, i+1)

# Convert the data to long-form for the specific column

df_melted = df.melt(id_vars=['Class/ASD'], value_vars=[col], var_name='col',
value_name='value')

# Use the melted DataFrame and specify x and hue

sb.countplot(x='value', hue='Class/ASD', data=df_melted)

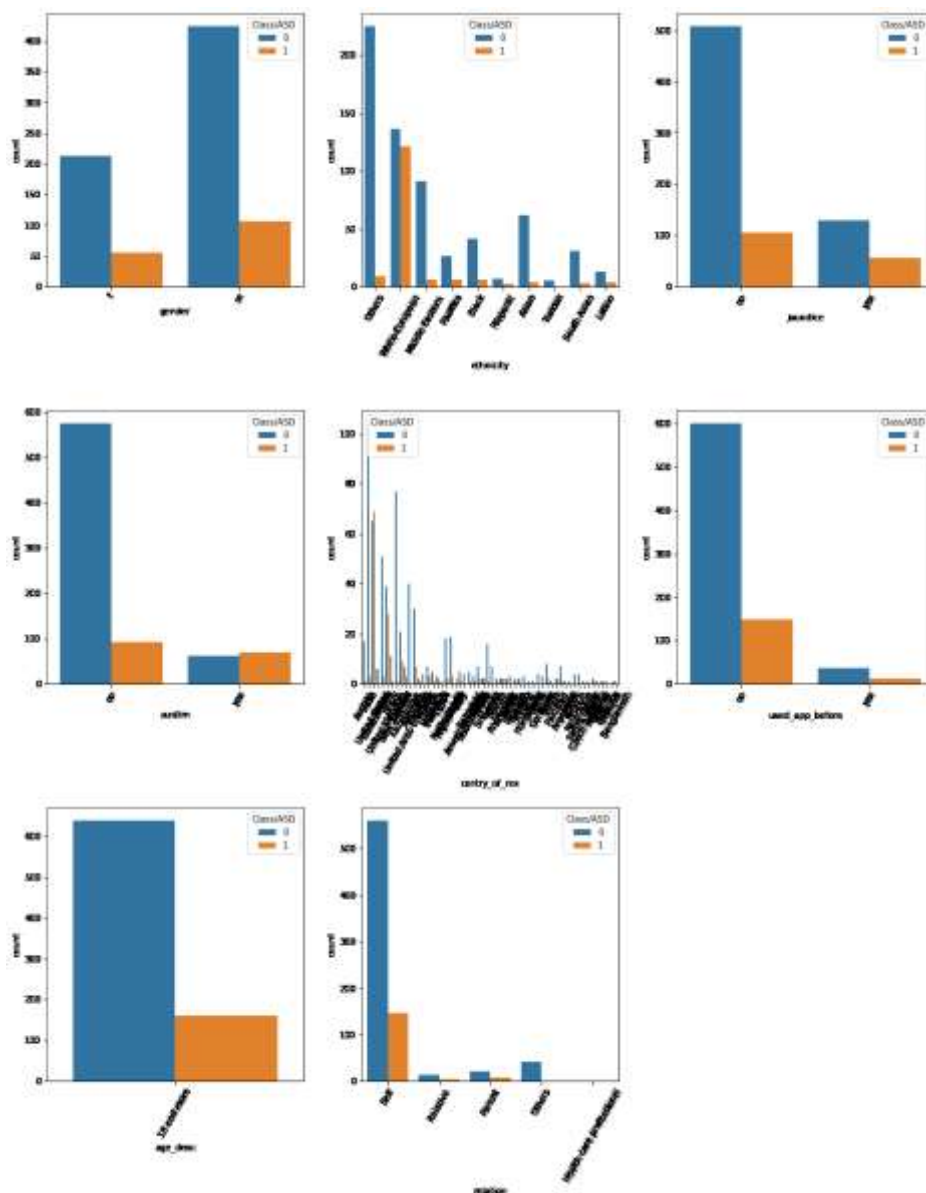
plt.xticks(rotation=60)

plt.tight_layout()

plt.show()
```

Output

Artificial Intelligence(AI) with it's Applications



From the above plots we can draw the following observations,

Age_desc is the same for all the data points.

This used_app_before feature doesn't seem to be useful or it will be the source of data leakage.

Here it seems like the chances of a male having autism is higher than a female but that is not true because we don't have an equal number of examples of males and females.

```
plt.figure(figsize=(15,5))
```

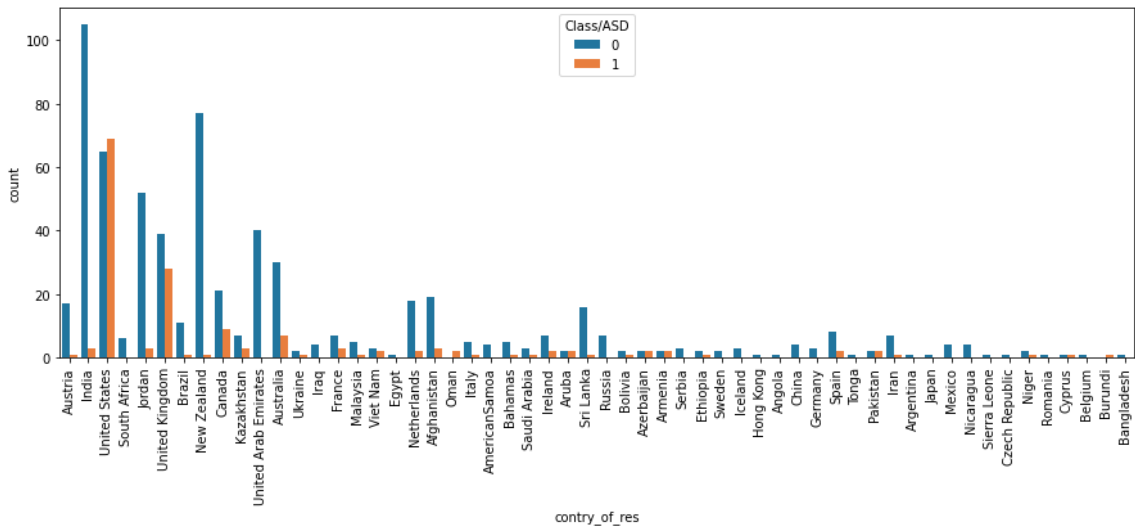
```
sb.countplot(data=df, x='contry_of_res', hue='Class/ASD')
```

Artificial Intelligence(AI) with it's Applications

```
plt.xticks(rotation=90)
```

```
plt.show()
```

Output



In some places approximately 50% of the data available for that country have autism and in some places, this is quite low. This implies that the geography of a person also gives an idea of having autism.

```
plt.subplots(figsize=(15,5))
```

```
for i, col in enumerate(floats):
```

```
    plt.subplot(1,2,i+1)
```

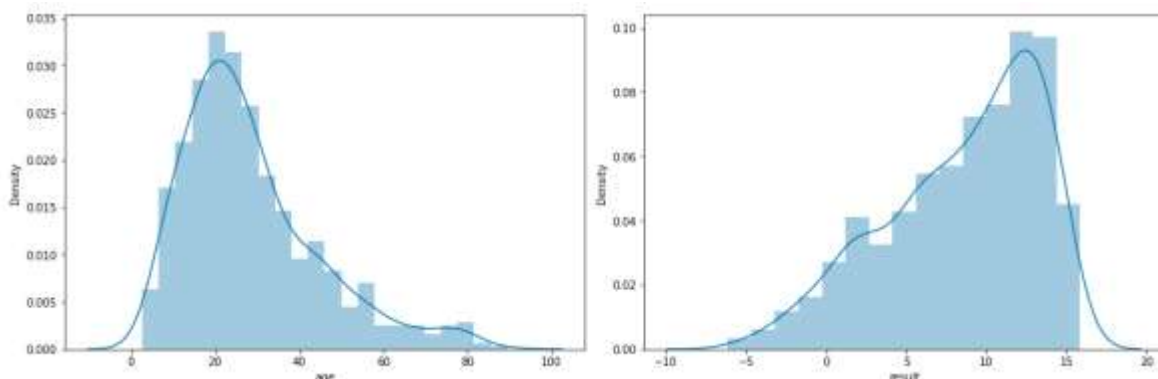
```
    sb.distplot(df[col])
```

```
plt.tight_layout()
```

```
plt.show()
```

Output

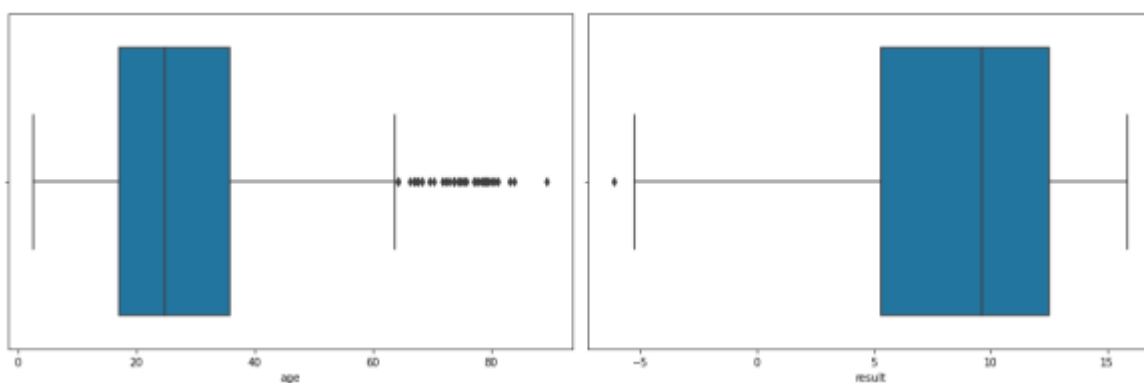
Artificial Intelligence(AI) with it's Applications



Both of the continuous data are skewed left one is positive and the right one is negatively skewed.

```
plt.subplots(figsize=(15,5))
for i, col in enumerate(floats):
    plt.subplot(1,2,i+1)
    sb.boxplot(df[col])
plt.tight_layout()
plt.show()
```

Output



Ah! some outliers in the result column. Let's remove that as it doesn't seem like too much loss of information.

```
df = df[df['result']>-5]
df.shape
```

Output

(798, 22)

So, here we lost only two data points.

Feature Engineering

Feature Engineering helps to derive some valuable features from the existing ones. These extra features sometimes help in increasing the performance of the model significantly and certainly help to gain deeper insights into the data.

```
# This functions make groups by taking
```

```
# the age as a parameter
```

```
def convertAge(age):
```

```
    if age < 4:
```

```
        return 'Toddler'
```

```
    elif age < 12:
```

```
        return 'Kid'
```

```
    elif age < 18:
```

```
        return 'Teenager'
```

```
    elif age < 40:
```

```
        return 'Young'
```

```
    else:
```

```
        return 'Senior'
```

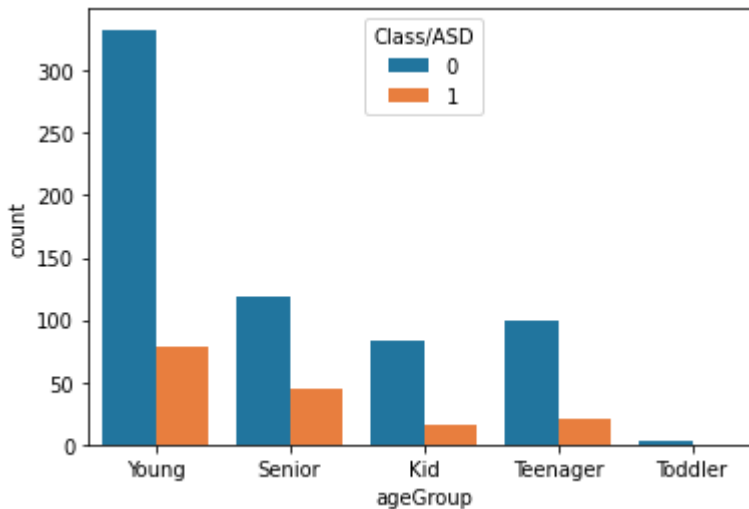
```
df['ageGroup'] = df['age'].apply(convertAge)
```

Now, let's explore the number of cases for each age group.

```
sb.countplot(x=df['ageGroup'], hue=df['Class/ASD'])
```

```
plt.show()
```

Output



Here we can conclude that the Young and Toddler group of people have lower chances of having Autism.

```
def add_feature(data):
```

```
    # Creating a column with all values zero
```

```
    data['sum_score'] = 0
```

```
    for col in data.loc[:, 'A1_Score': 'A10_Score'].columns:
```

```
        # Updating the 'sum_score' value with scores
```

```
        # from A1 to A10
```

```
        data['sum_score'] += data[col]
```

```
    # Creating a random data using the below three columns
```

```
    data['ind'] = data['austim'] + data['used_app_before'] + data['jaundice']
```

```
    return data
```

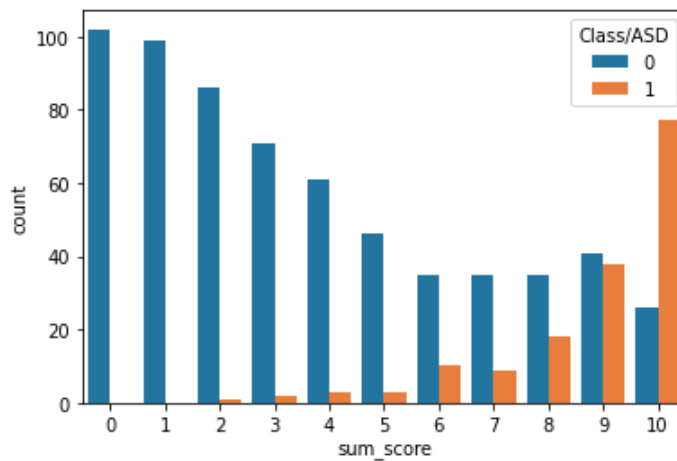
```
df = add_feature(df)
```

In the above code block, we have just summed up the clinical scores given from A1 to A10.

```
sb.countplot(x=df['sum_score'], hue=df['Class/ASD'])
```

```
plt.show()
```

Output



Another amazing observation higher the sum score higher the chances of having autism is higher as well and similarly for lower sum scores that are for less than 5 it is rare that the person has autism.

Applying log transformations to remove the skewness of the data.

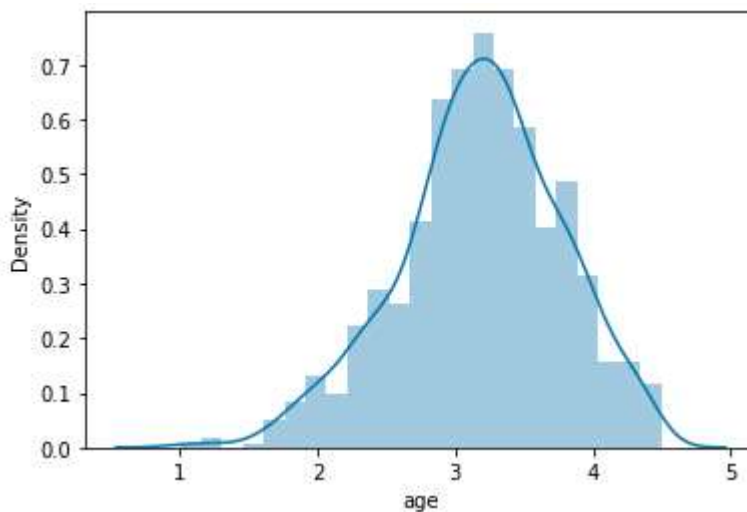
```
df['age'] = df['age'].apply(lambda x: np.log(x))
```

The age data given to us is positively skewed. As we know that skewed data impacts the performance of the model, we will apply log transformations to remove the skewness of the data.

```
sb.distplot(df['age'])
```

```
plt.show()
```

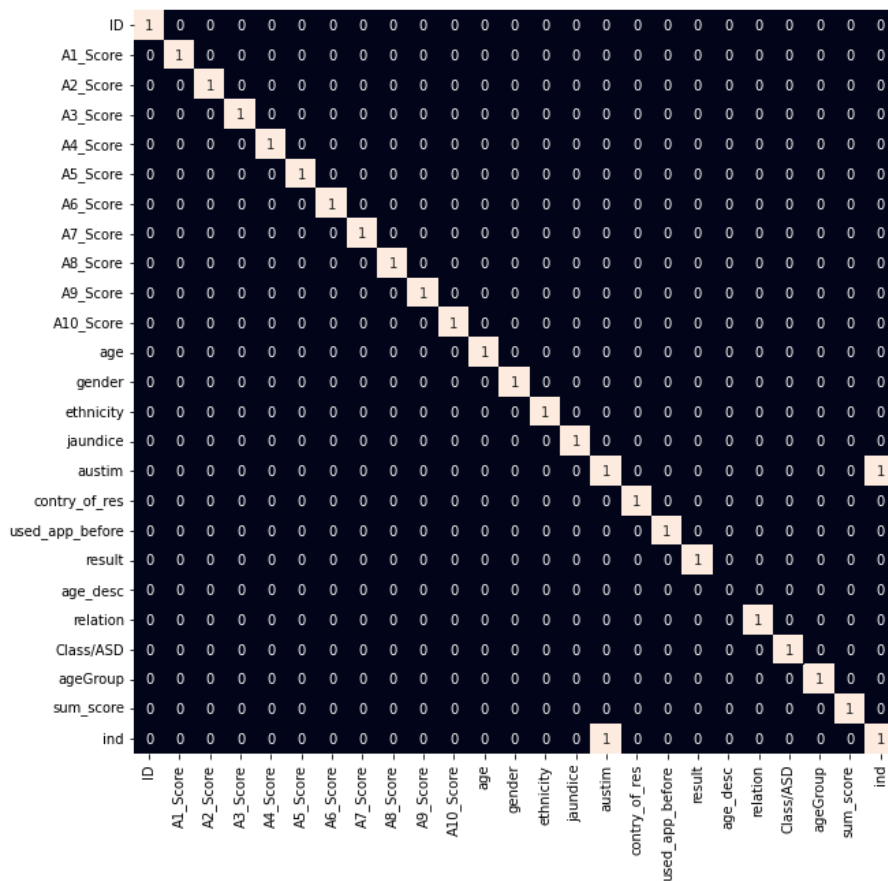
Output



Now, the skewness of the data has been removed successfully by using the log transformations.

```
def encode_labels(data):  
    for col in data.columns:  
        # Here we will check if datatype  
        # is object then we will encode it  
        if data[col].dtype == 'object':  
            le = LabelEncoder()  
            data[col] = le.fit_transform(data[col])  
    return data  
  
df = encode_labels(df)  
  
# Making a heatmap to visualize the correlation matrix  
plt.figure(figsize=(10,10))  
sb.heatmap(df.corr() > 0.8, annot=True, cbar=False)  
plt.show()
```

Output



From the above heat map, we can see that there are only one highly correlated features which we will remove before training the model on this data as highly correlated features do not help in learning useful patterns in the data.

Model Training

Now we will separate the features and target variables and split them into training and the testing data by using which we will select the model which is performing best on the validation data.

```
removal = ['ID', 'age_desc', 'used_app_before', 'austim']
```

```
features = df.drop(removal + ['Class/ASD'], axis=1)
```

```
target = df['Class/ASD']
```

Let's split the data into training and validation data. Also, the data was imbalanced earlier now we will balance it using the Random Over Sampler in this method we sample some

points from the minority class and repeat it multiple times so, that the two classes get balanced.

```
X_train, X_val, Y_train, Y_val = train_test_split(features, target, test_size = 0.2,  
random_state=10)
```

```
# As the data was highly imbalanced we will balance it by adding repetitive rows of  
minority class.
```

```
ros = RandomOverSampler(sampling_strategy='minority',random_state=0)
```

```
X, Y = ros.fit_resample(X_train,Y_train)
```

```
X.shape, Y.shape
```

Output

```
((1026, 20), (1026,))
```

Now, let's normalize the data to obtain stable and fast training,

```
# Normalizing the features for stable and fast training.
```

```
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
```

```
X_val = scaler.transform(X_val)
```

Now let's train some state-of-the-art machine learning models and compare them which fit better with our data.

```
models = [LogisticRegression(), XGBClassifier(), SVC(kernel='rbf')]
```

```
for model in models:
```

```
    model.fit(X, Y)
```

```
    print(f'{model} : ')
```

```
    print("Training Accuracy : ', metrics.roc_auc_score(Y, model.predict(X)))
```

```
    print("Validation Accuracy : ', metrics.roc_auc_score(Y_val, model.predict(X_val)))
```

```
    print()
```

```
LogisticRegression() :
```

```
Training Accuracy : 0.8664717348927876
```

```
Validation Accuracy : 0.782258064516129
```

XGBClassifier() :

Training Accuracy : 1.0

Validation Accuracy : 0.7491039426523298

SVC() :

Training Accuracy : 0.9405458089668616

Validation Accuracy : 0.8042114695340501

Model Evaluation

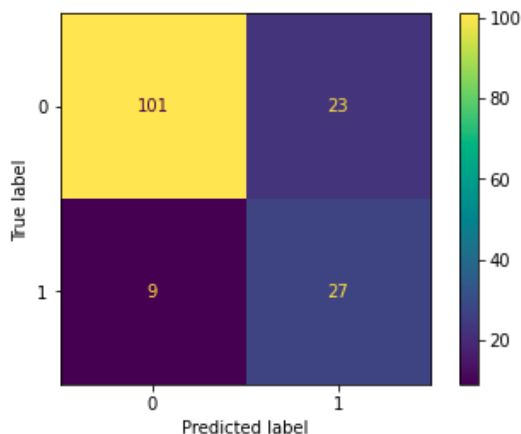
From the above accuracies, we can say that Logistic Regression and SVC() classifier perform better on the validation data with less difference between the validation and training data. Let's plot the confusion matrix as well for the validation data using the Logistic Regression model.

```
from sklearn.metrics import ConfusionMatrixDisplay
```

```
ConfusionMatrixDisplay.from_estimator(models[0], X_val, Y_val)
```

```
plt.show()
```

Output



6.4 Wine Quality Prediction

Importing libraries and Dataset:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression

import warnings
warnings.filterwarnings('ignore')
```

Now let's look at the first five rows of the dataset.

```
df = pd.read_csv('winequality.csv')
print(df.head())
```

Output

	type	fixed acidity	volatile acidity	citric acid	residual sugar	\
0	white	7.0	0.27	0.36	20.7	
1	white	6.3	0.30	0.34	1.6	
2	white	8.1	0.28	0.40	6.9	
3	white	7.2	0.23	0.32	8.5	
4	white	7.2	0.23	0.32	8.5	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	\
0	0.045	45.0	170.0	1.0010	3.00	
1	0.049	14.0	132.0	0.9940	3.30	
2	0.050	30.0	97.0	0.9951	3.26	
3	0.058	47.0	186.0	0.9956	3.19	
4	0.058	47.0	186.0	0.9956	3.19	

	sulphates	alcohol	quality
0	0.45	8.8	6
1	0.49	9.5	6
2	0.44	10.1	6
3	0.40	9.9	6
4	0.40	9.9	6

Artificial Intelligence(AI) with it's Applications

Let's explore the type of data present in each of the columns present in the dataset.

```
df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                -
0   type                                 6497 non-null   object
1   fixed acidity                        6487 non-null   float64
2   volatile acidity                    6489 non-null   float64
3   citric acid                         6494 non-null   float64
4   residual sugar                      6495 non-null   float64
5   chlorides                           6495 non-null   float64
6   free sulfur dioxide                 6497 non-null   float64
7   total sulfur dioxide                6497 non-null   float64
8   density                             6497 non-null   float64
9   pH                                  6488 non-null   float64
10  sulphates                           6493 non-null   float64
11  alcohol                             6497 non-null   float64
12  quality                             6497 non-null   int64
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

Now we'll explore the descriptive statistical measures of the dataset.

```
df.describe().T
```

Output

Artificial Intelligence(AI) with it's Applications

	count	mean	std	min	25%	50%	75%	max
fixed acidity	6487.0	7.216579	1.296750	3.80000	6.40000	7.00000	7.70000	15.90000
volatile acidity	6489.0	0.339691	0.164649	0.08000	0.23000	0.29000	0.40000	1.58000
citric acid	6494.0	0.318722	0.145265	0.00000	0.25000	0.31000	0.39000	1.66000
residual sugar	6495.0	5.444326	4.758125	0.60000	1.80000	3.00000	8.10000	65.80000
chlorides	6495.0	0.056042	0.035036	0.00900	0.03800	0.04700	0.06500	0.61100
free sulfur dioxide	6497.0	30.525319	17.749400	1.00000	17.00000	29.00000	41.00000	289.00000
total sulfur dioxide	6497.0	115.744574	56.521855	6.00000	77.00000	118.00000	156.00000	440.00000
density	6497.0	0.994697	0.002999	0.98711	0.99234	0.99489	0.99699	1.03898
pH	6488.0	3.218395	0.160748	2.72000	3.11000	3.21000	3.32000	4.01000
sulphates	6493.0	0.531215	0.148814	0.22000	0.43000	0.51000	0.60000	2.00000
alcohol	6497.0	10.491801	1.192712	8.00000	9.50000	10.30000	11.30000	14.90000
quality	6497.0	5.818378	0.873255	3.00000	5.00000	6.00000	6.00000	9.00000

Exploratory Data Analysis

EDA is an approach to analysing the data using visual techniques. It is used to discover trends, and patterns, or to check assumptions with the help of statistical summaries and graphical representations. Now let's check the number of null values in the dataset columns wise.

```
df.isnull().sum()
```

```
type          0
fixed acidity 10
volatile acidity  8
citric acid     3
residual sugar  2
chlorides       2
free sulfur dioxide  0
total sulfur dioxide  0
density         0
pH             9
sulphates       4
alcohol         0
quality         0
dtype: int64
```

Artificial Intelligence(AI) with it's Applications

Let's impute the missing values by means as the data present in the different columns are continuous values.

```
for col in df.columns:
```

```
    if df[col].isnull().sum() > 0:
```

```
        df[col] = df[col].fillna(df[col].mean())
```

```
df.isnull().sum().sum()
```

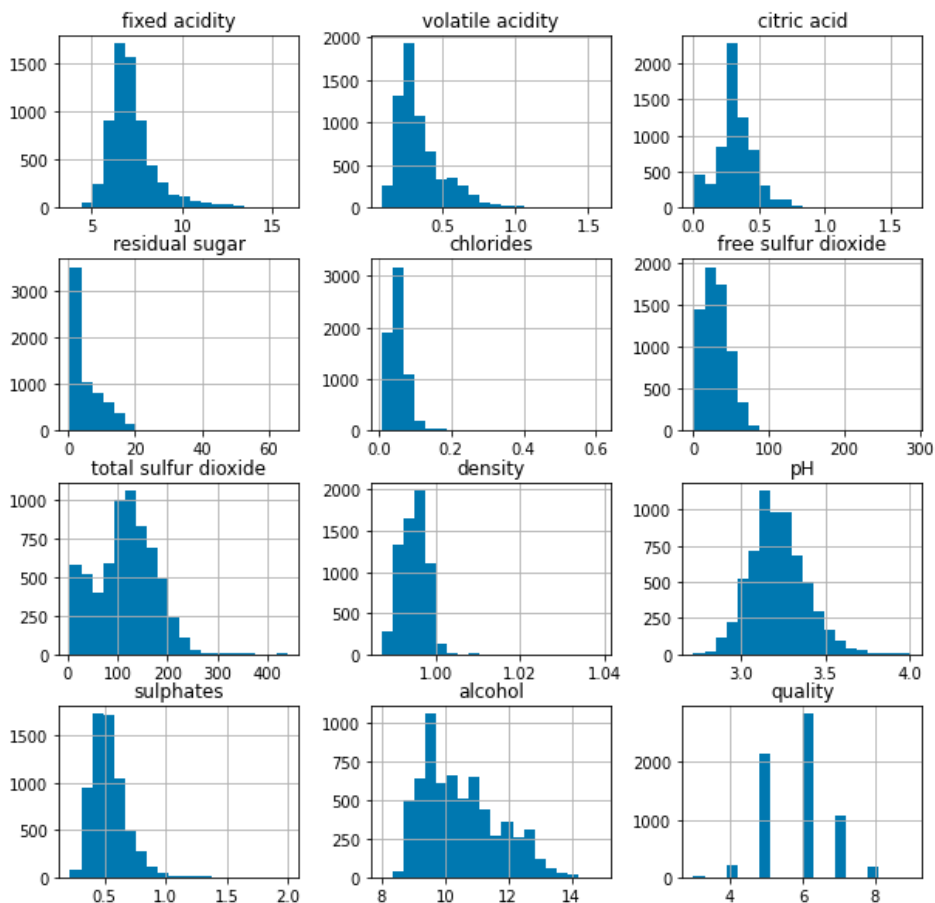
Output

0

Let's draw the histogram to visualise the distribution of the data with continuous values in the columns of the dataset.

```
df.hist(bins=20, figsize=(10, 10))
```

```
plt.show()
```

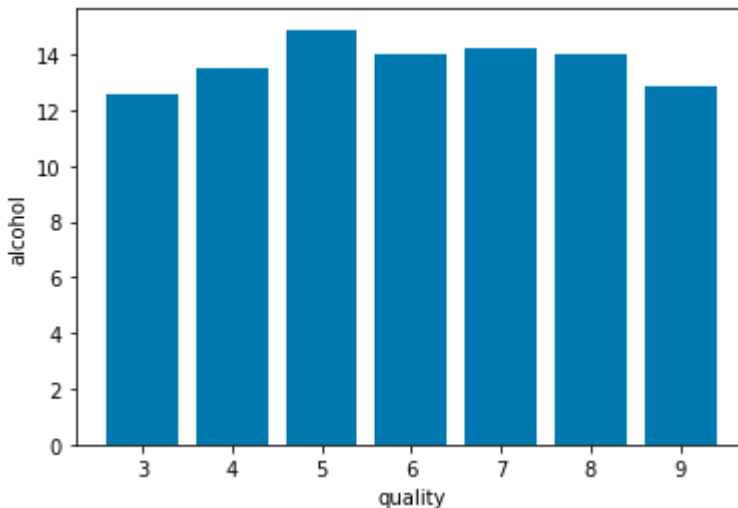


Artificial Intelligence(AI) with it's Applications

Now let's draw the count plot to visualise the number data for each quality of wine.

```
plt.bar(df['quality'], df['alcohol'])  
plt.xlabel('quality')  
plt.ylabel('alcohol')  
plt.show()
```

Output



There are times the data provided to us contains redundant features they do not help with increasing the model's performance that is why we remove them before using them to train our model.

```
# Convert 'object' columns to numerical if they represent numbers  
for col in df.columns:  
    if df[col].dtype == 'object':  
        try:
```

Artificial Intelligence(AI) with it's Applications

```
df[col] = pd.to_numeric(df[col], errors='coerce') # Convert to numeric, replace non-convertibles with NaN
```

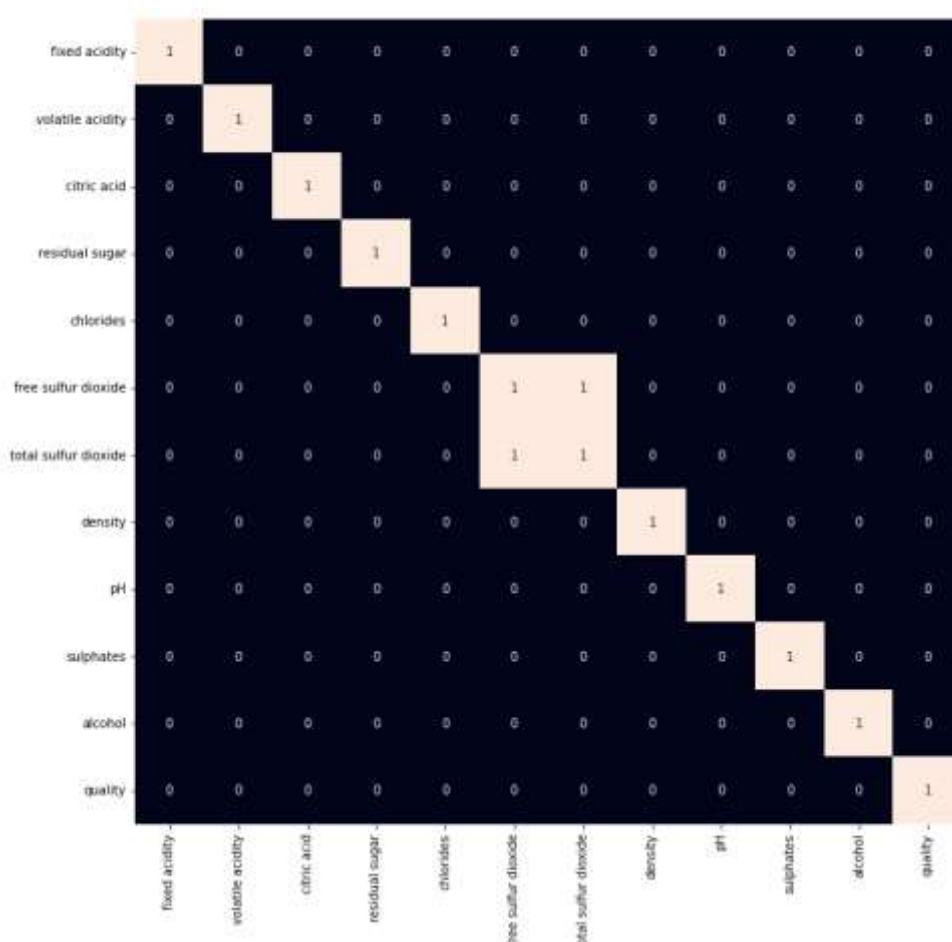
except:

```
pass # Skip columns that cannot be converted
```

```
plt.figure(figsize=(12, 12))
```

```
sb.heatmap(df.corr() > 0.7, annot=True, cbar=False)
```

```
plt.show()
```



From the above heat map we can conclude that the ‘**total sulphur dioxide**’ and ‘**free sulphur dioxide**’ are highly correlated features so, we will remove them.

```
df = df.drop('total sulfur dioxide', axis=1)
```

Model Development

Artificial Intelligence(AI) with it's Applications

Let's prepare our data for training and splitting it into training and validation data so, that we can select which model's performance is best as per the use case. We will train some of the state of the art machine learning classification models and then select best out of them using validation data.

```
df['best quality'] = [1 if x > 5 else 0 for x in df.quality]
```

We have a column with object data type as well let's replace it with the 0 and 1 as there are only two categories.

```
df.replace({'white': 1, 'red': 0}, inplace=True)
```

After segregating features and the target variable from the dataset we will split it into 80:20 ratio for model selection.

```
features = features.fillna(features.mean())
```

```
features = df.drop(['quality', 'best quality'], axis=1)
```

```
target = df['best quality']
```

```
xtrain, xtest, ytrain, ytest = train_test_split(  
    features, target, test_size=0.2, random_state=40)
```

```
# Impute missing values after splitting
```

```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(strategy='mean') # Or another strategy like 'median'
```

```
xtrain = imputer.fit_transform(xtrain)
```

```
xtest = imputer.transform(xtest)
```

```
xtrain.shape, xtest.shape
```

Output

```
((5197, 10), (1300, 10))
```

Normalising the data before training help us to achieve stable and fast training of the model.

```
norm = MinMaxScaler()
```

```
xtrain = norm.fit_transform(xtrain)
```

```
xtest = norm.transform(xtest)
```

As the data has been prepared completely let's train some state of the art machine learning model on it.

```
models = [LogisticRegression(), XGBClassifier(), SVC(kernel='rbf')]
```

```
for i in range(3):
```

```
    models[i].fit(xtrain, ytrain)
```

```
    print(f'{models[i]} :')
```

```
    print('Training Accuracy : ', metrics.roc_auc_score(ytrain, models[i].predict(xtrain)))
```

```
    print('Validation Accuracy : ', metrics.roc_auc_score(
        ytest, models[i].predict(xtest)))
```

```
    print()
```

Output

LogisticRegression() :

Training Accuracy : 0.6975101024661644

Validation Accuracy : 0.6855058693719925

XGBClassifier() :

Training Accuracy : 0.9762240429934201

Validation Accuracy : 0.8045662590288206

SVC() :

Training Accuracy : 0.7203202525576721

Validation Accuracy : 0.7073819229472522

Model Evaluation

From the above accuracies we can say that Logistic Regression and SVC() classifier performing better on the validation data with less difference between the validation and training data. Let's plot the confusion matrix as well for the validation data using the Logistic Regression model.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

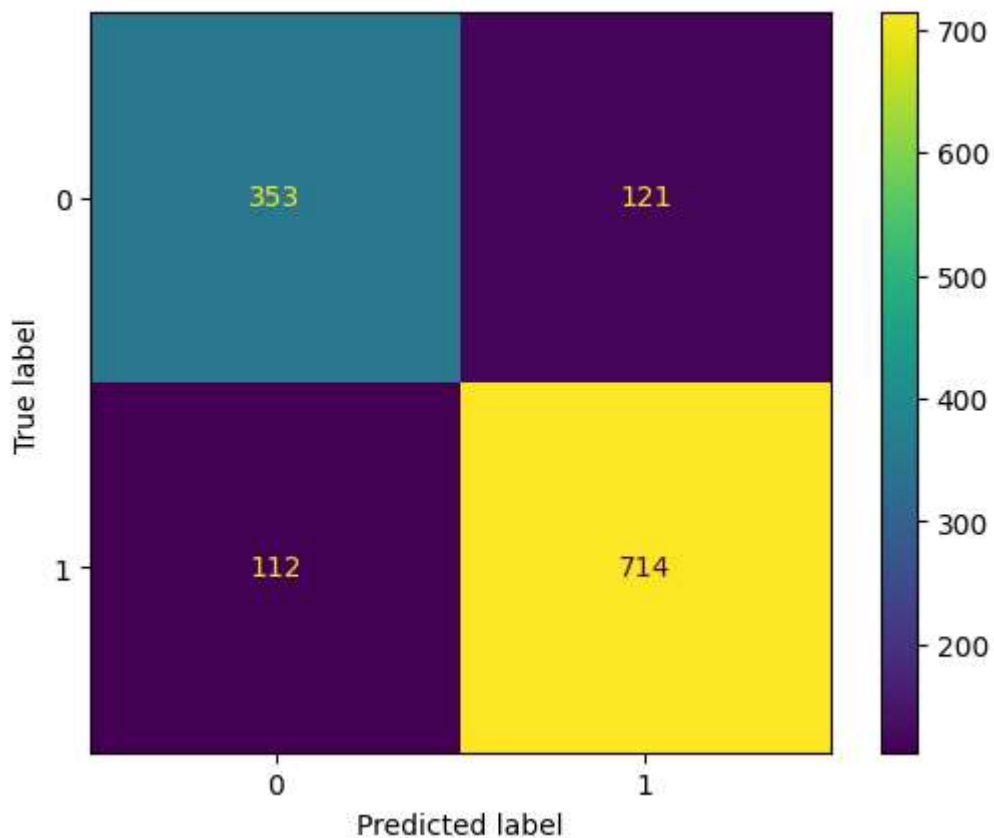
# Assuming 'models[1]' is your trained classifier

cm = confusion_matrix(ytest, models[1].predict(xtest))

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=models[1].classes_)
# Assuming your model has a 'classes_' attribute

disp.plot()

plt.show()
```



Let's also print the classification report for the best performing model.

```
print(metrics.classification_report(ytest,  
                                   models[1].predict(xtest)))
```

Output

Metric	Class 0	Class 1	Accuracy	Macro Avg	Weighted Avg
Precision	0.76	0.86	-	0.81	0.82
Recall	0.74	0.86	-	0.80	0.82
F1-Score	0.75	0.86	-	0.81	0.82
Support	474	826	1300	-	-

6.5 Calories Burnt Prediction

Importing Libraries and Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBRegressor
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor
```

```
import warnings
warnings.filterwarnings('ignore')
```

Now let's load the dataset into the panda's data frame and print its first five rows.

```
df = pd.read_csv('calories.csv')
df.head()
```

Output

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	male	68	190.0	94.0	29.0	105.0	40.8	231.0
1	14861698	female	20	166.0	60.0	14.0	94.0	40.3	66.0
2	11179863	male	69	179.0	79.0	5.0	88.0	38.7	26.0
3	16180408	female	34	179.0	71.0	13.0	100.0	40.5	71.0
4	17771927	female	27	154.0	58.0	10.0	81.0	39.8	35.0

Artificial Intelligence(AI) with it's Applications

Now let's check the size of the dataset.

```
df.shape
```

Output

```
(15000, 9)
```

Let's check which column of the dataset contains which type of data.

```
df.info()
```

Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15000 entries, 0 to 14999
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   User_ID     15000 non-null  int64
1   Gender      15000 non-null  object
2   Age         15000 non-null  int64
3   Height      15000 non-null  float64
4   Weight      15000 non-null  float64
5   Duration    15000 non-null  float64
6   Heart_Rate  15000 non-null  float64
7   Body_Temp   15000 non-null  float64
8   Calories    15000 non-null  float64
dtypes: float64(6), int64(2), object(1)
memory usage: 1.0+ MB
```

Now we will check the descriptive statistical measures of the data.

```
df.describe()
```

Artificial Intelligence(AI) with it's Applications

Output

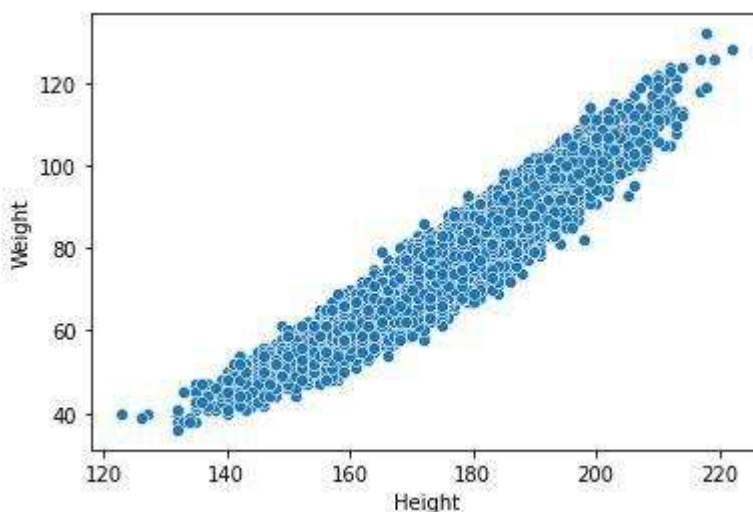
	count	mean	std	min	25%	50%	75%	max
User_ID	15000.0	1.497736e+07	2.872851e+06	10001159.0	12474190.75	14997285.0	17449278.75	19999647.0
Age	15000.0	4.278980e+01	1.698026e+01	20.0	28.00	39.0	56.00	79.0
Height	15000.0	1.744651e+02	1.425811e+01	123.0	164.00	175.0	185.00	222.0
Weight	15000.0	7.496687e+01	1.503566e+01	36.0	63.00	74.0	87.00	132.0
Duration	15000.0	1.553060e+01	8.319203e+00	1.0	8.00	16.0	23.00	30.0
Heart_Rate	15000.0	9.551853e+01	9.583328e+00	67.0	88.00	96.0	103.00	128.0
Body_Temp	15000.0	4.002545e+01	7.792299e-01	37.1	39.60	40.2	40.60	41.5
Calories	15000.0	8.953953e+01	6.245698e+01	1.0	35.00	79.0	138.00	314.0

Exploratory Data Analysis

```
sb.scatterplot(x='Height', y='Weight', data=df)
```

```
plt.show()
```

Output



So, we have a kind of linear relationship between these two features which is quite obvious.

```
features = ['Age', 'Height', 'Weight', 'Duration']
```

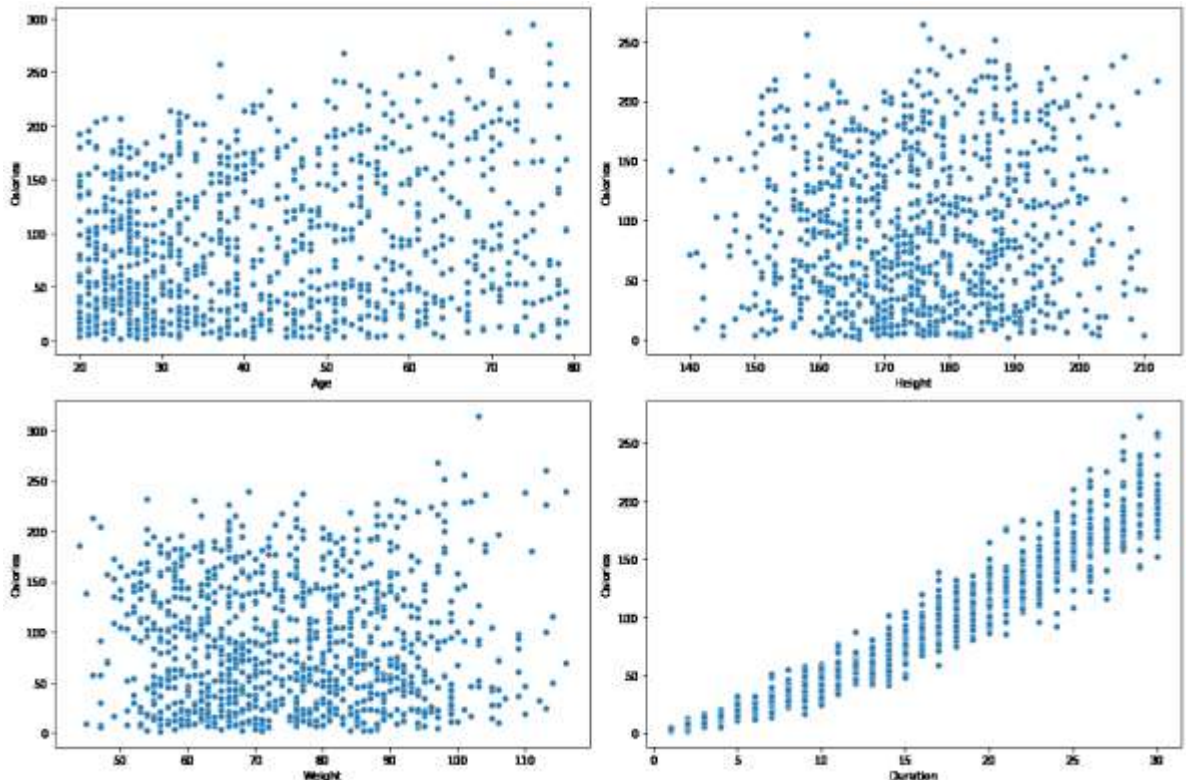
```
plt.subplots(figsize=(15, 10))
```

```
for i, col in enumerate(features):
```

```
    plt.subplot(2, 2, i + 1)
```

```
x = df.sample(1000)
sb.scatterplot(x=col, y='Calories', data=x)
plt.tight_layout()
plt.show()
```

Output



As expected higher is the duration of the workout higher will be the calories burnt. But except for that, we cannot observe any such relation between calories burnt and height or weight features.

Here we can observe some real-life observations

- ❖ The average height of the boys is higher than girls.
- ❖ Also, the weight of the girls is lower than that of the boys.
- ❖ For the same average duration of workout calories burnt by men is higher than that of women.

```
features = df.select_dtypes(include='float').columns
```

```
plt.subplots(figsize=(15, 10))
```

```
for i, col in enumerate(features):
```

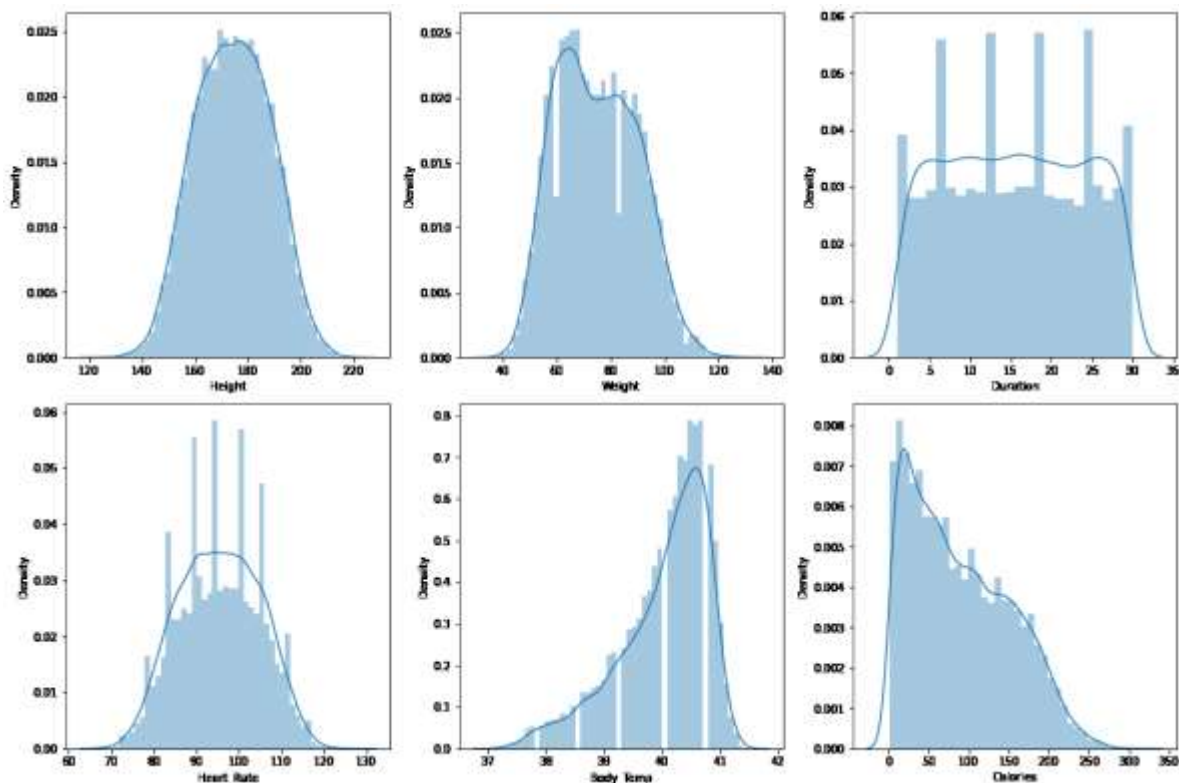
```
    plt.subplot(2, 3, i + 1)
```

```
    sb.distplot(df[col])
```

```
plt.tight_layout()
```

```
plt.show()
```

Output



The distribution of the continuous features follows close to normal distribution except for some features like Body_Temp and Calories.

```
df.replace({'male': 0, 'female': 1},
```

```
         inplace=True)
```

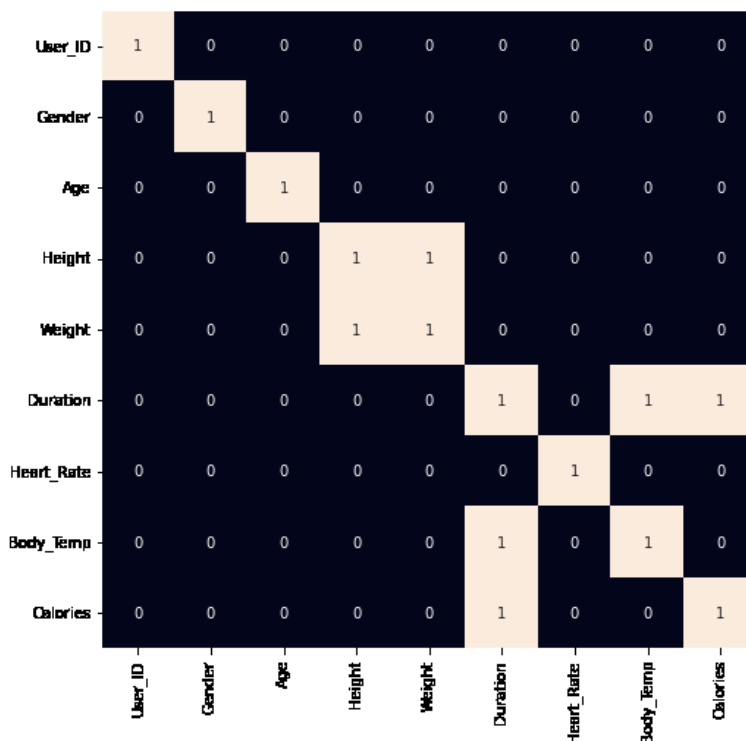
```
df.head()
```

Output

	User_ID	Gender	Age	Height	Weight	Duration	Heart_Rate	Body_Temp	Calories
0	14733363	0	68	190.0	94.0	29.0	105.0	40.8	231.0
1	14861698	1	20	166.0	60.0	14.0	94.0	40.3	66.0
2	11179863	0	69	179.0	79.0	5.0	88.0	38.7	26.0
3	16180408	1	34	179.0	71.0	13.0	100.0	40.5	71.0
4	17771927	1	27	154.0	58.0	10.0	81.0	39.8	35.0

```
plt.figure(figsize=(8, 8))
sb.heatmap(df.corr() > 0.9,
            annot=True,
            cbar=False)
plt.show()
```

Output



Artificial Intelligence(AI) with it's Applications

Here we have a serious problem of data leakage as there is a feature that is highly correlated with the target column which is calories.

```
to_remove = ['Weight', 'Duration']  
df.drop(to_remove, axis=1, inplace=True)
```

Model training

Now we will separate the features and target variables and split them into training and testing data by using which we will select the model which is performing best on the validation data.

```
features = df.drop(['User_ID', 'Calories'], axis=1)  
target = df['Calories'].values
```

```
X_train, X_val,\  
    Y_train, Y_val = train_test_split(features, target,  
                                      test_size=0.1,  
                                      random_state=22)  
X_train.shape, X_val.shape
```

Output

```
((13500, 5), (1500, 5))
```

Now, let's normalize the data to obtain stable and fast training.

Normalizing the features for stable and fast training.

```
scaler = StandardScaler()  
X_train = scaler.fit_transform(X_train)  
X_val = scaler.transform(X_val)
```

Now let's train some state-of-the-art machine learning models and compare them which fit better with our data.

```
from sklearn.metrics import mean_absolute_error as mae
```

```
models = [LinearRegression(), XGBRegressor(),
           Lasso(), RandomForestRegressor(), Ridge()]

for i in range(5):

    models[i].fit(X_train, Y_train)

    print(f'{models[i]} : ')

    train_preds = models[i].predict(X_train)
    print("Training Error : ', mae(Y_train, train_preds))

    val_preds = models[i].predict(X_val)
    print('Validation Error : ', mae(Y_val, val_preds))
    print()
```

Output

LinearRegression() :

Training Error : 17.893463692619434

Validation Error : 18.007896272831253

XGBRegressor(base_score=None, booster=None, callbacks=None,
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, device=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=None, max_bin=None,

Artificial Intelligence(AI) with it's Applications

```
max_cat_threshold=None, max_cat_to_onehot=None,  
max_delta_step=None, max_depth=None, max_leaves=None,  
min_child_weight=None, missing=nan, monotone_constraints=None,  
multi_strategy=None, n_estimators=None, n_jobs=None,  
num_parallel_tree=None, random_state=None, ...) :
```

Training Error : 7.89463304294701

Validation Error : 10.12050432946533

Lasso() :

Training Error : 17.915089584958036

Validation Error : 17.995033362288662

RandomForestRegressor() :

Training Error : 3.9877936746031746

Validation Error : 10.451300301587302

Ridge() :

Training Error : 17.893530494767777

Validation Error : 18.00781790803129